Catarina Félix de Oliveira

Metalearning for multiple-domain Transfer Learning

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2019

Catarina Félix de Oliveira

Metalearning for multiple-domain Transfer Learning

Programa Doutoral em Ciência de Computadores

Orientador: Prof. Doutor Carlos Manuel Milheiro de Oliveira Pinto Soares
Prof. Doutor Alípio Mário Guedes Jorge

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
2019

Agradecimentos

Este trabalho foi parcialmente financiado por fundos nacionais através da FCT – Fundação para a Ciência e a Tecnologia, I.P., no âmbito do projeto com referência: UID/EEA/50014/2019.

Gostaria de agradecer aos meus orientadores, Carlos Soares e Alípio Jorge, por acreditarem e confiarem no meu trabalho. Pelo apoio, incentivo e paciência.

Agradeço ao INESC TEC a oportunidade de participação em diversos projetos. Ao Hugo Ferreira, por toda a disponibilidade. Aos meus colegas, por todos os momentos partilhados, tanto os cafés como as trocas de ideias. À Marta e à Grasiela por estarem sempre disponíveis profissional e pessoalmente. Ao professor Álvaro Figueira, por me ter possibilitado o primeiro contacto com a investigação científica.

Agradeço à minha família que sempre me apoiou e incentivou ao longo deste percurso. Agradeço principalmente ao Pedro por compreender, tão pequenino, a minha falta de disponibilidade, e ocasional mau feitio. À minha mãe e ao Jorge por, para além de também lidarem com o ocasional mau feitio, me terem apoiado, incentivado e aconselhado.

Por fim (são família também) e o que vocês aturaram não se deseja a ninguém. Dario (eu não escrevi "Dário"), obrigada! Bruno, muito obrigada, por tudo! Sempre acreditaste mais em mim do que eu.

Sem todos vocês, nada disto teria sido possível. Obrigada!

Catarina Félix de Oliveira

Resumo

Os processos de parametrização e inicialização de redes neuronais requerem conhecimento específico dos utilizadores e consistem, normalmente, em testar diversas possibilidades. Assim sendo, podem exigir elevado poder computacional e tempo de computação, bem como tempo dispendido pelo utilizador.

Propomos a utilização de *metalearning* e *transfer learning* para reduzir o tempo e o esforço necessários a estes processos. *Metalearning* e *transfer learning* são áreas de *machine learning* que pretendem reduzir tanto os recursos computacionais como o esforço necessário, por parte do utilizador, no processo de aprendizagem. Para isso, ambas tiram partido de conhecimento obtido em experiências anteriores. Por um lado, o *metalearning* utiliza informação sobre tarefas anteriores com vista à seleção de um modelo preditivo para uma nova tarefa. Por outro lado, o *transfer learning* consiste na transferência de conhecimento a partir de uma tarefa anterior com o objetivo de ajudar no processo de aprendizagem de uma nova tarefa. A avaliação empírica da nossa abordagem é feita utilizando datasets de regressão.

No nosso trabalho, numa primeira fase, o *metalearning* é utilizado para selecionar uma parametrização capaz de alto desempenho. Para isso, propomos um conjunto de *landmarkers* específicos para redes neuronais que, juntamente com *metafeatures* tradicionais, podem ser utilizados para construir um meta-modelo de seleção de parâmetros para redes neuronais. Os resultados obtidos sugerem que o *metalearning* é uma boa abordagem ao problema, dado que o nosso meta-modelo é capaz de selecionar parametrizações que tendem a obter melhores resultados do que utilizar os parâmetros que mais frequentemente dão origem a bons resultados.

De seguida, o *transfer learning* é utilizado para inicializar as redes neuronais. Em vez de valores aleatórios, inicializamos as redes com pesos obtidos de redes treinadas anteriormente. Para isso, propomos métodos simples de mapeamento de variáveis, para que a transferência de pesos seja efetuada entre as variáveis mais apropriadas.

Os resultados obtidos sugerem que o *transfer learning* pode ser utlizado para inicializar redes neuronais de forma a acelerar o processo de treino, desde que haja uma boa rede neuronal de origem a partir da qual se selecionem os pesos.

Por fim, o *metalearning* é utilizado para selecionar a rede a partir da qual os pesos serão transferidos. Para isso, propomos um conjunto de *landmarkers* específicos para transferência que, juntamente com *metafeatures* tradicionais, podem ser utilizados para construir um meta-modelo para seleção da origem dos pesos. Os resultados obtidos sugerem que o *metalearning* é uma boa abordagem ao problema, dado que o processo de treino das redes neuronais resultantes é mais rápido, sem prejudicar o desempenho.

Os nossos meta-modelos facilitam a configuração (incluindo parametrização e inicialização) de redes neuronais, reduzindo o tempo e poder computacional necessários tanto à configuração como ao treino das redes que, mesmo assim, atingem bom desempenho. A metodologia encontra-se disponível através da aplicação `R` NN configurer e a biblioteca `R nnetConf`. Ambos estão disponíveis online e permitem a configuração automática de redes neuronais, incluindo os processos de parametrização e de inicialização.

Abstract

In neural networks, parameterisation and initialisation processes require user expertise and usually consist in experimenting with several different settings. Therefore, the processes may need high computational resources and computational time, as well as time spent by the user.

We propose the use of metalearning and transfer learning to reduce the time and effort needed on these processes. Metalearning and transfer learning are subfields of machine learning that aim at reducing both computational effort and user involvement on the learning process. For that, both take advantage of knowledge obtained on previous experiments. On the one hand, metalearning uses information about previous tasks to help in the process of selecting a predictive model for a new task. On the other hand, transfer learning consists in transferring knowledge from a previous learning task to help in the learning process of a new task. We empirically evaluate the proposed approach on benchmark regression datasets.

In our work, we start by using metalearning to select a high-performance parameterisation. For that, we propose neural network specific landmarkers that, together with traditional metafeatures, can be used to build a metamodel for parameter selection in neural networks. Results suggest that metalearning is a good approach to this problem, since our metamodel is able to select parameterisations that usually yield higher performance than the one obtained by considering the parameter values that more frequently lead to high performance.

Then, transfer learning is used to initialise the neural networks. Instead of random values, we initialise them with weights obtained from previously trained networks. For that, we propose simple feature mapping methods, so that the transfer of weights is performed between the most appropriate features. Results suggest that transfer learning can be used to initialise the networks in order to have faster training processes, provided that there is a source network from where the weights can be selected.

Finally, metalearning is used to select the source network from which the weights will be transferred. For that, we propose transfer specific landmarkers that, together with traditional metafeatures, can be used to build a metamodel for source network selection. Results suggest that metalearning is a good approach for the problem, since the resulting neural networks' training process is faster, without harming their predictive performance.

Our metamodels make the configuration (including parameterisation and initialisation) of neural networks easier, while reducing the time and computational power required for the configuration and training of neural networks, still reaching high predictive performance. This methodology is available as the the `R` Shiny application NN configurer, and the `R` library `nnetConf`. Both are available online and allow automatic configuration of neural networks, including the parameterisation and initialisation processes.

Keywords

Metalearning, Transfer Learning, Neural Networks

Contents

List of Tables

List of Figures

Acronyms

ACC	Prediction accuracy
aRRMSE	Average relative root mean squared error
BL	Baseline model
CFS	Correlation-based feature selection
DL	Deep learning
DM	Data mining
DT	Top down induction of decision trees
ES	Column title on the datasets table, representing the datasets used on the empirical study of the performance of neural networks
KL	Kullback-Leibler divergence
LDA	Linear discriminant analysis
LM	Linear model
ML	Machine learning
MOR	Multi-output regression
MSE	Mean squared error
MtL	Metalearning
MTRS	Multi-target regressor stacking
MZOE	Mean zero-one error
NN	Neural network
OR	Ordinal regression
PS	Column title on the datasets table, representing the datasets used on the metalearning for parameter selection in neural networks
RC	Regressor chains
RF	Random forest
RFE	Recursive feature elimination
ROC	Receiver operating characteristic
RRMSE	Relative root mean squared error

SST	Column title on the datasets table, representing the datasets used on the metalearning for source selection in heterogeneous transfer learning for neural networks
ST	Single-target
SVM	Support vector machine
TL	Transfer learning
WT	Column title on the datasets table, representing the datasets used on the weights transfer in heterogeneous domain neural networks

Transfer Learning Related Acronyms

ARC-t	Asymmetric regularised cross-domain transformation
CL-SCL	Cross-language text classification using structural correspondence learning
CoCC	Co-clustering based classification
CP-MDA	Conditional probability based multi-source domain adaptation
DAMA	Domain adaptation using manifold alignment
DAS	Domain adaptation of sentiment classifiers
DSM	Domain selection machine
DTM	Deep trasnfer via markov logic
DTMKL	Domain transfer multiple kernel learner
FAM	Feature augmentation method
GFK	Geodesic flow kernel
HeMap	Heterogeneous spectral mapping
HFA	Heterogeneous feature adaptation
HFP	Heterogeneous feature prediction
HHTL	Hybrid heterogeneous transfer learning
HTLIC	Heterogeneous transfer learning for text classifiers
JDA	Joint distribution adaptation
KLIEP	Kullback-leibler importance estimation procedure
KMM	Kernel mean matching
MMKT	Multi model knowledge transfer
MOMAP	Multiple outlook mapping
PDM	Predictive distribution matching
RAP	Relational adaptive bootstrapping
ROD	Rank of domain
SCL	Structural correspondence learning

SFA	Spectral feature alignment
SHFA	Semi-supervised HFA
SHFR	Sparse heterogeneous feature representation
STC	Self-taught clustering
TCA	Transfer component analysis
TDA	Transfer discriminative analysis
TPLSA	Topic-bridged probabilistic semantic analysis
TTI	Translator of text to images

Notation

δ	Error signal at a specific hidden unit on a neural network
ϵ	Error function minimised with neural network backpropagation
η	Neural network learning rate
λ	Neural network weight decay
μ	Neural network momentum
A	Number of algorithms
a	A specific algorithm
subscript a	Index of a specific algorithm
b	Value for p_d: use $\mathcal{N}\left(0, \sqrt{1/x}\right)$
b', b''	Bias units
c	Class
$\mathcal{D}$	Dataset's domain
D	Number of datasets
d	A specific dataset
subscript d	Index of a specific dataset
E	Number of instances of a dataset
superscript e	Index of a specific instance of a dataset
f	Predictive function fitted by machine learning tasks
subscript F	Forest-based algorithms
g	Activation function applied to the inputs of the hidden layer of the neural network
$\tilde{g}$	Activation function applied to the inputs of the output layer of the neural network
h	Unit of the neural network's hidden layer
I	Number of independent variables in a (target) dataset, number of nodes on the input layer of a (target) neural network
subscript i	Index of a specific independent variable of a (target) dataset, index of the corresponding input unit on the (target) neural network

IT	Number of iterations used to limit the neural network for obtaining the landmarkers
J	Number of dependent variables in a dataset
subscript j	Index of a specific dependent variable of a dataset
K	Number of classes
$KCor$ mapping	Feature mapping for transfer based on the Kendall correlation
KL mapping	Feature mapping for transfer based on the Kulback-Leibler divergence
L	Number of independent variables in a (source) dataset, number of nodes on the input layer of a (source) neural network
subscript l	Index of a specific independent variable of a (source) dataset, index of the corresponding input unit on the (source) neural network
subscript L	Linear algorithms
$\mathcal{M}$	Performance metric considered for neural networks:
$MSE0$	Predictive performance of a neural network before training
$duration$	Time needed for the neural network to converge
MSE	Predictive performance of a neural network
MSE_MEAN	Predictive performance of a neural network obtained with the baseline model that predicts the average of the trainset for each instance on the testset
M	Mapping function for transfer
m	Machine learning model
m'	Metalearning model
MF_{PS}	Metafeatures designed for parameter selection
MF_T	Metafeatures designed for source selection for transfer
$MM1$	Metric that evaluates if the metamodel correctly identifies negative and positive transfers
$MM2$	Metric that evaluates if the metamodel correctly identifies the best possible transfer
$MM3$	Metric that evaluates if the metamodel correctly identifies a good transfer
N	Number of units on the neural network's hidden layer
subscript n	Index of the neural network's hidden unit
subscript new	Identifier of a new dataset on the metalearning process for algorithm recommendation
o	Order of a class

$P(X)$	Probability distribution
p	Parameterisation:
$p^{BEST.MAJ}$	Parameterisation composed by the best performing parameter value for the majority of the datasets
p^{grid}	Best parameterisation found by grid search for each dataset
p^{meta}	Parameterisation suggested by the metalearning process
$p^{RANK.TOP5.*}$	Parameterisation with highest ranking
$p^{ROBUST.AVG}$	Parameterisation expected to generate lower average mse
$p^{ROBUST.VAR}$	Parameterisation expected to generate lower variance in mse
p^{WORST}	Parameterisation composed by the worst performing parameter value for the majority of the datasets
p_a	Neural network's parameter abstol: the value to use dor the abstol fitting criterion
p_D	Neural network's parameter distribution: the distribution of weights used to initialise the neural network
p_d	Neural network's parameter decay: the value to use for the weight decay
p_n	Neural network's parameter size: the number of units on the hidden layer
P_R	Performances obtained in randomly initialised neural networks
P_T	Performances obtained neural networks initialised by transferred weights
$PCor$ mapping	Feature mapping for transfer based on the Pearson correlation
R mapping	Random feature mapping for transfer
superscript R	Neural network initialised with a random set of weights
$SCor$ mapping	Feature mapping for transfer based on the Spearman correlation
SS	Size of the subsample used for obtaining the landmarkers
subscript S	Source dataset
subscript T	Target dataset
superscript T	Neural network initialised with transferred weights
$\mathcal{T}$	Dataset's task
t	Neural network iteration
u	Value for p_d: use $\mathcal{U}[0, 1]$
W	Number of weights of a fully connected neural network
w	Weights of the neural network connections between input and hidden layers

$\tilde{w}$	Weights of the neural network connections between hidden and output layers
$\mathcal{X}$	Dataset's feature space
x	Independent variable of a dataset
$\mathcal{Y}$	Label space
y	Dependent variable of a dataset
$\hat{y}$	Prediction for the dependent variable of a dataset

Chapter 1

Introduction

Applying machine learning (ML) to a task typically implies collecting training data, obtaining a model (training), and applying the model to new data (testing). This is done even when the new task is related to one or more previously solved tasks.

In some situations, the new task may be related to one or more previously approached tasks. For example, let us assume we need to analyse data on 1000 industrial machines. We have one dataset for each machine and we plan to learn each model with the best performing algorithm from the set of algorithms A1, A2 and A3. First, let us consider a simplified scenario: after processing 800 datasets we discovered that algorithm A1 clearly obtained the best results for all the datasets processed. Given the experience obtained on the previous learning processes, it would be expected that algorithm A1 would also obtain the best results for the remaining 200 datasets. Thus, we could ignore algorithms A2 and A3 and only apply A1 to the rest of the datasets, significantly reducing the time needed for the ML process.

Now, let us consider a more complex scenario: after processing 800 datasets, we discovered that algorithm A1 obtains the best results for datasets with less than 500 examples, while for larger datasets A2 is the best performing algorithm. In this case, there is a dataset characteristic (number of examples) that helps determining the algorithm expected to obtain better results. Now, for each of the remaining 200 datasets, we count the number of examples and, with this, determine the expected best algorithm. Therefore, instead of applying all the algorithms to each dataset, we only apply the selected one. This reduces the computational effort needed for the ML process.

Metalearning (MtL) can be used in situations as the one referred above (Brazdil et al., 2008). Typically, MtL uses information (called *metafeatures*) about previous tasks to help in the process of selecting a predictive model for a new task, reducing both computational effort and user involvement. Further information about this topic is presented in Section 2.3.

Another example of the ML process applied to related tasks is the following: let us imagine we are dealing with data obtained from a set of sensors installed in an industrial machine M1. First, we collect the training data (outputs from the sensors), obtain a model, and finally apply it to new data gathered from machine M1. Later, we need to analyse data obtained from a set of the same sensors installed in a different machine M2, similar to machine M1. We will restart the ML process from the beginning: collect the sensor data, obtain the model and apply it to new data from machine M2. However, because of the similarities between the machines, and since the sensors installed in machines M1 and M2 are the same, the data obtained from both machines will probably be related. Reusing the model obtained for machine M1 and applying it to the data gathered in machine M2 would reduce the time needed for the ML process.

Transferring knowledge from a previous learning task (source) to help in a new one (target) is called Transfer Learning (TL). By taking advantage of the knowledge obtained in previous tasks it enables the use of different tasks or data distributions for the training and testing parts of the process to improve the target task's performance (Pan and Yang, 2010). The use of TL implies addressing three issues. First, we need to select the knowledge to be transferred (**what to transfer**). Then, we need to define the methods used for the transfer process (**how to transfer**). Finally, we need to ensure that the transfer will only be performed when it in fact improves the target task's performance (**when to transfer**). More information on this subject can be found in Section 2.4.

Both MtL and TL use information about a previously learned domain to efficiently and effectively learn in a new, unseen domain. As more and more ML tasks are dealt with, we can store the information on the processes used to address them and exploit it when approaching new learning tasks. On the one hand, MtL uses past experience selectively, given the characteristics of the new task (Brazdil et al., 2008). On the other hand, TL values previous experience for learning new concepts better (Pan and Yang, 2010). However, not all the previous tasks are expected to contain useful knowledge. This suggests that TL and MtL may be used together. For example, MtL could be used to decide whether the transfer should or should not be performed.

Neural Networks (NNs) are a classical ML model inspired in the principles of the work of our own brains. They have been recently growing in popularity due to the wide range of possible configurations and manageable convergence properties. According to Bishop (1994), the human brain can roughly be described as a set of neurons and the connections among them. When information is being processed, it passes from a neuron to the next one through the connections. Depending on the information to process, the connections have different strengths. In the same way, an artificial NN is composed by several connected processing units. Also, the connections have associated weights that need to be defined in the beginning of the training process.

NNs are highly parameterisable models that can achieve high performance, provided that the parameter values are well chosen (Ripley, 2007). The parameters can control, for example, the NN architecture or the training process stopping criteria, among others. The process of choosing the parameter values (parameterisation) for NNs requires user expertise (Venables and Ripley, 2002; Ripley, 2007). Besides, usually the data scientist starts by experimenting with several different parameterisations. This requires high computational resources, time, and user effort.

Furthermore, when configuring a NN, the data scientist also needs to define a set of values to be used as the initial set of weights. The process of defining the initial set of weights (initialisation) of a NN influences its behaviour. Thus, the initial weights need to be carefully chosen. Besides user expertise, due to the need of performing several tests with different sets of initial weights, this process also requires high computational resources, time, and user effort.

1.1 Motivation and Objectives

As referred, NN parameterisation and initialisation are difficult and time-consuming tasks that require user expertise. We aim at reducing the time and effort the data scientists need to spend on these processes, by using MtL together with TL, as depicted in Figure 1.1.

By taking advantage of experience acquired previously (on previously trained NNs), we intend to use MtL to select a high-performance parameterisation. Furthermore, we aim at using TL to initialise the NNs. The TL process consists in transferring weights (*what to transfer?*), between the most adequate connections of the source and target datasets (*how to transfer?*). TL will take the new data into account and select

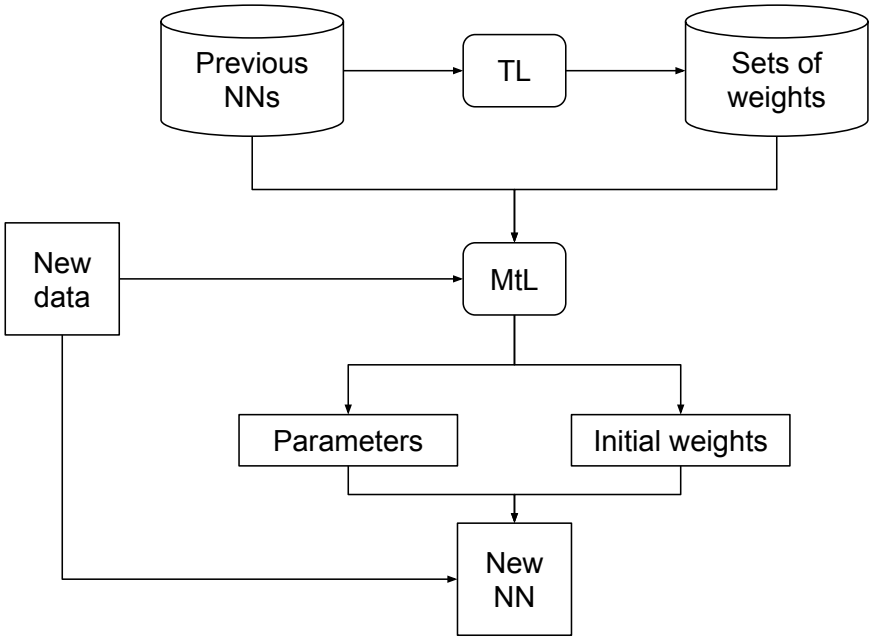


Figure 1.1: Use of MtL together with TL.

a set of weights from each source network. However, not every transfer will lead to a high-performance NN. Thus, we intend to use MtL to predict if transferring from a certain source NN will lead the target NN to perform better than when randomly initialised (*when to transfer?*), preventing *negative transfer* (Rosenstein et al., 2005).

One of the main challenges of MtL is the design of suitable data metafeatures (Brazdil et al., 2008) (Subsection 2.3.2). Useful metafeatures contain information about the data that describe the behaviour of the learning algorithm(s). We aim at finding metafeatures that can characterise the NNs (for the parameterisation task) and the transfer process (for the source selection task).

Given the above, our work aims at answering the following questions:

RQ1 How do different parameter values impact the performance of NNs?

RQ2 Can MtL be used to support the parameterisation of NNs?

RQ3 What is the impact of TL (weights transfer) on NNs?

RQ4 Can MtL be used to support TL in NNs?

Our ultimate goal is to be able to provide the data scientists with a method for full automatic configuration (both parameterisation and initialisation) of NNs. Given a new dataset, we extract its characteristics and use them to suggest the parameterisation as well as an initial set of weights to be used. This enables the development of a competitive NN for the dataset at hand, with almost no need of user intervention in the process.

1.2 Contributions

This project encompasses four phases, depicted in Figure 1.2, and described next.

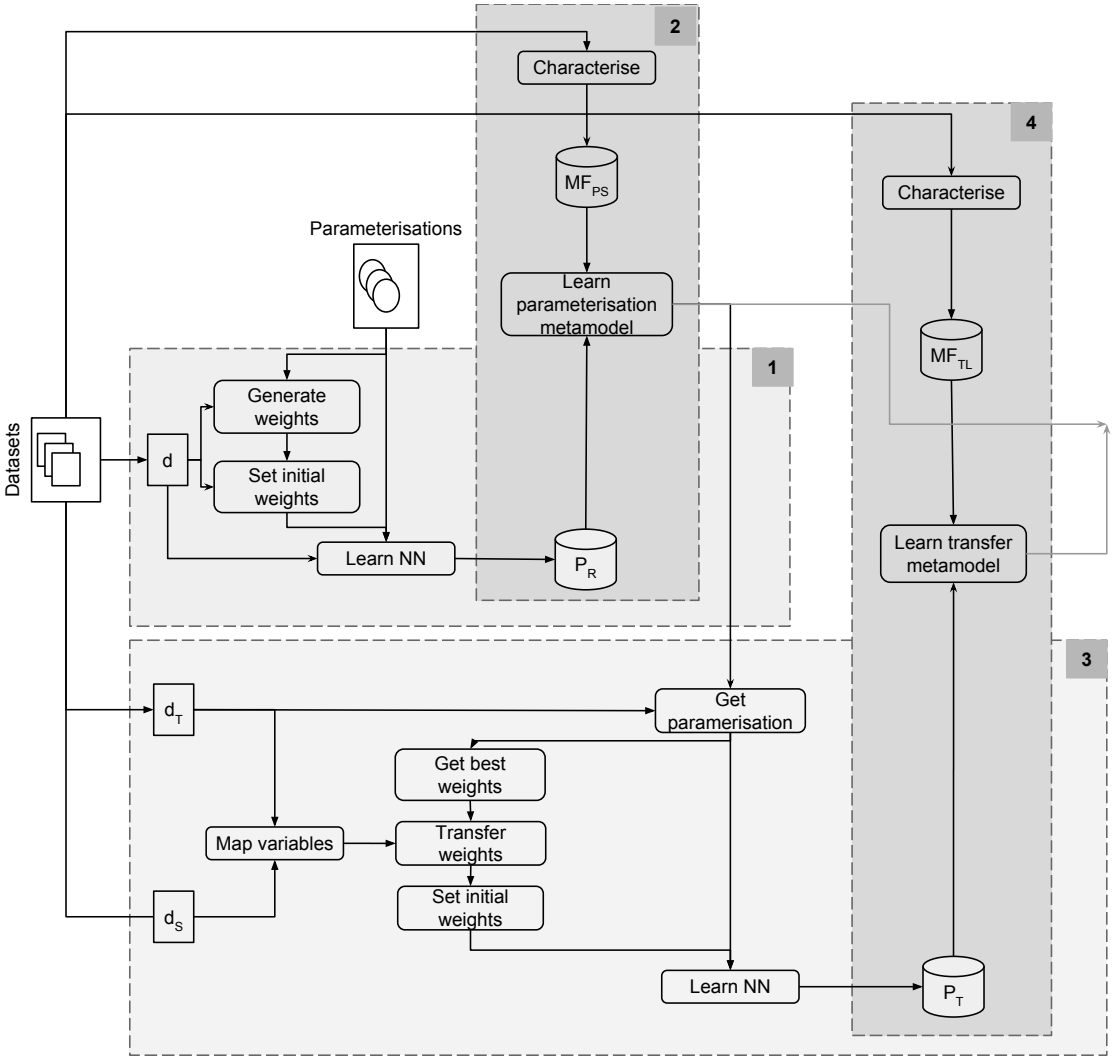


Figure 1.2: MtL for multiple domain TL experimental process.

1. **Empirical study of the performance of NNs** (Chapter 3)
   To answer Research Question **RQ1**, we studied the impact of different parameterisations in NN performance. For this, as depicted in Figure 1.2 [*1*], we analysed several NNs following a grid search of 120 possible parameter value combinations. Results suggest that, as expected, there is no parameter combination with generally high performance for all the datasets. Instead, the best performing parameterisation depends on the dataset.

   Contributions:

   - Empirical results on the impact of different parameter values on NNs;
   - A group of parameterisations that can be used for NNs to generally achieve average high performance. By using these, the data scientist can achieve relatively fast and accurate NNs that do not require user experience or time spent on parameter tuning;
   - A group of partial parameterisations that can be used to reduce the grid for the parameter selection. With this, the user experience and time required for parameter tuning significantly decrease.

2. **MtL for Parameter Selection in NNs** (Chapter 4)
   As expected, the results obtained in the previous phase suggest that NN parameterisation depends on the data itself. Additionally, as NNs may require high computational resources for training, running several networks only to choose the best performing configuration may be computationally expensive. Bearing this in mind, and to address Research Question **RQ2**, we studied the use of MtL for the parameter selection problem in NNs (Figure 1.2 [*2*]). We characterise the NNs obtaining their metafeatures. Then, we build the metamodel by using the metafeatures together with the parameterisations' performance obtained in the previous phase. Results suggest that MtL can be used to select accurate parameterisations for NNs.

   Contributions:

   - Two different sets of NN landmarkers that, together with the remaining metafeatures, capture the NNs characteristics, allowing the MtL process to select high performance parameterisations;

- Empirical results on the use of MtL for parameter selection in NNs comparing classification and regression approaches with different sets of metafeatures.

3. **Weights Transfer in Heterogeneous Domain NNs** (Chapter 5)
The results presented in Chapter 3 suggest that the starting point of a NN (i.e., the initial weights) significantly influences its performance. As depicted in Figure 1.2 [*3*], we study the impact of TL, namely the transfer of weights, in NNs (Research Question **RQ3**). We test several TL settings for each dataset considered. Results suggest that TL can be used to successfully initialise NNs, provided that the source network (the one from which the weights will be transferred) is well chosen.

Contributions:

- A mapping process for features in different domain datasets;
- A weights transfer method for NNs;
- Empirical results on the use of TL (through mapped weights transfer) in NNs comparing the performance of NNs initialised with transferred weights with the performance of randomly initialised NNs.

4. **MtL to support TL** (Chapter 6)
The results of phase 3 suggest that higher performance networks can be obtained by transferring weights from a previously learned NN, provided that the source of the weights is well chosen. In order to reach our main objective, reflected in Research Question **RQ4**, we study the use of MtL to choose what NN to use as source of the weights to be transferred for a specific target NN (Figure 1.2 [*4*]). We characterise the transfers and use this information, together with the results obtained on the previous phase, to build the metamodel. With this, we wish to provide the data scientists with a method to accurately initialise a NN, by providing them with a set of initial weights. Results suggest that the use of MtL for this task allows faster training NNs, without harming their predictive performance.

Contributions:

- A set of metafeatures capable of describing the differences between the source and target datasets;

- A set of landmarkers for TL in NNs that, together with the remaining metafeatures, characterise the transfers, thus allowing the MtL process to accurately predict the impact of a specific transfer on the NN's performance;
- Empirical results on the use of MtL for selecting the network to use as source of weights to transfer to a target NN comparing the use of three regression approaches with different sets of metafeatures for creating the metamodel.

5. **Available Resources**
   We implemented two resources that can be used by the data scientist to fully configure neural networks for the `R`'s `nnet` implementation. These resources were built according to this research, and are described in Section 6.4:

   - **NN configurer**: an `R` Shiny application, publicly available at (https://catarinafelix.shinyapps.io/nn_shiny/) and also available for download at (https://gitlab.com/catarinafelix/nn_shiny);
   - `nnetConf`: an `R` library, available for download at (https://gitlab.com/catarinafelix/nnetconf).

1.2.1 Summary of Contributions

Given that NN configuration (parameterisation and initialisation) is difficult and time consuming (Chapter 3), and transfer improvement in NNs depend on correctly selecting the source network (Chapter 5), we propose an approach that configures competitive NN for new datasets. Our approach is able to use metalearning to make more efficient the selection of parameter settings (Chapter 4) and network to use as source of weights to initialise the NN (Chapter 6).

1.3 Publications

During the course of the research, we have published the three following papers:

1. **Metalearning for Multiple-Domain Transfer Learning** (Félix et al., 2015), a preliminary study concerning the improvements TL can achieve in NNs' performance (phase 3). In it we describe two experiments: 1) compare random transfer

(made between randomly chosen variables) with direct transfer (performed between correspondent variables, in related datasets); 2) study the behaviour of the weights transfer between similar variables, compared to the random transfer. Results suggest that transfer between similar datasets is advantageous, and the advantages increase when the transfer is performed between similar variables. This way, finding similarities between source and target datasets and between their variables may help in the TL process.

2. **Can metalearning be applied to transfer on heterogeneous datasets?** (Félix et al., 2016), another study on the improvements achieved by TL in NNs (phase 3). In this case, only the mapped transfers are studied, however with a more extensive experimental setup. Results suggest that, when the source network is well chosen, transferring weights can accelerate the learning of the target network. Furthermore, it indicates that MtL can be used to support TL by selecting the network to act as source of the weights for the transfer.

3. **Using MtL for Parameter Tuning in NNs** (Félix et al., 2017), a preliminary study on the MtL's ability to select a high-performance parameterisation for NNs (phase 2). We describe a classification approach to use MtL to select a high-performance parameterisation for NNs. Results suggest that MtL can be used for parameterising NNs to obtain NNs that perform almost as well as the ones selected with grid search.

Our follow up publication plan includes the four following journal papers:

1. **Empirical study on neural networks performance with different parameter settings**, in preparation;

2. **Metalearning approach for automatic parameter tuning in neural networks**, in preparation;

3. **Transfer of weights to improve neural networks performance**, to be prepared;

4. **Metalearning to prevent negative transfer in neural networks**, to be prepared.

1.4 Outline of the document

The rest of the document is organised as follows. First, in Chapter 2 we present some concepts related to our work. After that, we present the four parts that encompass our research (referred to in Section 1.2): Empirical study of the performance of Neural networks (Chapter 3), Metalearning for Parameter Selection in Neural Networks (Chapter 4), Weights Transfer in Heterogeneous Domain Neural Networks (Chapter 5), and Metalearning to support Transfer learning (Chapter 6). Finally, in Chapter 7, we present the conclusions and future work of our research.

Chapter 2

State of the Art

Data mining (DM) is the process of discovering patterns in data (Witten et al., 2016). There are some restrictions to this: 1) the process must be automatic or, at least, semi-automatic; 2) the patterns found must be useful (i.e., must lead to some advantage), and; 3) large quantities of data must be present.

Machine learning (ML) techniques aim at analysing the data to find meaningful patterns. The data used in ML is commonly represented in tabular form (*datasets*). ML problems can be divided into two categories: *unsupervised* and *supervised* learning problems. In unsupervised learning problems there is no dependent variable, while in supervised learning problems, at least one dependent variable is present on the dataset. There are specific tasks to unsupervised learning problems (e.g.: clustering, association) each with its own performance metrics. Our research focuses on supervised learning problems, where the value of the dependent variable is present on the data and can be considered for building the prediction model, and it can also be used for evaluating the models' performance. Common types of ML tasks are classification and regression, which will be described later in Section 2.1.

A typical supervised ML modelling process consists of three phases: 1) retrieving the data; 2) building a model – or several models – fitting the data (training phase), and; 3) applying each model to the data and evaluating its performance (testing phase). Before deploying a model on new data, it is developed using the following approach (or a similar one): the dataset is split into *training* and *testing* sets, to be used in the second and third phases, respectively. This happens because we want to avoid *overfitting*, i.e., that the model fits too much to the training data that, besides patterns, it also captures noise.

There are many models that can be used for each ML task. A ML model that has been receiving increased attention is neural networks (NNs). A NN is composed by a set of processing units, organised in layers. The first layer is where the inputs to the NN are provided. These values are processed on the internal layers and the result is computed on the output layer. NNs are inspired on the human brain's behaviour and will be detailed in Section 2.2.

Furthermore, each algorithm may have a set of parameters that must be defined in order to achieve maximum performance for a specific dataset. Choosing a fast high-performance algorithm (and parameter configuration) is not an easy task. *Metalearning* (MtL) is a sub-field of ML that aims at helping the data scientist with this issue. By using knowledge extracted using ML approaches from past experiments (data, algorithms, algorithm parameterisation and algorithm performance on the data), it aims at choosing a high-performance algorithm for the data at hand. For this, MtL tries to predict the algorithms' performance on a new, unseen task. This subject will be more thoroughly explained in Section 2.3.

However, there may be a limited supply of training data, or the models' training process can be computationally expensive. An approach to overcome these issues is *transfer learning* (TL). TL aims at improving a model's performance by transferring knowledge obtained on previously trained models. TL is another sub-field of machine learning and will be described in Section 2.4.

In the following sections we explain and exemplify some machine learning tasks, neural networks, metalearning and transfer learning. For simplification, in the examples, we are going to consider a generic dataset containing E instances of I independent variables $x_1, \ldots, x_I$ and one dependent variable y. Thus, x_i^e represents the value of the ith independent variable in the eth instance of the dataset, $\hat{y}$ represents the predictions for the dependent variable and $\hat{y^e}$ represents the eth value of the predictions.

2.1 Machine learning tasks

Here we describe the machine learning tasks used for metalearning in this work: classification and regression (including ordinal regression and multi-output regression). We will consider a single target problem for all the techniques, except for multi-output regression (Subsection 2.1.2.2), where we consider that the dataset has J dependent variables $y_1, \ldots, y_J$.

2.1.1 Classification

Classification consists in creating a model m that tries to fit the function f such that

$$\hat{y} = f(x_1, \ldots, x_I) \tag{2.1}$$

that best approximates the true output of y. In classification problems, each instance of the dependent variable belongs to exactly one of a finite set of candidate classes: $\{c_1, \ldots, c_K\}$ where K is the number of classes y can take.

There are many algorithms for classification (Friedman et al., 2001; Han et al., 2011), including those used in this work: linear discriminant analysis (LDA), top down induction of decision trees (which, for simplification, will be referred to as *decision trees* – DT) and random forest (RF). The implementations used were `lda` (Venables and Ripley, 2002), `rpart` (Therneau et al., 2015), and `randomForest` (Liaw et al., 2002), respectively.

The evaluation of the predictive performance of classification methods can be measured with several metrics. For the purpose of this work, since this is a generic study, we use the prediction accuracy (ACC), because it is a commonly used performance metric for classification.

$$ACC = \frac{|e \in \{1, \ldots, E\} : \hat{y}^e = y^e|}{E} \tag{2.2}$$

2.1.2 Regression

Classification and regression are very similar, except for the target variable. Regression consists in creating a model m that tries to fit the function f such that

$$\hat{y} = f(x_1, \ldots, x_I) \tag{2.3}$$

that best approximates the true output of y. In regression problems, the dependent variable is continuous ($y^e \in \mathbb{R}$).

As in classification, many algorithms are available for regression (Friedman et al., 2001; Han et al., 2011). In these experiments we used standard machine learning algorithms, namely: linear model (LM), top down induction of decision trees (DT) and random forest (RF). The implementations used were `lm` (Team et al., 2013), `rpart` (Therneau et al., 2015), and `randomForest` (Liaw et al., 2002), respectively.

Likewise, many metrics can be used for evaluation. In our experiments we used two different performance metrics. A common evaluation metric for regression tasks is the

Mean Squared Error (MSE):

$$MSE = \frac{1}{E}\sum_{e=1}^{E}(y^e - \hat{y^e})^2, \tag{2.4}$$

One disadvantage of using MSE in empirical studies involving many datasets is that the interpretation of the values depends on the scale of the target variable. Another evaluation metric is the Relative Root Mean Squared Error ($RRMSE$):

$$RRMSE = \sqrt{\frac{\sum_{e=1}^{E}(y^e - \hat{y}^e)^2}{\sum_{e=1}^{E}(y^e - \bar{y})^2}} \tag{2.5}$$

In spite of this, MSE is still the most popular evaluation measure in regression.

2.1.2.1 Ordinal Regression

Ordinal Regression (OR) is a machine learning task where the labels are ordered categorical values (e.g., *low*, *average*, *high*) (Gutiérrez et al., 2016). The task consists in creating a model m that tries to fit the function f such that

$$\hat{y} = f(x_1, \ldots, x_I) \tag{2.6}$$

that best approximates the true output of y. In ordinal regression problems, each instance of the dependent variable belongs to exactly one among a finite set of candidate classes: $\{c_1, \ldots, c_K\}$ where y can take K classes with order $o(c_1) < o(c_2) < \ldots < o(c_K)$.

Although standard classification methods can be used, there are some specific algorithms for ordinal regression, as is the case of the ones used here: top down induction of decision trees (DT), implemented in `ctree` (Hothorn et al., 2006) and random forest (RF), implemented in `cforest` (Hothorn et al., 2005; Strobl et al., 2007, 2008).

Several measures can be used to evaluate ordinal regression approaches (Baccianella et al., 2009). We use Mean Zero-One Error ($MZOE$), because it is not affected by the results' scales, and is related to the accuracy:

$$MZOE = \frac{|e \in \{1, \ldots E\} : \hat{y}^e \neq y^e|}{E} \tag{2.7}$$

2.1.2.2 Multi-output Regression

Multi-output Regression (MOR) is a ML task that consists in simultaneously predicting several target regression variables in the same dataset. The main assumption

behind these approaches is that the values of the target variables are not independent.

The baseline approach for this problem is single target (ST) prediction: independently generating J different models for the J different target variables and predicting each of them separately:

$$m_j : \hat{y}_j = f(x_1, \ldots, x_I) \tag{2.8}$$

The dependence between target variables is ignored by the single target approaches. However, there are specific approaches for multi-output regression (Spyromitros-Xioufis et al., 2012):

- **Multi-target regressor stacking (MTRS):** this technique encompasses two phases. First, ST method is used to generate J models, and the J targets of the testset are predicted:

  $$m'_j : \hat{y}'_j = f(x_1, \ldots, x_I) \tag{2.9}$$

  In the second phase, those intermediate predictions $(\hat{y}'_1, \ldots, \hat{y}'_J)$ are used as attributes to compute the final J models that will obtain the final predictions of the target variables:

  $$m_j : \hat{y}_j = f(x_1, \ldots, x_I, \hat{y}'_1, \ldots, \hat{y}'_J) \tag{2.10}$$

- **Regressor Chains (RC):** first, a random chain (permutation) of target variables is selected. Then, the original attributes are used to predict the first target on the chain ($m_1 : \hat{y}_1 = f(x_1, \ldots, x_I)$). The prediction is added to the attributes and the second target on the chain is predicted ($m_2 : \hat{y}_2 = f(x_1, \ldots, x_I, \hat{y}_1)$). The process is repeated until all the targets have been predicted:

  $$m_j : \hat{y}_j = f(x_1, \ldots, x_I, \hat{y}_1, \ldots, \hat{y}_{J-1}) \tag{2.11}$$

These approaches learn multiple models using traditional regression algorithms, such as the ones used in this work: linear model (LM), top down induction of decision trees (DT), and random forest (RF). The implementations considered were `lm` (Team et al., 2013), `rpart` (Therneau et al., 2015), and `randomForest` (Liaw et al., 2002), respectively.

Multiple evaluation measures exist for this task (Borchani et al., 2015). We use the average relative root mean squared error ($aRRMSE$), with $RRMSE$ as defined earlier (Equation 2.5):

$$aRRMSE = \frac{1}{J} \sum_{j=1}^{J} RRMSE_j \tag{2.12}$$

2.2 Neural Networks

A ML model that has been receiving increased attention is Neural Networks (NNs), which are inspired on the human brain's behaviour. Although the NNs' training time can be long, once the model is fully trained, it can efficiently predict the outputs of new examples.

Next, we provide an overview of NNs according to Bishop (1994), including their biological context (Subsection 2.2.1), a comparison of shallow and deep NNs (Subsection 2.2.2, where we explain our choice to use shallow NNs), the one hidden layer architecture (Subsection 2.2.3) and, finally, the NN training process (Subsection 2.2.4). For further details on NNs, also refer to Bishop et al. (1995) and Bishop (2006).

2.2.1 Biological Context

First, let us introduce the biological NN structure according to Bishop (1994). The human brain contains neurons that are composed by several *dendrites* (providing inputs to the neuron) and an *axon* (working as the neuron's output). The neurons are connected and the connections, called *synapses*, allow the communication between neurons. When neurons "fire", they send an electrical impulse that propagates through the cell body, to the *axon* and then the *synapse*. From here, the electrical impulse acts as an input for the subsequent neuron. Each *synapse* has an associated strength and the combination of all the inputs received, when compared to a certain threshold, will define if the neuron will "fire" an electrical impulse to the subsequent neuron.

The behaviour of the artificial NNs is analogous to the biological ones. We focus on the multi-layer perceptron which contains units (*neurons*) organised in layers: the input layer, at least one hidden layer, and the output layer. The units on one layer are connected to the units in the subsequent layer. The output of each unit passes through the connections (*synapses*) to the units in the next layer. This simulates the input and output through *dendrites* and *axon*, respectively. The connections between units have associated weights (*synapse* strength) that influence the impact of the information passed to the next unit.

Each layer has an associated activation function. The input values from the previous layer's units are fed to the activation function, which aggregates them into a single value that is passed onto the following layer's units. The middle layers are said to be hidden because their activation values are not directly accessible from the outside.

2.2.2 Shallow vs. Deep Networks

The number of hidden layers in NNs determines if they are shallow or deep. Shallow networks contain few hidden layers and include one-hidden-layer NNs. Deep networks, on the other hand, are composed by multiple hidden layers (Pasupa and Sunhem, 2016).

Deep learning (DL) methods have achieved high performance in many domains such as image, video and document classification (Zheng et al., 2016). DL learns data representation by using multiple processing layers, discovering the complex structure of high dimensional data with multiple levels of abstraction (Zheng et al., 2016).

For example, for image recognition, layers in different depths represent different information (Liu and Zaidi, 2016). In this case, the first layers represent the pixels. Then, the middle layers represent the edges composed by the pixels identified on the first layers. Finally, the higher layers represent the concepts composed by the edges identified on the middle layers.

DL networks take into account higher order interactions among the features, which is advantageous when facing large amounts of data (Liu and Zaidi, 2016). However, for deep learning to achieve high performance, large datasets are required (Pasupa and Sunhem, 2016; Zheng et al., 2016; Liu and Zaidi, 2016). Besides, they also need high computational power (Pasupa and Sunhem, 2016) and require long training time (Zheng et al., 2016).

Studies have shown that, for smaller datasets, shallow learning achieves better results than deep learning, yielding lower-biased models and superior convergence (Liu and Zaidi, 2016), achieving higher performance than deep artificial neural networks (Pasupa and Sunhem, 2016; Liu and Zaidi, 2016).

We wish to analyse the relationship between the data and the learning process. The more complex the learning process, the harder this task is. Besides, we use traditional datasets, in tabular form, which could be less suitable for deep learning. This way, we consider the simplest version of NNs: multi-layer perceptron with one-hidden layer (instead of deep networks).

2.2.3 One hidden layer architecture

We now provide an example of a neural network with an architecture similar to the one adopted in our work. We assume a dataset with the same structure as the one referred to in the beginning of this section: E instances of I independent variables $x_1, \ldots, x_I$ and one dependent variable y ($J = 1$). Also, let us consider a NN with a single hidden layer, as depicted in Figure 2.1.

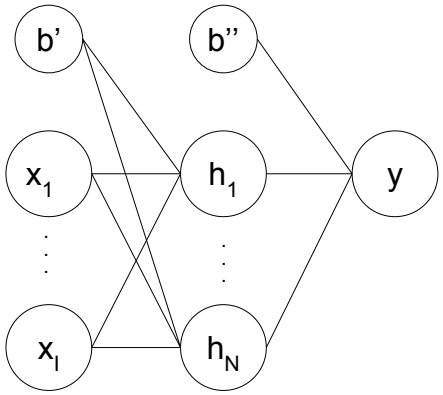


Figure 2.1: Neural network with I input units, one hidden layer with N units, and one output unit.

This NN is composed of three layers. The input layer contains I units corresponding to the independent variables of the dataset. Besides those, there is also a *bias* unit (b') whose value is set to +1. The hidden layer has N units ($h_1, \ldots, h_N$). This value is one of the NN's parameters. This layer also has a *bias* unit (b''). The output layer contains one unit which corresponds to the dependent variable.
Both the input and the hidden layers are fully connected to the following layers, respectively the hidden and output layers. This means that all the input units are connected to all the hidden units, and these are connected to the output unit. This way, the number of weights of a fully connected NN can be obtained by:

$$W = (I + 1) \times N + (N + 1) \times J \tag{2.13}$$

Each connection has a corresponding weight: w_{ni} is the weight of the connection between the ith input unit and the nth hidden unit; $\tilde{w}_{jn}$ is the weight of the connection from the nth hidden unit to the jth output unit.

2.2.4 Training process

NNs compute the predictions of each of its output units, y_j, as:

$$y_j = \tilde{g}\left(\sum_{n=0}^{N} \tilde{w}_{jn} g\left(\sum_{i=0}^{I} w_{ni} x_i\right)\right) \tag{2.14}$$

where g and $\tilde{g}$ are the activation functions applied to the inputs of the hidden and the output layers, respectively; w_{n0} and $\tilde{w}_{j0}$ are the weights of the bias units in the input and hidden layers, respectively.

The typical training process of NN models consists in iteratively adjusting the weights seeking the minimisation of a certain error function ϵ in a process called *error back-propagation*. Backpropagation evaluates the derivatives of the error function with respect to the weights, leading to the update of the weights in each iteration. The error signal at the nth unit in the hidden layer is:

$$\delta_n = \frac{\partial \epsilon}{\partial \sum_{i=0}^{I} w_{ji} x_i} \tag{2.15}$$

Typically, in each iteration t, the weights of all the connections (between the input and hidden layers, but also between the hidden and output layers) are updated according to:

$$\Delta w_{ji}^{(t)} = -\eta \frac{\partial \epsilon}{\partial w_{ji}}\bigg|_{w^{(t)}} + \mu \Delta w_{ji}^{(t-1)} \tag{2.16}$$

where η is the *learning rate* (used to determine how much the update will influence the weights) and μ is the *momentum* (used to avoid local minima), which are both parameterisable values.

2.2.4.1 Tuning parameter values

Since the training of a NN is an iterative process it is important to define when the process should stop. However, because of the non-linearity of the error functions, it is hard to define a stopping criterion. If we stop to early, the model may be too biased, if we stop to late the model may overfit.

In Ripley (2007) the authors state that it is very important to choose good starting and stopping points for iterative models like NNs. For the starting points, they suggest a random set of weights whose values should not be too large because, otherwise the units will be in a saturated state (the units always output values near to 0 or 1).

To avoid saturation, the authors consider another factor, called *weight decay* (λ) that tries to reduce the magnitude of the weights in each iteration by:

$$w_{ji} = w_{ji} - \eta \sum_{e} y_i^e \delta_j^e - 2\eta\lambda w_{ji} \tag{2.17}$$

where y_i^e is the output of the ith unit for the eth instance of the dataset, and δ_j^e is the error signal at unit j obtained with the backpropagation algorithm. Weight decay is also parameterizable and it is not easy to select the best value (Venables and Ripley, 2002).

2.3 Metalearning

As explained earlier, NNs have multiple parameters to be set, with significant impact on the model obtained and its performance. This is true for other algorithms as well. Besides, it is also necessary to choose the algorithms that are best suited for the task at hand.

Metalearning (MtL) aims at helping in the process of selecting a predictive algorithm to use on a given dataset (Brazdil et al., 2008). MtL is a sub-field of ML where algorithms are applied to (meta)data on ML experiments. Its objective is to take advantage of information obtained from previous tasks to improve the performance in new tasks. A recent survey on this subject can be found in Lemke et al. (2015), where the authors overview MtL and the most common techniques used.

This technique is mainly used for the *algorithm selection problem* (Brazdil and Giraud-Carrier, 2018), and has also been used to address the most common tasks - classification (Brazdil et al., 2003; Ali et al., 2018), regression (Gama and Brazdil, 1995), time series (Prudêncio and Ludermir, 2004) and clustering (Pimentel and de Carvalho, 2018). These approaches were then extended, for instance, to: selecting parameter settings for a single algorithm (Gomes et al., 2012); the whole data mining process (Serban et al., 2013); problems from domains other than machine learning, e.g.: different optimisation problems (Abreu et al., 2009; Smith-Miles, 2009; Pavelski et al., 2018; Gutierrez-Rodríguez et al., 2019; Chu et al., 2019); and also data streams (Gama and Kosina, 2011). Furthermore, MtL approaches have been used for automatic parameter tuning (Molina et al., 2012). Also, a preliminary study developed within this research, and included in Chapter 4, approaches the use of MtL for parameter selection in neural networks (Félix et al., 2017).

2.3.1 Algorithm Recommendation

The Algorithm Selection Problem, originally formulated by Rice (1976), consists in determining the best algorithm to use for a certain dataset. MtL can take advantage of information previously obtained on several datasets with several algorithms to approach this problem (Brazdil et al., 2008). This knowledge is used to build a

metamodel that, given a new dataset, gives the system the ability to recommend the best algorithm(s).
Figure 2.2 illustrates the MtL process for the algorithm recommendation problem.

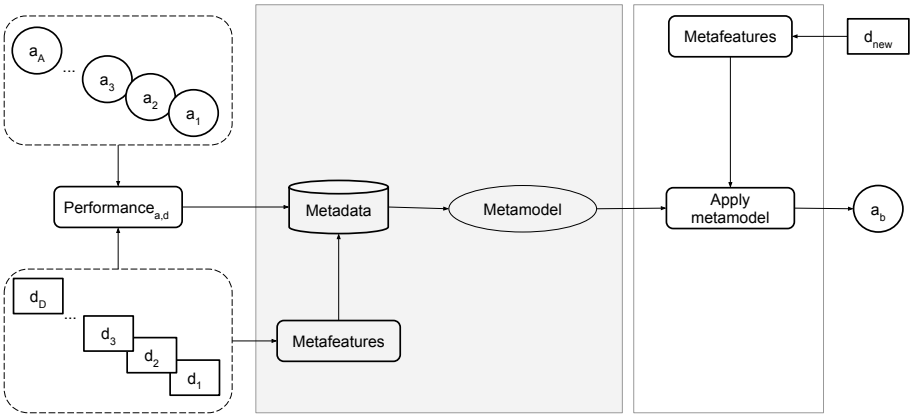


Figure 2.2: Metalearning process for the algorithm recommendation problem.

The process starts (in the left part of the figure) with D datasets $d_1, \ldots, d_D$ and A algorithms $a_1, \ldots, a_A$ (possibly with associated parameter settings). In a preliminary phase, the algorithms are applied to the datasets, and the performance obtained by each algorithm on each dataset is saved.
Then the metalearning process is performed (shaded part of the figure). The datasets are characterised and the resulting characteristics – metafeatures (Subsection 2.3.2) – are saved. The metadata is composed of the performances obtained on the previous phase and the metafeatures computed here.
The metadata is used to build a metadataset with the same structure as a general ML dataset (described at the beginning of this section): E instances of I independent variables $x_1, \ldots, x_I$ (the metafeatures) and one dependent variable y. The dependent variable may be, for example, the best algorithm, the performance of a given algorithm, or the performance rank of a given algorithm.
Then, metalearning computes a (meta)model m' that tries to fit the function f such that

$$\hat{y} = f(x_1, \ldots, x_I) \tag{2.18}$$

which best approximates the true output of y. The learning task to apply to the metadataset will depend on the nature of the dependent variable.
When a new dataset (d_{new}) is studied (right part of the figure), the first step is to compute its metafeatures. The metamodel obtained previously is then applied to this new metadata in order to select the algorithm and/or set the parameter values that best suits the new dataset.

2.3.2 Metafeatures

Metafeatures are values that represent characteristics of a ML experiment, of its input, or of its output, and aim at describing the knowledge obtained in the past. The design of metafeatures that contain useful information about the performance of algorithms is one of the main challenges in metalearning (Brazdil et al., 2008).
Metafeatures are typically categorised as *Simple, statistical and information-theoretic*; *model-based*, and; *landmarkers* (Brazdil et al., 2008). Next, we describe the different categories, providing some examples. For a list of frequently used metafeatures, please refer to Vanschoren (2018).

Simple, statistical and information-theoretic metafeatures: represent the characteristics of the dataset and are the most commonly used.

Simple metafeatures include the number of examples (Brazdil et al., 2003; Gama and Brazdil, 1995; Kalousis et al., 2004), number of attributes (Gama and Brazdil, 1995), number of nominal attributes (Ali et al., 2018), proportion of symbolic attributes (Brazdil et al., 2003; Kalousis et al., 2004) and proportion of missing values (Brazdil et al., 2003; Kalousis et al., 2004). Some other examples are correlation and dissimilarity (Pimentel and de Carvalho, 2018) for selecting clustering algorithms, number of jobs and machines (Pavelski et al., 2018) on a MtL approach for the flowshop problem, or capacity and demand (Gutierrez-Rodríguez et al., 2019) for selecting meta-heuristics for the vehicle routing problem.

Statistical measures include skewness (Gama and Brazdil, 1995) and kurtosis (Gama and Brazdil, 1995), but also mean, median and standard deviation of attributes (Chu et al., 2019) or default accuracy (Ali et al., 2018).

Information-theoretic measures include entropy of classes (Brazdil et al., 2003; Gama and Brazdil, 1995; Kalousis et al., 2004; Ali et al., 2018) or attributes (Gama and Brazdil, 1995; Ali et al., 2018) and mean mutual information of class and attributes (Brazdil et al., 2003; Gama and Brazdil, 1995).

Model-based metafeatures: based on the model applied to the data. Examples of this type of metafeatures are: error correlation of pairs of algorithms among different datasets (Kalousis et al., 2004), flowshop objective (Pavelski et al., 2018), fitness distance correlation (Chu et al., 2019) and number of trees (Ali et al., 2018).

Landmarkers: a quick estimate of the predictive power of an algorithm on the data. They can be obtained in one of two ways: running a simplified version of the algorithm on the data, or running the algorithm on a smaller portion of the dataset. Example landmarkers are: the *decision node* (Bensusan and Giraud-Carrier, 2000), one hidden layer NN performance (Félix et al., 2017) or information on feasible solutions (Gutierrez-Rodríguez et al., 2019).

The design of metafeatures that contain useful information about the performance of algorithms is one of the main challenges in metalearning (Brazdil et al., 2008). For example, in Félix et al. (2017) (and also included in Chapter 4) a set of metafeatures with neural network specific landmarkers was proposed. The specific neural network landmarkers improve the metamodel's performance.

2.4 Transfer Learning

Traditional ML and DM methods work under the assumption that the training and testing data are drawn from the same distribution. When the distribution changes, ML models need to be rebuilt from scratch in order to match the new data distribution. This process can be computationally expensive or even impossible if we have large datasets, slow learning processes or if there is no possibility of saving the training data.

There is a need for high-performance learners trained on old data that can be applied to the new data. This can be achieved by transfer learning (TL). TL is inspired in the human ability of reusing learned information (Pan and Yang, 2010). For example, it is easier to recognise pears after learning how to recognise apples. Also, it is easier to learn to play a musical instrument (say, the piano) if one has previous musical knowledge (for example, by knowing how to play the guitar) compared to a person with no musical knowledge at all. Transfer learning aims at producing a model for a target problem with limited training data (or none at all), by exploring knowledge obtained on a different source problem.

2.4.1 Definition and notation

TL can be characterised by the presence or absence of labelled instances in the source and target domains. In the literature, there is no consensus in the names given to each transfer scenario, when concerning this issue. The same setup is given different names by different authors, as shown on Table 2.1.

Table 2.1: Classification of TL mechanisms according to the existence of source and target labelled data.

Target \ Source	Present	Absent
Present	Supervised (Chattopadhyay et al., 2012; Daumé III, 2009) Semi-supervised (Blitzer et al., 2006; Gong et al., 2012; Liu et al., 2017) Inductive (Pan and Yang, 2010) Supervised informed (Cook et al., 2013; Feuz and Cook, 2015)	Unsupervised (Pan and Yang, 2010) Unsupervised informed (Cook et al., 2013; Feuz and Cook, 2015)
Absent	Semi-supervised (Chattopadhyay et al., 2012; Daumé III, 2009) Unsupervised (Blitzer et al., 2006; Gong et al., 2012; Liu et al., 2017) Transductive (Pan and Yang, 2010) Supervised uninformed (Cook et al., 2013; Feuz and Cook, 2015)	Unsupervised (Pan and Yang, 2010) Unsupervised uninformed (Cook et al., 2013; Feuz and Cook, 2015)

In the case we have abundant labelled source data, different names are given to the problem and these are mostly related with the amount of labelled target data: if it is present but limited, some authors name it *supervised* transfer learning (Chattopadhyay et al., 2012; Daumé III, 2009) and others name it *semi-supervised* transfer learning (Blitzer et al., 2006; Gong et al., 2012; Liu et al., 2017); if there is no labelled target data some authors name it *semi-supervised* transfer learning (Chattopadhyay et al., 2012; Daumé III, 2009) and others name it *unsupervised* transfer learning (Blitzer et al., 2006; Gong et al., 2012; Liu et al., 2017).

A different nomenclature is adopted in Pan and Yang (2010), where the authors separate the problems by the existence of labelled source data. If there is none, the problem is called *unsupervised* transfer learning. If labelled source data is present together with some labelled target data, they call it *inductive* transfer learning. Otherwise, if labelled source data is present, but there is no labelled target data, they call it *transductive* transfer learning.

A final example is the nomenclature used by Cook et al. (2013) and Feuz and Cook (2015). In this case, the presence or absence of labelled source data determines the problem to be *supervised* or *unsupervised*, respectively. On the other hand, the presence or absence of labelled target data determines if the problem is *informed* or *uninformed*, respectively. In the remainder of this chapter, we refer to the presence or absence of labelled data on the source and domains instead of using any of the classifications referred above.

To formally define transfer learning, first we will introduce some notation. For consistency, the notation and definition match the ones used in two recent transfer learning surveys (Pan and Yang, 2010; Weiss et al., 2016). For illustration we will continue using the dataset introduced in the beginning of this chapter: a generic dataset containing E instances of I independent variables $x_1, \ldots, x_I$ and one dependent variable y. Thus, x_i^e is the value of the ith independent variable in the eth instance of the dataset.

Notation: A domain $\mathcal{D}$ is defined by two parts: a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \left\{x^1, \ldots, x^E\right\} \in \mathcal{X}$. Considering the generic dataset, x^e is the eth feature vector (instance), E is the number of feature vectors in X, $\mathcal{X}$ is the space of all possible feature vectors, and X is a particular learning sample. For a given domain $\mathcal{D}$, a task $\mathcal{T}$ is defined by two parts: a label space $\mathcal{Y}$ and a predictive function $f(.)$, which is learned by the feature vector and label pairs $\{x^e, y^e\}$, where $x^e \in X$ and $y^e \in \mathcal{Y}$. Considering the generic dataset, $\mathcal{Y}$ is the set of possible values for the dependent variable, and $f(x)$ is the learner that predicts the label value for the instance x.

From the definitions above, we have a domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ and task $\mathcal{T} = \{\mathcal{Y}, f(.))\}$. Now, $\mathcal{D}_S$ is defined as the source domain data where $\mathcal{D}_S = \left\{(x_S^1, y_S^1), \ldots, (x_S^E, y_S^E)\right\}$, where $x_S^e \in \mathcal{X}$ is the eth data instance of $\mathcal{D}_S$ and $y_S^e \in \mathcal{Y}$ is the corresponding label for x_S^e. In the same way, $\mathcal{D}_T$ is defined as the target domain data where $\mathcal{D}_T = \left\{(x_T^1, y_T^1), \ldots, (x_T^E, y_T^E)\right\}$, where $x_T^e \in \mathcal{X}$ is the eth data instance of $\mathcal{D}_T$ and $y_T^e \in \mathcal{Y}$ is the corresponding label for x_T^e.
Furthermore, the source task is denoted as $\mathcal{T}_S$, the target task as $\mathcal{T}_T$, the source predictive function as $f_S(.)$, and the target predictive function as $f_T(.)$.

Definition: *Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and a learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.*
Given the notation and definition we will now discuss the situations in which transfer learning can occur. A domain can be defined as $\mathcal{D} = \{\mathcal{X}, P(X)\}$ and a task can be defined as $\mathcal{T} = \{\mathcal{Y}, f(.))\}$, which is the same as $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$. Therefore, we have that $\mathcal{D}_S = \{\mathcal{X}_S, P(X_S)\}$ and $\mathcal{T}_S = \{\mathcal{Y}_S, P(Y_S|X_S)\}$ for the source problem. The same happens for the target problem: $\mathcal{D}_T = \{\mathcal{X}_T, P(X_T)\}$ and $\mathcal{T}_T = \{\mathcal{Y}_T, P(Y_T|X_T)\}$. This way, transfer learning can occur when we have at least one of the following situations:

- $\mathcal{X}_S \neq \mathcal{X}_T$: the domains' feature spaces are different. This is claaed *heterogeneous transfer learning* (Day and Khoshgoftaar, 2017) and its most common approach consists in aligning the feature spaces. Similarly, when the feature spaces are the same ($\mathcal{X}_S = \mathcal{X}_T$) it is called *homogeneous transfer learning*. It usually aims at reducing distribution differences.

- $P(X_S) \neq P(X_T)$: this happens when the domains have the same features, but their marginal distributions are different (e.g., different frequencies in domain-specific features). A common approach in this case is domain adaptation, which consists in altering a source domain trying to make its distribution closer to the target's.

- $\mathcal{Y}_S \neq \mathcal{Y}_T$: there is a mismatch in the class space (e.g. different number of classes in the source and target problems).

- $P(Y_S|X_S) \neq P(Y_T|X_T)$: the conditional probability distribution of the source and target domains are different. This happens, for example, when the same feature value has two different meanings on the source and target domains.

There are three issues to take into account in transfer learning: *what*, *how* and *when* to transfer (Pan and Yang, 2010). The first question, *what to transfer?*, concerns the type of information transferred between the problems. The question *how to transfer?* concerns the algorithms used for the transfer of information between problems. The last question, *when to transfer?*, means to know in which situations the transfer should be performed.

2.4.2 *What* and *how* to transfer?

The first two questions (*what to transfer?* and *how to transfer?*) are closely related. Next, we will categorise the TL mechanisms in terms of the type of information transferred between problems (*what to transfer?*) while, at the same time, we will present algorithms used for the transfer of information between problems (*how to transfer?*). At the end of this subsection (Table 2.2), we present a summary of this TL categorisation. The transferred information belongs to one of four categories – instances, parameters, relational knowledge or features:

1. **Instance transfer** occurs when instances from the source domain are used for training the model for the target domain. This type of transfer occurs mostly on homogeneous TL scenarios. For example, the algorithm TrAdaBoost (Dai et al., 2007b) uses parts of the labelled train data (source) that have the same

distribution as the test data (target) to help constructing the target classification model. Also, the algorithm *kernel mean matching (KMM)* (Huang et al., 2007) tries to match distributions in source and target feature spaces. Another example is the algorithm *Kullback-Leibler Importance Estimation Procedure (KLIEP)* (Sugiyama et al., 2008) that uses the Kullback-Leibler divergence to find important instances to be transferred from the source to the target problem. In Liu et al. (2018) an ensemble framework (TrResampling) is proposed to transfer instances for classification tasks.

2. **Parameter transfer** occurs when the source and target learners share parameters or when ensemble learners are created by combining multiple source learners to form an improved target learner. Approaches to this type of transfer include weighting several source models according to target characteristics (Gao et al., 2008), from within a group of classifiers finding the source classifier that minimizes the error on the target (that happens in Yao and Doretto (2010) in algorithms MultiSource TrAdaBoost – that handles the conditional distribution differences between domains – and TaskTrAdaBoost), weighted training with source data to predict target pseudo-labels and with all this information then predict the target final labels. This is the case of algorithms *Conditional Probability based Multi-source Domain Adaptation (CP-MDA)* (Chattopadhyay et al., 2012) and *Domain Selection Machine (DSM)* (Duan et al., 2012b). These algorithms handle both marginal and conditional distribution differences between the domains. Finally, another approach is to directly transfer the parameters between problems. This is the case in algorithm *Multi Model Knowledge Transfer (MMKT)* (Tommasi et al., 2010), that handles the conditional distribution differences between domains.

3. **Relational knowledge transfer** occurs when the transferred knowledge is based on some relationship between the source and target domains. This is the least used approach in TL. There are some examples of this type of transfer in the literature. Algorithm *Deep Transfer via Markov logic (DTM)* (Davis and Domingos, 2009) discovers structural regularities in the source and instantiates them with predicates from the target problem. Another example is the algorithm *Relational Adaptive bootstraPping (RAP)* (Li et al., 2012), which uses sentiment words as a link between source and target domains and iteratively builds a target classifier from the two domains by scoring sentence structure patterns, while trying to avoid the marginal distribution differences between the domains. In Xiong et al. (2018), models are transferred to improve anomaly detection.

In another approach (Saeedi et al., 2016) the authors transfer data mapping between sensors.

4. **Feature transfer** occurs when features are transferred across domains. This type of transfer is the most used when dealing with heterogeneous TL settings. Feature transfer can be defined as symmetric or asymmetric (Weiss et al., 2016):

   (a) In **symmetric** feature transfer, a common latent feature space between the domains is discovered.

      i. For **homogeneous** TL problems, usually the aim is to overcome the marginal distribution differences among the domains. This can be achieved by discovering a set of latent features between the source and target problems, as in the algorithms *Domain Adaptation of Sentiment classifiers (DAS)* (Glorot et al., 2011) and *Transfer Component Analysis (TCA)* (Pan et al., 2011). Other approaches include finding correspondences between features (Wang and Mahadevan, 2008), learn feature representations by modelling co-occurrence between domain-independent and domain-specific features (as in algorithm *Spectral Feature Alignment (SFA)* (Pan et al., 2010)), or finding domain-independent features (as in algorithm *geodesic flow kernel (GFK)* (Gong et al., 2012)).

      ii. For **heterogeneous** TL problems the most usual approaches are discovering common features, clustering and feature augmentation. In the first technique, the algorithms find common sets of (present or latent) features between the domains. The target model is trained with the source data and applied to the target problem. This happens, for example, in Blitzer et al. (2007), Blitzer et al. (2008), Pan et al. (2008), and Raina et al. (2007) and also in the algorithms *Structural Correspondence Learning (SCL)* (Blitzer et al., 2006), *Topic-bridged probabilistic semantic analysis (TPLSA)* (Xue et al., 2008), *Heterogeneous Spectral Mapping (HeMap)* (Shi et al., 2010), *Translator of Text to Images (TTI)* (Qi et al., 2011), *Domain Adaptation using Manifold Alignment (DAMA)* (Wang and Mahadevan, 2011) and *Heterogeneous Transfer Learning for Text Classification (HTLIC)* (Zhu et al., 2011). The clustering technique consists in clustering source and target data simultaneously to infer common structures between the domains. This is the case in the algorithms *Co-clustering based classification (CoCC)* (Dai et al., 2007a), *Self-taught clustering (STC)* (Dai et al., 2008) and *Trans-*

fer Discriminative Analysis (TDA) (Wang et al., 2008) The feature augmentation technique consists in adding target and common features to the source feature set. This technique is implemented in algorithms *Heterogeneous feature adaptation (HFA)* (Duan et al., 2012c) and *Semi-supervised HFA (SHFA)* (Li et al., 2014). Other approaches include modelling the relevance of features by using metafeatures (Lee et al., 2007) and use manually paired sets of features to be transferred. This last approach is used for example in the algorithm *Cross-Language Text Classification using Structural Correspondence Learning (CL-SCL)* (Prettenhofer and Stein, 2010) by translating words from English to other languages to be able to use the models created for texts written in English to classify texts in other languages.

(b) In **asymmetric** feature transfer, the source features are re-weighted to resemble the target features.

i. For **homogeneous** TL problems, the most common approach is to first learn target pseudo-labels by using the source problem for training and then using the pseudo-labels to learn the final target labels. This technique can be used to approximate the domains marginal (as happens in the *Domain Transfer Multiple Kernell Learner (DTMKL)* (Duan et al., 2012a)) or conditional distribution (as in the *Feature Augmentation Method (FAM)* (Daumé III, 2009)), and even both (as is the case of the algorithm *Joint Distribution Adaptation (JDA)* (Long et al., 2013)).

ii. For **heterogeneous** TL problems, usually a transformation from the source to the target is found. This happens in *Multiple Outlook MAPping (MOMAP)* (Harel and Mannor, 2010), *Asymmetric Regularized cross-domain transformation (ARC-t)* (Kulis et al., 2011), *Sparse Heterogeneous Feature Representation (SHFR)* (Zhou et al., 2014b) and *Hybrid Heterogeneous Transfer learning (HHTL)* (Zhou et al., 2014a). Another approach consists in training the target model on a set of similar source features. This is the case of the algorithm *Heterogeneous Feature Prediction (HFP)* (Nam et al., 2017), where the similarity of features is obtained by a Kolmogorov-Smirnov test.

Table 2.2 contains a summary of the referred TL algorithms, considering *what* and *how* to transfer. Since the problems considered on the algorithms described do not match our problems, instead of reusing one of the referred algorithms, we create a weight transfer algorithm described later on Subsection 5.2.1.

Table 2.2: Summary of transfer learning algorithms.

Instance Transfer	**Parameter Transfer**	**Rel. Knw. Transfer**
KMM (Huang et al., 2007)	CP-MDA (Chattopadhyay et al., 2012)	DTM (Davis and Domingos, 2009)
KLIEP (Sugiyama et al., 2008)	DSM (Duan et al., 2012b)	RAP (Li et al., 2012)
	MMKT (Tommasi et al., 2010)	
Feature Transfer		
Homogeneous	Heterogeneous	
Symmetric: DAS (Glorot et al., 2011)	SCL (Blitzer et al., 2006)	CoCC (Dai et al., 2007a)
Symmetric: TCA (Pan et al., 2011)	TPLSA (Xue et al., 2008)	STC (Dai et al., 2008)
Symmetric: SFA (Pan et al., 2010)	HeMap (Shi et al., 2010)	TDA (Wang et al., 2008)
Symmetric: GFK (Gong et al., 2012)	TTI (Qi et al., 2011)	HFA (Duan et al., 2012c)
	DAMA (Wang and Mahadevan, 2011)	SHFA (Li et al., 2014)
	HTLIC (Zhu et al., 2011)	CL-SCL (Prettenhofer and Stein, 2010)
Asymm.: DTMKL (Duan et al., 2012a)	MOMAP (Harel and Mannor, 2010)	HHTL (Zhou et al., 2014a)
Asymm.: FAM (Daumé III, 2009)	ARC-t (Kulis et al., 2011)	HFP (Nam et al., 2017)
Asymm.: JDA (Long et al., 2013)	SHFR (Zhou et al., 2014b)	

2.4.3 *When* to transfer?

The ultimate objective of knowing *when to transfer* is to avoid *negative transfer*: when the transfer can harm the learning process in the target task. This issue is referred in Rosenstein et al. (2005), where the authors wish to identify when transfer learning will hurt the performance of the algorithm instead of improving it.

In the literature, there are several approaches used to try to avoid negative transfer, for example:

- Measuring data relatedness, group (or cluster) the several tasks at hand, and then only transfer between tasks that belong to the same group (Bakker and Heskes, 2003; Ben-David and Schuller, 2003; Argyriou et al., 2008; Ge et al., 2014);
- Selecting a limited amount of target data to be labelled (Liao et al., 2005);
- Removing misleading source instances (Jiang and Zhai, 2007; Ngiam et al., 2018);
- Accounting for measures that illustrate the gain in transferring, like *trade-off of transferring* (Blitzer et al., 2008), *transferability* (Eaton et al., 2008) or *PDM: Predictive Distribution Matching* (Seah et al., 2013);
- Choosing only some of the source data to be transferred (Mahmud and Ray, 2008) or just proper subsets of common features (Wang et al., 2008);

- Weight the transferred information, such that the most related sources have higher weights (Tommasi et al., 2010), which can be extended by also weighting the instances to be transferred (Yao and Doretto, 2010);

- Selecting only the most relevant domains (Duan et al., 2012b), which can be done by using specific metrics, as is the example of *ROD: Rank of Domain* (Gong et al., 2012) to evaluate which source domain to choose for transfer.

In our work, we aim to use MtL to help preventing negative transfer. This way, instead of reusing the referred metrics, we generate metafeatures that will be used on the MtL process to try to predict when the transfer will have a positive impact.

2.4.4 Metalearning and Transfer Learning

Some work has been done recently in using metalearning together with transfer learning. Metafeatures are used in Biondi and Prati (2015) for calculating similarities between the datasets. The algorithm used for this task is the *k-nearest neighbours*. In Aiolli (2012) and Do and Ng (2006) there is no use of metafeatures, since the transfers are made without choosing the best source dataset to use with a certain target dataset. In Aiolli (2012), metalearning is used to find matrix transformations capable of producing good kernel matrices for the source tasks. The matrices will then be transferred for the target tasks. The results are evaluated by performance measures as accuracy (Do and Ng, 2006) or area under the ROC curve (Biondi and Prati, 2015; Aiolli, 2012).
The transferred objects found on the studied literature are SVM parameter settings in Biondi and Prati (2015), the kernel matrices in Aiolli (2012) and the *parameter function* (responsible for mapping statistics to parameters in "bag-of-words" text classification problems) in Do and Ng (2006).
In Baghoussi and Mendes-Moreira (2018), several base learners are used for several datasets. Then, the metaknowledge obtained in the first step is transferred to the data.
Finally, our previous publications Félix et al. (2015) and Félix et al. (2016) (included in Chapter 5) are preliminary studies on the impact of transfer learning on homogeneous neural networks. The results obtained in both works suggest that transfer learning, by transferring weights, can be used to improve the neural networks performance.

2.5 Summary

As referred in Chapter 1, our research aims at providing data scientists with methods to overcome the difficulties in tuning NNs. As described in Section 2.2, as the NN training is an iterative process, it is important to correctly define its starting point and stopping criterion. Nonetheless, it is also important to define the architecture to be used and avoid unit saturation on the training process. For this, we need to correctly parameterise and initialise a NN.

As referred, parameterisation is a difficult and time-consuming process that requires user expertise. It is expected to be difficult to find a single combination of parameter values that generally leads to high performance. This way, we aim at using MtL to help in the parameterisation process, by taking advantage of experience acquired on previously learned NNs. For this, as explained in Section 2.3, we will need to design a set of informative metafeatures capable of describing the NNs' behaviour when faced with different combinations of parameter values. MtL will approach the parameter selection problem with the ML tasks referred in Section 2.1.

It was also referred in Section 2.2 that it is very important to choose a good starting point for a NN, i.e., its initial weights. The most common approach for this problem is starting the NNs with a random set of weights. However, training in NNs is an iterative process that consists in iteratively adjusting the connections' weights to minimise the NNs' prediction error. If the initial weights are too far from the optimal values, the training process will take too much time. Bearing this in mind, we wish to provide data scientists with a method to initialise the NNs, by transferring weights (relational knowledge transfer) from NNs trained previously for datasets with domains different from the one at hand (heterogeneous TL).

However, we need to make sure that the transfer will not harm the NNs' performance instead of improving it. We aim at using MtL to predict the transfer's impact and with this prevent negative transfer. For this, MtL will take advantage of previous transfer experiments to predict if transferring between specific source and target datasets will have positive or negative impact on the NN's performance. This will depend on source and target data characteristics, and also characteristics related to the transfers itself. This way, we will need to design informative metafeatures capable of describing the source and target data, but also the transfer behaviour. MtL will approach the source network selection problem with the ML tasks referred in Section 2.1.

Chapter 3

Empirical Study of the Performance of Neural Networks

Machine Learning algorithms typically have parameters that potentially enable their adaptation to new tasks. These added degrees of freedom are also a source of human and computational time consumption since finding a good combination of parameters is rarely a trivial task. Specifically, in Neural Network (NN) learning there are no generally accepted rules for parameter selection given a new learning problem. The commonly accepted solution is to perform a more or less intense and blind search in a promising region of the parameters' space for each new dataset in hand.

In this chapter we aim at answering **RQ1**(Chapter 1): *How do different parameter values impact the performance of NNs?* To answer this question, we start by studying the impact of running neural networks with several different parameter value combinations (parameterisations). We study the parameterisations' average performance, but also the impact on performance of each parameter value separately. However, as an average good parameterisation may not be good for every dataset, we also study the robustness of the results of each parameterisation.

Next, we identify the datasets used for our study and the neural network implementation considered (Section 3.1). Then we define the experimental setup (Section 3.2) considered for the study described in this chapter, followed by the results obtained (Section 3.3). Just before the Summary (Section 3.5), this chapter also includes a "cheat-sheet" (Section 3.4) that can be used to select subsets of parameters that lead to average high performance. With this, the data scientist can significantly reduce the grid search needed.

3.1 Datasets and neural network implementation

We use the same datasets for all the four phases of our research. These are described next, followed by the neural network implementation considered throughout the study.

3.1.1 Datasets

Throughout our research we use a group of benchmark regression datasets composed of numerical variables collected from UCI (Lichman, 2013), shown on Table 3.1.

Table 3.1: UCI Datasets used and number of datasets generated from them.

id	name	nr. examples	nr. attributes	nr. datasets
1	Airfoil Self-Noise	1503	5	1
2_*	CBMNPP[1]	11934	15	2
3	Combined Cycle Power Plant	9568	4	1
4	Communities and Crime	1993	101	1
s 5_*	Communities and Crime Unnormalized	1901	119	18
6	Concrete Compressive Strength	1030	8	1
7	Computer Hardware	208	9	1
8, 9	Challenger USA Space Shuttle O-Ring	23	2	2
10	Online News Popularity	39644	58	1
11_*	Parkinsons Telemonitoring	5875	21	2
12_*	Concrete Slump Test	103	9	3
13	Buzz in social media	28179	96	1
14, 15	Wine Quality	1599/4898	11	2
16	Yacht Hydrodynamics	308	6	1

Some of these datasets (marked with *) have more than one dependent variable. In that case, several datasets are created by splitting the original dataset by dependent variable. The result is the group of datasets shown on Table A.1 in Appendix A. With this, we can also evaluate and compare the behaviour of the neural networks for similar and different datasets.

[1]Condition Based Maintenance of Naval Propulsion Plants

3.1.2 Neural network implementation

We study shallow neural networks for regression on the datasets referred previously. The implementation considered for empirical validation is R's `nnet` (Venables and Ripley, 2002), which implements backpropagation for feed-forward NNs with only one hidden layer. This implementation allows several parameters to be set. In our study, we focus on the following:

- *size*: the number of units in the hidden layer;
- *Wts*: initial weights of the network;
- *decay*: value for the decay parameter (to be considered in the weight update in each iteration as referred in Equation 2.17);
- *abstol*: stop iterating if the fitting criterion falls below this value, indicating a perfect fit.

Besides, we set the parameter `linout` (use linear instead of logistic output units) to `true`, because we are looking at regression problems, and `maxit` (maximum number of iterations) to 100000, a number high enough so that the neural network does not stop before convergence is reached. Moreover, we set `formula = target ~ .` so that the neural network considers all the datasets' features for constructing the model.

3.2 Experimental Setup

With this experimental approach we aim at studying the neural networks performance according to their `size` (number of units in the hidden layer), the initial weights to use (`Wts`), weight penalty (`decay`) and the fitting criterion (`abstol`).
These experiments enable us to determine whether it is possible to find a good set of parameters and how dependent it is of the dataset. We will also collect data that will be fundamental for the metalearning study.
Figure 3.1 shows the schema of the phases considered in our research, first presented in Chapter 1 (Figure 1.2). The shaded area of the figure represents the study described in this chapter.
We performed a grid search over several neural network parameters, measuring the final predictive and computational performance. The study of the behaviour of neural networks with different parameterisations is conducted by using all the 37 datasets referred in Table A.1 in Appendix A, marked in the column "ES". The grid search

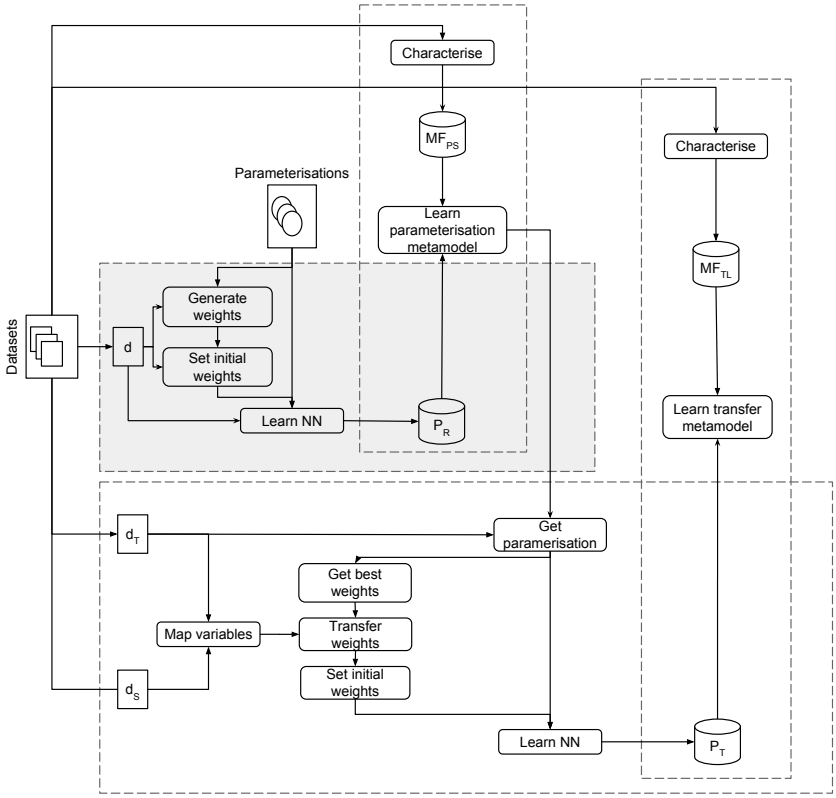


Figure 3.1: Metalearning for multiple domain Transfer learning experimental process. The shaded area corresponds to the experimental setup considered for this study.

follows the experiment performed in Félix et al. (2017). For each dataset, we test every possible combination of:

- **distribution (p_D):** the distribution from which the neural network initial set of weights is generated: $p_D \in \{u, b\}$, where
  - $u = \mathcal{U}[0, 1]$: the weights are randomly generated following a uniform distribution of values between 0 and 1;
  - $b = \mathcal{N}\left(0, \sqrt{1/I}\right)$, the distribution recommended in Bishop et al. (1995): the weights are randomly generated following a normal distribution with mean 0 and standard deviation $1/I$, where I is the number of independent variables of the dataset.
- **size (p_n):** number of units in the hidden layer: $p_n \in \{3, 5, 10, 20\}$
- **decay (p_d):** parameter for weight decay: $p_d = 1 \times 10^{-i}, i \in [0, 4]$
- **abstol (p_a):** stopping criterion: $p_a = 1 \times 10^{-j}, j \in [3, 5]$

In this experiment we are, then, considering $2 \times 4 \times 5 \times 3 = 120$ different parameter combinations. From this point forward, we refer to a combination of the above-mentioned parameter values as a *parameterisation*: $p = (p_D, p_n, p_d, p_a)$. Table D.1 in Appendix D contains the 120 parameterisations considered.
For simplification, in this chapter, we will refer to a specific parameterisation by its *id*, also presented on the table. For example, p^1 refers to the parameterisation with $id = 1$: $p^1 = (b, 3, 0.0001, 0.00001)$.
Since the initial weights are generated randomly, for each neural network parameterisation, we generate 20 different sets of weights. With this, we have $20 \times 120 = 2400$ different neural network parameterisations that are applied to each of the datasets considered.

3.2.1 Performance evaluation

The neural networks' performance is estimated with 10-fold cross-validation: we generate ten random disjoint samples that fully cover the dataset and repeat the machine learning process ten times. Each iteration (fold), a different sample is considered as testset and the remaining data is used as trainset, where the network learns until convergence.
For each fold, we evaluate the following:

- **MSE0**: the predictive performance in terms of MSE (Equation 2.4) on the testset before training. This allows us to evaluate the network's starting point;
- **duration**: the time needed for convergence;
- **MSE**: the predictive performance in terms of MSE (Equation 2.4) on the testset after training.

At the end of the process, the neural network's performance is the set of averages for each of the metrics obtained for the folds. These values are saved (in the figure, in P_R), together with the parameterisation and the initial and average final weights.

3.3 Results

Next, we evaluate the parameterisations, identifying the average best (Subsection 3.3.1) and worst (Subsection 3.3.2) performing ones and their rankings (Subsection 3.3.3). We study the impact of individual parameter values on the performance (Subsection 3.3.4), followed by a robustness analysis of the results (Subsection 3.3.5).
After that, we study the behaviour of neural networks for four datasets in which we obtained particularly low performance (Subsection 3.3.6). Finally, we evaluate neural networks for similar datasets (Subsection 3.3.7).

3.3.1 Best parameterisations

First, we want to assess if there is a parameterisation that generally originates high performance. This way, for each dataset, we determined the ones that originate the lower MSE (best parameterisations, p^{grid}). For completion, the results are presented on Table D.3a in Appendix D.

The histograms in Figure 3.2 show the frequency of each parameter value in the best parameterisations.

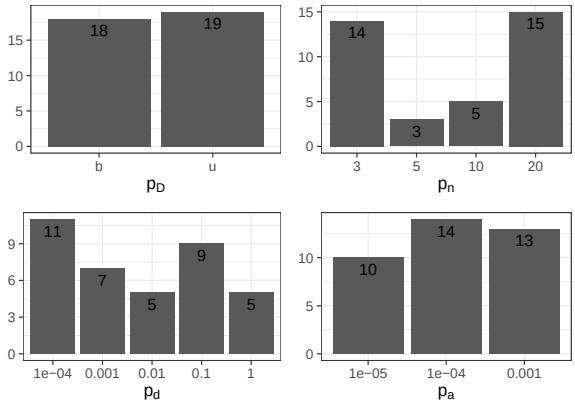


Figure 3.2: Histogram of parameters for best parameterisation found.

As we can see in the histograms, we cannot generalise the best value for any of the parameters. However, if we consider the best performing parameter value for the majority of the datasets, a good parameterisation to start with would be:

- $p^{BEST.MAJ} = (u, 20, 0.0001, 0.0001) = p^{107}$.

Besides that, some of the best p^{grid} parameterisations repeat for more than one dataset, as presented on Table 3.2.

Table 3.2: Repeated best parameterisations.

ID	p_D	p_n	p_d	p_a	**datasets**
62	u	3	0.0001	0.0001	5_4, 5_12, 5_18
3	b	3	0.0001	0.001	5_17, 7
47	b	20	0.0001	0.0001	2_2, 16
51	b	20	0.001	0.001	11, 11_2
66	u	3	0.001	0.001	5_2, 5_14

3.3.2 Worst parameterisation

The worst parameterisations for each dataset are presented on Table D.3b in Appendix D. The histogram in Figure 3.3 shows the distribution of the parameter values in the worst parameterisations.

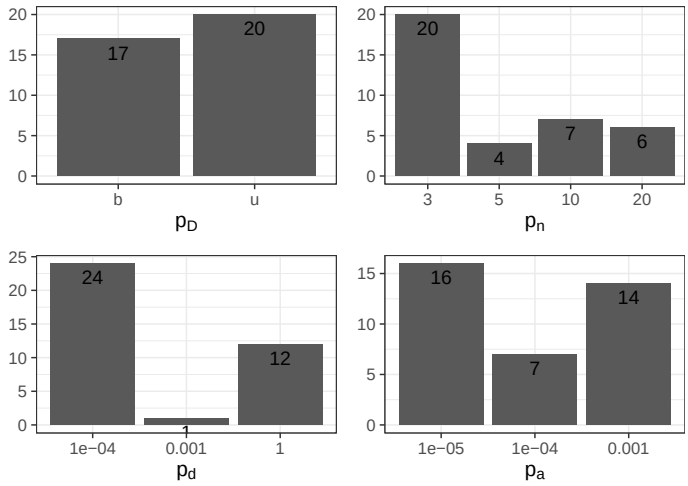


Figure 3.3: Histogram of parameters for worst parameterisation found.

With this, we could say that parameterisation $p^{WORST} = (u, 3, 0.0001, 0.00001) = p^{64}$ is a bad choice. Table 3.3 shows the parameterisations p^{WORST} that appear for more than one dataset.

Table 3.3: Repeated worst parameterisations.

ID	p_D	p_n	p_d	p_a	**datasets**
13	b	3	1	0.00001	2_1, 3, 5_18, 11_2, 12_3
14	b	3	1	0.001	2_2, 5_10, 5_12, 7
62	u	3	0.0001	0.0001	5_9, 5_11, 5_15, 11
93	u	10	0.0001	0.001	5_8, 5_14, 13
1	b	3	0.0001	0.00001	5_5, 5_13
3	b	3	0.0001	0.001	5_1, 5_7
91	u	10	0.0001	0.00001	5_17, 12_1
92	u	10	0.0001	0.0001	5_6, 12_2

3.3.3 Ranking parameterisations

We were not able to generalise a good or bad parameterisation. However, we can try to find parameterisations that, in average, correspond to higher performance. We ranked the parameterisations for each dataset in terms of performance. Table 3.4 shows the average top- and bottom-5 parameterisations, and their average rankings.

Table 3.4: Ranking: top- and bottom-5 parameterisations (average).

Top-5		Bottom-5	
id	**rank**	**id**	**rank**
112	40.14	**14**	80.95
54	40.16	**13**	81.11
114	40.73	**76**	81.65
52	40.95	**92**	82.81
115	41.00	**91**	84.62

We must highlight that the parameterisations on the top-5 share some parameter values:

- $p_D^{52} = p_D^{54} = b$ or $p_D^{112} = p_D^{114} = p_D^{115} = u$
- $p_n^{112} = p_n^{54} = p_n^{114} = p_n^{52} = p_n^{115} = 20$
- $p_d^{112} = p_d^{54} = p_d^{114} = p_d^{52} = 0.01$
- $p_a^{112} = p_a^{52} = p_a^{115} = 0.00001$ or $p_a^{54} = p_a^{114} = 0.001$

Thus, a parameterisation containing these parameters can be used for an average high performance:

$$p^{RANK.TOP5.A} = (b, 20, 0.01, 0.00001) = p^{52}$$
$$p^{RANK.TOP5.B} = (b, 20, 0.001, 0.001) = p^{54}$$
$$p^{RANK.TOP5.C} = (u, 20, 0.01, 0.00001) = p^{112}$$
$$p^{RANK.TOP5.D} = (u, 20, 0.001, 0.001) = p^{114}$$

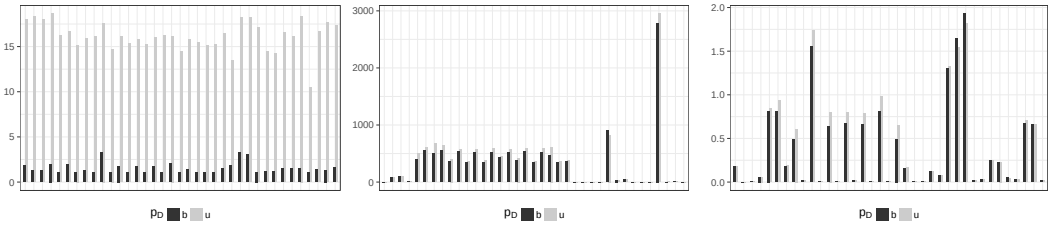
Table D.2 in Appendix D shows the top-5 parameterisations for each dataset. Some parameterisations appear in the top-5 more than once (Table 3.5). All the parameterisations in bold share the parameter $p_D = b$. This suggests that neural networks should be initialised with weights generated following that distribution.

Table 3.5: Parameterisations that appear on top-5 more than once.

freq	2	3	4	5	6	7
p	**11**, **14**, **15**, **16**, **17**, **18**, **27**, **48**, 63, 73, 75, 85, 87, 110, 112, 118, 119, 120	**40**, **46**, **51**, **54**, 61, 65, 66, 74, 100, 102, 106, 109, 113, 116	**3**, **13**, **47**, **49**, **55**, 101, 107, 108, 111	**1**, **42**, **50**, **52**	**2**	**53**

3.3.4 Parameters vs Performance

Rankings hide information, since two parameterisations with different rankings may have similar performance on a specific dataset. This way, we evaluate the impact of single parameter values on the neural networks' performance metrics considered: MSE0, duration and MSE (Figure 3.4).

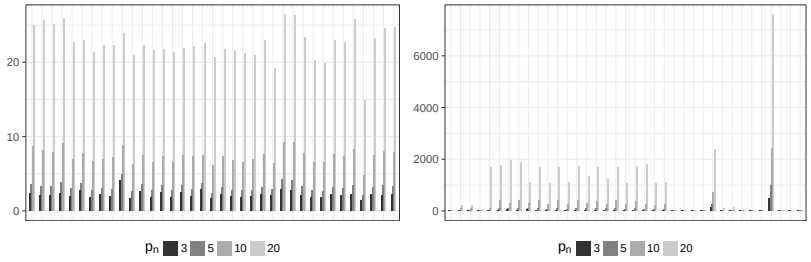


(a) MSE0 (b) duration (c) MSE

Figure 3.4: Neural network performance by p_D. The xx axis corresponds to the datasets on the same order as in Table A.1 in Appendix A. The yy axis refers to the value of each performance metric.

The neural networks with $p_D = b$ have a better starting point than the networks with $p_D = u$, as can be seen if Figure 3.4a. This suggests that the first are faster to converge. This has proven to be true, since networks with $p_D = b$ almost always have lower duration (see Figure 3.4b). Besides that, as we can see in Figure 3.4c, the MSE of the networks with $p_D = b$ is also generally lower, confirming the final hypothesis on the previous section.

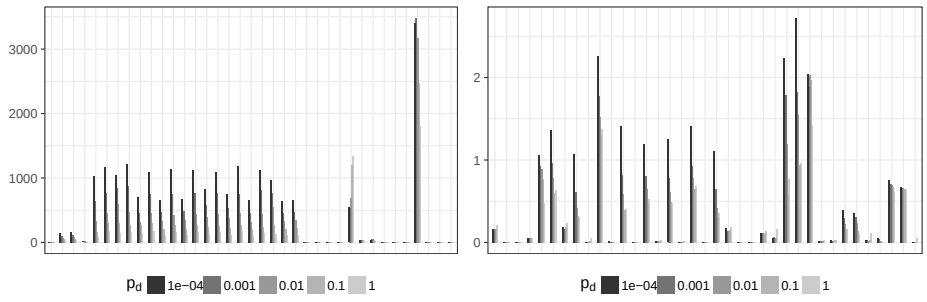
The value used for the parameter p_n influences the starting point: networks with more hidden units have higher MSE0 than smaller networks, as can be seen in Figure 3.5a. Furthermore, larger networks take longer to converge, as shown in Figure 3.5b.



(a) MSE0 (b) duration

Figure 3.5: Neural network performance by p_n. The xx axis corresponds to the datasets on the same order as in Table A.1 in Appendix A. The yy axis refers to the value of each performance metric.

As for parameter p_d (value for parameter *decay*), smaller values generally imply longer training processes (Figure 3.6a) and, in most cases, worst performance (Figure 3.6b).



(a) duration (b) MSE

Figure 3.6: Neural network performance by p_d. The xx axis corresponds to the datasets on the same order as in Table A.1 in Appendix A. The yy axis refers to the value of each performance metric.

There does not seem to be a general connection between p_n and the performance achieved. Instead, it seems to depend on the data. Also, the value used for p_d does not influence the networks' starting point. This is because this value is used for the weight update during the training process and the MSE0 is measured before. As for the parameter p_a (value used for *abstol*), there does not seem to be any general relationship between its value and the neural networks starting point, duration or performance. For completion, the plots showing these results are presented in Appendix B: Figures B.1, B.2a, B.2b, B.3a and B.3b, respectively.

3.3.5 Robustness of the performance results

Next, we analyse the parameterisations in terms of average performance over the datasets. The histogram in Figure 3.7 shows that there is a group of seven parame-

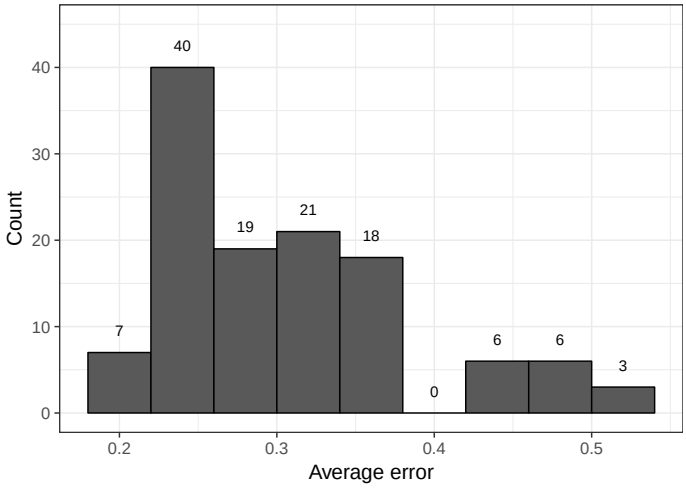


Figure 3.7: Histogram of average error by parameterisation.

terisations that originate the lower average error. These are presented on Table 3.6.

Table 3.6: Parameterisations that originate the lower average error.

ID	p_D	p_n	p_d	p_a	ID	p_D	p_n	p_d	p_a	ID	p_D	p_n	p_d	p_a
25	b	5	0.1	0.00001	40	b	10	0.1	0.00001	101	u	10	0.1	0.0001
26	b	5	0.1	0.0001	41	b	10	0.1	0.0001					
27	b	5	0.1	0.001	42	b	10	0.1	0.001					

We need to highlight that all these parameterisations have in common $p_d = 0.1$ and all except one $p_D = b$. Also, 1/3 of the parameterisations originate errors between 0.22

and 0.26. Besides that, there is a set of fifteen parameterisations (Table 3.7) that originate worst performance.

Table 3.7: Parameterisations that originate worst performance.

ID	p_D	p_n	p_d	p_a	ID	p_D	p_n	p_d	p_a	ID	p_D	p_n	p_d	p_a
1	b	3	0.0001	0.00001	61	u	3	0.0001	0.00001	91	u	10	0.0001	0.00001
2	b	3	0.0001	0.0001	62	u	3	0.0001	0.0001	92	u	10	0.0001	0.0001
3	b	3	0.0001	0.001	63	u	3	0.0001	0.001	93	u	10	0.0001	0.001
16	b	5	0.0001	0.00001	76	u	5	0.0001	0.00001					
17	b	5	0.0001	0.0001	77	u	5	0.0001	0.0001					
18	b	5	0.0001	0.001	78	u	5	0.0001	0.001					

Here we need to highlight the only parameter value common to all these parameterisations: $p_d = 0.0001$. However, besides the average, we also need to consider the variance in the neural networks (Figure 3.8). We can see that most of the

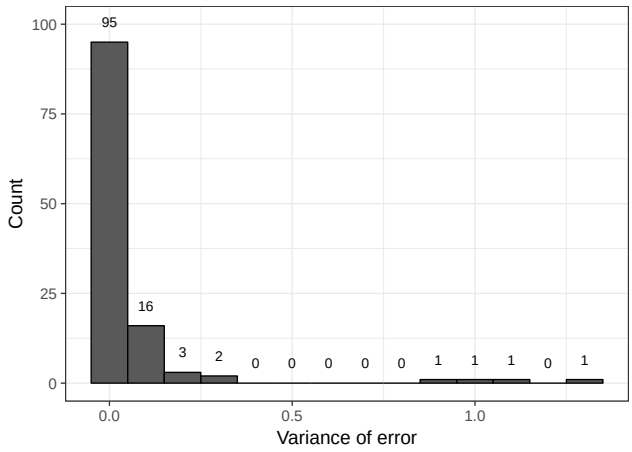


Figure 3.8: Histogram of variance in error by parameterisation.

parameterisations have low variance, suggesting robust results. There is nonetheless a set of four parameterisations with very high variance (Table 3.8).

Table 3.8: Parameterisations with high variance.

ID	p_D	p_n	p_d	p_a
1	b	3	0.0001	0.00001
16	b	5	0.0001	0.00001
61	u	3	0.0001	0.00001
62	u	3	0.0001	0.0001

Note that these also belong to the set of parameterisations that originate worst performance. Figure 3.9 shows the comparison of the average and variance of the MSE.

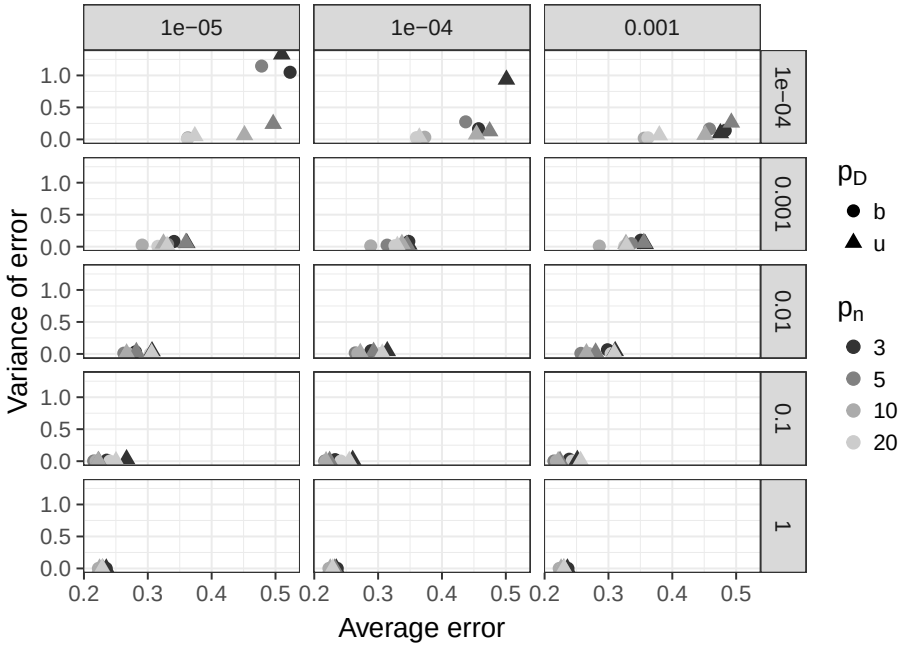


Figure 3.9: Average and variance in error by parameterisation. The columns refer to different p_a values and the rows to different p_d values.

In the figure, different shapes represent p_D and the colours represent (p_n). Each column represents a different p_a value and each row a different p_d value. Both variance and average seem to decrease with increasing p_d. Figure 3.10 contains the same information, but only for $p_d \in \{0.1, 1\}$.

We see that $p_d = 0.1$ gives us lower error with higher variance, while with $p_d = 1$ we obtain a slightly higher average error, but lower variance. In both cases the p_d parameter value that originates better results is $p_d = b$. As for p_n, the average best results are obtained with the value 10. The parameter p_a seems to influence the variance ($p_a = 0.00001$ originates smaller variance on the results) but not the average error. This way, if parameterisation $p^{ROBUST.AVG}$ is used, an average lower MSE will be achieved but the values will have higher variance. On the other hand, if we do not mind having slightly higher average MSE, but lower variance, parameterisation $p^{ROBUST.VAR}$ can be used:

- $p^{ROBUST.AVG} = (b, 10, 0.1, 0.00001)$
- $p^{ROBUST.VAR} = (b, 10, 1, 0.00001)$

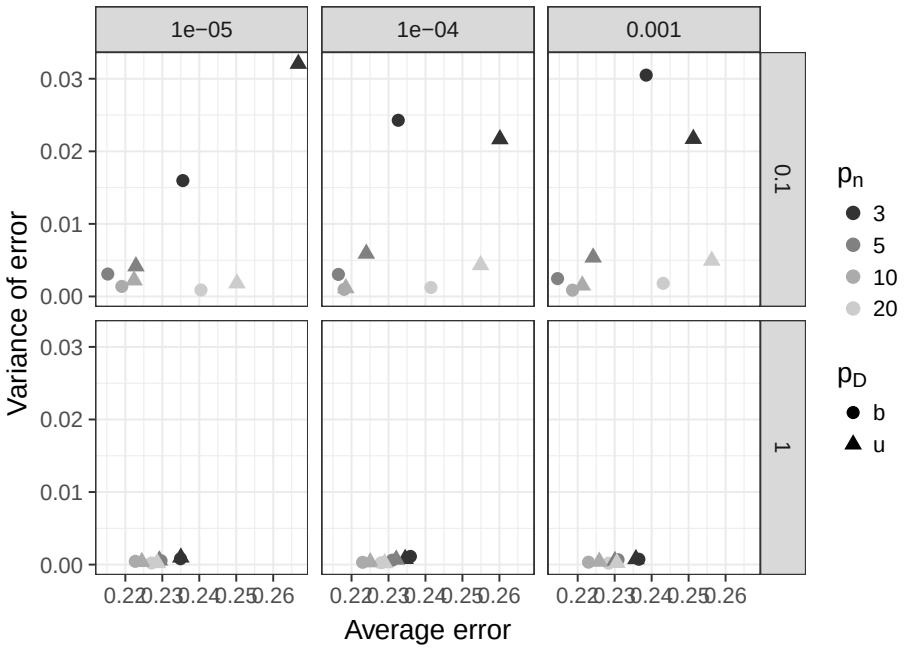


Figure 3.10: Average and variance in error by parameterisation: higher decay. The columns refer to different p_a values and the rows to different p_d values.

3.3.6 Difficult datasets

With our experiments we were able to detect that not every parameterisation is good for learning a certain dataset. In fact, we discovered that some parameterisations are "bad" for a given dataset.

Here we consider that a **good** parameterisation is one with which: 1) the network learning process makes the neural network learn ($MSE < MSE0$); 2) the network learning process is useful: the neural network performance is better than a benchmark method – here the one that predicts the average of the trainset for each example in the testset ($MSE < MSE\_MEAN$).

A **bad** parameterisation is one with which the neural network does not learn and/or is not useful. We have identified some cases (we call them *problems*) where this occurs:

P1: $MSE > MSE0$
P2: $MSE > MSE\_MEAN$

Problem P1 occurs in 1.63% of the NNs executed, P2 in 5.18% and in 1.60% of the networks both problems occur simultaneously, as shown on Table 3.9.

Table 3.9: Total distribution of problems (percentage).

P1 (%)	**P2 (%)**	**Both (%)**
1.63	5.18	1.60

However, this behaviour is not uniform among the datasets, as shown on Table 3.10.

Table 3.10: Distribution of problems over datasets (percentage).

Dataset	**P1 (%)**	**P2 (%)**	**Both (%)**	**Dataset**	**P1 (%)**	**P2 (%)**	**Both (%)**
4	11.71	25.21	11.42	**5_1**	2.13	6.29	2.13
8	8.54	37.75	8.21	**5_3**	4.04	10.04	4.04
9	13.38	59.79	12.96	**5_5**	2.00	5.75	2.00
12_1	0.00	0.04	0.00	**5_7**	4.21	9.96	4.21
12_2	0.04	0.04	0.04	**5_9**	2.67	6.67	2.67
14	0.00	1.63	0.00	**5_11**	3.21	7.04	3.21
				5_13	2.33	6.33	2.33
				5_15	4.50	10.04	4.50
				5_16	0.00	0.04	0.00

This table only contains information on the datasets in which the problems occur (15 out of the total 37). Dataset 10 has a particular behaviour and will be referred to later in separate. As for the remaining, some of the datasets show this behaviour more often: 4, 8 and 9. Datasets 8 and 9 are too small (2 attributes, 1 target, 23 examples) so maybe the networks cannot learn properly. For this, these datasets will not be used for the rest of the research. Next we analyse the distribution of errors over the parameters (Table 3.11).

Table 3.11: Distribution of problems over parameters (percentage).

(a) p_D.

p_D	**P1 (%)**	**P2 (%)**	**Both (%)**
b	2.84	4.54	2.78
u	0.42	5.83	0.42

(b) p_d.

p_d	**P1 (%)**	**P2 (%)**	**Both (%)**
0.0001	4.13	11.57	4.12
0.001	2.07	7.88	2.05
0.01	1.52	5.61	1.49
0.1	0.41	0.86	0.35
1	0.03	0.00	0.00

Considering the parameter p_D, we can see (Table 3.11a) that the neural networks with $p_D = b$ are the ones in which more problems occur, although in low percentage.

For the p_d parameter we can see (Table 3.11b) that there seems to be a relationship between the parameter value and the percentage of problems found: there are more problems with lower p_d values.
As for parameters p_n and p_a there doesn't seem to be any connection between the value used for the parameter and the problem ratio. For completion, these results are presented on Tables C.1a and C.1b in Appendix C, respectively.

3.3.6.1 Special case: dataset 10 (percentage)

As referred, this dataset has special behaviour. In this dataset, problem P1 occurs in 50.42% of the neural networks executed, P2 in 97.92% and in 50.42% of the networks both problems occur simultaneously (Table 3.12).

Table 3.12: Total distribution of problems for dataset 10 (percentage).

P1 (%)	**P2 (%)**	**Both (%)**
50.42	97.92	50.42

Considering the distribution from which the initial weights were generated, we can see (Table 3.13) that the neural networks with $p_D = b$ are the ones in which more problems occur.

Table 3.13: Distribution of problems over p_D for dataset 10.

p_D	**P1 (%)**	**P2 (%)**	**Both (%)**
b	98.67	99.83	98.67
u	2.17	96.00	2.17

These results suggest that this dataset may be responsible for the higher ratio of *problems* verified in neural networks parameterised with $p_D = b$.
As for the remaining parameters there doesn't seem to be any connection between the value used for the parameter and the problem ratio.
For completion, these results are presented on Tables C.2a, C.2b and C.2c in Appendix C.

3.3.7 Similar datasets

We have referred that some of the datasets have been obtained from the same source dataset. When looking separately at these, some of the parameterisations appear in the top-5 for more than one dataset in the same group (Table 3.14).

Table 3.14: Parameterisations that appear more than once in the top-5 for datasets on the same group.

Group	5_* (18)				8, 9 (2)	11_* (2)	12_* (3)
Nr.	5	4	3	2	2	2	2
p	4	79 70, 81, 120	5, 60, 65, 67, 59, 68, 83, 86	7, 9, 45, 56, 58	19	8	46, 47

We have made some observations from these results:

- Parameterisation p^{19} appears in the top-5 parameterisations in both datasets for the group 8, 9;
- The same happens with parameterisation p^{8} for the group 11_*;
- Parameterisations p^{46} and p^{47} appear in the top-5 for two of the three datasets in group 12_*;
- For group 5_*parameterisation p^{4} which appears in the top-5 of almost 28% of the group's datasets, and parameterisation p^{79} for 22%;
- The other ones presented on the table appear in the top-5 for three or two of the groups' datasets.

This suggests that there might be some characteristic on the data that makes a parameterisation more suitable for learning it. In the next chapter we describe the use of metalearning to select high-performance parameterisations, by taking advantage of data characteristics.
We now present some sets of parameter values that can be used to parameterise neural networks in order to achieve high performance or, at least, reduce the grid needed for tuning the parameters.

3.4 Cheat Sheet

We started with a grid search with 120 entries and were able to find some parameter combinations that can lead to good performance of neural networks for regression problems. Table 3.15 presents a summary of the parameterisations found.

Table 3.15: Good parameterisations for neural networks in regression problems.

	p_D	p_n	p_d	p_a
$p^{BEST.MAJ}$	u	20	0.0001	0.0001
$p^{RANK.TOP5.A}$	b	20	0.01	0.00001
$p^{RANK.TOP5.B}$	b	20	0.001	0.001
$p^{RANK.TOP5.C}$	u	20	0.01	0.00001
$p^{RANK.TOP5.D}$	u	20	0.001	0.001
$p^{ROBUST.AVG}$	b	10	0.1	0.00001
$p^{ROBUST.VAR}$	b	10	1	0.00001

By analysing the impact of individual parameter values on the neural networks, we also found some partial parameterisations that have proven to be good starting points for minimising the neural networks' MSE0, duration and MSE while reducing the grid size to $1/8$, $1/40$ or $1/10$, respectively. These are presented on Table 3.16.

Table 3.16: Good partial parameterisations for neural networks in regression problems, useful for reducing the grid search.

	p_D	p_n	p_d	p_a	grid size
$p^{PART.MSE0}$	b	3	?	?	15
$p^{PART.DURATION}$	b	3	1	?	3
$p^{PART.MSE}$	b	?	1	?	12

3.5 Summary

We studied the performance of neural networks for regression problems. We performed a grid search over several parameters in order to study the behaviour of different parameterisations and the impact of the single parameter values separately. We also studied the robustness of the parameterisations' performance in several datasets.
We discovered that there are certain parameter values that can make the neural networks have better starting points ($p_D = b$, $p_n = 3$), learn better ($p_D = b$) or

faster ($p_D = b$, $p_n = 3$, $p_d = 1$). Besides that, we also identified four datasets for which the neural networks had generally bad performance. However, as expected, we did not find a parameterisation that generally leads to high performance in all of the datasets.

Finally, summarising our findings, we suggested a group of neural network parameterisations expected to lead to high performance. Furthermore, we also suggested a set of partial parameterisations that can be used to construct a smaller grid for the parameter value selection. In both cases, the time needed for selecting the parameter values for neural networks significantly decreases, while not affecting their predictive performance.

Chapter 4

Metalearning for Parameter Selection in Neural Networks

In this chapter we look for an answer to **RQ2** (Chapter 1): *Can metalearning be used to support the parameterisation of neural networks?* For that, we propose two sets of metafeatures to characterise the datasets. These metafeatures aim at capturing the datasets' characteristics that can be used to select a high-performance parameterisation for configuring a neural network for the dataset at hand.
Next, we describe the metafeatures considered (Section 4.1) followed by the experimental setup (Section 4.2) used to validate our method. Next, we discuss the results obtained (Section 4.3) and, finally, we present a summary (Section 4.4) of the findings.

4.1 Metafeatures for Parameter Selection

The purpose of the metafeatures is to characterise the datasets. These characteristics are then used by the metalearning and mapped, in this case, to the best parameter values for a specific dataset.
We propose two different sets of metafeatures for the parameter selection task: MF^1_{PS} and MF^2_{PS}. The first set is composed of traditional metafeatures and NN-specific landmarkers. The second set is constructed based on the first: it contains the same traditional metafeatures, and a larger set of NN-specific landmarkers, which are expected to better characterise the datasets. The sets of metafeatures are described next.

4.1.1 MF^1_{PS} metafeatures

The first set of metafeatures, MF^1_{PS}, is composed of 78 features. These are divided into nine groups:

- **G1**: metafeatures describing the dataset
- **G2**: metafeatures describing the attributes
- **G3**: metafeatures describing the relationship between attributes
- **G4**: metafeatures describing the attributes' distributions
- **G5**: metafeatures describing the existence of outliers on the dependent variables
- **G6**: metafeatures describing the dependent variable's distribution
- **G7**: metafeatures describing the relationship between the independent
- **G8**: general landmarkers
- **G9**: Neural network specific landmarkers

Let us focus on the last group, *Neural network specific landmarkers*. Landmarkers are quick estimates of the algorithms performance on the data and are normally obtained in one of two ways:

1. Apply the algorithm to a subset of the data
2. Apply a simplification of the algorithm to the data

This way, we can obtain estimates of the algorithms performance, without needing as much computational resources as for applying the algorithm to the complete dataset. In the case of **G9**, the neural networks specific landmarkers follow the second approach: applying simpler models (neural networks with 1 or 3 hidden units) to the data, and evaluating the predictions according to metric $\mathcal{M} \in mse0, mse, w.mean.dif, w.sd.dif$, where $mse0$ is the MSE obtained without training, mse is the MSE obtained after training, and $w.mean.dif, w.sd.dif$ are the mean and standard deviation of the difference between the network's initial and final weights.

The nine groups of metafeatures, describe the following information:

G1: metafeatures describing the dataset	
n.examples	number of instances on the dataset
n.attrs	number of attributes on the dataset
r.n.attrs.n.examples	$\frac{n.attrs}{n.examples}$
r.n.examples.n.attrs	$\frac{n.examples}{n.attrs}$

G2: metafeatures describing the attributes	
n.bin.fea	number of features containing only two distinct values
n.h.outlier	number of features with outliers
n.tri.fea	number of features containing only three distinct values
r.num.bin.fea.n.examples	$\frac{n.bin.fea}{n.examples}$
r.n.h.outlier.n.attrs	$\frac{n.h.outlier}{n.attrs}$
r.n.h.outlier.n.examples	$\frac{n.h.outlier}{n.examples}$
r.num.tri.fea.n.attrs	$\frac{n.tri.fea}{n.attrs}$
r.num.tri.fea.n.examples	$\frac{n.tri.fea}{n.examples}$
r.num.bin.fea.n.attrs	$\frac{n.bin.fea}{n.attrs}$

G3: metafeatures describing the relationship between attributes	
avg.abs.attr.correlation	average absolute attribute correlation
prop.cor.gt.50	proportion of attributes with mutual correlation over 50%

G4: metafeatures describing the attributes' distributions	
avg.skewness	average skewness of the attributes
avg.abs.skewness	average absolute skewness of the attributes
avg.kurtosis	average kurtosis of the attributes
avg.means	average mean of the attributes
avg.sds	average standard deviation of the attributes

G5: metafeatures describing the existence of outliers on the dependent variables	
target.h.outlier	$\frac{standarddeviation}{standarddeviationofalphatrimmedmean}$
target.has.outliers	1 (target variable has outliers), or 0 (otherwise)

G6: metafeatures describing the dependent variable's distribution	
range.target.rel.avg	range of the target variable's values relative to its average
target.coefficient.variation	$\frac{targetvaluesstandarddeviation}{targetvaluesmean}$
abs.target.coefficient.variation	absolute value of target.coefficient.variation
target.cv.sparsity	0 ($target.coefficient.variation < 0.2$), 1 ($target.coefficient.variation < 0.5$), or 2 (otherwise)
target.abscv.sparsity	0 ($abs.target.coefficient.variation < 0.2$), 1 ($abs.target.coefficient.variation < 0.5$), or 2 (otherwise)
target.stationarity	1 (target's standard deviation higher than its mean), 0 (otherwise)
target.hist.sparsity	standard deviation of the proportions of a histogram with 100 bins of target values
avg.mean.res.dist.adjacent.target	average mean distance between each target value and its two neighbours (sorted by value)

G7: metafeatures describing the relationship between independent and dependent variables	
prop.target.cor.gt.50	proportion of attributes with correlation to target over 50%
avg.abs.target.correlation	average absolute correlation between attributes and target

G8: general landmarkers	
r.squared	R^2 coefficient of multiple linear regression
clustering.{3, 5, 10, 20}	number of points in each cluster, for models with 3, 5, 10 and 20 clusters
d.tree.leaves	number of leaves in a decision tree
d.tree.mse	MSE obtained by decision tree algorithm
mean.mse	MSE obtained by predicting the average of the target values

G9: Neural network specific landmarkers	
l.nnet.1h.M	the metric $\mathcal{M}$ (M) of a neural network with one hidden unit
l.nnet.3h.M	the metric $\mathcal{M}$ (M) of a neural network with three hidden units

4.1.2 MF^2_{PS} metafeatures

The second set of metafeatures, MF^2_{PS} (with 12795 metafeatures) is based on MF^1_{PS}. The metafeatures in groups G1 to G8 are maintained and the NN specific landmarkers (G9) are replaced by new ones. To obtain this set of landmarkers we trained several neural networks, considering different parameterisations: N, D and A take the same values as parameters p_n, p_d and p_a in the grid search.
As referred in Subsections 2.3.2 and 4.1.1, to obtain landmarkers we either:

1. Apply the algorithm to a subset of the data. Here we execute the full neural networks in a smaller part of the dataset (we use $SS \in \{10, 25, 50, 100, 0.1\times|E|\}$).

2. Apply a simplification of the algorithm to the data. Simpler models can be obtained with smaller neural networks (one hidden unit, for example) or by limiting the number of iterations (we use $IT \in \{0..10\}$).

The performance indicators are obtained by computing one of the metrics $\mathcal{M} \in \{mse0, mse, time, learn, w.sd, w.mean, w.cv\}$, where $mse0$ is the network's mse with zero learning iterations, mse is the network's mean squared error after convergence, $time$ is the amount of time needed for convergence, $learn = mse0 - mse$ and the metrics relative to weights (w) refer to the differences between initial and final weights:

$w.sd$ is the standard deviation of the weights differences, $w.mean$ is its mean, and $w.cv$ is its coefficient of variance ($w.cv = \frac{w.sd}{w.mean}$).

This way, the NN specific landmarkers (**G10**) refer to:

G10: neural network specific landmarkers	
lm_N_D_A_IT_SS_M	the metric $\mathcal{M}$ (M) of a neural network with N hidden units, decay=D and abstol=A, run for a maximum of IT iterations or using a sub-sample of size SS
lm_1h_best_decay	the best decay found for neural networks with only 1 unit on the hidden layer
lm_1h_best_abstol	the best abstol found for neural networks with only 1 unit on the hidden layer
lm_IT_it_M	the metric $\mathcal{M}$ (M) of neural networks run for a maximum of IT iterations
lm_SS_ss_M	the metric $\mathcal{M}$ (M) of neural networks run with a sub-sample of size SS
lm_N_D_A_IT_d_NA_M	the difference in the metric $\mathcal{M}$ (M) when running the networks for IT and *IT-1* iterations with N hidden units, decay=D and abstol=A, not considering the subsample size (NA)
lm_N_D_A_NA_SS_d_M	the difference in the metric $\mathcal{M}$ (M) when running the networks with sub-samples with sizes SS and *SS-1* with N hidden units, decay=D and abstol=A, not considering the number of iterations

To summarise, we are considering the following two sets of metafeatures:

MF_{PS} metafeatures
$MF^1_{PS} = G1 + G2 + G3 + G4 + G5 + G6 + G7 + G8 + G9$
$MF^2_{PS} = G1 + G2 + G3 + G4 + G5 + G6 + G7 + G8 + G10$

4.2 Experimental setup

Our hypothesis is that *metalearning can be used for automatic selection of parameters for regression neural networks.* We predict the parameter values that yield higher

performance, using the parameterisations' performance data obtained in the previous phase and the metafeatures described in the previous section. We are only considering the performance data on the datasets marked in column "PS" on Table A.1 in Appendix A and, from here further, networks parameterised with $p_D = b$.

We evaluate the metamodels in terms of accuracy of the predictions (meta-level evaluation) and performance of the NNs parameterised with the predicted values (base-level evaluation). If the NNs parameterised with the predicted values have high performance, then we have evidence to support our hypothesis.

Figure 4.1 shows the schema of the phases considered in our research, first presented in Chapter 1 (Figure 1.2). The shaded area of the figure represents the study described in

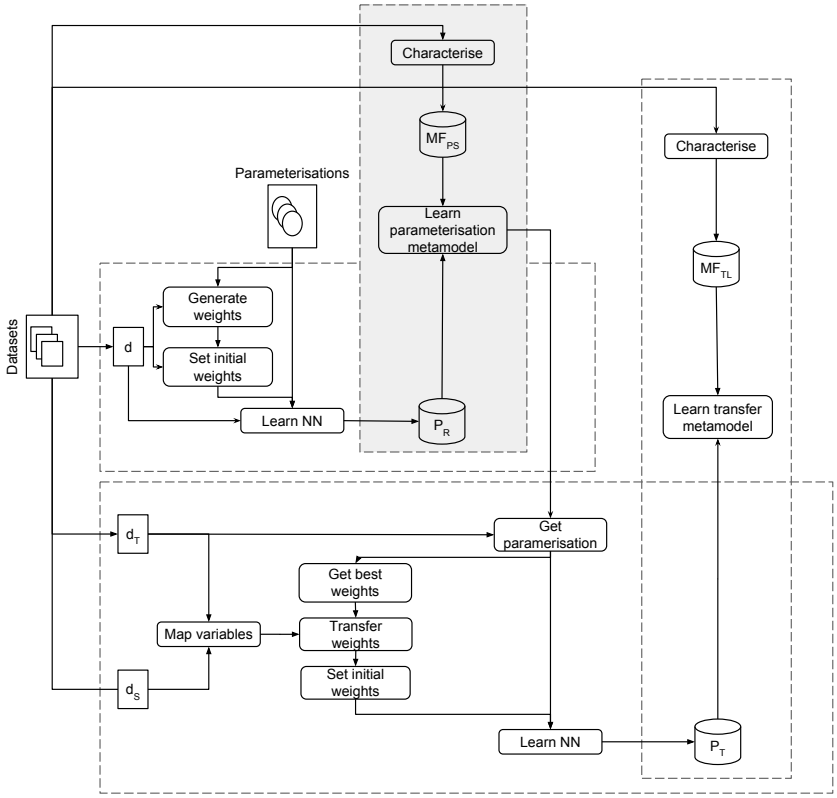


Figure 4.1: Metalearning for multiple domain transfer learning experimental process. The shaded area corresponds to the experimental setup considered for learning the parameterisation metamodel.

this chapter. The first step is to characterise the datasets, obtaining their metafeatures (MF_{PS}, detailed in Section 4.1). The metadataset is composed by the metafeatures and the best parameterisation found for each dataset.

Given the large number of metafeatures, we performed feature selection using `R` package `caret` (from Jed Wing et al., 2017). First, we removed the features with high

mutual correlations (over 0.75) following the recommendation of `findCorrelation`, which searches through the correlation matrix and returns the columns that should be removed to reduce pair-wise correlations.

Then, we performed Recursive Feature Elimination (RFE) with the aim of selecting the smallest set of features that ensures the best predictive performance. This method starts off by creating a model using all of the variables, estimating their importance and discarding some of the least important ones. The process is repeated recursively comparing the model's performance for each subset of features. The output is the set of features that originated the best performance. We used the method `rfe` to select the best model. This method can be parameterised by defining the types of functions (we use `ldaFuncs`, `lmFuncs` and `rfFuncs`), and the sizes of the subsets of features (we use $sizes = [1, \ldots, I]$, where I is the total number of features) that should be considered.

Metamodels are generated for different meta-learning approaches. We start by considering the set of metafeatures MF^1_{PS} (Subsection 4.1.1) and use a classification (Subsection 2.1.1) approach. Then, since the values we are trying to predict are numerical, a regression (Subsection 2.1.2) approach is used for the same metadata. Because the metatarget values are discontinuous and have an underlying order, we also include ordinal regression (OR, subsection 2.1.2.1)) techniques. Then we repeat the classification and regression approaches considering the set of metafeatures MF^2_{PS} (Subsection 4.1.2). Finally, to test if the metatargets are related, we consider a multi-output regression (Subsection 2.1.2.2) approach.

In the multi-output regression approach, we consider the techniques referred to in Subsection 2.1: Multi-target regressor stacking (MTRS) and Regressor Chains (RC). Since the number of possible RC chains is small (6), we consider all of them:

RC1: $p_n \rightarrow p_d \rightarrow p_a$	RC4: $p_d \rightarrow p_a \rightarrow p_n$
RC2: $p_n \rightarrow p_a \rightarrow p_d$	RC5: $p_a \rightarrow p_n \rightarrow p_d$
RC3: $p_d \rightarrow p_n \rightarrow p_a$	RC6: $p_a \rightarrow p_d \rightarrow p_n$

For example, the first chain, *RC1*, consists in starting with the prediction of parameter p_n, then the parameter p_d and, finally, the parameter p_a.

The approaches and algorithm implementations considered are presented on Table 4.3.

For simplification we refer to the implementations used for all the approaches by A_L (linear algorithms), A_T (tree-based algorithms) and A_F (forest-based algorithms), as shown in the table.

Table 4.3: Algorithm implementations considered.

Approach	A_L	A_T	A_F
Classification	`lda`	`rpart`	`randomForest`
Regression	`lm`	`rpart`	`randomForest`
Ordinal Regression		`ctree`	`cforest`
Multi-output Regression	`lm`	`rpart`	`randomForest`

4.2.1 Performance evaluation

The performance evaluation is based on leave-one-out cross validation. Our meta-dataset contains one example for each of the 35 datasets considered. The metalearning process is repeated 35 times, each considering a different instance as testset, and the remaining 34 as the trainset.
Furthermore, the evaluation is performed at two levels: meta and base. We also compare these results with a baseline (BL) model: recommending the most frequent value for each parameter.
The meta-level evaluation measures the predictive performance of the model used in metalearning. The evaluation metrics used at this level are the ones described in Subsection 2.1:

- **ACC** (Equation 2.2): for classification approaches;
- **RRMSE** (Equation 2.5): for (single and multi-output) regression approaches;
- **MZOE** (Equation 2.7): for ordinal regression approaches;

For the base-level evaluation we measure the MSE of the NN parameterised with the values suggested by the metalearning models. We also measure the performance of the NNs parameterised in the following three different ways:

- **best:** using the set of parameters found from the grid that yielded the best performance (average MSE=0.141);
- **baseline**: using the parameter configuration suggested by the baseline model referred above (average MSE=0.282);
- **worst**: using the set of parameters found from the grid that yielded the worst performance (average MSE=0.563).

For easier comparison, we present the base-level results as the relative improvement in performance with respect to the worst NNs. When compared to those, the NNs configured with the parameters recommended by the baseline model have an improvement

of 50%, and the NNs configured with the best parameters found have an improvement of 75%. These values are also presented in the results section below.

4.3 Results

In this section we present the results of our experiments for determining the metalearning's ability to predict the best parameters for a NN. We start by comparing the results of the classification and regression approaches using the MF^1_{PS} metafeatures only. We then present the results of the same approaches for the metadataset composed of the extended MF^2_{PS} metafeatures. Finally, we present the results of multi-output regression on the metadataset containing MF^2_{PS} metafeatures.

4.3.1 Classification Approach with MF^1_{PS} metafeatures

Our first set of experiments addresses the problem of NN parameter selection as a classification approach, as in Félix et al. (2017) with MF^1_{PS} metafeatures. The baseline performance for each parameter p_n, p_d and p_a, is: 43%, 34% and 37%. [1] The meta-accuracy results are shown on Table 4.4.

Table 4.4: Meta-level evaluation of the classification experiments with MF^1_{PS} metafeatures, in terms of accuracy percentage.

	A_L	A_F	A_T	BL
p_n	**54**	51	40	43
p_d	43	**51**	37	34
p_a	34	**54**	51	37
avg	44	**52**	43	38

Compared to the baseline, the gains obtained with A_L for the prediction of the size of the hidden layer (p_n) represents a 26% increase in accuracy. The gains obtained with A_F in the other two problems were 50% and 46% respectively for the p_d and p_a predictions. On average, the best performing algorithm is A_F, with an accuracy 37% higher than the baseline. Compared to the ones obtained in Félix et al. (2017), these results show that, as expected, the increase in the number of examples produces better metalearning results.

[1] The values obtained here are different from the original ones (Félix et al., 2017), because we considered more datasets in our work.

4.3.1.1 Base-level evaluation

Figure 4.2 shows the improvement of the NNs configured with the parameters recommended by metalearning.

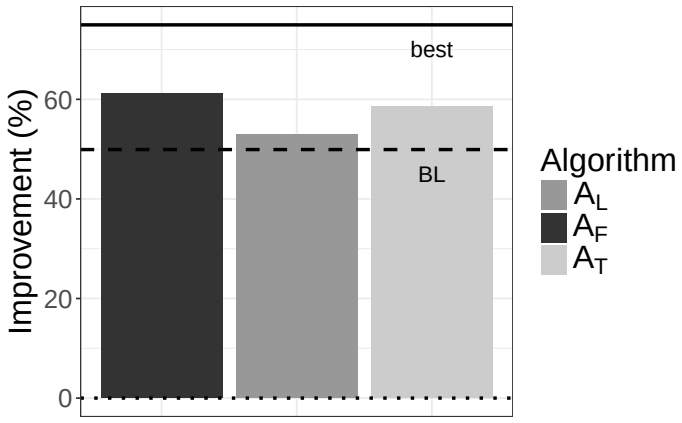


Figure 4.2: Base-level evaluation of the classification experiments with MF^1_{PS} metafeatures, in terms of improvement.

The NNs trained with the parameters recommended by A_F (average MSE=0.219) obtain a 61% improvement, which is half way between the improvement of the baseline model and the improvement of the best configuration found in the grid. Therefore, metalearning seems to be useful for parameter prediction.

4.3.2 Single-target Regression Approach with MF^1_{PS} metafeatures

The metatargets domains are small, which first motivated a classification approach. However, the values are numerical and, thus, the problem can also be addressed as regression. Additionally, there is an intermediate approach – ordinal regression – that is suitable for target variables with small but ordered domains.

We also perform traditional classification and regression with *ordered* target values: instead of using, for example, the original values 3, 5, 10 and 20 for parameter p_n, we use the values 1, 2, 3 and 4 to express the order of the original values. Since we are considering a regression approach, the results are evaluated at the meta-level in terms of RRMSE. The results are presented on Table 4.5.

For parameter p_n, the best result is achieved with A_T on the ordered targets and represents a performance 23% above the baseline. As for parameters p_d and p_a, the A_F on the original target obtained performance 37% and 43% higher than the baseline,

Table 4.5: Meta-level evaluation of the regression experiments with MF^1_{PS} metafeatures, in terms of RRMSE.

	original			ordered			
	A_L	A_F	A_T	A_L	A_F	A_T	BL
p_n	1.007	0.868	0.972	1.067	1.061	**0.811**	1.048
p_d	1.201	**0.756**	0.950	1.219	1.174	1.225	1.203
p_a	1.054	**0.748**	0.822	1.417	1.180	1.540	1.318
avg	1.087	**0.791**	0.915	1.234	1.138	1.192	1.190

respectively. On average, the best results are achieved with A_F on original targets, representing an improvement in performance 34% over the baseline's.
For ordinal regression, the results were evaluated with MZOE (Table 4.6). On the

Table 4.6: Meta-level evaluation of the OR (and classification and regression) with ordered targets using the MF^1_{PS} metafeatures, in terms of MZOE percentage.

	ordinal		classification			regression			
	A_F	A_T	A_L	A_F	A_T	A_L	A_F	A_T	B_L
p_n	51	57	**49**	57	60	86	80	51	57
p_d	63	66	**51**	54	63	83	80	80	66
p_a	**49**	63	66	**49**	**49**	66	60	80	63
avg	54	62	55	**53**	57	78	73	70	62

table we compare these results with the ones obtained by using traditional regression and classification with ordered values in the targets.
The best results for targets p_n and p_d are obtained with classification on ordered targets using A_L, respectively 14% and 23% higher than the baseline. For target p_a the best result (MZOE=49), is 22% higher than the baseline. It can be achieved using either ordinal regression with A_F or classification with A_T or A_F. On average, the best results are obtained with classification on ordered targets using A_F, which has a performance 15% higher than the baseline's.

4.3.2.1 Base-level evaluation

Figure 4.3 shows the improvement of the NNs configured with the parameters recommended by metalearning.
In this case, the best result is achieved with regression on the original targets using A_F (average MSE=0.188). The NNs trained with these recommended parameters obtain a 67% improvement.

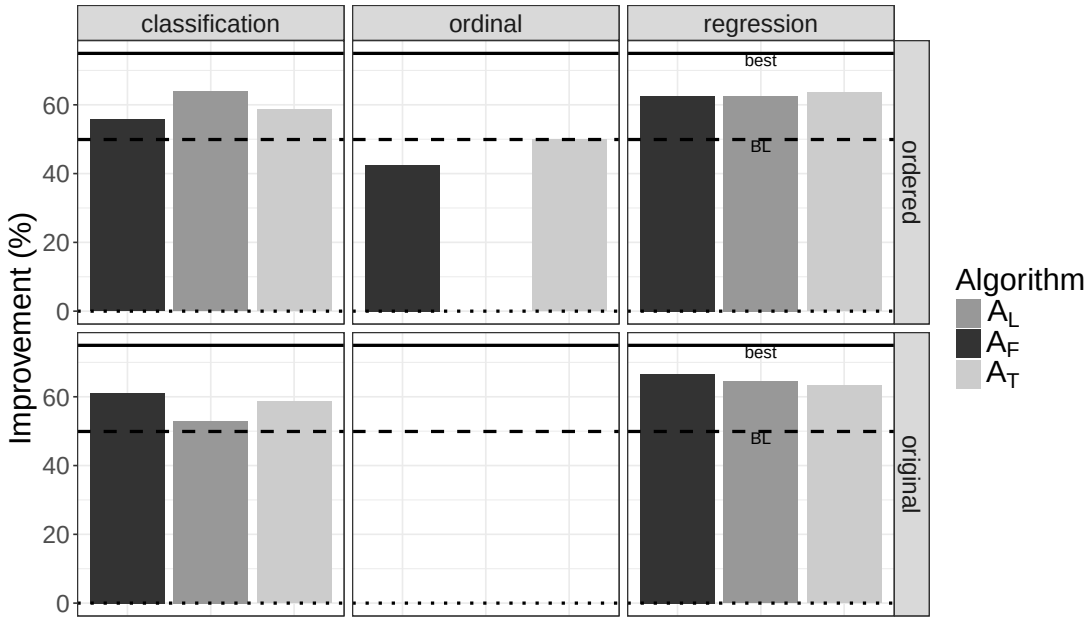


Figure 4.3: Base-level evaluation of the experiments with MF^1_{PS} metafeatures, in terms of improvement.

Also, the neural networks are 14% more accurate than the ones suggested by the classification metamodel referred before. This suggests that for this problem, as expected, regression is more suited than classification.

4.3.3 Classification and Single-target Regression Approaches with MF^2_{PS} metafeatures

To increase the performance of our framework, we created a new metadataset (MF^2_{PS} – explained in Subsection 4.1.2) and conducted experiments using the same approaches described above. Table 4.7 shows the results of the classification approach in terms of accuracy.

The best results for all the parameters are obtained with A_F. For p_n, the best performing approach uses the original target values (with a performance 72% higher than the baseline). For p_d and p_a, the best approach uses the ordered target values (with performance 68% and 124% higher than the baseline, respectively).

On average, the best result is obtained with the ordered targets using A_F. This represents an accuracy 71% higher than the baseline's. Additionally, this is 25% above the average best accuracy obtained with the MF^1_{PS} set of metafeatures.

Table 4.7: Meta-level evaluation of the classification experiments with MF^2_{PS} metafeatures, in terms of accuracy percentage.

	original			ordered			
	A_L	A_F	A_T	A_L	A_F	A_T	BL
p_n	54	**74**	3	49	54	0	43
p_d	43	51	14	40	**57**	14	34
p_a	59	23	6	69	**83**	6	37
avg	52	49	8	53	**65**	7	38

For the regression approaches we measure RRMSE and the results are presented on Table 4.8.

Table 4.8: Meta-level evaluation of the regression experiments with MF^2_{PS} metafeatures, in terms of RRMSE.

	original			ordered			
	A_L	A_F	A_T	A_L	A_F	A_T	BL
p_n	1.128	1.1	2.534	**1.048**	1.407	2.835	1.048
p_d	1.136	**0.105**	0.793	1.212	1.077	1.053	1.203
p_a	0.928	**0.495**	2.18	0.867	1.318	2.556	1.318
avg	1.064	**0.567**	1.836	1.042	1.267	2.148	1.190

The best results for parameter p_n are obtained with the A_L algorithm on ordered target values with the same performance as the baseline model. For the parameters p_d and p_a the best results are achieved with A_F using the original target values, with RRMSE 91% and 62% lower than the baseline, respectively.

On average, the best result is obtained with A_F using the original targets, which represents a 52% increase in performance when compared to the baseline's. In addition to this, the best average RRMSE obtained before with the set of metafeatures MF^1_{PS} was 0.791 and now it is 28% lower.

The results in terms of MZOE are presented on Table 4.9.

The best results are obtained with A_F for classification using ordered target values, improving the baseline MZOE by 19%, 35% and 73% for the problems p_n, p_d and p_a, respectively. The average MZOE is 35%, which represents a performance improvement of 44% over the baseline's. The average best MZOE obtained with MF^1_{PS} metafeatures was 53% and now with MF^2_{PS} it is 34% higher.

Table 4.9: Meta-level evaluation of the ordinal-regression (and classification and regression) using ordered targets experiments with MF^2_{PS} metafeatures, in terms of MZOE.

	ordinal		classification			regression			
	A_L	A_T	A_L	A_F	A_T	A_L	A_F	A_T	B_L
p_n	51	57	57	100	**46**	83	80	94	57
p_d	60	66	66	86	**43**	94	63	63	66
p_a	31	63	51	94	**17**	51	69	71	63
avg	47	62	58	93	**35**	76	71	76	62

4.3.3.1 Base-level evaluation

Figure 4.4 shows the improvement of the NNs configured with the parameters recommended by MF^2_{PS} based metalearning models.

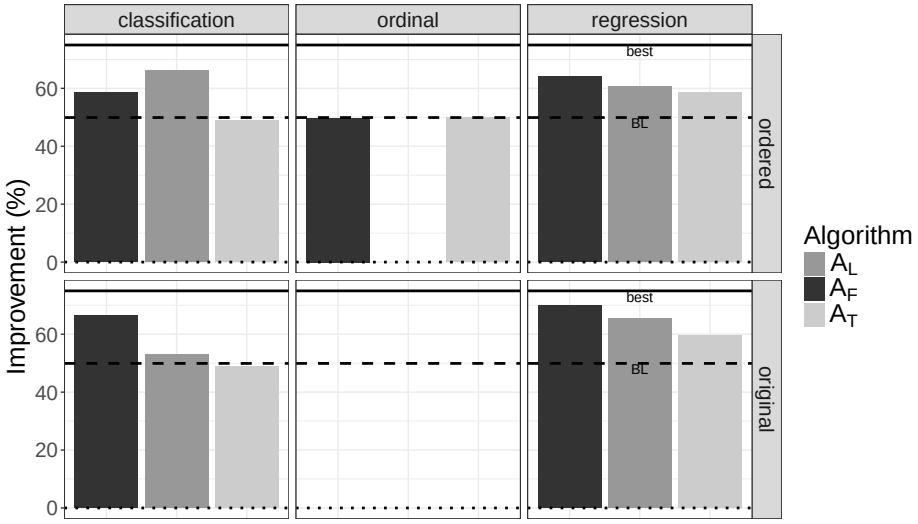


Figure 4.4: Base-level evaluation of the experiments with MF^2_{PS} metafeatures, in terms of improvement.

As with MF^1_{PS}, the best result based on the MF^2_{PS} metafeatures is achieved with regression using the original targets and A_F (average MSE=0.17). Now, the NNs configured with the recommended parameters achieve an improvement of 70%, which is 10% higher than with MF^1_{PS} (Subsection 4.3.2). Meta- and base-level results suggest that, as expected, the new set of metafeatures MF^2_{PS} leads to more accurate results than the ones obtained with MF^1_{PS} metafeatures.

4.3.4 Multi-output Regression Approach with MF^2_{PS} metafeatures

Finally, we drop the strong assumption that the targets (parameters) are not related. Thus, we consider multi-output regression techniques on the MF^2_{PS} metadataset. The meta-level performance results are presented on Table 4.10. Here we see that the best

Table 4.10: Meta-level evaluation of the multi-output regression experiments with MF^2_{PS} metafeatures, in terms of RRMSE.

	p_n			p_d			p_a		
	A_L	A_F	A_T	A_L	A_F	A_T	A_L	A_F	A_T
MTRS	**0.220**	1.106	0.464	0.302	0.160	0.790	1.654	**0.356**	2.180
RC1	1.128	1.096	2.534	1.205	1.243	1.001	0.934	1.194	1.041
RC2	1.128	1.096	2.534	1.076	1.216	1.121	1.017	1.132	0.977
RC3	1.254	1.096	0.843	1.136	**0.099**	0.793	2.347	1.136	1.552
RC4	1.439	1.236	0.843	1.136	**0.099**	0.793	0.879	0.984	1.626
RC5	1.224	0.935	1.111	1.560	1.252	1.304	0.928	0.486	2.180
RC6	2.308	1.045	1.213	1.256	1.270	1.275	0.928	0.486	2.180
baseline	1.048			1.203			1.318		

results are obtained with A_F for p_d and p_a (with increases in performance of 92% and 73% respectively compared to the baseline's), but for p_n, the best algorithm is A_L (with an increase in performance of 79% when compared to the baseline). The average meta-level performance (average relative root mean squared error – $aRRMSE$, as presented in Equation 2.12 in Chapter 2) for each of the multi-output regression methods is presented on Table 4.11.

Table 4.11: Meta-level evaluation of the multi-output regression experiments with MF^2_{PS} metafeatures, in terms of aRRMSE.

	A_L	A_F	A_T
MTRS	0.725	**0.541**	1.145
RC1	1.089	1.177	1.525
RC2	1.074	1.148	1.544
RC3	1.579	0.777	1.063
RC4	1.151	0.773	1.087
RC5	1.237	0.891	1.531
RC6	1.497	0.934	1.556
baseline	1.19		

The best average performance is obtained with A_F and MTRS method, which represents a performance 55% higher than the baseline's and 5% higher than the one obtained by the best single target approach (Table 4.8 RRMSE=0.567).

4.3.4.1 Base-level evaluation

Figure 4.5 shows the improvement of the NNs configured with the parameters recommended by the multi-output metalearning models.

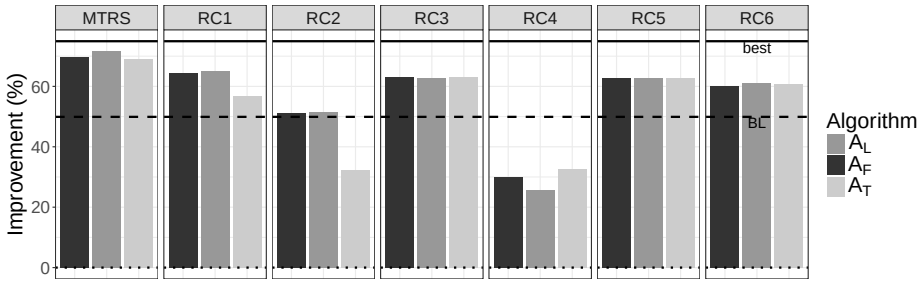


Figure 4.5: Base-level evaluation of the multi-output regression experiments with MF^2_{PS} metafeatures, in terms of improvement.

The average performance of the NNs configured with the parameters recommended by A_F and MTRS is a MSE of 0.16, which represents a 72% improvement compared to the worst performing networks, which is 6% higher than the results obtained by the best single target method. This suggests that the targets are related, and that there is an advantage in predicting them together, when compared to using single target models that discard the relations between the targets.
Additionally, the base-level performance is very close to the best possible value obtained using grid search (only 13% higher). This shows that metalearning, especially multi-target regressor stacking for multi-output regression, can be used to predict a good configuration of parameters for NNs.

4.4 Summary

In this chapter we described a metalearning methodology for combined parameter recommendation in neural network learning. For this, the datasets are characterised through metafeatures that can be used to predict the performance of a wide set of neural network configurations. We propose two different sets of metafeatures: MF^1_{PS} and MF^2_{PS}. The latter is based on MF^1_{PS}, but we have replaced the NN specific landmarkers.

The results were evaluated in terms of meta- and base-level performance. Results indicate that metalearning is a good approach for this problem and that, as expected, the improved set of metafeatures increases the metalearning predictive performance. Also, for this problem, regression techniques perform better than classification and ordered regression does not seem to bring any advantage over the other approaches considered.

The best base-level result was obtained with multi-output regression. This suggests that the target variables are, in fact, related and that there is an advantage in predicting all the parameters simultaneously, instead of considering the problems separately.

The average MSE of the NNs configured with the predicted parameters achieve MSE values only slightly above the ones optimised by grid-search. As a result, the data scientist need not put any effort or time into selecting the NN parameters. Instead, with our method's suggestions, he can obtain a NN with good performance.

Chapter 5

Weights Transfer in Heterogeneous Domain Neural Networks

In this chapter we describe the work performed to answer **RQ3** (Chapter 1): *What is the impact of Transfer Learning (weights transfer) on Neural Networks?* We study the transfer of weights from previously learned neural networks (source) to new ones (target) expecting to obtain faster training neural networks, without harming their performance.
We are using the same datasets considered previously. Because of that, in most cases, the source and target domains have very different natures. This way, we are performing heterogeneous weights transfer.
This requires mapping the source domain's features into the target domain's features. We propose several simple methods for feature mapping in heterogeneous weights transfer, presented in Section 5.1. We also propose a weight transfer process (see Section 5.2) that takes the mapping into account.
We empirically evaluate the proposed methods by analysing the impact of the transfers on the target network's performance. We present the experimental setup used (Section 5.3) and the results obtained (Section 5.4). Finally, we present a summary of our findings (Section 5.5).

5.1 Mapping Process

In our transfer learning scenario, we have a new learning task and the corresponding dataset (the target dataset d_T). Given a previously learned network NN_S from a source dataset, the aim is to (at least partially) reuse NN_S to train a new network NN_T from d_T.

Datasets d_S and d_T may have different natures and different feature sets with different sizes: L and I are the number of features on the source and target datasets, respectively. In our transfer algorithm we will initialise NN_T as having I input nodes, and one output node. The number of hidden nodes depends on the parameterisation used for each target dataset. Weights of NN_T are selected from the weights of NN_S.
Different strategies can be used for weight selection. In our scenario, we start by mapping the features in the target dataset with features in the source dataset.
In this section we propose several simple methods for mapping the features. The objective of the mapping process is to find the most adequate source feature for each target feature. This way, a generic mapping method is represented by the generic mapping function:

$$M_{x_i} = x_l \tag{5.1}$$

that assigns to each target feature i the most adequate source feature l.
The considered datasets only contain numerical features and thus, relationships between the domains can be found in many ways. One possible method of finding similarities between the domains is applying statistical methods to them.
In this case we are considering two different main approaches: KL mapping – using the similarities between source and target features distributions (described in Subsection 5.1.1), and Correlation mapping – using the relatedness of the datasets independent and dependent variables (see Subsection 5.1.2).
Besides the mapping methods explained next, we also use a baseline mapping that consists in randomly selecting a source feature to be mapped to each target feature:

$$M^R_{x_i} = random(x_1, \ldots, x_L), \forall i \in \{1...I\} \tag{5.2}$$

5.1.1 Kulback Leibler Mapping

In the first approach we evaluate the influence of the source and target features distributions on the transfer results. In this case, the transfer is performed considering the similarity between the features' distribution.

For each target feature, we select the source feature with closer distribution. For this, we obtain the Kulback Leibler (KL) divergence of each pair of source/target features. Since we wish to perform the transfer between the features with the most similar distributions, each target feature will be mapped with the source feature which presents the lower KL-divergence. In this case, the generic mapping (Equation 5.1) is instantiated with:

$$M^{KL}_{x_i} = argmin_{x_l}\{KL(x_i, x_l)\}, \forall i \in \{1...I\}, l \in \{1...L\} \tag{5.3}$$

For example, let us consider we wish to perform a KL-mapped transfer of weights between the neural networks corresponding to the source and target datasets presented on Table 5.1.

Table 5.1: Source and Target datasets.

(a) Source dataset

x_1	x_2	x_3	x_4	y_S
10.2	37.7	77.3	3.5	14.9
33.3	0.9	23.8	62.6	87.1
59.1	12.6	8.2	44.7	93.7
15.4	93.8	75.8	45.0	12.3
25.9	60.5	61.1	65.8	48.5
87.4	51.9	59.8	75.8	75.6
88.9	91.4	92.4	57.2	8.9
64.9	93.6	93.0	12.8	24.2
89.9	36.2	27.8	99.3	59.3
8.9	87.5	35.6	11.8	4.5

(b) Target dataset

x_1	x_2	x_3	y_T
30.8	74.6	95.7	9.6
52.9	53.7	7.2	30.8
45.7	45.1	40.1	24.9
17.0	94.9	62.0	26.4
58.6	48.6	58.1	84.4
67.9	9.2	35.0	71.2
0.9	14.3	9.4	20.7

The source dataset is composed of four independent variables (x_1 to x_4) and the dependent variable y_S. The target dataset is composed of three independent variables (x_1 to x_3) and y_T as dependent variable.
First, we need to obtain the KL-divergence between all the source and target datasets' features, as shown on Table 5.2.

Table 5.2: KL-divergences obtained for the source and target datasets presented on Table 5.1. i and l represent the indexes of the target and source features, respectively.

i	1	1	1	1	2	2	2	2	3	3	3	3
l	1	2	3	4	1	2	3	4	1	2	3	4
KL	0.063	0.023	**0.008**	0.070	0.039	0.098	0.065	**0.013**	0.060	0.167	0.117	**0.028**

Then, for each target feature, we choose the most adequate source feature. In the case of KL mapping, following Equation 5.3, for each target feature we select the source features presenting the lower KL-divergence (represented in bold in the table). Given this, the mapping will be:

$$\{M_{x_1}^{KL}, M_{x_2}^{KL}, M_{x_3}^{KL}\} = \{x_3, x_4, x_4\} \tag{5.4}$$

5.1.2 Correlation Mapping

In the second approach we assess the relatedness of the datasets' independent and dependent variables. The relatedness of two variables is obtained by their correlation.

We obtain the relatedness of every source and target feature separately. Then, for each target feature, we will select the source feature with the most similar relatedness. In this case, the generic mapping (Equation 5.1) is instantiated with:

$$M_{x_i}^{*Cor} = argmin_{x_l}\{|corr(x_i, y_T) - corr(x_l, y_S)|\}, \forall i \in \{1...I\}, l \in \{1...L\} \quad (5.5)$$

The correlation mapping is performed in three different ways: using Kendall (M^{KCor}), Pearson (M^{PCor}) and Spearman (M^{SCor}) correlations.
For example, considering the source and target datasets referred on Table 5.1, first we obtain the correlations of each dataset's independent variable and the corresponding dependent variable as shown on Table 5.3 for Spearman correlation.

Table 5.3: Source and Target datasets' Spearman correlations.

(a) Source dataset

l	1	2	3	4
cor	0.382	-0.758	-0.636	0.467

(b) Target dataset

i	1	2	3
cor	0.786	-0.214	-0.250

Then we obtain the absolute difference between the correlations (Table 5.4).

Table 5.4: Absolute differences in correlations.

i	1	1	1	1	2	2	2	2	3	3	3	3
l	1	2	3	4	1	2	3	4	1	2	3	4
Dif.Cor	0.404	1.543	1.422	**0.319**	0.596	0.543	**0.422**	0.681	0.632	0.508	**0.386**	0.717

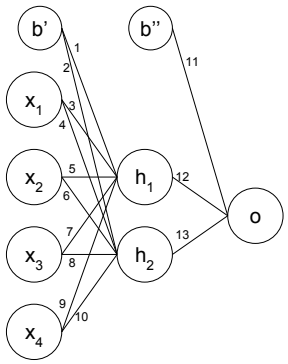
Finally, similarly to what happens for KL mapping, for each target dataset's independent variable, we choose the source dataset's independent variable with lower difference (values represented in bold). In this case the mapping will be:

$$\{M_{x_1}^{SCor}, M_{x_2}^{SCor}, M_{x_3}^{SCor}\} = \{x_4, x_3, x_3\} \quad (5.6)$$

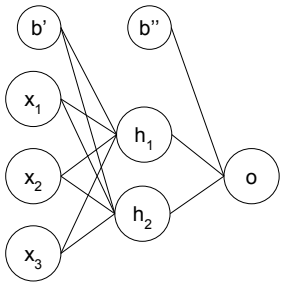
5.2 Weights transfer method

The weights are transferred from a source neural network (NN_S) to a target neural network (NN_T) with the same number of hidden layers and units. Because of this, the weights of the connections with origin on the bias and the hidden units are directly transferred between the networks.

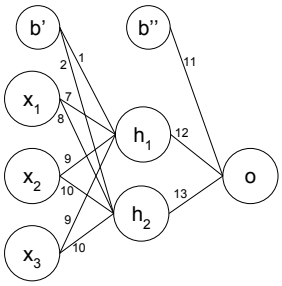
The rest of the weights (relative to the connections between the input and hidden layers) are transferred in a different way. The weights of the connections ending in each of the hidden units of the source network are transferred to the same hidden unit on the target network, and the origin of each connection is chosen according to the mapping referred on the previous section.
For example, let us assume we want to transfer the weights between the source (5.1a) and target (5.1b) neural networks depicted in Figure 5.1 corresponding to the source and target datasets referred on Table 5.1.



(a) Source Network



(b) Target Network



(c) Target Network after transfer.

Figure 5.1: Example Neural Networks for transfer.

Both neural networks have two nodes on the hidden layer and one on the output layer and the respective bias (b' and b") nodes. The source network has four nodes on the input layer (corresponding to the independent variables x_1 to x_4), while the target network has three (x_1 to x_3). The numbers in the source network in Figure 5.1a correspond to the best set of weights obtained for the network:
$W_S = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]$.

Also, let us assume that we are using KL mapping method, that determines that the mapping is obtained from Equation 5.4:

$$\{M_{x_1}^{KL}, M_{x_2}^{KL}, M_{x_3}^{KL}\} = \{x_3, x_4, x_4\}$$

This means that the source dataset's feature x_3 is the most appropriate for transferring for target dataset's feature x_1, and source dataset's feature x_4 for both target features x_2 and x_3.
With this, we will transfer all the weights of the connections originating in the second input unit of the source network to the connections originating in the first input unit of the target network. We will proceed the same way for the rest of the variables. As referred, the weights of the connections with origin on the bias and the hidden nodes are transferred directly.
At the end of the transfer process the set of weights that will be used to initialise the target neural network will be: $W_{T \leftarrow S} = [1, 2, 7, 8, 9, 10, 9, 10, 11, 12, 13]$, as depicted in Figure 5.1c.

5.2.1 Transfer Learning algorithm

The transfer of weights between each pair of datasets is performed according to Algorithm 1.

Input: d_S, d_T, NN_S
Output: $NN_{T \leftarrow S}$
Map $vars(d_S)$ into $vars(d_T)$
foreach $x_i \in vars(d_T)$ **do**
    $w_{x_i h_n} = w_{M_{x_i}^F h_n}$
end
foreach *hidden node* h *of* NN_T **do**
    $w_{h_n y_T} = w_{h_n y_S}$
end
foreach *bias node* b *of* NN_T **do**
    $w_{b'_T h_n} = w_{b'_S h_n}$
    $w_{b''_T y_T} = w_{b''_S y_S}$
end
Algorithm 1: Transfer algorithm.

The input d_S corresponds to the source dataset that is composed by L independent variables $x_l : l \in \{1, \ldots, L\}$ and the dependent variable y_S. d_T is the target dataset

and consists of $x_i : i \in \{1, \ldots, I\}$ as independent and y_T as dependent variables. NN_S is the neural network learned from the source dataset.
The first step of the algorithm consists in mapping the variables from the source to the ones on the target datasets (line 1). For each target dataset's variable x_i we find the most appropriate source dataset's variable x_l. The mapping process is conducted as explained in Section 5.1, uses the entire dataset and is performed independently of the weights transfer step.
The transfer step consists in transferring the variables weights according to the mapping (lines 2-4), then the hidden nodes' weights (lines 5-7) and, finally, the bias nodes' weights (lines 8-11). The output of the algorithm is the set of weights transferred from the source, to be used to initialise the target neural network ($NN_{T \leftarrow S}$).

5.3 Experimental Setup

Our hypothesis is that *a neural network model can converge faster (or more accurately) on a target dataset if, instead of randomly generated, the initial weights are transferred from a source network trained previously.* We test the proposed transfer method on all source/target combinations (the source is always different from the target).
We assess the results of our methods against the usual random weight initialisation method, and also the baseline random mapping referred. If the transfer of weights shortens the network's convergence time or improves its predictive performance, then we have evidence to support our hypothesis.
Figure 5.2 shows the schema of the phases considered in our research, first presented in Chapter 1 (Figure 1.2). The shaded area of the figure represents the experimental setup considered for this study.
As depicted in the figure, in the first phase of this research project, we trained several neural networks for each of the datasets and saved the results (in P_R). Now, to perform the transfers, we take each pair of source/target datasets (the source is always different from the target) and perform the mapping process. Then, for each target dataset the weights transfer is performed according to the parameterisation selected by metalearning (p^{meta}, described in Chapter 4), but also the best parameterisation found (p^{grid}).
Each target neural network is trained starting with the initial set of weights transferred from the source neural network. Besides the parameterisation, performance values (MSE and duration) and best set of weights found, we now also save the source dataset and the mapping method used for the transfer (in P_T).
The experiment was performed using a subset of 28 of the datasets used in the

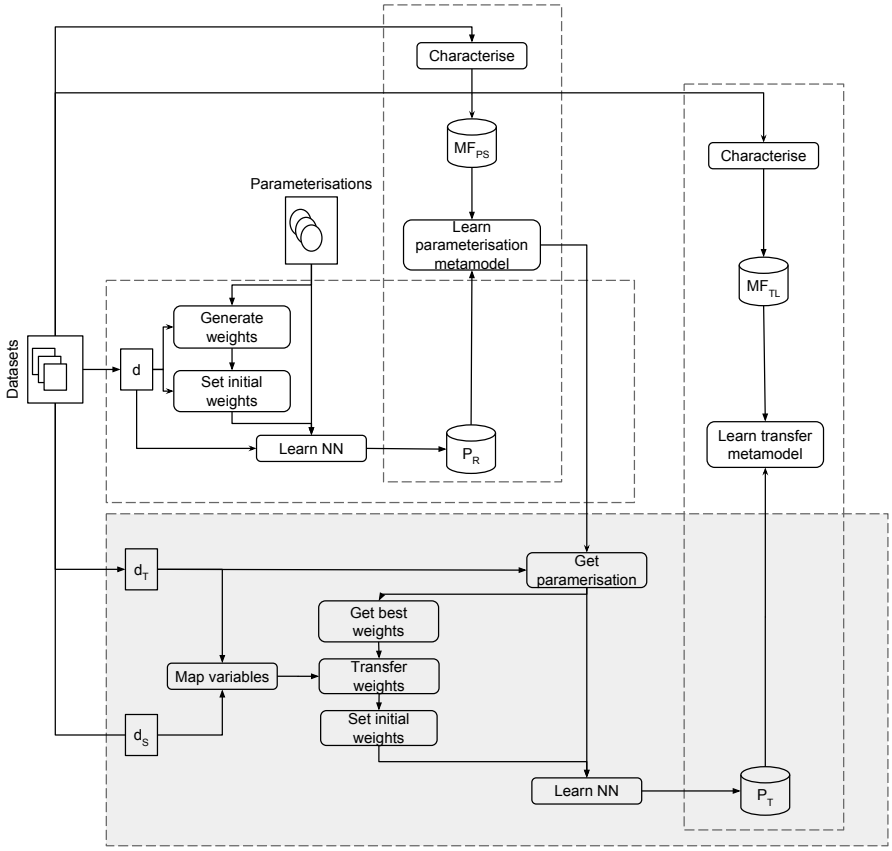


Figure 5.2: Metalearning for multiple domain Transfer learning experimental process. The shaded area corresponds to the experimental setup considered for studying the impact of the transferring weights between neural networks.

remainder of the work (marked in the column "WT" in the Table A.1 in Appendix A), because some datasets require very high computational power (the 7 datasets missing in this case are: 5_4, 5_5, 5_10, 5_11, 5_13, 5_15 and 13).

5.3.1 Performance evaluation

The neural networks evaluation is conducted in the same way as for the first part of our research: it is estimated with 10-fold cross-validation. We generate ten random samples of the dataset and repeat the learning process ten times. Each time (fold), a different sample is considered as testset and the remaining data is used as trainset, where the network learns until convergence.
For each fold, we evaluate the following:

- **MSE0**: the predictive performance in terms of MSE (Equation 2.4) on the testset before training. This allows us to evaluate the network's starting point;

- **duration**: the time needed for convergence;
- **MSE**: the predictive performance in terms of MSE (Equation 2.4) on the testset after training.

At the end of the process, the neural network's performance is the set of averages for each of the metrics obtained for the folds. These values are saved (in the figure, in P_R), together with the parameterisation and the initial and average final weights.
Then, we analyse the transfer learning results for predictive and computational performance using the three metrics ($\mathcal{M}$) referred:

1. $\mathcal{M}$=**MSE0**: If transfer learning reduces this value, it means that the networks initialised with transferred weights have a starting point closer to the expected result;
2. $\mathcal{M}$=**duration**: If transfer learning reduces this value it means that, when initialised with transferred weights, the networks' learning process is faster. This may also be a consequence of a lower MSE0;
3. $\mathcal{M}$=**MSE**: If transfer learning reduces this value, it means that the neural networks initialised with transferred weights are more accurate after training.

For each metric we compare the performance obtained on the randomly initialised neural networks ($\mathcal{M}^R$) and the ones obtained on the transfer initialised neural networks ($\mathcal{M}^T$). With this, we obtain the impact achieved by each transfer:

$$impact = \frac{\mathcal{M}^R - \mathcal{M}^T}{\mathcal{M}^R} \tag{5.7}$$

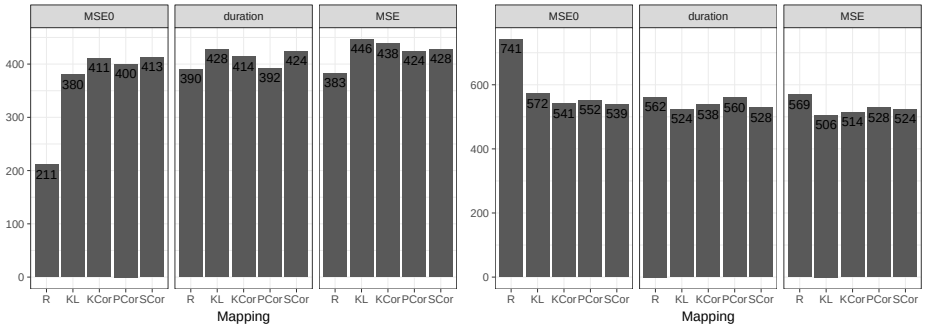
expressed in percentage. Positive *impact* values mean that the transfer of weights brings advantages for the neural network learning process (positive transfer). In the same way, negative impact means that a *negative transfer* has occurred.

5.4 Results

We now analyse the results obtained with transfer learning for both p^{meta} and p^{grid} parameterisations. In each, we analyse the *positive* and *negative transfers* achieved, and also the impact transfer learning has on the neural networks' performance when considering the mapped transfer (transfer of weights considering the proposed mapping methods).

5.4.1 Transfer of weights with p^{meta}

Figure 5.3 shows the number of positive and negative transfers obtained by random transfer and mapped transfers. We can see that random transfer gives origin to

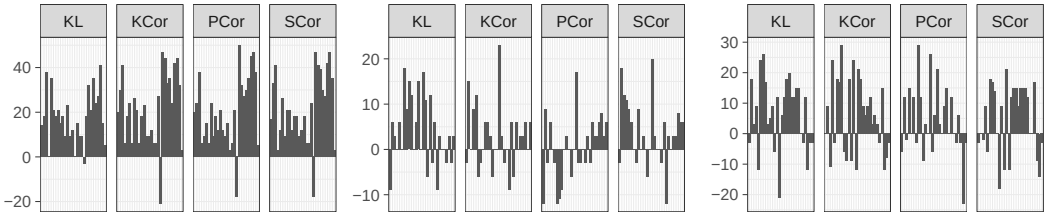


(a) Positive transfers (b) Negative transfers

Figure 5.3: Proportion of positive and negative transfers by method.

poorer results: fewer positive transfers, and more negative transfers. This suggests that, as hypothesised, mapping the datasets' features for the transfer of weights is advantageous.

Next, we analyse the positive transfers obtained for each dataset. In Figure 5.4 we can see, for each evaluation metric, the proportion of positive transfers obtained by transferring with the mapping methods.



(a) MSE0 (b) duration (c) MSE

Figure 5.4: Comparison of proportion of the mapped positive transfers (relative to the random transfer) for each dataset. The xx axis represents the target datasets on the same order as in Table A.1 in Appendix A. The yy axis, for each performance metric, corresponds to the proportion of positive transfers (relative to the proportion of positive transfer obtained by random transfer) when considering each mapping method.

The results are presented for each target dataset, by the same order as in column "WT" of Table A.1 in Appendix A. Positive values mean that mapped transfer leads

to more positive transfers than random transfer. In the figure we can see that, as hypothesised, mapped transfer generally leads to positive transfers more often than random transfer. There are, however, some exceptions, presented on Table 5.5.

Table 5.5: Datasets in which random transfer shows larger proportion of positive transfers for each evaluation metric considered.

(a) duration

d_T	R	KL	KCor	PCor	SCor
*1	62	53	59	50	59
2_2	59	62	59	56	71
5_1	94	94	88	91	100
5_2	38	56	35	26	38
5_3	35	44	35	24	32
5_6	35	50	41	26	44
5_9	50	56	44	50	50
5_12	32	47	32	26	26
5_17	15	26	18	12	18
5_18	91	85	88	88	91
*7	94	91	85	91	91
*11_1	65	56	59	62	53
12_3	15	12	18	21	18
16	12	9	12	15	18

(b) MSE0

d_T	R	KL	KCor	PCor	SCor
*7_1	65	62	44	47	47

(c) MSE

d_T	R	KL	KCor	PCor	SCor
1	97	94	97	91	94
2_2	76	79	65	74	74
4	56	44	53	56	50
5_2	15	41	32	12	32
5_6	62	65	56	74	62
5_7	74	79	65	65	56
5_9	100	94	91	100	88
*5_14	97	76	85	91	85
12_2	79	76	76	76	79
*15	97	85	85	94	88
*16	79	76	71	56	65
*17	94	91	91	91	91

We can see, for each transfer evaluation metric, the datasets for which positive transfer is more often achieved with random transfer. Only for the ones marked with (*) there is not a single mapped transfer with more positive transfers than random transfer. Even for these, the mapped transfers' proportion of positive transfers is not much lower than the random transfer's.

The results still suggest that mapping the source/target datasets' features for the transfer of weights is advantageous. However, the proportion of positive transfers is not enough to assess which mapping method leads to the best transfer results.

Figure 5.5 shows the impact obtained for each performance metric in each dataset (with the same order as the ones marked in the column "WT" on Table A.1 in Appendix A), when considering the different mapping methods.

We can see that the random transfers are outperformed by the mapped transfers. This is supported by Tables G.1a, G.1b and G.1c in Appendix G, that show the best transfer for each dataset, when considering each performance metric: MSE0, duration and MSE, respectively.

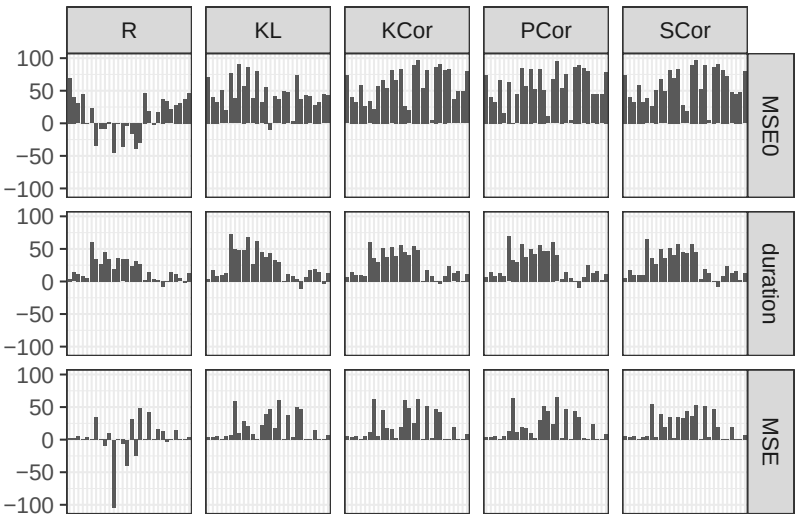


Figure 5.5: Impact of the transfers for each measure and method. The columns represent the different mapping methods and the rows represent the different performance metrics. The xx axis represents the target datasets on the same order as in Table A.1 in Appendix A. The yy axis represents the highest *impact* obtained for each target dataset.

Furthermore, Figure 5.6 shows the frequencies of the mapping methods on the best transfer achieved for each target dataset. The random mapping does not appear on the best transfers.

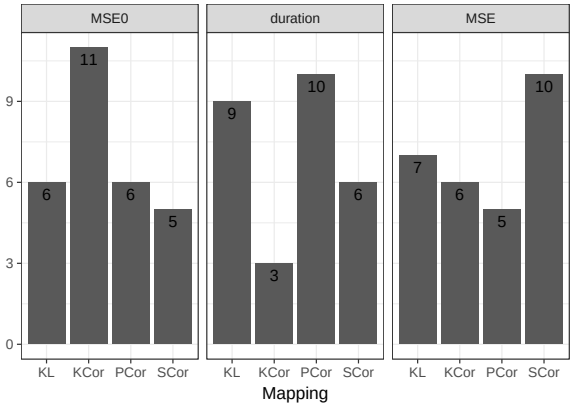


Figure 5.6: Proportion of best transfer by method.

Considering the results of each mapping method separately, we can obtain the impact of the transfers. Table 5.6 shows the average impact of transfers when considering each mapping method for each metric referred before (MSE0, duration, MSE).
On average, *Cor-mapped transfers lead to the higher impacts. From within these,

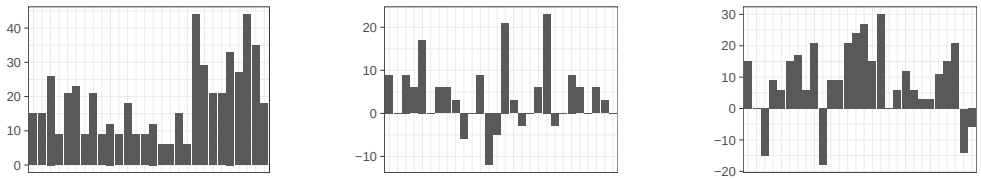
Table 5.6: Average impact for each method.

Mapping	MSE0	duration	MSE	avg
R	9	20	4	11
KL	45	28	12	28
KCor	57	27	12	32
PCor	56	28	12	32
SCor	57	27	13	33

SCor is slightly better for metrics MSE0 and duration, although the other (KCor and PCor are very near). This way, since the impacts are on average very similar, from now on we will consider SCor mapping method only.

5.4.2 Transfer of weights with p^{grid}

The results described so far in this chapter were obtained by parameterising the neural networks with the suggestion made in the previous chapter (p^{meta}) and using the transferred weights. We now analyse the impact of transfer for neural networks parameterised with the best parameterisation found in the grid search (p^{grid}) aiming at studying the impact of transfer learning in the case where the parameterisation suggested is closer to the one obtained by grid-search.
First, we analyse the proportion of positive transfers achieved with SCor-mapped transfers. Figure 5.7 shows the comparison of the proportion of positive transfers obtained when considering SCor mapping method.



(a) MSE0 (b) duration (c) MSE

Figure 5.7: Percentage of positive transfers obtained with SCor-mapped transfer (relative to the random transfer). The xx axis represents the target datasets on the same order as in Table A.1 in Appendix A. The yy axis, for each performance metric, corresponds to the proportion of positive transfers (relative to the proportion of positive transfer obtained by random transfer) when considering SCor mapping method.

Similarly to Figure 5.4, positive values mean that SCor-mapped transfer leads to positive transfers more often than random transfer. We observe that SCor-mapped

transfer generally leads to more positive transfers than random transfer, except for five datasets for the duration metric and four for the MSE metric. These are presented on Table 5.7, where we can see that the differences are not too large.

Table 5.7: Datasets in which random transfer shows larger proportion of positive transfers for each evaluation metric considered.

(a) duration

d_T	R	SCor
5_7	47	41
5_12	56	44
5_14	26	21
5_18	41	38
11	3	0

(b) MSE

d_T	R	SCor
2_2	71	56
5_7	74	56
16	79	65
17	56	50

However, as stated before, the proportion of positive transfers is not enough to assess the best transfer results. Figure 5.8 shows the impact obtained by random transfer and SCor-mapped transfer for each performance metric in each dataset.

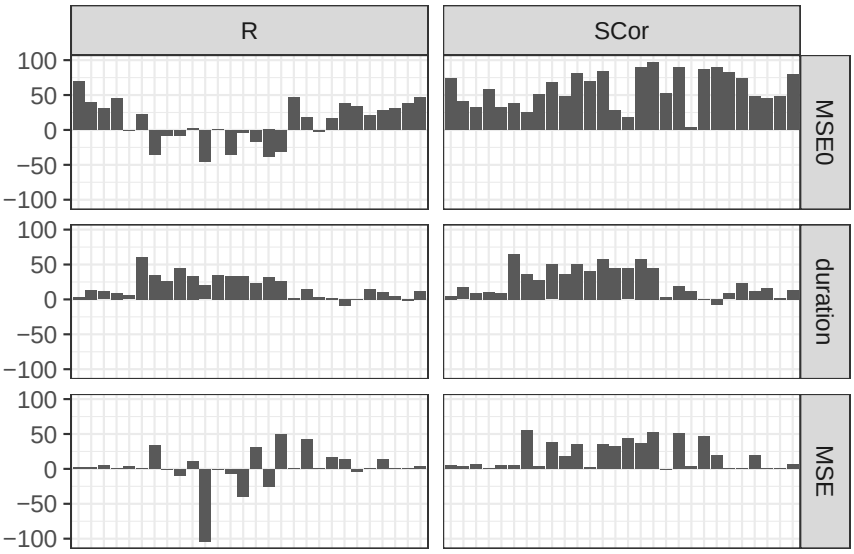


Figure 5.8: Impact of the transfers for each metric with random transfer and SCor-mapped transfer. The xx axis represents the target datasets on the same order as in Table A.1 in Appendix A. The yy axis represents the highest *impact* obtained for each target dataset.

As we can see in the figure, there is only one case in which SCor-mapped transfer leads to negative transfer, while random mapping leads to several negative transfers.

Furthermore, we present the best transfer found for each target dataset with the SCor-mapped transfer for both parameterisations considered: p^{meta} (Table H.1, Appendix H) and p^{grid} (Table I.1, Appendix I).

5.5 Summary

In this chapter we studied the impact of transfer learning on neural networks for regression problems. The experiments were performed with a set of 28 datasets and we tried transfer learning for every possible combination of source/target datasets (the source is always different from the target). The transfers were performed considering two parameterisations: the one suggested by the metalearning method described in the previous chapter; and the best parameterisation found in a grid search. The objective of using this last one is to assess the transfer learning results if the metalearning model suggested a parameterisation that is closer to the best possible (from within the ones tested).

Results indicate that, provided that the source dataset is well chosen, transfer learning can be used to initialise neural networks in order to increase their computational performance, while not harming (and, sometimes, even increasing) their predictive performance.

Also, this is not due to the initial weights having a distribution closer to the optimal. If it was the case, random transfer would lead to similar impacts as mapped transfer and this is not the case. Mapped transfers lead to higher impacts and SCor-mapped transfer revealed to achieve better results and so it was chosen to be used for the rest of the work. In the following chapter we study how metalearning can be used to select a good source dataset for a new target dataset.

Chapter 6

Metalearning for source selection in heterogeneous transfer learning for neural networks

In this chapter we describe the study performed to answer **RQ4** (Chapter 1): *Can metalearning be used to support transfer learning in neural networks?* Our objective is to use metalearning to predict if transferring weights from a specific source network will make the target network converge faster, without harming its performance. This will be performed according to the transfers' characteristics (metafeatures).
We propose seven sets of metafeatures for the selection of the source network for a specific target network (Section 6.1). These metafeatures aim at capturing the transfers' characteristics that can be used to decide whether a specific transfer will be advantageous for a determined target dataset. The metafeatures are then used by the metalearning that tries to map the data characteristics to the impact of a certain transfer.
We perform an extensive experimental setup (Section 6.2) to validate our method and its results are presented in Section 6.3). We present two resources developed that allow data scientists to fully configure neural networks for the `R` package `nnet` (Section 6.4) and finish with a summary of our observations (Section 6.5).

6.1 Metafeatures for source network selection

The purpose of the metafeatures is to characterise the transfers. These characteristics are then used by the metalearning and mapped to the impact of the transfers on the neural networks.

We propose metafeatures for the task of selecting a neural network to be used as source for initialising a specific target network. Our objective is that, by initialising the target network with weights coming from a previously learned source network, the target network's performance will be improved.
We start by generating three separate groups of metafeatures specific for characterising the transfers (described next in Subsections 6.1.1, 6.1.2 and 6.1.3) and then aggregate the groups of metafeatures in seven sets (see Subsection 6.1.4).

6.1.1 Simple metafeatures

The simple metafeatures aim at characterising the similarity between the datasets and are based on groups G1 to G8 referred in Section 4.1. Besides those, for each dataset, we compute the correlations of the independent variables with the dependent variable and use some statistics of the correlations. To illustrate the datasets' similarity, these metafeatures are obtained by the absolute difference between the source and target metafeatures. For example, let us consider the metafeature *n.examples*. If the source dataset contains 300 examples and the target contains 400, the value for this metafeature (d_n.examples) will be 100. This set is composed of 48 metafeatures:

G11	
d_n.bin.fea	d_avg.mean.res.dist.adjacent.target
d_n.h.outlier	d_r.squared
d_n.examples	d_clustering.{3.5.10.20}
d_n.attrs	d_d.tree.leaves
d_n.tri.fea	d_d.tree.mse
d_avg.abs.attr.correlation	d_mean.mse
d_avg.skewness	d_r.num.bin.fea.n.attrs
d_avg.abs.skewness	d_r.num.bin.fea.n.examples
d_avg.kurtosis	d_r.n.h.outlier.n.attrs
d_prop.cor.gt.50	d_r.n.h.outlier.n.examples
d_prop.target.cor.gt.50	d_r.num.tri.fea.n.attrs
d_avg.means	d_r.num.tri.fea.n.examples
d_avg.sds	d_r.n.attrs.n.examples
d_range.target.rel.avg	d_r.n.examples.n.attrs
d_target.coefficient.variation	d_min
d_abs.target.coefficient.variation	d_mean
d_target.cv.sparsity	d_max

Continued on next page

G11 (cont.)	
d_target.abscv.sparsity	d_sd
d_target.h.outlier	d_min_abs
d_target.has.outliers	d_mean_abs
d_target.stationarity	d_max_abs
d_avg.abs.target.correlation	d_sd_abs
d_target.hist.sparsity	

6.1.2 Correlation-based metafeatures

As referred on the previous chapter, the mapping of the variables for transfer is performed by measuring difference in the relatedness of the datasets' independent and dependent variables (using Spearman correlation). By measuring the average minimum and maximum differences, we create the correlation-based metafeatures aiming at characterising the similarity of the source and target datasets' variables. This set is composed of 2 metafeatures:

G12	
avgmin	average of the minimum correlation differences
avgmax	average of the maximum correlation differences

6.1.3 Source selection specific landmarkers

Landmarkers are performance estimators. In this case, the landmarkers consist in running a simpler version of the model on the entire dataset. We initialise the neural networks with the transferred weights and limit its maximum number of iterations to 1, 10 and 100. We measure the *mse0* (the network's initial mean squared error, without training) and the performance indicators: $\mathcal{M} \in \{mse, time, learn, w.sd, w.mean, w.cv\}$, where mse is the network's mean squared error after convergence, $time$ is the amount of time needed for convergence, $learn = mse0 - mse$ and the metrics relative to weights (w) refer to the differences between initial and final weights: $w.sd$ is the standard deviation of the weights differences, $w.mean$ is its mean, and $w.cv$ is its coefficient of variance ($w.cv = \frac{w.sd}{w.mean}$).
This set contains 19 metafeatures and aims at describing the behaviour of each particular transfer.

G13	
lm.transfer.mse0	neural network's MSE0
lm.transfer_M_IT	the metric $\mathcal{M}$ (M) of a neural network that run for a maximum of IT iterations, $IT \in \{1.10.100\}$

6.1.4 Sets of metafeatures

The sets of metafeatures considered characterise the differences between the source and target datasets, their variables, and also include estimators of the transfers performance. However, combining the sets of metafeatures may be advantageous.
Because of this, we generated seven combinations of metafeatures to evaluate which of the characteristics are more informative for predicting the impact of each transfer on the target network:

MF_T metafeatures
$MF_T^1 = G11$
$MF_T^2 = G12$
$MF_T^3 = G13$
$MF_T^4 = G11 + G12$
$MF_T^5 = G11 + G13$
$MF_T^6 = G12 + G13$
$MF_T^7 = G11 + G12 + G13$

6.2 Experimental Setup

Our hypothesis is that *metalearning can be used to determine if a specific transfer will make the target network converge faster, without harming its performance.* We try to predict the impact of the transfers and recommend the best one (the one with highest predicted impact).
We evaluate the metamodels' accuracy (meta-level evaluation) and the impact of the recommended transfers (base-level evaluation). For this part of the research we are only considering the datasets marked in column "SST" on Table A.1 (Appendix A). If the recommended transfers have positive impact on the NNs' performance, then we have evidence to support our hypothesis.
Figure 6.1 shows the schema of the phases considered in our research, first presented in Chapter 1 (Figure 1.2). The shaded area of the figure represents the experimental setup considered for this study.

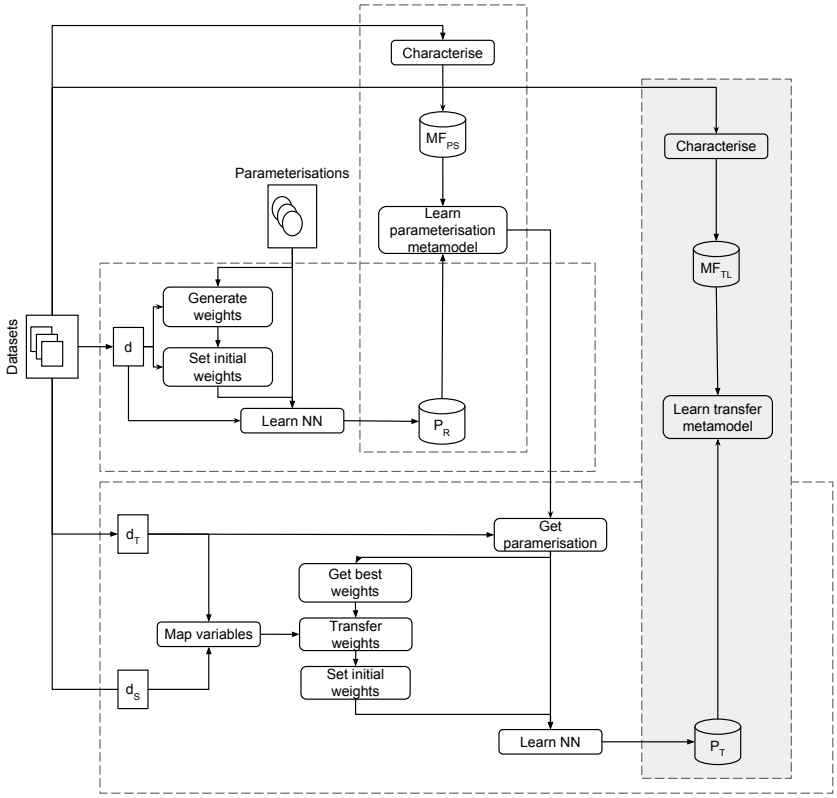


Figure 6.1: Metalearning for multiple domain Transfer learning experimental process. The shaded area corresponds to the experimental setup considered for learning the transfer metamodel.

The first step is to characterise the datasets, obtaining their metafeatures (MF_T, detailed in Section 6.1). Then we use the impact values of SCor-mapped transfers presented in Chapter 5 to build the metamodel (for p^{meta} and also p^{grid}).
We aim at predicting the source dataset that leads to a higher positive impact for each target dataset. Since the values are continuous, we consider the regression approach and the algorithms referred in 2.1.2. We perform the predictions by using the original impact values, but also a scaled transformation of them.
We also use the correlation-based feature selection (CFS) method to remove features with high mutual correlations. We consider three CFS settings: no CFS (cutoff=$\emptyset$), CFS with cutoff=0.75 and CFS with cutoff=0.5. The final number of attributes obtained for each metadataset with each of the cutoff values is presented on Table 6.5.

In the case of the set of metafeatures MF_T^2, as there are only two metafeatures, CFS removes both of them. This is represented with NA in the table.
First, we try to predict positive and negative transfers. For this we use the values pre-

Table 6.5: Number of meta-attributes used after CFS.

Cutoff	MF_T^1	MF_T^2	MF_T^3	MF_T^4	MF_T^5	MF_T^6	MF_T^7
$\emptyset$	48	2	19	50	67	21	69
0.75	19	NA	11	21	30	13	32
0.5	11	NA	8	12	20	10	20

dicted by regression and consider negative prediction values as predictions of negative transfers and positive prediction values as predictions of positive transfers.
In a second step, we try to predict the best (or a good) transfer for each target dataset. For this, we use the same prediction values and consider that the predicted best transfer is the one with highest predicted value. If the highest value is still negative, we consider that the model is predicting that the best possible outcome will be a negative transfer (the user will have better results with randomly initialised neural networks).

6.2.1 Performance evaluation

The metalearning results evaluation is based in leave-one-out cross validation, more specifically "leave-one-target-out" cross validation. Our metadataset contains 34 examples for each of the 28 target datasets considered (for each target, every dataset except itself is considered as source). The metalearning process is repeated 28 times, each considering the examples relative to a specific target dataset as testset and the remaining data as trainset.
Furthermore, the evaluation is performed in two levels:

- meta-level: evaluate the predictive performance of the metalearning models used. We use three different metrics:
  - **MM1:** evaluates if the model correctly identifies negative and positive transfers (i.e., positive and negative impacts);
  - **MM2:** evaluates if the model correctly identifies the best possible transfer (i.e., the highest impact);
  - **MM3:** evaluates if the model correctly identifies a good transfer (i.e., a positive transfer).
- base-level: evaluate the true impact of following the suggestions made by the metalearning (i.e., the true impact of performing the suggested transfer).

For comparison, we consider two baseline models:

- BL_1: the model that predicts the average impact value of the trainset for each instance of the testset. This baseline is used when evaluating MM1;
- BL_2: the model that predicts that the best transfer for each target dataset is the one that considers the source dataset with higher proportion of positive transfers. Here we have $BL_2^{duration}$ and BL_2^{MSE} because we have different results for each of the metatargets considered.

6.3 Results

Since the results obtained with different CFS settings are similar, we only present here the results of the experiments performed without considering CFS. The same way, the scaling of the metatargets does not change the results much. This way, we only present the results of using the original values of the metatarget. For completion, the tables in Appendixes E and F show the complete results. In the following tables, the highest performance is represented in bold.

6.3.1 Metalearning results for p^{meta} parameterisation

We start by applying metalearning to the results obtained for parameterisation p^{meta} and analysing MM1: the accuracy in predicting positive and negative transfers. Table 6.6 shows these results for the two metatargets.

Table 6.6: MM1 accuracies (percentage) on p^{meta} neural networks.

	duration			MSE		
	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	62	63	**64**	**55**	**55**	**55**
MF_T^2	53	55	57	52	53	**55**
MF_T^3	57	51	57	**55**	53	**55**
MF_T^4	**64**	62	**64**	54	**55**	**55**
MF_T^5	61	60	61	54	50	**55**
MF_T^6	58	52	53	**55**	48	**55**
MF_T^7	61	59	62	54	47	**55**
BL_1		55			55	

For the metatarget duration, the highest accuracy is 16% higher than the baseline. As for metatarget MSE, the highest accuracy is the same as the one obtained by the

baseline model BL_1, suggesting that this is a more difficult problem, which may be related to the iterative nature of the neural networks' training process.
The algorithms A_F and A_T obtain good results for both metatargets. As for the set of metafeatures used, MF_T^1 and MF_T^4 can both be used to obtain the highest accuracies. Both sets contain the simple metafeatures. This suggests that the similarity between the source and target datasets can be used to predict if a certain transfer will improve the neural network's performance.
We then evaluate MM2: the metalearning ability to predict the best transfer possible for each target dataset. The baselines BL_2 used here show the effect of choosing the source dataset with more hits for each metric. In this case 5_18 for duration and 5_8 for MSE. These results are shown on Table 6.7. Here we are only considering A_L and A_F, because the models generated by A_T were not able to predict a single maximum.

Table 6.7: MM2 accuracies (percentage) on p^{meta} neural networks.

	duration		MSE	
	A_F	A_L	A_F	A_L
MF_T^1	25	14	0	4
MF_T^2	4	11	4	4
MF_T^3	4	0	0	7
MF_T^4	**32**	18	4	0
MF_T^5	18	11	4	0
MF_T^6	7	4	0	**7**
MF_T^7	29	11	0	0
BL_2	14		14	

The accuracies presented on the table show that, for the metatarget duration, the highest accuracy (32) is 1.3 times higher than the baseline's and was achieved with A_F algorithm and the set of metafeatures MF_T^4. This suggests that the metafeatures characterising the similarity between source and target datasets and between their variables are the ones with more information about the impact of the transfers.
As for metatarget MSE, the results show that it is difficult to predict which is the best possible transfer for a given target dataset to improve its predictive performance. The best performing models have accuracies 50% lower than the ones obtained by the baseline model BL_2. The sets of metafeatures used in the best performing models are MF_T^3 and MF_T^6. Both these sets include the landmarkers, which suggests that these are the best possible estimators of the neural networks' performance.
The last meta-level metric to be analysed is MM3: the ability of the models to predict a good (positive) transfer. These results are presented on Table 6.8.

Table 6.8: MM3 accuracies (percentage) on p^{meta} neural networks.

	duration		MSE	
	A_F	A_L	A_F	A_L
MF_T^1	75	57	57	54
MF_T^2	46	68	39	36
MF_T^3	54	29	57	46
MF_T^4	75	57	57	57
MF_T^5	71	61	39	54
MF_T^6	57	46	54	46
MF_T^7	**82**	61	46	**64**
BL_2	64		46	

MF_T^7 is the set of metafeatures that originates the highest MM3 accuracies for both metatargets. This set of metafeatures is composed by all the three groups of metafeatures considered. This suggests that, to predict if a certain transfer will improve a neural network's performance, besides the similarity of the source and target datasets and their variables, we also need to consider the transfer performance estimators (landmarkers).

Finally, we perform the base-level evaluation to assess the impact of the metalearning predictions on the neural networks' performance. We are considering only the models with highest MM3 accuracies, because these are the ones that predict a higher number of positive transfers. The impact of the models is presented on Table 6.9.

There is one case for metatarget MSE in which the models suggest that no transfer should be performed (marked with NA on D_S field). In this case the impact is 0, because the model's suggestion is not to transfer.

There are several cases for each metatarget in which the transfer leads to a positive impact in both evaluation components: 16 for duration and 9 for MSE. As for negative transfers for both evaluation components, there are three for each metatarget. The number of positive transfers is usually higher than the number of negative transfers, especially for the duration metatarget.

The results also show that datasets from the same group are normally paired for transfer. This happens for datasets in group 5_* for both the metadatasets, 11_* and 12_* for metadataset duration. Another example is when the same dataset is chosen as source to transfer to datasets of the same group, as is the case of datasets 11_* with metatarget MSE. This suggests that the metalearning was able to find characteristics in the datasets that correctly capture the similarities between datasets.

Our main objective is to provide the user with faster neural networks, without harming the predictive performance. When considering metatarget duration (i.e., trying to

Table 6.9: True Impact of the metalearning predictions on the neural networks parameterised with p^{meta}.

(a) duration

ds	dt	duration	MSE
12_2	1	1	1
4	2_1	9	1
4	2_2	8	4
12_2	3	-13	0
11	4	8	0
5_9	5_1	45	-14
5_18	5_2	25	34
5_13	5_3	15	-2
5_18	5_6	42	-7
5_9	5_7	21	9
5_18	5_8	53	36
5_11	5_9	21	2
5_8	5_12	52	-5
5_17	5_14	36	16
5_14	5_16	-4	7
5_18	5_17	53	37
5_8	5_18	46	53
3	6	-6	-15
12_2	7	7	1
3	10	-4	-5
11_2	11	15	5
11	11_2	-9	-29
12_2	12_1	19	-3
12_3	12_2	36	1
12_2	12_3	10	-6
4	15	11	0
3	16	1	0
12_2	17	20	-1
Average		19	4
POS		23	14
NEG		5	10

(b) MSE

ds	dt	duration	MSE
12_1	1	10	2
17	2_1	-10	-1
13	2_2	-2	5
7	3	-8	0
7	4	8	1
5_2	5_1	19	-4
5_5	5_2	15	10
5_2	5_3	-18	-12
12_1	5_6	-3	3
5_16	5_7	-11	17
5_7	5_8	17	22
NA	5_9	0	0
5_8	5_12	52	-5
5_7	5_14	35	10
5_5	5_16	-11	4
5_1	5_17	-12	-195
5_4	5_18	6	-82
12_3	6	-3	3
4	7	9	1
5_13	10	-19	-24
13	11	-6	2
13	11_2	-17	-13
4	12_1	-5	-9
4	12_2	12	1
6	12_3	-3	-2
12_1	15	1	1
5_5	16	-31	0
13	17	1	2
Average		1	-9
POS		12	15
NEG		15	10

predict the impact of the transfers in terms of time needed for the neural networks to converge) we obtain neural networks that are, in average, 19% faster, while 4% more accurate.

Also, when considering the MSE metatarget, we can still recommend faster convergence neural networks, while losing only 10% in predictive performance. This suggests that metalearning can be used to select a good source dataset for a given target dataset.

6.3.2 Metalearning results for p^{grid} parameterisation

Next, we analyse the results obtained in a situation where the choice of parameters is optimal (neural networks parameterised with p^{grid}). Besides analysing the results, we will also compare them to the ones shown for p^{meta}

We start by analysing the results in terms of MM1: the models' ability to predict if a specific transfer will have positive or negative impact on the performance. The results are presented on Table 6.10. Concerning the algorithms, A_F is the best for

Table 6.10: MM1 accuracies (percentage) for p^{grid} neural networks.

	duration			MSE		
	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	68	73	74	64	61	64
MF_T^2	68	64	70	62	63	64
MF_T^3	69	66	65	65	63	64
MF_T^4	76	73	75	64	63	64
MF_T^5	78	72	74	64	61	64
MF_T^6	72	66	66	**66**	63	64
MF_T^7	**79**	71	75	64	64	64
BL_1		67			59	

both metatargets.

For predicting the impact in terms of duration all the metafeatures considered have useful information. For predicting the impact in terms of MSE the most important information consists in the similarity between the datasets' variables and the estimators of the transfer performance (landmarkers).

Comparing these results with the ones obtained previously, we see that the best performing model is one of the same as before for metatarget MSE, while for duration we now have another model. However, if we look at the values obtained by the best models for the previous analysis we can see that their results are also high (around 75).

Next, we analyse MM2: the models' ability to predict which is the best transfer possible for a specific target dataset. These results are shown on Table 6.11. As

Table 6.11: MM2 accuracies (percentage) for p^{grid} neural networks.

	duration		MSE	
	A_F	A_L	A_F	A_L
MF_T^1	7	**18**	4	4
MF_T^2	7	**18**	4	0
MF_T^3	0	4	0	4
MF_T^4	14	14	4	0
MF_T^5	7	14	**11**	4
MF_T^6	4	4	0	0
MF_T^7	11	14	7	0
BL_2	18		11	

happened in the previous analysis, A_T is not capable of predicting a single maximum, and therefore we ignore it for this analysis. In this case, the best performing algorithm for MSE is still A_F, while for duration it is A_L, although the performance is the same

as the baseline's.

As for the metafeatures, we have different sets for each metatarget, and also the sets are different from the ones obtained on the previous analysis. The results suggest that predicting the best possible transfer for a specific target dataset is a difficult task, even for neural networks parameterised with the best configuration found in the grid search.

Finally, we analyse the results in terms of MM3: the models' ability to predict a good transfer for a specific target dataset. The results are presented on Table 6.12. In this case the best results are obtained with A_F for both metatargets.

Table 6.12: MM3 accuracies (percentage) for p^{grid} neural networks.

	duration		MSE	
	A_F	A_L	A_F	A_L
MF_T^1	43	57	50	43
MF_T^2	57	61	43	32
MF_T^3	39	18	43	32
MF_T^4	57	57	**54**	39
MF_T^5	**64**	61	39	43
MF_T^6	50	57	46	32
MF_T^7	**64**	57	36	43
BL_2	57		36	

As for the metafeatures, for the metatarget duration, the best results can be obtained with MF_T^5 or MF_T^7. Both these sets contain the simple metafeatures and the landmarkers. This suggests that, to predict the impact in terms of duration, we need to consider the similarity between the source and target datasets, but also the estimators of the transfer performance.

When considering the metatarget MSE, the best results are obtained with MF_T^4, which is composed by the simple and correlation-based metafeatures. This suggests that, for predicting the impact in terms of MSE, we need to consider the similarity between the datasets, but also between their variables.

When comparing these results with the ones obtained previously, we see that the best performing model here is the same for duration metatarget, but not for MSE metatarget. However, in this case, the model with highest performance in the previous analysis (using A_L and MF_T^7) still has a performance above the baseline's.

This way, for the base-level evaluation, we will use here the same models as for the p^{meta} neural networks. The impacts of following the predictions on the neural networks' performance are shown on Table 6.13. For each metatarget there are cases in which the model suggests not to transfer. These are marked with NA on the d_S field and

represent an impact of 0 in each evaluation component.

Table 6.13: True Impact of the metalearning predictions on the neural networks parameterised with p^{grid}.

(a) duration

d_S	d_T	duration	MSE
NA	1	0	0
5_4	2_1	5	3
5_6	2_2	3	0
NA	3	0	0
NA	4	0	0
5_5	5_1	65	-18
5_17	5_2	31	22
5_7	5_3	24	0
5_18	5_6	47	7
5_9	5_7	25	9
5_18	5_8	39	-387
5_11	5_9	27	-8
5_6	5_12	39	-39
5_6	5_14	-1	-38
5_5	5_16	11	42
5_18	5_17	58	37
5_17	5_18	45	32
NA	6	0	0
NA	7	0	0
NA	10	0	0
NA	11	0	0
11	11_2	-34	-12
12_2	12_1	7	-9
NA	12_2	0	0
12_2	12_3	-27	-172
NA	15	0	0
NA	16	0	0
NA	17	0	0
Average		13	-19
POS		14	7
NEG		3	8

(b) MSE

d_S	d_T	duration	MSE
5_13	1	-22	-2
17	2_1	-2	2
13	2_2	-2	1
4	3	-30	0
7	4	6	1
5_14	5_1	17	-19
NA	5_2	0	0
4	5_3	-15	-52
NA	5_6	0	0
11_2	5_7	-3	-6
5_15	5_8	-12	-5
5_11	5_9	27	-8
5_8	5_12	58	-5
NA	5_14	0	0
5_5	5_16	11	42
5_5	5_17	7	7
5_12	5_18	22	29
NA	6	0	0
4	7	-15	32
NA	10	0	0
13	11	-25	-16
13	11_2	-43	-25
2_1	12_1	-19	-3
15	12_2	7	0
15	12_3	-60	-154
12_1	15	-3	0
11_2	16	2	0
13	17	-9	0
Average		-4	-6
POS		9	7
NEG		14	11

There are cases in which the suggested transfer leads to positive impacts in both the evaluation components: 9 for duration and 6 for MSE metatargets. As for negative impact in all the components, there are only 3 cases for metatarget duration and 8 for metatarget MSE.

Also, we can see that for each metatarget we have cases in which both impact values are positive: 6 for metatarget MSE and 9 for metatarget duration and negative in 3 cases for metatarget duration and 8 for metatarget MSE. These values are similar to the ones obtained in the previous analysis.

As before, we can see that datasets in group 5_* are normally chosen as source for transfers in which the target is another dataset from the same group. Also, dataset 13 is chosen as source for both datasets in group 11_*, when considering metatarget MSE. This suggests that metalearning was able to detect similarities between the datasets and predicts the best source according to them.

As stated, our objective is to provide the user with a way to have faster neural networks, without harming their predictive performance. In average, the highest impact in the duration is obtained with the model that predicts metatarget duration. We here have neural networks 13% faster, although 19% less accurate. This loss in accuracy may be due to the performance obtained by randomly initialised neural networks already being very high (metric ($\mathcal{M}^R$) very low), making them difficult to outperform. Besides, as the impact is obtained as defined in Equation 5.3.1, if we have $\mathcal{M}^R$ very near zero, the impact will be very high. For example, let us look at the datasets with the highest negative impacts in MSE, presented on Table 6.14. The

Table 6.14: Highest negative impacts of the transfers.

D_S	D_T	$\mathcal{M}^R$	$\mathcal{M}^T$	impact
5_18	5_8	0.000157	0.000767	-387
12_2	12_3	0.008253	0.22473	-172

$\mathcal{M}^R$ in these cases is already very low. Even with low performances in and the impact is very high due to this fact.

This way, even with results harder to outperform, metalearning still seems to be a good approach for selecting the source dataset for a transfer. In average, we will obtain neural networks 13% faster without losing too much predictive performance in the majority of the target datasets.

6.4 Developed resources

For dissemination purposes, and to enable the easy use of our methodologies, we have developed two different ways of using our metamodels to fully configure neural networks: an `R` Shiny application, and an `R` library.

With this, the data scientist can use the metamodels described in this research to predict both parameterisation and initialisation of neural networks, thus reducing the effort and time needed for the task.

Both resources work only with numerical datasets for regression tasks, such as the ones presented in our research and follow the four step methodology depicted in Figure 6.2.

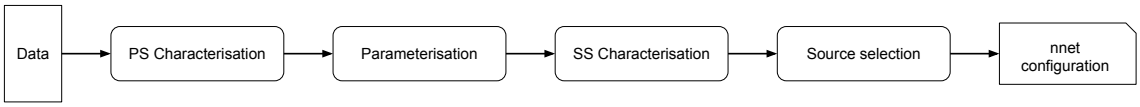


Figure 6.2: Methodology followed on the resources developed.

First, in the *PS Characterisation* step, the dataset is characterised for the parameter selection task. In this step, the set of metafeatures MF^2_{PS} (Section 4.1) is computed. The metafeatures are then used in the *Parameterisation* phase to select a high-performance set of parameter values, according to the best metamodel found for this task (Chapter 4).
In the third step, *SS Characterisation*, the dataset is characterised for the source selection task. The set of metafeatures MF^7_T (Section 6.1) is computed. This set of metafeatures contains the transfer specific landmarkers. The landmarkers estimate the performance of neural networks initialised with transferred weights on the dataset at hand. For this, the mapping process (Section 5.1) is performed for all the source datasets available (the ones used in this research, and presented on Table A.1 in Appendix A).
Finally, these metafeatures are used in the *Source selection* step to select the network from which the weights should be transferred. This is performed according to the best metamodel found for this task (Chapter 6). At the end of the process, the user can obtain the full neural network configuration, including the recommended values for parameters *size*, *decay* and *abstol* and the initial weights to feed to the neural network.

6.4.1 NN configurer Application

We developed an `R` Shiny application, publicly available at (`https://catarinafelix.shinyapps.io/nn_shiny/`) and also available for download at (`https://gitlab.com/catarinafelix/nn_shiny`). Figure 6.3 shows the screen of *NN configurer* GUI during the metafeature computation phase.
After the characterisation, the user can download the metafeatures computed. Then, he can obtain the parameterisation recommended by our metamodel. After this, he can choose to use transferred weights or skip this step and simply use randomly generated weights.
In case the user prefers transferred weights, the transfer specific landmarkers are computed, the source network is selected and, finally, the full neural network configuration is presented. Otherwise, the configuration presented will include the parameter values

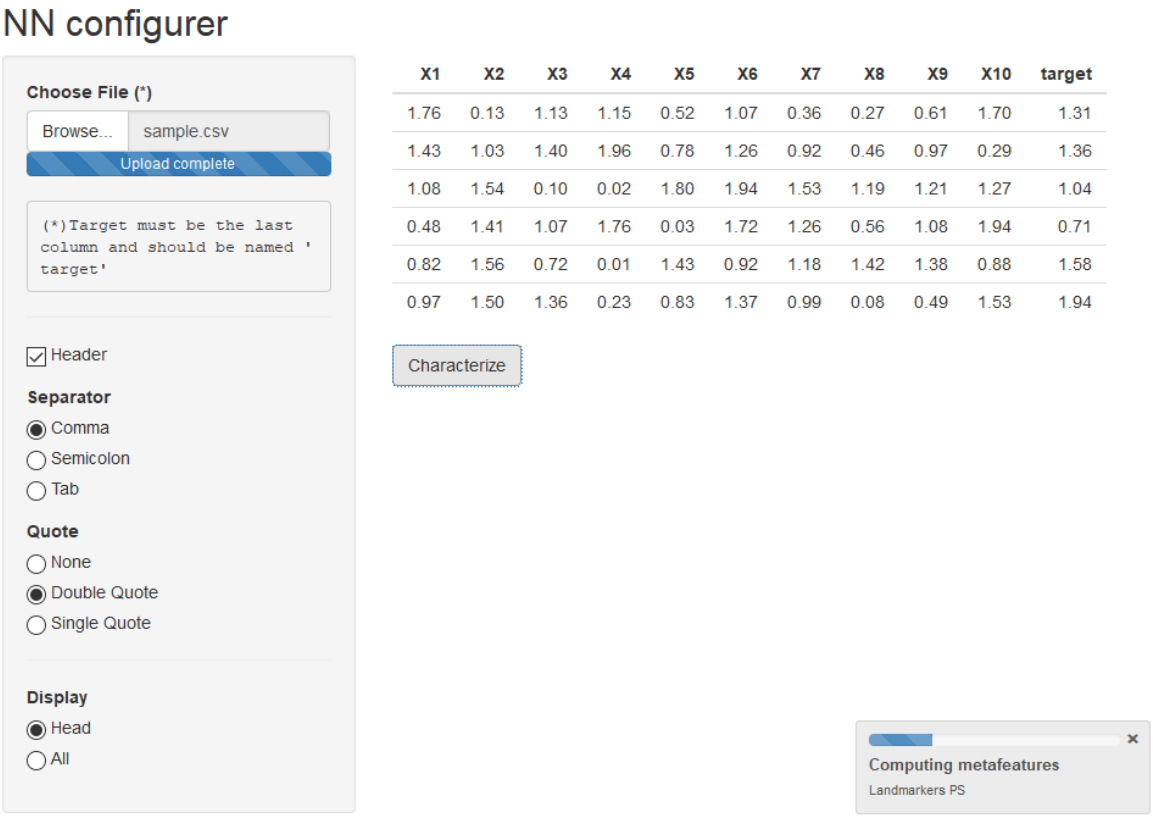


Figure 6.3: *NN configurer* GUI during the metafeature computation phase.

recommended previously, and a set of randomly generated initial weights.

6.4.2 nnetConf library

We have also developed an `R` library, available at https://gitlab.com/catarinafelix/nnetconf. The command to obtain the full neural network configuration for a dataset named "data" is:

```
configure(data)
```

In this case, the library will compute the metafeatures and neural network specific landmarkers and recommend a parameterisation. It will then compute the transfer specific landmarkers and recommend the set of weights to be transferred. At the end of the process, the full neural network configuration is presented to the user.
However, if the user wants to skip the transfer and use randomly generated weights instead, the command to use is:

```
configure(data, notransfer=T)
```

In this case, the configuration presented will include the parameter values recommended and, similarly to what happens in *NN configurer*, a set of randomly generated initial weights.

6.5 Summary

In this chapter, our objective was to create a model that could choose good source networks for a given target network, i.e., when transferring weights from that source neural network to the target neural network, we would have a positive impact on the neural network's performance, avoiding negative transfer.

We conducted experiments considering two different parameterisations. For the neural networks parameterised with p^{meta} (the parameterisation suggested in Chapter 4) we were able to achieve neural networks that were, on average, 19% faster and 4% more accurate than randomly initialised networks. For the neural networks parameterised with p^{grid} (the best parameterisation found by grid search) we obtained neural networks 13% faster than randomly initialised networks, although 19% less accurate than those, in average. This loss in average accuracy may be related to the randomly initialised neural networks already having high performance which, besides being difficult to outperform, make negative impacts be very high.

The results obtained suggest that metalearning can be used to select a source network from which to transfer weights to a given target network. Instead of spending time experimenting with different sets of initial weights, the data scientist can use our metamodel to select the weights to initialise the neural network in order to have a fast training model without losing much predictive performance.

Furthermore, we presented two resources that can be used by any data scientist to fully configure neural networks: a R Shiny application and a R library, both capable of fully configuring neural networks for the nnet package.

Chapter 7

Conclusion

Neural networks are difficult to configure. Both parameterisation and initialisation tasks are difficult and time consuming, also requiring user expertise. Our objective was to provide the data scientist with methods to perform these tasks. For that, we proposed using metalearning together with transfer learning.

We started by trying to answer **RQ1**: **How do different parameter values impact the performance of neural networks?** For this, we studied several parameter configurations and different datasets, aiming at finding a configuration that yields good average performance (Chapter 3). The results obtained show that there are certain parameter values that can make the neural networks learn better or faster. This way, we suggested a group of (full and partial) neural network parameterisations that can lead to good performance. However, the results also indicate that there might be some characteristic of the data that makes a certain parameterisation more suited for a specific dataset.

This way, to try to answer **RQ2**: **Can metalearning be used to support the parameterisation of neural networks?**, we then propose a metalearning methodology to select a high-performance parameterisation for neural networks. This methodology is presented in Chapter 4. We designed different sets of metafeatures to characterise the datasets and used them to predict the parameterisations to be used. Results suggest that metalearning is a good approach for selecting a high-performance parameterisation, especially when using the more extensive set of neural network specific landmarkers. The best result was obtained with multi-output regression, suggesting that the parameter values are, in fact, related and that there is an advantage in predicting all of them simultaneously, instead of considering separate problems. The average performance of the neural networks configured with the predicted parameters are only slightly lower than the ones optimised by grid-search. With this, the data scientist can use our methodology's predictions to obtain a neural network with high

performance.

We then studied the use of transfer learning to help on the neural networks' initialisation task, trying to answer **RQ3**: **What is the impact of transfer learning (weights transfer) on neural networks?**. Our objective was to improve the neural networks' performance by initialising them with weights obtained from previously learned neural networks, instead of using randomly generated initial weights. Besides, we propose methods for mapping the source and target features, in order to perform the transfers between the most adequate ones. We performed several different transfers, considering the different mapping methods and evaluated the impact of the transfer of weights on neural networks (Chapter 5). We compared the performance of randomly initialised networks with the performance of networks initialised with weights transferred from previously learned ones. The experiments were performed using parameterisations p^{meta} (suggested by metalearning) and p^{grid} (found from grid-search). The results obtained suggest that transfer learning can be used to initialise neural networks in order to accelerate their training process with minimal loss in predictive performance. Furthermore, the mapped transfers obtained better results than random transfers, suggesting that the improvement brought by transfer learning is not only related to the initial weights' distribution. The higher transfer impacts revealed that the transfer of weights is a suitable approach for initialising neural networks, provided that the source network is well chosen.

Finally, to answer **RQ4**: **Can metalearning be used to support transfer learning in neural networks?**, we studied the use of metalearning to perform the source network selection for a given target network, preventing negative transfer (i.e., so that the transfer has a positive impact on the neural network's performance). This part of the work is presented in Chapter 6. We designed several sets of metafeatures to characterise the similarity among source and target datasets and their variables, and also landmarkers that estimate the impact of the transfers. The experiment was conducted considering both parameterisations referred above. The results obtained suggest that metalearning can be used to select a source network from which to transfer weights to a given target network in order to have a fast training model without much loss in predictive performance. With this, the data scientist can use our methodology to obtain a faster training neural network with almost no loss in predictive performance.

Ultimately, the data scientist can use our methodology to obtain a high-performance neural network configuration. The methodology is also available as one of the two resources we developed: NN configurer (`https://catarinafelix.shinyapps.io/nn_shiny/` and downloadable at `https://gitlab.com/catarinafelix/nn_shiny`), an `R`

Shiny application; or the `R` library `nnetConf` (https://gitlab.com/catarinafelix/nnetconf) described in Section 6.4.

First, he can use the parameter selection metamodel to parameterise the network. Then, according to the parameterisation selected, he can use the source selection metamodel to select a source network from where the initial weights can be transferred to his new neural network. With this, the data scientist will reduce the time needed for both parameterisation and training of the network, still reaching high predictive performance.

The results can be improved in the future. We aim at creating new neural-network- and transfer-specific landmarkers that more accurately capture the characteristics needed for the metamodels that select the parameterisation and the source network. Furthermore, we can consider both models as a single one, and try to predict both configurations simultaneously, instead of considering them as separate problems. Finally, the models could be applied to different architecture neural networks, possibly including deep models, to improve their performance.

References

Abreu, P., Soares, C., and Valente, J. M. (2009). Selection of heuristics for the job-shop scheduling problem based on the prediction of gaps in machines. In *International Conference on Learning and Intelligent Optimization*, pages 134–147. Springer.

Aiolli, F. (2012). Transfer learning by kernel meta-learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 81–95.

Ali, R., Khatak, A. M., Chow, F., and Lee, S. (2018). A case-based meta-learning and reasoning framework for classifiers selection. In *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication*, page 31. ACM.

Argyriou, A., Maurer, A., and Pontil, M. (2008). An algorithm for transfer learning in a heterogeneous environment. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 71–85. Springer.

Baccianella, S., Esuli, A., and Sebastiani, F. (2009). Evaluation measures for ordinal regression. In *Intelligent Systems Design and Applications, 2009. ISDA'09. Ninth International Conference on*, pages 283–287. IEEE.

Baghoussi, Y. and Mendes-Moreira, J. (2018). Instance-based stacked generalization for transfer learning. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 753–760. Springer.

Bakker, B. and Heskes, T. (2003). Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4(May):83–99.

Ben-David, S. and Schuller, R. (2003). Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines*, pages 567–580. Springer.

Bensusan, H. and Giraud-Carrier, C. (2000). Discovering task neighbourhoods through landmark learning performances. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 325–330. Springer.

Biondi, G. O. and Prati, R. C. (2015). Setting parameters for support vector machines using transfer learning. *Journal of Intelligent & Robotic Systems*, 80(1):295–311.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bishop, C. M. (1994). Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832.

Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Wortman, J. (2008). Learning bounds for domain adaptation. In *Advances in neural information processing systems*, pages 129–136.

Blitzer, J., Dredze, M., and Pereira, F. (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 440–447.

Blitzer, J., McDonald, R., and Pereira, F. (2006). Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 120–128. Association for Computational Linguistics.

Borchani, H., Varando, G., Bielza, C., and Larrañaga, P. (2015). A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(5):216–233.

Brazdil, P., Carrier, C. G., Soares, C., and Vilalta, R. (2008). *Metalearning: Applications to data mining*. Springer Science & Business Media.

Brazdil, P. and Giraud-Carrier, C. (2018). Metalearning and algorithm selection: progress, state of the art and introduction to the 2018 special issue.

Brazdil, P. B., Soares, C., and Da Costa, J. P. (2003). Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277.

Chattopadhyay, R., Sun, Q., Fan, W., Davidson, I., Panchanathan, S., and Ye, J. (2012). Multisource domain adaptation and its application to early detection of fatigue. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(4):18.

Chu, X., Cai, F., Cui, C., Hu, M., Li, L., and Qin, Q. (2019). Adaptive recommendation model using meta-learning for population-based algorithms. *Information Sciences*, 476:192–210.

Cook, D., Feuz, K. D., and Krishnan, N. C. (2013). Transfer learning for activity recognition: A survey. *Knowledge and information systems*, 36(3):537–556.

Dai, W., Xue, G.-R., Yang, Q., and Yu, Y. (2007a). Co-clustering based classification for out-of-domain documents. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 210–219. ACM.

Dai, W., Yang, Q., Xue, G.-R., and Yu, Y. (2007b). Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, pages 193–200. ACM.

Dai, W., Yang, Q., Xue, G.-R., and Yu, Y. (2008). Self-taught clustering. In *Proceedings of the 25th international conference on Machine learning*, pages 200–207. ACM.

Daumé III, H. (2009). Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*.

Davis, J. and Domingos, P. (2009). Deep transfer via second-order markov logic. In *Proceedings of the 26th annual international conference on machine learning*, pages 217–224. ACM.

Day, O. and Khoshgoftaar, T. M. (2017). A survey on heterogeneous transfer learning. *Journal of Big Data*, 4(1):29.

Do, C. B. and Ng, A. Y. (2006). Transfer learning for text classification. In *Advances in Neural Information Processing Systems*, pages 299–306.

Duan, L., Tsang, I. W., and Xu, D. (2012a). Domain transfer multiple kernel learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):465–479.

Duan, L., Xu, D., and Chang, S.-F. (2012b). Exploiting web images for event recognition in consumer videos: A multiple source domain adaptation approach. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1338–1345. IEEE.

Duan, L., Xu, D., and Tsang, I. (2012c). Learning with augmented features for heterogeneous domain adaptation. *arXiv preprint arXiv:1206.4660*.

Eaton, E., Lane, T., et al. (2008). Modeling transfer relationships between learning tasks for improved inductive transfer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 317–332. Springer.

Félix, C., Soares, C., and Jorge, A. (2015). Metalearning for multiple-domain transfer learning. In *MetaSel@PKDD/ECML*, pages 67–79.

Félix, C., Soares, C., and Jorge, A. (2016). Can metalearning be applied to transfer on heterogeneous datasets? In *International Conference on Hybrid Artificial Intelligence Systems*, pages 332–343. Springer.

Félix, C., Soares, C., Jorge, A., and Ferreira, H. (2017). Using metalearning for parameter tuning in neural networks. In *European Congress on Computational Methods in Applied Sciences and Engineering*, pages 1081–1090. Springer.

Feuz, K. D. and Cook, D. J. (2015). Transfer learning across feature-rich heterogeneous feature spaces via feature-space remapping (fsr). *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(1):3.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.

from Jed Wing, M. K. C., Weston, S., Williams, A., Keefer, C., Engelhardt, A., Cooper, T., Mayer, Z., Kenkel, B., the R Core Team, Benesty, M., Lescarbeau, R., Ziem, A., Scrucca, L., Tang, Y., Candan, C., and Hunt., T. (2017). *caret: Classification and Regression Training*. R package version 6.0-76.

Gama, J. and Brazdil, P. (1995). Characterization of classification algorithms. In *Portuguese Conference on Artificial Intelligence*, pages 189–200. Springer.

Gama, J. and Kosina, P. (2011). Learning about the learning process. In *International Symposium on Intelligent Data Analysis*, pages 162–172. Springer.

Gao, J., Fan, W., Jiang, J., and Han, J. (2008). Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 283–291. ACM.

Ge, L., Gao, J., Ngo, H., Li, K., and Zhang, A. (2014). On handling negative transfer and imbalanced distributions in multiple source transfer learning. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(4):254–271.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520.

Gomes, T. A., Prudêncio, R. B., Soares, C., Rossi, A. L., and Carvalho, A. (2012). Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13.

Gong, B., Shi, Y., Sha, F., and Grauman, K. (2012). Geodesic flow kernel for unsupervised domain adaptation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2066–2073. IEEE.

Gutiérrez, P. A., Pérez-Ortiz, M., Sanchez-Monedero, J., Fernández-Navarro, F., and Hervas-Martinez, C. (2016). Ordinal regression methods: survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):127–146.

Gutierrez-Rodríguez, A. E., Conant-Pablos, S. E., Ortiz-Bayliss, J. C., and Terashima-Marín, H. (2019). Selecting meta-heuristics for solving vehicle routing problems with time windows via meta-learning. *Expert Systems with Applications*, 118:470–481.

Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Harel, M. and Mannor, S. (2010). Learning from multiple outlooks. *arXiv preprint arXiv:1005.0027*.

Hothorn, T., Bühlmann, P., Dudoit, S., Molinaro, A., and Van Der Laan, M. J. (2005). Survival ensembles. *Biostatistics*, 7(3):355–373.

Hothorn, T., Hornik, K., and Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3):651–674.

Huang, J., Gretton, A., Borgwardt, K. M., Schölkopf, B., and Smola, A. J. (2007). Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608.

Jiang, J. and Zhai, C. (2007). Instance weighting for domain adaptation in nlp. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 264–271.

Kalousis, A., Gama, J., and Hilario, M. (2004). On data and algorithms: Understanding inductive performance. *Machine learning*, 54(3):275–312.

Kulis, B., Saenko, K., and Darrell, T. (2011). What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1785–1792. IEEE.

Lee, S.-I., Chatalbashev, V., Vickrey, D., and Koller, D. (2007). Learning a meta-level prior for feature relevance from multiple related tasks. In *Proceedings of the 24th international conference on Machine learning*, pages 489–496. ACM.

Lemke, C., Budka, M., and Gabrys, B. (2015). Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130.

Li, F., Pan, S. J., Jin, O., Yang, Q., and Zhu, X. (2012). Cross-domain co-extraction of sentiment and topic lexicons. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 410–419. Association for Computational Linguistics.

Li, W., Duan, L., Xu, D., and Tsang, I. W. (2014). Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1134–1148.

Liao, X., Xue, Y., and Carin, L. (2005). Logistic regression with an auxiliary data source. In *Proceedings of the 22nd international conference on Machine learning*, pages 505–512. ACM.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

Lichman, M. (2013). UCI machine learning repository.

Liu, F., Zhang, G., Lu, H., and Lu, J. (2017). Heterogeneous unsupervised cross-domain transfer learning. arxiv preprint. *arXiv preprint arXiv:1701.02511*.

Liu, N. and Zaidi, N. A. (2016). Artificial neural network: deep or broad? an empirical study. In *Australasian joint conference on artificial intelligence*, pages 535–541. Springer.

Liu, X., Liu, Z., Wang, G., Cai, Z., and Zhang, H. (2018). Ensemble transfer learning algorithm. *IEEE Access*, 6:2389–2396.

Long, M., Wang, J., Ding, G., Sun, J., and Yu, P. S. (2013). Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207.

Mahmud, M. and Ray, S. (2008). Transfer learning using kolmogorov complexity: Basic theory and empirical evaluations. In *Advances in neural information processing systems*, pages 985–992.

Molina, M., Luna, J., Romero, C., and Ventura, S. (2012). Meta-learning approach for automatic parameter tuning: A case study with educational datasets. *International Educational Data Mining Society*.

Nam, J., Fu, W., Kim, S., Menzies, T., and Tan, L. (2017). Heterogeneous defect prediction. *IEEE Transactions on Software Engineering*.

Ngiam, J., Peng, D., Vasudevan, V., Kornblith, S., Le, Q. V., and Pang, R. (2018). Domain adaptive transfer learning with specialist models. *arXiv preprint arXiv:1811.07056*.

Pan, S. J., Kwok, J. T., and Yang, Q. (2008). Transfer learning via dimensionality reduction. In *AAAI*, volume 8, pages 677–682.

Pan, S. J., Ni, X., Sun, J.-T., Yang, Q., and Chen, Z. (2010). Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th international conference on World wide web*, pages 751–760. ACM.

Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q. (2011). Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210.

Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Pasupa, K. and Sunhem, W. (2016). A comparison between shallow and deep architecture classifiers on small dataset. In *Information Technology and Electrical Engineering (ICITEE), 2016 8th International Conference on*, pages 1–6. IEEE.

Pavelski, L., Delgado, M., and Kessaci, M.-E. (2018). Meta-learning for optimization: A case study on the flowshop problem using decision trees. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

Pimentel, B. A. and de Carvalho, A. C. (2018). A new data characterization for selecting clustering algorithms using meta-learning. *Information Sciences*.

Prettenhofer, P. and Stein, B. (2010). Cross-language text classification using structural correspondence learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 1118–1127. Association for Computational Linguistics.

Prudêncio, R. B. and Ludermir, T. B. (2004). Meta-learning approaches to selecting time series models. *Neurocomputing*, 61:121–137.

Qi, G.-J., Aggarwal, C., and Huang, T. (2011). Towards semantic knowledge propagation from text corpus to web images. In *Proceedings of the 20th international conference on World wide web*, pages 297–306. ACM.

Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. (2007). Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM.

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier.

Ripley, B. D. (2007). *Pattern recognition and neural networks*. Cambridge university press.

Rosenstein, M. T., Marx, Z., Kaelbling, L. P., and Dietterich, T. G. (2005). To transfer or not to transfer. In *NIPS 2005 workshop on transfer learning*, volume 898, pages 1–4.

Saeedi, R., Ghasemzadeh, H., and Gebremedhin, A. H. (2016). Transfer learning algorithms for autonomous reconfiguration of wearable systems. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 563–569. IEEE.

Seah, C.-W., Ong, Y.-S., and Tsang, I. W. (2013). Combating negative transfer from predictive distribution differences. *IEEE transactions on cybernetics*, 43(4):1153–1165.

Serban, F., Vanschoren, J., Kietz, J.-U., and Bernstein, A. (2013). A survey of intelligent assistants for data analysis. *ACM Computing Surveys (CSUR)*, 45(3):31.

Shi, X., Liu, Q., Fan, W., Philip, S. Y., and Zhu, R. (2010). Transfer learning on heterogenous feature spaces via spectral transformation. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 1049–1054. IEEE.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6.

Spyromitros-Xioufis, E., Tsoumakas, G., Groves, W., and Vlahavas, I. (2012). Multi-label classification methods for multi-target regression. *ArXiv e-prints*.

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., and Zeileis, A. (2008). Conditional variable importance for random forests. *BMC bioinformatics*, 9(1):307.

Strobl, C., Boulesteix, A.-L., Zeileis, A., and Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC bioinformatics*, 8(1):25.

Sugiyama, M., Nakajima, S., Kashima, H., Buenau, P. V., and Kawanabe, M. (2008). Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440.

Team, R. C. et al. (2013). R: A language and environment for statistical computing.

Therneau, T., Atkinson, B., and Ripley, B. (2015). rpart: Recursive partitioning and regression trees. r package version 4.1–10.

Tommasi, T., Orabona, F., and Caputo, B. (2010). Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *Proceedings of IEEE Computer Vision and Pattern Recognition Conference*, number EPFL-CONF-192668.

Vanschoren, J. (2018). Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*.

Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

Wang, C. and Mahadevan, S. (2008). Manifold alignment using procrustes analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1120–1127. ACM.

Wang, C. and Mahadevan, S. (2011). Heterogeneous domain adaptation using manifold alignment. In *IJCAI proceedings-international joint conference on artificial intelligence*, volume 22, page 1541.

Wang, Z., Song, Y., and Zhang, C. (2008). Transferred dimensionality reduction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 550–565. Springer.

Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, 3(1):9.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Xiong, P., Zhu, Y., Sun, Z., Cao, Z., Wang, M., Zheng, Y., Hou, J., Huang, T., and Que, Z. (2018). Application of transfer learning in continuous time series for anomaly detection in commercial aircraft flight data. In *2018 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 13–18. IEEE.

Xue, G.-R., Dai, W., Yang, Q., and Yu, Y. (2008). Topic-bridged plsa for cross-domain text classification. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 627–634. ACM.

Yao, Y. and Doretto, G. (2010). Boosting for transfer learning with multiple sources. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 1855–1862. IEEE.

Zheng, Y., Zhong, G., and Dong, J. (2016). Marginal deep architectures: Deep learning for small and middle scale applications.

Zhou, J. T., Pan, S. J., Tsang, I. W., and Yan, Y. (2014a). Hybrid heterogeneous transfer learning through deep learning. In *AAAI*, pages 2213–2220.

Zhou, J. T., Tsang, I. W., Pan, S. J., and Tan, M. (2014b). Heterogeneous domain adaptation for multiple classes. In *Artificial Intelligence and Statistics*, pages 1095–1103.

Zhu, Y., Chen, Y., Lu, Z., Pan, S. J., Xue, G.-R., Yu, Y., and Yang, Q. (2011). Heterogeneous transfer learning for image classification. In *AAAI*.

Appendices

A Datasets

Table A.1: Complete list of UCI Datasets used for the Empirical study of the performance of neural networks (*ES*, Chapter 3), Metalearning for Parameter Selection in Neural Networks (*PS*, Chapter 4), Weights Transfer in Heterogeneous Domain Neural Networks (*WT*, Chapter 5) and Metalearning for source selection in heterogeneous transfer learning for neural networks (*SST*, Chapter 6).

ES	PS	WT	SST	id	Dataset	Dependent Variable
✓	✓	✓	✓	1	Airfoil Self-Noise	Scaled.sound.pressure.level
✓	✓	✓	✓	2_1	Condition Based Maintenance of Naval Propulsion Plants	GTCompressor
✓	✓	✓	✓	2_2		GTTurbine
✓	✓	✓	✓	3	Combined Cycle Power Plant	PE
✓	✓	✓	✓	4	Communities and Crime	ViolentCrimesPerPop
✓	✓	✓	✓	5_1	Communities and Crime Unnormalized	murders
✓	✓	✓	✓	5_2		murdPerPop
✓	✓	✓	✓	5_3		rapes
✓	✓			5_4		rapesPerPop
✓	✓			5_5		robberies
✓	✓	✓	✓	5_6		robbbPerPop
✓	✓	✓	✓	5_7		assaults
✓	✓	✓	✓	5_8		assaultPerPop
✓	✓	✓	✓	5_9		burglaries
✓	✓			5_10		burglPerPop
✓	✓			5_11		larcenies
✓	✓	✓	✓	5_12		larcPerPop
✓	✓			5_13		autoTheft
✓	✓	✓	✓	5_14		autoTheftPerPop
✓	✓			5_15		arsons
✓	✓	✓	✓	5_16		arsonsPerPop
✓	✓	✓	✓	5_17		violentPerPop
✓	✓	✓	✓	5_18		nonViolPerPop
✓	✓	✓	✓	6	Concrete Compressive Strength	Concrete.compressive.strength.MPa..megapascals.
✓	✓	✓	✓	7	Computer Hardware	ERP
✓				8	Challenger USA Space Shuttle O-Ring (erosion only)	Number.experiencing.thermal.distress
✓				9	Challenger USA Space Shuttle O-Ring (erosion or blowby	Number.experiencing.thermal.distress
✓	✓	✓	✓	10	Online News Popularity	shares
✓	✓	✓	✓	11_1	Parkinsons Telemonitoring	motor_UPDRS
✓	✓	✓	✓	11_2		total_UPDRS
✓	✓	✓	✓	12_1	Concrete Slump Test	SLUMP.cm.
✓	✓	✓	✓	12_2		FLOW.cm.
✓	✓	✓	✓	12_3		Compressive.Strength..28.day..Mpa.
✓	✓			13	Buzz in social media	ND
✓	✓	✓	✓	15	Wine Quality (red)	quality
✓	✓	✓	✓	16	Wine Quality (white)	quality
✓	✓	✓	✓	17	Yacht Hydrodynamics	Residuary.resistance.per.unit.weight.of.displacement

B Auxiliary plots

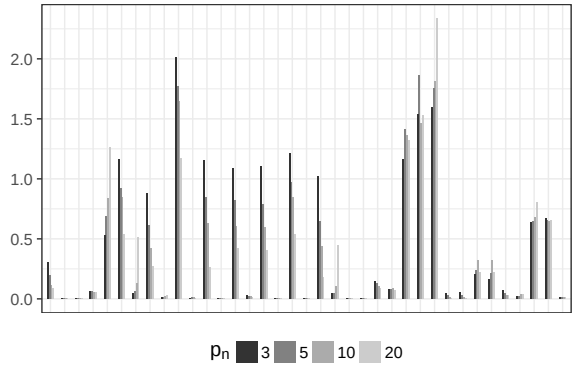
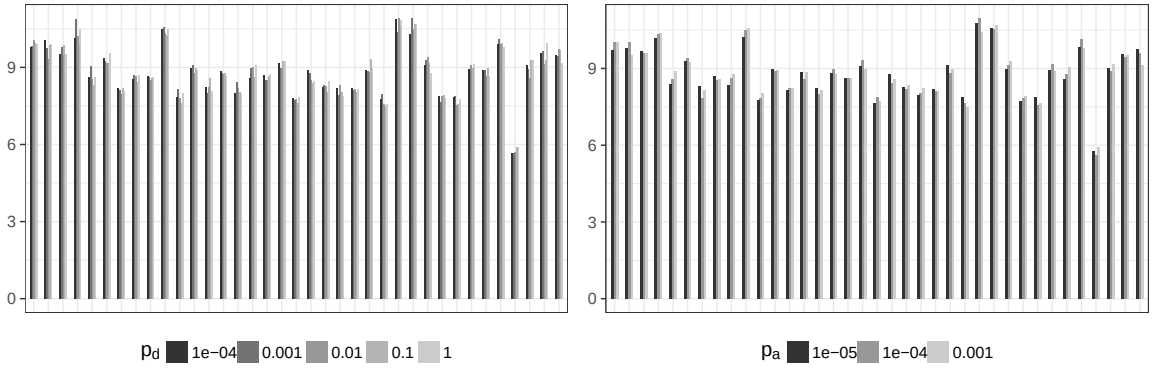
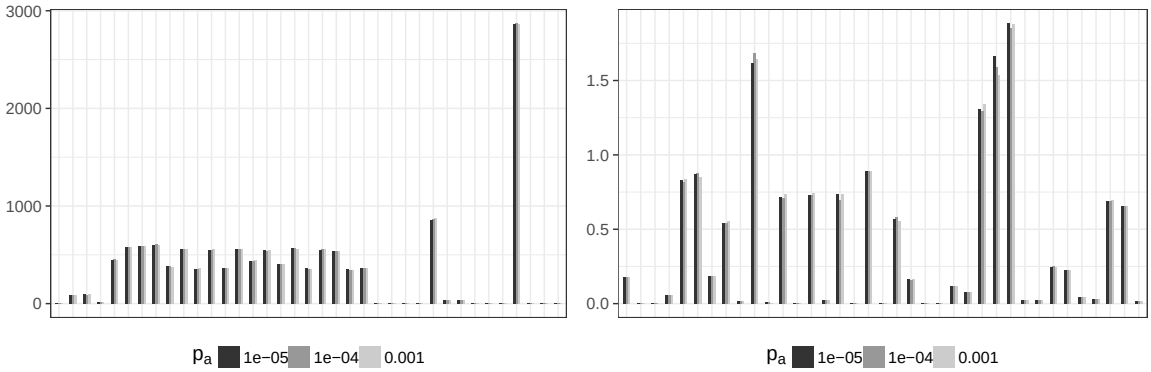


Figure B.1: Neural network MSE by p_n. The xx axis refers to the datasets used in the same order as on Table A.1 in Appendix A. The yy axis corresponds to the MSE obtained for each dataset.



(a) Neural network MSE0 by p_d (b) Neural network MSE0 by p_a

Figure B.2: Neural network MSE0. The xx axis refers to the datasets used in the same order as on Table A.1 in Appendix A. The yy axis corresponds to the $MSE0$ obtained for each dataset.



(a) Neural network duration by p_a

(b) Neural network MSE by p_a

Figure B.3: Neural network performance by p_a. The xx axis refers to the datasets used in the same order as on Table A.1 in Appendix A. The yy axis corresponds to the performance metric obtained for each dataset.

C Auxiliary tables

Table C.1: Distribution of problems over parameters (percentage).

(a) Parameter p_n

p_n	P1 (%)	P2 (%)	Both (%)
3	2.87	6.30	2.85
5	1.45	4.59	1.43
10	0.86	4.72	0.81
20	1.36	5.13	1.32

(b) Parameter p_a

p_a	P1 (%)	P2 (%)	Both (%)
0.00001	1.68	5.17	1.65
0.0001	1.61	5.14	1.56
0.001	1.61	5.25	1.60

Table C.2: Distribution of problems over parameters for dataset 10_1 (percentage).

(a) Parameter p_n

p_n	P1 (%)	P2 (%)	Both (%)
3	51.67	93.50	51.67
5	50.33	98.17	50.33
10	49.83	100.00	49.83
20	49.83	100.00	49.83

(b) Parameter p_d

p_d	P1 (%)	P2 (%)	Both (%)
0.0001	51.04	94.79	51.04
0.001	50.42	95.83	50.42
0.01	50.42	98.96	50.42
0.1	51.67	100.00	51.67
1	48.54	100.00	48.54

(c) Parameter p_a

p_a	P1 (%)	P2 (%)	Both (%)
0.00001	49.88	98.50	49.88
0.0001	50.63	97.38	50.63
0.001	50.75	97.88	50.75

D Parameterisations

id	p_D	p_n	p_d	p_a
1	b	3	0.0001	0.00001
2	b	3	0.0001	0.0001
3	b	3	0.0001	0.001
4	b	3	0.001	0.00001
5	b	3	0.001	0.0001
6	b	3	0.001	0.001
7	b	3	0.01	0.00001
8	b	3	0.01	0.0001
9	b	3	0.01	0.001
10	b	3	0.1	0.00001
11	b	3	0.1	0.0001
12	b	3	0.1	0.001
13	b	3	1	0.00001
14	b	3	1	0.0001
15	b	3	1	0.001
16	b	5	0.0001	0.00001
17	b	5	0.0001	0.0001
18	b	5	0.0001	0.001
19	b	5	0.001	0.00001
20	b	5	0.001	0.0001
21	b	5	0.001	0.001
22	b	5	0.01	0.00001
23	b	5	0.01	0.0001
24	b	5	0.01	0.001
25	b	5	0.1	0.00001
26	b	5	0.1	0.0001
27	b	5	0.1	0.001
28	b	5	1	0.00001
29	b	5	1	0.0001
30	b	5	1	0.001
31	b	10	0.0001	0.00001
32	b	10	0.0001	0.0001
33	b	10	0.0001	0.001

Continued on next page

id	p_D	p_n	p_d	p_a
34	b	10	0.001	0.00001
35	b	10	0.001	0.0001
36	b	10	0.001	0.001
37	b	10	0.01	0.00001
38	b	10	0.01	0.0001
39	b	10	0.01	0.001
40	b	10	0.1	0.00001
41	b	10	0.1	0.0001
42	b	10	0.1	0.001
43	b	10	1	0.00001
44	b	10	1	0.0001
45	b	10	1	0.001
46	b	20	0.0001	0.00001
47	b	20	0.0001	0.0001
48	b	20	0.0001	0.001
49	b	20	0.001	0.00001
50	b	20	0.001	0.0001
51	b	20	0.001	0.001
52	b	20	0.01	0.00001
53	b	20	0.01	0.0001
54	b	20	0.01	0.001
55	b	20	0.1	0.00001
56	b	20	0.1	0.0001
57	b	20	0.1	0.001
58	b	20	1	0.00001
59	b	20	1	0.0001
60	b	20	1	0.001
61	u	3	0.0001	0.00001
62	u	3	0.0001	0.0001
63	u	3	0.0001	0.001
64	u	3	0.001	0.00001
65	u	3	0.001	0.0001
66	u	3	0.001	0.001
67	u	3	0.01	0.00001
68	u	3	0.01	0.0001

Continued on next page

id	p_D	p_n	p_d	p_a
69	u	3	0.01	0.001
70	u	3	0.1	0.00001
71	u	3	0.1	0.0001
72	u	3	0.1	0.001
73	u	3	1	0.00001
74	u	3	1	0.0001
75	u	3	1	0.001
76	u	5	0.0001	0.00001
77	u	5	0.0001	0.0001
78	u	5	0.0001	0.001
79	u	5	0.001	0.00001
80	u	5	0.001	0.0001
81	u	5	0.001	0.001
82	u	5	0.01	0.00001
83	u	5	0.01	0.0001
84	u	5	0.01	0.001
85	u	5	0.1	0.00001
86	u	5	0.1	0.0001
87	u	5	0.1	0.001
88	u	5	1	0.00001
89	u	5	1	0.0001
90	u	5	1	0.001
91	u	10	0.0001	0.00001
92	u	10	0.0001	0.0001
93	u	10	0.0001	0.001
94	u	10	0.001	0.00001
95	u	10	0.001	0.0001
96	u	10	0.001	0.001
97	u	10	0.01	0.00001
98	u	10	0.01	0.0001
99	u	10	0.01	0.001
100	u	10	0.1	0.00001
101	u	10	0.1	0.0001
102	u	10	0.1	0.001
103	u	10	1	0.00001

Continued on next page

id	p_D	p_n	p_d	p_a
104	u	10	1	0.0001
105	u	10	1	0.001
106	u	20	0.0001	0.00001
107	u	20	0.0001	0.0001
108	u	20	0.0001	0.001
109	u	20	0.001	0.00001
110	u	20	0.001	0.0001
111	u	20	0.001	0.001
112	u	20	0.01	0.00001
113	u	20	0.01	0.0001
114	u	20	0.01	0.001
115	u	20	0.1	0.00001
116	u	20	0.1	0.0001
117	u	20	0.1	0.001
118	u	20	1	0.00001
119	u	20	1	0.0001
120	u	20	1	0.001

Table D.2: Ranking - Top 5 parameterisations (each dataset).

1_1	**2_1**	**2_2**	**3_1**	**4_1**	**5_1**	**5_2**	**5_3**	**5_4**	**5_5**	**5_6**	**5_7**	**5_8**
52	108	47	113	74	27	66	38	62	112	61	101	1
113	106	46	53	75	25	65	37	61	101	1	40	2
51	107	107	54	73	106	3	39	52	53	2	42	64
50	47	108	50	14	108	4	102	112	102	66	55	16
114	46	48	52	13	111	2	42	53	107	32	53	65

5_9	**5_10**	**5_11**	**5_12**	**5_13**	**5_14**	**5_15**	**5_16**	**5_17**	**5_18**	**6_1**	**7_1**	**8_1**
101	54	117	62	46	66	55	35	3	62	50	3	86
102	52	55	3	42	1	49	36	62	18	116	2	85
100	53	115	61	47	65	53	69	63	17	49	62	87
40	55	116	1	100	5	52	2	18	16	109	1	27
42	113	57	2	40	116	54	78	17	77	53	63	12

9_1	**10_1**	**11_1**	**11_2**	**12_1**	**12_2**	**12_3**	**13_1**	**15_1**	**16_1**	**17_1**
70	15	51	51	11	75	111	100	28	118	47
72	13	50	111	59	73	96	42	89	119	106
11	14	110	109	120	13	110	101	13	120	108
71	74	49	50	118	74	109	85	43	103	48
26	30	111	49	119	90	95	87	15	58	107

Table D.3: Best and worst parameterisations for each dataset.

(a) Best

dataset	p_D	p_n	p_d	p_a
1_1	b	20	0.01	0.00001
2_1	u	20	0.0001	0.001
2_2	b	20	0.0001	0.0001
3_1	u	20	0.01	0.0001
4_1	u	3	1	0.0001
5_1	b	5	0.1	0.001
5_2	u	3	0.001	0.001
5_3	b	10	0.01	0.0001
5_4	u	3	0.0001	0.0001
5_5	u	20	0.01	0.00001
5_6	u	3	0.0001	0.00001
5_7	u	10	0.1	0.0001
5_8	b	3	0.0001	0.00001
5_9	u	10	0.1	0.0001
5_10	b	20	0.01	0.001
5_11	u	20	0.1	0.001
5_12	u	3	0.0001	0.0001
5_13	b	20	0.0001	0.00001
5_14	u	3	0.001	0.001
5_15	b	20	0.1	0.00001
5_16	b	10	0.001	0.0001
5_17	b	3	0.0001	0.001
5_18	u	3	0.0001	0.0001
6_1	b	20	0.001	0.0001
7_1	b	3	0.0001	0.001
8_1	u	5	0.1	0.0001
9_1	u	3	0.1	0.00001
10_1	b	3	1	0.001
11_1	b	20	0.001	0.001
11_2	b	20	0.001	0.001
12_1	b	3	0.1	0.0001
12_2	u	3	1	0.001
12_3	u	20	0.001	0.001
13_1	u	10	0.1	0.00001
15_1	b	5	1	0.00001
16_1	u	20	1	0.00001
17_1	b	20	0.0001	0.0001

(b) Worst

dataset	p_D	p_n	p_d	p_a
1_1	u	3	0.0001	0.001
2_1	b	3	1	0.00001
2_2	b	3	1	0.001
3_1	b	3	1	0.00001
4_1	u	5	0.0001	0.001
5_1	b	3	0.0001	0.001
5_2	b	20	0.0001	0.001
5_3	u	3	0.0001	0.00001
5_4	b	20	1	0.00001
5_5	b	3	0.0001	0.00001
5_6	u	10	0.0001	0.0001
5_7	b	3	0.0001	0.001
5_8	u	10	0.0001	0.001
5_9	u	3	0.0001	0.0001
5_10	b	3	1	0.001
5_11	u	3	0.0001	0.0001
5_12	b	3	1	0.001
5_13	b	3	0.0001	0.00001
5_14	u	10	0.0001	0.001
5_15	u	3	0.0001	0.0001
5_16	u	20	0.001	0.00001
5_17	u	10	0.0001	0.00001
5_18	b	3	1	0.00001
6_1	u	3	1	0.00001
7_1	b	3	1	0.001
8_1	u	5	0.0001	0.0001
9_1	b	5	0.0001	0.00001
10_1	u	20	0.0001	0.00001
11_1	u	3	0.0001	0.0001
11_2	b	3	1	0.00001
12_1	u	10	0.0001	0.00001
12_2	u	10	0.0001	0.0001
12_3	b	3	1	0.00001
13_1	u	10	0.0001	0.001
15_1	u	20	0.0001	0.001
16_1	b	20	0.0001	0.00001
17_1	u	5	1	0.001

E Complete evaluation results for transfer learning with p^{meta} parameterisations

Table E.1: Accuracies (percentage) for meta-target MSE0.

	No CFS						CFS(cutoff=0.75)						CFS(cutoff=0.5)					
	No scaling			Scaling			No scaling			Scaling			No scaling			Scaling		
	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	54	57	57	54	57	57	54	57	57	55	57	57	55	55	57	55	55	57
MF_T^2	61	57	57	61	57	57	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	57	56	57	57	56	57	57	56	57	57	56	57	57	56	57	57	56	57
MF_T^4	57	**72**	57	57	**72**	57	57	69	57	57	69	57	56	55	57	57	55	57
MF_T^5	57	55	57	57	55	57	57	54	57	57	54	57	57	56	57	57	56	57
MF_T^6	57	58	57	57	58	57	57	58	57	57	58	57	57	58	57	57	58	57
MF_T^7	57	66	57	57	66	57	57	68	57	57	68	57	57	66	57	57	66	57

Table E.2: Accuracies (percentage) for meta-target duration.

	NO CFS						CFS(cutoff=0.75)						CFS(cutoff=0.5)					
	No scaling			Scaling			No scaling			Scaling			No scaling			Scaling		
	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	62	63	64	62	63	64	62	63	66	62	63	66	61	66	60	61	66	60
MF_T^2	53	55	57	53	55	57	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	57	51	57	56	51	57	51	53	52	51	53	52	52	53	51	52	53	51
MF_T^4	64	62	64	63	62	64	63	63	**67**	62	63	**67**	63	64	64	62	64	64
MF_T^5	61	60	61	61	60	61	64	59	63	63	59	63	62	61	64	62	61	64
MF_T^6	58	52	53	58	52	53	55	52	55	55	52	55	54	52	47	53	52	47
MF_T^7	61	59	62	61	59	62	63	58	66	63	58	66	63	60	62	63	60	62

Table E.3: Accuracies (percentage) for meta-target MSE.

	NO CFS						CFS(cutoff=0.75)						CFS(cutoff=0.5)					
	No scaling			Scaling			No scaling			Scaling			No scaling			Scaling		
	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	55	55	55	54	55	55	54	54	55	54	54	55	53	53	55	54	53	55
MF_T^2	52	53	55	52	53	55	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	55	53	55	54	53	55	55	53	55	55	53	55	56	54	55	55	54	55
MF_T^4	54	55	55	55	55	55	54	53	55	54	53	55	54	52	55	54	52	55
MF_T^5	54	50	55	55	50	55	53	49	55	53	49	55	54	50	55	54	50	55
MF_T^6	55	48	55	54	48	55	53	50	55	53	50	55	55	51	55	55	51	55
MF_T^7	54	47	55	54	47	55	53	49	55	53	49	55	54	49	55	54	49	55

Table E.4: Percentage of cases when metalearning correctly predicts the best transfer, with meta-target MSE0.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	7	7	7	7	0	4	7	4	0	0	4	0
MF_T^2	11	7	**14**	7	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	4	7	0	7	4	4	4	4	7	11	11	11
MF_T^4	4	4	4	4	7	11	7	11	0	11	0	11
MF_T^5	0	4	0	4	4	4	4	4	0	0	0	0
MF_T^6	0	**14**	4	**14**	11	11	4	11	7	7	7	7
MF_T^7	0	4	0	4	4	7	0	7	7	11	4	11

Table E.5: Percentage of cases when metalearning correctly predicts the best transfer, with meta-target duration.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	25	14	25	14	25	11	25	11	21	7	25	7
MF_T^2	4	11	4	11	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	4	0	4	0	0	4	4	4	4	7	4	7
MF_T^4	32	18	32	18	29	7	29	7	25	11	25	11
MF_T^5	18	11	14	11	18	11	18	11	18	11	21	11
MF_T^6	7	4	11	4	11	11	14	11	11	14	14	14
MF_T^7	29	11	25	11	25	11	21	11	21	4	21	4

Table E.6: Percentage of cases when metalearning correctly predicts the best transfer, with meta-target MSE.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	0	4	0	4	11	4	0	4	11	7	4	7
MF_T^2	4	4	4	4	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	0	7	4	7	0	4	7	4	7	4	4	4
MF_T^4	4	0	11	0	7	7	4	7	7	4	4	4
MF_T^5	4	0	0	0	0	0	0	0	7	0	11	0
MF_T^6	0	7	0	7	0	7	7	7	4	7	7	7
MF_T^7	0	0	4	0	0	4	0	4	0	4	7	4

Table E.7: Percentage of cases when metalearning correctly predicts a good transfer, with meta-target MSE0.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	46	39	50	39	36	39	36	39	46	43	43	43
MF_T^2	75	93	79	93	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	57	46	57	46	46	50	46	50	61	54	54	54
MF_T^4	57	50	46	50	46	57	57	57	50	54	39	54
MF_T^5	46	46	43	46	32	32	32	32	36	39	43	39
MF_T^6	64	71	57	71	61	68	61	68	57	75	61	75
MF_T^7	46	54	46	54	46	61	43	61	50	68	46	68

Table E.8: Percentage of cases when metalearning correctly predicts a good transfer, with meta-target duration.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	75	57	75	57	75	75	75	75	71	54	71	54
MF_T^2	46	68	46	68	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	54	29	50	29	46	32	54	32	39	36	43	36
MF_T^4	75	57	75	57	75	75	75	75	75	82	75	82
MF_T^5	71	61	71	61	71	71	75	71	79	79	82	79
MF_T^6	57	46	57	46	57	46	64	46	61	54	61	54
MF_T^7	82	61	75	61	75	68	79	68	75	79	79	79

Table E.9: Percentage of cases when metalearning correctly predicts a good transfer, with meta-target MSE.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	57	54	54	54	46	57	46	57	39	57	46	57
MF_T^2	39	36	46	36	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	57	46	43	46	39	46	54	46	46	39	43	39
MF_T^4	57	57	54	57	46	54	43	54	50	54	43	54
MF_T^5	39	54	36	54	39	36	36	36	46	39	46	39
MF_T^6	54	46	43	46	39	43	39	43	46	46	43	46
MF_T^7	46	64	39	64	43	39	43	39	43	54	54	54

F Complete evaluation results for transfer learning with p^{grid} parameterisations

Table F.1: Accuracies (percentage) for meta-target MSE0.

	NO CFS						CFS(cutoff=0.75)						CFS(cutoff=0.5)					
	No scaling			Scaling			No scaling			Scaling			No scaling			Scaling		
	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	58	57	59	58	57	59	58	59	59	58	59	59	58	56	59	59	56	59
MF_T^2	61	59	59	61	59	59	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	59	58	59	59	58	59	59	59	59	59	59	59	59	59	59	59	59	59
MF_T^4	60	**69**	59	60	**69**	59	60	**69**	59	60	**69**	59	59	57	59	59	57	59
MF_T^5	60	54	59	59	54	59	59	57	59	60	57	59	59	58	59	59	58	59
MF_T^6	59	61	59	59	61	59	59	60	59	59	60	59	60	59	59	60	59	59
MF_T^7	60	64	59	60	64	59	60	67	59	60	67	59	60	64	59	60	64	59

Table F.2: Accuracies (percentage) for meta-target duration.

	NO CFS						CFS(cutoff=0.75)						CFS(cutoff=0.5)					
	No scaling			Scaling			No scaling			Scaling			No scaling			Scaling		
	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	68	73	74	68	73	74	68	74	74	69	74	74	68	73	76	68	73	76
MF_T^2	68	64	70	68	64	70	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	69	66	65	68	66	65	63	64	65	63	64	65	62	65	63	61	65	63
MF_T^4	76	73	75	76	73	75	78	73	75	78	73	75	76	73	75	75	73	75
MF_T^5	78	72	74	78	72	74	77	74	71	77	74	71	77	72	73	77	72	73
MF_T^6	72	66	66	71	66	66	70	63	70	69	63	70	70	63	70	69	63	70
MF_T^7	**79**	71	75	**79**	71	75	78	72	74	**79**	72	74	78	72	73	78	72	73

Table F.3: Accuracies (percentage) for meta-target MSE.

	NO CFS						CFS(cutoff=0.75)						CFS(cutoff=0.5)					
	No scaling			Scaling			No scaling			Scaling			No scaling			Scaling		
	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T	A_F	A_L	A_T
MF_T^1	64	61	64	64	61	64	64	60	63	63	60	63	63	61	64	63	61	64
MF_T^2	62	63	64	62	63	64	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	65	63	64	65	63	64	65	62	64	65	62	64	65	61	64	64	61	64
MF_T^4	64	63	64	64	63	64	64	59	64	64	59	64	63	62	64	64	62	64
MF_T^5	64	61	64	64	61	64	63	61	64	64	61	64	64	60	64	64	60	64
MF_T^6	66	63	64	65	63	64	65	60	64	65	60	64	65	60	64	**66**	60	64
MF_T^7	64	64	64	64	64	64	64	63	64	64	63	64	64	61	64	64	61	64

Table F.4: Percentage of cases when metalearning correctly predicts the best transfer, with meta-target MSE0.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	4	0	0	0	0	0	4	0	4	0	4	0
MF_T^2	11	11	11	11	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	**21**	4	18	4	0	7	7	7	4	14	4	14
MF_T^4	0	0	0	0	4	0	7	0	0	7	4	7
MF_T^5	7	0	7	0	7	0	4	0	0	0	7	0
MF_T^6	14	18	18	18	14	18	7	18	14	18	14	18
MF_T^7	7	4	4	4	0	0	4	0	4	11	4	11

Table F.5: Percentage of cases when metalearning correctly predicts the best transfer, with meta-target duration.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	7	18	11	18	7	14	11	14	7	14	7	14
MF_T^2	7	18	4	18	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	0	4	4	4	4	4	4	4	7	0	4	0
MF_T^4	14	14	14	14	18	11	18	11	11	11	11	11
MF_T^5	7	14	7	14	11	7	14	7	14	14	7	14
MF_T^6	4	4	7	4	11	18	7	18	14	14	14	14
MF_T^7	11	14	14	14	14	14	14	14	**21**	14	**21**	14

Table F.6: Percentage of cases when metalearning correctly predicts the best transfer, with meta-target MSE.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	4	4	7	4	4	0	4	0	4	4	4	4
MF_T^2	4	0	4	0	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	0	4	0	4	0	0	0	0	4	4	0	4
MF_T^4	4	0	7	0	**11**	0	7	0	7	0	0	0
MF_T^5	**11**	4	7	4	7	4	7	4	4	4	4	4
MF_T^6	0	0	7	0	0	0	7	0	**11**	0	4	0
MF_T^7	7	0	4	0	4	4	4	4	7	0	4	0

Table F.7: Percentage of cases when metalearning correctly predicts a good transfer, with meta-target MSE0.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	32	18	36	18	29	29	32	29	29	32	36	32
MF_T^2	64	89	71	89	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	68	46	54	46	50	61	57	61	50	54	54	54
MF_T^4	43	50	43	50	43	36	57	36	36	39	36	39
MF_T^5	46	36	50	36	43	32	50	32	39	43	46	43
MF_T^6	68	89	71	89	86	93	64	93	71	**96**	79	**96**
MF_T^7	50	46	46	46	46	36	50	36	46	46	50	46

Table F.8: Percentage of cases when metalearning correctly predicts a good transfer, with meta-target duration.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	43	57	43	57	54	50	46	50	46	50	46	50
MF_T^2	57	61	57	61	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	39	18	36	18	43	50	43	50	43	32	39	32
MF_T^4	57	57	57	57	61	54	61	54	54	54	50	54
MF_T^5	64	61	64	61	64	54	61	54	64	50	61	50
MF_T^6	50	57	50	57	46	50	43	50	46	50	50	50
MF_T^7	64	57	64	57	**68**	50	61	50	**68**	54	64	54

Table F.9: Percentage of cases when metalearning correctly predicts a good transfer, with meta-target MSE.

	No CFS				CFS(cutoff=0.75)				CFS(cutoff=0.5)			
	No scalling		Scalling		No scalling		Scalling		No scalling		Scalling	
	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L	A_F	A_L
MF_T^1	50	43	46	43	39	36	43	36	39	46	39	46
MF_T^2	43	32	36	32	NA	NA	NA	NA	NA	NA	NA	NA
MF_T^3	43	32	43	32	39	43	32	43	32	29	36	29
MF_T^4	54	39	46	39	50	36	**57**	36	46	50	39	50
MF_T^5	39	43	32	43	39	39	32	39	25	46	32	46
MF_T^6	46	32	54	32	39	43	46	43	39	32	32	32
MF_T^7	36	43	36	43	39	39	36	39	36	50	29	50

G Transfer learning impact on the neural networks' performances

Table G.1: Higher improvement for NNs parameterised with p^{meta} parameterisation.

(a) MSE0

d_S	d_T	mapping	imp.
3_1	1_1	KCor	61.4
5_14	2_1	KCor	33.5
5_5	2_2	SCor	18.4
7_1	3_1	PCor	91.1
5_3	4_1	SCor	19.6
5_7	5_1	KL	77.5
5_6	5_2	KL	38.1
5_9	5_3	KL	90.5
5_18	5_6	PCor	80.4
5_13	5_7	KL	85.4
5_18	5_8	KCor	84.3
5_13	5_9	KL	79.2
5_18	5_12	PCor	83.6
5_6	5_14	KL	55.6
5_18	5_16	KCor	22.1
5_18	5_17	KCor	89.9
5_17	5_18	SCor	96.8
7_1	6_1	PCor	37.0
5_18	7_1	SCor	88.7
5_6	10_1	KCor	4.4
5_18	11_1	SCor	86.5
5_18	11_2	KCor	89.3
5_18	12_1	PCor	83.3
11_1	12_2	KCor	82.6
17_1	12_3	PCor	43.3
12_2	15_1	KCor	49.0
12_2	16_1	KCor	49.0
12_3	17_1	KCor	84.8

(b) duration

d_S	d_T	mapping	imp.
6_1	1_1	PCor	14.5
5_4	2_1	KL	11.1
2_1	2_2	KL	16.9
6_1	3_1	SCor	37.1
12_3	4_1	PCor	18.6
5_9	5_1	KL	74.4
5_4	5_2	KL	42.3
5_9	5_3	KL	48.0
5_8	5_6	PCor	44.7
5_13	5_7	KL	66.1
5_18	5_8	KCor	55.9
5_3	5_9	KL	58.2
5_8	5_12	SCor	52.4
5_6	5_14	KL	40.9
5_2	5_16	KCor	17.1
5_18	5_17	PCor	56.4
5_17	5_18	PCor	68.1
12_3	6_1	PCor	5.3
5_17	7_1	SCor	44.1
7_1	10_1	KL	10.5
4_1	11_1	SCor	24.8
5_8	11_2	PCor	3.0
12_3	12_1	SCor	19.5
12_1	12_2	PCor	38.0
12_2	12_3	PCor	13.0
11_2	15_1	KCor	18.8
11_2	16_1	SCor	9.2
12_2	17_1	PCor	24.1

(c) MSE

d_S	d_T	mapping	imp.
3_1	1_1	KCor	3.0
5_16	2_1	SCor	2.8
4_1	2_2	KL	6.8
2_1	3_1	KCor	0.3
12_3	4_1	PCor	12.4
12_2	5_1	SCor	8.4
5_13	5_2	PCor	63.4
7_1	5_3	KL	11.8
5_9	5_6	SCor	17.1
5_17	5_7	KL	21.0
5_17	5_8	KCor	41.6
5_10	5_9	PCor	9.6
2_2	5_12	SCor	34.9
5_3	5_14	PCor	22.9
5_17	5_16	SCor	15.6
5_18	5_17	SCor	36.8
5_8	5_18	SCor	53.0
5_7	6_1	KCor	6.1
5_8	7_1	SCor	5.8
12_3	10_1	PCor	5.6
5_2	11_1	KL	15.1
5_16	11_2	KL	5.9
2_1	12_1	KCor	1.6
5_10	12_2	KL	1.5
5_12	12_3	SCor	1.1
6_1	15_1	SCor	2.4
2_1	16_1	KCor	1.3
5_18	17_1	KL	5.1

Table G.2: Higher improvement for NNs parameterised with p^{grid} parameterisation.

(a) MSE0

d_S	d_T	mode	value
3_1	1_1	PCor	74
5_14	2_1	KL	40
5_6	2_2	KCor	33
12_2	3_1	PCor	67
5_1	4_1	SCor	33
5_13	5_1	KL	76
5_6	5_2	KL	38
5_9	5_3	KL	90
5_8	5_6	PCor	85
5_13	5_7	KL	85
5_18	5_8	PCor	84
5_13	5_9	KL	80
5_18	5_12	PCor	84
5_6	5_14	KL	56
5_18	5_16	KCor	20
5_18	5_17	KCor	90
5_17	5_18	SCor	97
7_1	6_1	PCor	55
5_18	7_1	SCor	90
5_6	10_1	KCor	4
5_18	11_1	SCor	87
5_18	11_2	KCor	90
5_18	12_1	PCor	84
11_1	12_2	KCor	83
7_1	12_3	SCor	47
12_2	15_1	KCor	49
12_2	16_1	KCor	49
5_18	17_1	KCor	79

(b) duration

d_S	d_T	mode	value
6_1	1_1	PCor	7
5_2	2_1	KL	17
4_1	2_2	R	11
15_1	3_1	PCor	13
5_8	4_1	KL	13
5_13	5_1	KL	72
5_4	5_2	KL	50
5_15	5_3	KL	48
5_8	5_6	PCor	58
5_13	5_7	KL	68
5_17	5_8	KCor	52
5_3	5_9	KL	62
5_8	5_12	SCor	58
5_17	5_14	PCor	46
5_4	5_16	PCor	47
5_18	5_17	PCor	61
5_17	5_18	KCor	49
15_1	6_1	SCor	4
5_8	7_1	SCor	19
5_2	10_1	SCor	13
15_1	11_1	KL	3
5_17	11_2	KCor	-4
12_3	12_1	KCor	9
12_1	12_2	PCor	25
5_17	12_3	KL	19
3_1	15_1	PCor	17
11_2	16_1	PCor	2
5_17	17_1	SCor	13

(c) MSE

d_S	d_T	mode	value
2_1	1_1	KCor	6
5_7	2_1	KL	4
5_4	2_2	SCor	6
1_1	3_1	KCor	1
5_3	4_1	KL	5
2_2	5_1	PCor	13
5_13	5_2	PCor	63
3_1	5_3	PCor	12
1_1	5_6	KCor	45
5_17	5_7	KL	21
2_1	5_8	SCor	35
5_3	5_9	PCor	3
2_2	5_12	SCor	35
5_8	5_14	KCor	60
5_17	5_16	KCor	49
5_18	5_17	SCor	37
5_3	5_18	PCor	66
5_17	6_1	PCor	2
2_2	7_1	KCor	51
12_1	10_1	KL	4
11_2	11_1	KL	50
11_1	11_2	KL	48
2_2	12_1	PCor	2
5_10	12_2	KCor	1
5_12	12_3	PCor	23
6_1	15_1	KCor	1
2_1	16_1	KCor	1
5_9	17_1	PCor	8

H Best SCor transfers for p^{meta} NNs

Table H.1: Impacts of the best transfers found for neural networks parameterised with p^{meta}.

(a) MSE0

ds	dt	impact
3	1	61
5_1	2_1	33
5_5	2_2	18
7_1	3	91
5_3	4	20
5_5	5_1	46
5_8	5_2	26
5_13	5_3	62
5_17	5_6	68
5_5	5_7	49
5_18	5_8	83
5_5	5_9	67
5_18	5_12	84
5_18	5_14	23
5_18	5_16	21
5_18	5_17	90
5_17	5_18	97
3	6_1	27
5_18	7_1	89
5_18	10_1	4
5_18	11	87
5_18	11_2	89
11	12_1	79
5_14	12_2	77
16	12_3	28
5_15	15	46
12_2	16	48
12_3	17	85

(b) duration

ds	dt	impact
5_2	1	13
4	2_1	9
1	2_2	10
6_1	3	37
12_2	4	17
5_5	5_1	54
5_10	5_2	28
5_11	5_3	19
5_18	5_6	42
5_5	5_7	33
5_18	5_8	53
5_5	5_9	39
5_8	5_12	52
5_17	5_14	36
5_18	5_16	15
5_18	5_17	53
5_17	5_18	61
5_6	6_1	2
5_17	7_1	44
12_3	10_1	8
4	11	25
3	11_2	0
12_3	12_1	20
12_3	12_2	36
12_2	12_3	10
3	15	17
11_2	16	9
12_3	17	22

(c) MSE

ds	dt	impact
3	1	3
5_16	2_1	3
5_3	2_2	6
2_1	3	0
12_2	4	11
12_2	5_1	8
5_8	5_2	55
12_1	5_3	8
5_9	5_6	17
5_8	5_7	19
5_18	5_8	36
16	5_9	9
2_2	5_12	35
5_5	5_14	19
5_17	5_16	16
5_18	5_17	37
5_8	5_18	53
5_10	6_1	4
5_8	7_1	6
12_1	10_1	2
5_14	11	12
5_4	11_2	5
12_2	12_1	-3
5_10	12_2	2
5_12	12_3	1
6_1	15	2
2_1	16	1
5_15	17	5

I Best SCor transfers for p^{grid} NNs

Table I.1: Impacts of the best transfers found for neural networks parameterised with p^{grid}.

(a) MSE0

ds	dt	impact
3	1	74
5_17	2_1	40
5_8	2_2	33
16	3	58
5_1	4	33
5_5	5_1	38
5_8	5_2	26
5_13	5_3	52
5_17	5_6	68
5_5	5_7	49
5_17	5_8	82
5_5	5_9	70
5_18	5_12	84
5_18	5_14	27
5_18	5_16	18
5_18	5_17	90
5_17	5_18	97
7_1	6_1	52
5_18	7_1	90
5_12	10_1	4
5_18	11	87
5_18	11_2	90
11	12_1	82
5_14	12_2	73
7_1	12_3	47
5_15	15	45
12_2	16	48
5_18	17	79

(b) duration

ds	dt	impact
6_1	1	5
4	2_1	17
5_2	2_2	10
16	3	10
5_14	4	9
5_5	5_1	65
5_10	5_2	36
5_13	5_3	27
5_17	5_6	50
5_5	5_7	37
5_17	5_8	51
5_5	5_9	41
5_8	5_12	58
5_8	5_14	45
5_4	5_16	44
5_18	5_17	58
5_17	5_18	45
15	6_1	4
5_8	7_1	19
5_2	10_1	13
12_3	11	-1
5_6	11_2	-9
12_3	12_1	8
12_3	12_2	23
5_17	12_3	12
3	15	16
11_2	16	2
5_17	17	13

(c) MSE

ds	dt	impact
2_1	1	6
5_3	2_1	3
5_4	2_2	6
1	3	1
5_1	4	5
5_2	5_1	5
5_8	5_2	55
5_5	5_3	4
1	5_6	39
5_8	5_7	19
2_1	5_8	35
11	5_9	2
2_2	5_12	35
7_1	5_14	32
5_12	5_16	43
5_18	5_17	37
5_15	5_18	52
5_5	6_1	-1
2_2	7_1	51
12_1	10_1	3
11_2	11	47
5_4	11_2	19
11_2	12_1	1
5_10	12_2	1
5_7	12_3	19
6_1	15	1
2_1	16	1
11	17	7