APLICAÇÃO DAS REDES NEURONAIS ADITIVAS GENERALIZADAS À MEDICINA

CARLOS JOSÉ BRÁS GERALDES

Tese para obtenção de grau de Doutor em Saúde das Populações

na Especialidade em Bioestística

na NOVA Medical School | Faculdade de Ciências Médicas.

Setembro, 2016.

APLICAÇÃO DAS REDES NEURONAIS ADITIVAS GENERALIZADAS À MEDICINA

Nome do autor: Carlos José Brás Geraldes.
Orientador: Ana Luísa Trigoso Papoila, Professora Auxiliar.
Coorientador: Patrícia Xufre Gonçalves da Silva Casqueiro, Professora Auxiliar.

Tese para obtenção de grau de Doutor em Saúde das Populações na Especialidade em Bioestatística.

Setembro, 2016.

À minha família

Ipse se nihil scire id unum sciat.

Sócrates

Agradecimentos

Quero dedicar estas primeiras linhas à Professora Doutora Ana Luísa Papoila que me aceitou como seu aprendiz e se predispôs a iluminar-me o caminho não só na ciência como também na vida. Enalteço e testemunho a sua total entrega, na maior parte das vezes com sacrifício pessoal, ao crescimento, quer intelectual, quer emocional dos seus aprendizes. Sinto-me honrado de ter sido um deles e espero vir a honrar o trabalho que teve comigo.

À Professora Doutora Patrícia Xufre pela sua disponibilidade em todos os momentos que necessitei da sua ajuda. Destaco a prontidão, a qualquer momento, para o atendimento e esclarecimento das minhas dúvidas, bem como das constantes revisões do meu trabalho ao longo destes anos.

Ao Centro de Estatística da Universidade de Lisboa, pelo apoio que me deu, especialmente no financiamento da minha participação em congressos, formações e publicações.

À minha esposa pela paciência e compreensão que demonstrou face aos transtornos familiares que um trabalho deste tipo inevitavelmente acarreta. Quero lhe agradecer os momentos que me substituiu no governo e educação das nossas filhas e de ter suportado os meus momentos de maior impaciência e frustração.

Às minhas filhas que todos os dias me orgulham e que me deram o alento para continuar.

Aos meus pais, os meus primeiros mestres, lhes dedico este trabalho como fruto da educação, carinho e amor que desde sempre me propiciaram. Quero lhes agradecer por tudo o que me deram e continuam a dar. Por todo o apoio e compreensão que me permitem suportar os maus momentos.

Resumo

A aplicação das redes neuronais artificiais tem vindo a aumentar em várias áreas de conhecimento, muito em parte devido à flexibilidade que caracteriza estes modelos, e que permite modelar não só dados provenientes de situações reais mais complexas, tais como o reconhecimento de padrões e de voz, mas também dados mais simples como, por exemplo, os que refletem relações entre várias variáveis independentes e uma resposta (variável dependente). Uma arquitetura bastante utilizada é a do Perceptrão Multicamada (MLP), no entanto, apesar da sua popularidade em várias áreas do conhecimento, este modelo é menos utilizado no âmbito clínico do que os Modelos Lineares Generalizados (GLMs) e os Modelos Aditivos Generalizados (GAMs). O principal motivo deve-se ao facto da MLP funcionar de forma opaca quanto à interpretabilidade dos efeitos de cada variável explicativa na resposta. Assim sendo, a aplicação de uma rede neuronal aditiva generalizada (GANN) no âmbito da Medicina é bastante mais promissor uma vez que, além de produzir boas estimativas, também os seus resultados são, à partida, interpretáveis. Por outro lado, no que respeita aos modelos de regressão, vários autores referem que uma má escolha da função de ligação pode afetar, em determinados casos, o desempenho do modelo, pelo que foram desenvolvidos estudos no intuito de melhorar este aspecto através da flexibilização da função de ligação quer de uma forma paramétrica quer não paramétrica. No caso da GANN, dada a exiguidade de estudos sobre esta arquitetura, constatamos que nada foi feito quanto à estimação da função de ligação recorrendo a métodos paramétricos, tornando-se este um dos objetivos deste estudo. Adicionalmente, a estimação da função de ligação recorrendo a métodos não paramétricos também foi sugerida e analisada. Outro objetivo do estudo passou pela introdução de métodos para a determinação de intervalos de confiança das funções parciais e da função da razão de possibilidades de modo a permitir uma melhoria substancial na interpretabilidade deste tipo de rede neuronal.

Palavras-chave: Rede Neuronal Aditiva Generalizada; Função de ligação flexível; Função da razão de possibilidades

Abstract

The application of artificial neural networks has been increasing in many areas of knowledge, much due to the flexibility that features these models and they allow to model data not only from more complex situations, such as pattern recognition and voice data but also simpler such as, for example, those that reflect relationships between multiple independent variables, and a outcome (dependent variable). A widely used architecture is the Multi Layer Perceptron (MLP). However, despite its popularity in several areas of knowledge, this model is less used in the clinical context that the Generalized Linear Models (GLMs) and the Generalized Additive Models (GAMs), due to the fact that, the MLP works as a black box. In the other hand, the Generalized Additive Neural Network (GANN) besides producing good estimates, allows the interpretation of the effect of each covariate on the outcome. Those features makes this type of neural network much more promising in the clinical area. On the other hand, concerning the regression models, several authors have reported that a bad choice of the link function can affect, in some cases, the model performance; so, studies have been developed in order to solve this issue by the introduction of a flexible link function either (parametric or nonparametric). In the case of a GANN, given the low number of studies about this architecture, we find that nothing has been done regarding the estimation of the link function using parametric methods, becoming this one goal of the present study. The link function estimation by using nonparametric methods was also approached. Another goal included the implement of methods for the determination of confidence intervals of the partial functions and the odds ratio function, which substantially improved the interpretability of the results of this type of neural network.

Keywords: Generalized Aditive Neural Network; Flexible link function; odds ratio function

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

A análise de dados em Medicina representa uma parte essencial, quer na produção de conhecimento quer na tomada de decisão nesta área. Este processo engloba um conjunto de procedimentos que visa a organização, validação, transformação e modelação de dados, cujo objetivo é o de extrair informação útil a partir dos mesmos.

A origem dos dados pode ser de natureza experimental ou de natureza observacional. No primeiro caso, os dados são obtidos a partir de uma experiência cujas variáveis são manipuladas deliberadamente (*e.g.* exposição a um determinado fator) com o objetivo de provocar um efeito na variável resposta (*e.g.* ensaios clínicos). Pelo estudo desse efeito, é possível determinar o resultado da intervenção efetuada sobre as variáveis explicativas (Smith, 2012). No segundo caso, os dados são provenientes de estudos em que se procede à observação, medição e registo da informação, sem se efetuar qualquer tipo de intervenção.

Em ambos os casos, a qualidade do estudo também advém da capacidade de avaliação dos fatores e de que forma estes influenciam o resultado. Num contexto da Medicina, esta avaliação apresenta-se como um desafio tanto para a gestão hospitalar como para o ato médico, onde a qualidade da decisão é crítica para a eficácia do tratamento bem como para a integridade do doente.

Para a boa prática e gestão clínica é fundamental o desenvolvimento de modelos para a tomada de decisão médica. A admissão de doentes críticos nas Unidades de Cuidados Intensivos (UCIs) constitui um bom exemplo. Estes serviços têm como missão a prestação de cuidados de saúde a pacientes em situação crítica, o que constitui um desafio à gestão hospitalar tendo em conta os pesados orçamentos que são necessários para a manutenção da qua-

lidade de resposta. Por isso, no dia a dia das UCIs, terão que ser tomadas decisões com base na eficácia do tratamento *versus* o seu custo.

Para auxiliar essas decisões, utilizam-se métricas, normalmente obtidas a partir de modelos de regressão, sendo os mais utilizados o Modelo Linear Generalizado (*Generalized Linear Model* - GLM) e o Modelo Aditivo Generalizado (*Generalized Additive Model* - GAM). Estes são geralmente orientados para a quantificação do risco de mortalidade e caracterizam-se por um número reduzido de variáveis, a partir das quais se extrai uma pontuação que reflete o estado da gravidade do doente além de uma estimativa de mortalidade intra-hospitalar.

Como alternativa aos modelos de regressão encontram-se as redes neuronais artificiais (Bishop, 1995). Com efeito, as redes neuronais permitem resolver não só problemas que envolvam relações complexas, tais como reconhecimento de padrões, como também outro tipo de relações. Uma arquitetura bastante utilizada é a do Perceptrão Multicamada (*Multi Layer Perceptron* - MLP), no entanto, apesar da sua popularidade noutras áreas do conhecimento, este modelo é menos utilizado no âmbito clínico do que os modelos de regressão.

Uma das razões prende-se com o facto de não possuir, de forma clara, um melhor desempenho que o demonstrado pelas metodologias estatísticas em situações similares (Tu, 1996; Plate et al., 1997). Outra razão tem a ver com o facto do MLP funcionar de forma opaca quanto à interpretabilidade dos efeitos de cada variável explicativa na resposta.

Neste sentido, a aplicação de uma Rede Neuronal Aditiva Generalizada (*Generalized Additive Neural Network* - GANN) é bastante mais promissor, uma vez que, além de produzir boas estimativas, também os seus resultados são, à partida, interpretáveis.

No entanto, no caso dos modelos de regressão, vários autores referem que uma má escolha da função de ligação, pode afetar (em determinados casos) o desempenho do modelo, o que levou ao desenvolvimento de estudos que permitissem a melhoria deste aspeto através da flexibilização da função de ligação quer de uma forma paramétrica quer não paramétrica. Assim, dada a exiguidade de estudos sobre a GANN, e de forma a melhorar a sua equiparação aos modelos existentes, foi estabelecido como um dos objetivos principais, a estimação da função de ligação recorrendo a métodos paramétricos. Adicionalmente, a estimação da função de ligação recorrendo a métodos não

paramétricos (através de um MLP) desenvolvido por de Waal et al. (2009) foi igualmente abordada, tendo sido, no presente estudo, introduzidas algumas melhorias.

Outro dos objetivos principais passou pela introdução de metodologias que melhorassem a interpretabilidade do modelo, nomeadamente, a determinação de intervalos de confiança das funções parciais e da função de razão de possibilidades.

Ao ser melhorada a equiparação das GANNs aos modelos de regressão, através das funcionalidades introduzidas, atrás referidas, esta rede neuronal configura-se como uma boa alternativa a utilizar em aplicações no âmbito da Medicina.

Assim, no capítulo 2, começaremos por abordar a teoria subjacente aos modelos de regressão mais utilizados a nível clínico; o GLM e o GAM. A descrição destes modelos torna-se relevante uma vez que é com base neles que se procurará construir uma rede neuronal que mimetize as suas características principais.

Além da descrição dos modelos de regressão, importa igualmente descrever as metodologias de amostragem e avaliação, pelo que, no capítulo 3 começaremos por abordar, com uma visão geral, os processos de amostragem que permitem validar os modelos, bem como algumas métricas relevantes para a sua avaliação e que também podem ser utilizados no âmbito das redes neuronais.

No capítulo 4, são introduzidos os aspetos comuns existentes entre os modelos de regressão atrás referidos e as redes neuronais. De facto, constata-se, por exemplo, que o Perceptrão Simples corresponde a uma arquitetura de rede neuronal equivalente ao GLM. De igual forma, alguns métodos que se aplicam aos modelos de regressão em geral, nomeadamente as metodologias para a estimação dos intervalos de confiança, também podem ser utilizados nas redes neuronais.

No capítulo 5 dá-se inicio à análise da GANN que pretende mimetizar os modelos de regressão GAM e que constituirá a base deste estudo. Os primeiros desenvolvimentos desta arquitetura foram introduzidos por Sarle (1994), num estudo onde são analisadas as relações existentes entre as diferentes topologias de redes neuronais e os modelos estatísticos correspondentes. Posteriormente, foi introduzido por Potts (1999), um conjunto de procedimentos

para a construção interativa da GANN, tendo sido, mais tarde, desenvolvido por Du Toit (2006) e de Waal e du Toit (2007) uma metodologia para seleção automática do modelo. Para tal, foi importante a adoção de um esquema de codificação de forma a permitir que estas arquiteturas possam ser utilizadas por algoritmos de procura da GANN que melhor se adapta aos dados de uma forma automática. Foi também proposto por Du Toit (2006) um desses algoritmos a que denominaram por AutoGANN. Neste capítulo são igualmente analisados outros aspetos tais como os problemas de identificabilidade e sua resolução.

Outro desenvolvimento introduzido por de Waal et al. (2009), diz respeito à utilização de um MLP como função de ligação da rede neuronal. No entanto, constata-se que a literatura existente é bastante exígua, pelo que a implementação da rede com um MLP como função de ligação, teve de sofrer novos desenvolvimentos neste estudo. Assim, é proposta uma abordagem que permita estimar a função de ligação pela utilização de um MLP, cuja função de ativação do neurónio de saída pertence à família de transformações assimétricas de Aranda-Ordaz.

Como um dos principais objetivos deste estudo, introduziu-se a mesma classe de funções (Aranda-Ordaz) como função de ativação do neurónio de saída da GANN original, permitindo a estimação de uma função de ligação paramétrica. Com efeito, verificou-se que em determinados cenários, esta configuração é a mais indicada.

Ambas as arquiteturas encontram-se descritas no capítulo 6.

No capítulo 7 é efetuado um estudo de simulação para uma análise comparativa do desempenho de uma GANN com função de ligação logística, e o das duas GANNs com função de ligação flexível descritas no capítulo 6, utilizando-se quatro cenários para o efeito.

No capítulo 8 foi introduzida mais uma inovação na GANN com função de ligação paramétrica flexível através da adaptação do algoritmo que se utiliza nos GAMs para a estimação dos OR.

São aplicadas as metodologias já existentes e as desenvolvidas a um caso real no capítulo 9 em que são analisadas várias variáveis presentes numa amostra recolhida numa UCI, respeitante a 996 doentes, com o intuito de estimar a probabilidade de morte dos doentes à entrada desta unidade.

No capítulo 10 encontram-se as conclusões globais do estudo.

Importa também salientar que a GANN implementada nos estudos existentes foi desenvolvida com base no SAS® Enterprise Miner, que não se encontra disponível para utilização em geral. Esta limitação obrigou à construção de raiz (para o presente estudo) de uma plataforma que, para além de implementar as GANNs, também implementa o MLP, assim como os respetivos métodos de amostragem, aprendizagem, de procura e seleção de modelos.

Para o efeito, recorreu-se à linguagem JAVA, tendo o projeto da plataforma sido implementado por camadas e testado o *software* utilizando o método em "caixa branca" (*i.e.* além da análise dos valores das entradas e saídas de cada objeto produzido, também foram analisadas as variáveis e os métodos do mesmo de forma a despistar possíveis erros no sistema). Além desse cuidado, foi igualmente efetuada uma operação de engenharia inversa sobre a *toolkit* de redes neuronais do MATLAB, para comparar os valores da atualização dos pesos sinápticos utilizando, por exemplo, métodos de RPROP ou Levenberg Marquadt com os produzidos pela plataforma CG-GANN e, desta forma, validar os resultados obtidos pela mesma. Embora este processo tenha sido moroso, permitiu assim garantir a qualidade dos componentes de *software*, de forma a inovar a estrutura da GANN através da utilização de funções de ligação flexíveis bem como introduzir novos métodos com vista à obtenção de intervalos de confiança e à estimação da razão de possibilidades no contexto das GANNs. O núcleo central da plataforma CG-GANN encontra-se descrita no Apêndice A com o respetivo código fonte.

Capítulo 2

Modelos de Regressão

2.1 Introdução

De uma forma geral, os métodos de prognóstico baseiam-se numa análise do histórico de acontecimentos utilizada para a explicação ou projeção de acontecimentos futuros.

A análise do problema tem como foco principal o estudo da relação entre variáveis, permitindo a análise da influência que uma ou mais variáveis explicativas têm sobre uma variável de interesse ou variável resposta. Concretamente, em termos metodológicos, a análise dessa relação passa pela elaboração de um modelo que represente o sistema de interesse e que poderá ser efetuado com recurso à aplicação de modelos de regressão (Amaral Turkman, 2000).

Para a construção destes modelos, coloca-se o problema relativo à definição da forma funcional da relação entre as variáveis explicativas e a variável resposta. Dependendo do grau de conhecimento que se possui *a priori* relativamente às associações entre aquelas variáveis, o modelo de regressão daí resultante poderá ser de natureza paramétrica (total conhecimento das associações atrás referidas), semi-paramétrica ou não paramétrica (no caso desse conhecimento ser, à partida, respetivamente, parcial ou inexistente).

Nestes últimos casos, a forma funcional é determinada diretamente a partir do conjunto de dados, pelo que, para a estimação deste tipo de modelos é exigida uma amostra com maior dimensão do que no caso da regressão paramétrica. As abordagens tradicionais para a estimação da forma funcional nos casos desconhecidos, incluem os estimadores do núcleo (Kernel), análise

utilizando *splines*, análise de Fourier e, mais recentemente, análise utilizando onduletas (*wavelets*).

Neste capítulo, serão abordados modelos de regressão de natureza paramétrica, onde se inclui o tradicional Modelo Linear Generalizado (*Generalized Linear Model* - GLM) e o modelo de regressão de natureza semi-paramétrica onde se inclui o Modelo Aditivo Generalizado (*Generalized Additive Model* - GAM). Este funciona como extensão do GLM, para o caso em que a forma funcional é desconhecida e/ou eventualmente não linear.

2.2 Modelo Linear Generalizado

O Modelo Linear Generalizado (GLM) tem como "antecessor" o modelo linear normal introduzido no séc. XIX por Legendre e Gauss, sendo, este último, o modelo dominante até meados do séc. XX. No entanto, verificou-se que em algumas situações este modelo não seria o mais adequado, tendo sido desenvolvido para estes casos, modelos não lineares ou não normais. Destes podemos destacar, a título de exemplo, o modelo complementar log-log para ensaios de diluição (Fisher, 1922); os modelos probit (Bliss, 1935) e o modelo logit (Berkson, 1944).

Estes modelos foram posteriormente, entre outros, resumidos e unificados numa classe mais geral (Modelos Lineares Generalizados) por Nelder e Wedderburn (1972).

2.2.1 Definição do modelo

O GLM funciona como uma extensão do modelo linear normal em que a relação entre a combinação linear das variáveis explicativas (preditor linear) e a variável resposta é especificada de uma forma mais geral, admitindo outras funções para além da função identidade, de forma a que se possam modelar dados cuja função densidade de probabilidade (f.d.p.) associada à variável resposta Y seja outra que não a distribuição normal. Dentro dessas possibilidades encontra-se qualquer distribuição pertencente à família exponencial, ou seja, cuja f.d.p. se escreva na forma (Amaral Turkman, 2000):

$$f(y|\theta,\phi) = e^{\left\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi)\right\}}. \tag{2.1}$$

Os parâmetros θ e ϕ, são ambos escalares em que o primeiro representa a forma canónica do parâmetro de localização de que se falará adiante e o segundo o parâmetro de dispersão, habitualmente conhecido. As funções a(.), b(.) e c(.) são funções reais conhecidas. Entre as principais distribuições da família exponencial encontram-se a normal, a binomial, a Poisson, a Gama e a Gaussiana Inversa (Amaral Turkman, 2000).

Estrutura de um GLM

A estrutura de um GLM é constituída por duas componentes relacionadas através de uma função de ligação. Uma das componentes é designada por componente aleatória em que se considera a resposta Y como variável aleatória com distribuição pertencente à família exponencial (ver eq. 2.1). Assim sendo, dado o vetor de covariáveis $\mathbf{x}_i$, as variáveis Y_i são condicionalmente independentes, com distribuição pertencente à família exponencial com $E(Y_i|\mathbf{x}_i) = \mu_i$, $i = 1, ..n$ e com um parâmetro de dispersão que não depende de i.

A outra componente, designada por estrutural ou sistemática assenta no preditor linear definido por:

$$\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}, \tag{2.2}$$

em que

$$\mathbf{X} = \begin{vmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots \\ 1 & x_{n1} & \dots & x_{np} \end{vmatrix}$$

corresponde a uma matriz de especificação, função dos vetores de covariáveis $\mathbf{x}_i$, $i = 1, .., n$, e $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_p)^T$ é um vetor de parâmetros de dimensão $p + 1$.

O valor esperado da variável resposta $E[Y_i|\mathbf{x}_i] = \mu_i$ relaciona-se com o preditor linear através de uma função de ligação $h(.)$, monótona e diferenciável, de tal forma que, para o i-ésimo indivíduo:

$$\mu_i = h(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) = h(\eta_i), \tag{2.3}$$

ou, inversamente,

$$g(\mu_i) = \eta_i, \tag{2.4}$$

em que $g(.) = h^{-1}(.)$ representa a inversa da função de ligação. No entanto, consoante a conveniência, também designaremos a função $g(.)$ de função de ligação.

Sempre que o preditor linear coincidir com o parâmetro canónico, *i.e.*, $\eta_i = \theta_i$ ($\theta_i = \mathbf{x}_i^T \boldsymbol{\beta}$), a função de ligação correspondente designa-se por função de ligação canónica.

A tabela 2.1 [1] apresenta algumas distribuições pertencentes à família exponencial e respetivas funções de ligação canónicas.

Distribuição de Y	Função de ligação canónica	Domínio de Y	$Var(Y)$
Normal ($N(\mu, \sigma^2)$)	identidade (μ)	$(-\infty, +\infty)$	σ^2
Binomial/n ($B(n, \pi)/n$)	*logit* ($log(\frac{\pi}{1-\pi})$)	$\{0, \frac{1}{n}, \ldots, 1\}$	$\frac{\pi(1-\pi)}{n}$
Poisson ($P(\lambda)$)	logarítmica ($log(\lambda)$)	{0,1,...}	λ
Gama ($Ga(\nu, \frac{\nu}{\mu})$)	recíproca ($-\frac{1}{\mu}$)	$(0, +\infty)$	$\frac{\mu^2}{\nu}$
Gaussiana Inversa ($IG(\mu, \sigma^2)$)	quadrática inversa ($-\frac{1}{2\mu^2}$)	$(0, +\infty)$	$\mu^3\sigma^2$

Tabela 2.1: Principais distribuições da família exponencial e respetiva função de ligação canónica.

Existem, porém, situações em que a opção pela função de ligação canónica não apresenta ser a melhor escolha, como por exemplo, no caso em que se tem $Y \in]0, +\infty[$. Neste caso, em particular, é apropriado utilizar-se uma função de ligação tal que os valores esperados sejam sempre positivos sem que haja necessidade de se impor restrições aos parâmetros do modelo.

GLM de variável resposta contínua

Modelo normal

Para o caso da variável resposta ser contínua, pode-se aplicar o modelo de regressão normal em que a função de ligação a utilizar é a função identidade, correspondendo diretamente ao modelo linear normal. Neste caso,

[1] Importa referir que a notação utilizada para designar, ao longo deste estudo, o logaritmo neperiano é *log(.)*

$$g(\mu_i) = \mu_i \Rightarrow \mu_i = \eta_i. \tag{2.5}$$

Modelo gama

No caso específico da resposta estar contida em $\mathbb{R}^+$, pode-se utilizar o modelo de regressão Gama. Neste caso, sendo $\mu_i > 0$, a utilização do modelo Gama com a função de ligação canónica respetiva (função recíproca) implicaria a imposição de restrições aos valores possíveis para os parâmetros $\boldsymbol{\beta}$ do preditor linear (Amaral Turkman, 2000). Para evitar essa situação, considera-se a seguinte função de ligação:

$$g(\mu_i) = log(\mu_i) \Rightarrow \mu_i = e^{\eta_i}. \tag{2.6}$$

Modelo gaussiano inverso

Relativamente ao modelo de regressão gaussiano inverso a utilização da função de ligação canónica implica que a relação entre o preditor linear e a variável resposta seja a seguinte:

$$g(\mu_i) = \frac{1}{\mu_i^2} \Rightarrow \mu_i = h(\eta_i) = \frac{1}{\sqrt{\eta_i}}. \tag{2.7}$$

À semelhança do que se passa com o modelo de regressão Gama, a escolha da função de ligação canónica implica, também neste caso, a imposição de restrições, nomeadamente $\eta_i > 0$. Neste caso, pode-se optar pela seguinte função de ligação:

$$g(\mu_i) = log(\mu_i) \Rightarrow \mu_i = h(\eta_i) = e^{\eta_i}. \tag{2.8}$$

GLM de variável resposta discreta

Modelo binomial

No caso do modelo Binomial, considera-se mY como tendo uma distribuição binomial de forma a que $Y \sim B(m, \pi)/m$. Assim, a função de massa probabilística (f.m.p.) é dada por (Amaral Turkman, 2000):

$$f(y|\pi) = \binom{m}{ym} \pi^{ym}(1-\pi)^{m-ym} = e^{(log(\pi)ym + m(1-y)log(1-\pi) + log\binom{m}{ym})}. \tag{2.9}$$

A partir da expressão anterior é deduzida a seguinte:

$$f(y|\pi) = e^{(m(y\theta - log(1+e^{\theta})) + log\binom{m}{ym})}, \tag{2.10}$$

em que $y \in (0, \frac{1}{m}, \frac{2}{m}, ..., 1)$ e $\theta = log(\frac{\mu}{1-\mu}) = log(\frac{\pi}{1-\pi})$.

Por utilizar a função de ligação *logit*, este modelo é designado por modelo de regressão logística. A sua inversa corresponde à função sigmoidal em que a relação entre o valor esperado de Y_i (a probabilidade de "êxito") e o preditor linear é:

$$\mu_i = \pi_i = \frac{1}{1 + e^{-\eta_i}}. \tag{2.11}$$

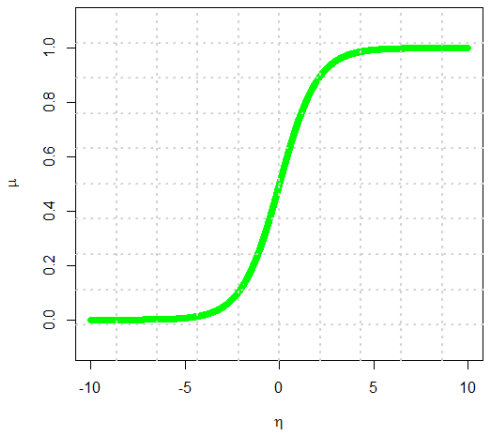

Figura 2.1: Função sigmoidal (logística).

Uma propriedade interessante da função sigmoidal (ver Fig. 2.1), é o facto de esta poder representar uma probabilidade, ou seja, para qualquer valor da componente sistemática, a resposta estimada irá tomar um valor entre 0 e 1. Também é interessante o facto de η_i descrever o logaritmo de uma possibilidade, uma vez que inclui o quociente de probabilidade de eventos com sucesso sobre a probabilidade de ocorrência de eventos sem sucesso. Como se pode verificar, estas características, entre outras, conferem interpretabilidade ao modelo, o que também contribui para a sua popularidade no âmbito clínico.

Este modelo poderá ser utilizado sempre que se pretenda associar a probabilidade de eventos de resposta binária ao preditor linear. No entanto,

devido ao facto de se ter $E(Y_i) = \mu_i \in [0, 1]$, a função de ligação $g(.)$ pode ser substituída por outras (não canónicas), cujo comportamento seja análogo à função logit. Um exemplo de função candidata é a função de ligação *probit*:

$$g(\mu_i) = \Phi^{-1}(\mu_i) = \eta_i \Leftrightarrow \mu_i = h(\eta_i) = \Phi(\eta_i), \tag{2.12}$$

em que Φ representa a função de distribuição normal padronizada. Outro exemplo é a função de ligação de *Gumbel*,

$$g(\mu_i) = log(-log(1 - \mu_i)). \tag{2.13}$$

Estas funções de ligação caracterizam, respetivamente, o modelo de regressão *probit* e o modelo de regressão complementar *log-log*.

Modelo de Poisson

Para o caso das respostas serem apresentadas sob a forma de contagens, pode-se utilizar o modelo de regressão de Poisson, em que a função de ligação é a função logarítmica:

$$g(\mu_i) = log(\mu_i) \Rightarrow \mu_i = h(\eta_i) = e^{\eta_i}. \tag{2.14}$$

2.2.2 Estimação do modelo

A estimação de parâmetros de um GLM deverá ser realizada tendo em conta a qualidade do ajustamento. Para a quantificar, pode-se recorrer à utilização de funções *score* baseadas em determinados estimadores, nomeadamente, no método dos mínimos quadrados ou no método de estimação da máxima verosimilhança (MLE). No caso de uma regressão linear simples com distribuição $N(0, \sigma^2)$, estes dois estimadores são equivalentes, no entanto, verifica-se que para um caso mais geral, a variância do MLE é menor do que a variância de qualquer outro estimador, pelo que, no caso do GLM, este método se apresenta como preferencial.

Assim, para a estimação dos parâmetros deste tipo de modelos, a verosimilhança, função de $\boldsymbol{\beta}$, obtém-se a partir da eq. 2.1 resultando no seguinte desenvolvimento:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} e^{[\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi)]} \Rightarrow L(\boldsymbol{\beta}) = e^{\sum_{i=1}^{n}[\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi)]}. \tag{2.15}$$

Normalmente, utiliza-se a função log-verosimilhança, por ser mais simples de maximizar, resultando na seguinte expressão:

$$l(\boldsymbol{\beta}) = log(L(\boldsymbol{\beta})) = \sum_{i=1}^{n}[\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)] = \sum_{i=1}^{n} l_i(\boldsymbol{\beta}), \tag{2.16}$$

em que se considera l_i como a contribuição de cada observação independente y_i para a log-verosimilhança.

Assim, a função *score* pode ser definida com base na log-verosimilhança, apresentando-se na forma do seguinte vetor p dimensional:

$$s(\boldsymbol{\beta}) = \left(\frac{\partial l(\boldsymbol{\beta})}{\partial \beta_1}, ..., \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_p}\right), \tag{2.17}$$

em que, para um parâmetro β_j específico ($j = 1, ..., p$) se tem:

$$\frac{\partial l(\boldsymbol{\beta})}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial l_i(\boldsymbol{\beta})}{\partial \beta_j}. \tag{2.18}$$

Considerando as seguintes relações:

$$\theta_i = \theta(\mu_i); \mu_i = h(\eta_i); \eta_i = {x_i}^T\boldsymbol{\beta}, \tag{2.19}$$

e aplicando a derivada da função composta, obtém-se a seguinte expressão:

$$\frac{\partial l_i(\boldsymbol{\beta})}{\partial \beta_j} = \frac{\partial l_i(\theta_i)}{\partial \theta_i}\frac{\partial \theta_i(\mu_i)}{\partial \mu_i}\frac{\partial \mu_i(\eta_i)}{\partial \eta_i}\frac{\partial \eta_i(\boldsymbol{\beta})}{\partial \beta_j}. \tag{2.20}$$

Considerando que:

$$\begin{cases} E[Y] = b'(\theta) \\ var(Y) = a(\phi)b''(\theta), \end{cases} \tag{2.21}$$

podem-se deduzir as seguintes expressões:

$$\frac{\partial l_i(\theta_i)}{\partial \theta_i} = \frac{y_i - b'(\theta_i)}{a(\phi)} = \frac{y_i - \mu_i}{a(\phi)}, \tag{2.22}$$

e

$$\frac{\partial \mu_i}{\partial \theta_i} = b''(\theta_i) = \frac{var(Y_i)}{a(\phi)}. \tag{2.23}$$

Tendo em atenção que para o i-ésimo indivíduo, a informação disponível apresenta-se na forma $(y_i, \mathbf{x}_i)$ em que $\mathbf{x}_i = (x_{i1}, ..., x_{ip})$ representa o vetor de especificação associado e que:

$$\frac{\partial \eta_i(\boldsymbol{\beta})}{\partial \beta_j} = x_{ij}, \tag{2.24}$$

a eq. 2.20 pode-se reescrever na forma:

$$\frac{\partial l_i(\boldsymbol{\beta})}{\partial \beta_j} = \frac{y_i - \mu_i}{a(\phi)} \frac{a(\phi)}{var(Y_i)} \frac{\partial \mu_i(\eta_i)}{\partial \eta_i} x_{ij}. \tag{2.25}$$

Chega-se assim à conclusão que o elemento genérico de ordem j da função *score* vem dado por:

$$s_j(\boldsymbol{\beta}) = \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_j} = \sum_{i=1}^{n} x_{ij} \frac{\partial \mu_i}{\partial \eta_i} \frac{(y_i - \mu_i)}{var(Y_i)}. \tag{2.26}$$

A maximização da função log-verosimilhança de um GLM, $l(\boldsymbol{\beta})$ implica a determinação da solução da equação $s(\hat{\boldsymbol{\beta}}) = 0$. Neste contexto, o sistema de equações anterior é geralmente constituído por equações não lineares, o que obriga à utilização de métodos numéricos para a sua resolução.

Para estes casos pode-se aplicar o método de Newton para processos iterativos em que

$$\hat{\boldsymbol{\beta}}^{(t+1)} = \hat{\boldsymbol{\beta}}^{(t)} - \frac{s(\hat{\boldsymbol{\beta}}^{(t)})}{s'(\hat{\boldsymbol{\beta}}^{(t)})}. \tag{2.27}$$

Por outro lado, considerando a eq. 2.17, a derivada da função *score* vem dada por:

$$\frac{\partial s(\boldsymbol{\beta})}{\partial \beta} = \mathcal{H}(\boldsymbol{\beta}) = \begin{vmatrix} \frac{\partial^2 l(\boldsymbol{\beta})}{\partial {\beta_1}^2} & \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \beta_1 \partial \beta_2} & \cdots & \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \beta_1 \partial \beta_p} \\ \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \beta_2 \partial \beta_1} & \frac{\partial^2 l(\boldsymbol{\beta})}{\partial {\beta_2}^2} & \cdots & \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \beta_2 \partial \beta_p} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \beta_p \partial \beta_1} & \frac{\partial^2 l(\boldsymbol{\beta})}{\partial \beta_p \partial \beta_2} & \cdots & \frac{\partial^2 l(\boldsymbol{\beta})}{\partial {\beta_p}^2} \end{vmatrix}, \tag{2.28}$$

em que $\mathcal{H}$ representa a matriz Hessiana da função log-verosimilhança $l(\boldsymbol{\beta})$. Desta forma, o algoritmo iterativo resultante corresponderá ao método Newton-Raphson em que

$$\hat{\boldsymbol{\beta}}^{(t+1)} = \hat{\boldsymbol{\beta}}^{(t)} - \mathcal{H}(\boldsymbol{\beta})^{-1} s(\hat{\boldsymbol{\beta}}^{(t)}). \tag{2.29}$$

No entanto, a aplicabilidade do método depende da existência e invertibilidade de $\mathcal{H}$ nas sucessivas iterações de $\hat{\boldsymbol{\beta}}^{(t)}$. Por outro lado, a convergência do método, a partir de qualquer valor inicial $\hat{\boldsymbol{\beta}}^{(0)}$ não se encontra garantida, pelo que, por estas razões e ainda pelo facto deste método ser computacionalmente exigente, é habitual aplicar-se uma outra variante correspondente ao método de Fisher. Este último introduz uma modificação ao método Newton-Raphson, substituindo a matriz Hessiana pela matriz de informação de Fisher (que corresponde à matriz de covariância da função *score*).

A matriz de informação de Fisher é definida por:

$$\mathcal{I}(\boldsymbol{\beta}) = -E\Big[\mathcal{H}(\boldsymbol{\beta})\Big] = E\Big[-\frac{\partial s(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\Big]. \tag{2.30}$$

Aplicando a relação anterior, a eq. 2.29 apresenta a seguinte forma:

$$\hat{\boldsymbol{\beta}}^{(t+1)} = \hat{\boldsymbol{\beta}}^{(t)} + [\mathcal{I}(\hat{\boldsymbol{\beta}})^{(t)}]^{-1} s(\hat{\boldsymbol{\beta}}^{(t)}), \tag{2.31}$$

em que a matriz $\mathcal{I}(\beta)$ além de ser semi-definida positiva, é de cálculo mais fácil. Para definir a matriz de informação de Fisher pode-se partir da eq. 2.30 e considerar que, para um elemento genérico (j, k) da matriz de informação (de dimensão $p \times p$), existe a seguinte relação (Amaral Turkman, 2000):

$$\mathcal{I}(\boldsymbol{\beta})_{jk} = -E\left[\frac{\partial^2 l_i(\boldsymbol{\beta})}{\partial \beta_j \partial \beta_k}\right] = E\left[\frac{\partial l_i(\boldsymbol{\beta})}{\partial \beta_j}\frac{\partial l_i(\boldsymbol{\beta})}{\partial \beta_k}\right]. \tag{2.32}$$

Considerando 2.26 (ver Dobson (2010)),

$$\mathcal{I}(\boldsymbol{\beta})_{jk} = E[s_j(\boldsymbol{\beta})s_k(\boldsymbol{\beta})] = E\left[\sum_{i=1}^{n} x_{ij}\frac{\partial\mu_i}{\partial\eta_i}\frac{(Y_i-\mu_i)}{var(Y_i)}\sum_{l=1}^{n} x_{lk}\frac{\partial\mu_l}{\partial\eta_l}\frac{(Y_l-\mu_l)}{var(Y_l)}\right]. \tag{2.33}$$

O resultado da equação anterior resulta numa soma, cujos termos são, na sua maioria, iguais a zero. Isto deve-se ao facto de as observações serem estatisticamente independentes, pelo que $E[(Y_i-\mu_i)(Y_l-\mu_l)] = 0$, se $i \neq l$.

Assim, a eq. 2.33 toma a seguinte forma:

$$\mathcal{I}(\boldsymbol{\beta})_{jk} = E\left[\sum_{i=1}^{n}\sum_{l=1}^{n} x_{ij}x_{lk}\frac{\partial\mu_i}{\partial\eta_i}\frac{\partial\mu_l}{\partial\eta_l}\frac{(Y_i-\mu_i)}{var(Y_i)}\frac{(Y_l-\mu_l)}{var(Y_l)}\right], \tag{2.34}$$

$$\mathcal{I}(\boldsymbol{\beta})_{jk} = E\left[\sum_{i=1}^{n} x_{ij}x_{ik}\frac{E(Y_i-\mu_i)^2}{var(Y_i)^2}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)^2\right], \tag{2.35}$$

$$\mathcal{I}(\boldsymbol{\beta})_{jk} = \sum_{i=1}^{n} x_{ij}x_{ik}\frac{E[(Y_i-\mu_i)^2]}{var(Y_i)^2}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)^2. \tag{2.36}$$

Considerando que $var(Y_i) = E[(Y_i-\mu_i)^2]$, a equação anterior fica assim reduzida à seguinte forma:

$$\mathcal{I}(\boldsymbol{\beta})_{jk} = \sum_{i=1}^{n} \frac{x_{ij}x_{ik}}{var(Y_i)}\left(\frac{\partial\mu_i}{\partial\eta_i}\right)^2. \tag{2.37}$$

A equação anterior toma a forma matricial

$$\mathcal{I}(\boldsymbol{\beta}) = \mathbf{X}^T\mathbf{W}\mathbf{X}, \tag{2.38}$$

em que $\mathbf{W}$ representa uma matriz diagonal de ordem n cujo i-ésimo elemento é dado por:

$$\omega_i = \frac{\left(\frac{\partial\mu_i}{\partial\eta_i}\right)^2}{var(Y_i)}. \tag{2.39}$$

Por outro lado, partindo da eq. 2.26 na sua forma matricial

$$S(\boldsymbol{\beta}) = \mathbf{X}^T\mathbf{A}(\boldsymbol{y}-\boldsymbol{\mu}), \tag{2.40}$$

em que $\boldsymbol{y} = (y_1, ..., y_n)^T$, $\boldsymbol{\mu} = (\mu_1, ..., \mu_n)^T$ e $\mathbf{A}$ representa uma matriz diagonal de ordem n cujo i-ésimo elemento é:

$$\alpha_i = \frac{\frac{\partial \mu_i}{\partial \eta_i}}{var(Y_i)}. \tag{2.41}$$

Com base em 2.38, deduz-se que

$$\mathbf{A} = \mathbf{W}\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\mu}}, \tag{2.42}$$

em que $\frac{\partial \boldsymbol{\eta}}{\partial \boldsymbol{\mu}}$ corresponde a uma matriz diagonal de ordem n, cujo i-ésimo elemento corresponde a $\frac{\partial \eta_i}{\partial \mu_i}$.

Antes de se aplicarem as formas matriciais, atrás deduzidas, à eq. 2.31, pode-se simplificá-la através da multiplicação de ambos os lados da equação por $\mathcal{I}(\boldsymbol{\beta})$, tomando a seguinte forma:

$$[\mathcal{I}(\hat{\boldsymbol{\beta}})^{(t)}]\hat{\boldsymbol{\beta}}^{(t+1)} = [\mathcal{I}(\hat{\boldsymbol{\beta}})^{(t)}]\hat{\boldsymbol{\beta}}^{(t)} + s(\hat{\boldsymbol{\beta}}^{(t)}). \tag{2.43}$$

Assim, se forem aplicadas as formas matriciais deduzidas em 2.36, 2.40 e 2.42 ter-se-á:

$$\mathbf{X}^T\mathbf{W}^{(t)}\mathbf{X}\hat{\boldsymbol{\beta}}^{(t+1)} = \mathbf{X}^T\mathbf{W}^{(t)}\mathbf{X}\hat{\boldsymbol{\beta}}^{(t)} + \mathbf{X}^T\mathbf{W}^{(t)}\frac{\partial \eta^{(t)}}{\partial \boldsymbol{\mu}^{(t)}}(\mathbf{y} - \boldsymbol{\mu}^{(t)}), \tag{2.44}$$

que é equivalente a

$$\mathbf{X}^T\mathbf{W}^{(t)}\mathbf{X}\hat{\boldsymbol{\beta}}^{(t+1)} = \mathbf{X}^T\mathbf{W}^{(t)}\mathbf{U}^{(t)}, \tag{2.45}$$

em que

$$\mathbf{U}^{(t)} = \mathbf{X}\hat{\boldsymbol{\beta}}^{(t)} + \frac{\partial \boldsymbol{\eta}^{(t)}}{\partial \boldsymbol{\mu}^{(t)}}(\mathbf{y} - \boldsymbol{\mu}^{(t)}). \tag{2.46}$$

Assim, para cada elemento u_i da matriz anterior, tem-se a seguinte relação:

$$u_i^{(t)} = \sum_{j=1}^{m} x_{ij}{\beta_j}^{(t)} + (y_i - \mu_i^{(t)})\frac{\partial \eta_i^{(t)}}{\partial \mu_i^{(t)}} = \eta_i^{(t)} + (y_i - \mu_i^{(t)})\frac{\partial \eta_i^{(t)}}{\partial \mu_i^{(t)}}. \tag{2.47}$$

Por fim, a partir de 2.45 obtém-se a seguinte relação:

$$\hat{\boldsymbol{\beta}}^{(t+1)} = [\mathbf{X}^T\mathbf{W}^{(t)}\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{W}^{(t)}\mathbf{U}^{(t)}. \tag{2.48}$$

O método de *scores* de Fisher pode ser visto como um método iterativo de mínimos quadrados ponderados que utiliza uma transformação da variável resposta. Em cada iteração, a estimação atual de $\boldsymbol{\beta}$ é usado para calcular uma nova variável u_i bem como para determinar a matriz $\mathbf{W}$. No momento seguinte, é efetuada uma regressão linear de $\mathbf{U}^{(t)}$ em X, com $\mathbf{W}^{(t)}$ como matriz de pesos de forma a obter novos valores de $\boldsymbol{\beta}$.

2.2.3 Estimação intervalar

No processo de inferência, a estimação pontual dos parâmetros da população não deixa de resultar de uma aproximação ao valor real, pelo que, é necessário avaliar a sua precisão através da estimação intervalar dos parâmetros (intervalos de confiança).

Para determinar os intervalos de confiança para $\boldsymbol{\beta}$, é necessário conhecer a distribuição de amostragem de $\hat{\boldsymbol{\beta}}$. Para tal, são importantes as características da função *score* $s(\boldsymbol{\beta})$. Partindo da eq. 2.26, e tendo em atenção que $E[Y_i] = \mu_i, \forall i$ verifica-se que

$$E[s_j(\boldsymbol{\beta})] = 0, \forall j \in \{1, ..., p\}. \tag{2.49}$$

Por outro lado, desenvolvendo a função de Taylor em torno de $\hat{\boldsymbol{\beta}}$ e retendo apenas os termos de 1ª ordem (Amaral Turkman, 2000),

$$s(\boldsymbol{\beta}) \approx s(\hat{\boldsymbol{\beta}}) + \frac{\partial s(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}|_{\boldsymbol{\beta}=\hat{\boldsymbol{\beta}}}(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}). \tag{2.50}$$

Considerando que o estimador de máxima verosimilhança $\hat{\boldsymbol{\beta}}$ é obtido a partir de $s(\hat{\boldsymbol{\beta}}) = 0$, de forma a maximizar a função de log-verosimilhança $l(\boldsymbol{\beta})$, sendo $-\mathcal{H}(\hat{\boldsymbol{\beta}}) = \frac{\partial s(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}|_{\boldsymbol{\beta}=\hat{\boldsymbol{\beta}}}$ a equação anterior toma a seguinte forma:

$$s(\boldsymbol{\beta}) \approx -\mathcal{H}(\hat{\boldsymbol{\beta}})(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}). \tag{2.51}$$

Por outro lado, para grandes amostras, pode-se considerar a seguinte relação (Amaral Turkman, 2000):

$$-\mathcal{H}(\hat{\boldsymbol{\beta}}) = \mathcal{I}(\boldsymbol{\beta}), \tag{2.52}$$

o que resulta na seguinte expressão:

$$(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}) \approx \mathcal{I}(\boldsymbol{\beta})^{-1} s(\boldsymbol{\beta}). \tag{2.53}$$

Partindo do pressuposto que $\mathcal{I}$ é uma matriz não singular, a partir da eq. 2.49 pode-se deduzir a seguinte propriedade de $\hat{\boldsymbol{\beta}}$:

$$E[s(\boldsymbol{\beta})] = 0 \Rightarrow E[\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}] = 0 \Rightarrow E(\hat{\boldsymbol{\beta}}) \approx \boldsymbol{\beta}. \tag{2.54}$$

Outra propriedade que se pode deduzir a partir das equações 2.53, 2.32 e tendo em consideração que $(\mathcal{I}(\boldsymbol{\beta})^{-1})^T = \mathcal{I}(\boldsymbol{\beta})^{-1}$, devido à simetria da matriz $\mathcal{I}$, é a seguinte:

$$cov(\hat{\boldsymbol{\beta}}) \approx E[(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})^T] = \mathcal{I}(\boldsymbol{\beta})^{-1} E[s(\boldsymbol{\beta}) s(\boldsymbol{\beta})^T] \mathcal{I}(\boldsymbol{\beta})^{-1} = \mathcal{I}(\boldsymbol{\beta})^{-1}. \tag{2.55}$$

Assim, através da aplicação do Teorema do Limite Central garante-se o seguinte resultado assintótico:

$$\hat{\boldsymbol{\beta}} \overset{a}{\sim} N_p(\boldsymbol{\beta}, \mathcal{I}^{-1}(\boldsymbol{\beta})). \tag{2.56}$$

ou seja, a distribuição assintótica de $\hat{\boldsymbol{\beta}}$ é normal multivariada, com valor médio $\boldsymbol{\beta}$ e matriz de covariância igual a $\mathcal{I}(\boldsymbol{\beta})^{-1}$ (Amaral Turkman, 2000).

A partir desta propriedade distribucional de $\hat{\boldsymbol{\beta}}$ determina-se o intervalo de confiança para $\boldsymbol{\beta}$.

Para além de englobar um grande número de modelos, os GLMs possuem a faculdade de serem bastante acessíveis, o que os torna preferenciais, nomeadamente no âmbito das aplicações médicas. Esta característica, aliada ao aumento da capacidade computacional que se tem vindo a registar, facilita bastante a aplicação destes modelos. No entanto, apesar destas vantagens, os GLMs não são isentos de problemas, uma vez que apresentam constrangimentos que advêm da rigidez das suas características fundamentais. Para fazer face a estes problemas, foram propostos novos desenvolvimentos contribuindo para uma evolução das caraterísticas em direção a uma maior flexibilidade.

De entre estes modelos destacam-se os GAMs que passaremos a descrever.

2.3 Modelo Aditivo Generalizado

Apesar da popularidade dos GLMs, a falta de flexibilidade inerente à relação linear assumida entre as variáveis explicativas e a resposta, pode conduzir a modelos mal especificados. Neste contexto, é natural que os GLMs tenham evoluído na direção de modelos que, à partida, considere como desconhecidas as formas funcionais entre os preditores e a resposta, nomeadamente os GAMs.

No entanto, antes de se definir este tipo de modelos, vejamos como se pode estimar uma f.d.p. de forma não paramétrica recorrendo a funções suavizadoras. Passaremos a descrever os dois tipos de suavizadores mais comummente utilizados.

Suavizadores Kernel

A ideia subjacente às funções suavizadoras do tipo Kernel, baseia-se na assunção de que a estimativa da f.d.p. $f(x)$ de uma variável aleatória X pode ser dada através da seguinte expressão:

$$\hat{f}_\lambda(x) = \frac{1}{2n\lambda}\#\{x_i \in [x-\lambda, x+\lambda[\}, \tag{2.57}$$

em que λ é designado por "janela" (*bandwith*) e que se assume como um parâmetro suavizador.

De forma mais genérica, pode-se definir uma função K, real, limitada e integrável tal que:

$$lim_{x\to\infty}|x|K(x) = 0, \tag{2.58}$$

denominada por função Kernel (é usual, considerar-se $K(x)$ como uma função densidade de probabilidade simétrica). As funções Kernel mais utilizadas são as seguintes:

- Uniforme em que $K(t) = \frac{1}{2}\mathbf{I}(\mid t \mid \leq 1)$;
- Epanechnikov em que $K(t) = \frac{3}{4}(1-t^2)\mathbf{I}(\mid t \mid \leq 1)$;

- Quártica em que $K(t) = \frac{15}{16}(1-t^2)^2\mathbf{I}(\mid t \mid \leq 1)$;
- Tricúbica em que $K(t) = (1-t^3)^3\mathbf{I}(0 \leq \mid t \mid < 1)$;
- Gaussiana em que $K(t) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}t^2}$;

em que $\mathbf{I}(.)$ representa a função indicatriz. Uma estimativa não paramétrica da f.d.p. de uma variável aleatória X, utilizando funções Kernel, é dada por:

$$\hat{f}_\lambda(x) = \frac{1}{n}\sum_{i=1}^{n} K_\lambda(x - x_i), \quad K_\lambda = \frac{1}{\lambda}K\Big(\frac{\cdot}{\lambda}\Big). \tag{2.59}$$

Vejamos, então, num contexto de regressão, como estimar $E(Y|X = x)$ recorrendo a funções Kernel,

$$m(x) = E(Y|X = x) = \int y f(y|x)dy = \int y\frac{f(x,y)}{f_X(x)}dy, \tag{2.60}$$

em que $f(x, y)$ representa a função densidade conjunta de X e Y e $f_X(x)$ a função densidade marginal de X.

Para o caso de X e Y serem ambas variáveis aleatórias, pode-se estimar $f(x, y)$ através de um Kernel multiplicativo da densidade para o caso bivariado:

$$\hat{f}_{\lambda_1,\lambda_2}(x,y) = \frac{1}{n}\sum_{i=1}^{n} K_{\lambda_1}(x - x_i)K_{\lambda_2}(y - y_i). \tag{2.61}$$

Substituindo na eq. 2.60 as expressões 2.59 e 2.61, obtém-se a estimativa Nadaraya-Watson (Nadaraya, 1964):

$$\hat{m}(x) = \frac{\sum\limits_{i=1}^{n} K_\lambda(x - x_i)y_i}{\sum\limits_{i=1}^{n} K_\lambda(x - x_i)}. \tag{2.62}$$

Considerando-se a variável X como sendo controlada, simultaneamente com o facto de $f_X(x)$ ser uma função conhecida, pode-se aplicar um conjunto de estimadores, nomeadamente, o de Priestley-Chao (Priestley, 1972), dando origem às seguintes estimativas:

$$\hat{m}(x) = \sum_{i=2}^{n}(x_i - x_{i-1})K_\lambda(x - x_i)y_i, \tag{2.63}$$

e o de Gasser-Muller (Gasser e Müller, 1979):

$$\hat{m}(x) = \sum_{i=1}^{n} \Big[\int_{s_{i-1}}^{s_i} K_\lambda(t-x)dt \Big] y_i, \tag{2.64}$$

em que $x \in [s_0, s_n]$ e $s_i = \frac{(x_{(i)} - x_{(i+1)})}{2}$, $i = 1, ..., n-1$.

Splines

A ideia subjacente a este tipo de suavizadores baseia-se na construção de um modelo que contém uma função suavizadora s de uma única covariável (função univariada). Essa função poderá ser traduzida pela seguinte expressão:

$$s(x) = \sum_{k=1}^{q} b_k(x)\beta_k, \tag{2.65}$$

em que $b_k(x)$ representa uma função base de ordem k e β_k o parâmetro de ordem k desconhecido. Se definirmos uma função suavizadora a partir de uma regressão polinomial genérica de ordem p, obter-se-á a seguinte função:

$$s(x) = \beta_0 + \sum_{k=1}^{p} x^k \beta_k. \tag{2.66}$$

No entanto, no caso da nuvem de pontos apresentar grandes irregularidades, a regressão polinomial poderá ser pouco flexível, uma vez que o ajustamento é efetuado de forma global. Neste caso, a regressão por *splines* procederá ao ajustamento recorrendo a intervalos, construídos a partir de pontos $(c_1, ..., c_m)$ denominados "nós" (Hastie e Tibshirani, 1990).

A função base é o pilar dos *splines* sendo que a mais simples é constituída por funções lineares "truncadas" do tipo $(x - c_k)_+$ em que, para um ponto genérico c_k:

$$(x - c_k)_+ = \begin{cases} x - c_k \quad se \quad x \geq c_k \\ \\ 0 \quad se \quad x < c_k. \end{cases} \tag{2.67}$$

Desta forma, o *spline* $s(x)$ é formado pela combinação linear das funções base $1, x, (x - c_1)_+, ..., (x - c_m)_+$, obtendo-se assim a seguinte função suavizadora:

$$s(x) = \beta_0 + \beta_1 x + \sum_{k=1}^{m} \beta_{1k}(x - c_k)_+. \tag{2.68}$$

O problema na utilização de funções base lineares é que não se pode garantir que não haja descontinuidade da primeira derivada da combinação linear dessas funções. Assim, é natural que se considerem funções não lineares em cada um dos intervalos. Uma generalização da função suavizadora de grau p nestas condições, é traduzida pela seguinte expressão:

$$s(x) = \beta_0 + \beta_1 x + ... + \beta_p x^p + \sum_{k=1}^{m} \beta_{pk}(x - c_k)_+^p. \tag{2.69}$$

A estimação do *spline* s pode ser efetuada recorrendo à minimização pelo método dos mínimos quadrados de:

$$\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \|^2, \tag{2.70}$$

em que $\boldsymbol{\beta} = (\beta_0, \beta_1, ..., \beta_p, \beta_{p1}, ..., \beta_{pm})^T$.

Um outro problema prende-se com o número de nós a utilizar. Efetivamente, se o número escolhido for pequeno, conduzirá a um subajustamento. No caso de se escolher um grande número de nós, a função ajustada poderá interpolar as pequenas irregularidades existentes na nuvem e que resultam de flutuações aleatórias (Hastie e Tibshirani, 1990).

Uma forma de mitigar a arbitrariedade que advém da escolha do número de nós, pode passar por controlar o grau de suavização do *spline* recorrendo a um termo penalizador que se adiciona a 2.70. Assim, a estimação do suavizador s passa pela minimização de:

$$\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \|^2 + \lambda \int [s''(t)]^2 dt, \tag{2.71}$$

Desta forma, o parâmetro λ controla o compromisso entre a adaptação do *spline* e o seu grau de suavização. Se $\lambda \to 0$, resulta num *spline* "despenalizado" (Wood, 2006), permitindo que $s''(x)$ tome valores elevados, dando ênfase a uma maior interpolação dos dados. Para o caso de $\lambda \to \infty$, a penalização é de tal maneira grande que conduz a que a função s coincida com a reta dos mínimos quadrados.

Parâmetro suavizador

A estimação do valor ideal de λ pode ser otimizada recorrendo a métodos como o da validação cruzada que, neste contexto, se traduz pela minimização

da função:

$$CV(\lambda) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{s}_\lambda^{-i}(x_i))^2, \lambda \geq 0, \tag{2.72}$$

em que $\hat{s}_\lambda^{-i}(x_i)$ corresponde à estimativa em x_i, calculada a partir de todas as observações exceto a i-ésima. O valor ideal de λ pode ser assim obtido, após o cálculo de $CV(\lambda)$ para vários valores de λ escolhendo o que minimiza a função.

Existem, no entanto, outras versões "simplificadas" deste algoritmo, computacionalmente mais eficazes. De entre estas, destaca-se a validação cruzada generalizada (*Generalized Cross Validation* - GCV), que se encontra descrita com maior detalhe em Craven e Wahba (1978). Relativamente à escolha do parâmetro suavizador, no caso dos suavizadores do tipo Kernel, sugere-se a consulta de Härdle (2004) e no caso de se pretender utilizar *splines*, podem-se consultar Hastie e Tibshirani (1990) ou Ruppert e Wand (1994).

2.3.1 Definição do modelo

O modelo é traduzido pela seguinte equação:

$$E(Y|X_1, X2, ..., X_p) = h(\beta_0 + f_1(X_1) + f_2(X_2) + ... + f_p(X_p)), \tag{2.73}$$

em que h representa a função de ligação do modelo, $X_1, ..., X_p$ representam as covariáveis e $f_j(.)$, $j = 1, ..p$ as funções suavizadoras designadas por funções parciais.

Por outro lado, tendo em conta que a expressão 2.73 pode ser também escrita na forma:

$$g(\mu_i) = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}). \tag{2.74}$$

2.3.2 Estimação do modelo

O método de estimação de um GAM é inspirado no método de estimação de um GLM. Assim, partindo da eq. 2.47 em que $\eta = g(\mu)$ com *g(.)* representando a inversa da função de ligação $h(.)$, pode-se considerar a seguinte aproximação:

$$g(y_i) \approx g(\mu_i) + (y_i - \mu_i)\frac{\partial \eta_i}{\partial \mu_i}. \tag{2.75}$$

Esta equação será utilizada no algoritmo de *scores* local (algoritmo 1) (Wood, 2006), para estimação de um GAM.

No passo 6 do algoritmo 1, é necessário utilizar um algoritmo iterativo, sendo o mais usual, neste tipo de modelos, o algoritmo *backfitting* (Wood, 2006) (ver algoritmo 2).

início

Estimar os valores iniciais $\beta_0^{(0)} = \frac{\sum_{i=1}^{n} y_i}{N}; \hat{\mathbf{f}}_1 = ... = \hat{\mathbf{f}}_p = \mathbf{0}$;

para *l=0,1,2,...* **faça**

Estimar $z_i^{(l)} = \hat{\eta}_i^{(l)} + (y_i - \hat{\mu}_i^{(l)})\left(\frac{\partial \hat{\eta}_i}{\partial \hat{\mu}_i}\right)_{(l)}$;

Obter $\hat{\omega}_i = \left(\frac{\partial \hat{\mu}_i}{\partial \hat{\eta}_i}\right)^2_{(l)} \left(\hat{V}_i^{(l)}\right)^{-1}$, em que $\hat{\eta}_i^{(l)} = \hat{\beta}_0^{(l)} + \sum_{j=1}^{p} \hat{f}_j^{(l)}(x_{ij})$, $\hat{\mu}_i^{(l)} = h(\hat{\eta}_i^{(l)})$, $\left(\frac{\partial \hat{\mu}_i}{\partial \hat{\eta}_i}\right)_{(l)} = \frac{\partial h}{\partial \eta}|_{\eta=\hat{\eta}^{(l)}}$, em que $\hat{V}_i^{(l)}$ representa a variância de Y calculada em $\hat{\mu}_i^{(l)}, i = 1, ..., n$;

Adaptar um modelo aditivo a partir de um algoritmo iterativo (*e.g. backfitting*) com base nas respostas ajustadas $\hat{\boldsymbol{z}}$ e nos pesos $\hat{\boldsymbol{\omega}}$ de forma a obter as respetivas funções parciais $\hat{\boldsymbol{f}}_j^{(l+1)}$, bem como $\hat{\eta}_i{}^{(l+1)}$ e $\hat{\mu}_i{}^{(l+1)}$, com $j = 1, ..., p$ e $i = 1, ..., n$;

se $\frac{\sum_{i=1}^{n} (\hat{\eta}_i^{(l+1)} - \hat{\eta}_i^{(l)})^2}{\sum_{i=1}^{n} \left(\hat{\eta}_i^{(l)}\right)^2} \leq \delta$ **então**

. **retorna** $\hat{\boldsymbol{f}}_j{}^{(l+1)}$, $\hat{\eta}_i{}^{(l+1)}$ e $\hat{\mu}_i{}^{(l+1)}$;

fim

senão

$\hat{\boldsymbol{f}}_j{}^{(l)} = \hat{\boldsymbol{f}}_j{}^{(l+1)}$, $\hat{\eta}_i{}^{(l)} = \hat{\eta}_i{}^{(l+1)}$ e $\hat{\mu}_i{}^{(l)} = \hat{\mu}_i{}^{(l+1)}$;

fim

fim

fim

Algoritmo 1: Algoritmo de *scores* local.

início
  $\hat{\beta}_0 = \overline{y}$;
  $\hat{\boldsymbol{f}}_j^{(0)} = 0$ ou obter $\hat{\boldsymbol{f}}_j^{(0)}$ a partir de uma regressão linear de Y em X_j, para $j = 1, ..., p$;
  para *l=0,1,2,...* **faça**
    para $j = 1, ..., p$ **faça**
      Estimar os resíduos parciais $\mathbf{r}^j = \mathbf{y} - \hat{\beta}_0 - \sum_{k \neq j} \hat{f}_k^{(l)}(\mathbf{x}_k)$;
      Estimar $\hat{\mathbf{f}}_j^{(l+1)} = \mathbf{S}_j \mathbf{r}^j$, em que $\mathbf{S}_j$ é uma matriz suavizadora $n \times n$ associada à covariável X_j;
    fim
    se $\frac{\sum_{i=1}^{n} \left(\hat{f}_j^{(l+1)}(x_{ij}) - \hat{f}_j^{(l)}(x_{ij}) \right)^2}{\sum_{i=1}^{n} \left(\hat{f}_j^{(l)}(x_{ij}) \right)^2} \leq \delta$ **então**
      retorna $\hat{\mathbf{f}}_j$
    fim
  fim
fim

Algoritmo 2: Algoritmo de *backfitting*.

2.3.3 Estimação intervalar

A construção de intervalos de confiança, no contexto dos GAMs, é uma tarefa complicada, uma vez que os termos do preditor correspondem a funções suavizadoras. Para tal, é necessário conhecer a distribuição de amostragem, exata ou assintótica, dos estimadores das funções parciais (Hastie e Tibshirani, 1990). A tarefa complica-se ainda mais tendo em conta a natureza iterativa do algoritmo de *scores* local, pelo que a solução passa pela obtenção de matrizes de suavização $\mathbf{G}_j, n \times n$, para cada função parcial $\boldsymbol{f}_j$, $j = 1, ..., p$, de forma a que genericamente se obtém:

$$\hat{\boldsymbol{f}}_j = \mathbf{G}_j \mathbf{z}, \tag{2.76}$$

em que $\mathbf{z}$ representa a variável dependente ajustada, obtida na última iteração. Desta forma, a matriz de covariâncias do estimador de uma função parcial genérica j pode ser expressa pela seguinte equação:

$$Cov(\hat{\boldsymbol{f}}_j) \approx \phi \boldsymbol{G_j} Cov(\boldsymbol{z}) \boldsymbol{G_j}^T, \tag{2.77}$$

podendo ser estimada por:

$$\widehat{Cov}(\hat{\boldsymbol{f}}_j) \approx \hat{\phi} \boldsymbol{G_j} \boldsymbol{W}^{-1} \boldsymbol{G_j}^T, \tag{2.78}$$

em que $\boldsymbol{W}$ representa uma matriz de pesos diagonal, obtida na última iteração do algoritmo para estimação de *scores* local e ϕ representa um parâmetro de dispersão que, no caso de ser desconhecido, deverá também ser estimado. Por outro lado verifica-se que:

$$\hat{\boldsymbol{f}}_j \overset{a}{\sim} N(\boldsymbol{f}_j, \hat{\phi} \boldsymbol{G_j} \boldsymbol{W}^{-1} \boldsymbol{G_j}^T), \tag{2.79}$$

podendo-se estabelecer a seguinte aproximação (Gu, 1992):

$$\hat{\phi} \boldsymbol{G_j} \boldsymbol{W}^{-1} \boldsymbol{G_j}^T \approx \hat{\phi} \boldsymbol{G_j} \boldsymbol{W}^{-1}. \tag{2.80}$$

Assim, pode-se obter de forma aproximada o seguinte intervalo de confiança para $f(x_{ij})$ (com grau de confiança $1 - \alpha$):

$$\hat{f}(x_{ij}) \pm Z_{1-\frac{\alpha}{2}} \sqrt{G_j^i W_j^{-1}}, \tag{2.81}$$

em que $Z_{1-\frac{\alpha}{2}}$ representa o quantil de probabilidade de uma Gaussiana padronizada e G_j^i, bem como W_j^{-1}, representam o i-ésimo elemento da respetiva diagonal principal de $\boldsymbol{G_j}$ e $\boldsymbol{W}^{-1}$. Para um maior detalhe deste processo, nomeadamente para a estimação da matriz genérica $\boldsymbol{G}_j$, pode-se consultar Hastie e Tibshirani (1990).

2.4 Modelos de regressão com função de ligação flexível

A função de ligação estabelece a relação entre o preditor linear e o valor esperado da saída do modelo. A sua determinação *a priori* pode conduzir a uma assunção abusiva da relação atrás referida, na medida em que pode não ser a mais adequada. Um destes exemplos é o da análise da função de ligação no caso de um conjunto de dados com variável de interesse dicotómica. Para este caso, assume-se *a priori* na generalidade dos casos, uma relação logística entre o valor esperado e o preditor linear (fig. 2.1).

Essa assunção, decorre, em primeiro lugar, da necessidade de limitar o valor esperado $0 < \mu < 1$ e, assim, desta forma garantir que o valor médio da variável resposta se encontre dentro do domínio da distribuição.

No entanto, em contextos mais específicos, são igualmente assumidas outras relações *a priori*, nomeadamente as correspondentes à função *probit* (ver eq. 2.12) e à função complementar log-log (ver eq. 2.13). A primeira (ver Fig. 2.2) é frequentemente utilizada em estudos toxicológicos, onde normalmente existe uma variável explicativa correspondente à dosagem de um determinado produto tóxico, e onde é especificado um nível de tolerância t, a partir do qual se verifica que o produto causa a morte do indivíduo (Agresti, 2003). A segunda é muito utilizada no estudo de organismos infecciosos com probabilidade de êxito $Y = 1$ (ver Fig. 2.3).

Porém, em determinados casos, nomeadamente nos modelos cuja curva de probabilidade associada se pronuncia de forma assimétrica, a má especificação da relação, pela utilização, por exemplo, de funções simétricas (*e.g.* *logit* e *probit*) pode conduzir a um viés substancial.

Assim, justifica-se olhar para a função de ligação como sendo desconhecida, e por isso, também sujeita a estimação tal como os restantes parâmetros do modelo. Mais detalhes sobre o problema da má especificação da função de ligação podem ser encontrados em Li e Duan (1989).

Por fim, todos os modelos abordados neste estudo podem ser utilizados com qualquer tipo de variável resposta, desde que a sua função de distribuição pertença à família exponencial. No entanto, neste estudo iremos abordar apenas casos orientados para a quantificação do risco de mortalidade em que a variável resposta é binária, dado ser uma situação muito comum na área

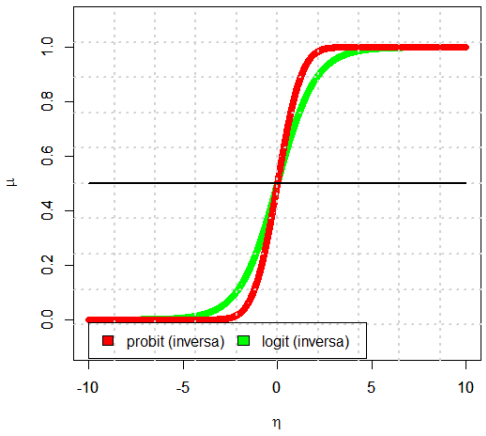

Figura 2.2: Inversa das funções probit e logit.

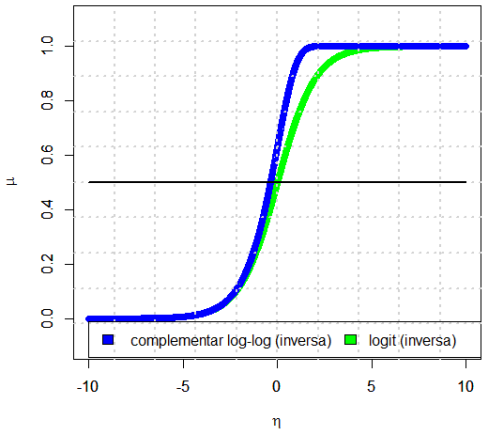

Figura 2.3: Inversa das funções complementar log-log e logit.

clínica, nomeadamente na Medicina Intensiva.

2.4.1 Estimação da função de ligação não paramétrica em modelos de regressão

Outra abordagem no capítulo dos GLMs passa pela estimação da função de ligação bem como da sua derivada recorrendo a suavizadores do tipo Kernel (Weisberg e Welsh, 1994).

No entanto, os custos computacionais destes modelos são maiores, pelo que há que equacioná-los face às vantagens que estes métodos oferecem, nomeadamente quanto à sua maior flexibilidade distribucional.

Um dos métodos mais usados para a estimação de GLMs com função de ligação desconhecida, passa pela implementação de um modelo SIM (*Single Index Model*) (Horowitz, 2009) e que pode ser formulado da seguinte forma:

$$E[Y|\mathbf{X}] = h(f_\beta(\mathbf{X})). \tag{2.82}$$

Neste caso, h representa uma função de ligação desconhecida e f uma função que agrega a combinação linear $\mathbf{X}\boldsymbol{\beta}$ num índice univariado, em que normalmente $f_\beta(\mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$.

À primeira vista, o facto de se utilizar uma função agregadora (índice), conduz à mitigação da "maldição da dimensionalidade", para além de facilitar a interpretabilidade que normalmente se apresenta como um problema nos modelos não paramétricos.

Os passos para estimação de um SIM são os seguintes:

1. Obter as estimativas de $\boldsymbol{\beta}$.
2. Obter as estimativas $f_{\hat{\beta}}(\mathbf{X})$ dos valores do índice $f_\beta(\mathbf{X})$.
3. Estimar a função de ligação $h(.)$ utilizando um método de regressão não paramétrico, como por exemplo, o estimador de Nadaraya-Watson.

Para um estudo mais profundo sobre este método, ver Ichimura (1993) ou Berkane et al. (1994) que utilizam um método iterativo para estimar $\boldsymbol{\beta}$. Em alternativa, podem-se utilizar métodos diretos para a estimação de $\boldsymbol{\beta}$, recorrendo aos métodos da derivada média (ADE e WADE - *Average Derivative* e *Weighted Average Derivative*) (Powell et al., 1989).

Relativamente à estimação da função de ligação nos GAMs importa salientar os estudos de Hastie e Tibshirani (1990) em que é proposta a estimação

da função de ligação de um GAM com recurso a métodos não paramétricos, mais concretamente através da estimação das funções parciais através do método Gauss-Newton na construção de uma versão generalizada do algoritmo de *scores* local. De igual forma, é de destacar outros estudos, nomeadamente o de Horowitz e Mammen (2004), em que são utilizados suavizadores do tipo Kernel para a estimação não paramétrica de um GAM com função de ligação desconhecida, e Roca-Pardiñas et al. (2004), onde é abordada a estimação das funções parciais, bem como da função de ligação de um GAM com resposta binária através de uma generalização do algoritmo de *scores* local (dando origem ao modelo G-GAM - *Generalized GAM*).

De uma forma resumida, o modelo é estimado iterativamente em dois passos. Num primeiro passo, fixa-se a função de ligação h e obtém-se a estimativa do preditor linear $\hat{\eta}$, através do algoritmo de *scores* local. Num segundo passo, a partir da estimativa obtida do preditor linear, estima-se a função de ligação. Estes passos são realizados iterativamente até se atingir um determinado critério de convergência (Roca-Pardiñas et al., 2004).

Quanto ao problema da identificabilidade, este é mitigado, recorrendo à imposição de restrições (Muggeo e Ferrara, 2008) e que podem passar por:

- Normalizar os coeficientes de forma a que $\|\boldsymbol{\beta}\| = 1$ (como proposto em Carroll et al. (1998), Yan Yu (2002) e Huang et al. (1188)),
- Padronizar o índice de forma a que $\sum_{i=1}^{n} \eta_i = 0$ e $\frac{\sum_{i=1}^{n} \eta_i^2}{n} = 1$ (como proposto em Green e Silverman (1993) e Cadarso-Suárez et al. (2005)),
- Fixar um dos coeficientes da regressão, $\beta = 1$ (como proposto em Härdle (2004)).

2.4.2 Estimação da função de ligação paramétrica em modelos de regressão

Dada a popularidade do modelo logístico devido à sua simplicidade, robustez, bem como à facilidade na interpretação dos parâmetros, procurou-se desenvolver várias generalizações deste modelo de forma a assegurar um melhor desempenho, apesar da escolha da função de ligação por vezes não ser a mais adequada. Neste contexto, e tomando por base este modelo, foram introduzidas famílias de transformações potência de forma a controlar os desvios

simétricos ou assimétricos do modelo logístico, abrindo caminho ao aparecimento de vários estudos que propõem classes de funções e em que a escolha da função de ligação depende de um ou mais parâmetros.

De entre esses estudos, existem várias propostas de classes de funções para uma variável resposta binária. Assim, destacam-se os estudos em que a escolha da função de ligação depende de um parâmetro como é o caso de Aranda-Ordaz (1981), Guerrero e Johnson (1982) e Morgan (1985) ou de dois parâmetros como é o caso de Prentice (1976), Pregibon (1980), Stukel (1988) e Czado (1992).

Esta solução apresenta-se como um compromisso entre a escolha da função de ligação *a priori* e a estimação não paramétrica da função desconhecida, baseada na assunção de que a melhor função de ligação pertence a uma classe de funções de ligação paramétrica.

Formalmente, os modelos de regressão com função de ligação flexível aludidos nesta secção são definidos pela seguinte expressão:

$$E(Y|X) = h(\eta, \boldsymbol{\psi}), \tag{2.83}$$

em que:

- Y representa a variável resposta,
- X é a matriz de especificação do modelo,
- h é a função de ligação pertencente à família $\{h(., \boldsymbol{\psi}) : \boldsymbol{\psi} \in \boldsymbol{\Psi}\}$,
- $\boldsymbol{\psi}$ representa o parâmetro ou vetor de parâmetros desconhecido, de que a função de ligação depende.

No caso de um GLM com função de ligação paramétrica, existem vários métodos de estimação. Um desses métodos, nomeadamente o proposto por Czado (1992), baseia-se na estimação conjunta dos parâmetros $\boldsymbol{\beta}$ da componente sistemática e dos z parâmetros da componente da função de ligação $\boldsymbol{\psi} = \psi_1 ... \psi_z$, através da maximização do logaritmo da função verosimilhança. Se o parâmetro ou vetor de parâmetros verdadeiro $\boldsymbol{\psi}_0$ se encontra dentro da família de funções de ligação escolhidas, e se se verificarem as condições de regularidade, então a estimativa de máxima verosimilhança conjunta $\hat{\boldsymbol{\delta}} = (\hat{\boldsymbol{\psi}}, \hat{\boldsymbol{\beta}})$ de $\boldsymbol{\delta} = (\boldsymbol{\psi}, \boldsymbol{\beta})$ é consistente e eficiente (Czado, 1992).

Assim, a partir da eq. 2.1 deduz-se a expressão do logaritmo da função de verosimilhança, a maximizar:

$$l(\boldsymbol{\delta}) = log(L(\boldsymbol{\delta})) = \sum_{i=1}^{n} [\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)]. \tag{2.84}$$

Considerando $\mu_i = b'(\theta_i) = h(\theta_i, \boldsymbol{\psi})$, p o número de covariáveis e z o número de parâmetros dos quais depende a função de ligação, pode-se deduzir o seguinte sistema de equações:

$$\frac{\partial l(\boldsymbol{\delta})}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial l_i(\boldsymbol{\delta})}{\partial \beta_j} = \sum_{i=1}^{n} \frac{(y_i - \mu_i)}{var(Y_i)} \frac{\partial \mu_i}{\partial \eta_i} x_{ij}, \tag{2.85}$$

em que $j = 1, ..., p$.

$$\frac{\partial l(\boldsymbol{\delta})}{\partial \psi_k} = \sum_{i=1}^{n} \frac{\partial l_i(\boldsymbol{\delta})}{\partial \psi_k} = \sum_{i=1}^{n} \frac{(y_i - \mu_i)}{var(Y_i)} \frac{\partial \mu_i}{\partial \psi_k}, \tag{2.86}$$

em que $k = 1, ..., z$.

Para resolver este sistema de equações pode-se utilizar o método iterativo de mínimos quadrados ponderados.

Existem outros métodos desenvolvidos para estimar os coeficientes $\boldsymbol{\beta}$, bem como os parâmetros da função de ligação, nomeadamente através da utilização do algoritmo delta ou através da utilização de gráficos de perfil da log-verosimilhança ou do desvio. Neste último caso, os valores dos parâmetros da função de ligação são colocados numa grelha e escolhe-se o que obtiver maior log-verosimilhança ou menor desvio (gráficos de perfil).

Este último método de estimação foi proposto por Papoila (2006) num estudo sobre GAMs com função de ligação flexível paramétrica em que se utilizou como exemplo um GAM com função de ligação baseada na família de transformações assimétricas de Aranda-Ordaz. Nesta classe de funções encontram-se incluídas, como casos particulares, a função *logit* para $\psi = 1$ e a função complementar log-log para $\psi \to 0$.

Esta família de transformações é definida da seguinte forma:

$$h(\eta, \psi) = aranda(\eta, \psi) = \begin{cases} 1 - (1 + \psi e^{\eta})^{-\frac{1}{\psi}} & \text{se } \psi e^{\eta} > -1 \\ 1 & \text{se } \psi e^{\eta} \leq -1, \end{cases} \tag{2.87}$$

em que $h(.)$ representa a função de ligação do modelo, η a componente sistemática e ψ é o parâmetro da função de ligação.

Na fig. 2.4, pode-se observar a flexibilidade da classe de funções da família de Aranda-Ordaz, tendo em conta valores específicos de ψ.

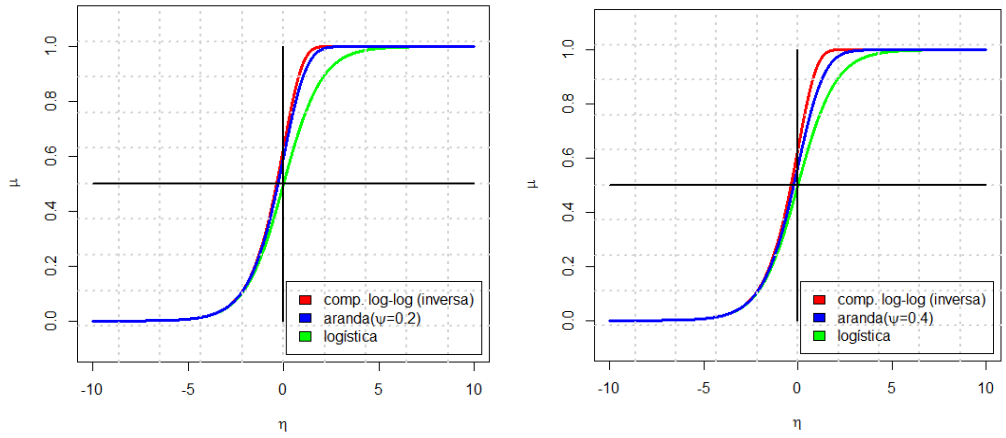

Figura 2.4: Representação gráfica das funções: inversa da complementar log-log, logística e Aranda-Ordaz com $\psi = 0.2$, $\psi = 0.4$.

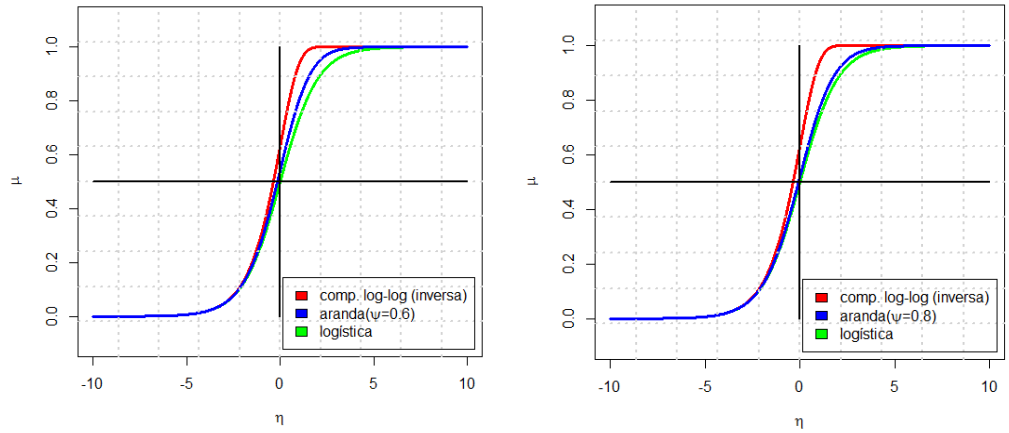

Figura 2.5: Representação gráfica das funções: inversa da complementar log-log, logística e Aranda-Ordaz com $\psi = 0.6$, $\psi = 0.8$.

Capítulo 3

Amostragem e avaliação de modelos

3.1 Introdução

A avaliação do desempenho global do modelo pode ser efetuada não só para aferir da sua adequação como também para o comparar com outros modelos, de forma a que se possa selecionar o que melhor infere as características da população em estudo. No entanto, torna-se necessário garantir a escolha do modelo mais generalista e, consequentemente, menos otimista. Para tal, é necessário recorrer a métodos de amostragem para posterior avaliação do modelo. Dada a habitual inexistência de dados externos para serem utilizados como amostra de teste é usual recorrer a processos de amostragem (Gama et al., 2012), através de estratégias que iremos abordar na secção 3.2.

Após a definição do processo de amostragem, é estimado o modelo e, posteriormente, medido o seu desempenho através de métricas bem definidas. Para tal existem várias medidas globais de desempenho que permitem avaliar e comparar modelos. Algumas dessas métricas são elencadas na secção 3.3.

3.2 Técnicas de Amostragem

Em alguns casos, a limitação que advém da pequena dimensão da amostra de dados vem dificultar não só o treino do modelo, como também a sua avaliação. No caso específico das redes neuronais, este problema torna-se ainda mais saliente dado que, no contexto da estimação dos pesos sinápticos da

rede, para além da existência de um conjunto de observações para treino do modelo, deverá ter-se em conta mais dois conjuntos disjuntos. Um deles terá como destino a validação do modelo, onde se inclui o processo de paragem da aprendizagem (conduzindo a estimativas menos otimistas) evitando o sobre ajustamento da rede à amostra de treino. A partir deste subconjunto de validação, é também selecionado o modelo quanto à sua topologia. No entanto, as estimativas obtidas com base na amostra de validação continuam a ser otimistas, pelo que se torna necessária a utilização de uma amostra adicional (teste), cujos dados não participaram nem no processo de aprendizagem nem no processo de validação. Assim sendo, este último subconjunto servirá para mimetizar uma amostra de teste externa ao modelo.

Seguir-se-á a descrição de alguns dos principais métodos de amostragem.

3.2.1 Método *Holdout*

Este método de amostragem consiste em dividir o conjunto de dados em duas amostras, uma para treino (desta amostra retira-se uma outra para validação no caso de o modelo a estimar ser uma rede neuronal) e outra para teste conforme se pode observar na fig. 3.1. É usual utilizar-se $\frac{2}{3}$ das observações para a primeira e as restantes para a última.

A vantagem de se utilizar o método *Holdout* tem a ver com a rapidez computacional. No entanto, apresenta como desvantagem uma grande variabilidade da medida de desempenho da amostra de teste, dado esta estar dependente da partição feita entre a amostra de treino (e validação) e a amostra de teste (Kohavi, 1995).

Assim, uma forma de tornar mais robusto o processo de validação passa pela introdução de procedimentos de reamostragem, *i.e.* pela geração de um conjunto de amostras a partir do conjunto de observações disponíveis (amostra original). Na prática, os métodos de reamostragem consideram a amostra original como sendo uma população finita de onde se retiram aleatoriamente outras amostras para inferência. No entanto, estes métodos apresentam um custo computacional mais elevado, pelo que a escolha do método de reamostragem terá de ser efetuada tendo em conta os recursos disponíveis. Seguidamente serão apresentados alguns destes métodos.

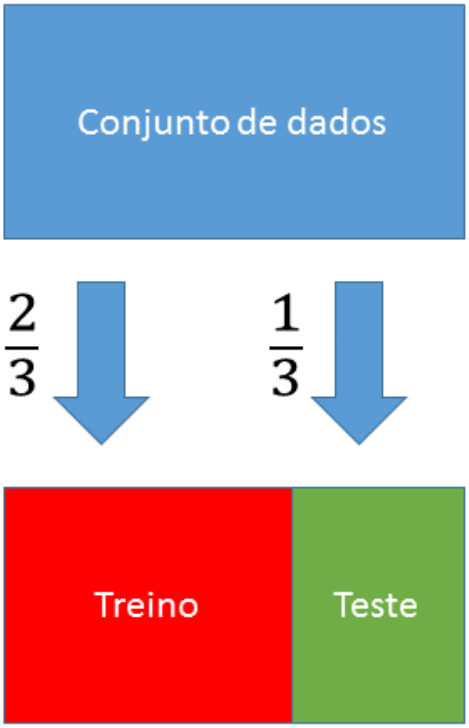

Figura 3.1: Método de amostragem *Holdout*.

3.2.2 Validação cruzada

O método da validação cruzada (*cross-validation*) consiste em dividir um conjunto de dados $\mathbf{S}$ em k subconjuntos (*folds*) mutuamente exclusivos $\mathbf{S}_1, ..., \mathbf{S}_k$ com dimensões aproximadamente iguais. O modelo é submetido a treino, validação (no caso de uma rede neuronal) e teste k vezes, sendo que em cada iteração $t \in \{1, ..., k\}$, o modelo é treinado com todos os indivíduos (inclui-se neste processo a validação) exceto aqueles que pertencem ao conjunto $\mathbf{S}_t$. Este último é utilizado para efetuar o teste do modelo (mimetizando uma amostra externa). No final, as estimativas produzidas pelos k subconjuntos de teste, já menos otimistas, poderão servir para avaliar o modelo. Este processo é ilustrado na fig. 3.2.

Os valores típicos de k vão de 3 a 10 (Rasmussen e Williams, 2005), dependendo do custo computacional que os recursos permitirem. No extremo em que $k = n$, este método é conhecido por *Leave-One-Out Cross-Validation* (LOO-CV) implicando, como já referido, custos computacionais elevados. Neste estudo, tendo em conta os recursos disponíveis, optou-se pela utiliza-

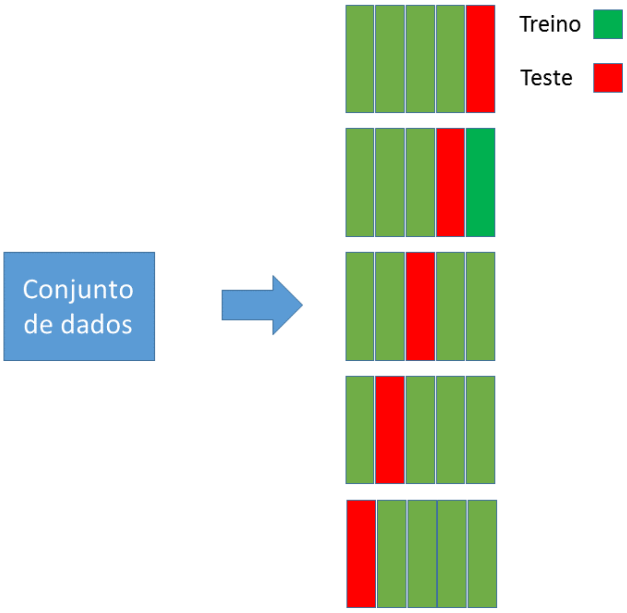

Figura 3.2: Exemplo do processo de validação cruzada em que os dados se encontram divididos em 5 subconjuntos (*5-fold cross-validation*).

ção de uma amostragem do tipo *5-Fold Cross Validation*.

3.2.3 Amostragem com reposição - *Bootstrap*

Neste método são geradas B amostras de treino a partir da amostra original $\mathbf{S}$ que, por sua vez, foi retirada de uma determinada população. As observações são selecionadas aleatoriamente com reposição a partir dessa amostra. Isto significa que uma determinada observação pode ser encontrada numa amostra mais de uma vez.

A ideia subjacente é a obtenção de um número elevado de amostras $\mathbf{S}^{*1}, ..., \mathbf{S}^{*B}$ a partir das quais se estima o desempenho do modelo. Se, por exemplo, considerarmos que a população de onde se extraiu a amostra original depende de um parâmetro θ, através deste tipo de amostragem obtém-se a distribuição de $\hat{\theta}(\mathbf{S}) : \hat{\theta}(\mathbf{S}^{*1}), ..., \hat{\theta}(\mathbf{S}^{*B})$.

Este método mostra ser exigente em termos computacionais pelo que é geralmente aplicado a amostras de dados de pequena dimensão.

3.3 Avaliação de modelos

3.3.1 Função desvio

A função desvio (*deviance*) é bastante utilizada nos modelos de regressão para determinar a qualidade da adaptação do modelo aos dados. O desvio é obtido por comparação do modelo proposto com o modelo saturado. Este último, por definição, possui tantos parâmetros quantos os dados observados, pelo que não dispõe da estrutura linear conforme descrita na eq. 2.2, tendo como matriz de especificação a matriz identidade ($n \times n$ em que n corresponde à dimensão da amostra). Neste modelo, as estimativas de máxima verosimilhança são as próprias observações (*i.e.* $\hat{\mu_i} = y_i$) (Amaral Turkman, 2000).

No pólo oposto encontra-se o modelo nulo em que todas as variáveis resposta Y_i possuem o mesmo valor médio μ. Isso significa que a matriz deste modelo é um vetor unitário.

A comparação entre o modelo em causa e o modelo saturado poderá ser efetuada, recorrendo à estatística de razão de verosimilhanças de Wilks que se traduz pela seguinte expressão (Hastie e Tibshirani, 1990):

$$D^*(\hat{\boldsymbol{\mu}}; \mathbf{y}) = 2\{l(\hat{\boldsymbol{\beta}}_s) - l(\hat{\boldsymbol{\beta}})\}, \tag{3.1}$$

em que $l(\hat{\beta}_s)$ representa o máximo da função log-verosimilhança do modelo saturado e $l(\hat{\beta})$ representa o máximo da função log-verosimilhança do modelo em avaliação. No entanto, ao desenvolver-se a equação anterior com base na eq. 2.16, verifica-se que a função D^* depende do parâmetro de dispersão ϕ como se pode verificar pela seguinte expressão:

$$D^*(\hat{\boldsymbol{\mu}}; \mathbf{y}) = 2\sum_{i=1}^{n} \frac{y_i(\theta_i^s - \theta_i) - [b(\theta_i^s) - b(\theta_i)]}{a(\phi)}. \tag{3.2}$$

O parâmetro de dispersão ϕ é, geralmente, desconhecido e tem que ser estimado. Em alternativa, pode-se considerar que na maioria das situações se observa $a(\phi_i) = \frac{\phi}{\omega_i}$, em que ω_i é uma constante conhecida e o parâmetro ϕ é comum a todas as observações. Assim, a eq. 3.2 toma a seguinte forma:

$$D^*(\hat{\boldsymbol{\mu}};\mathbf{y}) = 2\sum_{i=1}^{n} \frac{\omega_i[y_i(\theta_i^s - \theta_i) - [b(\theta_i^s) - b(\theta_i)]]}{\phi} = \frac{D(\hat{\boldsymbol{\mu}};\mathbf{y})}{\phi}, \tag{3.3}$$

em que a função $D^*(\hat{\boldsymbol{\mu}};\mathbf{y})$ é designada por desvio reduzido, e $D(\hat{\boldsymbol{\mu}};\mathbf{y})$ é o desvio do modelo corrente (Amaral Turkman, 2000).

Analisando o desvio, verifica-se que $D(\hat{\boldsymbol{\mu}};\mathbf{y}) \geq 0$, e à medida que se introduzem covariáveis no modelo nulo, o seu valor decresce até zero quando se atinge o modelo saturado.

Uma outra propriedade interessante do desvio é a possibilidade de poder ser utilizado para comparar dois modelos encaixados (em que um é submodelo do outro). Essa comparação pode ser feita com base na diferença dos desvios de cada modelo (Amaral Turkman, 2000):

$$-2(l_{M_2}(\hat{\boldsymbol{\beta}}_2) - l_{M_1}(\hat{\boldsymbol{\beta}}_1)) = \frac{D(\hat{\boldsymbol{\mu}}_2;\mathbf{y}) - D(\hat{\boldsymbol{\mu}}_\mathbf{1};\mathbf{y})}{\phi} \overset{a}{\sim} {\chi^2}_{p_1-p_2}. \tag{3.4}$$

em que M_1 e M_2 representam dois modelos, encontrando-se o último encaixado no primeiro. p_1 e p_2 correspondem, respetivamente, à dimensão dos vetores de coeficientes $\hat{\boldsymbol{\beta}}_1$, $\hat{\boldsymbol{\beta}}_2$ e $\hat{\boldsymbol{\mu}}_\mathbf{1}$, $\hat{\boldsymbol{\mu}}_\mathbf{2}$ representam as estimativas de máxima verosimilhança de $\boldsymbol{\mu}_\mathbf{1}$ e $\boldsymbol{\mu}_\mathbf{2}$.

Relativamente à distribuição de amostragem do desvio, só em determinadas condições é que esta é conhecida, nomeadamente nos casos em que a variável resposta segue uma distribuição normal ou uma gaussiana inversa. De facto, nesta situação sabe-se que a distribuição exata é a de um qui-quadrado com $n - p$ graus de liberdade, sendo p o número de parâmetros a estimar sob H_0. Porém, noutros casos, o desvio só pode ser aproximado a uma distribuição qui-quadrado, sendo que essa aproximação é geralmente má, mesmo para amostras de grande dimensão (Amaral Turkman, 2000).

No caso da variável resposta $Y_i \sim B(n_i, \pi_i)/n_i$, o desvio é determinado tendo em atenção que o máximo da função log-verosimilhança do modelo corrente é dado pela seguinte expressão:

$$l(\hat{\boldsymbol{\beta}}) = \sum_{i=1}^{n} \left(n_i \left[log(1 - \hat{\pi}_i) + y_i log\left(\frac{\hat{\pi}_i}{1 - \hat{\pi}_i} \right) \right] + log\binom{n_i}{n_i y_i} \right). \tag{3.5}$$

Para o caso particular do modelo saturado $\hat{\pi}_i = y_i$, a equação é reescrita na seguinte forma:

$$l(\hat{\boldsymbol{\beta}}_s) = \sum_{i=1}^{n} \left(n_i \left[log(1-y_i) + y_i log\left(\frac{y_i}{1-y_i}\right) \right] + log\binom{n_i}{n_i y_i} \right). \tag{3.6}$$

.

Aplicando a eq. 3.1 com os máximos das funções de log-verosimilhança dos modelos corrente e saturado, anteriormente descritos, a expressão resultante para a função desvio tem a seguinte forma:

$$D^*(\hat{\boldsymbol{\mu}}; \mathbf{y}) = 2 \sum_{i=1}^{n} n_i \left[y_i log\left(\frac{y_i}{\hat{\pi}_i}\right) + (1-y_i) log\left(\frac{1-y_i}{1-\hat{\pi}_i}\right) \right]. \tag{3.7}$$

Tendo em atenção a eq. 2.11, verifica-se que $\hat{\pi} = \hat{\mu}$, pelo que a equação anterior resulta na seguinte:

$$D^*(\hat{\boldsymbol{\mu}}; \mathbf{y}) = 2 \sum_{i=1}^{n} n_i \left[y_i log\left(\frac{y_i}{\hat{\mu}_i}\right) + (1-y_i) log\left(\frac{1-y_i}{1-\hat{\mu}_i}\right) \right]. \tag{3.8}$$

A tabela 3.1 mostra as expressões do desvio para outras distribuições (Amaral Turkman, 2000).

3.3.2 Estatística de Pearson generalizada

A estatística de Pearson generalizada também permite aferir a adequabilidade do modelo e é, geralmente, definida pela seguinte equação:

$$X^2 = \sum_{i=1}^{n} \frac{\omega_i (y_i - \hat{\mu}_i)^2}{V(\hat{\mu}_i)}, \tag{3.9}$$

em que $V(\hat{\mu}_i)$ representa a função de variância da distribuição de Y e é equivalente a $b''(\theta_i)$.

Modelo	Desvio
normal	$\sum_{i=1}^{n}(y_i-\hat{\mu}_i)^2$
Poisson	$2\left[\sum_{i=1}^{n} y_i log\left(\frac{y_i}{\hat{\mu}}\right) - \sum_{i=1}^{n}(y_i-\hat{\mu}_i)\right]$
gama	$2\sum_{i=1}^{n}\left[-log\left(\frac{y_i}{\hat{\mu}}\right) + \frac{y_i-\hat{\mu}_i}{\hat{\mu}_i}\right]$
gaussiano inverso	$\sum_{i=1}^{n}\frac{(y_i-\hat{\mu}_i)^2}{y_i\hat{\mu}_i)^2}$

Tabela 3.1: Função desvio para os modelos normal, Poisson, gama e gaussiano inverso.

Assim, ao se considerar, por exemplo, a distribuição normal, a respetiva função de ligação canónica θ é a função identidade (ver Tabela 2.1). Uma vez que se verifica através da eq. 2.1 que $b(\theta) = \frac{\theta^2}{2}$, $E(Y) = b'(\theta) = \theta$ e portanto $V(\hat{\mu}_i) = b''(\theta) = 1$, chegamos à seguinte expressão:

$$X^2 = \sum_{i=1}^{n}(y_i - \hat{\mu}_i)^2. \tag{3.10}$$

Na tabela 3.2 encontram-se representadas as expressões da função de variância de algumas das distribuições mais utilizadas:

Distribuição de Y	θ	$b(\theta)$	$E(Y) = b'(\theta)$	$V(\mu) = b''(\theta)$
Normal ($N(\mu, \sigma^2)$)	μ	$\frac{\theta^2}{2}$	θ	1
Binomial/n ($B(n,\pi)/n$)	$log(\frac{\pi}{\pi-1})$	$log(1+e^{\theta})$	$\pi = \frac{e^{\theta}}{1+e^{\theta}}$	$\pi(1-\pi)$
Poisson ($P(\lambda)$)	$log(\lambda)$	e^{θ}	$\lambda = e^{\theta}$	λ
Gama ($Ga(\nu, \frac{\nu}{\mu})$)	$-\frac{1}{\mu}$	$-log(-\theta)$	$\mu = -\frac{1}{\theta}$	μ^2
Gaussiana Inversa ($IG(\mu, \sigma^2)$)	$-\frac{1}{2\mu^2}$	$-\sqrt{-2\theta}$	$(-2\theta)^{-\frac{1}{2}}$	μ^3

Tabela 3.2: Principais distribuições da família exponencial com a respetiva função de variância.

Como referido anteriormente, tanto a função desvio como a estatística de Pearson generalizada podem ser utilizadas para avaliar a qualidade de ajustamento do modelo, através da quantificação da semelhança entre os valores

observados da variável resposta e os respetivos valores estimados.

Em geral, a distribuição assintótica dessas duas estatísticas é conhecida, podendo ser aproximada a um χ^2 com $n-p$ graus de liberdade, sendo exata no caso em que a distribuição da variável resposta é Gaussiana.

Quando o número de observações por padrão for muito pequeno, ou se uma das covariáveis for contínua, a distribuição de ambas as estatísticas é desconhecida, pondo em causa a determinação dos valores p.

3.3.3 Calibração

A calibração de um modelo possui uma estreita relação com a sua capacidade de generalização. No entanto, para este efeito, deve-se ter o cuidado em utilizar uma amostra que não tenha sido usada no treino do modelo (*e.g.* utilizar qualquer uma das técnicas de amostragem já referidas anteriormente na secção 3.2) de forma a evitar estimativas otimistas.

Como medida de calibração, pode-se utilizar o Erro Quadrático Médio (*Mean Squared Error - MSE*) em que para os casos de resposta binária, a estimação desta medida é dada por:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_i)^2. \tag{3.11}$$

n representa o número de indivíduos, y_i o valor observado e μ_i o valor estimado. Coincide com a estimação do *score* de Brier, que corresponde também a uma medida de calibração (Ohno-Machado, 1996).

3.3.4 Discriminação

Para estudar o desempenho discriminativo de um modelo pode-se recorrer à área sob a *Receiver Operating Characteristic (ROC) curve* (*AUC*). Esta medida é adequada para um modelo em que a resposta é dicotómica. Neste caso, cada observação possui uma probabilidade π associada ao evento de interesse. Para cada ponto de corte possível c, tem-se que $Y_c = 0$, se $\pi < c$ ou $Y_c = 1$, se $\pi > c$, correspondendo à matriz de confusão descrita na fig. 3.3:

A proporção de verdadeiros positivos (VP), denominada por sensibilidade, é dada por $\frac{VP}{VP+FP}$, em que FP representa a proporção de falsos positivos. A proporção de verdadeiros negativos (VN) corresponde à especificidade e é

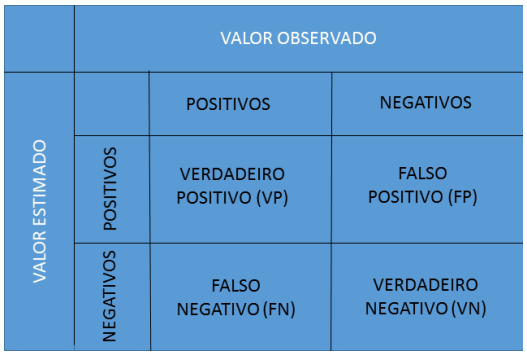

Figura 3.3: Matriz de confusão correspondente a um ponto de corte c.

dada por $\frac{VN}{VN+FN}$, em que FN representa a proporção de falsos negativos. Geralmente, a sensibilidade e a especificidade são características difíceis de conciliar, dado que a determinação de um ponto de corte ideal depende do compromisso entre estas duas medidas. No entanto, a curva ROC permite visualizar graficamente esse compromisso para cada valor de corte.

Para construir a curva ROC, no eixo das ordenadas coloca-se a sensibilidade e no eixo das abcissas a proporção de FP (1-especificidade). Para a determinação da AUC existem vários métodos, de entre os quais se destacam, a regra do trapézio (Fawcett, 2006), a estimação de máxima verosimilhança (Hanley e McNeil, 1982) ou a aproximação à estatística U de Wilcoxon-Mann-Whitney (Hanley e McNeil, 1982).

Uma $AUC = 0.5$ corresponde a uma discriminação fruto do acaso, enquanto que uma $AUC = 1$ corresponde a uma discriminação perfeita das classes envolvidas na resposta. Geralmente, consideram-se valores da AUC abaixo de 0.7 como correspondendo a uma fraca discriminação (Hanley e McNeil, 1982). Entre 0.7 e 0.8, a discriminação é razoável, entre 0.8 e 0.9 é boa e acima de 0.9 é excelente. Esta medida pode ser calculada sobre a amostra de teste, para uma avaliação menos otimista do modelo.

3.3.5 Critério de Informação de Akaike

O Critério de Informação de Akaike (*Akaike Information Criterion - AIC*) foi introduzido por Akaike (1973) e traduz-se numa medida que permite comparar modelos que, para além de possuírem a mesma distribuição da variável resposta, também se encontrem ajustados ao mesmo conjunto de dados.

Este critério é constituído por uma medida indicadora do ajustamento baseada na log-verosimilhança $log(L(\hat{\boldsymbol{\theta}}|y))$ do vetor de estimativas dos parâmetros do modelo, dado y e por um termo penalizador que depende da complexidade do modelo, traduzida pelo número de parâmetros associado W:

$$AIC = -2log(L(\hat{\boldsymbol{\theta}}|y)) + 2W. \tag{3.12}$$

A partir de um conjunto de modelos devidamente ajustados, o AIC pode ser utilizado para a seleção do modelo mais parcimonioso, *i.e.* aquele que apresenta um compromisso viés-variância mais otimizada (ver secção 4.8.1).

Posteriormente, foi proposto um critério alternativo desenvolvido sob o ponto de vista bayesiano (Schwarz, 1978), denominado por Critério de Informação Bayesiano (*Bayesian Information Criterion - BIC*), também conhecido por *Schwarz Bayesian Criterion* (SBC). Este critério vem referenciado como sendo assintoticamente consistente e capaz de identificar o modelo correto dentro do conjunto de modelos concorrentes.

Apesar de na literatura existirem várias formas de exprimir este critério, a definição genérica é dada por (Burnham e Anderson, 2003):

$$SBC = -2log(L(\hat{\boldsymbol{\theta}}|y)) + Wlog(n), \tag{3.13}$$

em que n representa a dimensão da amostra.

Capítulo 4

Rede Neuronal Artificial

4.1 Introdução

A rede neuronal artificial, inspirada no funcionamento do cérebro humano, pretende mimetizar a sua capacidade na construção de "modelos" a partir da observação de fenómenos externos. Essa mimetização tem por base uma estrutura de nós (neurónios) e arestas (sinapses). O número de neurónios e a forma como se encontram interligados definem a arquitetura de uma rede neuronal artificial.

Podem-se tipificar as redes neuronais em redes de alimentação direta (*feedforward*), em que a propagação do sinal é realizada no sentido da entrada para a saída da rede neuronal, ou recorrentes em que a topologia possui pelo menos um laço de realimentação (o sinal de saída é reconduzido para um ponto da rede)(Haykin, 1998). As arquiteturas mais utilizadas são do tipo *feedforward*, sendo a mais popular a denominada por Perceptrão Multicamada (*Multi Layer Perceptron* - MLP). O tema deste estudo tem como principal enfoque uma topologia alternativa - a Rede Neuronal Aditiva Generalizada (*Generalized Additive Neural Network* - GANN).

Para além da topologia da rede, existe um outro elemento fundamental ao bom desempenho de uma rede neuronal que é o processo de aprendizagem (treino da rede neuronal). Enquanto que o primeiro se encontra relacionado com a forma como os neurónios se encontram interligados, a aprendizagem engloba um conjunto de regras necessárias para o ajustamento dos pesos sinápticos da rede (o peso sináptico corresponde a uma ponderação que é aplicada ao sinal que atravessa a respetiva sinapse). Neste estudo, os processos de aprendizagem que irão ser abordados são de natureza supervisionada,

mais adequados a problemas de regressão. Neste tipo de aprendizagem, são conhecidas *a priori* as respostas correspondentes ao conjunto de dados de entrada, que servirão para supervisionar o processo de adaptação dos pesos sinápticos (aprendizagem supervisionada).

4.2 Arquitetura da rede neuronal

O neurónio artificial (assim como a sinapse) constitui um dos elementos básicos de uma rede neuronal artificial (Haykin, 1998), como já anteriormente referido. O seu papel consiste em combinar os sinais de entrada (normalmente através da operação de adição) e seguidamente aplicar uma função f - função de ativação (ver fig. 4.1) ao resultado dessa mesma combinação.

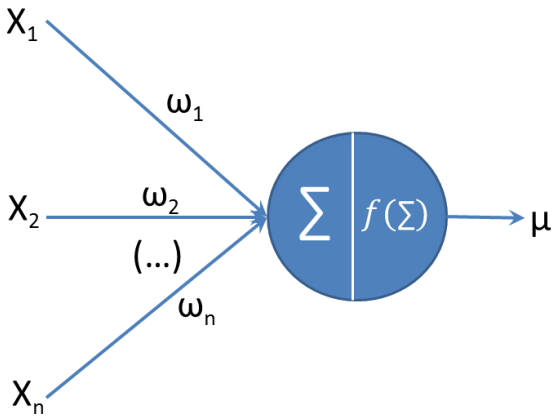

Figura 4.1: Modelo de um neurónio artificial com n sinapses de entrada.

O valor de saída constitui, assim, a resposta do neurónio para o conjunto das entradas. No caso de se definir a soma como a operação que combina as p entradas, a expressão que define a operação de um neurónio é a seguinte:

$$\mu = \sum_{j=1}^{p} X_j \omega_j, \tag{4.1}$$

em que X_j representa o valor de uma entrada genérica do neurónio, ponderado pelo peso sináptico ω_j que, por sua vez, pode tomar valores positivos ou negativos. O caso em que o peso é nulo equivale à ausência de sinapse.

Quanto à função de ativação a utilizar existem vários estudos, entre os quais destacamos o de Duch e Jankowski (1999) que apresentam uma análise sobre um conjunto largo de funções de ativação que se podem implementar no neurónio artificial. Nas primeiras redes neuronais, introduzidas por McCulloch e Pitts (1943), a função de ativação introduzida corresponde à função de Heaviside $\Theta(I, \theta)$, que se traduz pela seguinte expressão:

$$\Theta(I, \theta) = \begin{cases} 1 & I > \theta \\ 0 & I \leq \theta, \end{cases} \tag{4.2}$$

em que I representa a combinação das entradas do neurónio e θ o valor do limiar a partir do qual é decidido se a saída toma o valor 0 ou 1.

O facto deste tipo de função não ser diferenciável, irá limitar a utilização de alguns processos de aprendizagem, nomeadamente os que se baseiam no cálculo do gradiente. Assim sendo, torna-se necessária a utilização de outras funções, tais como a linear, a sigmoidal (onde se incluem, *e.g.* a tangente hiperbólica e a função logística $\frac{1}{1+e^{-x}}$) e a gaussiana. Esta última é utilizada numa variante - a Rede de função de base radial (RBF - *Radial Basis Functions*).

Os neurónios artificiais podem ser organizados por camadas, em que a topologia mais simples - o Perceptrão Simples - corresponde a uma rede neuronal constituída apenas por uma única camada (não se contando com a camada de entrada). Neste contexto, uma rede neuronal com duas ou mais camadas corresponde ao MLP. Seguidamente iremos caracterizar estas topologias.

4.3 Perceptrão Simples

O Perceptrão Simples define uma relação específica entre as covariáveis $(X_1, ..., X_p)$ e a variável resposta μ, através do vetor de pesos sinápticos $(\omega_0, \omega_1, ..., \omega_p)$, como se pode observar pela fig. 4.2.

Essa relação vem traduzida, para o i-ésimo indivíduo, pela seguinte expressão:

$$\mu_i = h(\omega_0 + \sum_{j=1}^{p} \omega_j x_{ij}), \tag{4.3}$$

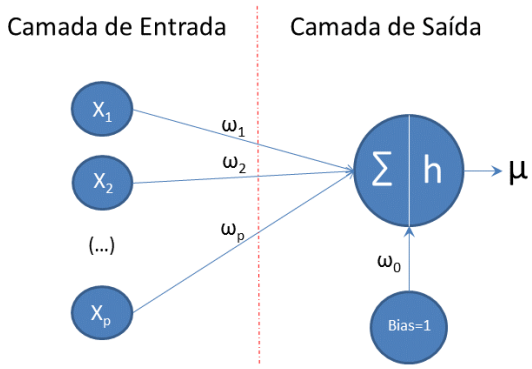

Figura 4.2: Perceptrão Simples.

em que h representa a função de ativação do neurónio de saída e ω_j, com $j = 0, 1, ..., p$, representa o peso associado a cada uma das sinapses do perceptrão.

Através de uma análise mais detalhada deste modelo, é possível constatar a sua equivalência a um Modelo Linear Generalizado (*Generalized Linear Model* - GLM [1]) (Tu, 1996). Com efeito, a função de ativação do neurónio de saída h corresponde à função de ligação do modelo de regressão. Por sua vez, ω_0 corresponde à ordenada na origem (*intercept* - β_0) que, juntamente com os restantes pesos $(\omega_1, ..., \omega_p)$, constituem o equivalente ao vetor de parâmetros $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)$ que fazem parte da componente sistemática de um GLM.

Embora esta topologia possa ser utilizada em problemas de regressão, dada a superfície de decisão se traduzir por um hiperplano, o modelo fica limitado a problemas linearmente separáveis. Este problema pode ser ultrapassado através da utilização de camadas intermédias (escondidas) entre a entrada e a saída da rede constituindo, assim, o MLP. Mais adiante irá ser demonstrado que, nesta última topologia, se se utilizar um número suficiente de neurónios em cada camada escondida, é possível aproximar uma superfície de decisão arbitrária utilizando redes com apenas uma camada intermédia.

[1] ver eq. 2.3

4.4 Perceptrão Multicamada

A popularidade do MLP no contexto dos modelos preditivos deve-se a algumas das suas características que a tornam mais acessível ao utilizador normal, nomeadamente, não ser necessário um profundo conhecimento na área da estatística para a utilizar. Qualquer conjunto de dados que possa ser analisado por métodos convencionais, nomeadamente através da regressão linear, pode ser também utilizado para o desenvolvimento de um modelo preditivo com base numa rede neuronal (Tu, 1996).

Como já atrás referido, nesta arquitetura os neurónios são organizados por camadas, em que um neurónio de uma determinada camada se encontra interligado com todos os neurónios das camadas anteriores. A camada de entrada é constituída por um número de neurónios separados, equivalente ao número de covariáveis em estudo. Por sua vez, as camadas intermédias por não serem diretamente observáveis, designam-se por camadas escondidas, e cada neurónio dispõe de uma sinapse extra ligada a um neurónio de valor de saída igual a um e designado por *bias* (viés). Esta sinapse não é diretamente influenciada por nenhuma covariável.

Na fig.4.3 encontra-se um exemplo de um MLP com uma camada escondida com q neurónios, p neurónios na camada de entrada e 1 neurónio na camada de saída.

Esta arquitetura, com as suas camadas intermédias, acrescenta maior flexibilidade ao modelo (eq. 4.4), em que para o i-ésimo indivíduo se tem:

$$\mu_i = h(\omega_0 + \sum_{j=1}^{q} \omega_j h(\omega_{0j} + \sum_{k=1}^{p} \omega_{jk} x_{ik})), \tag{4.4}$$

permitindo desta forma modelar qualquer função contínua, apenas com uma camada intermédia (escondida). Com efeito, esta característica é demonstrada pelo *teorema da aproximação universal* que passamos a citar (Hornik et al., 1989):

Teorema: Seja $\varphi(.)$ uma função contínua não constante, limitada e monótona crescente. Seja I_p um hipercubo unitário de dimensão p, $[0,1]^p$. O espaço das funções contínuas em I_p é denominado por $C(I_p)$. Então, dada qualquer função f $\in C(I_p)$ e $\epsilon > 0$, existe um inteiro M e conjuntos de constantes reais α_k, ν_k e ω_{jk} em que $k = 1, ..., q$ e $j = 1, ..., p$ tal que

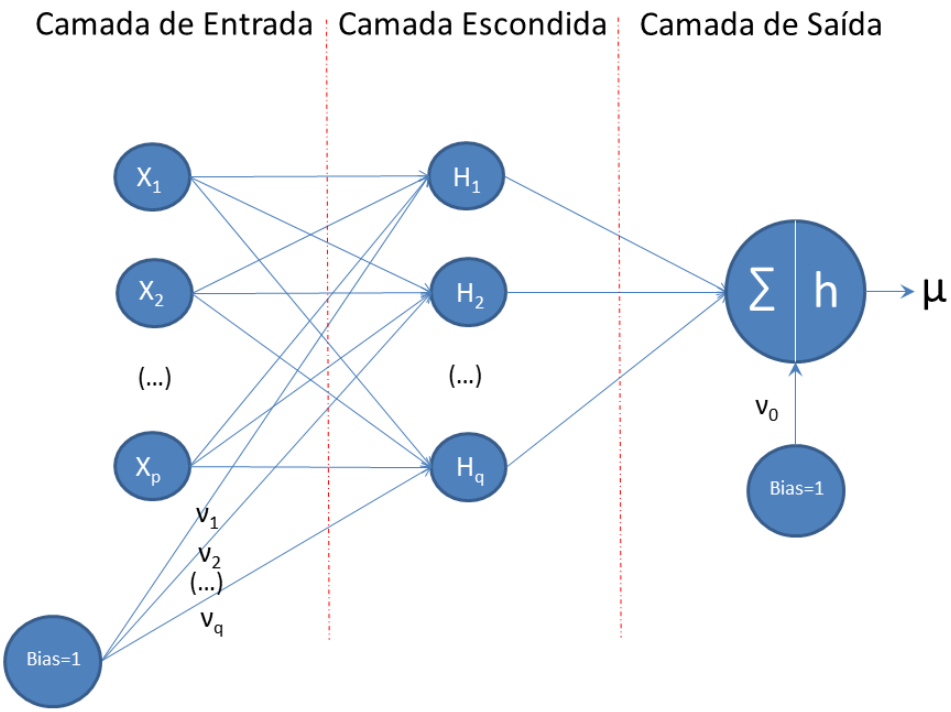

Figura 4.3: Perceptrão Multicamada.

$$F(x_1, ..., x_p) = \nu_0 + \sum_{k=1}^{q} \alpha_k \varphi(\sum_{j=1}^{p} \omega_{jk} x_j + \nu_k) \tag{4.5}$$

pode ser vista como uma aproximação da função $f(.)$ em que

$$|F(x_1, ..., x_p) - f(x_1, ..., x_p)| < \epsilon, \tag{4.6}$$

para todo $x_1, x_2, ..., x_p \in I_p$.

Considera-se para o efeito, um MLP com uma camada escondida com q neurónios escondidos, p neurónios na camada de entrada e um na camada de saída, em que os pesos sinápticos de entrada de cada neurónio escondido são representados por $\boldsymbol{\omega} = (\omega_{1k},...,\omega_{pk})$ e os pesos correspondentes ao *bias* por $\boldsymbol{\nu} = (\nu_0,...,\nu_q)$. Os pesos sinápticos que interligam a camada escondida da camada de saída são representados por $\boldsymbol{\alpha} = (\alpha_1,...,\alpha_q)$.

A partir da eq. 4.6 pode-se concluir que um MLP com uma camada escondida é suficiente para a aproximação ϵ de uma função contínua, dado um conjunto de treino, representado pelo conjunto de covariáveis x_1,...,x_p e uma função objetivo $f(x_1, ..., x_p)$ (Haykin, 1998). O problema, citando o mesmo autor, é que o teorema não define como determinar o número de neurónios numa camada escondida, de modo a otimizar o tempo de aprendizagem, a facilidade de implementação, ou (o mais importante) a capacidade de generalização. No entanto, a característica de "aproximador universal" de um MLP, em que consegue captar todas as interações entre as covariáveis, além de implicitamente detetar relações não lineares complexas entre elas e a variável resposta (Tu, 1996), vem reforçar a sua importância como estimador que tanto pode ser utilizado individualmente, ou como uma componente integrada noutras arquiteturas, tal como a GANN, como iremos analisar no capítulo 5.

4.5 Estimação dos pesos sinápticos

O método mais utilizado para a estimação dos pesos sinápticos de uma rede neuronal é o da retropropagação do erro, podendo ser definido como um algoritmo de minimização do risco empírico recorrendo ao método do gradiente (Marques, 2000). O risco empírico, por sua vez, corresponde à função de custo, através da qual se pode medir o desempenho da rede no conjunto de treino. Através da sua minimização, é possível estimar os pesos de forma a adaptar a rede ao conjunto de dados.

Efetivamente, a partir do custo médio pode-se obter o risco empírico (Vapnik, 1995):

$$\mathcal{R} = \frac{1}{N} \sum_{x \in \mathbf{X}} c(y(x), \mu(x)), \tag{4.7}$$

em que $c(y(x), \mu(x))$ representa o custo de aproximação dos valores observados y às estimativas de μ. Um caso particular é o critério de mínimos quadrados:

$$\mathcal{R} = \frac{1}{N} \sum_{x \in \mathbf{X}} e(x)^2, \tag{4.8}$$

em que

$$e(x) = y(x) - \mu(x), \tag{4.9}$$

representa o erro entre a saída pretendida e a saída da rede neuronal (Marques, 2000).

A minimização de 4.7 é feita com recurso a métodos numéricos, sendo o mais usual o algoritmo do gradiente descendente (regra delta). No entanto, para a sua aplicação é necessário que a função de ativação dos neurónios seja diferenciável, pelo que também é usual a escolha de uma função sigmoidal para o efeito.

Segundo a regra delta, o ajuste de um determinado peso sináptico, numa determinada iteração (época) t, é efetuado com base num algoritmo iterativo definido pelas seguintes equações:

$$\omega(t) = \omega(t-1) + \Delta\omega(t), \tag{4.10}$$

e

$$\Delta\omega(t) = -\alpha \left[\frac{\partial \mathcal{R}}{\partial \omega}\right]_{\omega=\omega(t-1)} = -\alpha g(t), \tag{4.11}$$

em que $g(t)$ representa o gradiente na época t de $\mathcal{R}$, $\omega(t-1)$ representa o valor de ω na época $t-1$ e α representa a taxa de aprendizagem (*learning rate*).

Se considerarmos o MLP da fig. 4.4, cada sinapse é numerada tendo em conta o neurónio emissor (j) e o neurónio recetor (k), sendo designada por ω_{jk}. Os sinais que entram no neurónio j são somados resultando s_j. A saída deste neurónio é afetada por uma função de transferência, de forma a que $\mu_j = f(s_j)$. O sinal é posteriormente emitido para o neurónio seguinte (k) afetado pelo peso ω_{jk}, de forma a que este último recebe o valor $\omega_{jk}\mu_j$ proveniente do neurónio emissor (fig. 4.5).

Desta forma, cada padrão $\boldsymbol{X}$ à entrada da rede é propagado para a saída da rede, sendo afetado pelo vetor de pesos sinápticos $\boldsymbol{\omega}$ e pela função de ativação de cada neurónio f, obtendo-se um vetor de saída $\boldsymbol{\mu}$.

Para que os pesos sinápticos possam ser modificados de forma a que o risco empírico seja minimizado (eq. 4.11), será necessário proceder à atualização iterativa dos primeiros, podendo-se utilizar, para tal, alguns métodos de otimização, nomeadamente o algoritmo de retropropagação que passamos a descrever.

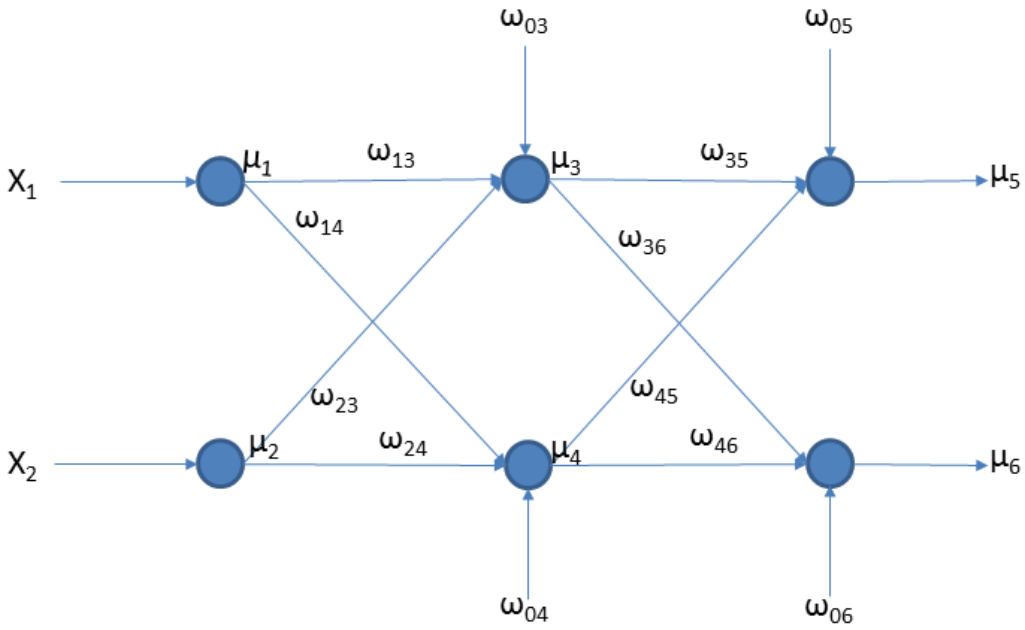

Figura 4.4: Perceptrão Multicamada.

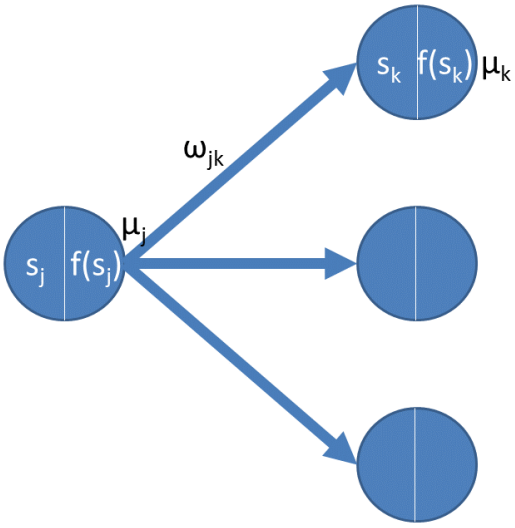

Figura 4.5: Propagação do sinal do neurónio emissor para o neurónio recetor (*forward*).

4.5.1 Algoritmo de retropropagação do erro

Até à década de 80, um dos problemas, que impossibilitava a utilização do MLP era o facto de, até então, não existir uma forma de estimar os parâmetros da rede em situações em que havia uma ou mais camadas escondidas. Porém, este obstáculo foi sendo resolvido por vários autores, culminando com a introdução do método de retropropagação do erro (algoritmo de *backpropagation*) cujo objetivo é o de calcular o gradiente da função do erro relativo aos pesos sinápticos, e atualizá-los na direção oposta do primeiro (Marques, 2000).

Considerando a eq. 4.11, o gradiente correspondente a cada sinapse da rede que liga a camada intermédia (escondida) à camada de saída é dado por:

$$g_{jk} = \frac{1}{N} \sum_{x \in \mathbf{X}} \Big(\frac{\partial c}{\partial \boldsymbol{\mu}_{out}} \Big)^T \frac{\partial \boldsymbol{\mu}_{out}}{\partial \omega_{jk}}, \tag{4.12}$$

em que $\boldsymbol{\mu}_{out}$ é o vetor que contém as saídas das unidades da última camada da rede (*e.g.* no caso da rede da fig. 4.4, $\boldsymbol{\mu}_{out} = [\mu_5 \quad \mu_6]^T$).

Uma vez que $\boldsymbol{\mu}_{out}$ depende de ω_{jk} através da saída k, aplicando a derivada da função composta (e consultando a fig. 4.5) obtém-se a seguinte expressão:

$$\frac{\partial \boldsymbol{\mu}_{out}}{\partial \omega_{jk}} = \frac{\partial \boldsymbol{\mu}_{out}}{\partial \mu_k} \frac{\partial \mu_k}{\partial \omega_{jk}} = \frac{\partial \boldsymbol{\mu}_{out}}{\partial \mu_k} \frac{\partial \mu_k}{\partial f_k} \frac{\partial f_k}{\partial s_k} \frac{\partial s_k}{\partial \omega_{jk}}. \tag{4.13}$$

Considerando que $\mu_k = f_k(s_k)$ e $\frac{\partial s_k}{\partial \omega_{jk}} = \mu_j$, a expressão anterior toma a seguinte forma:

$$\frac{\partial \boldsymbol{\mu}_{out}}{\partial \omega_{jk}} = \frac{\partial \boldsymbol{\mu}_{out}}{\partial \mu_k} \frac{\partial f_k}{\partial s_k} \mu_j. \tag{4.14}$$

Assim, através da equação anterior, o gradiente g_{jk} (eq. 4.12) a aplicar aos pesos sinápticos entre a camada intermédia e a de saída é dado pela seguinte expressão:

$$g_{jk} = \frac{1}{N} \sum_{x \in \mathbf{X}} \Big(\frac{\partial c}{\partial \boldsymbol{\mu}_{out}} \Big)^T \frac{\partial \boldsymbol{\mu}_{out}}{\partial \mu_k} \frac{\partial f_k}{\partial s_k} \mu_j. \tag{4.15}$$

Considerando uma grandeza auxiliar ϵ_k, designada por "erro visto no neurónio k" (Marques, 2000), para cada unidade da camada de saída, esta é

traduzida pela seguinte expressão:

$$\epsilon_k = \frac{\partial f_k}{\partial s_k}\Big(\frac{\partial c}{\partial \boldsymbol{\mu}_{out}}\Big)^T \frac{\partial \boldsymbol{\mu}_{out}}{\partial \mu_k} = \frac{\partial f_k}{\partial s_k}\frac{\partial c}{\partial \mu_k}. \tag{4.16}$$

Nas camadas intermédias, o valor de ϵ_j correspondente a cada um dos neurónios escondidos poderá ser determinado pela aplicação de um grafo que se obtém invertendo o sentido dos ramos do perceptrão, em que os pontos de divergência se transformam em pontos de soma (Marques, 2000), tal como se pode observar pela fig. 4.6.

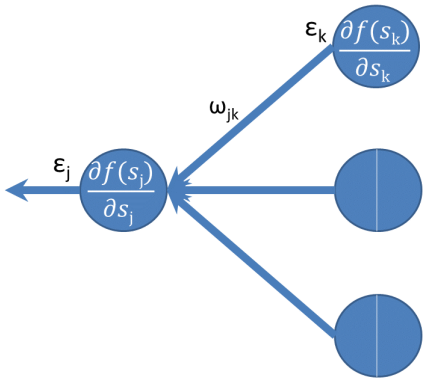

Figura 4.6: Cálculo de ϵ_j (*backward*).

Verifica-se assim, para as unidades escondidas:

$$\epsilon_j = \frac{\partial f_j}{\partial s_j}\sum_k \omega_{jk}\epsilon_k. \tag{4.17}$$

Em termos gerais, na fase de atualização dos pesos sinápticos, o perceptrão pode ser visto sob a perspetiva de uma rede, só que com o fluxo do sinal invertido (ver fig. 4.7). Nesta rede invertida, as variáveis ϵ correspondem às saídas dos neurónios (invertidos). À entrada da rede invertida é induzido o custo $\frac{\partial c}{\partial \mu_k}$, que será propagado para a sua saída (que corresponde à camada de entrada do perceptrão). A rede invertida é denominada por rede de retropropagação do erro (Marques, 2000).

No algoritmo de retropropagação (ver algoritmo 3), os pesos sinápticos são atualizados depois de processados todos os padrões da amostra de treino

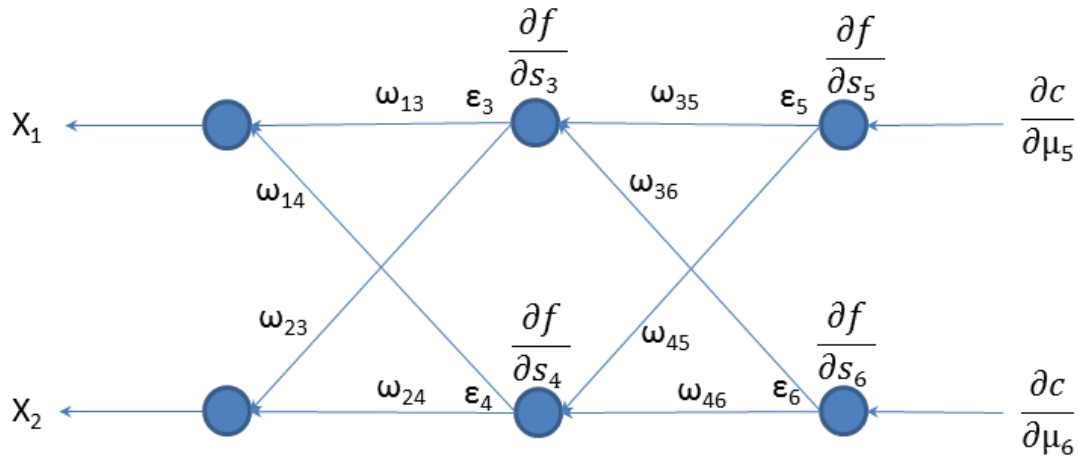

Figura 4.7: Rede equivalente de retropropagação do erro.

(treino em lotes/*batch training*).

Uma alternativa a este procedimento passa pela atualização dos pesos da rede sempre que é apresentado um novo padrão (treino em tempo real/*on-line training*) (Marques, 2000), de forma a que o gradiente da eq. 4.15 tendo em conta a eq. 4.16 corresponderá à seguinte expressão:

$$g_{jk} = \frac{1}{N} \frac{\partial f_k}{\partial s_k} \frac{\partial c}{\partial \mu_k} \mu_j. \tag{4.18}$$

4.5.2 Técnicas de aceleração da convergência

O algoritmo de retropropagação não garante, por si só, a convergência para um mínimo global da função do erro (Riedmiller e Braun, 1993). Além disso, após as primeiras iterações, verifica-se um ritmo de convergência lento, o que justifica a introdução de modificações para a mitigação destes problemas.

Neste sentido, pode-se introduzir um termo (momento - β), de forma a que:

$$\Delta\omega_{jk}(t) = -\alpha g_{jk}(t) + \beta\Delta\omega_{jk}(t-1), \tag{4.19}$$

que reforça variações no mesmo sentido que a variação do peso anterior

início
  Inicializar vetor de pesos sinápticos iniciais $\boldsymbol{\omega}(0)$ com valores aleatórios pequenos (*e.g.* $\omega_{jk}(0) \in]-1, 1[$);
  $t = 1$;
  para cada *individuo do conjunto de treino na camada de entrada* **faça**
    $s_k = \sum_j \omega_{jk} x_j + \omega_{0k}$;
    $\mu_k = f(s_k)$;
    repita
      Seja j o índice dos neurónios da camada anterior;
      $s_k = \sum_j \omega_{jk} \mu_j + \omega_{0k}$;
      $\mu_k = f(s_k)$;
    até *sinal seja propagado para a(s) saída(s) do perceptrão*;
    Obter os valores da resposta (μ_k);
    Calcular $\frac{\partial c}{\partial \mu_k}$;
    Calcular na camada de saída: $\epsilon_j = \frac{\partial f_j}{\partial s_j} \frac{\partial c}{\partial \mu_j}$
    repita
      Calcular $\epsilon_j = \frac{\partial (f_j)}{\partial (s_j)} \sum_k \omega_{jk} \epsilon_k$ em que k representa o índice dos neurónios da camada seguinte (no sentido da entrada para a saída da rede);
    até *sinal seja propagado para a(s) entrada(s) do perceptrão*;
  fim
  $\Delta\omega_{jk} = -\frac{\alpha}{N} \sum_{x \in \mathbf{X}} x_j \epsilon_k$ para a primeira camada.;
  $\Delta\omega_{jk} = -\frac{\alpha}{N} \sum_{x \in \mathbf{X}} \mu_j \epsilon_k$ para as restantes camadas.;
  $\omega_{jk}(t) = \omega_{jk}(t-1) + \Delta\omega_{jk}$;
  se *não se verifica condição de paragem* **então**
    $t = t + 1$;
    Voltar ao ponto 4;
  fim
fim

Algoritmo 3: Algoritmo de retropropagação.

(Marques, 2000), de forma a que tanto as alterações bruscas de direção, assim como a amplitude das oscilações tendem a ser reduzidas. Para garantir a convergência, o valor do momento deverá ser inferior a 1, sendo o valor típico de 0.9.

No entanto, o valor ótimo de β depende do problema tal como o valor ótimo de α, pelo que foram propostas outras técnicas de aceleração que se baseiam na utilização de ganhos diferentes e adaptativos para cada peso sináptico. Destas técnicas destacam-se o Resilient Backpropagation (RPROP) (Riedmiller e Braun, 1993) e o algoritmo *Levenberg-Marquadt* (Hagan et al., 1996), como sendo dos melhores algoritmos de aprendizagem e que passaremos a apresentar.

Algoritmo RPROP

Neste algoritmo, a atualização de cada peso sináptico é obtido diretamente por uma variável Δ_{ij} cujo valor, durante o processo de aprendizagem, varia consoante um ganho γ^+ ou γ^- que, por sua vez, está dependente da mudança de sinal do gradiente nas duas últimas iterações de forma que:

$$\Delta_{jk}(t) = \begin{cases} \gamma^+\Delta_{jk}(t-1),\ se\, g_{jk}(t)g_{jk}(t-1) > 0, \\ \gamma^-\Delta_{jk}(t-1),\ se\, g_{jk}(t)g_{jk}(t-1) < 0, \\ \Delta_{jk}(t-1),\ se\, g_{jk}(t)g_{jk}(t-1) = 0, \end{cases} \tag{4.20}$$

em que $0 < \gamma^- < 1 < \gamma^+$.

Assim, uma mudança de sinal do gradiente correspondente a um determinado peso sináptico ω_{jk}, indica que o valor da última atualização deste peso foi demasiado grande tendo-se "saltado" um mínimo local. Em consequência, diminui-se Δ_{ij} tendo em conta um ganho γ^-. Se o sinal do gradiente se mantém de uma iteração para outra, Δ_{ij} aumenta de forma a acelerar a sua convergência.

Depois de obtido Δ_{ij}, pode-se proceder à atualização dos pesos tendo em atenção o seguinte:

$$\Delta\omega_{jk}(t) = \begin{cases} -\Delta_{jk}(t),\ se\, g_{jk}(t) > 0, \\ \Delta_{jk}(t),\ se\, g_{jk}(t) < 0, \\ 0,\ se\, g_{jk} = 0. \end{cases} \tag{4.21}$$

A exceção à regra acontece quando $g_{jk}(t)g_{jk}(t-1) < 0$. Neste caso $\Delta\omega_{jk}(t) = -\Delta\omega_{jk}(t-1)$.

Assim sendo, o algoritmo RPROP pode ser implementado através dos procedimentos descritos no algoritmo 4:

```
1  início
2      para todos os pesos sináticos (incluindo bias) faça
3          se g_jk(t)g_jk(t−1) > 0 então
4              Δ_jk(t)=mínimo(Δ_jk(t−1)*γ⁺,Δ_max);
5              Δω_jk(t) = −sinal(g_jk(t)) * Δ_jk(t);
6              ω_jk(t+1) = ω_jk(t) + Δω_jk(t);
7          senão
8              se g_jk(t)g_jk(t−1) < 0 então
9                  Δ_jk(t)=máximo(Δ_jk(t−1)*γ⁻,Δ_min);
10                 ω_jk(t+1) = ω_jk(t) − Δω_jk(t−1);
11                 g_jk(t) = 0;
12             senão
13                 Δω_jk(t) = −sinal(g_jk(t)) * Δ_jk(t);
14                 ω_jk(t+1) = ω_jk(t) + Δω_jk(t);
15             fim
16         fim
17     fim
18 fim
```

Algoritmo 4: Algoritmo *RPROP*.

Os valores iniciais de Δ_{jk} (para $t = 0$) deverão ser escolhidos tendo como referência um valor proporcional aos pesos sinápticos iniciais (*e.g.* $\Delta_{jk}(0) = 0.1$). De igual forma deverão ser definidos os valores para Δ_{max} e Δ_{min} (*e.g.* $\Delta_{max} = 1.0$ e $\Delta_{min} = 10^{-6}$).

Algoritmo Levenberg-Marquadt

O algoritmo de retropropagação também pode ser utilizado para obter as segundas derivadas do erro através da matriz Hessiana, permitindo a utilização de algoritmos de otimização de segunda ordem, nomeadamente o algoritmo de Levenberg-Marquadt.

Considerando a função de erro $Er(\boldsymbol{\omega})$ e tendo em conta o vetor dos pesos sinápticos $\boldsymbol{\omega}$, a série de Taylor da função de erro, após a atualização dos pesos sinápticos, é dada por:

$$(Er(\boldsymbol{\omega} + \Delta\boldsymbol{\omega})) \simeq Er(\boldsymbol{\omega}) + \nabla Er(\boldsymbol{\omega})^t \Delta\boldsymbol{\omega} + \frac{1}{2}\Delta\boldsymbol{\omega}^t \nabla^2 Er(\boldsymbol{\omega_0})\Delta\boldsymbol{\omega}. \quad (4.22)$$

A minimização da eq. 4.22 permitirá obter os pontos de estacionariedade do gradiente de Er em $\Delta\boldsymbol{\omega}$, de forma a que:

$$\nabla(Er(\boldsymbol{\omega} + \Delta\boldsymbol{\omega})) \simeq \nabla Er(\boldsymbol{\omega}) + \nabla^2 Er(\boldsymbol{\omega})\Delta\boldsymbol{\omega} = 0. \quad (4.23)$$

A expressão anterior pode ser reescrita da seguinte forma:

$$\Delta\boldsymbol{\omega} = -\nabla^2 Er(\boldsymbol{\omega})^{-1} \nabla Er(\boldsymbol{\omega}). \quad (4.24)$$

Tendo em conta que $\nabla^2 Er(\boldsymbol{\omega})$ corresponde à matriz Hessiana da função E_r (Dias et al., 2004), esta pode não ser invertível o que impossibilita o cálculo da sua inversa. Para ultrapassar este problema pode-se utilizar uma modificação da matriz Hessiana de forma a que:

$$\mathbf{H}^* = \mathbf{H} + \lambda\mathbf{I}, \quad (4.25)$$

em que $\mathbf{I}$ é a matriz identidade e o valor de λ é tal que $\mathbf{H}^*$ seja positiva e invertível. Assim, a eq. 4.23 pode ser reescrita da seguinte forma:

$$\Delta\boldsymbol{\omega} = [\mathbf{H}^* + \lambda\mathbf{I}]^{-1}\mathbf{J}^t Er(\boldsymbol{\omega}) = [\mathbf{J}^t\mathbf{J} + \lambda\mathbf{I}]^{-1}\mathbf{J}^t Er(\boldsymbol{\omega}). \quad (4.26)$$

A matriz $\mathbf{J}$ corresponde à matriz Jacobiana de dimensão $n \times z$ (ver eq. 4.27), em que n representa o número de indivíduos e z o número total de pesos sinápticos.

$$\mathbf{J} = \begin{vmatrix} \frac{\partial Er(\mathbf{x}_1,\boldsymbol{\omega})}{\partial \omega_1} & \frac{\partial Er(\mathbf{x}_1,\boldsymbol{\omega})}{\partial \omega_2} & \cdots & \frac{\partial Er(\mathbf{x}_1,\boldsymbol{\omega})}{\partial \omega_z} \\ \frac{\partial Er(\mathbf{x}_2,\boldsymbol{\omega})}{\partial \omega_1} & \frac{\partial Er(\mathbf{x}_2,\boldsymbol{\omega})}{\partial \omega_2} & \cdots & \frac{\partial Er(\mathbf{x}_2,\boldsymbol{\omega})}{\partial \omega_z} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial Er(\mathbf{x}_n,\boldsymbol{\omega})}{\partial \omega_1} & \frac{\partial Er(\mathbf{x}_n,\boldsymbol{\omega})}{\partial \omega_2} & \cdots & \frac{\partial Er(\mathbf{x}_n,\boldsymbol{\omega})}{\partial \omega_z} \end{vmatrix}. \tag{4.27}$$

Considerando a função de erro $Er(\boldsymbol{\omega})$ correspondente ao Erro Quadrático Médio (*Mean Squared Error - MSE*):

$$Er(\boldsymbol{\omega}) = MSE, \tag{4.28}$$

o algoritmo Levenberg-Marquadt (LM) pode ser implementado recorrendo aos procedimentos do algoritmo 5 (Chen et al., 2003):

início
  Inicializar vetor de pesos sinápticos iniciais $\boldsymbol{\omega}^0$ com valores aleatórios pequenos (*e.g.* $\omega_{jk}^0 \in]-1, 1[$);
  Inicializar λ, β (tipicamente $\lambda = 0.01$, $\beta = 0.1$);
  Estimar MSE;
  Calcular a matriz Jacobiana $\mathbf{J}$;
  Calcular $\Delta\boldsymbol{\omega} = [\mathbf{J}^t\mathbf{J} + \lambda\mathbf{I}]^{-1}\mathbf{J}^t Er(\boldsymbol{\omega})$;
  Estimar MSE_{new} como valor provisório, resultante da atualização provisória dos pesos sináticos $\boldsymbol{\omega}_{new} = \boldsymbol{\omega} + \Delta\boldsymbol{\omega}$;
  se $MSE_{new} < MSE$ **então**
    $\boldsymbol{\omega} = \boldsymbol{\omega}_{new}$;
    $\lambda = \beta\lambda$;
    Voltar ao ponto 4;
  senão
    $\lambda = \frac{\lambda}{\beta}$;
    Voltar ao ponto 6;
  fim
fim

Algoritmo 5: Algoritmo *Levenberg-Marquadt*.

4.6 Estimação intervalar

A estimação intervalar nas redes neuronais de alimentação direta pode ser obtida recorrendo a algoritmos que também se utilizam nos modelos de re-

gressão, nomeadamente, os baseados no método delta ou no método de *bootstrap* (Efron e Tibshirani, 1994; Dybowski e Roberts, 2001).

4.6.1 Método delta

Considerando uma rede neuronal em que $\boldsymbol{\mu}_{\hat{\boldsymbol{\omega}}}$ representa o vetor médio dos pesos sinápticos $\hat{\boldsymbol{\omega}}$ que adaptam a rede neuronal à amostra de dados, uma aproximação de primeira ordem à série de Taylor em torno da estimativa da resposta $\hat{\mu}_y(x, \hat{\boldsymbol{\omega}})$ pode ser obtida pela seguinte expressão (Dybowski e Roberts, 2001):

$$\hat{\mu}_y(x, \hat{\boldsymbol{\omega}}) \approx \hat{\mu}_y(x, \boldsymbol{\mu}_{\hat{\boldsymbol{\omega}}}) + g(\mathbf{x})(\hat{\boldsymbol{\omega}} - \boldsymbol{\mu}_{\hat{\boldsymbol{\omega}}}), \tag{4.29}$$

em que $g(\mathbf{x})$ representa o vetor gradiente, cujo i-ésimo elemento corresponde à derivada parcial $\frac{\partial \hat{\mu}_y(x,\hat{\boldsymbol{\omega}})}{\partial \hat{\omega}_i}$ obtida para $\hat{\boldsymbol{\omega}} = \boldsymbol{\mu}_{\hat{\boldsymbol{\omega}}}$. Assim, a partir da expressão 4.30 poder-se-á obter a variância (Efron e Tibshirani, 1994):

$$\widehat{var}(\hat{\mu}_y(x, \boldsymbol{\mu}_{\hat{\boldsymbol{\omega}}})) = \mathbf{g}^T(\mathbf{x})\Sigma\mathbf{g}(\mathbf{x}), \tag{4.30}$$

em que Σ representa a matriz covariância relativamente a $\hat{\omega}$. Considerando que esta matriz possui uma relação direta com a Hessiana (ver eq. 4.25), e que a função de erro é a definida em 4.28, então σ^2_ϵ é estimada pela seguinte expressão (Tibshirani, 1996):

$$\sigma^2_\epsilon = \frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{\mu}_y(\mathbf{x}_i, \hat{\boldsymbol{\omega}})]^2. \tag{4.31}$$

A eq. 4.30 pode, então, ser escrita da seguinte forma:

$$\widehat{var}(\hat{\mu}_y(x, \boldsymbol{\mu}_{\hat{\boldsymbol{\omega}}})) = \sigma^2_\epsilon \mathbf{g}^T(\mathbf{x})\mathbf{H}^{-1}\mathbf{g}(\mathbf{x}). \tag{4.32}$$

Assim, assumindo uma distribuição Gaussiana do ruído, os intervalos de confiança podem ser determinados pela seguinte equação:

$$\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}}) \pm z_{0.025}\sqrt{\sigma^2_\epsilon \mathbf{g}^T(\mathbf{x})\mathbf{H}^{-1}\mathbf{g}(\mathbf{x})}. \tag{4.33}$$

No caso de se utilizar a estratégia de regularização (ver secção 4.8.2) para melhorar a capacidade de generalização da rede neuronal, tendo em atenção o coeficiente de regularização α (ver eq. 4.43), a expressão anterior deverá ser escrita da seguinte forma (Tibshirani, 1996):

$$\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}}) \pm z_{0.025}\sqrt{\mathbf{g}^T(\mathbf{x})\Big(\frac{\mathbf{H}}{\sigma_\epsilon^2} - \alpha\Big)^{-1}\mathbf{g}(\mathbf{x})}. \tag{4.34}$$

4.6.2 Método *bootstrap*

Considerando uma determinada amostra $\mathbf{S}$ retirada de uma população cuja distribuição depende de um parâmetro θ, o erro padrão pode ser obtido aplicando o método de *bootstrap* conforme descrito na secção 3.2.3, *i.e.* considerando várias amostras $\mathbf{S}^{*b}$ com reposição, através da seguinte expressão (Efron e Tibshirani, 1994):

$$\widehat{SE}_{boot}(\hat{\theta}(\mathbf{S})) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\Big[\hat{\theta}(\mathbf{S}^{*b}) - \hat{\theta}_{boot}\Big]^2}, \tag{4.35}$$

em que $\hat{\theta_{boot}} = \sum_{b=1}^{B}\frac{\hat{\theta}(\mathbf{S}^{*b})}{B}$ representa a estimativa de θ. Os valores típicos de B encontram-se no intervalo $[25, 200]$ (Dybowski e Roberts, 2001).

Num contexto de regressão, foram propostos dois algoritmos de *bootstrap* por Efron e Tibshirani (1994): *pairs sampling* e *residual sampling*. No primeiro, a amostragem é realizada diretamente a partir da amostra original $\mathbf{S}$ (ver algoritmo 6). O segundo (algoritmo 7) caracteriza-se pela amostragem com reposição dos resíduos associados à função $\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}})$.

```
1 início
2   Seja {(x_1, y_1), ..., (x_N, y_N)} a verdadeira amostra S;
3   b = 0;
4   repita
5     b = b + 1;
6     Formar um subconjunto S^{*b} = {(x_1, y_1)^{*b}, ..., (x_N, y_N)^{*b}},
        retirando N observações com reposição a partir da amostra S;
7     Treinar a rede neuronal com base na amostra S^{*b} e obter a
        função μ̂_y(x, ω̂^{*b});
8   até b = B;
9 fim
```

Algoritmo 6: *Bootstrap pairs sampling.*

início
  Seja $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ a verdadeira amostra $\mathbf{S}$;
  $b = 0$;
  Treinar a rede neuronal com base na amostra $\mathbf{S}$ e obter a função $\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}})$;
  Seja $\mathbf{R}$ o conjunto de resíduos $\{r_1, ..., r_N\}$ em que $r_i = y_i - \hat{\mu}_y(\mathbf{x}_i, \hat{\boldsymbol{\omega}})$;
  repita
    $b = b + 1$;
    Formar um subconjunto $\mathbf{R}^{*b} = \{\mathbf{r}_1{}^{*b}, ..., \mathbf{r}_N{}^{*b}\}$, retirando N observações com reposição a partir da amostra $\mathbf{R}$;
    Treinar a rede neuronal com base na amostra $\mathbf{S}^{*b} = \{(\mathbf{x}_1, \hat{\mu}_y(\mathbf{x}_1, \hat{\boldsymbol{\omega}}) + r_1^{*b}), ..., (\mathbf{x}_N, \hat{\mu}_y(\mathbf{x}_N, \hat{\boldsymbol{\omega}}) + r_N^{*b})\}$ e obter a função $\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}}^{*b})$;
  até $b = B$;
fim

Algoritmo 7: *Bootstrap residual sampling.*

Após o treino do modelo, utilizando qualquer uma das anteriores abordagens, a estimativa *bootstrap* da função $\hat{\mu}(\mathbf{x})$ é dada pela função "média" de $\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}}^{*1})$, ..., $\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}}^{*B})$, resultante da seguinte expressão:

$$\hat{\mu}_{y_{boot}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}}^{*b}). \tag{4.36}$$

A partir da eq. 4.35 deduz-se a expressão (Dybowski e Roberts, 2001):

$$\widehat{SE}_{boot}(\hat{\mu}_y(\mathbf{x}, .)) = \sqrt{\frac{1}{B-1} \sum_{b=1}^{B} \left[\hat{\mu}_y(\mathbf{x}, \hat{\boldsymbol{\omega}}^{*b}) - \hat{\mu}_{y_{boot}}(\mathbf{x})\right]^2}, \tag{4.37}$$

pelo que a estimação intervalar pode ser realizada pela aplicação da seguinte expressão (Heskes, 1997):

$$\hat{\mu}_{y_{mean}}(\mathbf{x}) \pm t_{0.025[B]} \widehat{SE}_{boot}(\hat{\mu}_y(\mathbf{x}, .)). \tag{4.38}$$

4.7 Normalização dos dados

Para o treino da rede é fundamental que as escalas das variáveis explicativas não sejam muito diferentes. Efetivamente, se as variáveis que apresentam

grandes amplitudes são combinadas com as de baixa amplitude, as primeiras podem "mascarar" o efeito das últimas, de forma a tornar os pesos sinápticos associados mais relevantes, face aos restantes. É também igualmente importante prevenir o crescimento exagerado dos pesos sinápticos, pelo que se deve não só escolher valores pequenos para os pesos iniciais, como proceder ao ajustamento das escalas para valores mais baixos.

Um procedimento normalmente utilizado na fase de pré-processamento, designado por normalização, permite a equalização das escalas de entrada transformando, simultaneamente, a sua amplitude por forma a que a rede funcione o mais dentro possível da zona de ativação dos neurónios. Para tal, existem vários métodos disponíveis, em que os mais utilizados são:

- Transformação linear - Os valores da j-ésima covariável X_j são linearmente transformados em $t_j(X_j)$ através da equação:

$$t_j(X_j) = l_m + \frac{(X_j - min(X_j))(l_M - l_m)}{max(X_j) - min(X_j)}, \tag{4.39}$$

  em que l_M e l_m representam respetivamente os limites máximos e mínimos da escala de uma variável genérica X_j.

- Padronização - A transformação dos valores da j-ésima covariável X_j é baseada na média e no desvio padrão da amostra de dados:

$$t_j(X_j) = \frac{X_j - \overline{X_j}}{S_j}, \tag{4.40}$$

  em que $\overline{X_j}$ representa a média amostral da variável X_j e S_j o desvio padrão amostral correspondente.

4.8 Generalização

O objetivo do treino de uma rede neuronal, tal como o de um modelo de regressão, não é encontrar uma representação do problema limitada aos dados de treino, mas sim encontrar um modelo generalista do fenómeno em estudo de onde se obtiveram os dados (Bishop, 1995). Esta característica está diretamente relacionada com a capacidade de interpolação não linear dos dados de entrada (Haykin, 1998), que se traduz na escolha de um modelo que possua uma boa capacidade de generalização e que produza boas estimativas

para novos valores introduzidos na rede. Esta interpolação depende essencialmente de dois fatores: da dimensão da amostra e do grau de flexibilidade da rede que, no caso deste último for excessivo, resultará na memorização da amostra de treino, no caso de ser deficitário, o modelo torna-se rígido com um viés grande. Em ambos os casos o modelo apresenta uma fraca interpolação dos dados.

A flexibilidade do modelo encontra-se diretamente relacionada com a sua complexidade, resultante do número de parâmetros a estimar. Se este número for reduzido, o modelo será pouco flexível para se adaptar aos dados da amostra de treino, conduzindo a uma fraca generalização, se for em número excessivo, o modelo resultante conduz a um sobre ajustamento dos dados que resulta igualmente num fraco desempenho.

Este problema pode ser melhor explicado recorrendo ao compromisso viés-variância (*bias-variance trade-off*), que influencia tanto o desempenho de uma rede neuronal como o de um modelo de regressão.

4.8.1 Compromisso viés-variância

Esta inter-relação define um compromisso que se procura otimizar tanto quanto possível, para resultar numa melhor precisão das estimativas da variável resposta. Assim, considerando a função $f(x)$ como sendo a estimada através do modelo $Y = f(x) + \epsilon$, para cada ponto x pertencente ao domínio de f, o MSE respetivo pode ser definido pela seguinte expressão (Bishop, 1995):

$$MSE(\hat{f}(x)) = E[(\hat{f}(x) - f(x))^2]. \tag{4.41}$$

A expressão anterior é constituída por dois termos, em que um depende do quadrado do viés e outro depende do quadrado da variância:

$$MSE(\hat{f}(x)) = (E[\hat{f}(x)] - f(x))^2 + var(\hat{f}(x)). \tag{4.42}$$

Esta expressão traduz a existência do compromisso entre o viés e a variabilidade, atrás referida. Um estimador com grande suavização resulta num decréscimo da variabilidade, mas num aumento do viés (fig. 4.8), com maior flexibilidade resulta no contrário (fig. 4.9). Na fig. 4.10 encontra-se um

exemplo de uma relação ótima entre o viés e a variância.

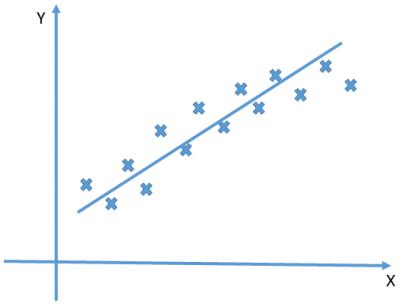

Figura 4.8: Ilustração das estimativas de um modelo sub ajustado.

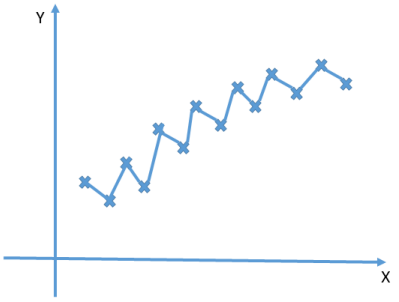

Figura 4.9: Ilustração das estimativas de um modelo sobre ajustado.

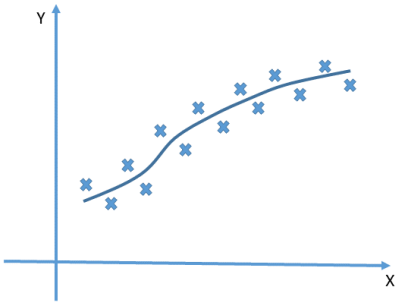

Figura 4.10: Ilustração das estimativas de um modelo devidamente ajustado.

A complexidade do modelo é definida pelo número de parâmetros a estimar que, no caso de um MLP, depende essencialmente do número de camadas intermédias utilizadas e no número de neurónios por cada uma dessas camadas. No entanto, este tópico representa um problema em aberto, dado não existirem *guidelines* para determinar *a priori* o número ótimo de camadas e neurónios escondidos.

4.8.2 Estratégias para melhoria da generalização nas redes neuronais

Para encontrar o melhor compromisso viés-variância, podem-se adotar as seguintes estratégias:

- Procura exaustiva - não existindo um método direto para determinar o número ótimo de neurónios intermédios numa rede, a determinação deste terá de passar por métodos baseados em tentativa e erro.

- Paragem antecipada (*early stopping*) - a ideia subjacente é de que o mau desempenho causado por uma maior flexibilidade da rede, pode ser mitigado pela paragem antecipada do treino. Ao mesmo tempo que se procede ao treino, o desempenho da rede é medido numa amostra de validação. No início do treino, verifica-se que o desempenho da rede, tanto sobre a amostra de treino como na de validação, vai progressivamente melhorando até que, em determinado ponto onde a rede começa por sobre ajustar-se, o desempenho medido na amostra de validação começa a deteriorar-se. Esse ponto de inflexão corresponde ao ponto onde se deve parar o processo de treino (fig. 4.11).

- Regularização - a ideia por detrás desta abordagem também se baseia na redução da flexibilidade do modelo pela limitação do número de pesos sinápticos aos eficientes, prevenindo o crescimento dos pesos relevantes para além do ponto ótimo de generalização e reduzindo o crescimento de pesos irrelevantes (*weight decay*). A introdução de um termo penalizador que tem em atenção a soma de todos os pesos sinápticos da rede (eq. 4.43), "empurra" os pesos que possuem um valor marginal para próximo de zero (o que equivale à supressão da sinapse). Assim, a função de erro é dada pela seguinte expressão:

$$Er^* = Er + \frac{\alpha}{2}\sum_{j=1}^{m}\omega_j^2, \tag{4.43}$$

em que o designado por coeficiente de regularização α controla o grau de suavização do termo penalizador.

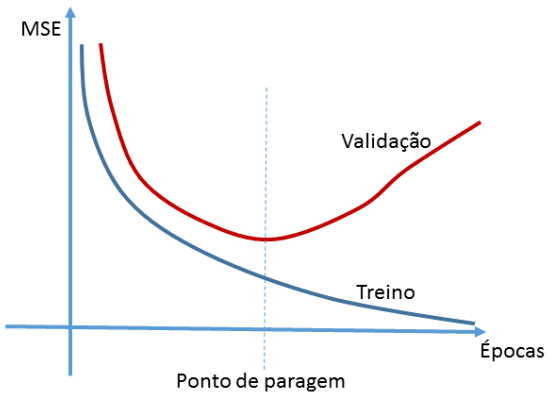

Figura 4.11: Ilustração da estratégia de paragem antecipada do treino.

Adicionalmente, existem ainda outros métodos que podem ser utilizados para a redução da complexidade do modelo, nomeadamente os algoritmos construtivos e de poda (e.g. algoritmo *brain damage*), que podem ser consultados com mais pormenor em Bishop (1995) e Samarasinghe (2006).

4.9 Discussão

O MLP apresenta-se, em termos teóricos, como um aproximador universal a qualquer função contínua. Adicionalmente, possui a capacidade de modelar todas as possíveis interações entre as variáveis explicativas (Tu, 1996) tornando-se particularmente interessante quando estas são em grande número. Neste contexto, e uma vez que num modelo aditivo as interações têm de ser identificadas e introduzidas pelo modelador, é expectável que se estas forem importantes para a obtenção da resposta, o MLP apresente melhores resultados que o GLM (Tu, 1996). Neste caso, o MLP pode ser utilizada para testar a existência dessas interações. Ao se utilizar um Perceptrão Simples (equivalente a um GLM) e um MLP, se o desempenho melhorar significativamente, poderá ser devido à existência de interações (Laurentiis e Ravdin, 1994).

No entanto, para a tomada de decisão na área clínica, é importante não só a capacidade preditiva do modelo como também a sua interpretabilidade.

Num MLP, os efeitos de uma determinada variável explicativa na resposta, depende de uma forma conjugada dos valores das restantes variáveis explicativas, o que a torna na prática numa "caixa negra". Neste contexto, a vantagem apresentada pelo MLP, no que toca à modelação das interações, é anulada pela dificuldade da sua análise.

Outro problema que afeta a utilização do MLP prende-se com a dificuldade na determinação do número de camadas, bem como do número de neurónios a introduzir em cada uma dessas camadas, para uma correta construção topológica do MLP. Quanto a este aspeto, não existem métodos diretos para determinar estas quantidades pelo que, certos autores propõem, numa primeira aproximação, a utilização de heurísticas (Walczak e Cerpa, 1999; Baum e Haussler, 1989). No entanto, e apesar de existir um grande número de defensores desta abordagem, existem outros que defendem que não existe uma forma eficiente de especificar a topologia da rede somente com base no número de entradas e saídas (Sharma, 2009), pelo que este problema permanece em aberto.

Por outro lado, apesar da capacidade do MLP na modelação de eventuais relações não lineares entre as variáveis explicativas e a variável resposta, vários estudos comparativos, entre esta arquitetura e o GLM, têm vindo a demonstrar a inexistência de uma vantagem clara no que respeita aos respetivos desempenhos (Tu, 1996).

As desvantagens, atrás referidas, ajudam a explicar a preferência pela utilização dos métodos de regressão no âmbito clínico, pelo que se justifica a introdução de uma arquitetura de rede neuronal que mimetize as características daqueles modelos, nomeadamente no que respeita à sua interpretabilidade.

Como iremos ver, o modelo objeto deste estudo, a GANN, alia a simplicidade da utilização do MLP à possibilidade da interpretação das relações entre as variáveis explicativas e a resposta, sendo esta uma característica muito importante no contexto clínico.

Capítulo 5

Rede Neuronal Aditiva Generalizada

5.1 Introdução

A Rede Neuronal Aditiva Generalizada (*Generalized Additive Neural Network* - GANN) surge como uma arquitetura inspirada a partir do Modelo Aditivo Generalizado (*Generalized Additive Model* - GAM). Com efeito, embora a GANN não seja um aproximador universal (tal como o Perceptrão Multicamada [1]) pode, em teoria, com a função de ligação correta, aproximar-se a qualquer modelo aditivo (de Waal et al., 2010).

Por outro lado, verifica-se em muitas situações um desempenho dececionante do MLP (Potts, 1999), tendo por base, nestes casos, uma deficiente caracterização da rede, produto da opacidade desta rede neuronal no que respeita à interpretabilidade do efeito de cada variável explicativa sobre a variável resposta. Essa característica de "caixa negra" dificulta o processo de escolha da topologia bem como dos parâmetros de rede mais adequados ao ajustamento de um modelo, como já foi anteriormente referido.

É neste âmbito que a GANN se pode apresentar como uma clara alternativa ao MLP, principalmente em problemas cujo objetivo não se esgota apenas na predição mas onde a interpretabilidade do modelo é uma característica muito importante.

Dado o paralelismo existente entre a GANN e o GAM, é natural que se consigam estabelecer semelhanças entre as componentes de ambos os mode-

[1] *Multi Layer Perceptron* - MLP

los. Efetivamente, verifica-se uma direta tradução das componentes aleatória e sistemática e das funções de ligação bem como de determinados aspetos topológicos a desenvolver mais adiante. Um exemplo, é a relação direta já demonstrada anteriormente entre o Modelo Linear Generalizado (*Generalized Linear Model* - GLM) e o Perceptrão Simples, sendo este, simultaneamente, um caso particular da arquitetura GANN.

5.2 Arquitetura da rede

Para a construção de uma rede neuronal que mimetize um GAM, a arquitetura correspondente deverá incorporar partes (subarquiteturas) que mimetizem, por sua vez, as várias componentes do modelo de regressão e que podem ser traduzidas pela seguinte equação:

$$\mu = h(\beta_0 + f_1(x_{1i}) + f_2(x_{2i}) + ... + f_p(x_{pi})), \tag{5.1}$$

em que $\mu = E[Y|X1, ..., X_p]$, h representa a função de ligação. As funções f_1,...,f_p representam as funções suavizadoras univariadas, também conhecidas como funções parciais. As p variáveis explicativas são representadas por X_1,...,X_p (*e.g.* características dos doentes) e β_0 representa a ordenada na origem.

Tirando partido da característica de um MLP como aproximador universal a qualquer função contínua, este último pode ser utilizado, integrado na GANN, para mimetizar uma função parcial genérica.

Assim, considerando um MLP constituído por uma entrada, uma saída e uma única camada escondida (fig. 5.1), acrescida de uma ligação direta (*skip layer*) entre a camada de entrada e a saída da rede de forma a incluir um termo linear ω_{0j}, esta subarquitetura corresponderá à atrás referida função parcial, que será definida pela seguinte equação:

$$f_j(x_{ji}) = \omega_{0j}x_{ji} + \omega_{1j}tanh(\omega_{01j} + \omega_{11j}x_{ji}) + ... + \omega_{hj}tanh(\omega_{0hj} + \omega_{1hj}x_{ji}). \tag{5.2}$$

Para se obter uma rede neuronal correspondente ao modelo definido pela expressão 5.1, cada subarquitetura (função parcial) deverá estar ligada ao neurónio de saída através de uma sinapse com um peso fixo $\omega_{fxj} = 1$. A este neurónio também deverá estar ligado uma sinapse que irá traduzir o *bias* global do modelo, correspondendo à ordenada na origem β_0 (eq. 5.1).

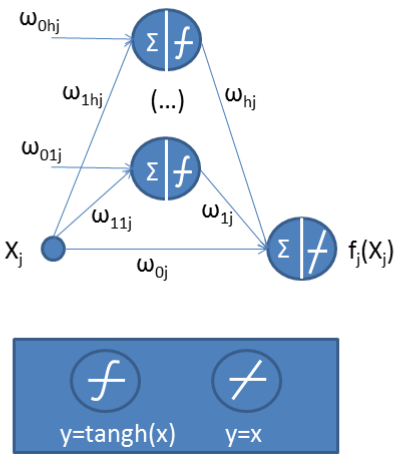

Figura 5.1: Subarquitetura correspondente à j-ésima função parcial.

A função de ligação $h(.)$ do modelo corresponderá à função de ativação do neurónio de saída. A arquitetura resultante, com função de ligação logística, encontra-se representada na fig. 5.2.

Tal como já referido, o GLM representa um caso particular de um GAM em que as funções parciais f_1,...,f_p são lineares, correspondendo a subarquiteturas de uma GANN constituídas apenas por uma *skip layer*, em que o conjunto, no seu global, forma uma estrutura equivalente ao Perceptrão Simples conforme a fig. 5.3.

Conseguindo-se estabelecer uma relação direta entre os pesos sinápticos da GANN e os coeficientes de um GLM, os primeiros podem ser inicializados recorrendo à estimação prévia dos segundos. Assim, nestas circunstâncias, será expectável que o desempenho desta rede neuronal seja equivalente ou superior à de um GLM.

Depois de definida a GANN, consegue-se depreender a principal característica que confere a sua interpretabilidade. Efetivamente, o facto das funções parciais dependerem de uma só variável, à semelhança de um GAM, torna possível uma análise isolada da relação de cada covariável com a resposta.

Essa análise pode ser efetuada com base nos resíduos parciais, que iremos abordar na secção 5.4.1.

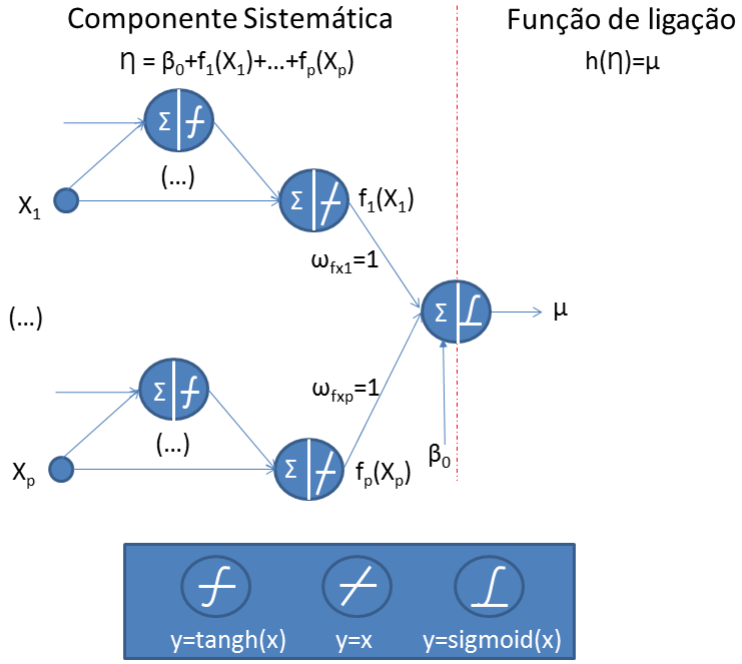

Figura 5.2: Arquitetura de uma GANN com função de ligação logística.

5.3 Estimação dos pesos sinápticos

Quanto à estimação dos pesos sinápticos, os métodos de treino utilizados para a estimação do MLP, descritos no capítulo 4, podem também ser utilizados para a estimação de uma GANN.

Por outro lado, é necessário ter em atenção que os problemas registados no processo de aprendizagem de um MLP também se aplicam ao caso da GANN, pelo que as estratégias utilizadas no treino do primeiro modelo podem também ser utilizadas no segundo.

5.4 Seleção de modelos

5.4.1 Seleção pela análise dos resíduos parciais

Como já anteriormente referido (ver secção 4.9), o número de neurónios a introduzir na camada escondida de um MLP é um assunto que ainda se

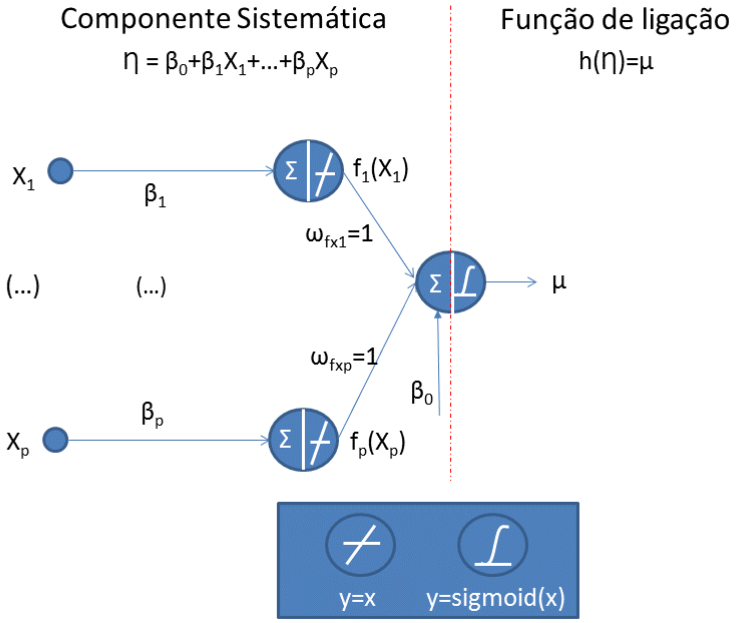

Figura 5.3: Arquitetura de uma GANN correspondente a um GLM.

mantém em aberto, uma vez não existir uma forma eficiente para o definir. Apesar da GANN utilizar esta arquitetura para definição das funções parciais, é possível através da adoção de técnicas já utilizadas nos modelos de regressão, nomeadamente nos GAMs, definir a arquitetura final. De facto, no caso específico de uma GANN, para a seleção do melhor modelo pode-se recorrer a análise informal de gráficos dos resíduos parciais, como já se faz com os GAMs. Para a obtenção destes gráficos, é importante a estimação dos resíduos parciais através da expressão (Potts, 1999):

$$pr_{ji} = g^{-1}[E(y)] - \beta_0 - \sum_{l \neq k} \hat{f}_l(x_l), \tag{5.3}$$

que é equivalente à expressão:

$$pr_{ji} = (g^{-1}(y_i) - g^{-1}(\hat{y}_i)) + \hat{f}_j(x_{ji}), \tag{5.4}$$

em que pr_{ji} representa o resíduo parcial da j-ésima variável, $\hat{y}_i$ é a estimativa da resposta do i-ésimo indivíduo, y_i representa o valor observado e $\hat{f}_j$ corresponde à estimativa da j-ésima função parcial.

No caso da função de ligação $g(.)$ ser uma função não linear, para a expressão anterior poderá ser utilizada uma aproximação de primeira ordem, resultando em:

$$pr_{ji} = \frac{\partial g^{-1}(y_i)}{\partial y}(y_i - \hat{y}_i) + \hat{f}_j(x_{ji}). \tag{5.5}$$

No caso da estimação de uma GANN, a análise visual utilizada na orientação do processo de seleção do modelo é efetuada sobre a leitura de gráficos, cujas abcissas e ordenadas representam respetivamente os valores da variável explicativa e os resíduos parciais. Com efeito, a análise visual dos resíduos possibilita a observação da sua relação com a respetiva função parcial e com base neste método, foi proposto por Potts (1999) um algoritmo interativo que permite conduzir o analista no processo de estimação do número de neurónios a introduzir em cada subarquitetura (MLP de base da função parcial) de uma GANN genérica.

Seguem-se, em detalhe, os passos que constituem o algoritmo interativo de Potts (1999):

Algoritmo interativo para a construção de uma GANN

1. Construir a GANN com um neurónio e uma *skip layer* para cada entrada, assumindo que os dados da amostra são previamente normalizados. Variáveis explicativas binárias (e.g. variáveis *dummy*) apenas possuem uma *skip layer*.

2. Cada entrada/função parcial irá corresponder, inicialmente, a 4 parâmetros sinápticos. No caso em que as variáveis são binárias, só existirá um parâmetro (correspondente a um MLP constituído apenas por uma *skip layer*).

3. Modelam-se os dados através de uma GLM de modo a gerar os valores iniciais a serem introduzidos como pesos sinápticos das *skip layers* existentes, bem como do parâmetro β_0 (*bias*) da GANN (ver fig. 5.2). Os restantes pesos são inicializados com base em valores aleatórios $\omega \sim N(0, 0.1)$.

4. Procede-se à adaptação da rede.

5. Através da observação do gráfico dos resíduos parciais, procede-se à remoção de neurónios no caso de existir uma relação linear entre os valores da variável explicativa e os resíduos parciais. No caso de esta

associação ser não linear, procede-se à adição de mais neurónios em cada subarquitetura que define a respetiva função parcial.

6. Repetem-se os passos a partir do ponto 4 só que, desta vez, podem-se aproveitar os valores resultantes desta época para inicializar os parâmetros da próxima.

Segue-se um exemplo com dados simulados em que se utiliza o algoritmo de Potts (1999).

Exemplo

Para exemplificar o algoritmo interativo, foi gerada uma amostra de dados a partir do algoritmo 8:

Entrada: $x_1, ..., x_p, f_1(.), ..., f_p(.)$, função de ligação $h(.)$
início
  Estimar $\eta = f_1(x_1) + ... + f_p(x_p)$;
  Estimar $\pi = h(\eta)$;
  para cada *indivíduo* i **faça**
    $b_i \sim U[0, 1]$;
    se $b_i \leq \pi_i$ **então**
      $y_i = 1$;
    senão
      $y_i = 0$;
    fim
  fim
  retorna *Amostra de dados simulados* $(y, x_1, ..., x_p)$
fim

Algoritmo 8: Algoritmo de simulação de dados.

São geradas duas variáveis, X_1 e X_2, ambas aleatórias, independentes e com distribuição Uniforme: $X_1 \sim U[-3, 3]$ e $X_2 \sim U[-2, 2]$. São também implementadas as seguintes funções parciais: $f_1(X_1) = X_1$ e $f_2(X_2) = {X_2}^2$.

A partir da execução deste algoritmo é gerada a variável resposta Y, de acordo com o seguinte modelo:

$$\mu = P[Y = 1|X_1 = x_1, X_2 = x_2] = h[f_1(x_1) + f_2(x_2)]. \tag{5.6}$$

Estes dados foram, então, utilizados para estimar uma GANN com função de ligação logística, através do método *5-Fold Cross Validation*.

Após os primeiros quatro passos do algoritmo interativo de Potts, em que as funções parciais correspondem a um MLP com *skip layer* e um neurónio escondido, obtiveram-se os gráficos 5.4 e 5.5 baseados nos valores dos resíduos parciais (ver eq. 5.5). É necessário ter em atenção que foi aplicada a restrição de identificabilidade $E[f_j(X_j)] = 0$, tal como se faz nos GAMs (Tibshirani e Hastie, 1987). Este assunto será abordado com maior detalhe na secção 5.5.

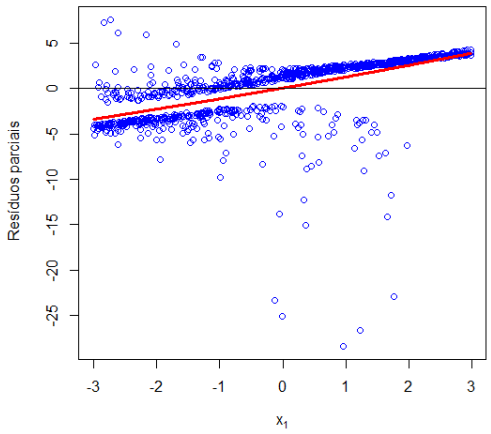

Figura 5.4: Gráfico dos resíduos parciais correspondente à variável explicativa X_1, obtido a partir de uma GANN com função de ligação logística e com a respetiva subarquitetura da função parcial da GANN, constituída por um neurónio escondido e uma *skip layer*.

No processo de orientação do analista para a escolha da complexidade do modelo, pode-se recorrer a um *spline* ajustado aos resíduos parciais. Se este *spline* resultar numa reta horizontal ou próximo, paralela ao eixo das abcissas, significa que a associação entre a variável explicativa e a resposta é fraca ou inexistente, pelo que nestes casos a variável poderá ser retirada do modelo. Este processo permite, então, também selecionar variáveis.

No caso do *spline* resultar numa reta com declive significativo, quer seja positivo ou negativo, tal como se pode observar na fig. 5.4, uma relação linear entre a variável explicativa e a resposta poderá ser a mais indicada. Assim,

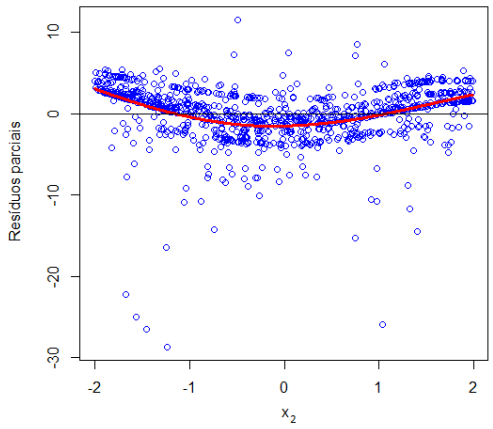

Figura 5.5: Gráfico dos resíduos parciais correspondente à variável explicativa X_2, obtido a partir de uma GANN com função de ligação logística e com a respetiva subarquitetura da função parcial da GANN constituída por um neurónio escondido e uma *skip layer*.

pode-se reduzir a complexidade do modelo, pela redução da complexidade do MLP em que se baseia a função parcial da GANN, com uma topologia que apenas possua uma *skip layer*.

Se o *spline* resultar numa função não linear, então a relação entre a variável explicativa e a resposta é não linear. Neste caso, podem-se adicionar neurónios à camada escondida. No entanto, dever-se-á ter em atenção o compromisso viés-variância, abordado na secção 4.8.1 e evitar adicionar um número excessivo de neurónios de forma a aumentar, em simultâneo, o viés e a variância (Du Toit, 2006).

Para auxiliar o analista nesta escolha, este pode recorrer a métricas utilizadas para a avaliação dos modelos (Potts, 1999). Neste exemplo, utilizamos o Erro Quadrático Médio (*Mean Squared Error* - MSE) obtido a partir da amostra de validação, de forma a contemplar o problema do compromisso viés-variância (ver secção 4.8.2). Verificou-se que não há uma melhoria significativa para um número acima de dois neurónios escondidos na subarquitetura da função parcial $f_2(X_2)$ da GANN em análise.

Desta forma, optou-se por uma *skip layer* correspondente à função parcial

$f_1(X_1)$ e um MLP com dois neurónios escondidos para $f_2(X_2)$, resultando nos respetivos gráficos das figuras 5.6 e 5.7, podendo-se observar a proximidade dos *splines* às funções parciais geradas.

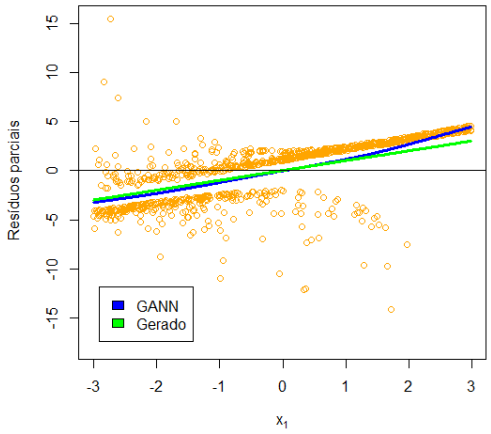

Figura 5.6: Gráfico dos resíduos parciais correspondente à variável explicativa X_1, obtido a partir de uma GANN com função de ligação logística e com o MLP da respetiva função parcial, constituído por apenas uma *skip layer*.

Com base na *package mgcv* (Wood, 2006) do software R (R Development Core Team, 2012), adaptámos ainda um GAM para comparar os gráficos dos resíduos e das funções parciais com os respetivos gráficos da GANN. Pelo que se pode observar, embora os modelos e os métodos de estimação se baseiem em diferentes conceitos, os resultados são bastante similares (ver figs. 5.8 e 5.9).

No entanto, este tipo de análise, por ser subjetiva, levanta alguns problemas, dado que a sua eficácia assenta sobretudo na experiência do analista. Desta forma, torna-se necessária a introdução, no processo de seleção do modelo, de um algoritmo que oriente de forma automática a construção da arquitetura da GANN, dispensando a decisão do analista de dados.

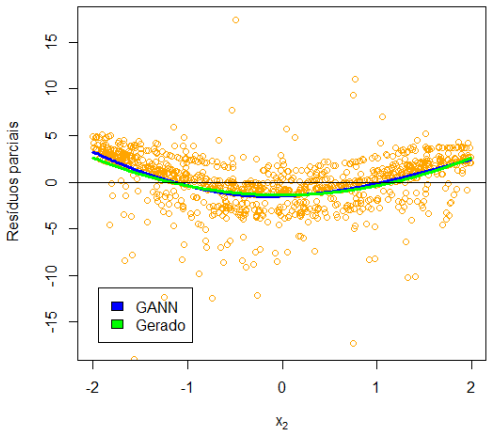

Figura 5.7: Gráfico dos resíduos parciais correspondente à variável explicativa X_1, obtido a partir de uma GANN com função de ligação logística e com o MLP da respetiva função parcial constituído por dois neurónios e uma *skip layer*.

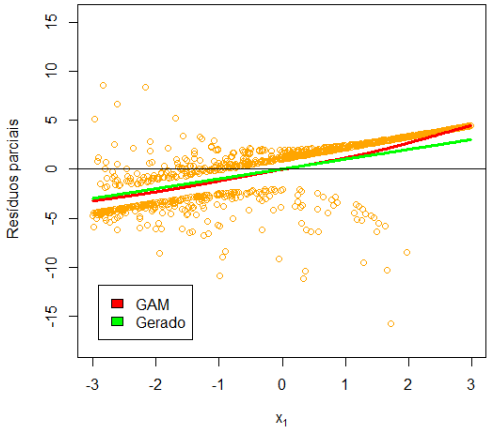

Figura 5.8: Gráfico dos resíduos parciais correspondente à variável explicativa X_1, obtido a partir de um GAM com função de ligação logística.

5.4.2 Seleção automática de modelos

Para se proceder à automatização do processo, há que definir o método de amostragem do modelo bem como as medidas de desempenho a utilizar no

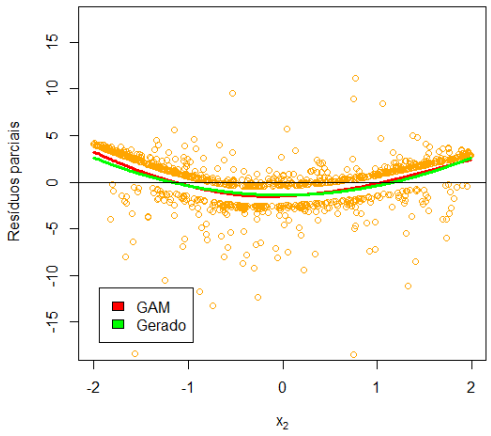

Figura 5.9: Gráfico dos resíduos parciais correspondente à variável explicativa X_2, obtido a partir de um GAM com função de ligação logística.

processo de seleção. Depois de definidas estas metodologias, pode-se aplicar um método de busca (*e.g.* procura cega, heurística, estocástica ou até mesmo exaustiva/*brute force*).

Neste sentido, foi introduzido por Du Toit (2006) um algoritmo de busca denominado por AutoGANN que organiza a seleção da melhor topologia através de uma árvore de procura. Neste algoritmo, uma rede neuronal com uma determinada topologia é representada por uma *string* baseada no esquema de codificação que se encontra na tabela 5.1. Por exemplo, ao ser aplicado o esquema de codificação proposto por Du Toit (2006) à topologia da fig. 5.10, a *string* correspondente será 253.

No intuito de aproximar o significado com o símbolo que codifica a subarquitetura da GANN, Bras-Geraldes et al. (2013) propõem uma modificação no esquema de codificação, de modo a facilitar a organização do processo de seleção, bem como a leitura da topologia adotada pelo modelo. O novo esquema proposto, descrito na tabela 5.2, codifica as características principais que definem a topologia de uma GANN: a existência ou ausência de *skip layer* e também o número de nós da camada escondida das subarquiteturas correspondentes às funções parciais. Esta informação é então integrada num vetor de dimensão equivalente ao número de variáveis explicativas.

Código	Descrição
0	atributo retirado do modelo
1	MLP apenas com *skip layer*
2	MLP sem *skip layer* e um nó na camada escondida
3	MLP com *skip layer* e um nó na camada escondida
4	MLP sem *skip layer* e dois nós na camada escondida
5	MLP com *skip layer* e dois nós na camada escondida
6	MLP sem *skip layer* e três nós na camada escondida
7	MLP com *skip layer* e três nós na camada escondida
8	MLP sem *skip layer* e quatro nós na camada escondida
9	MLP com *skip layer* e quatro nós na camada escondida

Tabela 5.1: Codificação da topologia de uma GANN segundo o algoritmo de Du Toit (2006).

Código	Descrição
null	atributo retirado do modelo
0	MLP apenas com *skip layer*
-1	MLP sem *skip layer* e um nó na camada escondida
1	MLP com *skip layer* e um nó na camada escondida
-2	MLP sem *skip layer* e dois nós na camada escondida
2	MLP com *skip layer* e dois nós na camada escondida
-3	MLP sem *skip layer* e três nós na camada escondida
3	MLP com *skip layer* e três nós na camada escondida
-4	MLP sem *skip layer* e quatro nós na camada escondida
4	MLP com *skip layer* e quatro nós na camada escondida

Tabela 5.2: Codificação da topologia de uma GANN segundo o algoritmo de Bras-Geraldes et al. (2013).

O sinal de cada coordenada corresponde à existência, no caso de ser positivo, ou ausência no caso negativo, de *skip layer* e o seu módulo representa o número de nós escondidos em cada subarquitetura, tendo sido estabelecida uma correspondência mais imediata para o analista, do grau de flexibilidade da função parcial resultante.

Este novo esquema de codificação dará origem ao vetor (-1,2,1) para codificar a topologia da fig. 5.10.

Apesar das diferenças entre os sistemas de codificação, o objetivo principal passa pela tradução da topologia de uma GANN numa *string* ou vetor.

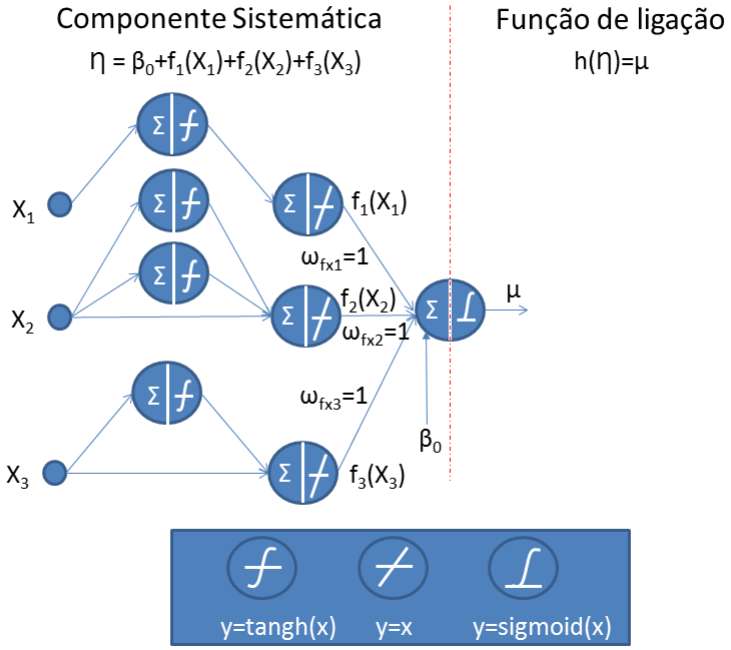

Figura 5.10: Exemplo de uma GANN com 3 entradas, em que o MLP base da função parcial $f_1(X_1)$, não possui *skip layer* e possui um neurónio na camada escondida, o correspondente a $f_2(X_2)$ possui *skip layer* e dois neurónios na camada escondida e a subarquitetura da função parcial $f_3(X_3)$ possui *skip layer* e um neurónio na camada escondida.

Este objetivo é satisfeito por ambos os sistemas, uma vez que o vetor resultante pode ser utilizado num processo de procura baseado na modificação das suas coordenadas.

Para concluir o processo de seleção automática de modelos é necessário definir um critério de comparação para se obter o melhor de entre os modelos concorrentes.

Para tal, Potts (1999) e de Waal e du Toit (2007) sugerem um dos seguintes critérios: o Critério de Informação de Akaike (*Akaike Information Criterion - AIC*), o Critério de Informação Bayesiano (*Schwarz Bayesian Criterion - SBC*) ou o MSE obtido a partir da validação cruzada (ver capítulo 3).

Para o caso de uma GANN, Du Toit (2006) considera que a distribuição dos resíduos é Gaussiana, pelo que a eq. 3.13, vem expressa na seguinte forma:

$$SBC = nlog(\hat{\sigma}^2) + Wlog(n), \tag{5.7}$$

em que

$$\hat{\sigma}^2 = \sum_{i=1}^{n} \frac{\hat{\epsilon}^2}{n}. \tag{5.8}$$

representa a estimativa de máxima verosimilhança de σ^2, W o número de parâmetros associado, n a dimensão da amostra e $\hat{\epsilon}$ representa o quadrado dos resíduos das estimativas da resposta.

De acordo com Du Toit (2006), o SBC penaliza mais a complexidade de um modelo que o AIC para $W > 7$.

Efetivamente, o SBC surge, a par do MSE, como o critério mais utilizado para a seleção de modelos na GANN uma vez que, em situações onde a dimensão da amostra é pequena, este método permite efetuar a avaliação do modelo sem necessidade de recorrer à utilização de amostra de teste (*in-sample model selection*) (Du Toit, 2006).

No entanto, dada a robustez da avaliação baseada no MSE obtido a partir da validação cruzada, optámos por utilizar este critério para seleção de modelos. Adicionalmente, como já referido na secção 3.2.2, tendo em conta os recursos disponíveis neste estudo, optou-se pela utilização de uma amostragem do tipo *5-Fold Cross Validation*. A partir do MSE obtido na amostra de validação é selecionado o melhor modelo tendo em conta não só os pesos sinápticos da rede, mas também o número de neurónios a introduzir na camada escondida de cada subarquitetura da GANN que define a respetiva função parcial.

Após a definição do sistema de codificação da topologia de uma GANN e da definição do critério de comparação das redes neuronais, a próxima etapa passa pela definição do algoritmo de busca que pretende selecionar, a partir de um conjunto de possíveis soluções, a GANN que melhor se adapta ao conjunto de dados existente.

5.4.3 O algoritmo AutoGANN

O algoritmo proposto por Du Toit (2006) (ver algoritmo 9) visa encetar uma busca local organizada a partir de uma topologia inicial da GANN.

```
1  início
2      modeloinicial := modelolinear;
3      open := [modeloinicial];
4      closed := [];
5      repita
6          Remove o modelo com maior fitness do conjunto open
             passando ao estatuto de modelo "pai";
7          Gerar os vários modelos com estatuto de modelo filho a partir
             do modelo pai;
8          para cada modelo "filho " faça
9              Estimar o respetivo fitness;
10             Adicionar o modelo "filho" à lista open;
11         fim
12         Adiciona modelo "pai" à lista closed;
13         Reordena a lista open por ordem decrescente do fitness;
14     até  open = [] OU criterioparagem for atingido;
15     retorna o melhor modelo do conjunto das duas listas open e
         closed;
16 fim
```

Algoritmo 9: Algoritmo AutoGANN.

A topologia a utilizar inicialmente poderá ser a equivalente ao GLM, em que os MLPs que suportam as respetivas funções parciais apenas possuem uma *skip layer*. Este modelo é traduzido por um vetor do tipo $(0, ..., 0)$, segundo o esquema da Tabela 5.2, e que será o esquema de codificação utilizado em todo o estudo.

Cada topologia é, então, descrita por um vetor e avaliada segundo uma métrica (*e.g.* MSE de validação ou SBC) de forma a que cada modelo possa ter associado um determinado *fitness* (este poderá ser calculado, por exemplo, com base no inverso do MSE de validação, ou seja, $fitness = \frac{1}{MSE}$).

Por outro lado, dever-se-á considerar que numa determinada iteração (geração), são gerados e estimados todos os modelos ("filhos") existentes na vizi-

nhança de um modelo "pai" genérico, ou seja, partindo de uma determinada topologia de GANN correspondente ao modelo "pai", obtém-se os "filhos" através da adição ou subtração de um neurónio em cada função parcial do primeiro modelo, procurando-se cobrir todas as combinações possíveis.

Na fig. 5.11, encontra-se um exemplo de árvore de procura gerada pela aplicação do algoritmo AutoGANN (ver algoritmo 9). Como se pode observar, o modelo "pai", com vetor associado $(0, 0)$, irá gerar os vetores com estatuto "filho" que se encontram na sua vizinhança $(-1, 0)$, $(1, 0)$, $(0, -1)$ e $(0, 1)$.

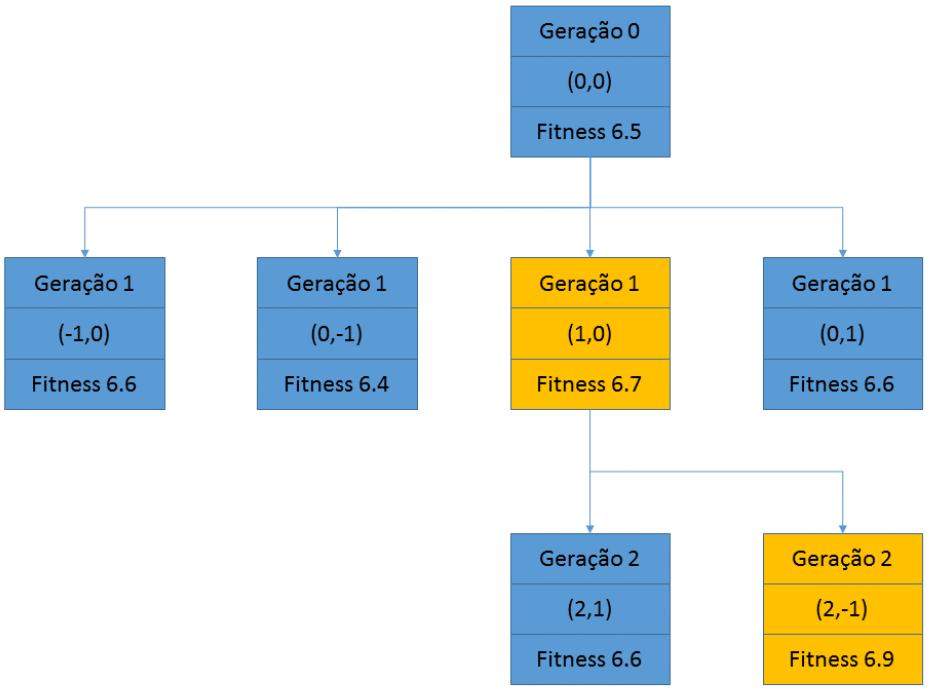

Figura 5.11: Exemplo de árvore de procura gerada pelo algoritmo AutoGANN.

O algoritmo é auxiliado por duas listas (ver algoritmo 9), sendo uma delas denominada por *open* e contém os modelos "filho" gerados e avaliados, e a outra, denominada por *closed*, inclui os modelos que já obtiveram o estatuto de "pai" (e a partir dos quais já foram gerados e avaliados os modelos "filho" respetivos). Após a avaliação do desempenho de cada um dos vetores

"filho", a lista *open* é reordenada por ordem decrescente com base no *fitness*, de forma a apurar-se o modelo com melhor desempenho dessa lista. Este último é então eleito como modelo "pai", sendo posteriormente retirado da lista *open* para dar início a uma nova geração de modelos "filho". Neste processo não são incluídos duplicados, ou seja, modelos que já foram anteriormente gerados e avaliados.

No exemplo da fig. 5.11, após a primeira iteração, o modelo representado pelo vetor $(1, 0)$ obteve o melhor *fitness*, pelo que servirá de modelo "pai" para a segunda geração.

Este algoritmo demonstra ser completo e robusto num espaço de procura limitado e em que haja tempo disponível para encontrar um bom modelo (du Toit e de Waal, 2010). No limite, este algoritmo é equivalente à procura exaustiva (*brute force*) uma vez que cobre todos os modelos possíveis, sendo a procura organizada de forma a encontrar mais cedo os melhores modelos. Sendo este método de procura local (uma vez que a árvore de procura se expande para modelos na vizinhança próxima do modelo "pai "), em espaços grandes (com um grande número de covariáveis) o tempo de procura irá ser muito significativo, pelo que, algumas modificações foram propostas por de Waal e du Toit (2011) de forma a mitigar este problema e que passaremos a descrever.

Multistep Expansion

Para reduzir o tempo de procura, foi introduzido no algoritmo AutoGANN um procedimento adicional (denominado por *Multistep Expansion*) e que passa pela introdução de um modelo extra cujo vetor representativo da sua topologia combina as coordenadas dos modelos "filho" que obtiveram um *fitness* melhor que o do modelo "pai".

Tomando como exemplo o cenário da fig. 5.11, verifica-se que alterando a primeira coordenada, os modelos "filho" apresentam melhor desempenho que o modelo pai, sendo que o melhor é o representado pelo vetor $(1, 0)$. Quanto à segunda coordenada, verifica-se que a adição de uma unidade resulta num modelo com melhor *fitness*. Assim sendo, será expectável que ao adicionar uma unidade a ambas as coordenadas do vetor que representa o modelo "pai", se obtenha um modelo "filho" promissor, como se pode observar na fig. 5.12.

Desta forma, a expansão poderá dar-se numa direção que inclua mais de uma coordenada. No entanto, ainda é possível melhorar o algoritmo,

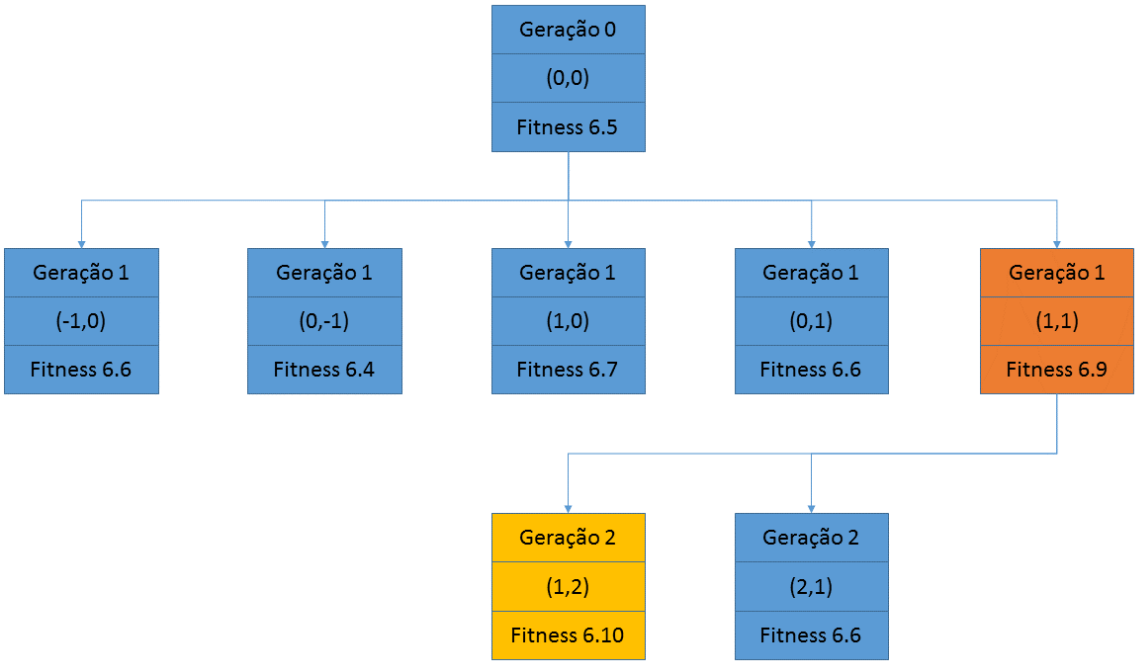

Figura 5.12: Exemplo de árvore de procura gerada pelo algoritmo AutoGANN com a funcionalidade *Multistep Expansion*.

permitindo o avanço de duas ou mais unidades por coordenada do vetor representativo da topologia da GANN.

Multistep Expansion melhorado

Continuando com o exemplo anterior, a partir do modelo "pai" $(0, 0)$ podem-se estimar os modelos "filho" ao longo de uma das coordenadas até se verificar que não há melhoria pelo aumento da flexibilidade da função parcial correspondente. Como se pode observar pela fig. 5.13, o modelo $(1, 0)$ apresenta o melhor desempenho. O mesmo processo deve ser repetido para as restantes coordenadas a partir deste modelo. Para mais detalhes sobre este algoritmo poderá ser consultado de Waal e du Toit (2011).

5.5 Identificabilidade

O problema da identificabilidade é uma questão que afeta em particular os modelos aditivos, não sendo a GANN uma exceção. Uma das formas para mitigar o problema da identificabilidade num modelo aditivo simples, segundo

Figura 5.13: Exemplo de aplicação da funcionalidade *Multistep Expansion* melhorado.

Tibshirani e Hastie (1987), passa pela aplicação da restrição $E[f_j(X_j)] = 0$, $f_j, j = 1...p$.

No caso dos GAMs, uma das formas identificadas para aplicar esta restrição, pode passar pela "reparametrização" do modelo durante a aplicação do método de estimação dos parâmetros. No caso da GANN, o método de estimação baseado na retropropagação do erro pode ser modificado de forma a que se aplique a restrição acima referida sem que as estimativas produzidas pelo modelo sejam alteradas.

Esta modificação ao nível do processo de estimação passa pela introdução de um passo adicional na fase de treino da rede, que se traduz pela obtenção das estimativas das funções parciais para cada época da fase de treino. É determinada a média das estimativas de cada função parcial e, durante a mesma época, procede-se ao treino da rede. Seguidamente, as médias apuradas são subtraídas às estimativas das funções parciais correspondentes, conforme a seguinte expressão:

$$\hat{f}_j^{\,*}(X_j) = \hat{f}_j(X_j) - E[\hat{f}_j(X_j)], j = 1...p. \tag{5.9}$$

Na fase de retropropagação do erro (durante a fase de treino), os parâmetros sinápticos são adaptados tendo em conta as estimativas corrigidas $\hat{f}_j^{\,*}(X_j)$. Para que este procedimento não enviese as estimativas, adiciona-se ao *bias* global $\hat{\beta}_0$, durante a mesma época, a soma das médias das estimativas apuradas, conforme a seguinte expressão:

$$\hat{\beta}_0^{\,*} = \hat{\beta}_0 + \sum_{j=1}^{p} E[\hat{f}_j(X_j)]. \tag{5.10}$$

Desta forma, as estimativas do preditor linear, assim como da resposta do modelo, são mantidas sem distorção.

Uma outra possibilidade para "centrar" as funções parciais em torno de zero, passa pela correção do modelo depois de estimado. Isso significa que após a obtenção das estimativas $\hat{f}_j$, e mantendo o mapa sináptico, à exceção do $\hat{\beta}_0$ (que será substituído por $\hat{\beta}_0{}^*$), é possível a correção do modelo implementando as equações 5.9 e 5.10. Para tal, há que adicionar ao neurónio de saída de cada função parcial, um *bias* $\beta_{xj} = -E[\hat{f}_j(X_j)]$ que atuará como parâmetro de localização da função, permitindo "centrá-la".

5.6 Conclusão

Muitas das metodologias utilizadas na implementação das GANNs foram inspiradas e resultaram do paralelismo existentes entre as redes neuronais e os modelos de regressão.

Assim sendo, embora o GAM e a GANN utilizem métodos distintos no processo de adaptação, os resultados que apresentam são bastante similares como seria expectável. Por isso, é natural que os métodos de avaliação utilizados para a estimação do primeiro modelo também possam ser utilizados ou adaptados no segundo. O mesmo se passa relativamente à seleção de modelos. Com efeito, os métodos utilizados nos GAMs, nomeadamente o processo de construção dos modelos conduzido pela análise dos resíduos parciais, também pode ser utilizado na GANN. De facto, é importante a determinação do grau de flexibilidade necessária para cada função parcial recorrendo-se, para isso, à interpretação do efeito de cada variável explicativa sobre a variável resposta, através de gráficos baseados nestes resíduos. Dada a subjetividade deste método, foi proposto por Du Toit (2006), um algoritmo para automatização da seleção do modelo.

Para tal, o mesmo autor propôs o algoritmo AutoGANN que, posteriormente, veio sofrer algumas modificações de forma a otimizar o tempo de procura, nomeadamente em problemas com um grande número de variáveis explicativas.

Nestes algoritmos de procura a topologia das redes são codificadas num vetor de modo a transformar o problema de seleção de modelos num processo de procura organizada. O facto de se utilizar um vetor para codificação da topologia da GANN, abre as portas para utilização de outros métodos, como os de procura global, onde se incluem os algoritmos genéticos (Bras-Geraldes et al., 2013). No entanto, o tempo de procura deverá ser melhorado pelo que

sugerimos este tema como um para possível desenvolvimento futuro.

Capítulo 6

Rede Neuronal Aditiva Generalizada com função de ligação flexível

6.1 Introdução

Como já referido, embora os métodos de estimação tradicionais aplicados às redes neuronais sejam distintos dos utilizados pelos modelos de regressão, verifica-se que grande parte da teoria utilizada por estes é também aplicável no contexto das redes neuronais, especialmente nas arquiteturas onde se pode estabelecer, diretamente, um paralelismo com os modelos de regressão. Como exemplos deste paralelismo podemos identificar a correspondência direta entre o Perceptrão Simples e o Modelo Linear Generalizado (*Generalized Linear Model* - GLM) e entre a Rede Neuronal Aditiva Generalizada (*Generalized Additive Neural Network* - GANN) e o Modelo Aditivo Generalizado (*Generalized Additive Model* - GAM).

No caso da arquitetura GANN, foi introduzida por de Waal et al. (2010) a implementação de uma rede com função de ligação flexível. A ideia subjacente a esse estudo passa pela utilização de um Perceptrão Multicamada *(Multi Layer Perceptron* - MLP) para definir esta função, tirando partido da sua característica como aproximador universal a uma qualquer função contínua (Du Toit, 2006). Esta abordagem encontra o seu paralelismo com o GAM com função de ligação não paramétrica (Cadarso-Suárez et al., 2005). Com efeito, o facto de se utilizar um MLP que funciona como função suavizadora, sem que *a priori* se assuma uma relação entre a variável resposta e o preditor linear, define este tipo de GANN como sendo de função de ligação

flexível não paramétrica.

À semelhança do proposto para o GAM em Papoila (2006), neste estudo iremos também utilizar uma abordagem paramétrica com a introdução na arquitetura GANN de uma função de ligação paramétrica pertencente à família de transformações assimétricas de Aranda-Ordaz (Aranda-Ordaz, 1981) (ver figs. 2.4 e 2.5). Esta solução representa um compromisso entre a GANN que utiliza uma função de ligação definida *a priori* (*e.g.* função logística) e a GANN que utiliza a função de ligação não paramétrica introduzida por de Waal et al. (2010) e de Waal e du Toit (2011) e que será objeto de uma análise mais detalhada neste estudo.

Devido ao facto do processo de aprendizagem, destes modelos, se basear no cálculo do gradiente, optou-se pela utilização de um caso particular da família de transformações assimétricas proposta por Aranda-Ordaz, em que o parâmetro $\psi \geq 0$ de forma a que:

$$h(\eta, \psi) = aranda(\eta, \psi) = \begin{cases} 1 - (1 + \psi e^{\eta})^{-\frac{1}{\psi}} & \text{se } \psi e^{\eta} > 0 \\ 1 - e^{-e^{\eta}} & \text{se } \psi = 0. \end{cases} \tag{6.1}$$

6.2 Função de ligação não paramétrica

6.2.1 Definição do modelo

Na fig. 6.1 pode-se observar a arquitetura proposta por de Waal et al. (2009, 2010). Comparando esta topologia com a arquitetura original proposta por Potts (1999) (ver fig. 5.1), verifica-se que esta última corresponde a um caso particular da primeira. Para que tal aconteça, basta que se considere o MLP, dedicado à estimação da função de ligação não paramétrica, constituído por apenas uma *skip layer* com o respetivo peso sináptico $\omega_{fl0} = 1$.

A função de ativação dos neurónios escondidos desse MLP pode ser do tipo sigmoidal ($\frac{1}{1+e^{-x}}$) (de Waal et al., 2009). Para o caso específico da variável resposta dicotómica, propõe-se neste estudo uma função de ativação $t(.)$, pertencente à família de transformações assimétricas de Aranda-Ordaz (onde também se encontra incluída a função logística), para o neurónio responsável pela resposta da rede neuronal, de forma a obter uma estimativa da probabilidade à saída da GANN. Esta escolha permite uma maior variedade de opções, quando comparada com a proposta original de de Waal et al. (2009), de forma a se poder abrir o espaço de procura no processo de estimação da

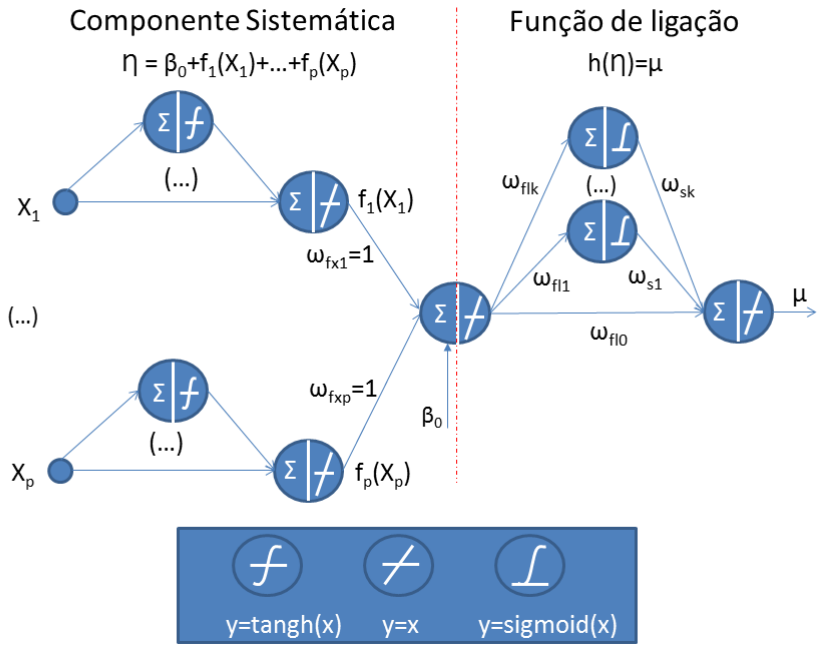

Figura 6.1: GANN com função de ligação flexível não paramétrica e função de ativação linear do neurónio de saída.

função de ligação não paramétrica.

As estimativas de $\hat{\mu}$ obtidas pela GANN não paramétrica são derivadas a partir da seguinte expressão:

$$\hat{\mu} = t\left(\hat{\eta}\omega_{fl0} + \frac{\omega_{s1}}{1+e^{-\hat{\eta}\omega_{fl1}}} + ... + \frac{\omega_{sk}}{1+e^{-\hat{\eta}\omega_{flk}}}\right). \tag{6.2}$$

Relativamente aos detalhes da implementação deste modelo, apesar da escassez de estudos nesta área, existem algumas considerações descritas por de Waal et al. (2009). De facto, são propostas melhorias, nomeadamente a introdução de um parâmetro β (*bias*) para correção da escala da combinação linear das saídas dos nós constituintes da camada escondida da função de ligação, bem como, a fixação do peso sináptico da entrada do primeiro nó escondido da função de ligação flexível ($\omega_{s1} = 1$) de forma a que as estimativas resultem numa função sigmoidal suave e não numa função em escada.

Assim, a estimativa de μ obtida a partir do modelo com função de ligação não paramétrica com as propostas de de Waal et al. (2009) incorporadas, resulta na seguinte expressão:

$$\hat{\mu} = t\left(\beta + \hat{\eta}\omega_{fl0} + \frac{1}{1+e^{-\hat{\eta}\omega_{fl1}}} + ... + \frac{\omega_{sk}}{1+e^{-\hat{\eta}\omega_{flk}}}\right). \tag{6.3}$$

Por outro lado, para uma variável resposta com distribuição de Bernoulli, de Waal et al. (2009) propõe a utilização da função logística como função de ativação do neurónio de saída no MLP que suporta a função de ligação. No entanto, verifica-se empiricamente, que o desempenho da estimação da função de ligação melhora substancialmente para $\omega_{fl0} = 1$. De igual forma, também se verifica empiricamente que a estimação da função de ligação depende do declive da função de ligação "real" relativamente ao declive da função de ativação escolhida para o neurónio de saída. Para que a estimação da função de ligação seja bem sucedida, o declive da função de ativação do neurónio de saída deverá ser, por um lado, inferior ao declive da função de ligação "real", e por outro, o mais próximo possível de forma a que o modelo produza boas estimativas.

Assim, para o caso de uma variável resposta com distribuição Bernoulli, além do proposto por de Waal et al. (2009), propomos neste estudo a introdução de uma função de ativação no neurónio de saída que possibilite várias opções quanto à definição do declive. Para tal, utilizamos como exemplo neste estudo, uma função de ativação pertencente à família de transformações assimétricas de Aranda-Ordaz (ver eq. 6.1).

6.2.2 Estimação do modelo

A estimação do modelo poderá ser realizada recorrendo aos mesmos procedimentos utilizados para a estimação de uma GANN com função de ligação definida *a priori* (ver capítulo 5). Quanto ao problema da identificabilidade em relação às funções parciais, este poderá ser resolvido recorrendo ao processo descrito na secção 5.5. No entanto, devido ao facto do *bias* geral β_0 também atuar como um parâmetro de localização da respetiva função de ligação, esta componente irá ser igualmente afetada pelo problema da identificabilidade. Este poderá ser mitigado considerando a solução proposta para o GAM não paramétrico por Cadarso-Suárez et al. (2005), em que foi implementada a restrição adicional $\hat{\beta}_0 = 0$ ou seja:

$$\hat{\eta} = \sum_{j=1}^{p} \hat{f}_j(X_j). \tag{6.4}$$

A implementação desta restrição encontra-se naturalmente realizada, uma vez que se pode considerar o parâmetro β_0 como fazendo parte da função de ligação. Como se pode observar pela fig. 6.2, a implementação da restrição 6.4 pode ser resolvida através da introdução de uma sinapse "fixa" (ω_{link}) com o valor 1 ao longo do treino, que irá separar a componente sistemática da função de ligação e integrar, ao mesmo tempo, o peso sináptico $\hat{\beta}_0^*$ dentro da função de ligação.

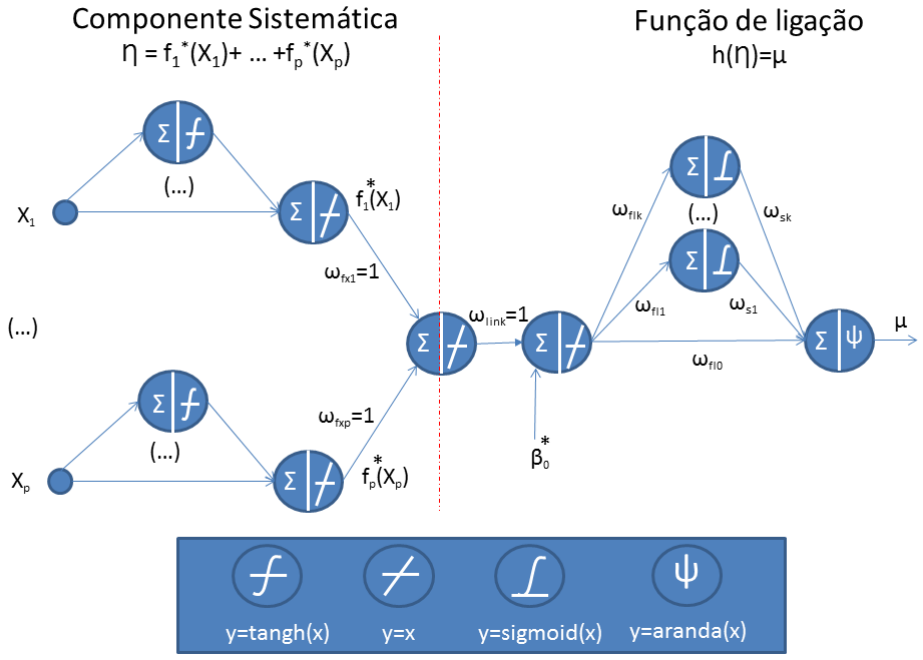

Figura 6.2: GANN Não paramétrica com restrições de identificabilidade implementadas.

6.3 Função de ligação paramétrica

6.3.1 Definição do modelo

Dadas as semelhanças entre a GANN e o GAM, é expectável que uma definição formal do primeiro modelo com função de ligação flexível paramétrica, seja equivalente à definição formal do segundo.

Assim, considerando uma GANN com entradas $X_1, ..., X_p$ representantes das p covariáveis do modelo, a componente sistemática da rede neuronal é dada pela seguinte expressão:

$$\eta = \beta_0 + \sum_{j=1}^{p} f_j(X_j), \tag{6.5}$$

em que a expressão da j-ésima função parcial é:

$$f_j(x_{ji}) = \omega_{0j}x_{ji} + \omega_{1j}tanh(\omega_{01j} + \omega_{11j}x_{ji}) + ... + \omega_{hj}tanh(\omega_{0hj} + \omega_{1hj}x_{ji}). \tag{6.6}$$

À semelhança da definição dos modelos de regressão em 2.83, a expressão que define a GANN com função de ligação flexível paramétrica será:

$$E(Y|X_1...X_p) = h(\eta, \boldsymbol{\psi}), \tag{6.7}$$

cuja componente aleatória Y possui uma distribuição pertencente à família exponencial, e $h(., \boldsymbol{\psi})$ representa uma função monótona crescente pertencente à família de funções paramétricas $H = \{h(., \boldsymbol{\psi}) : \boldsymbol{\psi} \in \Psi\}$.

Seguidamente, prosseguiremos com a implementação da GANN com função de ligação paramétrica para o caso particular da família de transformações assimétricas de Aranda-Ordaz conforme descrita em 6.1. Esta GANN será doravante designada por GANN Aranda.

Começando pela subarquitetura responsável pela componente sistemática, esta não sofre alterações, permanecendo tal como descrita na fig. 5.1. Na realidade, esta implementação da GANN servirá de base para implementar a função de ligação paramétrica, bastando substituir a função de ativação do neurónio de saída (definida *a priori*), que corresponde à função de ligação, por uma função de ativação pertencente à família de transformações de Aranda-Ordaz.

No que diz respeito à arquitetura da rede neuronal, a definição formal anterior resulta na topologia representada na fig. 6.3.

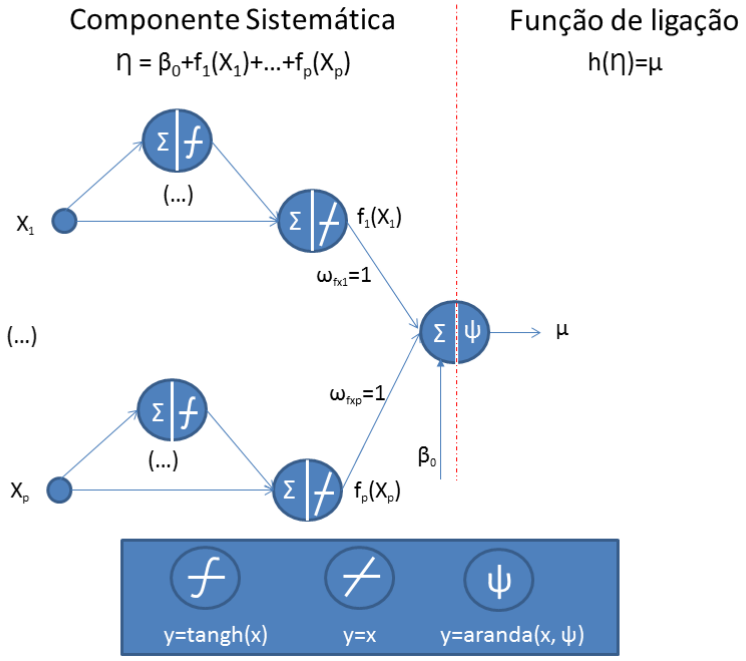

Figura 6.3: GANN com função de ligação paramétrica pertencente à família de transformações de Aranda-Ordaz.

Uma vez definido o modelo, será necessário enquadrar o parâmetro da função de ligação no processo de estimação global.

6.3.2 Estimação do modelo

Como já foi anteriormente referido, a estimação de uma rede neuronal é efetuada com base num processo de seleção de parâmetros (adaptação dos pesos sinápticos da rede) e por um processo de escolha dos seus hiperparâmetros (*e.g.* número de neurónios nas camadas intermédias).

Para a implementação do primeiro processo podem-se utilizar as mesmas técnicas de aprendizagem que se utilizam quer para adaptação de um MLP quer para a adaptação de uma GANN logística. Assim sendo, no caso de se

utilizarem técnicas baseadas no cálculo do gradiente, interessa determinar a derivada da função de ligação. No caso de esta pertencer à classe de transformações de Aranda-Ordaz (conforme descrita pela eq. 6.1), a derivada é dada pela seguinte expressão:

$$\frac{dh(\eta,\psi)}{d\eta} = \begin{cases} e^{\eta}(1+\psi e^{\eta})^{\frac{-1-\psi}{\psi}} & \text{se } \psi > 0 \\ -e^{\eta}e^{-e^{\eta}} & \text{se } \psi = 0. \end{cases} \tag{6.8}$$

Na fig. 6.4 podemos observar a variação desta derivada de acordo com diferentes valores para o parâmetro ψ.

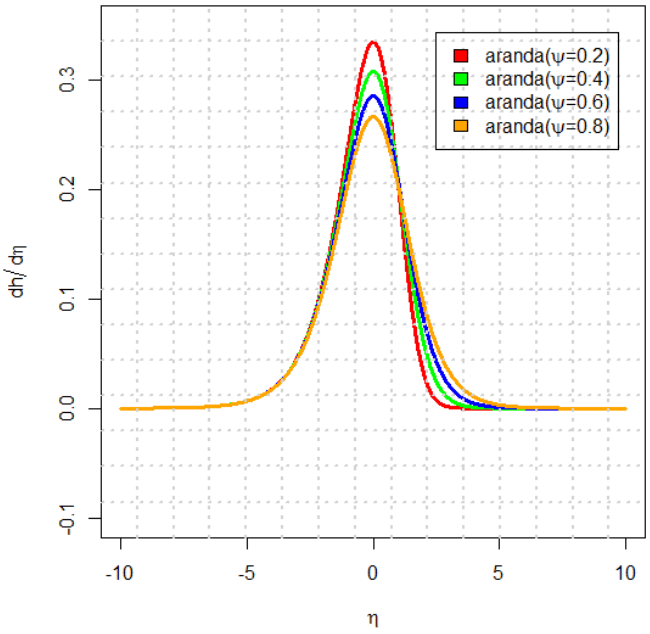

Figura 6.4: Alguns exemplos de derivadas da função de ligação pertencente à família de transformações de Aranda Ordaz, dado um parâmetro ψ específico.

Quanto à implementação do segundo processo, respeitante à definição da topologia da rede, pode-se considerar a função de ligação, dependente do parâmetro ψ, sujeita a um tipo de codificação semelhante ao utilizado no processo de estimação do número de neurónios escondidos (*e.g.* tabela 5.2). Para tal, pode-se utilizar uma codificação "ordinal" de forma a permitir a utilização dos métodos de seleção automática descritos no capítulo

3. A expressão proposta para se obter essa codificação deriva da seguinte informação:

- Valor máximo (ψ_{max}) e mínimo (ψ_{min}) de ψ da grelha que irá ser utilizada para selecionar o valor do parâmetro da função de ligação que otimiza o modelo.
- Número N de valores de ψ que constitui esta grelha. Este número deverá ser ímpar de forma a que codificação resultante seja simétrica em torno de zero.
- Cálculo do incremento de ψ dado pelo fórmula $\psi_{inc} = \frac{\psi_{max} - \psi_{min}}{N-1}$.
- Cálculo do valor mínimo ($alelo_{min} = -\frac{N-1}{2}$) a utilizar para codificar com número inteiro o valor de ψ_{min}.
- Cálculo do valor máximo ($alelo_{max} = \frac{N-1}{2}$) a utilizar para codificar com número inteiro o valor de ψ_{max}.

Assim, o código correspondente a um determinado $\psi = \psi_{min} + k\psi_{inc}$, em que $0 < k < N - 1$, é obtido a partir da expressão:

$$alelo(\psi) = alelo_{min} + \frac{(\psi - \psi_{min})(alelo_{max} - alelo_{min})}{\psi_{max} - \psi_{min}}. \tag{6.9}$$

Como exemplo, se pretender codificar uma grelha de 21 valores, $0.0 < \psi < 2.0$, obtém-se um incremento de $\psi_{inc} = 0.1$. Os códigos deverão estar compreendidos entre o valor mínimo $alelo_{min} = -10$ e o valor máximo $alelo_{max} = 10$, resultando na codificação expressa na tabela 6.1.

Código	Descrição
-10	Função de ligação Aranda-Ordaz com $\psi = 0$
-9	Função de ligação Aranda-Ordaz com $\psi = 0.1$
..	..
0	Função de ligação logística ($\psi = 1$)
..	..
9	Função de ligação Aranda-Ordaz com $\psi = 1.9$
10	Função de ligação Aranda-Ordaz com $\psi = 2.0$

Tabela 6.1: Codificação da função de ligação de uma GANN Aranda de acordo com uma grelha de valores de ψ.

Desta forma, um modelo arbitrário com p covariáveis poderá ser representado por um vetor $(H_1, H_2, ...H_p, alelo(\psi))$, em que H_j representa a codificação da topologia da subarquitetura correspondente à função parcial fj. De acordo com a tabela 6.1, o valor absoluto de $|H_j|$ corresponde ao número de neurónios da camada intermédia respetiva (relacionando-se com o grau de flexibilidade da subarquitetura) e o sinal de H_j representa a existência (+) ou não (-) de *skip layer* nessa mesma subarquitetura.

Tomando como exemplo a GANN descrita na fig. 6.5, verifica-se que a subarquitetura correspondente à primeira função parcial é constituída por 2 neurónios na camada intermédia e sem *skip layer*. Quanto à segunda subarquitetura, esta é constituída por 1 neurónio na camada intermédia e com *skip layer* incluída.

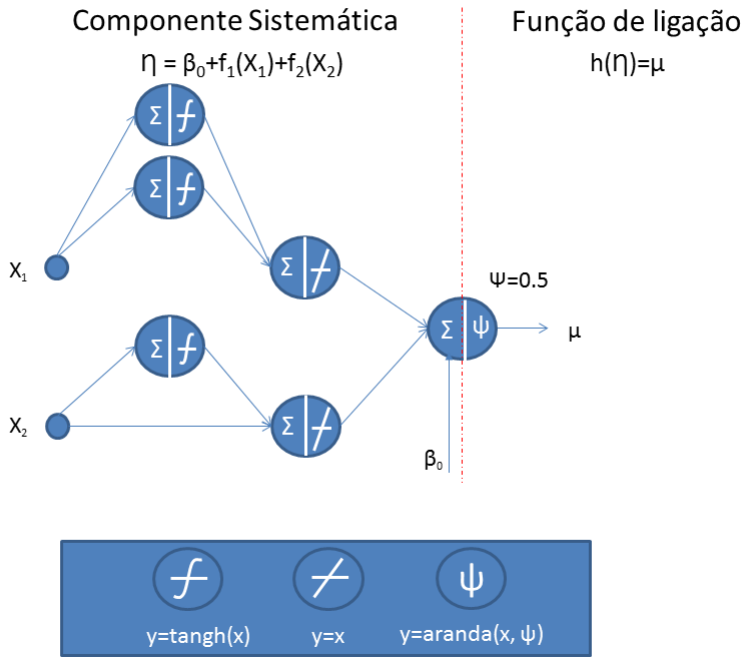

Figura 6.5: Exemplo de uma GANN com função de ligação pertencente à família de transformações de Aranda-Ordaz traduzida pelo vetor (-2,1,-5).

Tendo em atenção, neste exemplo, que a função de ativação do neurónio de saída (pertencente à família de funções assimétricas de Aranda Ordaz) tem como parâmetro $\psi = 0.5$, através das tabelas 5.2 e 6.1 pode-se definir o

vetor de códigos, identificador do modelo, como $(-2, 1, -5)$.

Desta forma, o processo de seleção começa pela definição dos modelos representados pelos referidos vetores. Posteriormente, os pesos sinápticos de cada um destes modelos são adaptados com base num processo de aprendizagem, em que se procura a otimização de uma função objetivo, recorrendo, por exemplo, à minimização do Critério de Informação Bayesiano (*Schwarz Bayesian Criterion - SBC*) ou do Erro Quadrático Médio (*Mean Squared Error - MSE*).

Desta forma, o processo de seleção pode-se basear num algoritmo de busca, em que o AutoGANN é um bom exemplo. De uma forma alternativa ou complementar podem-se também utilizar gráficos de perfil para estimação da função de ligação. Esses gráficos são construídos a partir de uma grelha de valores do parâmetro ψ da função de ligação flexível da GANN. Para cada um desses valores é estimado um modelo e medido, posteriormente, o seu desempenho de modo a obter o valor de ψ que conduz a um melhor desempenho da rede.

Quanto ao problema de identificabilidade do modelo, este pode ser abordado de uma forma semelhante ao caso da GANN com função de ligação definida *a priori* (Secção 5.5). Verifica-se, neste caso, que não é necessária a aplicação de restrições adicionais para além da restrição a que diz respeito a eq. 5.9 (acompanhada pela compensação introduzida pela eq. 5.10 para evitar a distorção das estimativas). Assim sendo, também neste caso se podem "centrar" as funções parciais em torno de zero, aplicando as equações atrás referidas durante o processo de adaptação dos parâmetros da rede ou, então, após o processo de adaptação através da introdução de um parâmetro de localização β_{xj} no neurónio de saída da função parcial f_j, como se pode observar na fig. 6.6.

6.4 Discussão

A utilização de um MLP como função de ligação de uma GANN (GANN Não paramétrica), foi introduzida com o intuito de funcionar como uma generalização do modelo de Potts baseado numa função de ligação definida *a priori*. Embora não seja este o modelo central no nosso estudo, pelo facto de não existirem muitos estudos sobre este assunto houve a necessidade de introduzir algumas inovações que constatámos serem necessárias à obtenção de um bom desempenho na estimação da função de ligação.

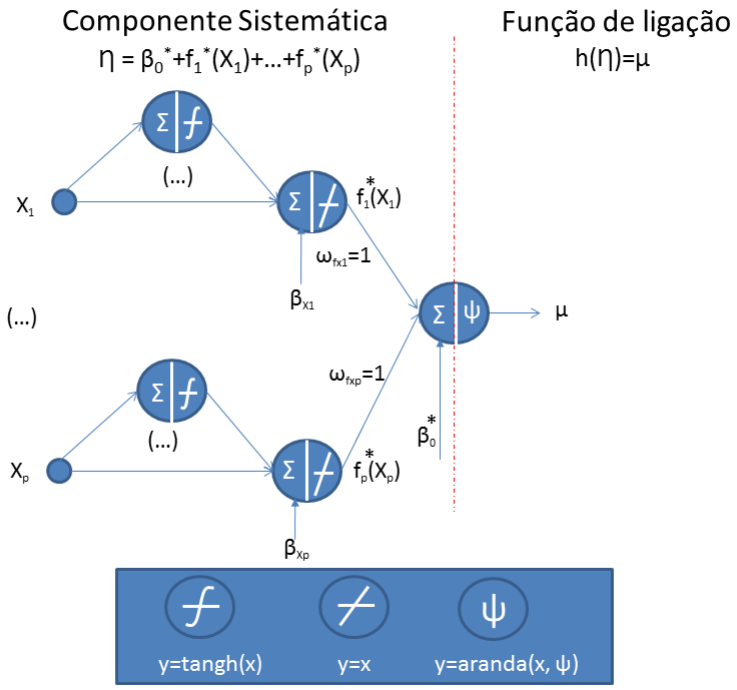

Figura 6.6: GANN paramétrica com restrições de identificabilidade implementadas.

Assim sendo, utilizou-se, para o caso de uma variável resposta com distribuição de Bernoulli, uma função de ativação do neurónio de saída do MLP (que define a função de ligação da GANN Não paramétrica) pertencente à família de transformações assimétricas de Aranda-Ordaz. A escolha desta família de funções permite, através da definição de um parâmetro ψ, selecionar uma função com declive arbitrário. Como referido anteriormente, constatou-se, empiricamente, que o bom desempenho da estimação da função de ligação depende do facto do declive "real" ser maior que o declive da função de ativação do neurónio de saída.

Um dos temas centrais deste estudo gira em torno da GANN com função de ligação paramétrica que representa uma solução intermédia entre a GANN Não paramétrica (já referida anteriormente) e a GANN com função de ligação definida *a priori*. Uma das vantagens, à partida, desta arquitetura tem a ver com a menor complexidade do modelo em relação à GANN Não paramétrica, ao mesmo tempo que oferece mais opções que uma GANN com

função de ligação definida *a priori*.

Adicionalmente, esta arquitetura também pode ser complementar à GANN com função de ligação não paramétrica, uma vez que poderá dar uma indicação do declive da função de ligação "real", de modo a que se possa utilizar essa informação para a definição da função de ativação do neurónio de saída da GANN com função de ligação não paramétrica. No entanto, sugerem-se mais estudos sobre a GANN Não paramétrica em trabalhos futuros.

Capítulo 7

Rede Neuronal Aditiva Generalizada com função de ligação flexível: Estudo de simulação

7.1 Introdução

O estudo de simulação que irá ser apresentado neste capítulo consta de um conjunto de análises comparativas entre uma Rede Neuronal Aditiva Generalizada (*Generalized Additive Neural Network* - GANN) com função de ligação logística (GANN Logística), uma GANN com função de ligação paramétrica pertencente à família de transformações de Aranda-Ordaz (GANN Aranda) e uma GANN (GANN Não paramétrica) cuja função de ligação se baseia num Perceptrão Multicamada (*Multi Layer Perceptron* - MLP). No que respeita à GANN Não paramétrica a função de ativação do neurónio de saída (do MLP que define a função de ligação) pertence à família de transformações assimétricas de Aranda-Ordaz, com parâmetro ψ fixo a 2.0.

Os dados foram simulados a partir do algoritmo 8 que descreve o processo de simulação também utilizado em Papoila (2006) e Cadarso-Suárez et al. (2005). Este começa pela geração de duas variáveis, X_1 e X_2, independentes e com distribuição Uniforme: $X_1 \sim U[-3, 3]$ e $X_2 \sim U[-2, 2]$. A partir desse mesmo algoritmo foram geradas as funções parciais $f_1(X_1) = X_1$ e $f_2(X_2) = {X_2}^2$, assim como a variável resposta Y, de acordo com o seguinte modelo:

$$\mu = P[Y = 1|X_1 = x_1, X_2 = x_2] = h[f_1(x_1) + f_2(x_2)]. \tag{7.1}$$

Efetuaram-se dois estudos de simulação com dados gerados a partir de famílias de transformações distintas. No primeiro, foi escolhida uma função de ligação $h(.)$ pertencente à família de transformações assimétricas de Aranda-Ordaz (ver eq. 6.1). Como foi já referido anteriormente (ver secção 2.4.2), esta função de ligação depende de um só parâmetro ψ. Excluindo o caso em que $\psi = 1$ (função logística), é expectável que a utilização desta função de ligação para geração de dados seja mais favorável à GANN Aranda. Assim sendo, num segundo estudo de simulação, utilizou-se uma função de ligação que não beneficia a GANN Aranda, pertencente à família de transformações Czado, definida através das seguintes expressões:

$$h(\eta, \boldsymbol{\psi}) = \frac{e^{f(\eta,\boldsymbol{\psi})}}{1 + e^{f(\eta,\boldsymbol{\psi})}}, \tag{7.2}$$

em que $f(\eta, \boldsymbol{\psi})$ é dada por:

$$f(\eta, \boldsymbol{\psi}) = \begin{cases} \eta_0 + \frac{(\eta-\eta_0+1)^{\psi_1}-1}{\psi_1} & se \quad \eta \geq \eta_0 \\ \eta_0 - \frac{(-\eta+\eta_0+1)^{\psi_2}-1}{\psi_2} & se \quad \eta < \eta_0. \end{cases} \tag{7.3}$$

As caudas direita e esquerda da função de ligação são controladas, respetivamente pelos parâmetros ψ_1 e ψ_2, de forma independente. A função de ligação logística representa um caso particular desta família quando $\eta_0 = 0$, $\psi_1 = 1.0$ e $\psi_2 = 1.0$.

Nas figs. 7.1 e 7.2 pode-se observar um exemplo de várias funções com caudas simétricas (correspondentes a parâmetros $\psi 1$ e $\psi 2$ com valores idênticos) bem como o gráfico das funções de ligação correspondentes, respetivamente.

Na fig. 7.3 pode-se observar um exemplo de várias funções com caudas assimétricas (correspondentes a parâmetros $\psi 1$ e $\psi 2$ com valores diferentes) e, adicionalmente, da função logística. Na fig. 7.4 estão representados os gráficos das funções de ligação correspondentes.

Neste segundo estudo de simulação foram, então, considerados dois cenários:

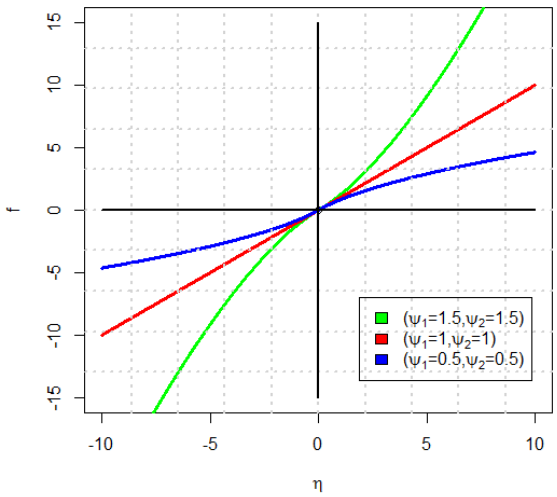

Figura 7.1: Exemplos de funções simétricas da família de transformações Czado para vários valores dos parâmetros $\psi 1$ e $\psi 2$.

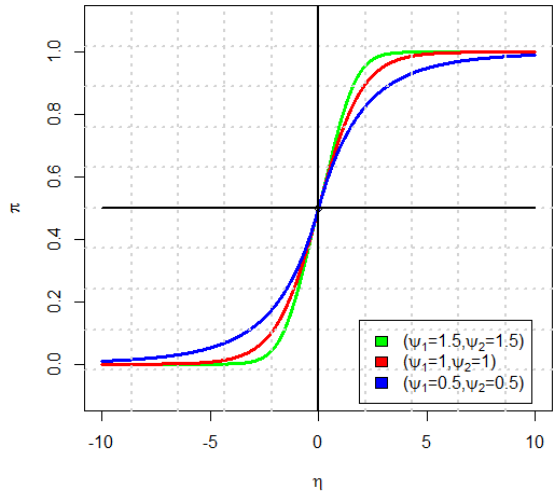

Figura 7.2: Funções de ligação correspondentes às funções apresentadas na fig. 7.1.

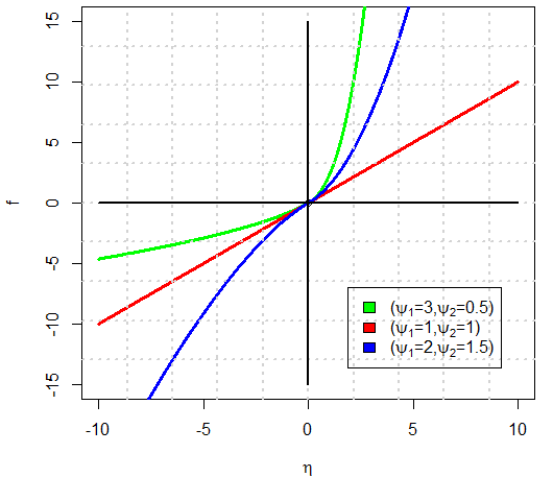

Figura 7.3: Exemplos de funções assimétricas da família de transformações Czado para duas diferentes combinações de valores dos parâmetros $\psi 1$ e $\psi 2$.

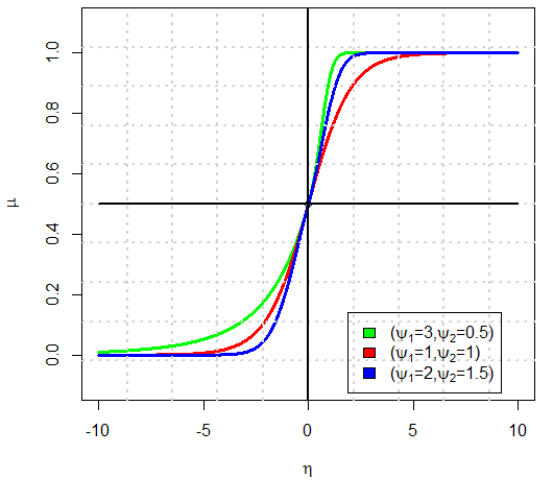

Figura 7.4: Funções de ligação correspondentes às funções apresentadas na fig. 7.3.

- Os dados foram gerados a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0$, $\psi_1 = 1.0$ e $\psi_2 = 1.0$, por forma a favorecer a GANN Logística.

- Os dados foram gerados a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0$ e $\psi_2 = 0.9$, por forma a não favorecer nenhuma das GANNs em análise.

Em ambos os estudos de simulação, tanto no caso dos dados gerados através da função de ligação pertencente à família de transformações de Aranda-Ordaz como no caso da família de transformações Czado, foram geradas 1000 amostras com uma dimensão de 1000 indivíduos cada.

Os modelos foram treinados utilizando o método de validação cruzada (*5-Fold Cross-Validation*), tendo sido selecionado o modelo com melhor Erro Quadrático Médio (*Mean Squared Error - MSE*) com base na amostra de validação. Posteriormente, registou-se o respetivo valor de MSE obtido a partir da aplicação deste modelo selecionado na amostra de teste.

Obtiveram-se, ainda, os gráficos das médias das estimativas da função de ligação obtidas a partir das 1000 amostras geradas. No caso dos gráficos das funções parciais, estes foram obtidos a partir da aplicação de um modelo de regressão linear (no caso de f_1) e de um modelo de regressão polinomial quadrático (no caso de f_2).

Uma vez que no estudo de simulação é importante a comparação das formas funcionais das funções parciais e da função de ligação estimadas a partir dos modelos em estudo, foi adicionado um termo no modelo simulado de modo a que as respetivas funções parciais sejam "centradas". Ao mesmo tempo, esse termo é adicionado como parâmetro de localização global ξ, de maneira a que o modelo definido em 7.1 é reescrito da seguinte forma:

$$\mu = P[Y = 1|X_1 = x_1, X_2 = x_2] = h(f_1(x_1)+f_2(x_2)-E[f_1(X_1)]-E[f_2(X_2)]+\xi), \tag{7.4}$$

em que

$$\xi = E[f_1(X_1)] + E[f_2(X_2)]. \tag{7.5}$$

Assim, é possível aplicar as restrições de identificabilidade 5.9 e 5.10 aos modelos em análise e, simultaneamente, compará-los com o modelo simulado.

7.2 Dados gerados com função de ligação Aranda-Ordaz

7.2.1 Resultados para $\psi = 0.5$

Seguidamente iremos apresentar uma análise exploratória dos valores obtidos a partir dos dados gerados com base no modelo descrito pela eq. 7.4 e cuja função de ligação $h(.)$ pertence à família de transformações de Aranda-Ordaz com parâmetro $\psi = 0.5$.

Comecemos pela função de ligação "média" que se obtém a partir das médias das estimativas de μ da GANN em análise, bem como das estimativas do seu preditor linear η. Como se pode observar pelo gráfico da fig. 7.5, a função "média" estimada pelo GANN Aranda é a que se encontra mais próxima da função de ligação do modelo simulado.

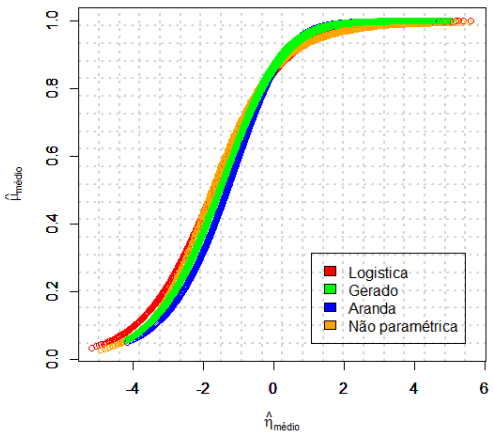

Figura 7.5: Médias das funções de ligação estimadas a partir de uma GANN Logística, GANN Aranda, GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 0.5$.

Relativamente à distribuição das probabilidades estimadas a partir das GANNs em análise, pode-se observar pela fig. 7.6 que a distribuição correspondente à GANN Aranda é a que mais se assemelha à gerada.

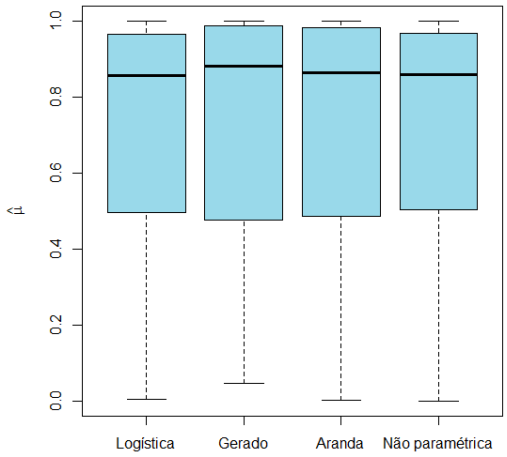

Figura 7.6: Distribuição das probabilidades estimadas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 0.5$.

Pelo gráfico da fig. 7.7, construído com base nos resultados da fig. 7.5, pode-se melhor observar a discrepância entre as funções de ligação "médias", estimadas a partir dos modelos em estudo.

Analisando a componente sistemática das GANNs em estudo, a fig. 7.8 é inconclusiva quanto à discrepância em relação à gerada das estimativas dos preditores lineares das respetivas GANNs. No entanto, o gráfico de calibração da fig. 7.9 salienta melhor essa discrepância.

Análise do desempenho preditivo e discriminativo

Ainda, relativamente ao poder preditivo das várias GANNs, foram utilizados os MSE obtidos a partir das amostras de teste. Considerando estas mesmas amostras, obtiveram-se as $AUCs$ (área sob a *Receiver Operating Characteristic (ROC) curve*) para medir o poder discriminativo. Assim, na fig. 7.10 podem-se observar as distribuições do MSE obtidas a partir da GANN logística, GANN Aranda e GANN não paramétrica. Como se pode constatar, o poder preditivo de todos os modelos é muito semelhante, observando-se, no entanto, uma pequena melhoria dos modelos com função de ligação flexível face ao modelo com função de ligação logística.

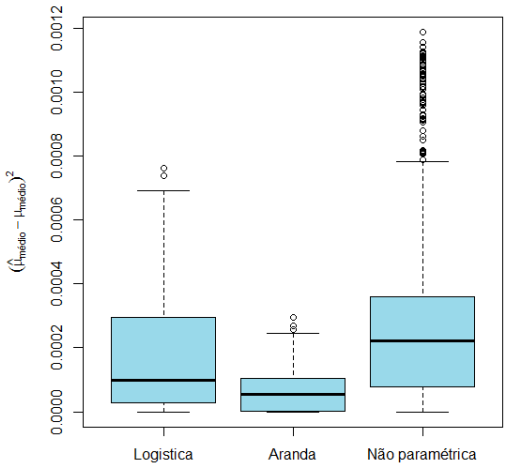

Figura 7.7: Distribuição do quadrado das distâncias das médias das probabilidades estimadas às médias das probabilidades geradas a partir de um modelo com função de ligação pertencente à família de transformações Aranda-Ordaz com $\psi = 0.5$.

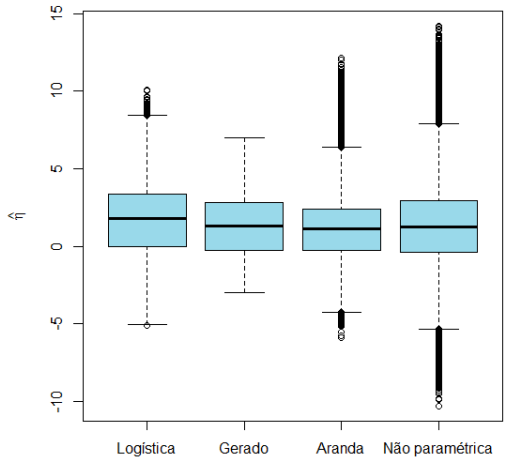

Figura 7.8: Distribuição das estimativas dos preditores lineares obtidas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 0.5$.

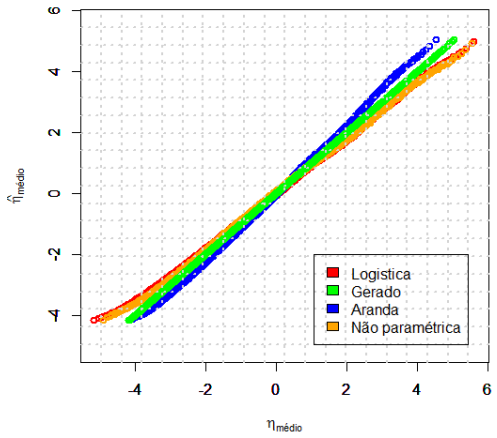

Figura 7.9: Gráfico de calibração das médias das estimativas do preditor linear, obtidas, a partir da GANN Logística, da GANN Aranda e da GANN Não Paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 0.5$.

Quanto ao poder discriminativo de cada uma das redes neuronais em estudo, a fig. 7.11 mostra a distribuição das $AUCs$. Como se pode observar, também aqui a semelhança entre os desempenhos dos vários modelos é visível embora de novo, com uma pequena vantagem por parte dos modelos com função de ligação flexível.

7.2.2 Resultados para $\psi = 1.0$

Como já referido anteriormente (ver secção 2.4.2), a função de ligação pertencente à classe de transformações de Aranda-Ordaz com parâmetro $\psi = 1.0$ é equivalente à função de ligação logística. Como seria de esperar, a GANN Logística foi a que melhor se ajustou aos dados, embora, os modelos com função de ligação flexível, nomeadamente a GANN Aranda, tenham apresentado resultados bastante satisfatórios.

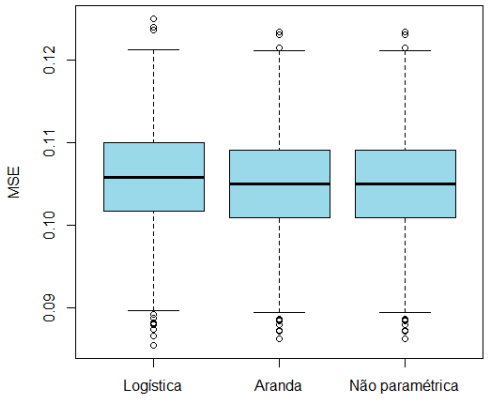

Figura 7.10: Distribuição do MSE da GANN logística, GANN Aranda e GANN não paramétrica para dados gerados a partir de um modelo com função de ligação pertencente à família de transformações de Aranda Ordaz com $\psi = 0.5$.

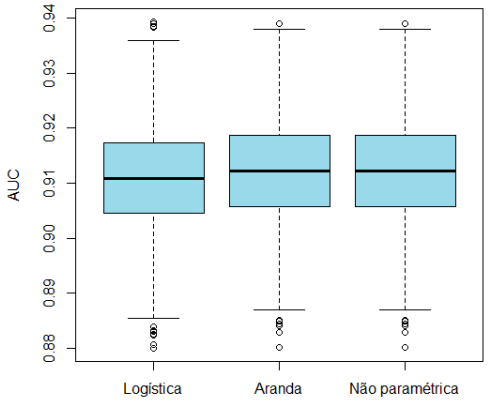

Figura 7.11: Distribuição das $AUCs$ da GANN logística, GANN Aranda e GANN não paramétrica para dados gerados a partir de um modelo com função de ligação pertencente à família de transformações de Aranda Ordaz com $\psi = 0.5$.

7.2.3 Resultados para $\psi = 1.5$

Tal como no caso anterior, procedeu-se à análise comparativa das GANNs Logística, Aranda e Não paramétrica relativamente ao modelo simulado com função de ligação de Aranda-Ordaz $\psi = 1.5$.

Começando pela função de ligação "média", pode-se observar pela fig. 7.12 que as estimativas obtidas pelas GANNs com função de ligação flexível se encontram muito próximas da função de ligação gerada.

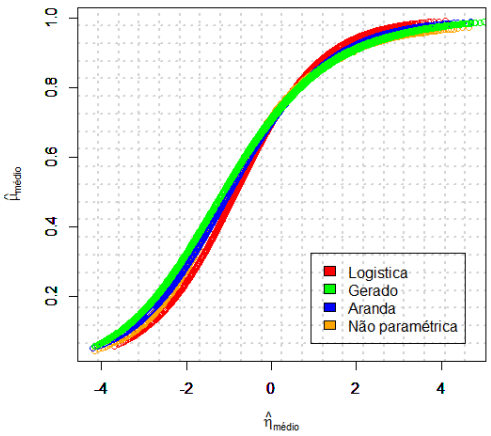

Figura 7.12: Médias das funções de ligação estimadas a partir de uma GANN Logística, GANN Aranda, GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 1.5$.

Na fig. 7.13, apresentamos o gráfico da distribuição das estimativas de μ para cada GANN. Constata-se que a sua leitura não é elucidativa quanto ao desempenho relativo de cada modelo. No entanto, analisando a distância das médias das estimativas de μ das várias GANNs relativamente às médias do gerado, pode-se observar, através do gráfico da fig. 7.14, a divergência, embora pequena, existente entre a GANN logística e o modelo simulado. Em contraste, verifica-se que as GANNs com função de ligação flexível, demonstram uma maior aproximação ao modelo real.

Seria expectável que a referida divergência entre a GANN Logística e o modelo simulado tivesse repercussões na distribuição das estimativas dos

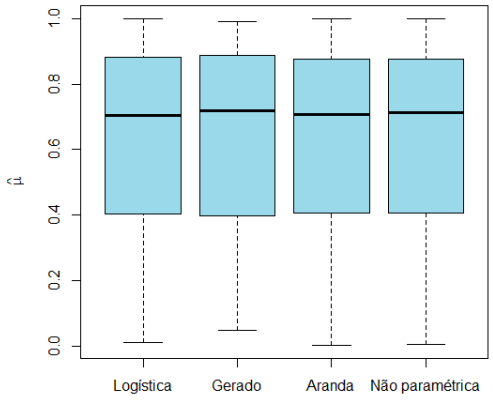

Figura 7.13: Distribuição das probabilidades estimadas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 1.5$.

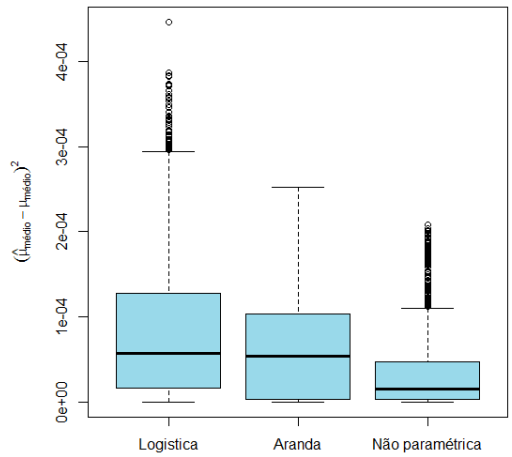

Figura 7.14: Distribuição do quadrado das distâncias das médias das probabilidades estimadas às médias das probabilidades geradas a partir de um modelo com função de ligação pertencente à família de transformações Aranda-Ordaz com $\psi = 1.5$.

preditores lineares de cada um dos modelos. No entanto, o gráfico da fig. 7.15 não é conclusivo, pelo que se pode recorrer ao gráfico de calibração dos preditores lineares (ver fig. 7.16), onde melhor se pode observar a divergência entre as GANNs em análise. Também aqui se observa uma melhoria dos modelos com função de ligação flexível face ao modelo com função de ligação logística.

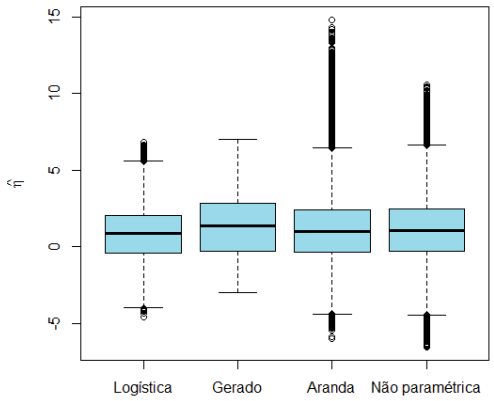

Figura 7.15: Distribuição das estimativas dos preditores lineares obtidas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 1.5$.

Análise do desempenho preditivo e discriminativo

Na fig. 7.17 podem-se comparar as distribuições do MSE obtidas a partir da amostra de teste, verificando-se, neste caso, uma semelhança entre as GANNs em análise.

Por último, quanto ao poder discriminativo, verifica-se que as $AUCs$ obtidas pelas GANNs em análise, igualmente, possuem distribuições semelhantes (fig. 7.18).

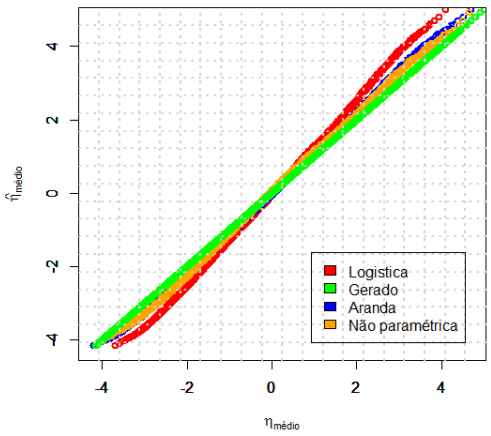

Figura 7.16: Gráfico de calibração das médias das estimativas do preditor linear, obtidas, respetivamente, a partir da GANN Logística, da GANN Aranda e da GANN Não Paramétrica, com base em dados gerados por um modelo simulado com função de ligação pertencente à família de transformações de Aranda-Ordaz com $\psi = 1.5$.

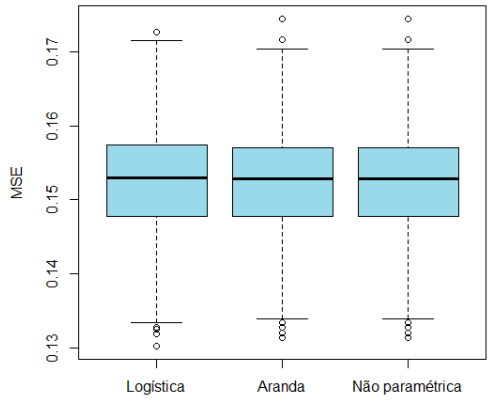

Figura 7.17: Distribuição do MSE da GANN logística, GANN Aranda e GANN não paramétrica para dados gerados a partir de um modelo com função de ligação pertencente à família de transformações de Aranda Ordaz com $\psi = 1.5$.

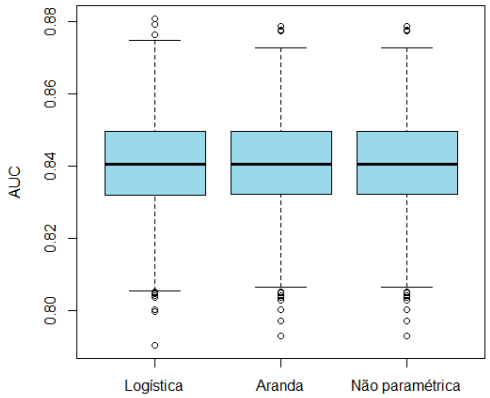

Figura 7.18: Distribuição das $AUCs$ da GANN logística, GANN Aranda e GANN não paramétrica para dados gerados a partir de um modelo com função de ligação pertencente à família de transformações de Aranda Ordaz com $\psi = 1.5$.

7.3 Dados gerados com função de ligação Czado

Nas secções anteriores verificou-se que a GANN com função de ligação flexível origina resultados muito semelhantes aos de uma GANN com função de ligação logística. No entanto, os cenários simulados basearam-se num modelo que, inevitavelmente, iria beneficiar a GANN Aranda. Assim sendo, na seguinte análise, o modelo simulado tem como base uma função de ligação pertencente à família de transformações Czado com três parâmetros (η_0, ψ_1 e ψ_2 - ver eq. 7.3), de forma a que a GANN Aranda não seja "beneficiada" relativamente à GANN Logística.

7.3.1 Resultados para $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$

Como seria de esperar, a média das funções de ligação estimadas pela GANN Logística encontra-se sobreposta à função de ligação do modelo simulado (fig. 7.19). Quanto à função de ligação "média" estimada pelas GANNs com função de ligação flexível, verifica-se que a da GANN Aranda é a que se encontra mais próxima da função de ligação do modelo simulado seguida pela GANN Não paramétrica, com uma distância negligenciável.

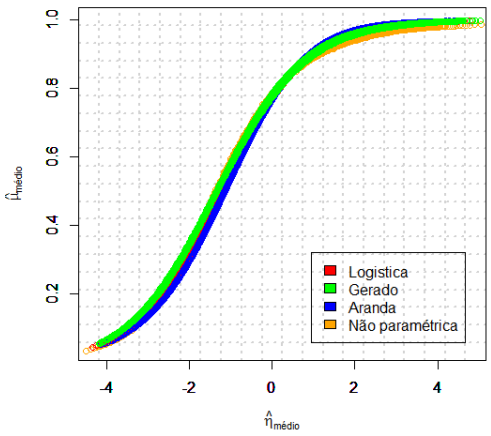

Figura 7.19: Médias das funções de ligação estimadas a partir de uma GANN Logística, GANN Aranda, GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

Relativamente à distribuição das estimativas de μ dos modelos, verifica-se que apresentam distribuições semelhantes como se pode observar pela fig. 7.20.

Tal como seria expectável, no que diz respeito às distâncias das médias das probabilidades estimadas pelas GANNs às médias das probabilidades obtidas a partir do modelo simulado, a GANN Logística é a que apresenta uma situação mais favorável. No entanto, como se pode constatar pela fig. 7.21, as distâncias correspondentes ao GANN Aranda são negligenciáveis.

Da mesma forma, as medianas das distribuições das estimativas dos preditores lineares obtidas a partir das GANNs são semelhantes e próximas da mediana da distribuição do preditor linear do modelo simulado, como se pode observar pela fig. 7.22.

Pelo gráfico de calibração da fig. 7.23, pode-se igualmente visualizar a convergência das estimativas dos preditores lineares das GANNs em análise, relativamente ao gerado. Como seria expectável, a GANN Logística apresenta uma distribuição de valores próxima à do modelo gerado. A GANN Não paramétrica apresenta um desempenho semelhante, e a GANN Aranda

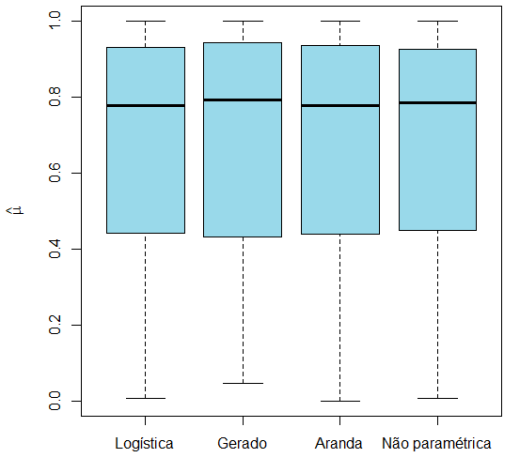

Figura 7.20: Distribuição das probabilidades estimadas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0 \psi_1 = 1.0, \psi_2 = 1.0$.

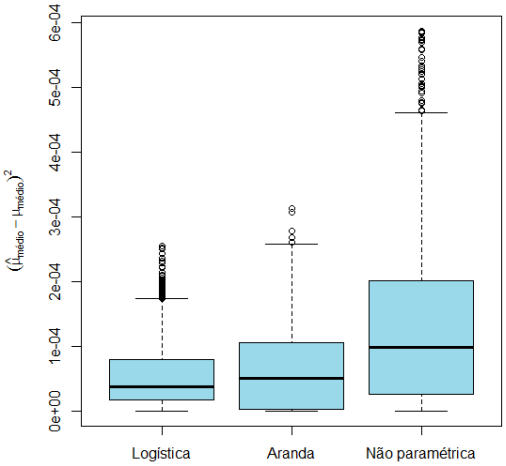

Figura 7.21: Distribuição do quadrado das distâncias das médias das probabilidades estimadas às médias das probabilidades geradas a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

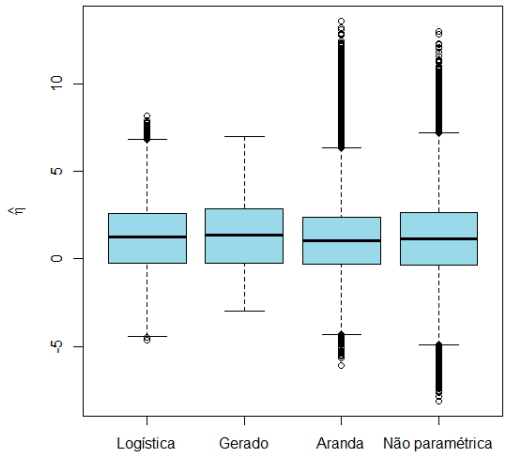

Figura 7.22: Distribuição das estimativas dos preditores lineares obtidas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

é a que mais se afasta do gerado.

Segue-se a análise das estimativas das funções parciais obtidas de cada uma das redes neuronais. Começando pela função $f_1(x_1) = x_1$, as médias das funções estimadas pelas GANNs podem ser visualizadas na fig. 7.24, onde se pode constatar pelos resultados obtidos pelas três GANNs são muito semelhantes.

No entanto, neste cenário, contra o que seria expectável, observou-se que a GANN Não paramétrica é a que apresenta a função parcial "média" com melhor desempenho como se pode observar pelo gráfico da fig. 7.25.

Na fig. 7.26 encontra-se representada a média das estimativas da função parcial f_2 obtidas por cada uma das GANNs em análise, a partir de dados simulados com base na função $f_2(x_2) = {x_2}^2$.

Como também se pode aqui observar, os resultados da GANN Logística são os que mais se aproximam da função gerada. Na fig. 7.27, pode-se observar a convergência das várias funções estimadas relativamente à função simulada.

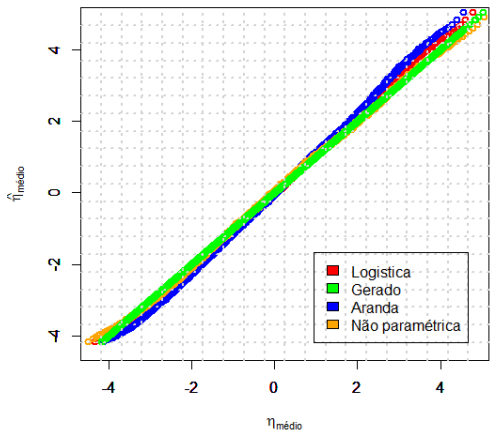

Figura 7.23: Gráfico de calibração das médias das estimativas do preditor linear, obtidas, a partir da GANN Logística, da GANN Aranda e da GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

Análise do desempenho preditivo e discriminativo

A partir da fig. 7.28 observa-se que as distribuições dos MSE obtidos a partir dos modelos em análise são equivalentes.

Por fim, na fig. 7.29, pode-se observar a distribuição das $AUCs$ para os vários modelos em análise, verificando-se também neste cenário, que todos os modelos apresentam distribuições semelhantes.

No entanto, é de salientar que, apesar deste cenário ser o mais favorável para a GANN Logística, as simulações demonstraram que as estimativas obtidas pelos modelos com função de ligação flexível, nomeadamente a GANN Não paramétrica, são igualmente muito satisfatórias.

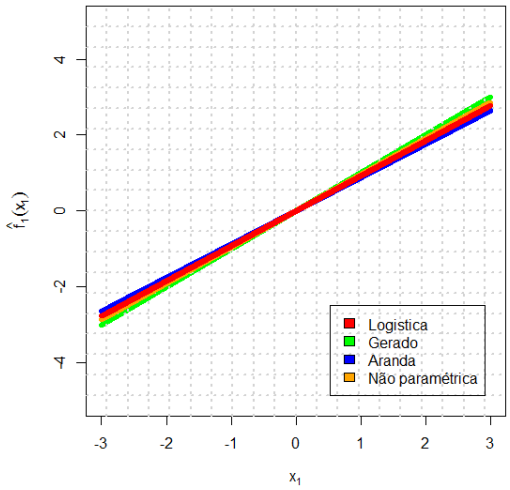

Figura 7.24: Médias obtidas a partir das estimativas das funções parciais da GANN Logística, GANN Aranda e GANN Não paramétrica, correspondente à função parcial $f_1(x_1) = x_1$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

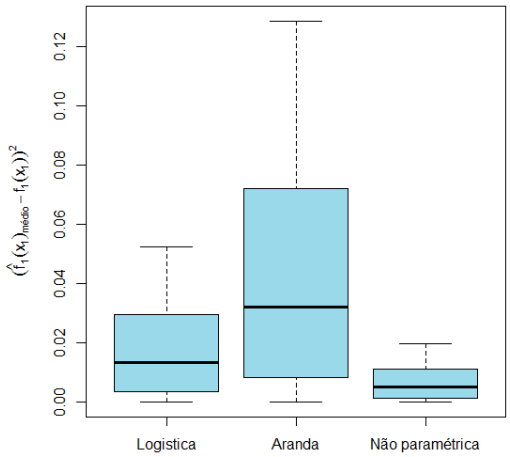

Figura 7.25: Distribuição do quadrado das distâncias das médias das estimativas das funções parciais relativamente à função parcial $f_1(x_1) = x_1$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

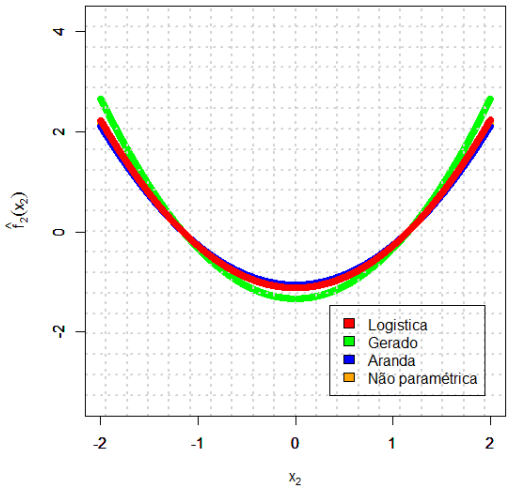

Figura 7.26: Médias obtidas a partir das estimativas das funções parciais da GANN Logística, GANN Aranda e GANN Não paramétrica, correspondente à função parcial $f_2(x_2) = {x_2}^2$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

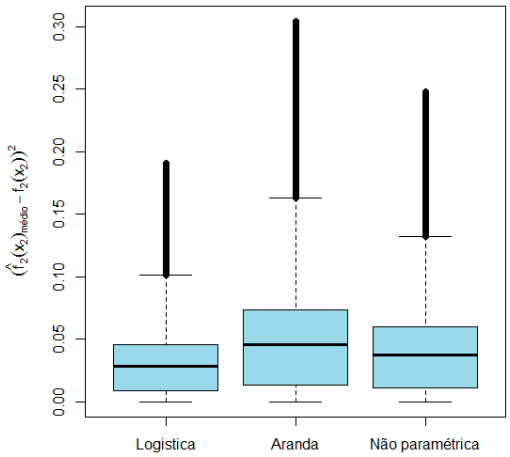

Figura 7.27: Distribuição do quadrado das distâncias das médias das estimativas das funções parciais relativamente à função parcial $f_2(x_2) = {x_2}^2$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

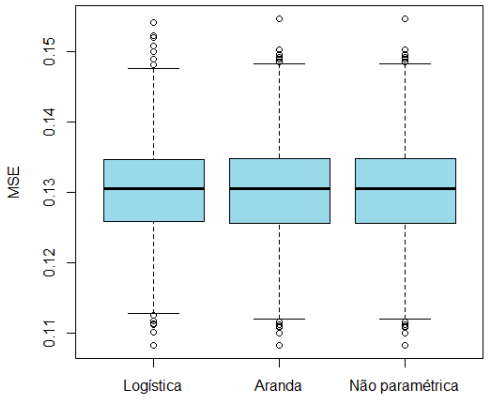

Figura 7.28: Distribuição do MSE da GANN Logística, GANN Aranda e GANN Não paramétrica para dados gerados a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

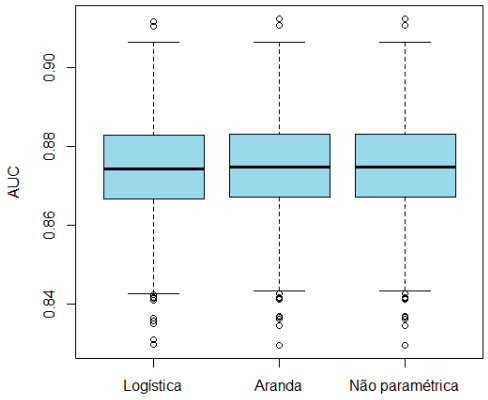

Figura 7.29: Distribuição das $AUCs$ da GANN Logística, GANN Aranda e GANN Não paramétrica para dados gerados a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = 0, \psi_1 = 1.0, \psi_2 = 1.0$.

7.3.2 Resultados para $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$

Nesta secção, foi utilizado um modelo para geração de dados com função de ligação pertencente à família de transformações Czado cujos parâmetros correspondem a um cenário que, à partida, não beneficia nenhuma das GANNs em estudo.

Começando pelas médias das funções de ligação estimadas, como se pode observar pela fig. 7.30, as que mais se aproximam à função de ligação do modelo simulado correspondem às GANNs com funções de ligação flexíveis.

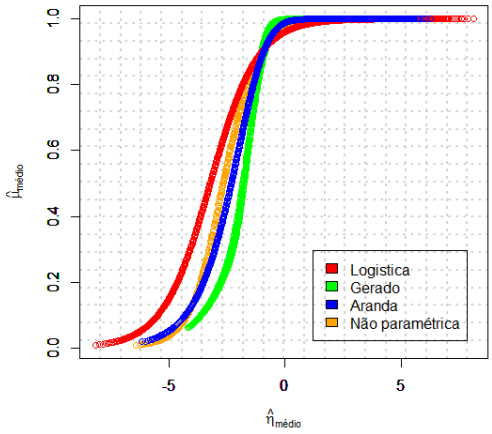

Figura 7.30: Médias das funções de ligação estimadas a partir de uma GANN Logística, GANN Aranda, GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

Na fig. 7.31 pode-se observar a distribuição das probabilidades estimadas a partir do modelo simulado. Verifica-se que as distribuições que mais se aproximam à do modelo gerado são as que correspondem às GANNs com função de ligação flexível.

O gráfico da fig. 7.32 vem confirmar o atrás observado, tendo-se registado que a distribuição das distâncias das médias das probabilidades estimadas pela GANN Logística às médias das geradas pelo modelo simulado é pior que a correspondente distribuição nos restantes modelos.

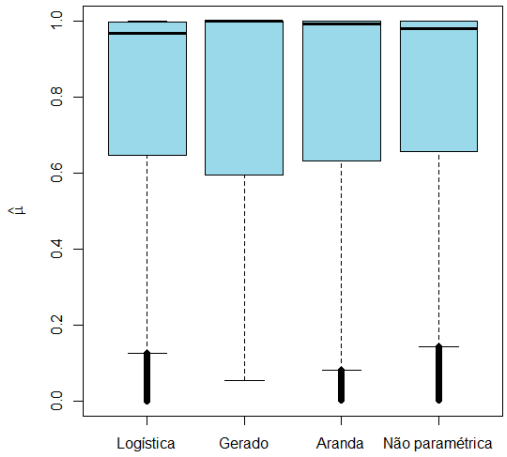

Figura 7.31: Distribuição das probabilidades estimadas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

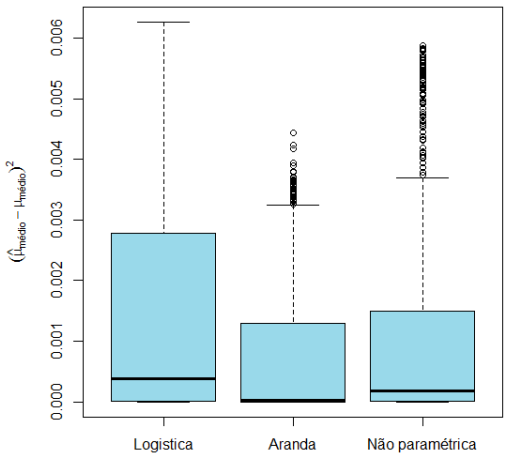

Figura 7.32: Distribuição do quadrado das distâncias das médias das probabilidades estimadas às médias das probabilidades geradas a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

Comparando-se a distribuição das estimativas dos preditores lineares de cada uma das GANNs com a distribuição do preditor linear do modelo simulado (ver fig. 7.33) verifica-se, também, neste caso, que a distribuição das estimativas do preditor linear obtida pela GANN Logística é a que mais diverge.

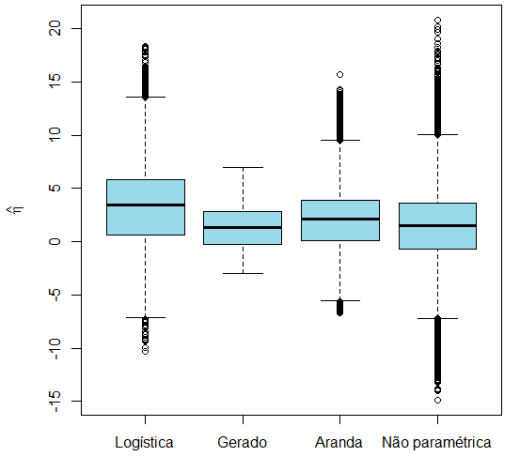

Figura 7.33: Distribuição das estimativas dos preditores lineares obtidas a partir da GANN Logística, GANN Aranda e GANN Não paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

Uma outra perspetiva pode ser obtida pela leitura do gráfico de calibração da fig. 7.34. Como se pode constatar, também neste caso, as GANNs com função de ligação flexível são as que se encontram mais próximas do gerado.

Ao analisarmos a média das estimativas da função parcial f_1 de cada uma das GANNs (ver fig. 7.35), observa-se que as funções estimadas pelas GANNs com função flexível, nomeadamente a estimada pela GANN Aranda, encontram-se mais próximas da gerada que a da GANN Logística.

Na fig. 7.36 pode-se visualizar noutra perspetiva, a proximidade das médias das estimativas das funções parciais relativamente à função gerada $f_1(x_1)$.

Na fig. 7.37 podem-se observar as estimativas das funções parciais para

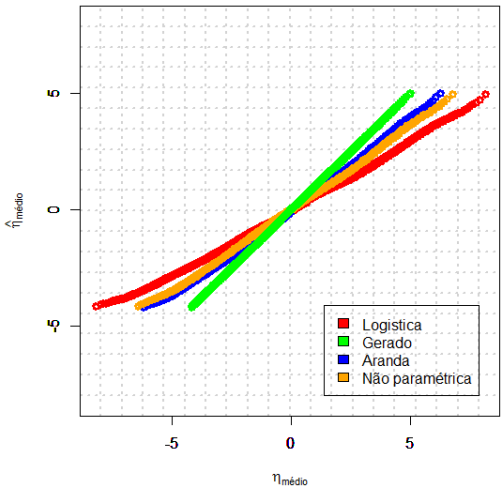

Figura 7.34: Gráfico de calibração das médias das estimativas do preditor linear, obtidas, a partir da GANN Logística, da GANN Aranda e da GANN Não Paramétrica, com base em dados gerados por um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

o caso da função gerada $f_2(x_2) = {x_2}^2$.

Também neste caso se observa uma maior proximidade das médias das funções estimadas através das GANNs com função de ligação flexível à gerada segundo o modelo simulado. De igual forma é de destacar a divergência da função estimada pela GANN Logística, como se pode observar na fig. 7.38.

Análise do desempenho preditivo e discriminativo

Na fig. 7.39 estão representadas as distribuições do MSE obtidas pelas GANNs em análise nas amostras de teste, com uma vantagem clara das GANNs com função de ligação flexível relativamente à GANN Logística.

Por fim, na fig. 7.40, pode-se observar a distribuição das $AUCs$ dos vários modelos em análise. Embora os três modelos tenham obtido valores muito semelhantes, as GANNs com função de ligação flexível apresentam medianas ligeiramente superiores à registada pela GANN Logística.

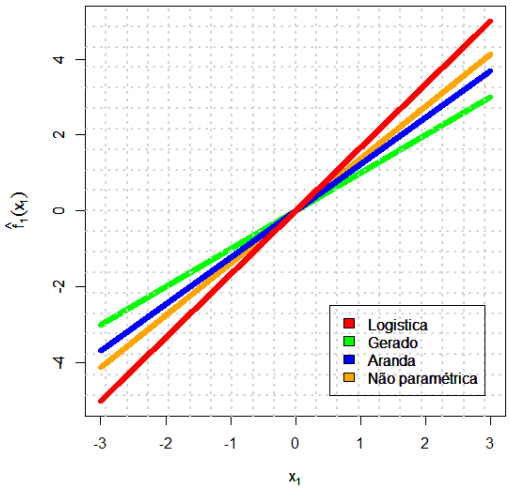

Figura 7.35: Média obtida a partir das estimativas das funções parciais da GANN Logística, GANN Aranda e GANN Não paramétrica, correspondente à função parcial $f_1(x_1) = x_1$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

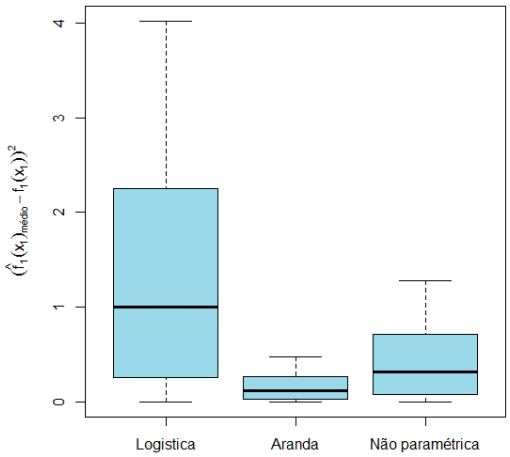

Figura 7.36: Distribuição do quadrado das distâncias das médias das estimativas das funções parciais relativamente à função parcial $f_1(x_1) = x_1$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

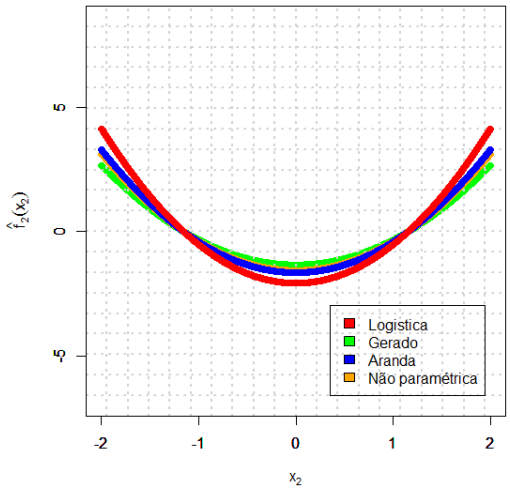

Figura 7.37: Média obtida a partir das estimativas das funções parciais da GANN Logística, GANN Aranda e GANN Não paramétrica, correspondente à função parcial $f_2(x_2) = {x_2}^2$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$..

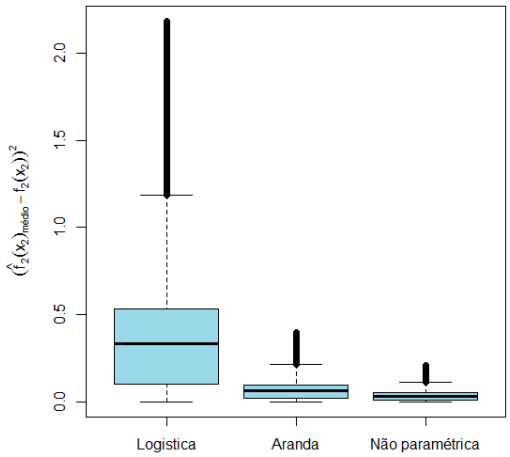

Figura 7.38: Distribuição do quadrado das distâncias das médias das estimativas das funções parciais relativamente à função parcial $f_2(x_2) = {x_2}^2$ do modelo simulado com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

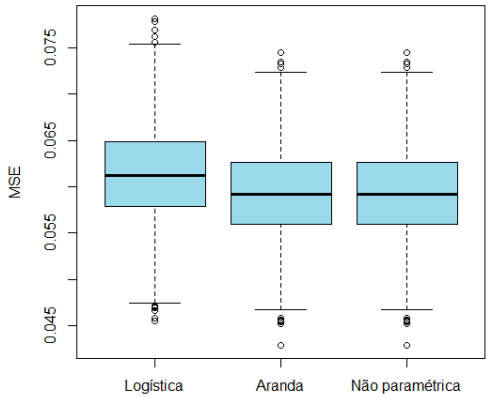

Figura 7.39: Distribuição do MSE obtido pela GANN Logística, GANN Aranda e GANN Não paramétrica com base em dados gerados a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

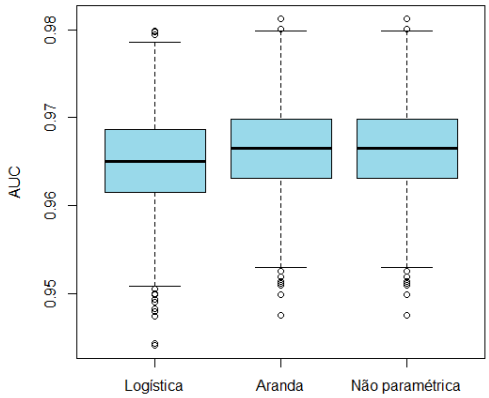

Figura 7.40: Distribuição das $AUCs$ da GANN logística, GANN Aranda e GANN não paramétrica para dados gerados a partir de um modelo com função de ligação pertencente à família de transformações Czado com $\eta_0 = -1, \psi_1 = 3.0, \psi_2 = 0.9$.

7.4 Discussão

Face ao observado, as simulações apontam no sentido de que uma GANN com função de ligação flexível, que também inclua a função logística na sua família de transformações, consegue obter melhores estimativas do que se utilizasse apenas a GANN com função de ligação logística definida *a priori*. Com efeito, a função de ligação flexível confere uma liberdade ao modelo de forma a permitir-lhe mais opções de ajustamento aos dados. Desta forma, o desempenho da GANN com função de ligação flexível consegue um desempenho quer preditivo, quer discriminativo igual ou superior à da GANN Logística.

Esta liberdade permite a obtenção de melhores estimativas do preditor linear e das funções parciais, conferindo assim uma interpretabilidade mais correta do modelo resultante.

Interessa também sublinhar que a utilização da função flexível se torna mais evidente quando a verdadeira forma funcional da função de ativação se afasta da função logística. De facto, quando foi simulado o modelo com função de ligação assimétrica pertencente à família de funções Czado com parâmetros $\eta_0 = -1, \psi_1 = 3.0$ e $\psi_2 = 0.9$, a GANN Logística obteve um menor desempenho relativamente aos restantes modelos.

No entanto, importa salientar que nos restantes cenários ficou demonstrada a robustez da função de ligação logística, nomeadamente nos casos em que a função de ligação "verdadeira" apresenta um declive pouco pronunciado (particularmente quando se utilizou a função logística e uma função pertencente à família de transformações de Aranda-Ordaz com $\psi = 1.5$).

A robustez da função de ligação logística foi também demonstrada no contexto dos GAMs (Papoila, 2006) corroborando, assim, o paralelismo existente entre as GANNs e os GAMs.

Capítulo 8

Rede Neuronal Aditiva Generalizada com função de ligação flexível: Razão de possibilidades

8.1 Introdução

Uma das áreas de conhecimento em que a análise e a modelação de dados são fundamentais é a Epidemiologia. Esta visa o estudo da distribuição bem como dos fatores determinantes da ocorrência de determinadas doenças em populações humanas específicas. Assim sendo, uma abordagem sistemática e imparcial da recolha, análise e interpretação dos dados é necessária. Os métodos utilizados em Epidemiologia baseiam-se na observação cuidadosa e na utilização de grupos de comparação válidos para avaliar associações, como por exemplo, verificar se o número de casos de doença numa determinada área, durante um período de tempo específico ou a frequência da exposição dos indivíduos a uma determinada doença, diferem do que se poderia esperar (Dicker et al., 2006).

De igual forma, há que salientar o facto da Epidemiologia procurar estabelecer relações causais com base em testes de hipóteses, em áreas científicas tais como a biologia e a medicina para explicar comportamentos relacionados com a saúde. No entanto, o escopo da Epidemiologia não se limita apenas a esta área da pesquisa, mas também fornece a base para orientar a ação prática e adequada das políticas de saúde pública (Dicker et al., 2006).

Os estudos epidemiológicos podem ser de natureza descritiva, ou analítica. No primeiro caso, o objetivo passa pela avaliação da frequência ou distribuição das doenças. No segundo caso, o enfoque dirige-se sobretudo ao estudo dos fatores que explicam tal distribuição. Para auxiliar a análise existem esquemas de estudos já delineados cuja aplicação depende, sobretudo, da forma como os doentes são observados, nomeadamente de uma forma retrospetiva, prospetiva ou transversal.

A partir desses estudos conseguem-se obter várias medidas, de entre as quais se destacam o risco relativo (RR) e a razão de possibilidades (OR). Neste capítulo será descrita a forma de estimar o OR a partir da utilização de uma Rede Neuronal Aditiva Generalizada (*Generalized Additive Neural Network* - GANN) com função de ligação flexível.

8.2 Estimação da razão de possibilidades

8.2.1 Definição

Em epidemiologia o OR é normalmente utilizado em estudos que se caracterizam pela procura da associação entre uma exposição e um efeito. No caso do efeito corresponder a uma doença, o OR traduz a relação entre a exposição a um fator específico e o desenvolvimento de uma doença (variável de natureza dicotómica). Neste contexto, o OR representa a possibilidade da ocorrência de doença, dada a exposição a um determinado fator X, em comparação com a possibilidade da ocorrência da mesma doença dada a ausência de exposição ao mesmo determinante X.

Assim, representando por $\pi_{dx} = P(Y = 1|X = x)$ a probabilidade de ocorrência da doença, dada a exposição/não exposição a um determinado fator X, a possibilidade de doença é dada por:

$$\Omega_{dx} = \frac{\pi_{dx}}{1 - \pi_{dx}}. \tag{8.1}$$

Tomando, como exemplo, o delineamento de um estudo típico de caso-controlo, como o especificado na fig. 8.1, em que são selecionados dois grupos de indivíduos, um composto pelos que contraíram a doença - os casos, e outro pelos que não possuem a doença - os controlos, então, a probabilidade de doença no grupo dos expostos será dada por $\pi_{de} = P(Y = 1|X = 1)$ e a probabilidade de doença nos dos não expostos será dada por $\pi_{dne} = P(Y = 1|X = 0)$. Desta forma, o OR pode ser representada pela seguinte equação:

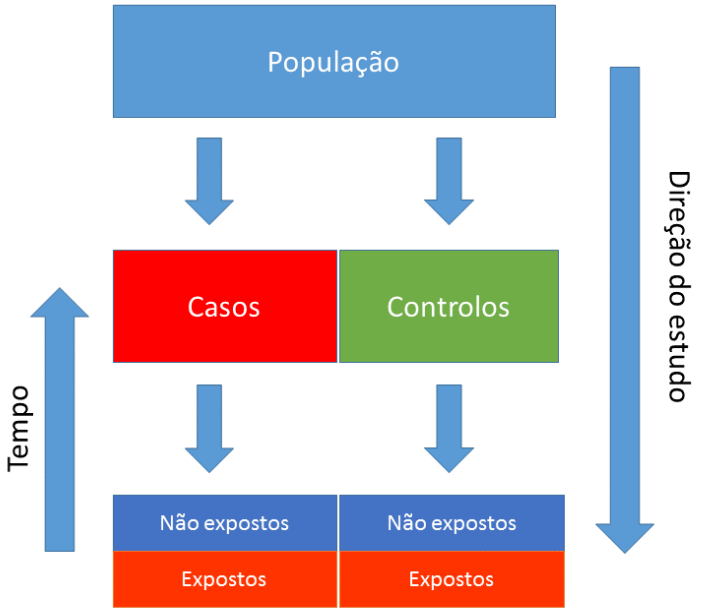

Figura 8.1: Estudo de caso-controlo.

$$OR = \frac{\Omega_{de}}{\Omega_{dne}} = \frac{\pi_{de}/(1-\pi_{de})}{\pi_{dne}/(1-\pi_{dne})}, \tag{8.2}$$

em que Ω_{de} representa a possibilidade de doença nos indivíduos expostos e Ω_{dne} representa a possibilidade de doença nos indivíduos não expostos.

8.2.2 Estimação da razão de possibilidades para uma variável explicativa binária

Tendo em conta um determinante de natureza binária X_k, poder-se-á deduzir o respetivo OR a partir do modelo de regressão $E[Y|\mathbf{X}] = h(\eta) = h(\beta_0 + \sum_{j=1}^{p} f_j(X_j))$, em que $h(.)$ representa uma função de ligação genérica e η o preditor linear do modelo com $j = 1, ..., p$.

Comecemos por considerar:

$$\begin{cases} \pi_{de} = P(Y = 1|\mathbf{X} = (X_1, ..., X_k = 1, ..., X_p)), \\ \pi_{dne} = P(Y = 1|\mathbf{X} = (X_1, ..., X_k = 0, ..., X_p)). \end{cases} \tag{8.3}$$

Por outro lado, as probabilidades de desenvolvimento da doença dada a exposição/não exposição ao fator k, podem ser expressas pelas seguintes equações:

$$\begin{cases} \pi_{de} = h(\eta_1) = h(\beta_0 + f_1(x_1) + ... + f_k(1) + ... + f_p(x_p)), \\ \pi_{dne} = h(\eta_0) = h(\beta_0 + f_1(x_1) + ... + f_k(0) + ... + f_p(x_p)). \end{cases} \tag{8.4}$$

Assim, partindo da eq. 8.2, consegue-se determinar uma estimativa do OR correspondente a uma exposição binária através da seguinte expressão:

$$OR_k = E_{\mathbf{X}}\left[\frac{\pi_{de}/(1-\pi_{de})}{\pi_{ne}/(1-\pi_{ne})}\right] = E_{\mathbf{X}}\left[\frac{h(\eta_1)/(1-h(\eta_1))}{h(\eta_0)/(1-h(\eta_0))}\right], \tag{8.5}$$

em que $E_{\mathbf{X}}$ representa o operador valor médio sobre as variáveis $\{X_j\}_{j\neq k}$. Desta forma, pode-se considerar a eq. 8.5 como uma expressão genérica para a estimação da razão de possibilidades correspondente a uma exposição binária arbitrária.

Contudo, em alguns casos particulares, nomeadamente naqueles em que a função de ligação corresponde a uma logística, a expressão 8.5 pode ser desenvolvida analiticamente. Com efeito, no caso da função de ligação corresponder a $h(\eta) = \frac{1}{1+e^{-\eta}}$, a equação anterior resulta na seguinte expressão:

$$OR_k = E_{\mathbf{X}}[e^{(\eta_1-\eta_0)}]. \tag{8.6}$$

Uma vez que $e^{(\eta_1-\eta_0)} = e^{(f_k(1)-f_k(0))} = e^{\beta_k} \Rightarrow OR_k = e^{\beta_k}$, verifica-se assim que o OR resulta numa constante. Esta solução é compatível com o facto da função parcial $f_k(X_k)$ poder ser linear, ($f_k = \beta_k X_k$), dado que a variável X_k é binária.

Transposto para uma GANN logística, o Perceptrão Multicamada (*Multi Layer Perceptron* - MLP) correspondente à função parcial $f_k(X_k)$ é constituído apenas por uma *skip layer*, tendo em conta a natureza dicotómica da

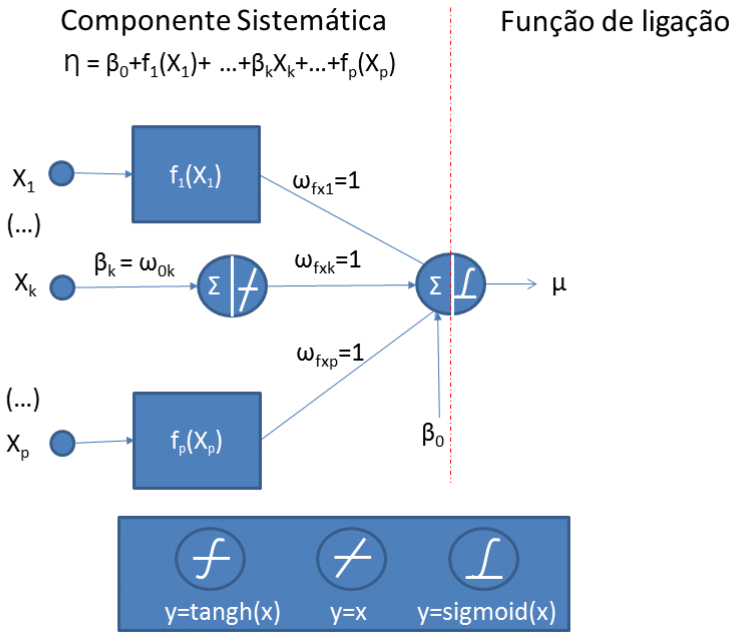

Figura 8.2: Arquitetura de uma GANN logística com variável de exposição X_k e resposta de natureza binária.

variável X_k, como se pode observar pela fig. 8.2.

Para este caso particular, a estimativa do OR_k corresponde ao valor estimado do peso sináptico ω_{0k}, depois de adaptada a GANN Logística.

Para outros tipos de função de ligação, nomeadamente de natureza flexível, onde se encontram incluídas as funções de ligação Aranda e a Não paramétrica (abordadas nos capítulos anteriores), a forma de estimação do OR pode não resultar numa forma analítica de fácil resolução.

Nestes casos, o problema pode ser resolvido através da utilização de um algoritmo proposto por Cadarso-Suárez et al. (2005). Embora este algoritmo tenha sido desenvolvido para a estimação da razão de possibilidades correspondente a uma variável de exposição X_k contínua, também se pode aplicar no contexto de uma variável binária.

8.2.3 Estimação da função razão de possibilidades para uma variável explicativa contínua

Dada a exiguidade de teoria sobre a GANN, à semelhança do que temos vindo a fazer ao longo deste estudo, para estimar o OR baseámo-nos numa proposta da área da estatística (Cadarso-Suárez et al., 2005), onde é implementada uma generalização do OR no caso de um Modelo Aditivo Generalizado (*Generalized Additive Model* - GAM) com função de ligação flexível não paramétrica. Neste algoritmo, o OR aplicado à k-ésima covariável X_k, num ponto genérico x, tendo em conta um valor de referência $x_0 \in [min(x_k), máx(x_k)]$, ajustado pelas restantes covariáveis, resulta na seguinte expressão:

$$OR_k^{x_0}(x) = E_{\mathbf{X}}\Big[\frac{\pi(X_1,...,x,...,X_p)/(1-\pi(X_1,...,x,...,X_p))}{\pi(X_1,...,x_0,...,X_p)/(1-\pi(X_1,...,x_0,...,X_p))}\Big]. \quad (8.7)$$

Desta forma, o estimador é dado por:

$$\widehat{OR}_k^{x_0}(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{\hat{\pi}(X_{i1},...,x,...,X_{ip})/(1-\hat{\pi}(X_{i1},...,x,...,X_{ip}))}{\hat{\pi}(X_{i1},...,x_0,...,X_{ip})/(1-\hat{\pi}(X_{i1},...,x_0,...,X_{ip}))}. \quad (8.8)$$

A equação anterior pode ser desenvolvida de forma a obter a seguinte expressão:

$$\widehat{OR}_k^{x_0}(x) = \frac{1}{n}\sum_{j=1}^{n}\frac{\hat{h}(\hat{\beta}_0+\hat{f}_1(X_{i1})+...+\hat{f}_k(x_0)+...+\hat{f}_p(X_{ip}))^{-1}-1}{\hat{h}(\hat{\beta}_0+\hat{f}_1(X_{i1})+...+\hat{f}_k(x)+...+\hat{f}_p(X_{ip}))^{-1}-1}. \quad (8.9)$$

A partir das equações 8.7 e 8.8 foi desenvolvido o algoritmo para a estimação do OR que se encontra descrito no algoritmo 10.

O procedimento para a estimação do $OR_k^{x_0}(x)$ começa pela definição do número de valores de x (pontos), pertencentes ao intervalo $[min(x_k), máx(x_k)]$, em que se pretende estimar esta função, e também pela definição do valor de referência x_0 pertencente ao mesmo intervalo. Através do número de valores de x desejado e fixado *a priori*, determina-se o incremento a utilizar para obter os referidos valores: $\{min(x_k), min(x_k)+inc, min(x_k)+$

Entrada: k,#pontos , $(y_i, x_{i1}, ..., x_{ip})$ em que $i = \{1, ..., n\}$

início
- Definir inc := $\frac{max(x_{ik}) - min(x_{ik})}{\#pontos}$;
- Definir valor de referência x_0;
- **para cada** *indivíduo i na variável k* **faça**
  - $x_{ik} = x_0$;
  - Obter as estimativas de π_{i0};
- **fim**
- $x = min(x_{ik})$;
- or := [];
- **repita**
  - **para cada** *indivíduo i na variável k* **faça**
    - $x_{ik} = x$;
    - Obter as estimativas de π_{ix};
    - Estimar $\widehat{OR}_k^{x_0}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{\hat{\pi}(x_{i1},...,x,...,x_{ip})/(1-\hat{\pi}(x_{i1},...,x,...,x_{ip}))}{\hat{\pi}(x_{i1},...,x_0,...,x_{ip})/(1-\hat{\pi}(x_{i1},...,x_0,...,x_{ip}))}$
  - **fim**
  - $x = x + inc$;
  - Adiciona $\widehat{OR}_k^{x_0}(x)$ ao conjunto or;
- **até** $x > max(x_{ik})$;
- **retorna** *or;*

fim

Algoritmo 10: Algoritmo para a estimação da função razão de possibilidades no caso de uma variável explicativa contínua X_k (Cadarso-Suárez et al., 2005).

$2inc, ..., máx(x_k)\}$.

Adicionalmente, com base no vetor $(Y, X_1, ..., X_p)$, fixa-se a variável X_k no valor x_0 para todos os indivíduos obtendo a correspondente probabilidade $\hat{\pi}_{i0}$. O mesmo procedimento é realizado para cada ponto do intervalo $x \in [min(x_k), máx(x_k)]$, de modo a obter os respetivos $\hat{\pi}_{ix}$.

Seguidamente, iremos aplicar este algoritmo a um caso com dados simulados, à semelhança do que foi realizado no capítulo anterior.

Estimação da função razão de possibilidades para uma variável explicativa contínua com dados simulados

O algoritmo para estimar a função $OR(X_k)$ de uma das variáveis explicativas contínuas X_k, foi implementado com uma GANN Aranda, tendo por base os 1000 modelos com função de ligação logística simulados no capítulo 7, para um valor de referência $x_0 = 0$. Por cada modelo, foram estimadas as funções OR com base em 100 valores de x_1 pertencentes ao intervalo $[-3, 3]$ no caso da primeira covariável, e o mesmo número de valores de x_2 pertencentes ao intervalo $[-2, 2]$ no caso da segunda covariável. Os 1000 modelos simulados têm por base dois cenários:

- os termos do preditor linear $\eta = f_1(X_1) + f_2(X_2)$, são definidos por $f_1(X_1) = X_1$ e $f_2(X_2) = {X_2}^2$;
- os termos do preditor linear $\eta = f_3(X_3) + f_4(X_4)$, são definidos por $f_3(X_3) = 2sen(0.5\pi(X_3 + 2))$ e $f_4(X_4) = {X_4}^2$.

Para facilitar a leitura dos ORs, pode-se recorrer à representação na escala logarítmica das suas estimativas (Cadarso-Suárez et al., 2005). Uma vez que os dados simulados foram gerados a partir de um modelo com função de ligação logística, é expectável que a média das estimativas de $log(\widehat{OR}_j^0(x))$ seja aproximada à correspondente função parcial gerada $f_j(x)$, $j = 1, ..., p$, tal como se pode observar nas figs. 8.3, 8.4 e 8.5.

8.3 Discussão

Ao longo deste capítulo ficaram patentes algumas das potencialidades da GANN com função de ligação paramétrica flexível na área da epidemiologia. Para tal, houve que adaptar a teoria existente na área dos GAMs, resultando

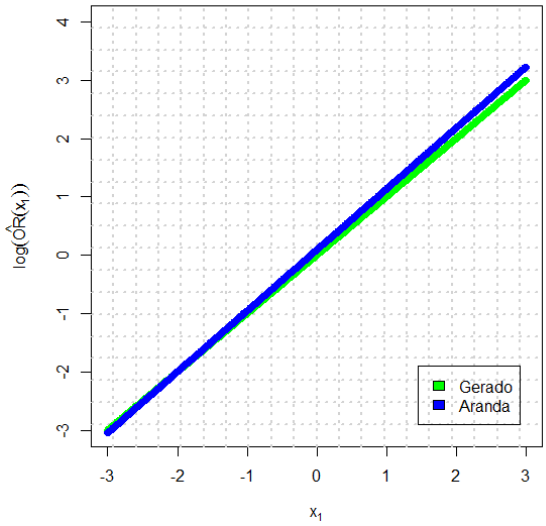

Figura 8.3: Médias de $log(\widehat{OR}(x_1))]$ obtidas a partir da GANN Aranda (primeiro cenário).

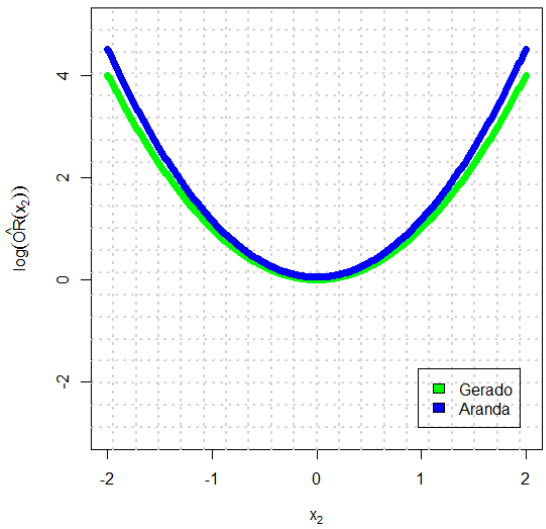

Figura 8.4: Médias de $log(\widehat{OR}(x_2))]$ obtidas a partir da GANN Aranda (primeiro cenário).

na implementação dos algoritmos descritos através das GANN Aranda e, em particular, através da GANN Logística.

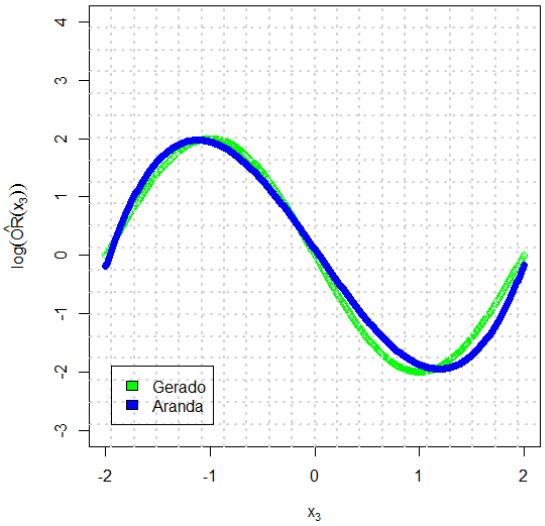

Figura 8.5: Médias de $log(\widehat{OR}(x_3))$] obtidas a partir da GANN Aranda (segundo cenário).

Por outro lado, a grande vantagem que a GANN oferece face a outro tipo de redes neuronais, nomeadamente ao MLP, encontra-se bem patente, através da possibilidade em estimar a função de OR de uma variável contínua. A já sobejamente referida opacidade das relações entre as variáveis explicativas e a variável de interesse no caso dos MLPs tem desmotivado a utilização das redes neuronais na área da Medicina face a outros modelos, nomeadamente aos Modelos Lineares Generalizados (*Generalized Linear Model* - GLMs) e aos GAMs. Esta desmotivação torna-se mais evidente nas situações onde é importante não apenas a obtenção das estimativas como também a interpretação do modelo, tal como ocorre usualmente na área da epidemiologia (Tu, 1996).

A GANN, ao permitir a interpretação da relação entre as covariáveis e a resposta, propicia que este tipo de arquitetura possa vir a ser utilizada com maior frequência na área da investigação clínica.

Capítulo 9

Aplicação a um caso real

9.1 Introdução

Nesta aplicação foi utilizada uma amostra recolhida numa Unidade de Cuidados Intensivos (UCI), respeitante a 996 doentes, tendo sido registadas as seguintes variáveis:

- X_{pas} - pressão arterial sistólica;
- X_{frc} - frequência cardíaca;
- X_K - potássio sérico;
- X_{vent} - doente ventilado.

Com esta informação recolhida à data de admissão, pretende-se estimar a probabilidade de morte dos doentes à entrada na UCI.

A variável resposta Y, diz respeito ao evento *morte*, durante o internamento na UCI, tendo-se registado que, dos 996 pacientes, 359 faleceram.

Na fase de pré-processamento, os dados das covariáveis contínuas foram normalizados para valores entre -1 e 1.

Para a codificação da arquitetura da Rede Neuronal Aditiva Generalizada (*Generalized Additive Neural Network* - GANN) utilizada para modelar as relações entre as covariáveis acima descritas e a resposta, utilizou-se o esquema descrito na Tabela 5.2. A seleção da melhor GANN Aranda foi efetuada a

partir de uma grelha elaborada com vários valores de ψ.

Numa primeira fase, utilizou-se um espaçamento entre parâmetros $\Delta\psi = 1$ (*i.e.* $\psi \in \{0, 1, ..., 20\}$). Após a identificação do valor de ψ que minimiza o Erro Quadrático Médio (*Mean Squared Error* - MSE), foi utilizada uma grelha mais fina em torno dessa estimativa com $\Delta\psi = 0.1$.

Para a seleção da melhor GANN Não paramétrica, elaborou-se também uma grelha mas, neste caso correspondendo a diferentes números de neurónios na camada escondida do Perceptrão Multicamada (*Multi Layer Perceptron* - MLP) que define a função de ligação desta arquitetura.

9.2 Definição do modelo e resultados

Relativamente ao conjunto de dados recolhido na *ICU* anteriormente referida, os modelos a estimar baseiam-se no seguinte preditor linear:

$$\eta = \beta_0 + f_{pas}(X_{pas}) + f_{frc}(X_{frc}) + f_K(X_K) + \beta_{vent}X_{vent}. \qquad (9.1)$$

No caso da GANN Aranda, como já referido anteriormente, a estimação do modelo para vários valores de $\boldsymbol{\psi}$ fundamenta-se na expressão 6.7. No caso da GANN Não paramétrica, a eq. 6.3 serve de base para a estimação deste modelo.

Como também já anteriormente referido (ver secção 6.4), verificou-se empiricamente que a função de ligação da GANN Aranda é uma boa indicadora de qual a função de ativação do neurónio de saída a utilizar pela GANN Não paramétrica. Assim sendo, estimar-se-á primeiramente a função de ligação da GANN Aranda. Utilizou-se, então, a grelha $\{0, 1, ..., 20\}$ referida na secção anterior, tendo-se verificado que o melhor modelo se encontrava na zona em que $\psi < 2.0$, como se pode constatar pela fig. 9.1.

Continuando com a estimação da função de ligação da GANN Aranda, após a identificação da zona onde se poderá encontrar o melhor valor para o parâmetro ψ, utilizou-se uma grelha mais fina ($\Delta\psi = 0.1$). Tendo-se obtido o gráfico da fig. 9.2, observa-se que a melhor estimativa corresponde a uma GANN Aranda com $\psi = 0.1$.

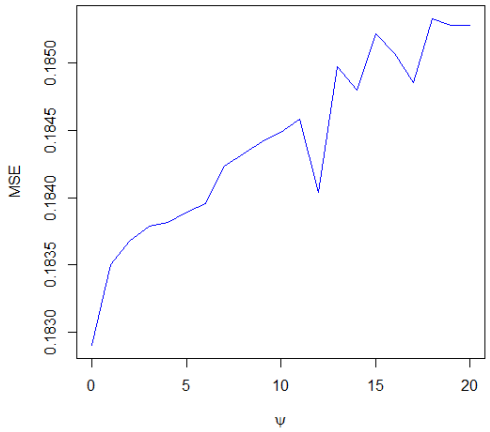

Figura 9.1: Gráfico de perfil com base no *MSE*, obtido a partir da amostra de teste para a grelha $\{0, 1, ..., 20\}$ $(\Delta\psi = 1)$.

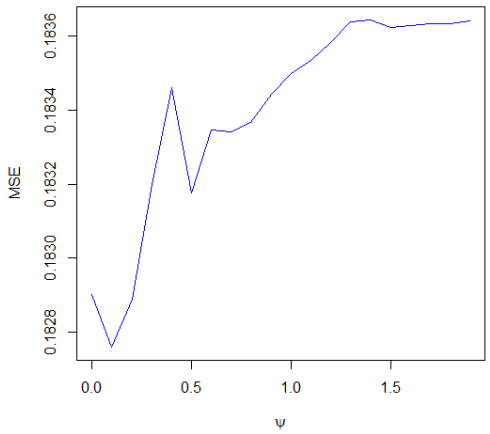

Figura 9.2: Gráfico de perfil obtido com base no MSE obtido a partir da amostra de teste para a grelha $\{0, 0.1, ..., 2.0\}$ $(\Delta\psi = 0.1)$.

No que diz respeito à função de ligação da GANN Não paramétrica, utilizou-se a estimativa de $\psi = 2.0$ obtido na primeira fase da estimação de ψ pela GANN Aranda.

Procedeu-se, ainda à estimação, do modelo através da GANN Logística, tendo sido quantificado o desempenho preditivo de qualquer um destes mo-

Tabela 9.1: Medidas do desempenho preditivo e discriminativo das GANNs em análise.

GANN	MSE	AUC	I.C. 95%
Logística	0.1885	0.7376	0.7044-0.7709
Aranda	0.1848	0.7445	0.7116-0.7775
Não paramétrica	0.1883	0.7395	0.7064-0.7727

delos através do MSE obtido a partir da amostra de teste. De igual forma, obteve-se o poder discriminativo de cada modelo com recurso à respetiva área sob a *Receiver Operating Characteristic (ROC) curve* (AUC).

Assim, na tabela 9.1 encontram-se resumidos os desempenhos atrás referidos. Na fig. 9.3 podem-se observar as curvas ROC dos resultados obtidos, respetivamente pela GANN Logística, GANN Aranda e GANN Não paramétrica.

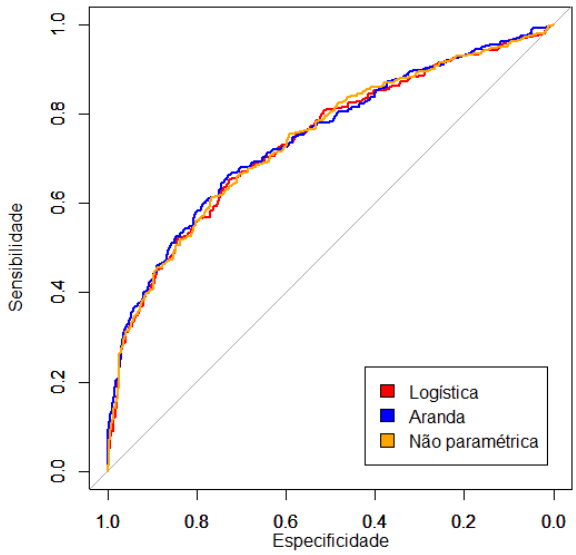

Figura 9.3: Curvas ROC da GANN Logística, GANN Aranda e GANN Não paramétrica.

Na fig. 9.4 podem-se observar as médias das funções de ligação estimadas a partir dos vários modelos em análise. Para este conjunto de dados, com base nos resultados anteriores, verificou-se que o desempenho das três

GANNs é muito semelhante, mostrando para a GANN Aranda uma ligeira superioridade.

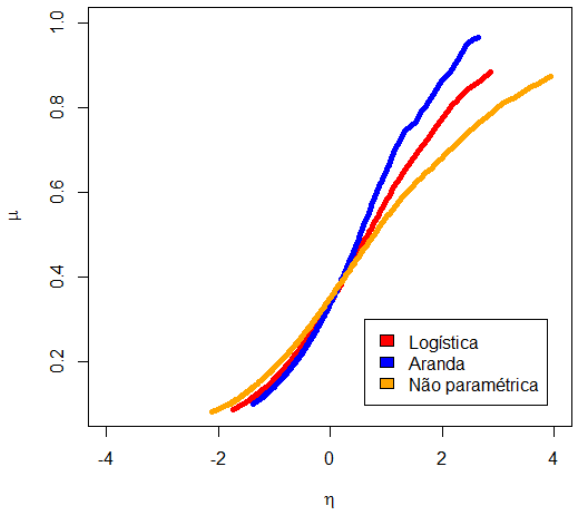

Figura 9.4: Médias das funções de ligação estimadas a partir da GANN Logística, da GANN Aranda e da GANN Não paramétrica.

Verifica-se assim, que a GANN Aranda apresenta estimativas ligeiramente melhores do que as dos restantes modelos.

Seguidamente, proceder-se-á a uma análise comparativa dos desempenhos de cada uma das redes neuronais com os modelos de regressão habitualmente utilizados na área clínica, nomeadamente, o Modelo Aditivo Generalizado (*Generalized Additive Model* - GAM) e o Modelo Linear Generalizado (*Generalized Linear Model* - GLM),

9.3 Análise comparativa com os modelos de regressão GAM e GLM

Para uma análise comparativa com os métodos habitualmente utilizados na área da investigação clínica, utilizaram-se como referência os modelos de regressão paralelos à GANN (GAMs e GLMs). Para uma escolha mais alargada de opções, incluiu-se em ambos os modelos de regressão, a função de liga-

Tabela 9.2: Medidas do desempenho preditivo e discriminativo dos modelos GLM e GAM.

Modelo	MSE	AUC	95% I.C.
GAM Aranda	0.1844	0.7443	0.7115-0.7770
GAM Logístico	0.1848	0.7440	0.7111-0.7768
GLM Aranda	0.1900	0.7327	0.6990-0.7664
GLM Logístico	0.1931	0.7309	0.6972-0.7646

ção flexível pertencente à família de transformações assimétricas de Aranda-Ordaz. O processo de estimação da função de ligação de cada um dos modelos de regressão baseou-se numa grelha idêntica à utilizada para a estimação da GANN Aranda, tendo-se obtido $\psi = 4.3$ para a função de ligação do GAM Aranda e $\psi = 0.0$ para a função de ligação do GLM Aranda. Os resultados respeitantes ao desempenho de cada uma destes modelos encontram-se descritos na tabela 9.2.

O modelo de regressão que registou o pior desempenho foi o GLM Logístico (sendo, contudo, o mais utilizado a nível da investigação clínica). O GAM Aranda apresentou o melhor desempenho. Assim, com base nos resultados destes dois modelos, pode-se construir uma escala de posição relativa do desempenho preditivo, baseada no MSE de acordo com a seguinte expressão:

$$\Psi_{MSE} = \frac{MSE_{modelo} - MSE_{glm}}{MSE_{gamAranda} - MSE_{glm}}, \tag{9.2}$$

em que Ψ_{MSE} representa a posição relativa de um determinado modelo. MSE_{glm} diz respeito ao MSE obtido pelo GLM Logístico e $MSE_{gamAranda}$ corresponde ao valor do MSE obtido pelo GAM Aranda.

De igual forma, procedeu-se à construção de uma escala de posição relativa do desempenho discriminativo de acordo com a seguinte expressão:

$$\Psi_{AUC} = \frac{AUC_{modelo} - AUC_{glm}}{AUC_{gamAranda} - AUC_{glm}}, \tag{9.3}$$

em que Ψ_{MSE} representa a posição relativa de um determinado modelo. AUC_{glm} diz respeito à AUC obtida pelo GLM Logístico e $AUC_{gamAranda}$ cor-

Tabela 9.3: Medidas do desempenho preditivo e discriminativo normalizadas.

Modelo	Ψ_{MSE}	Ψ_{AUC}
GAM Aranda	1.0	1.0
GLM Logística	0.0	0.0
GANN Aranda	0.95	1.02
GANN Logística	0.52	0.50
GANN Não paramétrica	0.55	0.65

responde à AUC obtida pelo GAM Aranda.

Estas métricas permitem analisar até que ponto os respetivos desempenhos de uma determinada GANN se encontram próximos ou afastados dos modelos de referência. Assim, no caso do valor de Ψ ser superior a 1, isto significa que o modelo em estudo excedeu o desempenho do GAM Aranda. No caso de ser inferior a 0 significa que o modelo em estudo possui pior desempenho que o GLM Logístico.

Os resultados destas medidas encontram-se resumidos na tabela 9.3.

Como se pode constatar, os desempenhos da GANN Aranda encontram-se muito próximos do GAM Aranda, ao passo que os desempenhos relativos da GANN Logística e da GANN Não paramétrica registaram valores mais baixos.

Seguidamente, procedeu-se a uma análise da associação de cada covariável com a resposta, recorrendo à GANN Aranda em estudo.

9.4 Análise das funções parciais e da função de razão de possibilidades

Nesta análise foram estimadas as funções parciais e as respetivas funções de OR com base num valor de referência, tendo sido considerada a mediana dos valores da amostra original. Adicionalmente, foram obtidos os respetivos intervalos de confiança a 95%, recorrendo ao método de *bootstrap* descrito na secção 4.6.

9.4.1 Pressão arterial sistólica

A pressão arterial sistólica (PAS), medida em milímetros de mercúrio ($mmHg$), representa a intensidade do fluxo de sangue que passa pelas artérias e que se traduz pela pressão do sangue no momento da sístole cardíaca. A contração do coração impulsiona o sangue contra as paredes das artérias, de forma a que uma maior contração corresponde a um aumento da pressão sistólica.

Na fig. 9.5 encontra-se representada a associação (com os respetivos intervalos de confiança) entre a PAS e o evento *morte*.

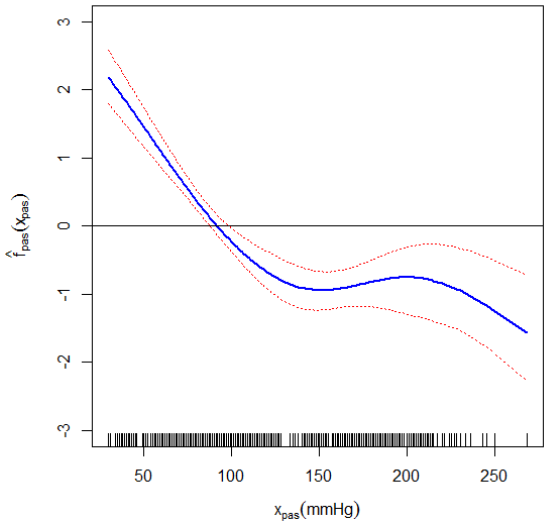

Figura 9.5: Estimativa da função parcial e respetivo intervalo de confiança a 95%, obtidos a partir de uma GANN Aranda, traduzindo a associação entre a pressão arterial sistólica (unidade expressa em mm_g) e o evento *morte*.

Como se pode observar, existe um maior risco de morte para valores baixos deste determinante. Este facto está de acordo com o que se regista em termos gerais no contexto de uma UCI, em que, normalmente, se verifica maior dificuldade na estabilização de doentes com valores de PAS muito baixos.

Para determinar a função OR considerou-se como valor de referência $x_0 = 97\ mm_g$ que corresponde à mediana desta variável (fig. 9.6).

Como se pode observar, doentes com pressão arterial inferior à mediana tem uma possibilidade de morte superior.

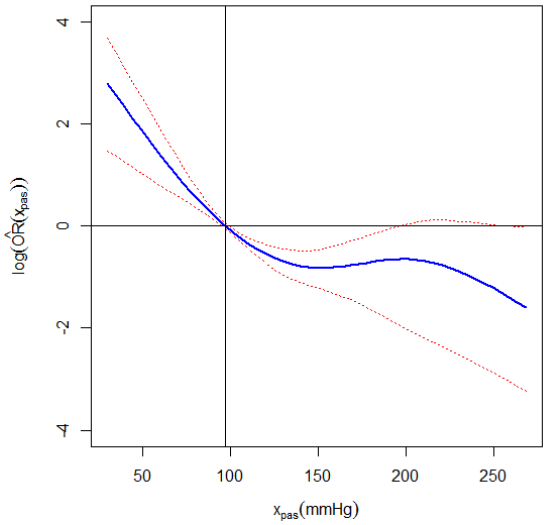

Figura 9.6: Estimativa da função OR e respetivo intervalo de confiança a 95%, correspondentes à pressão arterial sistólica em escala logarítmica, obtidos a partir de uma GANN Aranda. A linha vertical representa a mediana da pressão arterial sistólica.

9.4.2 Frequência cardíaca

A frequência cardíaca traduz o número de batimentos por minuto. Na fig. 9.7 encontra-se a estimativa da função parcial que ilustra a associação entre a covariável e a morte. Como se pode observar, para valores muito baixos, o risco de morte aumenta. O valor mais protetor situa-se em 74 batidas por minuto (BPM).

Relativamente à função OR, esta pode ser observada na fig. 9.8. Neste caso, o valor de referência, correspondente à mediana dos valores observados de X_{frc}, é $x_0 = 117\ BPM$.

9.4.3 Potássio sérico

O níveis de potássio são mantidos em valores normais através dos rins. Assim, uma redução até valores demasiado baixos pode-se dever a um funcionamento

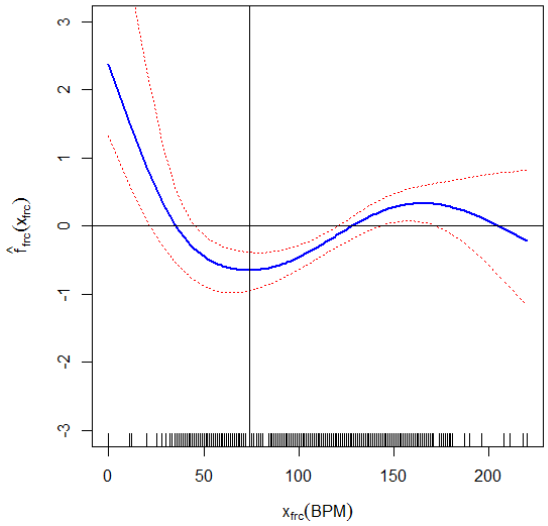

Figura 9.7: Estimativa da função parcial e respetivo intervalo de confiança a 95%, obtidos a partir de uma GANN Aranda, traduzindo a associação entre a frequência cardíaca (número de batimentos por minuto - BPM) e o evento *morte*. A linha vertical representa o mínimo desta função ($x_{frc} = 74\ BPM$).

anormal destes órgãos, ou a uma perda através do aparelho digestivo (devido a vómitos ou diarreia). Valores excessivamente baixos conduzem à hipocaliemia aumentando o risco de morte, embora sem significado estatístico (uma vez que o intervalo de confiança contém o valor 0) como se pode observar pela fig. 9.9.

Valores altos de potássio que, entre vários fatores, também podem ser causados por uma anomalia renal, conduzem à hipercaliemia. Como igualmente se pode observar pela fig. 9.9, valores de potássio superiores a 4.0 encontram-se associados a um aumento do risco de morte, embora só com significado estatístico a partir de $5.1 mmol/l$.

Valores entre 3.40 e 4.00 parecem ser protetores no que diz respeito à morte.

Quanto à função OR, esta pode ser observada na fig. 9.10, estimada com base no valor de referência mediano $x_0 = 3.6$. Uma vez que este valor se encontra próximo do valor mais protetor $x_0 = 3.7$, qualquer valor para a esquerda ou para a direita, conduz a um aumento do risco de mortalidade

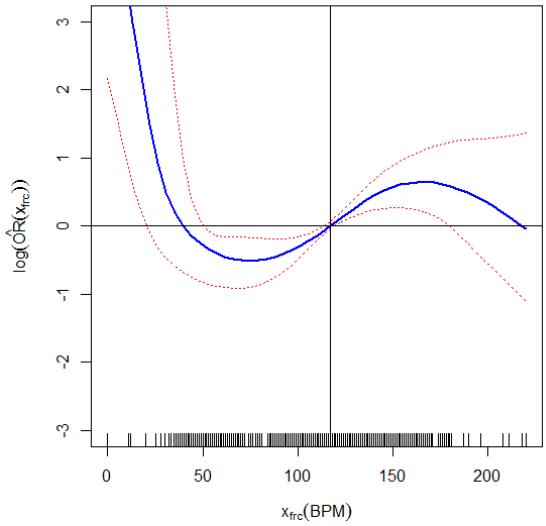

Figura 9.8: Estimativa da função OR e respetivo intervalo de confiança a 95% correspondentes à frequência cardíaca, em escala logarítmica, obtidos a partir de uma GANN Aranda. A linha vertical representa a mediana da frequência cardíaca.

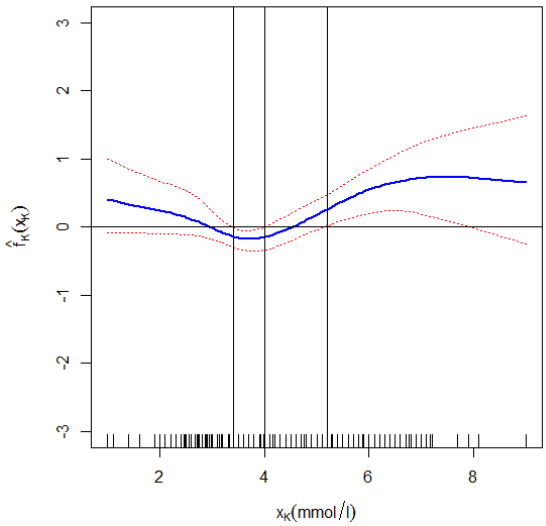

Figura 9.9: Estimativa da função parcial e respetivo intervalo de confiança a 95%, obtidos a partir de uma GANN Aranda, traduzindo a associação entre o potássio sérico (unidade expressa em $mmol/l$) e o evento *morte*.

em relação a esse valor de referência. No entanto para valores abaixo da mediana este aumento de risco não tem significado estatístico.

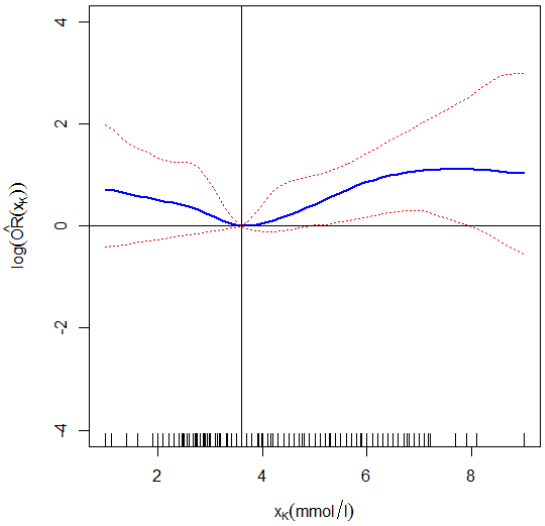

Figura 9.10: Estimativas da função OR e respetivo intervalo de confiança a 95% correspondentes ao potássio sérico em escala logarítmica, obtidos a partir de uma GANN Aranda. A linha vertical representa a mediana do potássio sérico.

9.4.4 Ventilação artificial

O doente é ligado a este mecanismo quando as suas funções respiratórias não são suficientes para mantê-lo vivo. Como seria expectável, o doente que é ligado ao ventilador já se encontra com um risco elevado de morte, como se pode observar pela fig. 9.11, tendo-se verificado que $\hat{\beta}_{vent} = 5.85$ (IC 95%: $2.76, 8.15$).

9.5 Discussão

Importa, em primeiro lugar, salientar o facto da aplicação a um caso real descrita neste capítulo, ter como base a teoria que foi sendo construída nos capítulos anteriores. O desenvolvimento dessa teoria fundamentou-se não apenas em métodos já existentes na área das GANNs como também em novas metodologias que foram introduzidas a partir do conhecimento existente

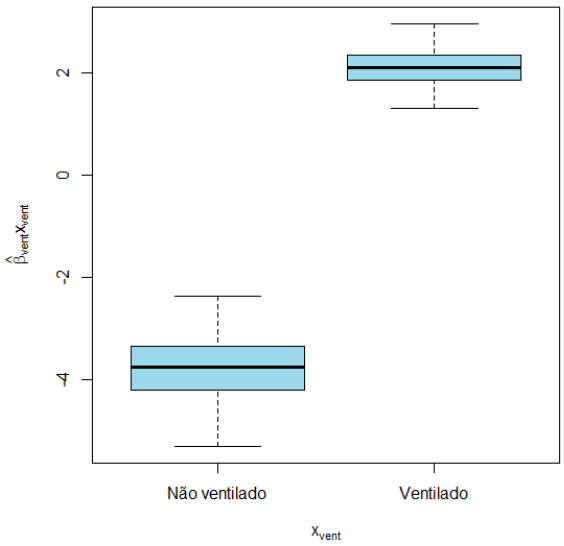

Figura 9.11: Distribuições dos valores de $\hat{\beta}_{vent}x_{vent}$ correspondentes às categorias "Ventilado" (doente ligado ao ventilador) e "Não ventilado".

quer na área das redes neuronais em geral, quer na área dos modelos de regressão.

Em consequência, a GANN que resultou dessa introdução de novas funcionalidades e melhorias, nomeadamente a GANN com função de ligação flexível paramétrica pertencente à família de transformações assimétricas de Aranda-Ordaz, apresenta um bom desempenho, tendo ficado, nesta aplicação, muito próximo do obtido pela sua congénere dos modelos de regressão (o GAM Aranda).

Efetivamente, a introdução de uma função de ligação flexível na arquitetura da GANN, confere-lhe a possibilidade de exploração de outras opções para além da tradicional função logística que em determinados cenários, podem levar a uma melhoria das estimativas.

Adicionalmente, há também a salientar a melhoria da interpretabilidade, com base na obtenção das estimativas das funções parciais e da função OR com os respetivos intervalos de confiança, tornando o modelo mais completo que a sua versão original.

Assim, acreditamos que as melhorias introduzidas podem motivar a utilização deste tipo de redes neuronais na área da investigação clínica, dada a

sua capacidade de adaptação aliada à interpretabilidade.

Capítulo 10

Conclusões

Para melhorar a aplicabilidade das Redes Neuronais Aditivas Generalizadas (*Generalized Additive Neural Networks* - GANNs) foi necessária equalizá-la, o mais possível, aos modelos de regressão, onde se encontram incluídos o Modelo Aditivo Generalizado ((*Generalized Additive Model* - GAM) e o Modelo Linear Generalizado (*Generalized Linear Model* - GLM), ambos, já com versões em que lhes é permitido utilizar uma função de ligação flexível. Uma vez que os modelos de regressão são dos mais utilizados no âmbito clínico, é expectável que a concretização deste objetivo confira alguma relevância à utilização das redes neuronais na área médica, em particular à aplicação da GANN com função de ligação flexível.

Para tal, houve que introduzir novas melhorias na arquitetura da GANN e desenvolver alguns métodos que permitiram melhorar a sua interpretabilidade. Efetivamente, a introdução da função de ligação flexível, que representa uma melhoria na arquitetura, só foi possível através do estudo do paralelismo existente entre cada um dos componentes dos modelos de regressão (componentes sistemática e aleatória onde se inclui a função de ligação) e o correspondente topológico nas redes neuronais.

A introdução da função de ligação flexível, face à utilização de uma função de ligação definida *a priori* (*e.g.* a função logística) vem dotar a GANN com mais opções que, em determinadas circunstâncias, permitem não só melhorar as estimativas da resposta, como também um melhor desempenho tanto do preditor linear, como das funções parciais, conferindo, assim, uma maior interpretabilidade ao modelo resultante.

No entanto, é necessário ter presente que, na maioria dos cenários, a função de ligação logística demonstra ser bastante robusta o que está em linha,

uma vez mais, com o que se constata nos GAMs. Com efeito, no estudo efetuado por Papoila (2006), cujo objeto de análise se centrou nos GAMs com função de ligação flexível, são apresentadas conclusões similares.

Apesar dos poucos estudos existentes na área das GANNs, foi também possível a introdução de métodos que configuram aspetos inovadores no funcionamento desta rede neuronal, de modo a incentivar a sua aplicabilidade no âmbito médico.

Como exemplo dessas melhorias, destacamos a estimação intervalar das funções parciais que definem a associação entre a variável explicativa e a resposta, bem como a resolução dos problemas de identificabilidade de modo a melhorar a utilidade da GANN em relação à GAM. No entanto, consideramos que a inovação mais importante no âmbito da interpretabilidade foi conseguida através da aplicação do método de estimação da função OR, independentemente da natureza da variável explicativa correspondente (*i.e.* contínua ou discreta), e independentemente da função de ligação estimada.

Ao longo deste estudo ficaram patentes as potencialidades da GANN com função de ligação flexível na área médica, face ao Perceptrão Multicamada (*Multi Layer Perceptron* - MLP), cuja opacidade das relações entre as variáveis explicativas e a variável de interesse tem desmotivado a utilização das redes neuronais na área da medicina face aos modelos de regressão.

Efetivamente, uma vez que os resultados obtidos pela GAM Aranda e pela GANN equivalente demonstram ser bastante similares (apesar da estimação destes modelos se basearem em métodos distintos no processo de adaptação), abre caminho para que se possam introduzir eventuais avanços, quer no domínio dos modelos de regressão, quer no domínio das redes neuronais.

Quanto ao processo de seleção de modelos utilizado na construção da GANN com função de ligação flexível, poder-se-á basear no algoritmo já existente - o AutoGANN, bastando adicionar genes ao vetor utilizado para definir a estrutura da rede, ou utilizar gráficos de perfil para determinar qual o melhor parâmetro a utilizar, à semelhança dos GAMs com função de ligação flexível.

No entanto, verificamos que o tempo de procura de um modelo ótimo pode ser significativo, especialmente nos casos em que o espaço de procura é grande devido a um elevado número de variáveis explicativas. O facto de se utilizar um vetor para codificação da topologia da GANN, abre as portas

para a utilização de outros métodos, nomeadamente os de procura global, em que podemos destacar como exemplo, os algoritmos genéticos (Bras-Geraldes et al., 2013). Sugere-se, assim, como um possível desenvolvimento futuro, a introdução de métodos de procura global ou mistos para reduzir o tempo de seleção de modelos.

Como nota final, face ao exposto em relação às melhorias introduzidas na GANN, acreditamos que este tipo de arquitetura possa vir a ser utilizada com maior frequência na área da investigação biomédica.

Apêndice A

Plataforma CG-GANN

A.1 Descrição geral

Para implementar os modelos utilizados neste estudo, desenvolveu-se uma plataforma de software, tendo-se recorrido à linguagem Java, para desenvolvimento de todas as classes que a compõem. Esta plataforma produz como *output* um conjunto de ficheiros que, posteriormente, são lidos por *scripts* desenvolvidos com recurso à linguagem R para apresentação dos resultados.

A plataforma foi construída numa lógica de camadas, em que cada uma reúne um conjunto de classes, responsável pelo processamento dos valores a um determinado nível e que passamos a descrever.

A.2 Primeira camada

A.2.1 Descrição

Nesta camada encontram-se definidas as classes que constituem a rede, nomeadamente, os nós e as sinapses. Também se encontram aqui incluídas classes abstratas a que designamos por componentes e supercomponentes. No contexto do *software* desenvolvido, uma componente corresponde a uma subtopologia genérica de uma rede neuronal e uma supercomponente corresponde a uma subtopologia genérica que agrega várias componentes. As classes abstratas, criadas nesta camada para implementação destes conceitos, servem como base para a definição de subarquiteturas mais específicas, sendo constituídas por variáveis e métodos que podem ser utilizados por qualquer subarquitetura.

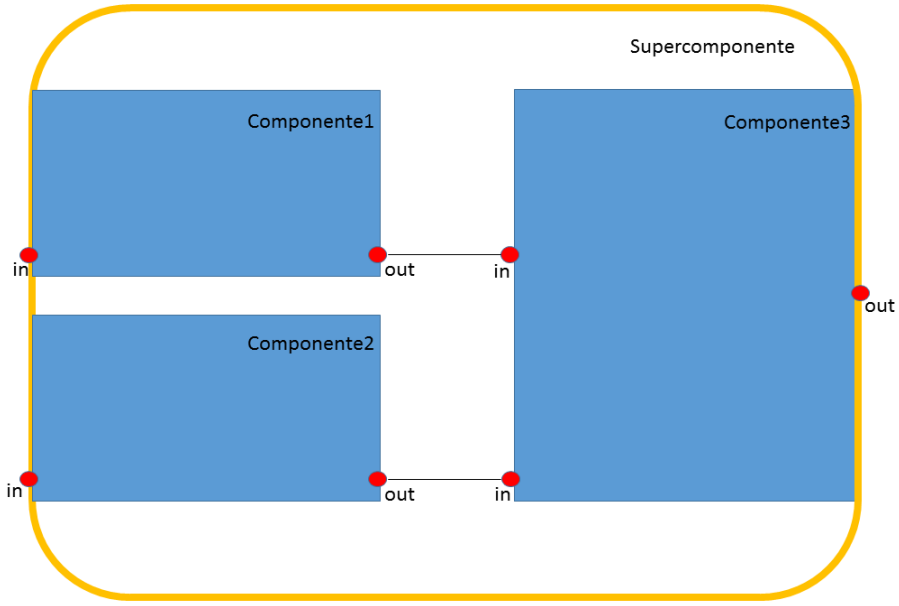

Figura A.1: Exemplo de um supercomponente que agrega várias subarquiteturas (componentes), no contexto do *software* desenvolvido neste estudo. Como se pode observar, cada componente, assim como o supercomponente, possui pontos de entrada e de saída para interligação com outro componente ou supercomponente.

A.2.2 Código Fonte

```
package cgANN.Topology;
public class Net {
        public double[][] dW;
        public double[][] W;
        public double[] Gi; //instantaneous Gradient
        public int maxsinapses;
        public int maxepocas;
        public Net(int Newmaxsinapses, int Newmaxepocas) {
                maxsinapses = Newmaxsinapses;
                maxepocas = Newmaxepocas;
                dW = new double[maxsinapses][maxepocas+1];
                W = new double[maxsinapses][maxepocas+1];
                Gi = new double[maxsinapses];
        }
        public void LoadW(double[] NewW, int epoca){
                for (int z=0; z<maxsinapses; z++){
                        W[z][epoca]=NewW[z];
                }
        }
        public void LoaddW(double[] NewdW, int epoca){
                for (int z=0; z<maxsinapses; z++){
                        dW[z][epoca]=NewdW[z];
                }
        }
}
```

Listagem A.1: Classe Net, composta por variáveis que guardam valores básicos obtidos pelas sinapses da rede, nomeadamente, os respetivos pesos e gradientes. Os métodos implementados, permitem o carregamento dos valores dos pesos sinápticos, bem como da diferença destes mesmos valores entre a época anterior e a época atual.

```
package cgANN.Topology;

import com.rits.cloning.Cloner;
import cgModel.InitW;

public abstract class Component {
        //Node Obligatory Topology Parameters
        public double[] inputsignal;
        public double[] outputsignal;
        public double[] inputerror;
        public double[] outputerror;
        public int n_syn;//number of synapses
        public int n_in;//Number of inputs
        public int n_out;//Number of outputs
        public double[] GMap;//Map of Gradients
        public Net network;//Synapse Parameters
        public Component(int Newn_in,int Newn_out){
                n_in = Newn_in;
                n_out = Newn_out;
                inputsignal = new double[n_in];
                outputerror = new double[n_in];
                inputerror = new double[n_out];
                outputsignal = new double[n_out];
        }
        public abstract void ff(int epoca);
        public abstract void bk(int epoca);
        public int LoadW(double[] NewW, int position, int epoca){
                double[] arraux = new double[n_syn];
                System.arraycopy( NewW, position, arraux, 0, arraux.length );
                network.LoadW(arraux, epoca);
                position+=arraux.length;
                return position;
        }
        public int LoaddW(double[] NewdW, int position, int epoca){
                double[] arraux = new double[n_syn];
                System.arraycopy( NewdW, position, arraux, 0, arraux.length );
                network.LoaddW(arraux, epoca);
                position+=arraux.length;
                return position;
        }
        public abstract void GradientCalc(int epoca, int ind);
        public abstract InitW LoadInitialWeights(InitW winit,int component_id);
        public Component deepCopy(){
                Cloner cloner=new Cloner();
                Component n = (Component) cloner.deepClone(this);
                return n;
        }

}
```

Listagem A.2: Classe Component que implementa em termos abstratos um conjunto de métodos responsáveis pela transmissão do sinal à entrada da subarquitetura para a saída e vice-versa.

```
package cgANN.Topology;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import cgModel.InitW;
import cgUtils.CGFunctions;
public abstract class SuperComponent extends Component{
        public List<Component> components;
        public int lastindex;
        public boolean serial;
        public SuperComponent(int n_in, int n_out) {
                super(n_in, n_out);
        }
        public void defineSuperComponent(ArrayList<? extends Component> Newsubtopologies
            ,boolean Newserial){
                serial = Newserial;
                components = new ArrayList<Component>();
                components.addAll(Newsubtopologies);
                lastindex = components.size()-1;
                if(serial){
                                        n_in = components.get(0).n_in;
                                        n_out = components.get(lastindex).n_out;
                }else{
                                        n_in=0;
                                        n_out=0;
                                        for(int component_id=0; component_id<=lastindex;
                                              component_id++){
                                                n_in+=components.get(component_id).n_in;
                                                n_out+=components.get(component_id).n_in
                                                      ;
                                        }
                }
                n_syn = getNumberofSynapses();
                GMap = new double[n_syn];
        }
        public void ff(int epoca) {
                if(serial){
                                        for(int component_id=0; component_id<=lastindex;
                                              component_id++){
                                                Component topology = components.get(
                                                      component_id);
                                                if(component_id==0){
                                                        for(int i=0; i<n_in;i++)
                                                              topology.inputsignal[i]=
                                                              inputsignal[i];
                                                }else{
                                                        for(int i=0; i<topology.n_in;i
                                                              ++) topology.inputsignal[i]=
                                                              components.get(component_id
                                                              -1).outputsignal[i];
                                                }
                                                topology.ff(epoca);
                                        }
                                        for(int i=0; i<n_out;i++) outputsignal[i] =
                                             components.get(lastindex).outputsignal[i];
                }else{
                                        int in=0;
                                        int out=0;
                                        for(int component_id=0; component_id<=lastindex;
                                              component_id++){
                                                Component topology = components.get(
                                                      component_id);
                                                for(int i=0; i<topology.n_in;i++){
                                                        topology.inputsignal[i]=
                                                              inputsignal[in];
                                                        in+=1;
                                                }
                                                topology.ff(epoca);
                                                for(int i=0; i<topology.n_out;i++){
                                                        outputsignal[out]=topology.
                                                              outputsignal[i];
                                                        out+=1;
                                                }
                                        }
                }
        }
        public void bk(int epoca) {
                if(serial){
                        for(int component_id=lastindex; component_id>=0; component_id--)
                                {
                                    Component topology = components.get(component_id);
```

```
                                if(component_id==lastindex){
                                        for(int i=0; i<n_out;i++) topology.inputerror[i
                                            ]=inputerror[i];
                                }else{
                                        for(int i=0; i<topology.n_out;i++) topology.
                                            inputerror[i]=components.get(component_id+1)
                                            .outputerror[i];
                                }
                                topology.bk(epoca);
                        }
                        for(int i=0; i<n_in;i++) outputerror[i] = components.get(0).
                            outputerror[i];
                }else{
                        int in=0;
                        int out=0;
                        for(int component_id=0; component_id<=lastindex; component_id++)
                            {
                                Component topology = components.get(component_id);
                                for(int i=0; i<topology.n_out;i++){
                                        topology.inputerror[i]=inputerror[out];
                                        out+=1;
                                }
                                topology.bk(epoca);
                                for(int i=0; i<topology.n_in;i++){
                                        outputerror[in]=topology.outputerror[i];
                                        in+=1;
                                }
                        }
                }
        }
        public int LoadW(double[] W, int position, int epoca){
                for(int component_id=0; component_id<=lastindex; component_id++){
                        Component topology = components.get(component_id);
                        position=topology.LoadW(W, position,epoca);
                }
                return position;
        }
        public int LoaddW(double[] dW, int position, int epoca){
                for(int component_id=0; component_id<=lastindex; component_id++){
                        Component topology = components.get(component_id);
                        position=topology.LoaddW(dW, position,epoca);
                }
                return position;
        }
        public void GradientCalc(int epoca,int ind){
                for(int component_id=0; component_id<=lastindex; component_id++){
                        Component topology = components.get(component_id);
                        topology.GradientCalc(epoca,ind);
                        if(component_id>0){
                                GMap=CGFunctions.JoinArray(GMap,topology.GMap);
                        }else{
                                GMap =  topology.GMap;
                        }
                }
        }
        public int getNumberofSynapses(){
                int result =0;
                Iterator<Component> it=this.components.iterator();
                while(it.hasNext()){
                        Component topology=(Component)it.next();
                        result += topology.n_syn;
                }
                return result;
        }
        public InitW LoadInitialWeights(InitW winit,int component_id){
                for(component_id = 0;component_id<components.size();component_id++){
                        winit = components.get(component_id).LoadInitialWeights(winit,
                            component_id);
                }
                return winit;
        }
}
```

Listagem A.3: Classe Supercomponente que implementa uma supercomponente.

```
package cgANN.Topology;

import cgUtils.Mathf;

public class ActiveFunction {

        public static double function(double Sum, String functype,boolean derivative,
            double alfa){
                double result=Double.POSITIVE_INFINITY;
                if(functype=="sigmoid"){
                        if(!derivative){
                                result = (Mathf.sigmoid(Sum));
                        }else{

                                result = (Mathf.dsigmoid(Sum));

                        }
                }
                if(functype=="tangh"){
                        if(!derivative){
                                result = (Mathf.tangh(Sum));
                        }else{

                                result = (Mathf.dtangh(Sum));

                        }
                }
                if(functype=="linear"){
                        if(!derivative){
                                result = (Mathf.linear(Sum));
                        }else{

                                result = (Mathf.dlinear(Sum));

                        }
                }
                if(functype=="aranda"){
                        if(!derivative){
                                result = (Mathf.aranda(Sum,alfa));
                        }else{

                                result = (Mathf.daranda(Sum,alfa));

                        }
                }
                return result;
        }
}
```

Listagem A.4: Classe ActiveFunction que implementa a função de ativação dos neurónios de uma determinada subarquitetura.

A.3 Segunda camada

Descrição

Nesta camada são especificadas as subarquiteturas que irão ser utilizadas para formar uma determinada topologia, através da junção destas tipo ”lego”, juntando a saída de um componente com a entrada de outro, conforme a fig. A.2. Para que esta abordagem resulte, os componentes têm de propagar corretamente os sinais da entrada para a respetiva saída, bem como num contexto de se utilizar o método de retropropagação, propagar os sinais da saída para a entrada, tornando assim possível a construção modular de estruturas mais complexas.

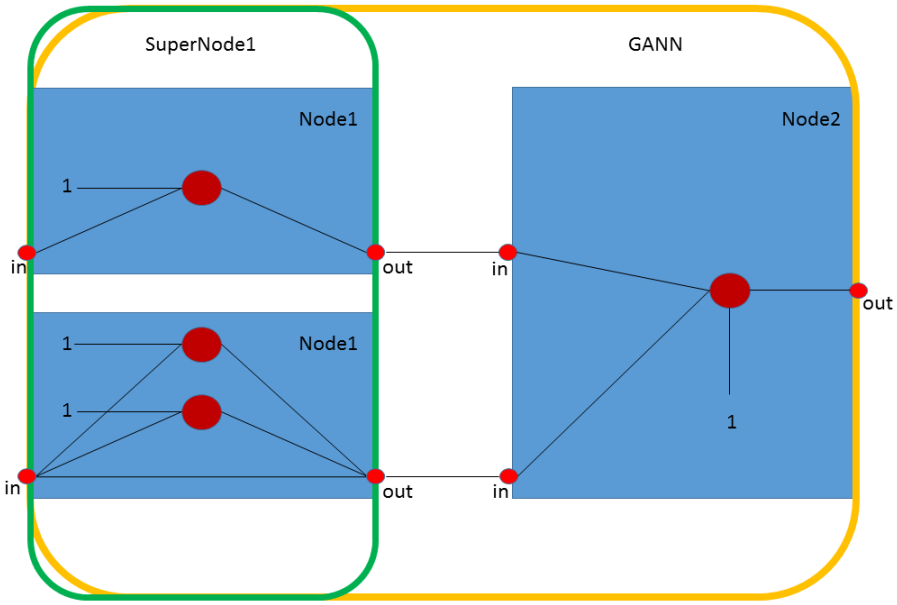

Figura A.2: Exemplo de um supercomponente do tipo ”GANN” composto por uma supercomponente do tipo ”SuperNode1” e por uma componente do tipo "Node2". O supercomponente do tipo ”SuperNode1” é constituído por um conjunto variável de componentes do tipo ”Node1”.

Código Fonte

```
package cgANN.GANN;

import java.util.ArrayList;

import org.apache.commons.lang.ArrayUtils;

import cgModel.InitW;
import cgUtils.CGFunctions;
import cgUtils.Mathf;
import cgANN.Topology.*;

public class Node1 extends Component{

        //Specific Topology Parameters
        public double[] Nb,Sb,Eb,Cb;
        public int nhl_abs;//number of hidden layer nodes
        public int nhl_signum;
        public double hidden; //output of tg(w01+w1)
        public double skip; //output of w0x
        public double S,E;//Summ and Error component of Output Node
        public String activefunction;
        public String linkfunction;
        public boolean intercept; //introduz intercept no node1
        public double betax;
        public double fmed;
        public double fxzero;//valor acumulado da função parcial para x=0
        public boolean fxintercept;
        public boolean identifiability;
        public Node1(int maxepocas,int Newnhl,String Newactivefunction, String
            Newlinkfunction, boolean Newintercept, boolean Newfxintercept, boolean
            Newidentifiability) {
                super(1,1);
                nhl_abs = (int) Math.abs(Newnhl);
                nhl_signum = (int) Math.signum(Newnhl);
                activefunction = Newactivefunction;
                linkfunction = Newlinkfunction;
                intercept = Newintercept;
                fxintercept = Newfxintercept;
                int aux = (intercept) ? 1 : 0;
                if(nhl_signum>=0){
                        n_syn = 3*nhl_abs+1+aux;
                }else{
                        n_syn = 3*nhl_abs+aux;
                }
                Nb = new double[nhl_abs];
                Sb = new double[nhl_abs];
                Eb = new double[nhl_abs];
                Cb = new double[nhl_abs];
                inputsignal[0] = 0;
                outputerror[0] = 0;
                network = new Net(n_syn, maxepocas);
                GMap = new double[n_syn];
        }
        public void ff(int epoca) {
                if(nhl_signum>=0){
                        double nlcsum = 0;
                        fxzero=0;
                        for ( int i=0; i<nhl_abs; i++ ){
                                Sb[i]=network.W[3*i+1][epoca-1]*inputsignal[0]+network.W
                                    [3*i+3][epoca-1];
                                if(activefunction=="tangh"){
                                        Nb[i]=Mathf.tangh(Sb[i]);
                                }
                                if(activefunction=="sigmoid"){
                                        Nb[i]=Mathf.sigmoid(Sb[i]);
                                }
                                if(activefunction=="mix"){
                                        Nb[i]=Mathf.mix(Sb[i],i);
                                }
                                nlcsum+=network.W[3*i+2][epoca-1]*Nb[i];
                                fxzero+=Mathf.tangh(network.W[3*i+3][epoca-1])*network.W
                                    [3*i+2][epoca-1];
                                if(!fxintercept) network.W[3*i+3][epoca-1]=0; //
                                    fxintercept=false
                        }
                        S = network.W[0][epoca-1]*inputsignal[0] + nlcsum;

                        if(intercept){
                                betax = network.W[n_syn-1][epoca-1];
                                S+=network.W[n_syn-1][epoca-1];
```

```
                        }
                        S=S-fmed; //Identifiability parameter

                        hidden = nlcsum/inputsignal[0];
                        skip = network.W[0][epoca-1]*inputsignal[0];
                }else{
                        double nlcsum = 0;
                        fxzero=0;
                        for ( int i=0; i<nhl_abs; i++ ){
                                Sb[i]=network.W[3*i][epoca-1]*inputsignal[0]+network.W
                                    [3*i+2][epoca-1];
                                if(activefunction=="tangh"){
                                        Nb[i]=Mathf.tangh(Sb[i]);
                                }
                                if(activefunction=="sigmoid"){
                                        Nb[i]=Mathf.sigmoid(Sb[i]);
                                }
                                if(activefunction=="mix"){
                                        Nb[i]=Mathf.mix(Sb[i],i);
                                }
                                nlcsum+=network.W[3*i+1][epoca-1]*Nb[i];
                                fxzero+=Mathf.tangh(network.W[3*i+2][epoca-1])*network.W
                                    [3*i+1][epoca-1];
                                if(!fxintercept) network.W[3*i+2][epoca-1]=0; //
                                    fxintercept=false
                        }
                        S = nlcsum;

                        if(intercept){
                                betax = network.W[n_syn-1][epoca-1];
                                S+=network.W[n_syn-1][epoca-1];
                        }

                        S=S-fmed; //Identifiability parameter

                        hidden = nlcsum/inputsignal[0];
                        skip = 0;
                }
                outputsignal[0] = ActiveFunction.function(S,linkfunction,false,0);
        }
        public void bk(int epoca) {
                double nlcsum = 0;
                E = inputerror[0]*ActiveFunction.function(S,linkfunction,true,0);

                if(nhl_signum>=0){
                        for ( int i=0; i<nhl_abs; i++ ){
                                Cb[i]=network.W[3*i+2][epoca-1]*E;
                                if(activefunction=="tangh"){
                                        Eb[i]=Cb[i]*Mathf.dtangh(Sb[i]);
                                }
                                if(activefunction=="sigmoid"){
                                        Eb[i]=Cb[i]*Mathf.dsigmoid(Sb[i]);
                                }
                                if(activefunction=="mix"){
                                        Eb[i]=Cb[i]*Mathf.dmix(Sb[i],i);
                                }
                                nlcsum += network.W[3*i+1][epoca-1]*Eb[i];
                        }
                        outputerror[0] = network.W[0][epoca-1]*E + nlcsum;
                }else{
                        for ( int i=0; i<nhl_abs; i++ ){
                                Cb[i]=network.W[3*i+1][epoca-1]*E;
                                if(activefunction=="tangh"){
                                        Eb[i]=Cb[i]*Mathf.dtangh(Sb[i]);
                                }
                                if(activefunction=="sigmoid"){
                                        Eb[i]=Cb[i]*Mathf.dsigmoid(Sb[i]);
                                }
                                if(activefunction=="mix"){
                                        Eb[i]=Cb[i]*Mathf.dmix(Sb[i],i);
                                }
                                nlcsum += network.W[3*i][epoca-1]*Eb[i];
                        }
                        outputerror[0] = nlcsum;
                }
        }
        public void GradientCalc(int epoca,int ind){
                if(nhl_signum>=0){
                        network.Gi[0]=E*inputsignal[0];
                        for ( int i=0; i<nhl_abs; i++ ){
                                network.Gi[3*i+1]=Eb[i]*inputsignal[0];
                                network.Gi[3*i+2]=Nb[i]*E;
                                network.Gi[3*i+3]=Eb[i];
```

```
                                if(!fxintercept) network.Gi[3*i+3]=0; //fxintercept=
                                    false
                        }
                        if(intercept) network.Gi[n_syn-1]=E;
                        for ( int i=0; i<n_syn; i++ ){
                                GMap[i] = network.Gi[i];
                        }
                }else{
                        for ( int i=0; i<nhl_abs; i++ ){
                                network.Gi[3*i]=Eb[i]*inputsignal[0];
                                network.Gi[3*i+1]=Nb[i]*E;
                                network.Gi[3*i+2]=Eb[i];
                                if(!fxintercept) network.Gi[3*i+2]=0; //fxintercept=
                                    false
                        }
                        if(intercept) network.Gi[n_syn-1]=E;
                        for ( int i=0; i<n_syn; i++ ){
                                GMap[i] = network.Gi[i];
                        }
                }

        }
        public InitW LoadInitialWeights(InitW winit, int component_id){
                double[] result = null;

                ArrayList<Double> node = new ArrayList<Double>();
                int offset;
                if(Math.signum(winit.nhl1_old[component_id])>=0){
                        offset = 1;
                }else{
                        offset = 0;
                }
                for(int z = 0; z < 3*Math.abs(winit.nhl1_old[component_id])+offset; z++)
                     {
                        node.add(winit.W[winit.pos_old]);
                        winit.pos_old+=1;
                }
                if(nhl_signum<0){
                        if(Math.signum(winit.nhl1_old[component_id])>=0){
                                node.remove(0);
                        }
                }else{
                        if(Math.signum(winit.nhl1_old[component_id])<0){
                                node.add(0,winit.NewW[winit.pos_w]);
                                winit.pos_w+=1;
                        }
                }
                int dif = nhl_abs-Math.abs(winit.nhl1_old[component_id]);
                if(dif>0){
                        for(int z=0; z<Math.abs(dif)*3;z++){
                                node.add(winit.NewW[winit.pos_w]);
                                winit.pos_w += 1;
                        }
                }else{
                        for(int z=0; z<Math.abs(dif)*3;z++){
                                int last_element = node.size()-1;
                                node.remove(last_element);
                        }
                }
                if(intercept){
                        node.add(winit.NewW[winit.pos_w]);
                        winit.pos_w+=1;
                }
                result = ArrayUtils.toPrimitive(node.toArray(new Double[node.size()]));
                winit.Wfinal = CGFunctions.JoinArray(winit.Wfinal,result);
                LoadW(result,0,0);
                return winit;
        }


}
```

Listagem A.5: Classe Node1.

```
package cgANN.GANN;

import cgANN.Task.OutputANNTask;
import cgANN.Task.TargetFunction;
import cgANN.Topology.*;
import cgUtils.CGFunctions;
import cgModel.InitW;
import cgSetup.Exit;;

public class Node2 extends Component{

        //Specific Topology Parameters
        public double S,E;//Summ and Error component of Output Node
        public String activefunction;
        public String linkfunction;
        public double alfa; //parameter of aranda
        public boolean fixweightsnode2;
        public boolean intercept;
        public double beta0;
        public double sumfmed;
        public boolean identifiability;
        public Node2(int n_in, int maxepocas, String Newlinkfunction, double Newalfa,
             boolean Newfixweightsnode2, boolean Newintercept, boolean Newidentifiability
             ) {
                super(n_in,1);
                n_out = 1;
                fixweightsnode2 = Newfixweightsnode2;
                intercept = Newintercept;
                n_syn = n_in+1;
                linkfunction = Newlinkfunction;
                alfa = Newalfa;
                network = new Net(n_syn, maxepocas);
                GMap = new double[n_syn];
        }
        public void ff(int epoca) {
                S = network.W[0][epoca-1];
                beta0 = network.W[0][epoca-1];
                for ( int i=0; i<n_in; i++ ){
                                S+=network.W[i+1][epoca-1]*inputsignal[i];
                }
                S=S+sumfmed; //Identifiability parameter
                outputsignal[0] = ActiveFunction.function(S,linkfunction,false,alfa);
        }
        public void bk(int epoca) {
                E = inputerror[0]*ActiveFunction.function(S,linkfunction,true,alfa);
                for ( int i=0; i<n_in; i++ ){
                                outputerror[i]=network.W[i+1][epoca-1]*E;
                }
        }
        public void GradientCalc(int epoca, int ind){
                network.Gi[0]=E;
                if(!intercept) network.Gi[0]=0;
                GMap[0]=network.Gi[0];
                for ( int i=0; i<n_in; i++ ){
                        network.Gi[i+1]=E*inputsignal[i];
                        if(fixweightsnode2) network.Gi[i+1]=0;//fixed parameters in
                             node2 to initial weights (i.e. 1)
                        GMap[i+1]=network.Gi[i+1];
                }

        }
        public InitW LoadInitialWeights(InitW winit, int component_id){
          if((winit.NewW.length-winit.pos_w)<n_syn) Exit.withError(2);
          double[] result = null;
          if(winit.source=="AOGLM"){

                for(int x=0;x<n_syn;x++){
                                double[] aux = new double[1];
                                if(winit.pos_old<winit.W.length){
                                                                        aux[0] = winit.W[
                                                                             winit.pos_old];
                                                                        if((fixweightsnode2)
                                                                             &&(x>=1)) aux
                                                                             [0]=1;   //fixed
                                                                             weights

                                                                        if((!intercept)&&(x
                                                                             <1)) aux[0]=0;
                                                                             //fixed weights
                                                                        result = CGFunctions.
                                                                             JoinArray(result,
                                                                             aux);
                                                                        winit.pos_old+=1;
```

```
                        }else{
                                                                aux[0] = winit.NewW[
                                                                    winit.pos_w];
                                                                if((fixweightsnode2)
                                                                    &&(x>=1)) aux
                                                                    [0]=1;  //fixed
                                                                    weights


                                                                //if((fixweightsnode2
                                                                    )&&(x>=1)) aux
                                                                    [0]=Math.sqrt(2)
                                                                    /2;  //fixed
                                                                    weights

                                                                if((!intercept)&&(x
                                                                    <1)) aux[0]=0;
                                                                    //fixed weights
                                                                result = CGFunctions.
                                                                    JoinArray(result,
                                                                    aux);
                                                                winit.pos_w+=1;
                        }
                }

                winit.Wfinal = CGFunctions.JoinArray(winit.Wfinal,result);
        }else{
                        result = new double[n_syn];
                        int offset = winit.pos_w;
                        for(int i=winit.pos_w;i<(offset+n_syn);i++){
                                result[i-offset]=winit.NewW[winit.pos_w];
                                if((fixweightsnode2)&&(i>=1)) result[i-offset]=1;  //
                                    fixed weights
                                winit.pos_w+=1;
                        }
                        winit.Wfinal = CGFunctions.JoinArray(winit.Wfinal,result);
        }
        LoadW(result,0,0);
        return winit;
    }
    public static void main(String[] args) throws Exception{
            double[][] training_sample = {{0.1,0.2,0.3,0.7},{0.4,0.5,0.6,0.9}};
            int n_ind = training_sample.length;
            int n_cov = training_sample[0].length-1;
            int maxepocas = 1;
            int epoca = 1;
            double alfa = 1;
            int[] nhl_old = new int[1];
            int[] nhl_outold = new int[1];
            nhl_old[0]=0;
            nhl_outold[0]=0;
            double[] Weights = {0.11,0.12,0.13,0.16};
            Node2 node2 = new Node2(n_cov,maxepocas,"sigmoid",alfa,true,false,false)
                ;
            OutputANNTask out = new OutputANNTask(n_ind,n_cov);
            for(int ind=0; ind<n_ind;ind++){
                    for(int i=0; i<n_cov;i++){
                            node2.inputsignal[i]=training_sample[ind][i];
                    }
                    out.observed[ind] = training_sample[ind][n_cov];
                    node2.ff(epoca);
                    out.estimated[ind] = node2.outputsignal[0];
                    node2.inputerror[0] = TargetFunction.dcdy(out,"QuadraticError
                        ",ind,epoca);
                    node2.bk(epoca);
            }
    }

}
```

Listagem A.6: Classe Node2.

```
package cgANN.GANN;

ava.util.ArrayList;

rg.apache.commons.lang.ArrayUtils;

gModel.InitW;
gUtils.CGFunctions;
gUtils.Mathf;
gANN.Topology.*;

lass Node3 extends Component{

//Specific Topology Parameters
public double[] Nb,Sb,Eb,Cb;
public int nhl_abs;//number of hidden layer nodes
ic int nhl_signum;
ic double hidden; //output of tg(w01+w1)
ic double skip; //output of w0x
ic double S,E;//Summ and Error component of Output Node
public String activefunction;
ic String linkfunction;
public double alfa; //parameter of aranda
public int intercept;// if true beta0 exists
public double beta1;

public Node3(int maxepocas,int Newnhl,String Newactivefunction, String Newlinkfunction,
     double Newalfa, boolean Newintercept) {
        super(1,1);
        nhl_abs = (int) Math.abs(Newnhl);
        nhl_signum = (int) Math.signum(Newnhl);
        activefunction = Newactivefunction;
        linkfunction = Newlinkfunction;
        alfa = Newalfa;
        if(nhl_signum>=0){
                        n_syn = 3*nhl_abs+1;
        }else{
                            n_syn = 3*nhl_abs;
        }
        intercept = (Newintercept) ? 1 : 0; //converts boolean to integer
n_syn+=intercept;
        Nb = new double[nhl_abs];
        Sb = new double[nhl_abs];
        Eb = new double[nhl_abs];
        Cb = new double[nhl_abs];
        inputsignal[0] = 0;
        outputerror[0] = 0;
        network = new Net(n_syn, maxepocas);
        GMap = new double[n_syn];
}
public void ff(int epoca) {
        if(nhl_signum>=0){
                            double nlcsum = 0;
                            for ( int i=0; i<nhl_abs; i++ )
                            {
                                    Sb[i]=network.W[3*i+1][epoca-1]*inputsignal[0]+
                                         network.W[3*i+3][epoca-1];
                                    if(activefunction=="tangh"){
                                                Nb[i]=Mathf.tangh(Sb[i]);
                                    }
                                    if(activefunction=="sigmoid"){
                                         Nb[i]=Mathf.sigmoid(Sb[i]);
                                }
                                    if(activefunction=="mix"){
                                         Nb[i]=Mathf.mix(Sb[i],i);
                                }
                                nlcsum+=network.W[3*i+2][epoca-1]*Nb[i];

                            }
                            if(intercept==1){
                                 S = network.W[n_syn-1][epoca-1];
                                 beta1 = network.W[n_syn-1][epoca-1];
                            }
                        S += network.W[0][epoca-1]*inputsignal[0] + nlcsum;

        }else{
                            double nlcsum = 0;
                for ( int i=0; i<nhl_abs; i++ )
                {
                        Sb[i]=network.W[3*i][epoca-1]*inputsignal[0]+network.W[3*i+2][
                            epoca-1];
                        if(activefunction=="tangh"){
                                    Nb[i]=Mathf.tangh(Sb[i]);
```

```
                                }
                                if(activefunction=="sigmoid"){
                                    Nb[i]=Mathf.sigmoid(Sb[i]);
                            }
                                if(activefunction=="mix"){
                                    Nb[i]=Mathf.mix(Sb[i],i);
                            }
                            nlcsum+=network.W[3*i+1][epoca-1]*Nb[i];
                        }
                        if(intercept==1){
                                S = network.W[n_syn-1][epoca-1];
                                beta1 = network.W[n_syn-1][epoca-1];
                        }
                                S += nlcsum;

                }
                outputsignal[0] = ActiveFunction.function(S,linkfunction,false,alfa);
        }
        public void bk(int epoca) {
                double nlcsum = 0;
        E = inputerror[0]*ActiveFunction.function(S,linkfunction,true,alfa);

        if(nhl_signum>=0){
                                for ( int i=0; i<nhl_abs; i++ ){
                                                        Cb[i]=network.W[3*i+2][epoca-1]*E;
                                                        if(activefunction=="tangh"){
                                                                Eb[i]=Cb[i]*Mathf.dtangh(Sb
                                                                        [i]);
                                                                }
                                                                if(activefunction==
                                                                        "sigmoid"){
                                                                                Eb[i]=Cb[i
                                                                                        ]*Mathf
                                                                                        .
                                                                                        dsigmoid
                                                                                        (Sb[i])
                                                                                        ;
                                                                }
                                                                if(activefunction==
                                                                        "mix"){
                                                                                Eb[i]=Cb[i
                                                                                        ]*Mathf
                                                                                        .dmix(
                                                                                        Sb[i],i
                                                                                        );
                                                                }
                                                        nlcsum += network.W[3*i+1][epoca-1]*Eb
                                                                [i];
                                }
                                outputerror[0] = network.W[0][epoca-1]*E + nlcsum;
        }else{
                                for ( int i=0; i<nhl_abs; i++ ){
                                                        Cb[i]=network.W[3*i+1][epoca-1]*E;
                                                        if(activefunction=="tangh"){
                                                            Eb[i]=Cb[i]*Mathf.dtangh(Sb[i]);
                                                                }
                                                                if(activefunction==
                                                                        "sigmoid"){
                                                                                Eb[i]=Cb[i
                                                                                        ]*Mathf
                                                                                        .
                                                                                        dsigmoid
                                                                                        (Sb[i])
                                                                                        ;
                                                                }
                                                                if(activefunction==
                                                                        "mix"){
                                                                                Eb[i]=Cb[i
                                                                                        ]*Mathf
                                                                                        .dmix(
                                                                                        Sb[i],i
                                                                                        );
                                                                }
                                                        nlcsum += network.W[3*i][epoca-1]*Eb[i
                                                                ];
                                }
                                        outputerror[0] = nlcsum;
        }
        }
        public void GradientCalc(int epoca,int ind){
                if(nhl_signum>=0){

                                        network.Gi[0]=0;//*
```

```
                                if(intercept==1) network.Gi[n_syn-1]=E;
                                for ( int i=0; i<nhl_abs; i++ ){

                                    network.Gi[3*i+1]=0;//*
                                            network.Gi[3*i+2]=Nb[i]*E;
                                            network.Gi[3*i+3]=Eb[i];
                                    }
                                    for ( int i=0; i<n_syn; i++ ){
                                        GMap[i] = network.Gi[i];
                                    }
        }else{
                                    if(intercept==1) network.Gi[n_syn-1]=E;
                                    for ( int i=0; i<nhl_abs; i++ ){

                                            network.Gi[3*i]=0;//*
                                        network.Gi[3*i+1]=Nb[i]*E;
                                        network.Gi[3*i+2]=Eb[i];
                                }
                                for ( int i=0; i<n_syn; i++ ){
                                    GMap[i] = network.Gi[i];
                                }
        }

}
public InitW LoadInitialWeights(InitW winit, int component_id){
        double[] result = null;
        if(winit.source=="AOGLM"){
            ArrayList<Double> node = new ArrayList<Double>();
            int offset;
            if(Math.signum(winit.nhl2_old[0])>=0){
                        offset = 1;
            }else{
                        offset = 0;
            }
            for(int z = 0; z < 3*Math.abs(winit.nhl2_old[0])+offset; z++){
                    node.add(winit.W[winit.pos_old]);
                    winit.pos_old+=1;
            }
            if(nhl_signum<0){
                    if(Math.signum(winit.nhl2_old[0])>=0){
                                                node.remove(0);
                    }
            }else{
                    if(Math.signum(winit.nhl2_old[0])<0){
                    node.add(0,winit.NewW[winit.pos_w]);
                    winit.pos_w+=1;
                }
            }
            int dif = nhl_abs-Math.abs(winit.nhl2_old[0]);
            if(dif>0){
                    for(int z=0; z<Math.abs(dif)*3;z++){
                            node.add(winit.NewW[winit.pos_w]);
                            winit.pos_w += 1;
                    }
            }else{
                    for(int z=0; z<Math.abs(dif)*3;z++){
                            int last_element = node.size()-1;
                        node.remove(last_element);
                    }
            }
            if(intercept==1) node.add(winit.beta0);
            node.set(0, 1.0);
            result = ArrayUtils.toPrimitive(node.toArray(new Double[node.size()]));
            winit.Wfinal = CGFunctions.JoinArray(winit.Wfinal,result);
        }else{
             result = new double[n_syn];
             for(int i=winit.pos_w;i<(winit.pos_w+n_syn);i++){
                 result[i-winit.pos_w]=winit.NewW[winit.pos_w];
                 winit.pos_w+=1;
             }
             winit.Wfinal = CGFunctions.JoinArray(winit.Wfinal,result);
    }
    if(nhl_signum>0){
            result[0] = 1;
            result[1] = 1; //skip layer and the first weight = 0
    }else{
            result[0]=1; //skip layer or the first weight = 0
    }
    LoadW(result,0,0);
    return winit;
}
```

Listagem A.7: Classe Node3.

```
package cgANN.GANN;

import java.util.ArrayList;
import cgANN.Topology.*;

public class SuperNode1 extends SuperComponent{
    //Specific Topology Parameters
        public Node1[] node1;
        public boolean intercept;
        public boolean fxintercept;
        public boolean identifiability;
    public int[] nhl;
    public SuperNode1(int Newn_in, int[] Newnhl,int maxepocas, String activefunction,
         String linkfunction, boolean Newintercept, boolean Newfxintercept, boolean
         Newidentifiability) {
        super(Newn_in,Newn_in);
        intercept = Newintercept;
        fxintercept = Newfxintercept;
        identifiability = Newidentifiability;
                node1 = new Node1[n_in];
                nhl = Newnhl;
                components = new ArrayList<Component>();
                for(int i=0; i<n_in;i++){
                    node1[i] = new Node1(maxepocas,nhl[i],activefunction,linkfunction,
                        intercept,fxintercept,identifiability);
                    components.add(node1[i]);
                }
                this.defineSuperComponent((ArrayList<? extends Component>) components,
                     false);
    }
}
```

Listagem A.8: Classe SuperNode1 que encapsula num supercomponente um conjunto variável de objetos do tipo "Node1" consoante o número de variáveis explicativas do modelo.

```
package cgANN.GANN;

import java.util.ArrayList;
import java.util.List;
import cgANN.Topology.*;

public class GANN extends SuperComponent{
        public List<Component> subtopologies;
        //Specific Topology Parameters
        public SuperNode1 supernode1;
        public Node2 node2;
        public int[] nhl;
        public GANN(int New_n_in,int New_n_out,int maxepocas,int[] Newnhl,String
             activefunction,String linkfunction,double alfa, boolean fixweightsnode2,
             ArrayList<Boolean> useIntercept, boolean identifiability) {
                super(New_n_in,New_n_out);
                nhl = Newnhl;
                subtopologies = new ArrayList<Component>();
                supernode1 = new SuperNode1(n_in,nhl,maxepocas,activefunction,"linear",
                     useIntercept.get(0),useIntercept.get(1),identifiability);
                subtopologies.add(supernode1);
                node2 = new Node2(n_in,maxepocas,linkfunction,alfa,fixweightsnode2,
                     useIntercept.get(2),identifiability);
                subtopologies.add(node2);
                this.defineSuperComponent((ArrayList<? extends Component>) subtopologies
                     ,true);
        }


}
```

Listagem A.9: Classe GANN que inclui um supercomponente do tipo ”SuperNode1” e um componente do tipo ”Node2” conforme se pode observar na fig. A.2.

```
package cgANN.GANN;

import java.util.ArrayList;
import java.util.List;
import cgANN.Topology.*;

public class GANNnonparam extends SuperComponent{
        public List<Component> subtopologies;
        //Specific Topology Parameters
        public SuperNode1 supernode1;
        public Node2 node2;
        public Node3 node3;
        public int[] nhl1;
        public int nhl2;

        public GANNnonparam(int New_n_in,int New_n_out,int maxepocas,int[] Newnhl,int[]
             Newnhl2,String activefunction,String linkfunction,double alfa, boolean
             fixweightsnode2,boolean identifiability) {
                super(New_n_in,New_n_out);
                nhl1 = new int[Newnhl.length];
                for(int i=0;i<n_in;i++){
                        nhl1[i]=Newnhl[i];
                }
                nhl2 = Newnhl2[0];
                subtopologies = new ArrayList<Component>();
                supernode1 = new SuperNode1(n_in,nhl1,maxepocas,activefunction,"linear",
                     false,true,identifiability);
                subtopologies.add(supernode1);
                node2 = new Node2(n_in,maxepocas,"linear",alfa,fixweightsnode2,true,
                     identifiability);
                subtopologies.add(node2);
                node3 = new Node3(maxepocas,nhl2,"sigmoid",linkfunction,alfa,true);
                subtopologies.add(node3);
                this.defineSuperComponent((ArrayList<? extends Component>) subtopologies
                     ,true);
        }
}
```

Listagem A.10: Classe GANNnonparam que inclui um supercomponente do tipo ”SuperNode1”, um componente do tipo ”Node2” e outro do tipo ”Node3”.

A.4 Terceira camada

A.4.1 Descrição

Nesta camada são definidas classes que associam tarefas básicas à topologia global implementada na camada anterior. De entre os métodos implementados, destacam-se os processos de treino, validação e teste, bem como o processo de atualização dos pesos sinápticos recorrendo a técnicas de aceleração de 1º e 2ª ordem, nomeadamente os métodos RPROP e Levenberg Marquadt.

A.4.2 Código Fonte

```
package cgANN.Task;

rt java.io.FileNotFoundException;
rt java.io.IOException;
rt java.util.ArrayList;
rt java.util.Arrays;
rt java.util.List;

rt cgANN.GANN.GANN;
rt cgANN.Topology.Component;
rt cgSetup.Cromossom;
rt cgSetup.Setup;
rt cgUtils.CGFunctions;
rt cgUtils.Mathf;

ic abstract class ANNTask  implements Cloneable{
        public Component ann;
        public int n_syn; //Total Number of synapses
        public int[] nhl;
        public double[][] GradientMap;
        public double[][] GradientMap2;
        public double[] MSE;
        public double[] MF;
        public double[] Error;
        public double[][] dW;
        public double[][] W;
        public double[][] delta,etav;
        public double[] lamda;
            public String targetfunc;
            public boolean identifiability;
            public int epoca_ref=68;

            public ANNTask(Setup setup,Cromossom cromossom) throws FileNotFoundException
                , IOException{
                    nhl = cromossom.inputgenes;
                    targetfunc = setup.targetfunc;
            }
            public void defineANNTask(Component ann,Setup setup){
                    n_syn=ann.n_syn;
                    GradientMap2 = new double[setup.maxepocas+1][n_syn];
                    dW = new double[setup.maxepocas+1][n_syn];
                    W = new double[setup.maxepocas+1][n_syn];
                    delta = new double[n_syn][setup.maxepocas+1];
                    lamda = new double[setup.maxepocas+1];
                    etav = new double[n_syn][setup.maxepocas+1];
                    MSE = new double[setup.maxepocas+1];
                    MF = new double[setup.numberofcov];
                    for (int z=0; z<n_syn; z++){
                     delta[z][0]=setup.delta0;
                     etav[z][0]=setup.eta;
                     dW[0][z]=0;
                              W[0][z]=0;
                              GradientMap2[0][z]=0;
                }
                    for (int i=1; i<(setup.maxepocas+1); i++){
                    for (int z=0; z<n_syn; z++){
                              delta[z][i]=0;
                              etav[z][i]=0;
```

```
                        dW[i][z]=0;
                                W[i][z]=0;
                                GradientMap2[i][z]=0;

                    }
                lamda[i]=0;
            }
                lamda[0]=setup.lamda;
                identifiability = setup.identifiability;
        }
        public void UpdateWeights(Setup setup,String method, int epoca) throws
             FileNotFoundException, IOException{
        if(method=="lm"){
                int comp = Double.compare(MSE[epoca],MSE[epoca-1]);
            if((comp<0)||epoca<2){
                double param = Math.exp(lamda[epoca-1]);
                        dW[epoca]=Mathf.InvertedHessian(Mathf.Jacobian(GradientMap,
                             Error),param).times(-1).times(Mathf.Gradient(GradientMap
                             )).transpose().getArray()[0];
                    for (int z=0; z<n_syn; z++){
                            W[epoca][z] = W[epoca-1][z]+dW[epoca][z];
                    }
                    lamda[epoca] = lamda[epoca-1]*setup.beta2;
                    }else{
                            lamda[epoca] = lamda[epoca-1]/(setup.beta2*setup.beta2);

                            double[] MSEaux = Arrays.copyOfRange(MSE, 1, epoca);
                            int lastepoca = Mathf.minindex(MSEaux);
                        for (int z=0; z<n_syn; z++){
                                dW[epoca][z] = dW[lastepoca][z];
                               W[epoca][z] = W[lastepoca][z];
                        }
                    }
    }
    if (method=="rprop"){
            for (int z=0; z<n_syn; z++){
                            if(GradientMap2[epoca][z]*GradientMap2[epoca-1][z]>=0){
                                if(GradientMap2[epoca][z]*GradientMap2[epoca-1][z
                                     ]==0){
                                     delta[z][epoca]=delta[z][epoca-1];
                                }else{
                                     delta[z][epoca]=delta[z][epoca-1]*setup.miuplus;
                                }
                            }else{
                                 delta[z][epoca]=delta[z][epoca-1]*setup.miuminus;
                            }
                            if(delta[z][epoca]>=setup.deltamax){
                                 delta[z][epoca]=setup.deltamax;
                            }
                            if(delta[z][epoca]<=setup.deltamin){
                                 delta[z][epoca]=setup.deltamin;
                            }
                            if(GradientMap2[epoca][z]>0){
                                dW[epoca][z]=-delta[z][epoca];
                            }
                            if(GradientMap2[epoca][z]<0){
                                dW[epoca][z]=delta[z][epoca];
                            }
                            if(GradientMap2[epoca][z]==0){
                                dW[epoca][z]=0;
                            }
                            if(GradientMap2[epoca][z]*GradientMap2[epoca-1][z]<0) dW
                                 [epoca][z] = -dW[epoca-1][z];
                            GradientMap2[epoca][z] = 0;
                            W[epoca][z] = W[epoca-1][z]+dW[epoca][z];
                        }
                }
        if (method=="grad"){
            for (int z=0; z<n_syn; z++){
                        dW[epoca][z] = -setup.eta*GradientMap2[epoca][z]+setup.
                             beta*dW[epoca-1][z];
                        W[epoca][z] = W[epoca-1][z]+dW[epoca][z];
                    }
            }
        if (method=="grad2"){

                for (int z=0; z<n_syn; z++){
                            if(GradientMap2[epoca][z]*GradientMap2[epoca-1][z]>=0){
                                    etav[z][epoca]=etav[z][epoca-1]*setup.miuplus;
                                    dW[epoca][z] = -etav[z][epoca]*GradientMap2[epoca
                                         ][z]+setup.beta*dW[epoca-1][z];
                            }else{
                            etav[z][epoca]=etav[z][epoca-1]*setup.miuminus;
```

```
                        dW[epoca][z] = -etav[z][epoca]*GradientMap2[epoca][z];
                    }
                        W[epoca][z] = W[epoca-1][z]+dW[epoca][z];
                }

        }
        if (method=="matlab"){
                for ( int i=0; i<n_syn; i++ ){
                 GradientMap2[epoca][i]=2*GradientMap2[epoca][i];
            }
            for (int z=0; z<n_syn; z++){
                        dW[epoca][z] = -setup.eta*GradientMap2[epoca][z];
                        W[epoca][z] = W[epoca-1][z]+dW[epoca][z];
                 }
            }
        if (method=="null"){
                for (int z=0; z<n_syn; z++){
                        dW[epoca][z] = 0;
                        W[epoca][z] = W[epoca-1][z]+dW[epoca][z];
                }
            }
        ann.LoaddW(dW[epoca],0,epoca);
                ann.LoadW(W[epoca],0,epoca);
    }
        public OutputANNTask PreTraining(double[][] training_sample, int epoca)
             throws FileNotFoundException, IOException{
             //Init Variables
             int n_ind =  training_sample.length;
             int n_cov =  training_sample[0].length-1;
             OutputANNTask out = new OutputANNTask(n_ind,n_cov);

             double[] f = new double[n_cov]; //partial function acumulator
         for(int i=0; i<n_cov;i++) f[i]=0;

             //PreTraining
             double[][] aux = new double[n_ind][n_cov];
             for(int ind=0; ind<n_ind;ind++){
                        //Feed forward
                             for(int i=0; i<n_cov;i++){
                                    ann.inputsignal[i]=training_sample[ind][i];
                             }
                             out.observed[ind] = training_sample[ind][n_cov];
                             ann.ff(epoca);
                             if(epoca==epoca_ref){
                             aux[ind]=getANNparams(epoca,n_cov);
                        }
                             for(int i=0; i<n_cov;i++) f[i]+=aux[ind][i];
                             out = probe(out,ind);
                        out.estimated[ind] = ann.outputsignal[0];
                             ann.inputerror[0] = ann.outputsignal[0]-training_sample[
                                  ind][n_cov];
                        //Error Energy
                        out.SquaredError[ind] = Math.pow(ann.inputerror[0],2);
             }
             //MF
             for(int i=0; i<n_cov;i++){
                 MF[i] = f[i]/n_ind;
             }
                 return out;
        }
        public OutputANNTask Training(double[][] training_sample, int epoca) throws
             FileNotFoundException, IOException{
         //Init Variables
                 int n_ind = training_sample.length;
         int n_cov = training_sample[0].length-1;

         OutputANNTask out = new OutputANNTask(n_ind,n_cov);
         GradientMap = new double[n_ind][n_syn];
         Error = new double[n_ind];

                 double[][] aux = new double[n_ind][n_cov];
         for(int ind=0; ind<n_ind;ind++){
                    if(identifiability) setIdentifiabilityConstraints(MF,ind);
                //Feed forward
                for(int i=0; i<n_cov;i++){
                    ann.inputsignal[i]=training_sample[ind][i];
                }
                out.observed[ind] = training_sample[ind][n_cov];
                ann.ff(epoca);
                if(epoca==epoca_ref){
                    aux[ind]=getANNparams(epoca,n_cov);
                }
                out = probe(out,ind);
```

```
                out.estimated[ind] = ann.outputsignal[0];
                ann.inputerror[0] = TargetFunction.dcdy(out,targetfunc,ind,epoca);

                //Back propagation
                    ann.bk(epoca);
                    ann.GradientCalc(epoca,ind);
                    //Error, Residuals and Gradient
                    out.Error[ind] = ann.inputerror[0];
                    Error[ind] = out.Error[ind];
                    out.SquaredError[ind] = Math.pow(out.Error[ind],2);
                for ( int i=0; i<n_syn; i++ ){
                        GradientMap[ind][i] = ann.GMap[i];
                   GradientMap2[epoca][i] += GradientMap[ind][i];
                    }
            }
            for ( int i=0; i<n_syn; i++ ){
                  GradientMap2[epoca][i]=GradientMap2[epoca][i]/n_ind;
            }

            //MSE
            out.MSE = Mathf.SumArray(out.SquaredError)/n_ind;
            MSE[epoca]=out.MSE;
            return out;
        }
        public OutputANNTask Validating(double[][] val_sample, int epoca) throws
            FileNotFoundException, IOException{
              //Init Variables
              int n_ind = val_sample.length;
              int n_cov = val_sample[0].length-1;
              OutputANNTask out = new OutputANNTask(n_ind,n_cov);

              //Validating
           for(int ind=0; ind<n_ind;ind++){
                  //Feed forward
                   for(int i=0; i<n_cov;i++){
                       ann.inputsignal[i]=val_sample[ind][i];
                   }
                   out.observed[ind] = val_sample[ind][n_cov];
                   ann.ff(epoca);
                   out = probe(out,ind);
                   out.estimated[ind] = ann.outputsignal[0];
                   ann.inputerror[0] = ann.outputsignal[0]-val_sample[ind][n_cov];
                   //Error Energy
                   out.SquaredError[ind] = Math.pow(ann.inputerror[0],2);
            }
            //MSE
            out.MSE = Mathf.SumArray(out.SquaredError)/n_ind;

                return out;
        }
        public OutputANNTask Testing(double[][] test_sample, int epoca) throws
            FileNotFoundException, IOException{
         //Init Variables
         int n_ind =  test_sample.length;
         int n_cov =  test_sample[0].length-1;
         OutputANNTask out = new OutputANNTask(n_ind,n_cov);

         //Testing
         double[][] aux = new double[n_ind][n_cov];
         for(int ind=0; ind<n_ind;ind++){
                   //Feed forward
                       for(int i=0; i<n_cov;i++){
                               ann.inputsignal[i]=test_sample[ind][i];
                       }
                       out.observed[ind] = test_sample[ind][n_cov];
                       ann.ff(epoca);
                       if(epoca==epoca_ref){
                       aux[ind]=getANNparams(epoca,n_cov);
                   }
                       out = probe(out,ind);
                   out.estimated[ind] = ann.outputsignal[0];
                       ann.inputerror[0] = ann.outputsignal[0]-test_sample[ind][n_cov];
                   //Error Energy
                   //out.Error[ind] = ann.inputerror[0];
                   out.SquaredError[ind] = Math.pow(ann.inputerror[0],2);

         }
         //MSE
         out.MSE = Mathf.SumArray(out.SquaredError)/n_ind;
              return out;
        }
        public abstract OutputANNTask probe(OutputANNTask out, int ind);
        public abstract void setIdentifiabilityConstraints(double[] MF, int ind);
```

```
        public abstract double getPartialFunctionsOutput(int cov);
        public abstract double[] getANNparams(int epoca, int n_cov);
        public abstract void setANNparams(double[][] params);
        public Object clone() throws CloneNotSupportedException {
                return super.clone();
        }
```

Listagem A.11: Classe abstrata ANNTask que implementa as tarefas de treino, validação e teste, bem como de atualização dos pesos em cada época de uma rede neuronal genérica.

```
package cgANN.Task;
public class OutputANNTask {
        public double[] observed;
        public double[] estimated;
        public double[][] fx;
        public double[][] betax;
        public double[][] fxzero;
        public double[] fmed;
        public double[][] hidden;
        public double[][] skip;
        public double[] beta0;
        public double[] beta1;
        public double[] S;
        public double MSE;
        public double[] Error;
        public double[] SquaredError;
        public double[] W;
        public double[] dW;
        public double[] alfa;
        public int n_ind;
        public int n_cov;
        public OutputANNTask(int Newn_ind, int Newn_cov){
                n_ind = Newn_ind;
                n_cov = Newn_cov;
                observed = new double[n_ind];
                estimated = new double[n_ind];
                fx = new double[n_ind][n_cov];
                betax = new double[n_ind][n_cov];
                beta0 = new double[n_ind];
                beta1 = new double[n_ind];
                Error = new double[n_ind];
                SquaredError = new double[n_ind];
                hidden = new double[n_ind][n_cov];
                skip = new double[n_ind][n_cov];
                fxzero = new double[n_ind][n_cov];
                fmed = new double[n_cov];
                S = new double[n_ind];
                alfa = new double[n_ind];
        }
}
```

Listagem A.12: Classe OutputANNTask que define o objeto onde ficam guardados os valores obtidos por cada uma das tarefas (treino, validação e teste) a serem utilizados por outras camadas.

```
package cgANN.Task;

public class TargetFunction {
        public static double dcdy(OutputANNTask out,String func,int ind,int epoca){
                double result=Double.POSITIVE_INFINITY;

                if(func=="QuadraticError"){

                result = (out.estimated[ind]-out.observed[ind]);

                }
                return result;
        }
}
```

Listagem A.13: Classe TargetFunction que implementa a função objetivo do modelo.

A.5 Quarta camada

A.5.1 Descrição

Nesta camada são inicializados os parâmetros, bem como criados os objetos que encapsulam os processos da terceira camada. Também nesta camada se pode obter, através do método *probe*, os valores de um qualquer ponto da rede neuronal, onde se incluem os valores das funções parciais bem como da saída da rede.

A.5.2 Código Fonte

```
package cgANN.GANN;

import java.util.ArrayList;
import java.util.List;

import cgANN.Task.ANNTask;
import cgANN.Task.OutputANNTask;
import cgModel.InitW;
import cgSetup.Cromossom;
import cgSetup.Setup;
import cgTREES.NodeLogTree;
import cgUtils.*;
public class GANNTask extends ANNTask{
        public GANN gann;
        public GANNTask(Setup setup,Cromossom cromossom,Sample sample,int seed) throws
            Exception{
                super(setup,cromossom);
                ann = new GANN(setup.numberofcov,1,setup.maxepocas,cromossom.inputgenes,
                    setup.activefunction,setup.linkfunction,cromossom.alfa,setup.
                    fixweightsnode2, setup.useIntercept,setup.identifiability);
                defineANNTask(ann,setup);
                InitW winit = new InitW(ann.n_syn,setup.cromossom_initial.inputgenes,
                    null,setup.useGLM);

                winit.getAOGLMWeights(sample,setup.rengine,cromossom.alfa,true,seed);

                W[0] = ann.LoadInitialWeights(winit,0).Wfinal;

        }
        public OutputANNTask probe(OutputANNTask out, int ind) {
                GANN gann = (GANN) ann;
                for(int i=0; i<out.n_cov;i++){
                    out.fx[ind][i]=gann.node2.inputsignal[i];
                    out.skip[ind][i]=gann.supernode1.node1[i].skip;
                    out.hidden[ind][i]=gann.supernode1.node1[i].hidden;
                    out.betax[ind][i]=gann.supernode1.node1[i].betax;
                    out.fxzero[ind][i]=gann.supernode1.node1[i].fxzero;
                }
                out.beta0[ind]=gann.node2.beta0;
                out.S[ind]=gann.node2.S;
                return(out);
        }
        public void setIdentifiabilityConstraints(double[] MF, int n_ind) {
                GANN gann = (GANN) ann;
                for(int i=0; i<gann.n_in;i++) gann.supernode1.node1[i].fmed=MF[i];
                gann.node2.sumfmed=Mathf.SumArray(MF);
        }

        public double getPartialFunctionsOutput(int cov){
                GANN gann = (GANN) ann;
                return(gann.supernode1.node1[cov].S);
        }
        public double[] getANNparams(int epoca,int n_cov){
                GANN gann = (GANN) ann;
                double[] params = new double[n_cov];
                for(int i=0;i<n_cov;i++){
                                params[i]=gann.node2.inputsignal[i];
                }
                return(params);
        }
```

```
        public void setANNparams(double[][] params){
                GANN gann = (GANN) ann;
                for(int i=0;i<params.length;i++){
                        double[] S = params[i];
                        for(int j=0;j<S.length-2;j++) gann.supernode1.node1[i].Sb[j]=S[j
                            ];
                        gann.supernode1.node1[i].S=S[S.length-2];
                        gann.inputerror[0]=S[S.length-1];
                }
        }
}
```

Listagem A.14: Classe GANNTask que implementa as tarefas definidas na camada anterior para o caso específico da GANN Aranda.

```
package cgANN.GANN;

import java.util.ArrayList;
import java.util.List;

import org.apache.commons.lang.ArrayUtils;

import cgANN.Task.ANNTask;
import cgANN.Task.OutputANNTask;
import cgModel.InitW;
import cgSetup.Cromossom;
import cgSetup.Setup;
import cgUtils.*;
public class GANNnonparamTask extends ANNTask{
        public GANNnonparam gannnp;
        public GANNnonparamTask(Setup setup,Cromossom cromossom,Sample sample,int seed)
            throws Exception{
                super(setup,cromossom);
                ann = new GANNnonparam(setup.numberofcov,1,setup.maxepocas,cromossom.
                    inputgenes,cromossom.nhl2,setup.activefunction,setup.linkfunction,
                    cromossom.alfa,setup.fixweightsnode2,setup.identifiability);
                defineANNTask(ann,setup);
                InitW winit = new InitW(ann.n_syn,setup.cromossom_initial.inputgenes,
                    setup.cromossom_initial.nhl2,setup.useGLM);

                    winit.getAOGLMWeights(sample,setup.rengine,cromossom.alfa,true,
                        seed);

                W[0] = ann.LoadInitialWeights(winit,0).Wfinal;
        }
        public OutputANNTask probe(OutputANNTask out, int ind) {
                GANNnonparam gannnp = (GANNnonparam) ann;
                for(int i=0; i<out.n_cov;i++){
                   out.fx[ind][i]=gannnp.supernode1.outputsignal[i];
                   out.skip[ind][i]=gannnp.supernode1.node1[i].skip;
                   out.hidden[ind][i]=gannnp.supernode1.node1[i].hidden;
                   out.betax[ind][i]=gannnp.supernode1.node1[i].betax;
                   out.fxzero[ind][i]=gannnp.supernode1.node1[i].fxzero;
                }
                out.beta0[ind]=gannnp.node2.beta0;
                out.beta1[ind]=gannnp.node3.beta1;
                out.S[ind]=gannnp.node2.S;
                return(out);
        }
        public void setIdentifiabilityConstraints(double[] MF, int n_ind) {
                GANNnonparam gannnp = (GANNnonparam) ann;
                for(int i=0; i<gannnp.n_in;i++) gannnp.supernode1.node1[i].fmed=MF[i];
                    //f[i]/n_ind;
                gannnp.node2.sumfmed=Mathf.SumArray(MF);
        }
        public void unsetIdentifiabilityConstraints(double[] MF, int n_ind) {
                GANNnonparam gannnp = (GANNnonparam) ann;
                for(int i=0; i<gannnp.n_in;i++) gannnp.supernode1.node1[i].fmed=-MF[i];
                    //f[i]/n_ind;
                gannnp.node2.sumfmed=-Mathf.SumArray(MF);
        }
        public double getPartialFunctionsOutput(int cov){
                GANNnonparam gannnp = (GANNnonparam) ann;
                return(gannnp.supernode1.node1[cov].S);
        }
        public double[] getANNparams(int epoca,int n_cov){
                GANNnonparam gannnp = (GANNnonparam) ann;
                double[] params = new double[n_cov];
                if(epoca==67){
                        for(int i=0;i<n_cov;i++){
                                params[i]=gannnp.supernode1.node1[i].S;
                        }
                }
                return(params);
        }
        public void setANNparams(double[][] params){
                GANNnonparam gannnp = (GANNnonparam) ann;
                for(int i=0;i<params.length;i++){
                        double[] S = params[i];
                        for(int j=0;j<S.length-1;j++) gannnp.supernode1.node1[i].Sb[j]=S
                            [j];
                        gannnp.supernode1.node1[i].S=S[S.length-1];
                }
        }
}
```

Listagem A.15: Classe GANNnonparamTask que implementa as tarefas definidas na camada anterior para o caso específico da GANN Não paramétrica.

A.6 Quinta camada

A.6.1 Descrição

Nesta camada encontram-se definidas classes que possibilitam a avaliação e validação do modelo a partir da informação obtida das camadas anteriores.

A.6.2 Código Fonte

```
package cgModel;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.rosuda.REngine.REngineException;
import org.apache.commons.lang.ArrayUtils;
import org.apache.commons.math3.util.Pair;

import cgANN.GANN.GANNTask;
import cgANN.GANN.GANNnonparamTask;
import cgANN.MLP.MLPTask;
import cgSetup.Cromossom;
import cgSetup.Setup;
import cgTREES.NodeTree;
import cgUtils.CGFunctions;
import cgUtils.Mathf;
public class AssessModel {
        public static NodeTree Evaluation(Setup setup, int seed, int[]
            Newcromossom, int ID,int parent_ID,int generation) throws
            Exception, IOException{
                Cromossom cromossom = new Cromossom(Newcromossom, setup.alfa
                    , setup.numberofcov, setup.allele_min, setup.allele_max)
                    ;
                OutputModel[] modelresult = new OutputModel[setup.kfolds];
                modelresult = CrossValidation(setup,cromossom,seed,
                    generation);
                NodeTree modelevaluation = getErrorMetrics(setup,cromossom,
                    ID,parent_ID,generation,modelresult);
                return(modelevaluation);
        }
        public static OutputModel[] CrossValidation(Setup setup,Cromossom
            cromossom,int seed,int generation) throws Exception, IOException
            , REngineException{
                OutputModel[] out = new OutputModel[setup.kfolds];
                //Cross Validation
                for(int k=0;k<setup.kfolds;k++){
                        switch(setup.topology){
                           case "AOGLM":
```

```
                                out[k] = ModelAOGLM.Output(setup.rengine,
                                    setup.sample[k],setup.kfolds,
                                    cromossom.alfa);
                            break;
                            case "GANN":
                                GANNTask ganntask = new GANNTask(setup,
                                    cromossom,setup.sample[k],seed);
                                out[k] = ModelANN.Output(setup,ganntask,
                                    setup.sample[k],setup.
                                    parameter_selection,setup.maxepocas);
                                CGFunctions.Savexls("src/output/bd/values
                                    /"+"testetreino"+k+".xls", "train",
                                    setup.sample[k].traindata);

                                for(int epoca=0;epoca<setup.maxepocas;
                                    epoca+=10) setup.logs.add(Integer.
                                    toString(generation),Integer.toString
                                    (k), cromossom.allgenes,"Weights","-"
                                    ,ganntask.W[epoca]);
                            break;
                            case "GANNnonparam":
                                GANNnonparamTask gannnptask = new
                                    GANNnonparamTask(setup,cromossom,
                                    setup.sample[k],seed);
                                out[k] = ModelANN.Output(setup,gannnptask
                                    ,setup.sample[k],setup.
                                    parameter_selection,setup.maxepocas);
                                for(int epoca=0;epoca<setup.maxepocas;
                                    epoca+=10) setup.logs.add(Integer.
                                    toString(generation),Integer.toString
                                    (k), cromossom.allgenes,"Weights","-"
                                    ,gannnptask.W[epoca]);
                            break;
                            case "MLP":
                                MLPTask mlptask = new MLPTask(setup,
                                    cromossom,setup.sample[k],seed);
                                out[k] = ModelANN.Output(setup,mlptask,
                                    setup.sample[k],setup.
                                    parameter_selection,setup.maxepocas);
                                for(int epoca=0;epoca<setup.maxepocas;
                                    epoca+=10) setup.logs.add(Integer.
                                    toString(generation),Integer.toString
                                    (k), cromossom.allgenes,"Weights","-"
                                    ,mlptask.W[epoca]);
                            break;
                        }
                        CGFunctions.Print("   k="+k+"\n");
                    }
                        return out;
                }
                public static NodeTree getErrorMetrics(Setup setup, Cromossom
                    cromossom, int ID,int parent_ID,int generation,OutputModel[] out
                    ) throws Exception, IOException{
                        /*CALCULATE NODE OUTPUT*/
                        double invMSEval = Mathf.algsifr(1/Mathf.MSE(GetValOutput(
                            out).getKey()),setup.numsigf);
                        double invMSEtest = Mathf.algsifr(1/Mathf.MSE(GetTestOutput(
                            out).getKey()),setup.numsigf);
                        int numberofones = GetNumberofOnes(out);
                        int numberofzeros = GetNumberofZeros(out);
                        double AUCtest = Mathf.algsifr(Mathf.AUC(GetTestOutput(out).
                            getKey()),setup.numsigf);
```

```
            double AUCval = Mathf.algsifr(Mathf.AUC(GetValOutput(out).
                getKey()),setup.numsigf);
            int NumberofParams = GetNumberofParams(out);
            Pair<double[][],String[]> outval = GetValOutput(out);
            Pair<double[][],String[]> outtest = GetTestOutput(out);
            double[][] outtrain = GetTrainingOutput(out,0); //just
                outputs the first training sample kfold=0
            double alfa = cromossom.alfa;
            NodeTree node = new NodeTree(
                                                        ID,
                                                        parent_ID,
                                                        generation,
                                                        invMSEval,
                                                        cromossom.
                                                            allgenes,
                                                        invMSEtest,
                                                    invMSEval,
                                                    numberofzeros,
                                                    numberofones,
                                                    AUCval,
                                                    AUCtest,
                                                    setup.seed1actual,
                                                    setup.seed2actual,
                                                    setup.seed3actual,
                                                    alfa,
                                                    NumberofParams,
                                                    outval,
                                                    outtest,
                                                    outtrain,
                                                    setup.min_sample,
                                                    setup.max_sample,
                                                    setup.
                                                        simulated_data
                                                        .getValue()
                                                );
            node.print();
            return node;

    }
    public static double[][] GetTrainingOutput(OutputModel[] out, int k)
        {
            int n_cov = out[k].outtraining[out[k].stop].fx[0].length;
            double[][] trainout = null;
            trainout = CGFunctions.JoinArraybyCol(trainout,out[k].
                outtraining[out[k].stop].estimated);
            trainout = CGFunctions.JoinArraybyCol(trainout,out[k].
                outtraining[out[k].stop].observed);
            trainout = CGFunctions.JoinArraybyCol(trainout,out[k].
                outtraining[out[k].stop].S);
            trainout = CGFunctions.JoinArraybyCol(trainout,out[k].
                outtraining[out[k].stop].beta0);
    for(int i=0;i<n_cov;i++){
            trainout = CGFunctions.JoinArraybyCol(trainout,CGFunctions.
                GetColumn(out[k].outtraining[out[k].stop].fx,i));
    }
    return trainout;
    }
    public static double[][] GetValInput(OutputModel[] out){
            double[][] result = null;
            int maxslices = out.length;
            for(int i=0;i<maxslices;i++){
```

```
                result = CGFunctions.JoinArray(result, out[i].
                    val_input);
        }
        return(result);
}
public static Pair<double[][],String[]> GetValOutput(OutputModel[]
    out){
        double[] estimatedval = null;
        double[] observedval = null;
        double[] lpval = null;
        double[] beta0val = null;
        double[] beta1val = null;
        double[][] inputval = null;
        double[][] fxval = null;
        double[][] fxzeroval = null;
        int maxslices = out.length;
        for(int i=0;i<maxslices;i++){
    estimatedval = CGFunctions.JoinArray(estimatedval, out[i].
        outvalidating[out[i].stop].estimated);
    observedval = CGFunctions.JoinArray(observedval, out[i].
        outvalidating[out[i].stop].observed);
    lpval = CGFunctions.JoinArray(lpval, out[i].outvalidating[out[i
        ].stop].S);
    beta0val = CGFunctions.JoinArray(beta0val, out[i].val_beta0);
    beta1val = CGFunctions.JoinArray(beta1val, out[i].val_beta1);
    inputval = CGFunctions.JoinArray(inputval,out[i].val_input);
    fxval = CGFunctions.JoinArray(fxval,out[i].val_fx);
    fxzeroval = CGFunctions.JoinArray(fxzeroval,out[i].val_fxzero);
}
        String[] title1 = {"estimado","observado","lp","beta0","
            beta1"};
        String[] title2 = new String[inputval[0].length];
        String[] title3 = new String[fxval[0].length];
        String[] title7 = new String[fxzeroval[0].length];

        for(int i=1;i<inputval[0].length+1;i++) title2[i-1]="input"+
            i;
        for(int i=1;i<fxval[0].length+1;i++) title3[i-1]="fx"+i;
        for(int i=1;i<fxzeroval[0].length+1;i++) title7[i-1]="fxzero
            "+i;

        String[] title = (String[]) ArrayUtils.addAll(title1,title2)
            ;
        title = (String[]) ArrayUtils.addAll(title,title3);
        title = (String[]) ArrayUtils.addAll(title,title7);

        double[][] valout = CGFunctions.JoinArraybyCol(estimatedval,
            observedval,lpval,beta0val,beta1val);
        valout = CGFunctions.JoinArraybyCol(valout,inputval);
        valout = CGFunctions.JoinArraybyCol(valout,fxval);
        valout = CGFunctions.JoinArraybyCol(valout,fxzeroval);
        return (new Pair(valout,title));


}
public static double[][] GetTestInput(OutputModel[] out){
        double[][] result = null;
        int maxslices = out.length;
        for(int i=0;i<maxslices;i++){
                result = CGFunctions.JoinArray(result, out[i].
                    test_input);
        }
```

```
        return(result);
    }
    public static Pair<double[][],String[]> GetTestOutput(OutputModel[]
        out){
        double[] estimatedtest = null;
        double[] observedtest = null;
        double[] lptest = null;
        double[] beta0test = null;
        double[] beta1test = null;
        double[][] fxtest = null;
        double[][] fxzerotest = null;
        double[][] inputtest = null;
        int maxslices = out.length;

        for(int i=0;i<maxslices;i++){
    estimatedtest = CGFunctions.JoinArray(estimatedtest, out[i].
        test_estimated);
    observedtest = CGFunctions.JoinArray(observedtest, out[i].
        test_observed);
    lptest = CGFunctions.JoinArray(lptest, out[i].test_S);
    beta0test = CGFunctions.JoinArray(beta0test, out[i].test_beta0);
    beta1test = CGFunctions.JoinArray(beta1test, out[i].test_beta1);
    inputtest = CGFunctions.JoinArray(inputtest,out[i].test_input);
    fxtest = CGFunctions.JoinArray(fxtest,out[i].test_fx);
    fxzerotest = CGFunctions.JoinArray(fxzerotest,out[i].test_fxzero
        );
    }
        String[] title1 = {"estimado","observado","lp","beta0","
            beta1"};
        String[] title2 = new String[inputtest[0].length];
        String[] title3 = new String[fxtest[0].length];
        String[] title7 = new String[fxzerotest[0].length];

        for(int i=1;i<inputtest[0].length+1;i++) title2[i-1]="input"
            +i;
        for(int i=1;i<fxtest[0].length+1;i++) title3[i-1]="fx"+i;
        for(int i=1;i<fxzerotest[0].length+1;i++) title7[i-1]="
            fxzero"+i;

        String[] title = (String[]) ArrayUtils.addAll(title1,title2)
            ;
        title = (String[]) ArrayUtils.addAll(title,title3);
        title = (String[]) ArrayUtils.addAll(title,title7);


        double[][] testout = CGFunctions.JoinArraybyCol(
            estimatedtest,observedtest,lptest,beta0test,beta1test);
        testout = CGFunctions.JoinArraybyCol(testout,inputtest);
        testout = CGFunctions.JoinArraybyCol(testout,fxtest);
        testout = CGFunctions.JoinArraybyCol(testout,fxzerotest);
        return (new Pair(testout,title));

    }
    public static int GetNumberofParams(OutputModel[] out){
        int numberofparams = out[0].n_syn;
        return numberofparams;
    }
    public static int GetNumberofInd(OutputModel[] out){
         return(GetTestOutput(out).getKey().length);
    }
    public static int GetNumberofOnes(OutputModel[] out){
        int result = 0;
```

```
                int maxslices = out.length;
                for(int i=0;i<maxslices;i++){
            result += (int) Mathf.SumArray(out[i].test_observed);
        }
                return(result);
        }
        public static int GetNumberofZeros(OutputModel[] out){
                 return (int) (GetNumberofInd(out) - GetNumberofOnes(out));
        }
}
```

Listagem A.16: Classe AssessModel que encapsula um conjunto de métodos que permitem a avaliação e validação do modelo.

```
package cgModel;

rg.rosuda.JRI.Rengine;

gSetup.Exit;
gUtils.CGFunctions;
gUtils.CRAN;
gUtils.Mathf;
gUtils.Sample;

lass InitW {
public double beta0;
public double[] betas;
public double[] betas_without_intercept;
public double[] W;
public double[][] trainvaldata;
public double[] NewW;
public double[] Wfinal;
public int n_cov;
public int n_syn;
public int[] nhl1_old;
public int[] nhl2_old;
public int pos_w,pos_old;
public String source;
public boolean useGLM;
public InitW(int Newn_syn,int[] Newnhl_old,int[] Newnhlout_old, boolean NewuseGLM){
        n_syn = Newn_syn;
        nhl1_old = Newnhl_old;
        nhl2_old = Newnhlout_old;
        pos_w=0;
    pos_old=0;
    W = new double[n_syn];
    useGLM = NewuseGLM;
}
public void getAOGLMWeights(Sample sample,Rengine rengine, double Newalfa, boolean
    includeintercept,int seed) throws Exception{
        source = "AOGLM";
        W = Mathf.getNormalRandomNumbers(0,0.1,n_syn,seed);
        NewW = Mathf.getUniformRandomNumbers(-0.5,0.5,n_syn,seed);
        trainvaldata = CGFunctions.JoinArray(sample.traindata,sample.valdata);
        n_cov = trainvaldata[0].length-1;
        if(useGLM){
                /*CALCULATE WINIT_BASIC*/
                int n_betas = n_cov+1;
                betas = new double[n_betas];
                betas_without_intercept = new double[n_betas-1];
                betas = CRAN.GetAOGLMbetas(rengine, trainvaldata, Newalfa);
                beta0 = betas[0];
                for(int i=0;i<n_betas;i++){
                        if(Double.isNaN(betas[i])){
                            betas[i]=NewW[pos_w];
                            pos_w+=1;
                        }
                }

                System.arraycopy(betas, 1, betas_without_intercept, 0,
                     betas_without_intercept.length);
                if(includeintercept){
                                System.arraycopy(betas_without_intercept, 0, W, 0,
                                    betas_without_intercept.length);
                                W[n_betas-1]=beta0;
                }else{
                                System.arraycopy(betas_without_intercept, 0, W, 0,
                                    betas_without_intercept.length);
                }
        }
}
```

Listagem A.17: Classe InitW que serve para inicialização dos pesos sinápticos.

```
package cgModel;

ava.io.FileNotFoundException;
ava.io.IOException;
gANN.Task.ANNTask;
gANN.Task.OutputANNTask;
gSetup.Setup;
gUtils.CGFunctions;
gUtils.Plotting;
gUtils.Sample;

lass ModelANN {
public static OutputModel Output(Setup setup,ANNTask anntask_init,Sample sample, String
     method, int maxepocas) throws FileNotFoundException, IOException,
     CloneNotSupportedException{
         ANNTask anntask = (ANNTask) anntask_init.clone();
         OutputModel out = new OutputModel(maxepocas,anntask.n_syn,sample.n_cov,sample.
              n_ind);
    OutputANNTask[] outtraining = new OutputANNTask[maxepocas+1];
    OutputANNTask[] outvalidating = new OutputANNTask[maxepocas+1];
    OutputANNTask[] outtesting = new OutputANNTask[maxepocas+1];

    double[] xaxis = new double[maxepocas+1];
    for(int epoca=1;epoca<=maxepocas;epoca++){
                outtraining[epoca] = anntask.Training(sample.traindata,epoca);
            outvalidating[epoca] = anntask.Validating(sample.valdata,epoca);
            outtesting[epoca] = anntask.Testing(sample.testdata,epoca);
            anntask.UpdateWeights(setup,method,epoca);
            xaxis[epoca]=epoca;
         }
    out.stop = OutputModel.GetMin(outvalidating);
         CGFunctions.Print("    stop="+out.stop+"\n");
         out.outtraining = outtraining;
         out.outvalidating = outvalidating;
         out.outtesting = outtesting;
         out.W = anntask.W[out.stop-1];
         out.n_syn = anntask.n_syn;
         out.train_error = outtraining[out.stop].MSE;
         out.val_error = outvalidating[out.stop].MSE;
         out.test_error = outtesting[out.stop].MSE;
         out.test_observed = outtesting[out.stop].observed;
         out.test_estimated = outtesting[out.stop].estimated;
         out.test_fx = out.outtesting[out.stop].fx;
         out.test_hidden = out.outtesting[out.stop].hidden;
         out.test_skip = out.outtesting[out.stop].skip;
         out.test_input = sample.testdata;
         out.val_input = sample.valdata;
         out.test_S = out.outtesting[out.stop].S;
         out.test_beta0 = out.outtesting[out.stop].beta0;
         out.test_beta1 = out.outtesting[out.stop].beta1;
         out.val_beta0 = out.outvalidating[out.stop].beta0;
         out.val_beta1 = out.outvalidating[out.stop].beta1;
         out.val_fx = out.outvalidating[out.stop].fx;
         out.val_fxzero = outvalidating[out.stop].fxzero;
         out.test_betax = out.outtesting[out.stop].betax;
         out.test_fxzero = out.outtesting[out.stop].fxzero;

    return out;
}
```

Listagem A.18: Classe ModelANN que providencia os resultados do desempenho da rede neuronal.

```
package cgModel;

gANN.Task.OutputANNTask;

lass OutputModel{
public OutputANNTask[] outtraining;
public OutputANNTask[] outvalidating;
public OutputANNTask[] outtesting;
public double[] W;
public double[] test_observed;
public double[] test_estimated;
public double[] test_S; //preditor linear
public double[][] test_fx;
public double[][] test_hidden;
public double[][] test_skip;
public double[][] test_input;
public double[][] val_input;
public double[][] val_fx;
public double[] test_beta0;
public double[] test_beta1;
public double[] val_beta0;
public double[] val_beta1;
public double[][] test_betax;
public double[][] test_fxzero;
public double[][] val_fxzero;
public double[] alfa;
public int stop;
public int[] nhl;
public int n_syn;
public int n_cov;
public int n_ind;
public double val_error,train_error,test_error;
public OutputModel(int maxepocas, int Newn_syn, int Newn_cov, int Newn_ind){
        outtraining = new OutputANNTask[maxepocas+1];
        outvalidating = new OutputANNTask[maxepocas+1];
        outtesting = new OutputANNTask[maxepocas+1];
        W = new double[n_syn];
        n_syn = Newn_syn;
        n_cov = Newn_cov;
        n_ind = Newn_ind;
        stop = 1;
}
public static double[] GetMSE(OutputANNTask[] anntask){
        double[] MSE = new double[anntask.length];
        MSE[0]=0;
        for(int i=1;i<anntask.length;i++){
                MSE[i]=anntask[i].MSE;
        }
        return MSE;
}
public static double[] GetEpochs(OutputANNTask[] anntask){
        double[] epocas = new double[anntask.length];
        epocas[0]=0;
        for(int i=1;i<anntask.length;i++){
                epocas[i]=i;
        }
        return epocas;
}
public static int GetMin(OutputANNTask[] anntask){
        int min=1;
        for(int i=2;i<anntask.length;i++){
                if(anntask[i].MSE<anntask[min].MSE){
                        min = i;
                }
        }
        return min;
}
```

Listagem A.19: Classe OutputModel que guarda os valores do desempenho do modelo a serem utilizados por outras camadas.

A.7 Sexta camada

A.7.1 Descrição

Esta camada é constituída por classes que implementam a procura do melhor modelo num espaço de procura previamente definido na camada acima.

A.7.2 Código Fonte

```
package cgSearch.Local;
import java.util.Arrays;

import cgModel.AssessModel;
import cgSearch.Operators.Generate_Population;
import cgSetup.Setup;
import cgTREES.ArrayofIntegersList;
import cgTREES.NodeTree;
import cgTREES.NodeTreeList;
import cgUtils.CGFunctions;

public class AutoANN {
        public NodeTreeList local_list;
        public AutoANN(Setup setup,int seed,int[] cromossom_init, NodeTreeList
             search_list, int NewID, int Newparent_ID, int step_ID,int generation_init)
             throws Exception{
                /*SETUP*/
                local_list = new NodeTreeList();

                int ID = NewID;
            int parent_ID = Newparent_ID;

            int[] cromossom_parent = cromossom_init.clone();
                for(int generation = generation_init; generation<setup.maxgeneration+
                     generation_init; generation++){
                        setup.logs.save(setup.outputpath+setup.outputfolder+"logs/",
                             "logs");//*logs
                        CGFunctions.Print(step_ID+".AUTO."+setup.topology+"."+
                             generation+" GENERATION="+generation+"\n");
                    CGFunctions.Print(" CROMOSSOM POPULATION = \n");
                    //Generate Population
                        ArrayofIntegersList cromossoms = Generate_Population.
                             rowIncrementalEvolution(setup.cromossom_basic,
                             cromossom_parent, local_list, setup.allele_min, setup.
                             allele_max, setup.increment, setup.improved_multistep);
                        cromossoms.print();
                        //Evaluation
                        for(int cromossom_id=0;cromossom_id<cromossoms.nodes.size();
                             cromossom_id++){
                                CGFunctions.Print("==================\n");
                             CGFunctions.Print(step_ID+".AUTO."+setup.topology+"."+
                                  generation+"."+cromossom_id+" ID="+ID);
                                     CGFunctions.Print("  CROMOSSOM = ");
                             CGFunctions.printArray(cromossoms.nodes.get(cromossom_id
                                  ));
                             CGFunctions.Print("cromossom_id="+cromossom_id+"\n");
                             int[] cromossom = cromossoms.nodes.get(cromossom_id);

                                NodeTree node = new NodeTree();
                    int index = search_list.alreadyExists(cromossom);
                    if(index<0){
                                node = AssessModel.Evaluation(setup,seed,cromossom,ID,
                                     parent_ID,generation);
                    }else{
                                CGFunctions.Print("\nNode already calculated generation=
                                     "+generation+"\n");
                                node = search_list.get(index).deepCopy();
                                node.ID = ID;
                                node.generation = generation;
                                node.parent_ID = parent_ID;
                                //node.time = System.currentTimeMillis();
                    }

                    if(Arrays.equals(cromossom,cromossom_init)) node.expanded=true;
                        local_list.add(node);
```

```
                    setup.outputdb.insertTable("Evaluation",node,setup.
                        inputfilename,"autogann",seed,-1,-1);
                    ID++;
                }
                if(setup.multistep_search){
                    int[] offspring = cgSearch.Operators.Crossover.
                        getMultiStepCromossom(local_list,parent_ID);
                    if(!(local_list.alreadyExists(offspring)>=0)){
                        CGFunctions.Print("1."+generation+"."+
                            cromossoms.nodes.size());
                            CGFunctions.Print("  CROMOSSOM = ");
                        CGFunctions.printArray(offspring);
                        CGFunctions.Print("cromossom_id="+
                            cromossoms.nodes.size()+"\n");

                        NodeTree node = new NodeTree();
                        int index = search_list.alreadyExists(
                            offspring);
                    if(index<0){
                        node = AssessModel.Evaluation(setup,seed
                            ,offspring,ID,parent_ID,generation);
                    }else{
                        CGFunctions.Print("\nNode already
                            calculated generation="+generation+"
                            \n");
                        node = search_list.get(index).deepCopy()
                            ;
                        node.ID = ID;
                        node.generation = generation;
                        node.parent_ID = parent_ID;
                        //node.time = System.currentTimeMillis()
                            ;
                    }
                        local_list.add(node);
                        //setup.outputdb.insertTable("Evaluation",node
                            ,setup.inputdb.filename,"autogann",seed
                            ,-1);
                        setup.outputdb.insertTable("Evaluation",node,
                            setup.inputfilename,"autogann",seed,-1,-1)
                            ;
                        ID++;
                    }
                }

                //Selection
                local_list.sortDescendent();

                //List of Best Nodes
                    setup.outputdb.insertTable("Bestmodels",local_list.get(0),
                        setup.inputfilename,"autogann",seed,-1,-1);

                int index_best_node = local_list.getNextNotExpandedNode();//
                    if there is no more expanded nodes index_best_node=0
                if(index_best_node==0){
                    if(local_list.direct_search){
                        local_list.best_direct_search_node=
                            local_list.nodes.get(
                            index_best_node).ID;
                    }
                }else{
                    local_list.direct_search=false;//From here the search goes
                        to secondary nodes
                if((!setup.exhausting_search)||(index_best_node==local_list.size()
                    )){
                        CGFunctions.Print("Finish AUTOANN "+step_ID
                            +"\n\n");
                        break;
                    }
                }
                NodeTree best_node = local_list.changeExpandedFlag(
                    index_best_node,true);

                cromossom_parent = best_node.cromossom.clone();
                parent_ID = best_node.ID;

            }
        }
}
```

Listagem A.20: Classe AutoANN que implementa o método de procura AutoGANN.

```
package cgSearch.Local;

import java.io.IOException;

import cgModel.AssessModel;
import cgSetup.Setup;
import cgTREES.NodeTree;
import cgTREES.NodeTreeList;

public class SingleSearch {
        public NodeTreeList listGANN;
    public SingleSearch(Setup setup,int seed, int[] cromossomGANN) throws IOException,
         Exception{
        listGANN = new NodeTreeList();
            NodeTree node = AssessModel.Evaluation(setup,seed,cromossomGANN,0,0,0);
            listGANN.add(node);
        listGANN.print();

    }
}
```

Listagem A.21: Classe SingleSearch que é utilizada nos casos em que apenas se pretende obter o s valores de desempenho para um modelo com determinadas características topológicas.

```
package cgSearch.Operators;

import java.util.Arrays;
import cgSetup.Setup;
import cgTREES.*;
import cgUtils.CGFunctions;
import cgUtils.Mathf;
import cgUtils.Rand_Number;
public class Generate_Population {
        public static ArrayofIntegersList rowIncrementalEvolution(int[] cromossom_basic,
            int[] cromossom_parent,NodeTreeList local_list,int[] allele_min, int[]
            allele_max, int increment, boolean improved_multistep){
                int numberofgenes = cromossom_parent.length;
                ArrayofIntegersList parents_pop = new ArrayofIntegersList();
                int population;
                int inc_max = 0;
                int inc=0;
                if(improved_multistep){
                        int[] dif = new int[allele_max.length];
                        population=0;
                        for(int i=0;i<allele_max.length;i++){
                          if(cromossom_parent[i]>=0){
                                  dif[i]=allele_max[i]-cromossom_parent[i];

                          }else{
                                  dif[i]=Math.abs(allele_min[i]-cromossom_parent[i]);

                          }

                        }
                        inc_max = Mathf.max(dif);
                        increment=0;
                }
            population = numberofgenes*2;
                int cromossom_id=0;
                if(Arrays.equals(cromossom_basic, cromossom_parent)){
                        parents_pop.add(cromossom_basic);
                }

                while((cromossom_id<population)||(increment<inc_max)){
                    if(inc==0){
                            inc=numberofgenes*2;
                            if(improved_multistep)increment+=1;
                            cromossom_id=0;
                    }
                    int[] cromossom = new int[numberofgenes];
                    for(int gene_id=0; gene_id<numberofgenes;gene_id++){
                            if(gene_id==cromossom_id){
                                    int incrementedcromossom = cromossom_parent[gene_id]-
                                        increment;
                                    if(incrementedcromossom>=allele_min[gene_id]){
                                       cromossom[gene_id]= incrementedcromossom;
                                    }else{
                                            cromossom[gene_id]=0;
                                    }
                            }
                            else if(gene_id==(cromossom_id-numberofgenes)){
                                    int incrementedcromossom = cromossom_parent[gene_id]+
                                        increment;
                                    if(incrementedcromossom<=allele_max[gene_id]){
                                       cromossom[gene_id]= incrementedcromossom;
                                    }else{
                                            cromossom[gene_id]=0;
                                    }
                            }
                            else if((gene_id!=cromossom_id)&&(gene_id!=(cromossom_id-
                                numberofgenes))){
                                    cromossom[gene_id]=cromossom_parent[gene_id];
                            }
                    }
                    cromossom_id+=1;
                    inc-=1;
                    if((local_list.alreadyExists(cromossom)<0)&&(parents_pop.
                        alreadyExists(cromossom)<0)){
                            parents_pop.add(cromossom);
                    }
                }
                if(parents_pop.nodes.size()==0){
                        CGFunctions.Print("************FINAL***************");
                }
                return parents_pop;
        }
```

```
        public static ArrayofIntegersList random(Rand_Number r,int population, Setup
            setup, int[] allele_max, int[] allele_min){
                    //setup=>numberofcov,exclusion_list,allele_min,allele_max
                    ArrayofIntegersList cromossom_parent = new ArrayofIntegersList();
                    int numberofgenes = setup.cromossom_basic.length;
                    int attempt=0;
                    int cromossom_id=0;
                    while(cromossom_id<population){
                        int[] cromossom = new int[numberofgenes];
                        for(int gene_id=0; gene_id<numberofgenes;gene_id++){
                                if(setup.exclusion_list==null){
                                        cromossom[gene_id] = r.GetRandInt(allele_min[
                                            gene_id],allele_max[gene_id]);
                                }else{
                                        if(setup.exclusion_list[gene_id]==0){
                                            cromossom[gene_id] = r.GetRandInt(
                                                allele_min[gene_id],allele_max[
                                                gene_id]);
                                        }else{
                                            cromossom[gene_id] = 0;
                                        }
                                }
                        }
                        if((cromossom_parent.alreadyExists(cromossom)<0)||attempt>
                            population){
                                cromossom_parent.add(cromossom);
                                cromossom_id+=1;
                                attempt=0;
                        }else{
                                attempt+=1;
                        }
                    }
                        return cromossom_parent;
        }
        public static ArrayofIntegersList simpleEvolution(int[] allele_min,int[]
            allele_max, int numberofbits){
                    ArrayofIntegersList cromossom_parent = new ArrayofIntegersList();
                    for(int i = allele_min[0];i<=allele_max[0];i++){
                        cromossom_parent.add(Mathf.Int2Bin(i, numberofbits));
                    }
                        return cromossom_parent;
        }
        public static void main(String[] args) throws Exception{
                NodeTreeList tree = new NodeTreeList();
        int[] a = {1,3,6};
        int[] b = {0,3,5};
        //Rengine r = new Rengine(null, false, null);
        Setup setup = new Setup(1);
        tree.add(new NodeTree());
        tree.add(new NodeTree());
        tree.add(new NodeTree());
                int[] cromossom_basic = {0,0,1};
                ArrayofIntegersList population = Generate_Population.
                    rowIncrementalEvolution(setup.cromossom_basic, cromossom_basic, tree
                    , setup.allele_min, setup.allele_max, setup.increment,true);
                population.print();
        }
}
```

Listagem A.22: Classe GeneratePopulation que implementa um operador que gera aleatoriamente uma população de modelos candidatos pertencentes a um determinado espaço de procura.

```
package cgSearch.Operators;

import cgTREES.*;

public class getNewPopulation {
        public static ArrayofIntegersList Elitism(NodeTreeList global_list,
             ArrayofIntegersList childs,int bestpop_number){
                    global_list.sortDescendent();
                    ArrayofIntegersList result = new ArrayofIntegersList();
                    result.nodes.addAll(childs.nodes);
                    for(int cromossom_id=0;cromossom_id<bestpop_number;cromossom_id++){
                            result.nodes.add(global_list.nodes.get(cromossom_id).
                                    cromossom);
                    }
                return(result);
        }
}
```

Listagem A.23: Classe getNewPopulation apara obter uma nova população de modelos candidatos pertencentes a um determinado espaço de procura.

```
package cgSetup;

import java.util.ArrayList;

import cgUtils.CGFunctions;
import cgUtils.Mathf;

public class Cromossom {
            public int[] allgenes;
            public int[] inputgenes;
            public int[] outputgenes;
            public double alfa;
            public int[] nhl2; //para o caso da não paramétrica
        public Cromossom(int[] cromossom, ArrayList<Double> Newalfa, int numberofcov,int
             [] allele_min,int[] allele_max){
            outputgenes = null;

            allgenes = new int[cromossom.length];
            allgenes = cromossom.clone();
            if(numberofcov>cromossom.length) Exit.withError(1);

            //Define inputgenes
            inputgenes = new int[numberofcov];
            //Define outputgenes
            outputgenes = new int[cromossom.length-numberofcov];
            for(int cromossom_id=0; cromossom_id<cromossom.length;cromossom_id++){
                    if(cromossom_id<numberofcov){
                            inputgenes[cromossom_id]=cromossom[cromossom_id];
                    }else{
                            outputgenes[cromossom_id-inputgenes.length]=cromossom[
                                 cromossom_id];
                    }
            }
            if(outputgenes.length>0){
                    int pos = cromossom.length-1;
                    if(allele_min[pos]==allele_max[pos]){
                            alfa = Newalfa.get(0);
                    }else{
                            //alfa = 0.2+3.6*Mathf.getNormalizednumber(allele_min[pos],
                                 allele_max[pos],cromossom[pos]);
                            alfa = 20+18*Mathf.getNormalizednumber(allele_min[pos],
                                 allele_max[pos],cromossom[pos]);
                            if(alfa<=0) alfa=0.1;
                    }

                    if(outputgenes.length>1){
                           nhl2 = new int[cromossom.length-numberofcov-1];
                           for(int cromossom_id=0; cromossom_id<outputgenes.length-1;
                                cromossom_id++){
                              nhl2[cromossom_id]=outputgenes[cromossom_id];
                           }
                    }
            }else{
                    alfa = Newalfa.get(0);
            }

        }


}
```

Listagem A.24: Classe Cromossom que define a topologia do modelo.

A.8 Sétima camada

A.8.1 Descrição

Esta camada é constituída por classes que implementam a parametrização da plataforma.

A.8.2 Código Fonte

```
package cgSetup;

import cgUtils.CGFunctions;

public class Exit {
        public static void withError(int number){
           switch(number){
              case 1:      CGFunctions.MessageBoxError("O número de genes é inferior ao
                   número de covariáveis"); break; //Cromossom
              case 2:      CGFunctions.MessageBoxError("O número de sinapses do componente é
                    superior à matriz de novos pesos"); break; //Node2
              case 3:      CGFunctions.MessageBoxError("O número de sinapses do componente é
                    diferente à matriz de novos pesos"); break; //InitW
        case 4:      CGFunctions.MessageBoxError("O número de genes é diferente do número
             de genes do cromossoma básico"); break; //Setup
        case 5:      CGFunctions.MessageBoxError("Erro na geração dos dados simulados");
             break; //Setup
        case 6:      CGFunctions.MessageBoxError("Número de vectores de parâmetros de
             Czado não é coincidente com seed2"); break; //Setup
        case 7:      CGFunctions.MessageBoxError("Indice de covariavel inexistente");
             break; //CGFunctions
        case 8:      CGFunctions.MessageBoxError("Dimensão do vetor titulo diferente da
             matriz fileout"); break; //CGFunctions
        case 9:      CGFunctions.MessageBoxError("Seed não existe"); break; //
             SimplifiedNodeTreeList
        case 10:     CGFunctions.MessageBoxError("Esta função não pode ser utilizada em
             regime diferente de KfoldCV ou Kfoldval"); break; //CGFunctions
           }
           System.exit(number);
    }
}
```

Listagem A.25: Classe Exit que implementa a saída do programa devido a determinados eventos.

```
package cgSetup;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;

import org.apache.commons.math3.util.Pair;
import org.rosuda.JRI.Rengine;
import org.rosuda.REngine.REngineException;

import com.rits.cloning.Cloner;

import cgModel.AssessModel;
import cgSearch.Global.geneticAlgorithm;
import cgSearch.Local.AutoANN;
import cgSearch.Local.SingleSearch;
import cgSearch.Operators.Generate_Population;
import cgSearch.Swarm.PSO;
import cgTREES.ArrayofIntegersList;
import cgTREES.NodeLogTree;
import cgTREES.NodeTree;
import cgTREES.NodeTreeList;
import cgTREES.SimplifiedNodeTree;
import cgTREES.SimplifiedNodeTreeList;
import cgUtils.CGFunctions;
import cgUtils.DB;
import cgUtils.Mathf;
import cgUtils.Rand_Number;
import cgUtils.Sample;

public class Setup {
        /*SETUP*/
        public int[] seed1,seed2,seed3;
        public int seed1max = 123;
        public int seed1min = 123;
        public int seed2max = 123;
        public int seed2min = 123;
        public int seed3max = 125;
        public int seed3min = 125;
        public int seed1actual, seed2actual, seed3actual;

        public int maxepocas=200;
        public int kfolds=5;
        public int maxslices;
        public int numsigf=3;//using in double calculations
        public int numberofbits=3; //used just in case of codification {0,1} and in MLP
             tolopology
        public int[] exclusion_list = null;


        /*LEARNING VARIABLES*/
        public double miuminus = 0.5;
        public double miuplus = 1.2;
        public double deltamin = 0.000001;
        public double deltamax  = 50;
        public double delta0 = 0.1;
        public double eta = 0.001;
        public double beta = 0.9;
        public double lamda = 0.001;
        public double beta2 = 0.1;

        /*NET CONFIG*/
        public String[] set_topology={"GANN","GANNnonparam","AOGLM","MLP"};
        public String[] set_parameter_selection={"lm","rprop","grad","grad2","matlab","
             null"};
        public String[] set_activationfunction={"sigmoid","tangh","mix","aranda"};
        public String[] set_targetfunc={"QuadraticError"};
        public String topology;
        public String parameter_selection;
        public String activefunction;
    public String targetfunc;
        public String linkfunction;
        public ArrayList<Double> alfa; //Aranda Ordaz parameter
        public double alfa_sample; //Aranda Ordaz parameter Sample generation
        public boolean fixweightsnode2 = true; //fixweightsnode2 = true weights fixed
        public boolean useGLM = true;
        public boolean only_with_skip_layer = false;
        public boolean identifiability = false;
```

```
        public ArrayList<Boolean> useIntercept;
        public int nhlmlp;

        /*SEARCH METHODS*/
        public String[] set_model_searchtype={"AutoANN","GA","DPSO","SingleSearch","
            ListSearch"};
        public String model_searchtype;

        /*AUTOGANN*/
        public boolean exhausting_search = true;
        public boolean multistep_search = true;
        public boolean improved_multistep = false;

        /*INIT OTHER VARIABLES*/
        public int numberofcov;
        //public double[][] Winit_basic;
        public int[] cromossom_basic;
        public Cromossom cromossom_initial;
        public int[] singlecromossom;
        SimplifiedNodeTreeList listofcromossoms = new SimplifiedNodeTreeList();
        public int numberofGenes;
        public int numberofextragenes = 0;

        /*GENETIC PARAMETERS*/
        public int maxgeneration = 700;
        public int population = 10;
        public int number_survivors;
        public int increment = 1;
        public String[] set_selection_method={"roulette","rank"};
        public String selection_method;
        public double prob_mut = 0.1;
        public int[] allele_min;
        public int[] allele_max;
        public String[] typeoffitness={"best_low"};
        public String[] typeofselection={"mix","elitism"};
        public String type_of_fitness;
        public String type_of_selection;
        public int spacedim = 3;
        public int[] velup;
        public int[] veldown;
        public double weight = 2;
        public double c1 = 2;
        public double c2 = 2;
        public double a = 0.9;
        public int[][] gene_position_value;

        /*OUTPUT and INPUT PATH FOLDER*/
        public String[] set_inputtype={"access","simulated_data","simulated_data_aranda"
            ,"simulated_data2","real_data"};
        public String[] set_outputtype={"access"};
        public String inputpath="src/input/";
        public String inputfolder="amostra_total/";
        public String outputpath="src/output/";
        public String outputfolder="bd/";
        public String inputfilename="icu18.mdb";
        public String inputtype;
        public String outputfilename = "output.accdb";
        public String outputtype;

        /*DATABASE SELECTION AND SAMPLE NORMALIZATION*/
        public DB outputdb;
        public DB inputdb;
        public String table="amostra_total";
        public String column ="all"; //"all" to be entire table
        public double criterium = 123;
        public Sample total_sample;
        public Sample[] sample;
        public int[] excludedcolumns = {0};
        public String[] set_bootstrap =  {"none","binary","bc"};
        public String bootstrap = set_bootstrap[0];

        /*EVALUATION*/
        public String[] set_evaluation = {"KFoldCV","HoldOutCV","InSample","KFoldval"};
        public String evaluation;

        /*SIMULATED DATA*/
    public int n  = 1000;
    public String[] set_xdist ={"normal","uniforme","uniforme2"};
    public String xdist;
    public Pair<double[][], double[][]> simulated_data;
    public ArrayList<double[]> czado; //Czado parameters
    public double[] min_sample;
    public double[] max_sample;
```

```
/*ODDS RATIO*/
public int n_points = 100; //number of points
public int col = 0; //covariable
public int bootmax = 1;
public int[] smoothedcromossom;
public double[] smoothed_test_pi;

    /*LOGS*/
    public NodeLogTree logs;

    /*RENGINE*/
    public Rengine rengine;

    public Setup(int config) throws Exception{
    /*SETUP*/

            /*SEARCH TYPE*/
            model_searchtype=set_model_searchtype[0]; //{"AutoANN","GA","DPSO","
                SingleSearch"}
            /*NET CONFIG*/
        topology = set_topology[0];//{"GANN","GANNnonparam","AOGLM"}
            parameter_selection = set_parameter_selection[0];//{"lm","rprop","grad
                ","grad2","matlab","null"}
            activefunction = set_activationfunction[1];//{"sigmoid","tangh","mix"}
        targetfunc = set_targetfunc[0];//{"QuadraticError"}
            linkfunction = set_activationfunction[0];//{"sigmoid","tangh","mix"}
            //alfa = 1;
            alfa = new ArrayList();
            useIntercept = new ArrayList();
            useIntercept.add(false); //Node1 betax
            useIntercept.add(true); //Node1 fxintercept tgh(beta)
            useIntercept.add(true); //Node2

            /*DATABASE SELECTION*/
        inputtype = set_inputtype[1]; //{"access","simulated_data"}
            outputtype = set_outputtype[0];//{"access"}

            /*EVALUATION*/
            evaluation = set_evaluation[0]; //{"KFoldCV","HoldOutCV"};

            /*SIMULATED DATA*/
        xdist=set_xdist[0];
        czado = new ArrayList();

            /*GENETIC and DPSO PARAMETERS*/

            selection_method = set_selection_method[0];
            type_of_fitness=typeoffitness[0]; //{"best_low"}
        type_of_selection=typeofselection[1]; //{"mix","elitism"}
            exclusion_list = null;
            switch(config){


                        case 1:
                            //Selecionar melhor fckbpv com teste
                              seed1max = 124;
                    seed1min = 124;
                    seed2max = 124;
                    seed2min = 124;
                    spacedim = 4;
                    numsigf = 7;
                        maxgeneration = 3000;
                    //maxgeneration = 3;
                        linkfunction = set_activationfunction[3];//{"sigmoid","
                            tangh","mix","aranda"}
                        topology = set_topology[0];//{"GANN","GANNnonparam","AOGLM
                            "} 0
                        model_searchtype = set_model_searchtype[0];//{"AutoANN","
                            GA","DPSO","SingleSearch","ListSearch"}
                        inputtype = set_inputtype[0]; //{"access","simulated_data
                            "} 3
                        parameter_selection = set_parameter_selection[0];//{"lm","
                            rprop","grad","grad2","matlab","null"}
                        evaluation = set_evaluation[0]; //{"KFoldCV","HoldOutCV","
                            InSample"};
                        useGLM = false;
                        fixweightsnode2 = true;
                            numberofextragenes = 1;
                            xdist=set_xdist[1];//{"normal","uniforme"}
                            useIntercept.set(0, false);
                            useIntercept.set(1, true);
                            useIntercept.set(2, true);
```

```
                                    alfa.add(10000000000000000000.0);
                                    multistep_search = true;
                                    only_with_skip_layer = true;
                                    improved_multistep=false;
                                    bootstrap="none";
                                    inputfilename="fckbpv.mdb";
                                    exclusion_list = new int[5];
                                    exclusion_list[3]=1;
                        break;


                            default:

                            break;
                    }

                /*LOGS*/
                logs = new NodeLogTree();
                rengine = new Rengine(null, false, null);
                    /*SEED Definition*/
                prepareSeed();
                    /*PREPARE WORK FOLDERS*/
                prepareOutput();
                /*PREPARE OUTPUT DATABASE*/
                prepareOutputDB();
                if((inputtype!=set_inputtype[0])&&(inputtype!=set_inputtype[4])){
                    //double interval = 0.5;
                    if((inputtype==set_inputtype[1])||(inputtype==set_inputtype[2])||(
                        inputtype==set_inputtype[3])){
                            numberofcov = 2;
                    }else{
                            numberofcov = 3;
                    }

                    //czado_param = prepareSimulatedData(czado_param_ini,czado_param_end,
                        interval);
                    //seed2min=0;seed2max=czado_param.size();
                }else{
                    /*PREPARE INPUT DATABASE*/

                        prepareInputDB(seed1min);
                }
                /*PREPARE CROMOSSOM BASIC*/
                setCromossomBasic();
    }
        public void setCromossomBasic(){
                numberofGenes = numberofcov + numberofextragenes;
                cromossom_basic = new int[numberofGenes];
                Arrays.fill(cromossom_basic, 0);
        }
        public void prepareSeed(){
                seed1 = new int[seed1max-seed1min+1];
                seed2 = new int[seed2max-seed2min+1];
                seed3 = new int[seed3max-seed3min+1];
                for(int seed1_id=0;seed1_id<seed1.length;seed1_id++){
                  for(int seed2_id=0;seed2_id<seed2.length;seed2_id++){
                        for(int seed3_id=0;seed3_id<seed3.length;seed3_id++){
                          seed1[seed1_id]=seed1_id+seed1min;
                          seed2[seed2_id]=seed2_id+seed2min;
                          seed3[seed3_id]=seed3_id+seed3min;
                        }
                  }
                }
        }
        public void prepareOutputDB() throws IOException, Exception{
        if(DB.create(outputpath+outputfolder,outputfilename,outputtype)){
            outputdb = new DB(outputpath+outputfolder,"output.accdb",outputtype);
            outputdb.prepare("Evaluation");
            outputdb.prepare("Bestmodels");
        }
    }
    public void prepareInputDB(int seed1){
        try {
                double[][] alldata = null;
            inputdb = new DB(inputpath+inputfolder,inputfilename,inputtype);
            alldata = inputdb.getData(this, inputtype,seed1);
            total_sample = new Sample(alldata);
            numberofcov = total_sample.n_cov;
                } catch (Exception e) {
                        CGFunctions.Print("Input Database does not exist");
                        e.printStackTrace();
                }
    }
```

```
    public void prepareSamples(int seed,int pos) throws FileNotFoundException,
         IOException, REngineException{
        if(inputtype!=set_inputtype[0]&&inputtype!=set_inputtype[4]){
                if(inputtype==set_inputtype[1]){
                        simulated_data = CGFunctions.getCzadoSimulatedData(seed,n,czado.
                             get(pos)[0],czado.get(pos)[1],czado.get(pos)[2],xdist); //
                             mudar
                    inputfilename="sd_czado_niu0_"+Mathf.algsifr(czado.get(pos)[0],
                         numsigf)+"_psi1_"+Mathf.algsifr(czado.get(pos)[1],numsigf)+"
                         _psi2_"+Mathf.algsifr(czado.get(pos)[2],numsigf);
                }
                if(inputtype==set_inputtype[2]){
                        simulated_data = CGFunctions.getArandaSimulatedData(seed,n,
                             alfa_sample,xdist); //mudar
                    inputfilename="sd_aranda_psi_"+Mathf.algsifr(alfa_sample,numsigf);
                }
                if(inputtype==set_inputtype[3]){
                        simulated_data = CGFunctions.getSimulatedData(seed,n,alfa_sample
                             ,xdist); //mudar
                    inputfilename="sd_generic_"+Mathf.algsifr(alfa_sample,numsigf);
                }
                total_sample = new Sample(simulated_data.getKey());

        }else{
                double[][] dummy = new double[total_sample.n_ind][3];
                double[] dummy2 = new double[total_sample.n_ind];
                simulated_data = new Pair(dummy,CGFunctions.JoinArraybyCol(dummy2,dummy2
                     ,dummy2,dummy2,dummy2));

                total_sample = new Sample(CGFunctions.getRealData(seed, this));
        }
        min_sample = new double[numberofcov];
                max_sample = new double[numberofcov];
                for(int cov=0;cov<numberofcov;cov++){
                        double[] x=CGFunctions.GetColumn(total_sample.alldata,cov);
                        min_sample[cov] = Mathf.min(x);
                        max_sample[cov] = Mathf.max(x);
                }
        /*SAMPLE NORMALIZATION*/
                //total_sample.Shuffle(seed);
        CGFunctions.Savexls(outputpath+outputfolder+"/lists/inputdata"+seed+"orig.xls",
             "inputdata", total_sample.alldata);
            total_sample.Normalize();
            if(bootstrap!="none") total_sample.Bootstrap(seed);
            CGFunctions.Savexls(outputpath+outputfolder+"/lists/inputdata"+seed+".xls",
                 "inputdata", total_sample.alldata);
            switch(evaluation){
                  case "KFoldCV":
                             kfolds=5;
                             maxslices = kfolds;
                  break;
                  case "HoldOutCV":
                             kfolds = 1;
                             maxslices = 3;
                  break;
                  case "InSample":
                             kfolds = 1;
                             maxslices = 5;
                  break;
                  case "KFoldval":
                             maxslices = kfolds;
                             maxslices = 5;
                  break;
                }
            sample = total_sample.GetSamplesCV(evaluation,kfolds,maxslices);
            maxslices = total_sample.maxslices;
    }
    public void prepareOutput(){
        try {
                Path path = FileSystems.getDefault().getPath(outputpath+"/"+outputfolder
                     );
                Path path1 = FileSystems.getDefault().getPath(outputpath+"/"+
                     outputfolder+"/values");
                Path path2 = FileSystems.getDefault().getPath(outputpath+"/"+
                     outputfolder+"/logs");
                Path path3 = FileSystems.getDefault().getPath(outputpath+"/"+
                     outputfolder+"/lists");
                Path path4 = FileSystems.getDefault().getPath(outputpath+"/"+
                     outputfolder,outputfilename);
                Path path5 = FileSystems.getDefault().getPath(outputpath+"/"+
                     outputfolder+"/lists","inputdata.xls");

                Files.deleteIfExists(path5);
```

```
            Files.deleteIfExists(path4);
            Files.deleteIfExists(path3);
            Files.deleteIfExists(path2);
            Files.deleteIfExists(path1);
            Files.deleteIfExists(path);
        } catch (IOException | SecurityException e) {
            System.err.println(e);
        }
        if(!(new File(outputpath+"/"+outputfolder)).mkdirs()) {
                CGFunctions.Print("Directory creation failed");
                System.exit(0);
        }
        if(!(new File(outputpath+"/"+outputfolder+"/values")).mkdirs()) CGFunctions.
            Print("Directory creation failed");
        if(!(new File(outputpath+"/"+outputfolder+"/logs")).mkdirs()) CGFunctions.Print(
            "Directory creation failed");
        if(!(new File(outputpath+"/"+outputfolder+"/lists")).mkdirs()) CGFunctions.Print
            ("Directory creation failed");
    }
        public void defineSearchParams() {


                allele_min = new int[numberofGenes];  /*need to update allele_min and
                    allele_max*/
                allele_max = new int[numberofGenes];

                cromossom_initial = new Cromossom(cromossom_basic, alfa, numberofcov,
                    allele_min, allele_max);


                velup = new int[numberofGenes];
                veldown = new int[numberofGenes];

                /*GENETIC and DPSO PARAMETERS*/

                number_survivors = (int) Math.ceil(3*population/5);
                for(int i=0;i<numberofGenes;i++){
                        switch(topology){
                         case "MLP":
                                 if(i<numberofGenes-1){
                                                 allele_min[i] = 0;
                                         allele_max[i] = 0;
                                 }else{
                                                 allele_min[i] = 0;
                                         allele_max[i] = maxgeneration;
                                 }
                         break;
                         case "GANNnonparam":
                                 allele_min[i] = 0;
                                 allele_max[i] = spacedim;
                                 if(i==numberofGenes-1){
                                                 allele_min[i] = 0;
                                         allele_max[i] = 0;
                                 }
                                 if(i==numberofGenes-2){
                                                 allele_min[i] = 0;
                                         allele_max[i] = 3;
                                 }
                         break;
                         default:

                           if((i<numberofGenes-numberofextragenes)&&(
                                only_with_skip_layer)){
                                         allele_min[i] = 0;
                                         allele_max[i] = spacedim;
                           }else{
                                //allele_min[i] = -spacedim;
                                    //allele_max[i] = spacedim;

                                   allele_min[i] = -9;
                                   allele_max[i] = 9;

                           }
                         break;
                        }


                }

                for(int i=0;i<numberofcov;i++){
                                velup[i] = Math.abs(allele_max[i]-allele_min[i]);
                                veldown[i] = -Math.abs(allele_max[i]-allele_min[i]);
```

```
        }
    }
    public Setup deepCopy(){
    Cloner cloner=new Cloner();
        Setup n = (Setup) cloner.deepClone(this);
        return n;
}
    public static void main(String[] args) throws Exception{
            int config = 6;
            Setup setup = new Setup(config);  //if config==0 default config
            for(int seed1_id=0;seed1_id<setup.seed1.length;seed1_id++){ //seed da
                amostra
              setup.seed1actual= setup.seed1[seed1_id];
              setup.prepareSamples(setup.seed1actual,0);
              for(int seed2_id=0;seed2_id<setup.seed2.length;seed2_id++){ //seed dos
                  pesos iniciais
                    setup.seed2actual = setup.seed2[seed2_id];
                    CGFunctions.Print("\n"+seed1_id+" SEED2="+setup.seed2actual+"\n
                        ");
             CGFunctions.Print("**********************\n");
             for(int seed3_id=0;seed3_id<setup.seed3.length;seed3_id++){ //seed das
                  posições iniciais
                   setup.seed3actual = setup.seed3[seed3_id];
              Rand_Number r = new Rand_Number(setup.seed3actual);
              //setup.defineSearchParams(setup.numberofcov+setup.numberofextragenes
                  ); //size of the cromossom +1
              setup.defineSearchParams();
              NodeTreeList search_list = new NodeTreeList();
              //search_list.loadfromCSV(setup.inputpath+setup.inputfolder, "
                  naoflexivel.csv",setup.numberofcov, false,";");
              switch(setup.model_searchtype){
                   case "SingleSearch":
                             SingleSearch ss1 = new SingleSearch(setup,setup.
                                 seed2actual,setup.singlecromossom);
                             ss1.listGANN.save_output(setup.outputpath+setup.
                                 outputfolder+"values/",setup.seed1actual);
                            //NodeTree node_ref = ss.listGANN.nodes.get(0);
                            //double[][] ss1_result = CGFunctions.OddsRatio(
                                setup.n_points,setup,setup.col,setup.
                                singlecromossom);
                          //CGFunctions.WriteCSV(ss1_result,setup.outputpath+setup
                              .outputfolder+"/values/ORoutCI"+setup.seed3actual+".
                              csv");
                          //double[][] result = CGFunctions.BernoulliBootStrap(
                              setup,setup.bootmax,node_ref,node_smoothed,setup.
                              n_points,setup.col);
                          //CGFunctions.WriteCSV(result,setup.outputpath+setup.
                              outputfolder+"/values/ORoutCI.csv");
                   break;
                   case "ListSearch":
                      SimplifiedNodeTree lsnode = setup.listofcromossoms.
                          getNodebySeed(setup.seed1actual);
                      setup.alfa.remove(0);
                      setup.alfa.add(lsnode.alfa);
                      SingleSearch ls = new SingleSearch(setup,setup.seed2actual,
                          lsnode.cromossom);
                      ls.listGANN.save_output(setup.outputpath+setup.outputfolder+"
                          values/",setup.seed1actual);
                      ls.listGANN.save_test(setup.outputpath+setup.outputfolder+"
                          values/",setup.seed1actual+".txt");
                      setup.singlecromossom = lsnode.cromossom;
                      double[][] ls_result = CGFunctions.OddsRatio3(setup.n_points,
                          setup,setup.col,setup.singlecromossom);
                    CGFunctions.WriteCSV(ls_result,setup.outputpath+setup.outputfolder
                        +"/values/ORoutCI"+setup.seed1actual+".csv");
                break;
                   case "AutoANN":
                      SimplifiedNodeTree auxnode = setup.listofcromossoms.
                          getNodebySeed(setup.seed1actual);
                      setup.alfa.remove(0);
                      setup.alfa.add(auxnode.alfa);
                      AutoANN autogann = new AutoANN(setup,setup.seed2actual,setup.
                          cromossom_basic,search_list,0,0,0,0);
                      autogann.local_list.save_output(setup.outputpath+setup.
                          outputfolder+"values/",setup.seed1actual);
                      /*setup.singlecromossom = autogann.local_list.get(0).cromossom
                          ;
                      double[][] ag_result = CGFunctions.OddsRatio(setup.n_points,
                          setup,setup.col,setup.singlecromossom);
                    CGFunctions.WriteCSV(ag_result,setup.outputpath+setup.outputfolder
                        +"/values/ORoutCI"+setup.seed1actual+".csv");*/
                      break;
                   case "GA":
```

```
                        ArrayofIntegersList cromossoms = Generate_Population.random(r,
                             setup.population, setup, setup.allele_max, setup.
                            allele_min);
                        new geneticAlgorithm(cromossoms,setup,setup.seed1actual,setup.
                            seed2actual,setup.seed3actual,0,0,0,0,search_list,r);
                    break;
                    case "DPSO":
                        ArrayofIntegersList initial_positions = Generate_Population.
                            random(r, setup.population, setup, setup.allele_max, setup
                            .allele_min);
                        ArrayofIntegersList initial_velocities = Generate_Population.
                            random(r, setup.population, setup, setup.velup, setup.
                            veldown);
                        //search_list.addnodes(autoganndb.readdb("select * from
                            Evaluation Where setup.seed1="+setup.seed1[seed1_id]));
                        new PSO(initial_positions, initial_velocities,setup,setup.
                            seed1actual,setup.seed2actual,setup.seed3actual,0,0,0,0,
                            search_list,r);
                    break;
                    }
                   }
                  }
                 }
                System.exit(0);
        }

}
```

Listagem A.26: Classe Setup onde a plataforma é parametrizada.

A.9 Classes e métodos auxiliares

A.9.1 Descrição

Foram implementadas classes e métodos que auxiliam as classes descritas anteriormente.

A.9.2 Código Fonte

```
package cgUtils;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.swing.JOptionPane;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;

import java.util.Set;

import org.apache.commons.lang.ArrayUtils;
import org.apache.commons.math3.util.Pair;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.poifs.filesystem.POIFSFileSystem;
import org.apache.poi.ss.usermodel.Cell;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.ui.RefineryUtilities;
import org.rosuda.REngine.REngineException;

import au.com.bytecode.opencsv.CSVWriter;
import cgModel.AssessModel;
import cgUtils.Rand_Number;
import cgSetup.Exit;
import cgSetup.Setup;
import cgTREES.NodeTree;
import cgTREES.NodeTreeList;

import com.pmstation.spss.*;



public class CGFunctions {
        public static void SPSSOut(String Path, double[][] data){
                try {
                    int n_var = data[0].length;
                    OutputStream out = new FileOutputStream(Path);

                    // Assign SPSS output to the file
```

```
                    SPSSWriter outSPSS = new SPSSWriter(out, "windows-1252");

                    // Creating SPSS variable description table
                    outSPSS.setCalculateNumberOfCases(false);
                    outSPSS.addDictionarySection(-1);

                    // Describing varible names and types
                    for(int k=0;k<n_var;k++){
                        outSPSS.addNumericVar("var"+k, 8, 4, "var"+k);
                    }

                    // Create SPSS varible value define table
                    outSPSS.addDataSection();

                    // Add values for all defined variables
                    for(int row=0;row<data.length;row++){
                        for(int col=0;col<data[0].length;col++){
                                outSPSS.addData(new Double(data[row][col]));
                        }
                    }



                    // Create SPSS ending section
                    outSPSS.addFinishSection();

                    // Close output stream
                    out.close();
                  }
                  catch (FileNotFoundException exOb) {
                    System.out.println("FileNotFoundException (GANN.main): " +
                        exOb.getMessage());
                    exOb.printStackTrace(System.out);
                    return;
                  }
                  catch (IOException exOb) {
                    System.out.println("IOException (GANN.main): " + exOb.getMessage()
                        );
                    exOb.printStackTrace(System.out);
                    return;
                  }
    }
    public static void WriteCSV(String filepath,String filename,String[] title,
        String[][] filecsvout){
            CSVWriter writer;
            boolean alreadyExists = new File(filepath+filename).exists();
            try {
                    writer = new CSVWriter(new FileWriter(filepath+filename,true),
                        ';');
                    if (!alreadyExists) writer.writeNext(title);
                    // feed in your array (or convert your data to an array)
                    String[][] arrstr = filecsvout;
                    for(int row=0; row<filecsvout.length;row++){
                                String[] entries=arrstr[row];
                                writer.writeNext(entries);
                    }
                 writer.close();
            } catch (IOException e) {
                    e.printStackTrace();
            }

    }
    public static void WriteCSV(double[][] filecsvout,String pathfilename){
            final char NO_QUOTE_CHARACTER = 0;
            CSVWriter writer;
            try {
                    writer = new CSVWriter(new FileWriter(pathfilename), ';',
                        NO_QUOTE_CHARACTER);
                    // feed in your array (or convert your data to an array)
                    String[][] arrstr = DoubletoString(filecsvout);
                    for(int row=0; row<filecsvout.length;row++){
                                String[] entries=arrstr[row];
                                writer.writeNext(entries);
                    }
                 writer.close();
            } catch (IOException e) {
                    e.printStackTrace();
            }

    }
    public static String[][] readCSV(String filepath,String filename,String
        separator) {
```

```
                String[][] result = null;
                    String csvFile = filepath+filename;
                    BufferedReader br = null;
                    String line = "";
                    String cvsSplitBy = separator;

                    try {
            int cont=1;
                            br = new BufferedReader(new FileReader(csvFile));
                            while ((line = br.readLine()) != null) {

                                    // use comma as separator
                                    String[] row = line.split(cvsSplitBy);
                result = CGFunctions.JoinArray(result, row);
                row = new String[result[0].length];
                CGFunctions.Print(cont+"\n");
                cont++;
                            }

                    } catch (FileNotFoundException e) {
                            e.printStackTrace();
                    } catch (IOException e) {
                            e.printStackTrace();
                    } finally {
                            if (br != null) {
                                    try {
                                            br.close();
                                    } catch (IOException e) {
                                            e.printStackTrace();
                                    }
                            }
                    }
                    return result;

                    //System.out.println("Done");
    }
        public static void Print(String string){
                System.out.print(string);
        }
        public static void Print(int integer){
                System.out.print(integer);
        }
        public static void Print(double value){
                System.out.print(value);
        }
        public static void MessageBox(String message){
        JOptionPane.showMessageDialog(null, message,"Mensagem",JOptionPane.
             INFORMATION_MESSAGE);
    }
        public static void MessageBoxError(String message){
        JOptionPane.showMessageDialog(null, message,"Erro",JOptionPane.ERROR_MESSAGE);
    }
        public static void printArray(String[][] array){
            for (int i = 0; i < array.length; i++){
              for (int j = 0; j < array[i].length; j++){
                System.out.print(array[i][j]);
                System.out.print("\t");
              }
              System.out.println();
            }
        }
        public static void printArray(String[] array){
              for (int j = 0; j < array.length; j++){
                System.out.print(array[j]);
                System.out.println();
              }
        }
        public static void printArray(double[][] array){
            for (int i = 0; i < array.length; i++){
              for (int j = 0; j < array[i].length; j++){
                System.out.print(array[i][j]);
                System.out.print("\t");
              }
              System.out.println();
            }
        }
        public static void printArray(double[] array){
                for (int j = 0; j < array.length; j++){
                System.out.print(array[j]);
                System.out.println();
            }
        }
        public static void printArray(int[][] array){
```

```
            for (int i = 0; i < array.length; i++){
              for (int j = 0; j < array[i].length; j++){
                System.out.print(array[i][j]);
                System.out.print("\t");
              }
              System.out.println();
            }
        }
        public static void printArray(int[] a) {
                        for (int k = 0; k < a.length; k++)
                            System.out.print("   " + a[k]);
                        System.out.println();
    }
        public static double[][] JoinArray(double[][] m1, double[][] m2){
        double[][] m;
                if((m1!=null)&&m2!=null){
                        m = new double[m1.length+m2.length][];
                    System.arraycopy(m1, 0, m, 0, m1.length);
                    System.arraycopy(m2, 0, m, m1.length, m2.length);
                    return m;
                }else{

                    if(m1!=null){
                        m = new double[m1.length][];
                        m = m1;
                    }else{
                        m = new double[m2.length][];
                        m = m2;
                    }
                    return m;
                }
        }
        public static String[][] JoinArray(String[][] m1, String[][] m2){
        String[][] m;
                if((m1!=null)&&m2!=null){
                        m = new String[m1.length+m2.length][];
                    System.arraycopy(m1, 0, m, 0, m1.length);
                    System.arraycopy(m2, 0, m, m1.length, m2.length);
                    return m;
                }else{

                    if(m1!=null){
                        m = new String[m1.length][];
                        m = m1;
                    }else{
                        m = new String[m2.length][];
                        m = m2;
                    }
                    return m;
                }
        }
        public static String[][] JoinArray(String[][] m, String[] m1){
                int n_col = m1.length;
                int n_rows = 1;
                if(m!=null) n_rows = m.length+1;
                String[][] result = new String[n_rows][n_col];
            for(int col=0;col<result[0].length;col++){
                    for(int row=0;row<result.length;row++){
                        if(n_rows==1){
                                result[row][col]=m1[col];
                        }else{
                                if(row<m.length){
                                                result[row][col]=m[row][col];
                                        }else{
                                                result[row][col]=m1[col];
                                        }
                        }

                        }
            }
                return result;
        }
        public static String[][] JoinArraybyCol(String[][] m, String[] m1){
                int n_col = 1;
                int n_rows = m1.length;
                if(m!=null) n_col = m[0].length+1;
                String[][] result = new String[n_rows][n_col];
            for(int col=0;col<result[0].length;col++){
                    for(int row=0;row<result.length;row++){
                        if(n_col==1){
                                result[row][col]=m1[row];
                        }else{
                                if(col<m[0].length){
```

```
                                                result[row][col]=m[row][col];
                                        }else{
                                                result[row][col]=m1[row];
                                        }
                        }

                        }
        }
                return result;
    }
    public static double[][] JoinArraybyCol(double[][] m, double[] m1){
                int n_col = 1;
                int n_rows = m1.length;
                if(m!=null) n_col = m[0].length+1;
                double[][] result = new double[n_rows][n_col];
            for(int col=0;col<result[0].length;col++){
                        for(int row=0;row<result.length;row++){
                            if(n_col==1){
                                    result[row][col]=m1[row];
                            }else{
                                    if(col<m[0].length){
                                                result[row][col]=m[row][col];
                                        }else{
                                                result[row][col]=m1[row];
                                        }
                            }

                            }
        }
                return result;
    }
    public static double[][] JoinArraybyCol(double[][] m, double[][] m1){
                int n_col = m[0].length+m1[0].length;
                int n_rows = m1.length;
                double[][] result = new double[n_rows][n_col];
            for(int col=0;col<result[0].length;col++){
                        for(int row=0;row<result.length;row++){
                            if(col<m[0].length){
                                    result[row][col]=m[row][col];
                            }else{
                                    result[row][col]=m1[row][col-m[0].length];
                            }
                            }
        }
                return result;
    }
    public static double[][] JoinArraybyCol(double[] m1, double[] m2, double[] m3,
         double[] m4, double[] m5){
                int size = 1;
                if(m2!=null){
                  size+=1;
    }
                if(m3!=null){
                  size+=1;
    }
                if(m4!=null){
                  size+=1;
}
                if(m5!=null){
                  size+=1;
}
                double[][] m = new double[m1.length][size];
            for(int row=0; row<m1.length; row++) {
                                m[row][0]=m1[row];
                                    if(m2!=null){
                                      m[row][1]=m2[row];
                        }
                                    if(m3!=null){
                                      m[row][2]=m3[row];
                        }
                                    if(m4!=null){
                                      m[row][3]=m4[row];
                        }
                                    if(m5!=null){
                                      m[row][4]=m5[row];
                        }
                }
                return m;
    }
    public static int[][] JoinArraybyCol(int[] m1, int[] m2, int[] m3){
                int size = 1;
                if(m2!=null){
                  size+=1;
```

```
        }
                if(m3!=null){
                  size+=1;
        }
                int[][] m = new int[m1.length][size];
            for(int row=0; row<m1.length; row++) {
                            m[row][0]=m1[row];
                                if(m2!=null){
                                  m[row][1]=m2[row];
                    }
                                if(m3!=null){
                                  m[row][2]=m3[row];
                    }
                }
                return m;
        }
        public static double[] JoinArray(double[] m1, double[] m2){
                double[] m;
                if((m1!=null)&&(m2!=null)){
                            m = new double[m1.length+m2.length];
                        System.arraycopy(m1, 0, m, 0, m1.length);
                        System.arraycopy(m2, 0, m, m1.length, m2.length);
                        return m;
                }else{

                        if(m1!=null){
                            m = new double[m1.length];
                            m = m1;
                        }else{
                            m = new double[m2.length];
                            m = m2;
                        }
                        return m;
                }
        }
        public static int[] JoinArray(int number, int[] m2){
                int[] m;
                int[] m1 = new int[1];
                m1[0]=number;
                if(m2!=null){
                            m = new int[1+m2.length];
                        System.arraycopy(m1, 0, m, m2.length, 1);
                        System.arraycopy(m2, 0, m, 0, m2.length);
                        return m;
                }else{
                        return m1;
                }
        }
        public static double[] GetColumn(double[][] array, int col){
                double[] column = new double[array.length];
            for (int i = 0; i < array.length; i++){
              column[i] = array[i][col];
            }
            return column;
        }
        public static String[] GetColumn(String[][] array, int col){
                String[] column = new String[array.length];
            for (int i = 0; i < array.length; i++){
              column[i] = array[i][col];
            }
            return column;
        }
        public static double[] GetRow(double[][] array, int r){
                double[] row = new double[array[0].length];
            for (int i = 0; i < array[0].length; i++){
              row[i] = array[r][i];
            }
            return row;
        }
        public static double[][] SetColumn(double[][] array, double[] vector, int col){
            for (int row = 0; row < array.length; row++){
              array[row][col]=vector[row];
            }
            return array;
        }
        public static double GetMaxArray(double[][] array){
                double max=array[0][0];
                for (int col = 0; col < array[0].length; col++){
                        for (int row = 0; row < array.length; row++){
                                if(array[row][col]>max){
                                        max=array[row][col];
                                }
                        }
```

```
                }
                return max;
        }
        public static double GetMaxArray(double[] array){
                double max=array[0];
                        for (int row = 0; row < array.length; row++){
                                if(array[row]>max){
                                        max=array[row];
                                }
                        }
                return max;
        }
        public static int[] Permute(int[] a,int seed){
                int[] b = (int[])a.clone();
                new java.util.Random(seed);
                for (int k = b.length - 1; k > 0; k--) {
                    int w = (int)Math.floor(Math.random() * (k+1));
                    int temp = b[w];
                    b[w] = b[k];
                    b[k] = temp;
                }
                return(b);
        }
        public static int[] toIntArray(final Collection<Integer> data){
            int[] result;
            // null result for null input
            if(data == null){
                result = null;
            // empty array for empty collection
            } else if(data.isEmpty()){
                result = new int[0];
            } else{
                final Collection<Integer> effective;
                // if data contains null make defensive copy
                // and remove null values
                if(data.contains(null)){
                    effective = new ArrayList<Integer>(data);
                    while(effective.remove(null)){}
                // otherwise use original collection
                }else{
                    effective = data;
                }
                result = new int[effective.size()];
                int offset = 0;
                // store values
                for(final Integer i : effective){
                    result[offset++] = i.intValue();
                }
            }
            return result;
        }
        public static int[] Set_except_Number(int[] set,int number){
             int[] output = null;
             for(int z=0;z<set.length;z++){
                    if(set[z]!=number){
                        output=CGFunctions.JoinArray(set[z],output);
                    }
                }

             return output;
    }
        public static List<Double> Double2List(double[] array){
        List<Double> list = new ArrayList<Double>();
        for(int i=0;i<array.length;i++){
                list.add(array[i]);
        }
        return list;
    }
        public static ArrayList<int[]> Integer2List(int[][] array){

        ArrayList<int[]> list = new ArrayList<int[]>();
        for(int i=0;i<array.length;i++){
                list.add(array[i]);
        }
        return list;
    }
    public static ArrayList<Integer> Integer2List(int[] array){

        ArrayList<Integer> intList = new ArrayList<Integer>();
        for (int index = 0; index < array.length; index++){
            intList.add(array[index]);
        }
```

```
        return intList;
    }
    public static int[] ArrayList2int(ArrayList<Integer> list){
        int[] result = new int[list.size()];
        Iterator<Integer> iterator = list.iterator();
        for (int i = 0; i < result.length; i++){
                result[i] = iterator.next().intValue();
        }
        return result;
    }
    public static double[][] ArrayDouble2double(ArrayList<double[]> list){
        double[][] result = new double[list.size()][];
        Iterator<double[]> iterator = list.iterator();
        for (int i = 0; i < result.length; i++){
                result[i] = iterator.next();
        }
        return result;
    }
    public static String[][] ArrayString2String(ArrayList<String[]> list){
        String[][] result = new String[list.size()][];
        Iterator<String[]> iterator = list.iterator();
        for (int i = 0; i < result.length; i++){
                result[i] = iterator.next();
        }
        return result;
    }
    public static int[][] List2int(ArrayList<ArrayList<Integer>> arrayoflists){
        ArrayList<Integer> list = arrayoflists.get(0);
        int[][] result = new int[arrayoflists.size()][list.size()];
        result[0] = ArrayList2int(list);
        for (int i = 1; i < result.length; i++){
            result[i] = ArrayList2int(arrayoflists.get(i));
        }
        return result;
    }
        public static int[] DoubletoInt(double[] arrdbl){
                int[] output = new int[arrdbl.length];
                for(int z=0;z<output.length;z++){
                        output[z]=(int) arrdbl[z];
                }
                return output;
        }
    public static String[][] DoubletoString(double[][] arrdbl){
      String[][] arrstr = new String[arrdbl.length][arrdbl[0].length];
          for(int row = 0; row < arrdbl.length; row++) {
                  for(int col = 0; col < arrdbl[0].length; col++){
                          arrstr[row][col]= Double.toString(arrdbl[row][col]);
                  }
          }
      return arrstr;
    }
    public static String DoubletoString(double[] arrdbl){
        String arrstr="";
          for(int row = 0; row < arrdbl.length; row++) {
                          arrstr= arrstr + Double.toString(arrdbl[row])+" ";
          }
        return arrstr;
      }
    public static String InttoString(int[] arrdbl){
        String arrstr="";
          for(int row = 0; row < arrdbl.length; row++) {
                          arrstr= arrstr + Integer.toString(arrdbl[row])+" ";
          }
        return arrstr;
    }
    public static String DoubletoString(int[] arrint){
        String arrstr="";
          for(int row = 0; row < arrint.length; row++) {
                          arrstr= arrstr + Double.toString(arrint[row])+" ";
          }
        return arrstr;
      }
    public static void Plot(String title, String titlex, String titley, double[] x,
         double[] y){
        XYSeriesCollection dataset = new XYSeriesCollection();
        XYSeries ROCGraphic = new XYSeries(title);
        for(int cutoff=0; cutoff<x.length;cutoff++){
                ROCGraphic.add(x[cutoff], y[cutoff]);
        }
        dataset.addSeries(ROCGraphic);
        JFreeChart chart = ChartFactory.createScatterPlot(
            title, // title
            titlex, titley,            // axis labels
```

```
            dataset,                    // dataset
            PlotOrientation.VERTICAL,
            true,                       // legend? yes
            true,                       // tooltips? yes
            false                       // URLs? no
        );
        ChartPanel chartPanel = new ChartPanel(chart);
        chartPanel.setPreferredSize(new java.awt.Dimension(800, 600));
        //setContentPane(chartPanel);
        XYPlot plot = chart.getXYPlot();
        final XYLineAndShapeRenderer renderer = new XYLineAndShapeRenderer();
        for(int i=0; i<3;i++){
                renderer.setSeriesLinesVisible(i, true);
                renderer.setSeriesShapesVisible(i, false);
        }
        plot.setRenderer(renderer);
    }
    public static double[][] StringtoDouble(String[][] arrstr){
        double[][] arrdbl = new double[arrstr.length][arrstr[0].length];
          for(int row = 0; row < arrstr.length; row++) {
                  for(int col = 0; col < arrstr[0].length; col++){
                          arrdbl[row][col]= Double.parseDouble(arrstr[row][col]);
                  }
          }
        return arrdbl;
    }
    public static int[] StringtoInteger(String[] arrstr){
        int[] arrint = new int[arrstr.length];
          for(int row = 0; row < arrstr.length; row++) {

                          arrint[row]= Integer.parseInt(arrstr[row]);

          }
        return arrint;
    }
    public static double[][] Sort(double[][] arraydbl){
        double[][] x = new double[arraydbl.length][arraydbl[0].length];
        ArrayList<double[]> a = new ArrayList<double[]>();
        for(int i=0;i<arraydbl.length;i++){
                a.add(arraydbl[i]);
        }
        Collections.sort (a, new Comparator<Object>() {
                public int compare(Object o1, Object o2) {
                        int result = 0;
                        double[] p1 = (double[]) o1;
                        double[] p2 = (double[]) o2;
                        if (p1[0]==p2[0]){
                            if (p1[1]<p2[1]){
                                    result=-1;
                            }
                            if (p1[1]>p2[1]){
                                    result=1;
                            }
                            if (p1[1]==p2[1]){
                                    result=0;
                            }
                        }
                        if (p1[0]<p2[0]){
                                    result=-1;
                            }
                            if (p1[0]>p2[0]){
                                    result=1;
                            }

                        return result;
                }
        });
        for(int i=0;i<arraydbl.length;i++){
                x[i]=a.get(i);
            }
        return(x);
    }
    public static int[][] Sort(int[][] arrayint){
        int[][] x = new int[arrayint.length][arrayint[0].length];
        ArrayList<int[]> a = new ArrayList<int[]>();
        for(int i=0;i<arrayint.length;i++){
                a.add(arrayint[i]);
        }
        Collections.sort (a, new Comparator<Object>() {
                public int compare(Object o1, Object o2) {
                        int result = 0;
                        int[] p1 = (int[]) o1;
                        int[] p2 = (int[]) o2;
```

```
                        if (p1[0]==p2[0]){
                            if (p1[1]<p2[1]){
                                    result=-1;
                            }
                            if (p1[1]>p2[1]){
                                    result=1;
                            }
                            if (p1[1]==p2[1]){
                                    result=0;
                            }
                        }
                        if (p1[0]<p2[0]){
                                    result=-1;
                            }
                            if (p1[0]>p2[0]){
                                    result=1;
                            }

                        return result;
                }
        });
        for(int i=0;i<arrayint.length;i++){
                    x[i]=a.get(i);
            }
        return(x);
    }
    public static double[][] SortAndClassify(double[][] arrdbl, int key){
      double[][] arr = new double[arrdbl.length][4];
      for(int row = 0; row < arrdbl.length; row++) {
          arr[row][0]=arrdbl[row][0];
          arr[row][1]=arrdbl[row][1];
          arr[row][2]=Mathf.algsift(arrdbl[row][0],key);
          arr[row][3]=1;//For counting purposes
      }
      double[][] result = CGFunctions.Sort(arr);
      return result;
    }
    public static String[][] Readxls(String path,int begin_row) throws
         FileNotFoundException, IOException{

                InputStream inputStream = null;
                           inputStream = new FileInputStream (path);
                POIFSFileSystem fileSystem = null;
                fileSystem = new POIFSFileSystem (inputStream);
                        HSSFWorkbook wb = new HSSFWorkbook(fileSystem);
                        HSSFSheet sheet = wb.getSheetAt(0);
                        int lastrow = sheet.getLastRowNum();
                        String[][] fileout = new String[lastrow+1][];
                        for(int index_row = begin_row; index_row < lastrow+1; index_row++)
                            {
                      HSSFRow row = sheet.getRow(index_row);
                                int lastcol = row.getLastCellNum();
                                String[] column = new String[lastcol];
                                for(int index_col = 0; index_col < lastcol; index_col++) {
                                    //column[index_col]=row.getCell(index_col).
                                          getNumericCellValue();
                                    Cell cell = row.getCell(index_col);
                                    cell.setCellType(Cell.CELL_TYPE_STRING);
                                    column[index_col]=cell.getStringCellValue();

                                    //column[index_col]=row.getCell(index_col).
                                         getStringCellValue();

                                }
                                fileout[index_row]=column;
                        }
                  return fileout;
        }
    public static double[][] Readxlsdbl(String path,int begin_row) throws
         FileNotFoundException, IOException{

                InputStream inputStream = null;
                    inputStream = new FileInputStream (path);
              POIFSFileSystem fileSystem = null;
              fileSystem = new POIFSFileSystem (inputStream);
                HSSFWorkbook wb = new HSSFWorkbook(fileSystem);
                HSSFSheet sheet = wb.getSheetAt(0);
                int lastrow = sheet.getLastRowNum();
                double[][] fileout = new double[lastrow+1][];
                for(int index_row = begin_row; index_row < lastrow+1; index_row++) {
                HSSFRow row = sheet.getRow(index_row);
                       int lastcol = row.getLastCellNum();
                       double[] column = new double[lastcol];
```

```
                for(int index_col = 0; index_col < lastcol; index_col++) {
                    column[index_col]=row.getCell(index_col).getNumericCellValue();
                    //column[index_col]=row.getCell(index_col).getStringCellValue();
                }
                fileout[index_row]=column;
            }
            return fileout;
    }
     public static void Savexls(String path, String name, double[][] fileout) throws
          FileNotFoundException, IOException{
         try {
                        HSSFWorkbook wb = new HSSFWorkbook();
                        HSSFSheet sheet = wb.createSheet(name);
                        for(int index_row = 0; index_row < fileout.length; index_row++) {
                    HSSFRow row = sheet.createRow((short) index_row);
                            for(int index_col = 0; index_col < fileout[0].length;
                                 index_col++) {
                                row.createCell(index_col).setCellValue(fileout[index_row
                                     ][index_col]);
                                //System.out.println("row->"+index_row+"col->"+index_col
                                     );
                            }
                        }
                        FileOutputStream file = new FileOutputStream(path);
                        wb.write(file);
                        file.close();
                        //System.out.println("Data is saved in excel file.");
                    } catch (Exception e) {}
         }
     public static void Savexls(String path, String name, String[][] fileout) throws
          FileNotFoundException, IOException{
         try {
                        HSSFWorkbook wb = new HSSFWorkbook();
                        HSSFSheet sheet = wb.createSheet(name);
                        for(int index_row = 0; index_row < fileout.length; index_row++) {
                    HSSFRow row = sheet.createRow((short) index_row);
                            for(int index_col = 0; index_col < fileout[0].length;
                                 index_col++) {
                                row.createCell(index_col).setCellValue(fileout[index_row
                                     ][index_col]);
                                //System.out.println("row->"+index_row+"col->"+index_col
                                     );
                            }
                        }
                        FileOutputStream file = new FileOutputStream(path);
                        wb.write(file);
                        file.close();
                        //System.out.println("Data is saved in excel file.");
                    } catch (Exception e) {}
         }
     public static void Savexls(String path, String name, double[] fileout) throws
          FileNotFoundException, IOException{
         try {
                        HSSFWorkbook wb = new HSSFWorkbook();
                        HSSFSheet sheet = wb.createSheet(name);
                        int index_col = 0;
                        for(int index_row = 0; index_row < fileout.length; index_row++) {
                    HSSFRow row = sheet.createRow((short) index_row);
                            row.createCell(index_col).setCellValue(fileout[index_row]);
                            //System.out.println("row->"+index_row+"col->"+index_col);
                        }
                        FileOutputStream file = new FileOutputStream(path);
                        wb.write(file);
                        file.close();
                        //System.out.println("Data is saved in excel file.");
                    } catch (Exception e) {}
         }
     public static void Savexls(String path, String name, String[] fileout) throws
          FileNotFoundException, IOException{
         try {
                        HSSFWorkbook wb = new HSSFWorkbook();
                        HSSFSheet sheet = wb.createSheet(name);
                        int index_col = 0;
                        for(int index_row = 0; index_row < fileout.length; index_row++) {
                    HSSFRow row = sheet.createRow((short) index_row);
                            row.createCell(index_col).setCellValue(fileout[index_row]);
                            //System.out.println("row->"+index_row+"col->"+index_col);
                        }
                        FileOutputStream file = new FileOutputStream(path);
                        wb.write(file);
                        file.close();
                        //System.out.println("Data is saved in excel file.");
                    } catch (Exception e) {}
```

```
    }
public static void ROCCurve(double[][] fileout){
            int result;
            int n_cutoff=1000;
            int total_ind = fileout.length;
            double[][] classifier = new double[total_ind][n_cutoff+1];
            double[] tp=new double[n_cutoff+1];
            double[] tn=new double[n_cutoff+1];
            double[] fp=new double[n_cutoff+1];
            double[] fn=new double[n_cutoff+1];
            double[] acc=new double[n_cutoff+1];
            double[] sens=new double[n_cutoff+1];
            double[] spec=new double[n_cutoff+1];
            double[] onemspec=new double[n_cutoff+1];
            double[] trapezoid=new double[n_cutoff+1];
            double auc=0;
            double[] eff=new double[n_cutoff+1];
            double[] ppv=new double[n_cutoff+1];
            double[] npv=new double[n_cutoff+1];
            double[] phi=new double[n_cutoff+1];
    for(int cutoff=1; cutoff<=n_cutoff;cutoff++){
            double threshold = (double)(cutoff)/n_cutoff;
            for(int ind=0; ind<fileout.length;ind++){
                    if(fileout[ind][0]>=threshold){
                            result=1;
                    }else{
                            result=0;
                    }
                    classifier[ind][cutoff]=fileout[ind][1]-result;
                    if((result==1)&&(fileout[ind][1]==1)){
                                    classifier[ind][cutoff]=2;//code for true
                                        positive
                    }
                    //-1=false positive,0=true negative,1=false negative,2=true
                        positive
                    if(classifier[ind][cutoff]==-1){
                            fp[cutoff]+=1;
                    }
            if(classifier[ind][cutoff]==0){
                            tn[cutoff]+=1;
                    }
            if(classifier[ind][cutoff]==1){
                            fn[cutoff]+=1;
                    }
            if(classifier[ind][cutoff]==2){
                            tp[cutoff]+=1;
                    }
            }
    }
            sens[0]=1;
            spec[0]=0;
            onemspec[0]=1;
            for(int cutoff=1; cutoff<=n_cutoff;cutoff++){
                acc[cutoff] = (tp[cutoff]+tn[cutoff])/total_ind;
                sens[cutoff] = tp[cutoff]/(tp[cutoff] + fn[cutoff]);
                spec[cutoff] = tn[cutoff]/(tn[cutoff] + fp[cutoff]);
                onemspec[cutoff]=1-spec[cutoff];
                eff[cutoff] = (sens[cutoff]+spec[cutoff])/2;
                trapezoid[cutoff]=(sens[cutoff]*(onemspec[cutoff-1]-onemspec[cutoff
                    ]))+0.5*((onemspec[cutoff-1]-onemspec[cutoff])*(sens[cutoff-1]-
                    sens[cutoff]));
                auc+=trapezoid[cutoff];
                ppv[cutoff] = tp[cutoff]/(tp[cutoff]+fp[cutoff]);
                npv[cutoff] = tn[cutoff]/(tn[cutoff]+fn[cutoff]);
                phi[cutoff] =(tp[cutoff]*tn[cutoff]-tp[cutoff]*tn[cutoff])/Math.sqrt
                    ((tp[cutoff]+fp[cutoff])*(tp[cutoff]+fn[cutoff])*(tn[cutoff]+tp[
                    cutoff])*(tn[cutoff]+fn[cutoff]));
            }
            System.out.println("AUC="+auc);
    Plotting figure = new Plotting("ROC Curve","Specificity","Sensitivity",onemspec,
        sens);
    figure.pack();
    RefineryUtilities.centerFrameOnScreen(figure);
    figure.setVisible(true);
    }
public static Double[][] Sort(Double[][] array){
    Arrays.sort(array, new Comparator<Double[]>(){
                @Override
                public int compare(Double[] int1, Double[] int2)
                {
                    Double numOfKeys1 = int1[1];
                    Double numOfKeys2 = int2[1];
                    return numOfKeys1.compareTo(numOfKeys2);
```

```
                    }
        });
        return(array);
    }
    public static Double[][] orderWithoutDuplicates(double[][] array){

           List<Double[]> aux = new ArrayList<Double[]>();

           for (double[] i : array){
                   Double[] z = ArrayUtils.toObject(i);
             if (!aux.contains(z)) aux.add(z);
           }
           if (aux.size() % 2 == 0){
                   double[] result1 = new double[aux.size()];
                   double[] result2 = new double[aux.size()];
                   for(int i=0;i<aux.size();i++){
                           result1[i]=aux.get(i)[0];
                           result2[i]=aux.get(i)[1];
                   }
                   double med = Mathf.mediana(result1);
                   double medorig = Mathf.mediana(result2);
                   Double[] value = new Double[2];
                   value[0]=med;
                   value[1]=medorig;
                   aux.add(value);
           }
           Double[][] result = aux.toArray(new Double[aux.size()][2]);
           return (Sort(result));
    }
    public static Double[] GetColumn(Double[][] array, int col){
                Double[] column = new Double[array.length];
             for (int i = 0; i < array.length; i++){
               column[i] = array[i][col];
             }
             return column;
         }
    public static double[][] OddsRatio2(int n_points,Setup setup2,int col,int[]
         cromossom) throws IOException, Exception{
         int n_ind = setup2.total_sample.n_ind;
         int n_cov = setup2.total_sample.n_cov;
         if(col>=setup2.total_sample.n_cov) Exit.withError(7);

         //Definição da variável xk
         double[] xaux = CGFunctions.GetColumn(setup2.total_sample.alldata_aux,col);
         double[] xorig_aux = CGFunctions.GetColumn(setup2.total_sample.alldata_original,
              col);

         double xmin = Mathf.min(xaux);
         double xmax = Mathf.max(xaux);
         double xorig_min = Mathf.min(xorig_aux);
         double xorig_max = Mathf.max(xorig_aux);

         double[] xs = new double[n_points+1];
         double[] xs_orig = new double[n_points+1];

         double inc = (xmax-xmin)/n_points;
         double inc_orig = (xorig_max-xorig_min)/n_points;

         //definição das variáveis que guardarão valores intermédios como pi e odds
         double[][] pi_total = null;
         String[][] pi_total_title = new String[1][n_points+1];
         String[][] odd_total_title = new String[1][n_points+1];
         double[][] odd_total = null;

         double x=xmin;
         double x_orig=xorig_min;
         double x_ref=Mathf.mediana(xaux);
         double xorig_ref=Mathf.mediana(xorig_aux);
         //Para cada valor de xk calcular o odds ratio
         for(int count=0;count<n_points+1;count++){
                 if(count==(n_points+1)/2){
                         x = x_ref;
                         x_orig = xorig_ref;
                 }
                 //Clonar uma amostra de dados com a coluna col = x
                 double[][] total_sample_x = new double[n_ind][n_cov+1];
                 for(int row = 0;row<total_sample_x.length;row++){
                         for(int co = 0;co<total_sample_x[0].length;co++){
                                 if(co!=col){
                                         total_sample_x[row][co] = setup2.total_sample.
                                              alldata[row][co];
                                 }else{
                                         total_sample_x[row][co] = x;
```

```
                                }


                        }
                }


                //A partir da amostra clonada dividi-la em subamostras para validação
                    cruzada
                setup2.total_sample.Slicing(setup2.maxslices);
                Sample tot_sample_x = new Sample(total_sample_x);
                tot_sample_x.Slicing(setup2.maxslices);
                Sample[] sample_x = tot_sample_x.GetSamplesCV(setup2.evaluation, setup2.
                    kfolds,setup2.maxslices);

                //Aplicar a amostra clonada como amostra de teste
                for(int k=0;k<setup2.maxslices;k++) setup2.sample[k].testdata=sample_x[k
                    ].testdata;

                //Adaptar o modelo com a amostra original e estimar p(X1,...,x,...,Xp)
                    em que x inclui o xref a partir da amostra de teste
                NodeTree node = AssessModel.Evaluation(setup2,setup2.seed2actual,
                    cromossom,0,0,0);
                //Obter os valores de pi para cada x incluindo o xref
                double[] p = CGFunctions.GetColumn(node.outtest.getKey(), 0);
                for(int i=0;i<p.length;i++){
                        if(p[i]>0.999999) p[i]=0.999999;
                        if(p[i]<0.000001) p[i]=0.000001;
                }
                //Guardar os valores de pi e os valores de odd=p/(1-p) numa matriz
                double[] odd = new double[n_ind];
                for(int i=0;i<n_ind;i++) odd[i] = p[i]/(1-p[i]);
                pi_total = CGFunctions.JoinArraybyCol(pi_total,p);
                odd_total = CGFunctions.JoinArraybyCol(odd_total,odd);

                //Atualizar contadores
                xs[count]=Mathf.algsifr(x,2);

                xs_orig[count]=Mathf.algsifr(x_orig,2);
                pi_total_title[0][count] = "p(x="+Double.toString(Mathf.algsifr(x,1))+",
                    X2)";
                odd_total_title[0][count] = "odd(x=("+Double.toString(Mathf.algsifr(x,1)
                    )+"),X2)";
                x+=inc;
                x_orig+=inc_orig;
        }

        //Gravar matriz com os valores de pi para cada x incluindo o xref
        String[][] pixlsout = CGFunctions.JoinArray(pi_total_title,CGFunctions.
            DoubletoString(pi_total));
        String[][] oddxlsout = CGFunctions.JoinArray(pi_total_title,CGFunctions.
            DoubletoString(odd_total));
        CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/PI.xls", "PI",
             pixlsout);
        CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/Odds.xls", "
            Odd", oddxlsout);

        double[] odd_ref = CGFunctions.GetColumn(odd_total,(n_points+1)/2);
                CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/Oddref
                    .xls", "Oddref", odd_ref);

                //Obter a razão de odds ou de possibilidades para cada valor x de xk
                    pertencente ao intervalo

                double[][] or = new double[n_ind][n_points+1];
                double[] or_total = new double[n_points+1];

                for(int point=0;point<n_points+1;point++){
                        double acum=0;
                        for(int i=0;i<n_ind;i++){
                                or[i][point]=odd_total[i][point]/odd_ref[i];
                                acum+=or[i][point];
                        }
                        or_total[point]=acum/n_ind;
                }

                double[][] or_out = CGFunctions.JoinArraybyCol(xs, xs_orig, or_total,
                    null, null);
                return or_out;

    }
    public static double[][] OddsRatio(int n_points,Setup setup2,int col,int[] cromossom
        ) throws IOException, Exception{
```

```
int n_ind = setup2.total_sample.n_ind;
int n_cov = setup2.total_sample.n_cov;
if(col>=setup2.total_sample.n_cov) Exit.withError(7);

//Definição da variável xk
double[] xaux = CGFunctions.GetColumn(setup2.total_sample.alldata,col);
double[] xorig_aux = CGFunctions.GetColumn(setup2.total_sample.alldata_original,
     col);

double xmin = Mathf.min(xaux);
double xmax = Mathf.max(xaux);
double xorig_min = Mathf.min(xorig_aux);
double xorig_max = Mathf.max(xorig_aux);

double[] xs = new double[n_points+1];
double[] xs_orig = new double[n_points+1];

double inc = (xmax-xmin)/(n_points-1);
double inc_orig = (xorig_max-xorig_min)/(n_points-1);

//definição das variáveis que guardarão valores intermédios como pi e odds
double[][] pi_total = null;
String[][] pi_total_title = new String[1][n_points+1];
String[][] odd_total_title = new String[1][n_points+1];
double[][] odd_total = null;

double x=xmin;
double x_orig=xorig_min;
double x_ref=Mathf.mediana(xaux);
//double x_ref=0;
double xorig_ref=Mathf.mediana(xorig_aux);
//double xorig_ref=0;
//Para cada valor de xk calcular o odds ratio
for(int count=0;count<n_points+1;count++){

        if(count==n_points){
                x = x_ref;
                x_orig = xorig_ref;
        }else{
                x=xmin+inc*count;
                x_orig=xorig_min+inc_orig*count;
        }
        //Clonar uma amostra de dados com a coluna col = x
        double[][] total_sample_x = new double[n_ind][n_cov+1];
        for(int row = 0;row<total_sample_x.length;row++){
                for(int co = 0;co<total_sample_x[0].length;co++){
                        if(co!=col){
                                total_sample_x[row][co] = setup2.total_sample.
                                     alldata[row][co];
                        }else{
                                total_sample_x[row][co] = x;
                        }


                }
        }


        //A partir da amostra clonada dividi-la em subamostras para validação
             cruzada
        setup2.total_sample.Slicing(setup2.maxslices);
        Sample tot_sample_x = new Sample(total_sample_x);
        tot_sample_x.Slicing(setup2.maxslices);
        Sample[] sample_x = tot_sample_x.GetSamplesCV(setup2.evaluation, setup2.
             kfolds,setup2.maxslices);

        //Aplicar a amostra clonada como amostra de teste
        for(int k=0;k<setup2.maxslices;k++) setup2.sample[k].testdata=sample_x[k
             ].testdata;

        //Adaptar o modelo com a amostra original e estimar p(X1,...,x,...,Xp)
             em que x inclui o xref a partir da amostra de teste
        NodeTree node = AssessModel.Evaluation(setup2,setup2.seed2actual,
             cromossom,0,0,0);
        //Obter os valores de pi para cada x incluindo o xref
        double[] p = CGFunctions.GetColumn(node.outtest.getKey(), 0);
        for(int i=0;i<p.length;i++){
                if(p[i]>0.999999) p[i]=0.999999;
                if(p[i]<0.000001) p[i]=0.000001;
        }
        //Guardar os valores de pi e os valores de odd=p/(1-p) numa matriz
        double[] odd = new double[n_ind];
        for(int i=0;i<n_ind;i++) odd[i] = p[i]/(1-p[i]);
```

```
                pi_total = CGFunctions.JoinArraybyCol(pi_total,p);
                odd_total = CGFunctions.JoinArraybyCol(odd_total,odd);

                //Atualizar contadores
                xs[count]=Mathf.algsifr(x,2);

                xs_orig[count]=Mathf.algsifr(x_orig,2);
                pi_total_title[0][count] = "p(x="+Double.toString(Mathf.algsifr(x,1))+",
                    X2)";
                odd_total_title[0][count] = "odd(x=("+Double.toString(Mathf.algsifr(x,1)
                    )+"),X2)";

        }

        //Gravar matriz com os valores de pi para cada x incluindo o xref
        String[][] pixlsout = CGFunctions.JoinArray(pi_total_title,CGFunctions.
            DoubletoString(pi_total));
        String[][] oddxlsout = CGFunctions.JoinArray(pi_total_title,CGFunctions.
            DoubletoString(odd_total));
        CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/PI.xls", "PI",
             pixlsout);
        CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/Odds.xls", "
            Odd", oddxlsout);

        //Obter o odd de referencia que corresponde ao xref = (xmax-xmin)/2
        double[] odd_ref = CGFunctions.GetColumn(odd_total,n_points);
                CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/Oddref
                    .xls", "Oddref", odd_ref);

                //Obter a razão de odds ou de possibilidades para cada valor x de xk
                    pertencente ao intervalo

                double[][] or = new double[n_ind][n_points+1];
                double[] or_total = new double[n_points+1];

                for(int point=0;point<n_points+1;point++){
                        double acum=0;
                        for(int i=0;i<n_ind;i++){
                                or[i][point]=odd_total[i][point]/odd_ref[i];
                                acum+=or[i][point];
                        }
                        or_total[point]=acum/n_ind;
                }

                double[][] or_out = CGFunctions.JoinArraybyCol(xs, xs_orig, or_total,
                    null, null);
                return or_out;

    }
    public static double[][] OddsRatio3(int n_points,Setup setup2,int col,int[]
        cromossom) throws IOException, Exception{
        int n_ind = setup2.total_sample.n_ind;
        int n_cov = setup2.total_sample.n_cov;
        if(col>=setup2.total_sample.n_cov) Exit.withError(7);

        //Definição da variável xk
        double[] xaux = CGFunctions.GetColumn(setup2.total_sample.alldata,col);
        double[] xorig_aux = CGFunctions.GetColumn(setup2.total_sample.alldata_original,
            col);
        double xref = Mathf.mediana(xaux);
        double xreforig = Mathf.mediana(xorig_aux);
        double xmin = Mathf.min(xaux);
        double xmax = Mathf.max(xaux);
        double xorig_min = Mathf.min(xorig_aux);
        double xorig_max = Mathf.max(xorig_aux);
        double inc_right = Math.abs(xmax-xref)/n_points;
        double inc_left = Math.abs(xref-xmin)/n_points;
        double incorig_right = 2*Math.abs(xorig_max-xreforig)/n_points;
        double incorig_left = 2*Math.abs(xreforig-xorig_min)/n_points;
        double[] xset = new double[n_points+1];
        double[] xorig = new double[n_points+1];
        xset[n_points/2]=xref;
        xorig[n_points/2]=xreforig;
        for(int i=0;i<n_points/2;i++){
                if(i==0){
                        xset[i]=xmin;
                        xset[n_points-i]=xmax;
                        xorig[i]=xorig_min;
                        xorig[n_points-i]=xorig_max;
                }else{
                        xset[i]=xset[i-1]+inc_left;
                        xset[n_points-i]=xset[n_points-i+1]-inc_right;
                        xorig[i]=xorig[i-1]+incorig_left;
```

```
                xorig[n_points-i]=xorig[n_points-i+1]-incorig_right;
        }

}


double[] xs = new double[n_points+1];
double[] xs_orig = new double[n_points+1];

//definição das variáveis que guardarão valores intermédios como pi e odds
double[][] pi_total = null;
String[][] pi_total_title = new String[1][n_points+1];
String[][] odd_total_title = new String[1][n_points+1];
double[][] odd_total = null;

double x;
double x_orig;

//Para cada valor de xk calcular o odds ratio
for(int count=0;count<n_points+1;count++){
        x=xset[count];
        x_orig=xorig[count];
        //Clonar uma amostra de dados com a coluna col = x
        double[][] total_sample_x = new double[n_ind][n_cov+1];
        for(int row = 0;row<total_sample_x.length;row++){
                for(int co = 0;co<total_sample_x[0].length;co++){
                        if(co!=col){
                                total_sample_x[row][co] = setup2.total_sample.
                                    alldata[row][co];
                        }else{
                                total_sample_x[row][co] = x;
                        }


                }
        }


        //A partir da amostra clonada dividi-la em subamostras para validação
            cruzada
        setup2.total_sample.Slicing(setup2.maxslices);
        setup2.sample[0].valdata = setup2.total_sample.slice[0].data;
        setup2.sample[0].testdata = total_sample_x;*/
        Sample tot_sample_x = new Sample(total_sample_x);
        tot_sample_x.Slicing(setup2.maxslices);
        Sample[] sample_x = tot_sample_x.GetSamplesCV(setup2.evaluation, setup2.
            kfolds,setup2.maxslices);
        //Aplicar a amostra clonada como amostra de teste
        for(int k=0;k<setup2.maxslices;k++) setup2.sample[k].testdata=sample_x[k
            ].testdata;

        //Adaptar o modelo com a amostra original e estimar p(X1,...,x,...,Xp)
            em que x inclui o xref a partir da amostra de teste
        NodeTree node = AssessModel.Evaluation(setup2,setup2.seed2actual,
            cromossom,0,0,0);
        //Obter os valores de pi para cada x incluindo o xref
        double[] p = CGFunctions.GetColumn(node.outtest.getKey(), 0);

        //Guardar os valores de pi e os valores de odd=p/(1-p) numa matriz
        double[] odd = new double[n_ind];
        for(int i=0;i<n_ind;i++) odd[i] = p[i]/(1-p[i]);
        pi_total = CGFunctions.JoinArraybyCol(pi_total,p);
        odd_total = CGFunctions.JoinArraybyCol(odd_total,odd);

        //Atualizar contadores
        xs[count]=Mathf.algsifr(x,2);

        xs_orig[count]=Mathf.algsifr(x_orig,2);
        pi_total_title[0][count] = "p(x="+Double.toString(Mathf.algsifr(x,1))+",
            X2)";
        odd_total_title[0][count] = "odd(x=("+Double.toString(Mathf.algsifr(x,1)
            )+"),X2)";

}

//Gravar matriz com os valores de pi para cada x incluindo o xref
String[][] pixlsout = CGFunctions.JoinArray(pi_total_title,CGFunctions.
    DoubletoString(pi_total));
String[][] oddxlsout = CGFunctions.JoinArray(pi_total_title,CGFunctions.
    DoubletoString(odd_total));
CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/PI.xls", "PI",
     pixlsout);
```

```
        CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/Odds.xls", "
            Odd", oddxlsout);

        //Obter o odd de referencia que corresponde ao xref = (xmax-xmin)/2
        double[] odd_ref = CGFunctions.GetColumn(odd_total,(n_points+1)/2);
                CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"values/Oddref
                    .xls", "Oddref", odd_ref);

                //Obter a razão de odds ou de possibilidades para cada valor x de xk
                    pertencente ao intervalo

                double[][] or = new double[n_ind][n_points+1];
                double[] or_total = new double[n_points+1];

                for(int point=0;point<n_points+1;point++){
                        double acum=0;
                        for(int i=0;i<n_ind;i++){
                                or[i][point]=odd_total[i][point]/odd_ref[i];
                                acum+=or[i][point];
                        }
                        or_total[point]=acum/n_ind;
                }

                double[][] or_out = CGFunctions.JoinArraybyCol(xs, xs_orig, or_total,
                    null, null);
                return or_out;

    }
    public static double[] getBoot(double[] estimated, int bootseed) throws IOException,
         Exception{
        double[] observed = new double[estimated.length];
        Rand_Number r = new Rand_Number(bootseed);
                for(int i=0;i<estimated.length;i++){
                        double b = r.GetRandDouble();
                        if(b <= estimated[i]){
                                observed[i]=1;
                        }else{
                                observed[i]=0;
                        }
                }
        return observed;
    }
    public static Sample[] Bootstrap(Setup setup,int seed) throws IOException, Exception
        {
      if((setup.evaluation!="KFoldCV")&&(setup.evaluation!="KFoldval")) Exit.withError
          (10);
      int boot = seed;
      Sample[] result = new Sample[setup.maxslices];
      int n_col = setup.total_sample.alldata[0].length;
      int n_ind = setup.total_sample.alldata.length;


      switch(setup.bootstrap){
          case "binary":

                  double[] out = new double[n_ind];
        out = getBoot(setup.smoothed_test_pi,boot);
        double[][] alldata = new double[n_ind][n_col];
        alldata = CGFunctions.SetColumn(setup.total_sample.alldata, out, n_col-1);
        Sample total_sample = new Sample(alldata);
        CGFunctions.WriteCSV(alldata,setup.outputpath+setup.outputfolder+"/lists/
            alldata_"+setup.seed3actual+".csv");
        total_sample.Slicing(setup.maxslices);
        result =  total_sample.GetSamplesCV(setup.evaluation, setup.kfolds,setup.
            maxslices);
        int i =0;
        for(int k=0;k<setup.maxslices;k++){
                for(int z=0;z<result[k].testdata.length;z++){
                        result[k].testdata[z][result[0].n_cov]=out[i];
                        i++;
                }
        }
         break;
         case "bc":
                  Rand_Number r = new Rand_Number(boot);
                  Sample total_sample2 = new Sample(setup.total_sample.alldata);
                  total_sample2.Slicing(setup.maxslices);
         result =  total_sample2.GetSamplesCV(setup.evaluation, setup.kfolds,setup.
             maxslices);
         for(int k=0;k<setup.maxslices;k++){
            int n_ind2=result[k].testdata.length;
            result[k].testdata = new double[n_ind2][n_col];
            for(int z=0;z<n_ind2;z++){
```

```
                int row = r.GetRandInt(0,n_ind-1);
                    result[k].testdata[z]=setup.total_sample.alldata[row];
        }
    }
        break;
        default:

        break;

    }
    return(result);
}
public static double[][] BernoulliBootStrap(Setup setup,int bootmax,NodeTree
    node_ref, NodeTree node_smoothed, int n_points, int col) throws IOException,
    Exception{
      double[][] result = null;
      Setup setup2 = setup.deepCopy();
      double[] test_pi = GetColumn(node_smoothed.outtest.getKey(), 0);
      for(int boot=0;boot<bootmax;boot++){
              if(boot>0){

                                  if(boot==1){
                                        double[][] result2 = new double[1][n_points+1];
                                    result2[0] = Mathf.transposeMatrix(CGFunctions.
                                         OddsRatio(n_points,setup2,col,node_smoothed.
                                         cromossom))[2];
                                result = CGFunctions.JoinArray(result,result2);
                                  }else{
                                        double[] out = getBoot(test_pi,boot);
                                //***************TEST SAMPLE = ALL SAMPLE
                                     **********************//
                                CGFunctions.SetColumn(setup2.total_sample.alldata, out,
                                     setup2.total_sample.n_cov);
                                double[][] result2 = new double[1][n_points+1];
                                    result2[0] = Mathf.transposeMatrix(CGFunctions.
                                         OddsRatio(n_points,setup2,col,node_ref.cromossom)
                                         )[2];
                                    result = CGFunctions.JoinArray(result,result2);
                                  }

                                  CGFunctions.WriteCSV(result,setup.outputpath+setup.
                                       outputfolder+"/values/ORoutCI"+boot+".csv");

              }else{
                                    double[][] result2 = Mathf.transposeMatrix(
                                         CGFunctions.OddsRatio(n_points,setup2,col,
                                         node_ref.cromossom));
                                result = CGFunctions.JoinArray(result,result2);
              }
              //CGFunctions.Savexls(setup2.outputpath+setup2.outputfolder+"/values/
                   ORoutteste"+boot+".xls","ORteste", setup2.total_sample.alldata);
              CGFunctions.Print("*************************"+boot);
      }
      return result;
}
public static void Calibration_Plots(double[][] fileout){
    int k = 10;//number of partitions
    //CGFunctions.printArray(fileout);
    double[][] arrsorted = CGFunctions.SortAndClassify(fileout,1);
    //CGFunctions.printArray(arrsorted);
    double[][] soma = new double[k][4];
    for(int a = 0; a < arrsorted.length ; a++) {
            soma[(int)(arrsorted[a][2]*k)][0]+=arrsorted[a][0];
            soma[(int)(arrsorted[a][2]*k)][1]+=arrsorted[a][1];
            soma[(int)(arrsorted[a][2]*k)][2]=arrsorted[a][2];
            soma[(int)(arrsorted[a][2]*k)][3]+=arrsorted[a][3];
    }
    double[][] yaxis = new double[soma.length][3];
    double[] xaxis = new double[soma.length];

    for(int a = 0; a < soma.length ; a++) {
            yaxis[a][0]=soma[a][0]/soma[a][3];
            yaxis[a][1]=soma[a][1]/soma[a][3];
            yaxis[a][2]=soma[a][2];
            xaxis[a]=soma[a][2];
    }
    String[] ystr = {"estimadas","observ","x"};
    Plotting figure = new Plotting("Calibration Plot", "x", ystr, xaxis, yaxis);
    figure.pack();
    RefineryUtilities.centerFrameOnScreen(figure);
    figure.setVisible(true);
    CGFunctions.Print("**CalPlot**");
    CGFunctions.printArray(xaxis);
```

```
        CGFunctions.Print("--------");
        CGFunctions.printArray(yaxis);
        CGFunctions.Print("********");
        }
    public static String ReplaceChar(String word, int[] oldchar,String newchar){
        String result = word;
        for(int i:oldchar){
                result = result.replace(String.valueOf((char) i), newchar);
        }
        return(result);
    }
    public static void PrintChar(String word){
        System.out.println("*****BEGIN*********");
        System.out.println(word);
        byte []bytes=word.getBytes();for(byte b:bytes) System.out.println(b);
        System.out.println("*****END*********");
    }
    public static Pair<double[][],double[][]> getCzadoSimulatedData(int seed,int n,
         double niu0,double psi1,double psi2,String xdist) throws REngineException{
                double[][] result=new double[n][3];
                Rand_Number r = new Rand_Number(seed);
                double[] X1 = new double[n];
                double[] X2 = new double[n];
                double[] fx1 = new double[n];
                double[] fx2 = new double[n];
                double[] fx3 = new double[n];
                double[] fx4 = new double[n];
                double[] fx5 = new double[n];
                double[] niu = new double[n];
                double[] f = new double[n];
                double[] pr = new double[n];
                double[] y = new double[n];
                for(int i=0;i<n;i++){
                    if(xdist=="normal"){
                        X1[i] = r.GetRandNormal(-0.3,0.1);
                                X2[i] = r.GetRandNormal(-0.3,0.1);
                    }

                    if(xdist=="uniforme"){
                                X1[i] = r.GetRandDouble(-3,3);
                                X2[i] = r.GetRandDouble(-2,2);
                    }
                    if(xdist=="uniforme2"){
                        X1[i] = r.GetRandDouble(-0.9,0.9);
                                X2[i] = r.GetRandDouble(-0.9,0.9);
                    }
                    fx1[i] = X1[i];
                        fx2[i] = Math.pow(X2[i],2);
                        niu[i] = fx1[i]+fx2[i];
                        f[i] = Mathf.czadoV2(niu[i],niu0,psi1,psi2).getKey();
                        pr[i] = Mathf.czadoV2(niu[i],niu0,psi1,psi2).getValue();
                        double b = r.GetRandDouble();
                        if(b <= pr[i]){
                                y[i]=1;
                        }else{
                                y[i]=0;
                        }
                        result[i][0]=X1[i];
                        result[i][1]=X2[i];
                        result[i][2]=y[i];
                }
                return(new Pair(result,CGFunctions.JoinArraybyCol(pr,f,niu,X1,X2)));
        }
    public static Pair<double[][],double[][]> getArandaSimulatedData(int seed,int n,
         double alfa,String xdist) throws REngineException{
                double[][] result=new double[n][3];
                Rand_Number r = new Rand_Number(seed);
                double[] X1 = new double[n];
                double[] X2 = new double[n];
                double[] fx1 = new double[n];
                double[] fx2 = new double[n];
                double[] niu = new double[n];
                double[] f = new double[n];
                double[] pr = new double[n];
                double[] y = new double[n];
                for(int i=0;i<n;i++){
                    if(xdist=="normal"){
                        X1[i] = r.GetRandNormal(-0.3,0.1);
                                X2[i] = r.GetRandNormal(-0.3,0.1);
                    }
                    if(xdist=="uniforme"){
                                X1[i] = r.GetRandDouble(-3,3);
                                X2[i] = r.GetRandDouble(-2,2);
```

```
                }
                fx1[i] = X1[i];
                    fx2[i] = Math.pow(X2[i],2);

                    niu[i] = fx1[i]+fx2[i];
                    f[i] = Mathf.aranda(niu[i], alfa);
                    pr[i] = Mathf.aranda(niu[i], alfa);

                    double b = r.GetRandDouble();
                    if(b <= pr[i]){
                            y[i]=1;
                    }else{
                            y[i]=0;
                    }
                    result[i][0]=X1[i];
                    result[i][1]=X2[i];
                    result[i][2]=y[i];
            }
            return(new Pair(result,CGFunctions.JoinArraybyCol(pr,f,niu,X1,X2)));
    }
public static Pair<double[][],double[][]> getSimulatedData(int seed,int n,double
     alfa,String xdist) throws REngineException{
            double[][] result=new double[n][5];
            Rand_Number r = new Rand_Number(seed);
            double[] X1 = new double[n];
            double[] X2 = new double[n];
            double[] X3 = new double[n];
            double[] X4 = new double[n];
            double[] fx1 = new double[n];
            double[] fx2 = new double[n];
            double[] fx3 = new double[n];
            double[] fx4 = new double[n];
            double[] niu = new double[n];
            double[] f = new double[n];
            double[] pr = new double[n];
            double[] y = new double[n];
            for(int i=0;i<n;i++){
                if(xdist=="normal"){
                     X1[i] = r.GetRandNormal(-0.3,0.1);
                             X2[i] = r.GetRandNormal(-0.3,0.1);
                             X3[i] = r.GetRandNormal(-0.3,0.1);
                             X4[i] = r.GetRandNormal(-0.3,0.1);
                }
                if(xdist=="uniforme"){
                             X1[i] = r.GetRandDouble(-2,2);
                             X2[i] = r.GetRandDouble(-2,2);
                             X3[i] = r.GetRandDouble(-2,2);
                             X4[i] = r.GetRandDouble(-2,2);
                }
                fx1[i] = X1[i];
                    fx2[i] = X2[i];
                    fx3[i] = X3[i];
                    fx4[i] = X4[i];


                    niu[i] = fx1[i]+fx2[i]+fx3[i]+fx4[i];
                    f[i] = Mathf.aranda(niu[i], alfa);
                    pr[i] = Mathf.aranda(niu[i], alfa);

                    double b = r.GetRandDouble();
                    if(b <= pr[i]){
                            y[i]=1;
                    }else{
                            y[i]=0;
                    }
                    result[i][0]=X1[i];
                    result[i][1]=X2[i];
                    result[i][2]=X3[i];
                    result[i][3]=X4[i];
                    result[i][4]=y[i];
            }
            return(new Pair(result,CGFunctions.JoinArraybyCol(pr,f,niu,X1,X2)));
    }
public static double[][] getRealData(int seed,Setup setup) throws REngineException{
    double[][] result = null;
    try {

            double[][] alldata = null;
        DB inputdb = new DB(setup.inputpath+setup.inputfolder,setup.inputfilename,
             setup.inputtype);
        alldata = inputdb.getData(setup, setup.inputtype,seed);
        int n_cov = alldata[0].length-1;
        int n = alldata.length;
```

```
            result=new double[n][n_cov];
            for(int i=0;i<n;i++) result[i]=alldata[i];


                } catch (Exception e) {
                        CGFunctions.Print("Input Database does not exist");
                        e.printStackTrace();
                }

                return(result);


        }
}
```

Listagem A.27: Classe CGFunctions que agrega um conjunto de métodos auxiliares da plataforma CG-GANN.

```
package cgUtils;

import java.util.Arrays;
import java.util.Random;

import org.apache.commons.math3.util.Pair;

import Jama.Matrix;

public class Mathf{
 public static double min(double[] t) {
    double minimum = t[0];
         for (int i=1; i<t.length; i++) {
                  if (t[i] < minimum) {
                                minimum = t[i];
                  }
         }
         return minimum;
 }
 public static double max(double[] t) {
         double maximum = t[0];
         for (int i=1; i<t.length; i++) {
                if (t[i] > maximum) {
                     maximum = t[i];
                }
         }
         return maximum;
 }
 public static int min(int[] t) {
             int minimum = t[0];
                  for (int i=1; i<t.length; i++) {
                           if (t[i] < minimum) {
                                         minimum = t[i];
                           }
                  }
                  return minimum;
}
public static int max(int[] t) {
                  int maximum = t[0];
                  for (int i=1; i<t.length; i++) {
                        if (t[i] > maximum) {
                                maximum = t[i];
                        }
                  }
                  return maximum;
}
 public static double tangh(double x){
         return Math.tanh(x);
 }
 public static double linear(double x){
         return x;
 }
 public static double sigmoid(double x){
         return(1/(1+Math.exp(-x)));
 }
 public static double mix(double x, int i){
          if(i%2 == 0){
                  return Math.tanh(x);
          }else{
                  return(1/(1+Math.exp(-x)));
          }
 }
 public static double aranda(double x,double alfa){
          if(alfa*Math.exp(x)>-1){
                return(1-Math.pow((1+alfa*Math.exp(x)),-1/alfa));
          }else{
                  return(1);
          }
 }
 public static double invaranda(double y, double alfa){
          if(alfa>0){
                  return(Math.log((Math.pow((1-y),-alfa)-1)/alfa));
          }else{
                  return(Math.log(-Math.log(1-y)));
          }
 }
 public static double dinvaranda(double y, double alfa){
          if(alfa>0){
                  return((alfa*Math.pow(1-y,-alfa-1))/(Math.pow(1-y,-alfa)-1));
          }else{
                  return((-1/Math.log(1-y))*(1/(1-y)));
          }
 }
```

```
public static Pair<Double,Double> czadoV2(double x, double niu0, double psi1, double
    psi2){
        double f,pr;
        if(x>=niu0){
                        f=niu0+(Math.pow(x-niu0+1,psi1)-1)/psi1;
        }else{
                        f=niu0-(Math.pow(-x+niu0+1,psi2)-1)/psi2;
        }
        pr = Mathf.sigmoid(f);
        return(new Pair(f,pr));
}
public static double czado(double x, double niu0, double psi1, double psi2){
        double f,pr;
        if(x>=niu0){
                        f=niu0+(Math.pow(x-niu0+1,psi1)-1)/psi1;
        }else{
                        f=niu0-(Math.pow(-x+niu0+1,psi2)-1)/psi2;
        }
        pr = Mathf.sigmoid(f);
        return(pr);
}
public static double dczado(double x, double niu0, double psi1, double psi2){
        double dfdx,dprdf;
        if(x>=niu0){
                        dfdx=Math.pow((x-niu0+1),(psi1-1));
        }else{
                dfdx=Math.pow(-(-x+niu0+1),(psi2-1));
        }
        return(dsigmoid(czadoV2(x,niu0,psi1,psi2).getKey())*dfdx);
}
public static double dtangh(double x){
       return 1-Math.tanh(x)*Math.tanh(x);
}
public static double dlinear(double x){
       return 1;
}
public static double dsigmoid(double x){
   return(sigmoid(x)*(1-sigmoid(x)));
}
public static double dmix(double x, int i){
        if(i%2 == 0){
                return (1-Math.tanh(x)*Math.tanh(x));
        }else{
                return(sigmoid(x)*(1-sigmoid(x)));
        }
}
public static double daranda(double x,double alfa){
   if(alfa*Math.exp(x)>-1){
               return(Math.exp(x)*Math.pow((1+alfa*Math.exp(x)),(-1-alfa)/alfa));
   }else{
       return(0);
   }
}
public static double logit_link(double y){
           return(Math.log(y/(1-y)));
}
public static double norm(double[] vector){
        double result = 0;
        for(int row=0; row<vector.length;row++){
                result+=vector[row]*vector[row];
        }
        result = Math.sqrt(result);
        return result;
}
public static double[][] transposeMatrix(double [][] m){
    double[][] temp = new double[m[0].length][m.length];
    for (int i = 0; i < m.length; i++)
        for (int j = 0; j < m[0].length; j++)
            temp[j][i] = m[i][j];
    return temp;
}
public static Matrix Jacobian(double[][] GMap,double[] Error){
         double[][] J = new double[GMap.length][GMap[0].length];
         for(int row=0; row<GMap.length;row++){
                    for(int col=0; col<GMap[0].length;col++){
                            J[row][col]=GMap[row][col]/Error[row];
                    }
         }
         return new Matrix(J);
}
public static Matrix InvertedHessian(Matrix Jacobian,double param){
           Matrix JtJ = Jacobian.transpose().times(Jacobian);
           Matrix I = Matrix.identity(JtJ.getRowDimension(), JtJ.getColumnDimension());
```

```
            Matrix lamdaI = I.times(param);
            Matrix Hessian=JtJ.plus(lamdaI);
            return Hessian.inverse();
 }
 public static Matrix Gradient(double[][] GMap){
            double[] G = new double[GMap[0].length];
            for(int row=0; row<GMap.length;row++){
               for(int col=0; col<GMap[0].length;col++){
                        G[col]+=GMap[row][col];
               }
            }
            return new Matrix(G,G.length);
 }
 public static double[] Normalize_min_max(double minimum,double maximum, double[] array)
     {
            for(int row=0; row<array.length;row++){
                        array[row]=-1+2*(array[row]-minimum)/(maximum-minimum);
                }
            return(array);
 }
 public static double Normalize_min_max(double minimum,double maximum, double value){
            double result;
        result =-1+2*(value-minimum)/(maximum-minimum);
            return(result);
 }
 public static double getNormalizednumber(double minimum,double maximum, double number){

            double result=(number-minimum)/(maximum-minimum);

            return(result);
}
 public static double SumArray(double[] arraydbl){
           double result = 0;
           for(int row=0; row<arraydbl.length;row++){
                    result += arraydbl[row];
           }
           return result;
 }
 public static int SumArray(int[] arrayint){
           int result = 0;
           for(int row=0; row<arrayint.length;row++){
                    result += arrayint[row];
           }
           return result;
 }
 public static double sdev(int n_ind, double[] array, double MSE){
           double acum = 0;
       for (int ind=0;ind<n_ind;ind++){
           acum+=Math.pow((array[ind]-MSE),2);
       }
       return(Math.sqrt(acum/(n_ind-1)));
 }
 public static int minindex(double[] array){
           int result = 0;
           for(int row=0; row<array.length;row++){
                    if(array[row]<array[result]){
                             result = row;
                    }
           }
       return(result);
 }
 public static double MSE(double[][] array){
           double result=0;
           for(int row=0; row<array.length;row++){
                    result+=(array[row][1]-array[row][0])*(array[row][1]-array[row][0]);
           }
           result = result/array.length;
           return result;
 }
public static double mediana(double[] array){
        Arrays.sort(array);
        double median;
        if (array.length % 2 == 0){
                median = ((double)array[array.length/2] + (double)array[array.length/2 -
                     1])/2;
        }else{
                median = (double) array[array.length/2];
        }
        return(median);
}
public static double[] residuals(double[][] array){
           double[] result= new double[array.length];
           for(int row=0; row<array.length;row++){
```

```
                    result[row]=(array[row][1]-array[row][0]);
            }
            return result;
 }
 public static double[] dist(double[][] array){
            double[] result = new double[array.length];
            for(int row=0; row<array.length;row++){
                    result[row]=Math.pow(array[row][1]-array[row][0],2);
            }
            return result;
}
 public static double algsift (double x, int numbsif){
         int k = (int) Math.pow(10, numbsif);
     double fract;
     double whole;
     if ( x > 0 ){
       whole = Math.floor(x);
       fract = Math.floor( (x - whole) * (double)k) / (double)k;
     } else {
       whole = Math.ceil(x);
       fract = Math.ceil( (x - whole) * (double)k) / (double)k;
     }

     return whole + fract;
   }
 public static double algsifr (double x, int numbsif){
         int k = (int) Math.pow(10, numbsif);
     double number = Math.round( x * (double)k) / (double)k;
     return number;
 }
 public static double H(double[][] arraydbl1){
         int maxg=10;
         double[] E = new double[maxg];
         double[] O = new double[maxg];
         double[] erro = new double[maxg];
         double[] pi = new double[maxg];
         double[] H = new double[maxg];
         double[][] arraydbl2=CGFunctions.Sort(arraydbl1);
     Sample group = new Sample(arraydbl2);
     group.Slicing(maxg);
     for(int g=0;g<maxg;g++){
         E[g]=SumArray(CGFunctions.GetColumn(group.slice[g].data, 0));
         O[g]=SumArray(CGFunctions.GetColumn(group.slice[g].data, 1));
         erro[g]=O[g]-E[g];
         pi[g]=E[g]/group.slice[g].data.length;
         H[g]=Math.pow(erro[g],2)/(E[g]*(1-pi[g]));
     }
     double Hres=SumArray(H);
         return Hres;
 }
 public static double AUC(double[][] fileout){
                 int result;
                 int n_cutoff=1000;
                 int total_ind = fileout.length;
                 double[][] classifier = new double[total_ind][n_cutoff+1];
                 double[] tp=new double[n_cutoff+1];
                 double[] tn=new double[n_cutoff+1];
                 double[] fp=new double[n_cutoff+1];
                 double[] fn=new double[n_cutoff+1];
                 double[] acc=new double[n_cutoff+1];
                 double[] sens=new double[n_cutoff+1];
                 double[] spec=new double[n_cutoff+1];
                 double[] onemspec=new double[n_cutoff+1];
                 double[] trapezoid=new double[n_cutoff+1];
                 double auc=0;
                 double[] eff=new double[n_cutoff+1];
                 double[] ppv=new double[n_cutoff+1];
                 double[] npv=new double[n_cutoff+1];
                 double[] phi=new double[n_cutoff+1];
        for(int cutoff=1; cutoff<=n_cutoff;cutoff++){
                 double threshold = (double)(cutoff)/n_cutoff;
                 for(int ind=0; ind<fileout.length;ind++){
                         if(fileout[ind][0]>=threshold){
                                 result=1;
                         }else{
                                 result=0;
                         }
                         classifier[ind][cutoff]=fileout[ind][1]-result;
                         if((result==1)&&(fileout[ind][1]==1)){
                                         classifier[ind][cutoff]=2;//code for true
                                                 positive
                         }
```

```
                        //-1=false positive,0=true negative,1=false negative,2=true
                            positive
                        if(classifier[ind][cutoff]==-1){
                                fp[cutoff]+=1;
                        }
                if(classifier[ind][cutoff]==0){
                                tn[cutoff]+=1;
                        }
                if(classifier[ind][cutoff]==1){
                                fn[cutoff]+=1;
                        }
                if(classifier[ind][cutoff]==2){
                                tp[cutoff]+=1;
                        }
                    }
            }
                    sens[0]=1;
                    spec[0]=0;
                    onemspec[0]=1;
                    for(int cutoff=1; cutoff<=n_cutoff;cutoff++){
                        acc[cutoff] = (tp[cutoff]+tn[cutoff])/total_ind;
                        sens[cutoff] = tp[cutoff]/(tp[cutoff] + fn[cutoff]);
                        spec[cutoff] = tn[cutoff]/(tn[cutoff] + fp[cutoff]);
                        onemspec[cutoff]=1-spec[cutoff];
                        eff[cutoff] = (sens[cutoff]+spec[cutoff])/2;
                        trapezoid[cutoff]=(sens[cutoff]*(onemspec[cutoff-1]-onemspec[cutoff
                            ]))+0.5*((onemspec[cutoff-1]-onemspec[cutoff])*(sens[cutoff-1]-
                            sens[cutoff]));
                        auc+=trapezoid[cutoff];
                        ppv[cutoff] = tp[cutoff]/(tp[cutoff]+fp[cutoff]);
                        npv[cutoff] = tn[cutoff]/(tn[cutoff]+fn[cutoff]);
                        phi[cutoff] =(tp[cutoff]*tn[cutoff]-tp[cutoff]*tn[cutoff])/Math.sqrt
                            ((tp[cutoff]+fp[cutoff])*(tp[cutoff]+fn[cutoff])*(tn[cutoff]+tp[
                            cutoff])*(tn[cutoff]+fn[cutoff]));
                    }
                    System.out.println("AUC="+auc);
        return auc;
            }
    public static double bestaccuracy(double[][] fileout){
                    int result;
                    int n_cutoff=1000;
                    int total_ind = fileout.length;
                    double[][] classifier = new double[total_ind][n_cutoff+1];
                    double[] tp=new double[n_cutoff+1];
                    double[] tn=new double[n_cutoff+1];
                    double[] fp=new double[n_cutoff+1];
                    double[] fn=new double[n_cutoff+1];
                    double[] acc=new double[n_cutoff+1];
                for(int cutoff=1; cutoff<=n_cutoff;cutoff++){
                      double threshold = (double)(cutoff)/n_cutoff;
                      for(int ind=0; ind<fileout.length;ind++){
                            if(fileout[ind][0]>=threshold){
                                    result=1;
                            }else{
                                    result=0;
                            }
                            classifier[ind][cutoff]=fileout[ind][1]-result;
                            if((result==1)&&(fileout[ind][1]==1)){
                                        classifier[ind][cutoff]=2;//code for true
                                            positive
                            }
                            //-1=false positive,0=true negative,1=false negative,2=true
                                positive
                            if(classifier[ind][cutoff]==-1){
                                    fp[cutoff]+=1;
                            }
                if(classifier[ind][cutoff]==0){
                                    tn[cutoff]+=1;
                            }
                if(classifier[ind][cutoff]==1){
                                    fn[cutoff]+=1;
                            }
                if(classifier[ind][cutoff]==2){
                                    tp[cutoff]+=1;
                            }
                        }
                }
                    double bestacc = 0;
                    for(int cutoff=1; cutoff<=n_cutoff;cutoff++){
                        acc[cutoff] = (tp[cutoff]+tn[cutoff])/total_ind;
                        if(acc[cutoff]>bestacc) bestacc=acc[cutoff];
                    }
            return bestacc;
```

```
 }
 public static int[] Int2Bin(int number, int numberofdigits){
            String maxAmpStr = Integer.toBinaryString(number);
            String left = "";
            for (int i=0; i<(numberofdigits-maxAmpStr.length()); i++){
                left = "0"+left;
            }
            maxAmpStr = left + maxAmpStr;
            char[] arr = maxAmpStr.toCharArray();
            int[] binaryarray = new int[numberofdigits];
            for (int i=0; i<maxAmpStr.length(); i++){
                if (arr[i] == '1'){
                    binaryarray[i] = 1;
                }
                else if (arr[i] == '0'){
                    binaryarray[i] = 0;
                }
            }
        return (binaryarray);
 }
 public static int Bin2Int(int[] number){
        int intnumber=0;

        for(int i=0;i<number.length;i++){
                intnumber += (int) Math.pow(2,i)*number[number.length-i-1];

        }

        return intnumber;
 }
 public static double SBC(double MSE, int n_ind,int n_params){
        double result;
        int  n = n_ind;
        int k = n_params;
        result = n*Math.log(MSE)+k*Math.log(n);
        return result;
 }
 public static double CalibrationPlotArea(double[][] fileout){
       double result = 0;
       int k = 10;
    double[][] arrsorted = CGFunctions.SortAndClassify(fileout,1);
    double[][] soma = new double[k][4];
    for(int a = 0; a < arrsorted.length ; a++) {
                soma[(int)(arrsorted[a][2]*k)][0]+=arrsorted[a][0];
                soma[(int)(arrsorted[a][2]*k)][1]+=arrsorted[a][1];
                soma[(int)(arrsorted[a][2]*k)][2]=arrsorted[a][2];
                soma[(int)(arrsorted[a][2]*k)][3]+=arrsorted[a][3];
        }
    for(int a = 0; a < k ; a++) {
        result+=Math.abs(soma[a][0]-soma[a][1])/soma[a][3];
    }
    return(result);
 }
 public static double variance(double[] arraydbl){
        double var=0;
        double sq_a=0;
        for(int row=0; row<arraydbl.length;row++){
                sq_a += Math.pow(arraydbl[row],2);
        }
        double av_a = SumArray(arraydbl)/arraydbl.length;
        double av_of_sq_a = sq_a/arraydbl.length;
        double sq_of_av_a = Math.pow(av_a,2);
        var = av_of_sq_a-sq_of_av_a;
        return var;
 }
 public static double[] getNormalRandomNumbers(double u, double s, int n_syn, int seed){
        Random generator = new Random(seed);
    double[] InitW = new double[n_syn];
        for(int z=0; z<n_syn;z++){
                        InitW[z] = s*generator.nextGaussian()+u;
        }
        return InitW;
}
public static double[] getSkewnessNormalRandomNumbers(double u, double s, int n_syn, int
     seed){
        Random generator = new Random(seed);
    double[] InitW = new double[n_syn];
        for(int z=0; z<n_syn;z++){
                        //InitW[z] =
        }
        return InitW;
}
public static double[] getUniformRandomNumbers(double a, double b, int n_syn, int seed){
```

```
        Random generator = new Random(seed);
        double[] InitW = new double[n_syn];
        for(int z=0; z<n_syn;z++){
                        InitW[z] = a + (b-a)*generator.nextDouble();
        }
        return InitW;
}


}
```

Listagem A.28: Classe Mathf que agrega um conjunto de métodos matemáticos auxiliares.

```
package cgUtils;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
public class Sample {
        public double[][] alldata;
        public double[][] alldata_original;
        public double[][] alldata_aux;
        public double[][] traindata;
        public double[][] valdata;
        public double[][] testdata;
        public double[] externaldata;
        public Slice[] slice;
        public double norm_1;
        public int n_cov,n_ind;
        public int kfold,maxslices;
        public Sample(double[][] Newalldata){
                alldata = new double[Newalldata.length][Newalldata[0].length];
                alldata_original = new double[Newalldata.length][Newalldata[0].length];
                alldata_aux = new double[Newalldata.length][Newalldata[0].length];
                for(int row=0;row<Newalldata.length;row++){
                        for(int col=0;col<Newalldata[0].length;col++){
                                alldata[row][col] = Newalldata[row][col];
                                alldata_original[row][col] = Newalldata[row][col];
                                alldata_aux[row][col] = Newalldata[row][col];
                        }
                }
                traindata = null;
                valdata = null;
                testdata = null;
                externaldata = null;
                n_cov = alldata[0].length-1;
                n_ind = alldata.length;
        }
        public void Shuffle(int seed){
                int n_ind = alldata.length;
                int n_cov = alldata[0].length-1;
                int[] row = new int[n_ind];
                int[] newrow = new int[n_ind];
                double[][] output = new double[n_ind][n_cov+1];
                for(int ind=0; ind<n_ind;ind++){
                        row[ind]=ind;
                }
                newrow = CGFunctions.Permute(row,seed);
                for(int i=0; i<n_ind;i++){
                  for(int col=0; col<n_cov;col++){
                  output[i][col]=alldata[newrow[i]][col];
                  }
                  output[i][n_cov]=alldata[newrow[i]][n_cov];
                }
                alldata = output;
        }
        public void Normalize(){
                int n_cov = alldata[0].length-1;
                for(int col=0; col<n_cov;col++){
                        double[] aux = CGFunctions.GetColumn(alldata, col);
                        double maximum = Mathf.max(aux);
                        double minimum = Mathf.min(aux);
                        if(Math.abs(minimum)<=1 && Math.abs(maximum)<=1){

                        }else{
                                double[] norm_col = Mathf.Normalize_min_max(minimum,
                                    maximum,CGFunctions.GetColumn(alldata,col));
                                norm_1 = Mathf.Normalize_min_max(minimum,maximum,1);
                                CGFunctions.SetColumn(alldata,norm_col,col);
                        }
                }
        }
        public void Bootstrap(int boot){
                double[][] result=new double[n_ind][alldata[0].length];
                if(boot==1){
                        for(int i=0;i<n_ind;i++) result[i]=alldata[i];
                  }else{
                        Rand_Number r = new Rand_Number(boot);
                        for(int i=0;i<n_ind;i++){
                                int row = r.GetRandInt(0,n_ind-1);
                                result[i]=alldata[row];

                        }
                }
                for(int i=0;i<n_ind;i++) alldata_aux[i]=alldata[i];
                alldata = new double[alldata.length][alldata[0].length];
                for(int col=0; col<alldata[0].length;col++){
```

```
                    for(int row=0; row<n_ind;row++){
                            alldata[row][col]=result[row][col];
                    }
            }

    }
    public void Slicing(int maxslices){
            int n_ind = alldata.length;
            int ips = (int) Mathf.algsift((double)(n_ind/maxslices),0);//individuals
                per slice
        slice=new Slice[maxslices];
        for(int id=0; id<maxslices;id++){
                            int begin=id*ips;
                            int end=(id+1)*ips;
                            if(id==(maxslices-1)){
                                    end=n_ind;
                    }
        slice[id] = new Slice(alldata,begin,end);
        }
    }
    public double[][] JoinSlices(int[] slice_id){
            double[][] result = null;
            for(int a=0;a<slice_id.length;a++){
              result = CGFunctions.JoinArray(result,slice[slice_id[a]].data);
            }
            return result;
    }
    public double[][] GetwithoutSlice(int slice_id,int maxslices){
            double[][] output = null;
    int[] set = new int[maxslices-1];
    int a =0;
            for(int k=0; k<maxslices;k++){
                    if(k!=slice_id){
                                    set[a]=k;
                                    a+=1;
                    }
            }
            output = JoinSlices(set);
            return output;
    }
    public Sample[] GetSamplesCV(String evaluation, int kfolds, int Newmaxslices)
         throws FileNotFoundException, IOException {
              Sample[] result = new Sample[kfolds];
              maxslices = Newmaxslices;
              switch(evaluation){
              case "KFoldCV":
                      maxslices = kfolds;
                      this.Slicing(maxslices);
                      int[] slice_id = new int[maxslices-2];
                            for(int k=0;k<maxslices;k++){
                              result[k] = new Sample(alldata);
                              result[k].testdata = slice[k].data;
                              for(int z=0;z<slice_id.length;z++){

                                      if((k+z+1)>=maxslices){
                                              slice_id[z] = k+z+1-maxslices;
                                      }else{
                                              slice_id[z] = k+z+1;
                                      }
                              }
                              result[k].traindata = JoinSlices(slice_id);
                              if(k<1){
                          result[k].valdata = slice[k+maxslices-1].data;
                      }else{
                          result[k].valdata =  slice[k-1].data;
                      }

                        }
              break;
              case "HoldOutCV":
                      this.Slicing(maxslices);
                      result[0] = new Sample(alldata);
                  result[0].traindata = GetwithoutSlice(0, maxslices);
                  result[0].valdata = slice[0].data;
                  result[0].testdata = slice[0].data;
              break;
              case "InSample":

                      this.Slicing(maxslices);
                      result[0] = new Sample(alldata);
                  result[0].traindata = GetwithoutSlice(0, maxslices);
                  result[0].valdata = slice[0].data;
                  result[0].testdata = alldata;
```

```
                    break;
                    case "KFoldval":

                            this.Slicing(maxslices);
                            for(int k=0;k<maxslices;k++){
                                    result[k] = new Sample(alldata);
                                    result[k].valdata = slice[k].data;
                                    result[k].testdata = alldata_aux;
                                    result[k].traindata = new Sample(GetwithoutSlice(k,
                                         maxslices)).alldata;
                            }
                    break;
                  }

                    return result;
        }
        public static void main(String[] args) throws Exception{


    }
}
```

Listagem A.29: Classe Sample que implementa e prepara a amostra de treino, validação e teste.

```
package cgUtils;

public class Slice {
        public double[][] data;
        public Slice(double[][] Alldata, int begin, int end) {
                data = new double[end-begin][Alldata[0].length];
                for(int row=begin; row<end ;row++){
                        for(int col=0;col<Alldata[0].length;col++){
                          data[row-begin][col]=Alldata[row][col];
                        }
                }
        }
}
```

Bibliografia

Agresti, A. (2003). *Categorical Data Analysis*. Wiley Series in Probability and Statistics. Wiley.

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In Petrov, B. N. e Csaki, F., editors, *Second International Symposium on Information Theory*, pages 267–281, Budapest. Akadémiai Kiado.

Amaral Turkman, M.A. e Silva, G. (2000). *Modelos Lineares Generalizados - da Teoria á Prática*. Edições SPE,Lisboa.

Aranda-Ordaz, F. J. (1981). On two families of transformations to additivity for binary response data. *Biometrika*, 68(2):357–363.

Baum, E. B. e Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1(1):151–160.

Berkane, M., Kano, Y., e Bentler, P. M. (1994). Pseudo maximum likelihood estimation in elliptical theory: Effects of misspecification. *Computational Statistics & Data Analysis*, 18(2):255–267.

Berkson, J. (1944). Application of the logistic function to bioassay. *J. Amer. Statist. Ass.*, 39:375–365.

Bishop, C. (1995). *Neural networks for pattern recognition*. Clarendon Press.

Bliss, C. (1935). The calculation of the dosage-mortality curve. *Annals of Applied Biology*, 22(1):134–167.

Bras-Geraldes, C., Papoila, A., Xufre, P., e Diamantino, F. (2013). Generalized additive neural networks for mortality prediction using automated and genetic algorithms. In *IEEE 2nd International Conference on Serious Games and Applications for Health, SeGAH 2013, Vilamoura, Portugal*, pages 1–8. IEEE.

Burnham, K. e Anderson, D. (2003). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer New York.

Cadarso-Suárez, C., Roca-Pardiñas, J., Figueiras, A., e González-Manteiga, W. (2005). Non-parametric estimation of the odds ratios for continuous exposures using generalized additive models with an unknown link function. *Statistics in medicine*, 24(8):1169–1184.

Carroll, R. J., Fan, J., Gijbels, I., e Wand, M. (1998). Generalized partially linear single-index models. *Journal of the American Statistical Association*, 92:477–489.

Chen, T.-c., Han, D.-j., Au, F. T. K., e Tham, L. G. (2003). Acceleration of levenberg-marquardt training of neural networks with variable decay rate. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 1873–1878 vol.3.

Craven, P. e Wahba, G. (1978). Smoothing noisy data with spline functions. *Numerische Mathematik*, 31(4):377–403.

Czado, C. (1992). On link selection in generalized linear models. In Fahrmeir, L., Francis, B., Gilchrist, R., e Tutz, G., editors, *Advances in GLIM and Statistical Modelling*, volume 78 of *Lecture Notes in Statistics*, pages 60–65. Springer New York.

de Waal, D. A., Campher, S., e du Toit, J. V. (2010). Estimating those transformations that produce the best-fitting additive model: Smoothers versus universal approximators. In *Proceedings of the IADIS international conference on Intelligent Systems and Agents 2010 and European Conference Data Mining 2010*, pages 61–68. António Palma dos Reis and Ajith P. Abraham.

de Waal, D. A. e du Toit, J. V. (2007). Generalized additive models from a neural network perspective. In *Proceedings of the 7th IEEE International Conference on Data Mining, ICDM 2007, Omaha, Nebraska*, pages 265–270. IEEE.

de Waal, D. A. e du Toit, J. V. (2011). Automation of generalized additive neural networks for predictive data mining. *Applied Artificial Intelligence*, 25(5):380–425.

de Waal, D. A., du Toit, J. V., e de la Rey., T. (2009). A flexible generalized link function for credit scoring. In *Credit Scoring & Credit Control XI*, Scotland. University of Edinburgh Management School.

Dias, F. M., Antunes, A., Vieira, J., e Mota, A. M. (2004). Implementing the levenberg-marquardt algorithm on-line: a sliding window approach with early stopping. *2nd IFAC Workshop on Advanced Fuzzy/Neural Control*.

Dicker, R. C., Coronado, F., Koo, D., e Parrish, R. G. (2006). *Principles of epidemiology in public health practice; an introduction to applied epidemiology and biostatistics*. U.S. Department of Health and Human Services, Centers for Disease Control and Prevention (CDC).

Dobson, A. (2010). *An Introduction to Generalized Linear Models, Second Edition*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis.

Du Toit, J. V. (2006). *Automated Construction of Generalized Additive Neural Networks for Predictive Data Mining*. PhD thesis, School for Computer, Statistical and Mathematical Sciences, North-West University, South Africa.

du Toit, J. V. e de Waal, D. A. (2010). Spam detection using generalized additive neural networks. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2010*.

Duch, W. e Jankowski, N. (1999). Survey of neural transfer functions. *Neural Computing Surveys*, 2:163–213.

Dybowski, R. e Roberts, S. J. (2001). Confidence intervals and prediction intervals for feed-forward neural networks. In *Clinical Applications of Artificial Neural Networks*, pages 298–326. University Press.

Efron, B. e Tibshirani, R. (1994). *An Introduction to the Bootstrap*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874.

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London*, pages 309–368.

Gama, J., Ponce de Leon Carvalho, A., Facelli, K., Lorena, A. C., e Oliveira, M. (2012). *Extração de conhecimento de dados*. Edições Sílabo.

Gasser, T. e Müller, H.-G. (1979). *Smoothing Techniques for Curve Estimation: Proceedings of a Workshop held in Heidelberg, April 2–4, 1979*, chapter Kernel estimation of regression functions, pages 23–68. Springer Berlin Heidelberg, Berlin, Heidelberg.

Green, P. e Silverman, B. (1993). *Nonparametric Regression and Generalized Linear Models: A roughness penalty approach*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis.

Gu, C. (1992). Penalized likelihood regression: a bayesian analysis. *Statistica Sinica*, (2):pp. 255–64.

Guerrero, V. M. e Johnson, R. A. (1982). Use of the box-cox transformation with binary response models. *Biometrika*, 69(2):pp. 309–314.

Hagan, M. T., Demuth, H. B., e Beale, M. (1996). *Neural Network Design*. PWS Publishing Co., Boston, MA, USA.

Hanley, J. A. e McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143:29–36.

Härdle, W. (2004). *Nonparametric and Semiparametric Models*. Springer Series in Statistics. Springer Berlin Heidelberg.

Hastie, T. e Tibshirani, R. (1990). *Generalized additive models*. CRC Monographs on Statistics & Applied Probability. Chapman & Hall/CRC.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR.

Heskes, T. (1997). Practical confidence and prediction intervals. In Jordan, M. I. e Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, pages 176–182. MIT Press.

Hornik, K., Stinchcombe, M., e White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2:359–366.

Horowitz, J. (2009). *Semiparametric and Nonparametric Methods in Econometrics*. Springer Series in Statistics. Springer New York.

Horowitz, J. L. e Mammen, E. (2004). Nonparametric estimation of an additive model with a link function. *Annals of Statistics*, 32(6):2412–2443.

Huang, Y., Pepe, M., e Feng, Z. . (1181-1188). Evaluating the predictiveness of a continuous marker. *Biometrics*, 63.

Ichimura, H. (1993). Semiparametric least squares (sls) and weighted sls estimation of single-index models. *Journal of Econometrics*, 21:157–178.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Laurentiis, M. e Ravdin, P. M. (1994). Survival analysis of censored data: Neural network analysis detection of complex interactions between variables. *Breast Cancer Research and Treatment*, 32(1):113–118.

Li, K. e Duan, N. (1989). Regression analysis under link violation. *Annals of Statistics*, 17:1009–1052.

Marques, J. (2000). *Reconhecimento de padrões: métodos estatísticos e neuronais*. Coleção Ensino da Ciência e da Tecnologia. Instituto Superior Técnico.

McCulloch, W. e Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133.

Morgan, B. J. T. (1985). The cubic logistic model for quantal assay data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 34(2):pp. 105–113.

Muggeo, V. M. R. e Ferrara, G. (2008). Fitting generalized linear models with unspecified link function: A P-spline approach. *Computational Statistics and Data Analysis*, 52(5):2529–2537.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability and its Applications*, 9:141–142.

Nelder, J. A. e Wedderburn, R. W. M. (1972). Generalized linear models. *Journal of the Royal Statistical Society, Series A, General*, 135:370–384.

Ohno-Machado, L. (1996). *Medical Applications of Artificial Neural Networks: Connectionist Models of Survival*. PhD thesis, Stanford, CA, USA. AAI9620524.

Papoila, A. L. (2006). *Modelos Aditivos Generalizados em Análise de Sobrevivência*. PhD thesis, Faculdade de Ciências, Universidade de Lisboa, Lisbon, Portugal.

Plate, T., Band, P., Bert, J., e Grace, J. (1997). A comparison between neural networks and other statistical techniques for modelling the relationship between tobacco and alcohol and cancer. *Advances in Neural Information Processing*, (9):967–973.

Potts, W. J. E. (1999). Generalized additive neural networks. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge Discovery and Data mining*, KDD '99, pages 194–200, New York, NY. ACM.

Powell, J. L., Stock, J. H., e Stoker, T. M. (1989). Semiparametric Estimation of Index Coefficients. *Econometrica*, 57(6):1403–1430.

Pregibon, D. (1980). Goodness of link tests for generalized linear models. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 29(1):pp. 15–23, 14.

Prentice, R. L. (1976). A generalization of the probit and logit methods for dose response curves. *Biometrics*, 32(4):pp. 761–768.

Priestley, M. B. C. (1972). Nonparametric curve fitting. *Journal of the Royal Statistical Society*, 34:385–392.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Rasmussen, C. e Williams, C. (2005). *Gaussian Processes for Machine Learning*. Adaptive Computation And Machine Learning. MIT Press.

Riedmiller, M. e Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591.

Roca-Pardiñas, J., González-Manteiga, W., Febrero-Bande, M., Prada-Sánchez, J. M., e Cadarso-Suárez, C. (2004). Predicting binary time series of so2 using generalized additive models with unknown link function. *Environmetrics*, 15(7):729–742.

Ruppert, D. e Wand, M. P. (1994). Multivariate locally weighted least squares regression. *Ann. Statist.*, 22(3):1346–1370.

Samarasinghe, S. (2006). *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*. CRC Press.

Sarle, W. S. (1994). Neural networks and statistical models. In *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, pages 1538–1550, Cary, NC. SAS Institute.

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464.

Sharma, A. K. (2009). Effectiveness of heuristic rules for model selection in connectionist models to predict milk yield in dairy cattle. *TECHNIA - International Journal of Computing Science and Communication Technologies*, 2(1):384–386.

Smith, G. (2012). *Essential Statistics, Regression, and Econometrics*. Academic Press.

Stukel, T. A. (1988). Generalized logistic models. *Journal of the American Statistical Association*, 83(402):pp. 426–431.

Tibshirani, R. (1996). A comparison of some error estimates for neural network models. *Neural Computation*, 8:152–163.

Tibshirani, R. e Hastie, T. (1987). Local likelihood estimation. *Journal of the American Statistical Association*, 82:559–567.

Tu, J. V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

Walczak, S. e Cerpa, N. (1999). Heuristic principles for the design of artificial neural networks. *Information and Software Technology*, 41(2):107–117.

Weisberg, S. e Welsh, A. H. (1994). Adapting for the missing link. *The Annals of Statistics*, 22(4):1674–1700.

Wood, S. (2006). *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis.

Yan Yu, D. R. (2002). Penalized spline estimation for partially linear single-index models. *Journal of the American Statistical Association*, 97(460):1042–1054.