NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

A Study of Generalization in Regression

Proposal of a New Metric and Loss Function to better understand and improve Generability

Nuno Tiago Falcão Alpalhão

Dissertation presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

A Study of Generalization in Regression

Proposal of a New Metric and Loss Function to better understand and improve Generability

by

Nuno Tiago Falcão Alpalhão

Dissertation presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

Advisor: Leonardo Vanneschi

January 2021

Generalization in Regression

Acknowledgements

To all who helped me.

Abstract

Intuitively Generalization in Machine Learning can be understood as a models ability to apply its trained or acquired knowledge to a previously unseen scenario. In the recent years there has been an exponential growth in machine learning models both efficiency and accuracy, yet the current research is still trying to understand and trust how well models can perform on previously unseen data.

For this thesis we propose a study of machine learning's theoretical background to further expand the notion of generalization and it's limitation's, enabling us to derive its commonly accepted approximation, definitions that we will use to present a new generalization metric or score more consistent in detecting and providing understanding of the occurrence of generalization.

Additionally a new loss function will be presented in order to mitigate generalization error inherit to a noisy sample, where extensive tests suggest that our loss function has a higher rate of convergence while producing statistically similar or even better results when compared with classical loss functions.

Keywords: Generalization; Machine Learning; Loss Function; Metric; Noise;

Resumo

Intuitivamente generalização em Aprendizagem Automática pode ser entendida como a capacidade de um modelo em aplicar o seu conhecimento treinado ou adquirido a um cenário nunca antes visto. Nos últimos anos, tem existido um crescimento exponencial tanto na eficiência quanto na precisão dos modelos de Aprendizagem Automática, no entanto a pesquisa atual ainda se debate bastante em como entender e confiar na capacidade de execução dos modelos em dados nunca antes vistos.

Para esta tese, propomos um estudo dos fundamentos teóricos da Aprendizagem Automática para expandir ainda mais a noção de generalização e suas limitações, permitindo-nos derivar sua aproximação comummente aceita. Definições estas que usaremos para apresentar uma nova métrica de generalização mais consistente na detecção da ocorrência ou não de generalização.

Adicionalmente, uma nova função de perda será apresentada a fim de mitigar o erro de generalização herdado de uma amostra ruidosa, onde testes extensivos sugerem que nossa função de perda tem uma taxa de convergência significantemente mais alta produzindo resultados estatisticamente semelhantes ou até melhores quando comparada com as funções de perda clássicas.

Palavras-chave: Generalização; Aprendizagem Automática; Função de Perda; Métrica; Ruido;

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network.

GP Genetic Programming.

MAE Mean Absolute Error.

ML Machine Learning.

MSE Mean Squared Error.

Chapter 1

Introduction

In the current scenario where machine learning models are increasingly more used in our societal infrastructures the need to ensure they'll perform as expected in all possible situations is higher than ever. In the recent years there has been an exponential growth in machine learning models both efficiency and accuracy, yet the need to understand and trust how well they will perform on previously unseen data is still a highly debated topic, in other words *how well does a model generalize it's given problem?*

1.1 Proposal Overview

For this thesis we purpose a study of the mathematical basis of supervised machine learning in order to further understand the notion of generalization. The purpose is to use said theory to arrive at the commonly accepted approximation of generalization, which will enable us to to show the limitations inherent from a chosen model and problem set separately, making the link between the results derived from theory with the common tools that machine learning users commonly use.

Definitions required to later introduce a new more robust generalization metric called *generalization score* that aims to offer more explainability than its predecessor.

All of these notions will later serve as inspiration to propose a new loss function intended to improve a machine learning's model generalization, where a broad study of it's statistical validity will be undergone.

Chapter 2

Generalization in Supervised Machine Learning

Intuitively **Generalization in Supervised Machine Learning** (ML) can be seen as a models ability to apply its acquired knowledge to a previously unseen scenario. In this chapter we will go through the theoretical foundations of ML that led to the classical definition of generalization which will serve as groundwork for the new proposed metrics later in this work.

2.1 Statistical Learning Theory on PAC

Statistical learning theory (SLT) [Moh+12] can be understood as the formal background of ML based on the mathematical fields of statistics, measure theory and functional analysis [Lan93]. Probably approximately correct (PAC) [Evg+00] learning is a sub field of computational learning theory focused on the conditions in which a function can generalize a problem [Gol10].

2.1.1 Data and Probability Spaces

Any ML problem begins with a set of pairs (x, y) or dataset S from two spaces, the inputs $x \in X$ and the outputs $y \in Y$ both obtained from a random sampling [Tah16] S, i.e:

$$S := \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\} \tag{2.1}$$

We assume that our random sampling S comes from an underlying probability distribution on a probability space defined by the product space $D = X \times Y$ with a joint probability measure $\mu(x, y) = \mu(d)$ on D [KBH19]. In other words, the pairs $(x_1, y_1), ...(x_n, y_n)$

can be seen as independent and identically distributed (i.i.d.) [KBH19] samples from D through μ.

For the next steps we will build our results solely on the definitions of X, Y and our measure μ.

2.1.2 Hypothesis Space

The intended objective is to predict an outcome y from a target space Y of possible outcomes from given a set of features x in X, in other words to infer a function f that maps [BS51]:

$$f_\lambda : X \to Y \tag{2.2}$$

We define the *hypothesis space* $\mathcal{F}$ [Moh+12] as a space of all possible hypotheses or functions for mapping the feature space into the target space, normally constrained by a given structure.
The parameter λ in equation (2.2) can be understood as the variation between functions in $\mathcal{F}$, we will use it when in need to differentiate between different functions.

For example imagine we want to produce a polynomial to fit a given problem, the λ parameter is the vector of the coefficients of the polynomial.

2.1.3 Loss Function

In order to evaluate how a function $f_\lambda(x)$ fits a pair (x, y) we define a loss function or fitness function l as a mapping:

$$l\colon Y \times Y \to \mathbb{R}_+ \tag{2.3}$$

Intuitively it is used to verify how close two values in Y are, naturally $l(f(x), y) = 0$ is equivalent to a perfect prediction of f on the pair (x, y).

2.1.4 Learning Algorithm

Given a random sampling S, an *hypothesis space* $\mathcal{F}$ and a loss function l, a learning algorithm Λ is a function that maps:

$$\Lambda : S \times \mathcal{F} \to \mathcal{F} \tag{2.4}$$

Producing a new f_{λ^*} in order to minimize the loss function over the whole sample S.

Using our polynomial example, imagine we want to fit a polynomial to a given sample S , we can use Newton's Polynomial algorithm [CC12] to produce such an f_λ.

2.1.5 Statistical Risk

Until this point we have only seen loss or error point wise, but remember the objective of a learning algorithm is to produce a function $f \in \mathcal{F}$ that minimizes l in all of Y, there is a need to produce a global error for all pairs $(x, y) \in S$ and if possible over all pairs in D, given this setting the *statistical risk* or *error* of f over all the the underlying space is defined as:

$$L_D(f) := \mathbb{E}_\mu[l(f(X), Y)] = \int_D l(f)\, d\mu \tag{2.5}$$

Intuitively by integrating over D with respect to μ, in other words applying the expected value [KBH19], we average the error of all the pairs (x, y).

Naturally the learning algorithm's objective is to produce an f such that:

$$f = \underset{f_\lambda \in \mathcal{F}}{\operatorname{argmin}}\, \mathbb{E}_\mu[l(f_\lambda(X), Y)] \tag{2.6}$$

For every $x \in X$, let $\mu(y|x)$ with respect to x be the conditional probability measure on Y and μ_X be the marginal probability measure [KBH19] on X.

For most ML datasets it is safe to assume that X is a compact domain [Sem65] or a manifold [Ghr14] in the euclidean space (e.g. $X \subset \mathbb{R}^n$) and Y bounded, requirements needed for:

For every integrable function [Nel15] $\varphi : X \times Y \to \mathbb{R}_+$, Fubini's Theorem [CS] states that:

$$\int_{X\times Y} \varphi(x, y) d\mu = \int_X \left(\int_Y \varphi(x, y)\, d\mu(y|x) \right) d\mu_X \tag{2.7}$$

Which means that by taking into consideration the composite function $l \circ f : X \times Y \to \mathbb{R}_+$ we can:

$$\mathbb{E}_\mu[l(f)] = \int_X \left(\int_Y l(f(x), y))\, d\mu(y|x) \right) d\mu_X \tag{2.8}$$

Intuitively we are separating (in the integral) the spaces X and Y so that we can look at the domain D as the product of input domain X and target Y.

We then proceed to define $f_\mu : X \to Y$ as:

$$f_\mu(x) = \int_Y y\, d\mu(y|x) \tag{2.9}$$

The function f_μ is called the *regression function* [CS] of μ and from this construction it can be understood as the true input-output function reflecting the underlying distribution D.

Fixing $x \in X$ and considering the integrable composite function $l \circ f : X \times Y \to \mathbb{R}_+$, then to infer an optimum f it is sufficient to minimize our expected value solely on Y:

$$f = \underset{c \in \mathcal{F}}{\operatorname{argmin}} \int_Y l(c, y)\, d\mu(y|x) \tag{2.10}$$

Thus we'll consider the average measurement of *error* or *risk* of f as:

$$L_D(f) = \int_Y l(f, y)\, d\mu(y|x) \tag{2.11}$$

Note that while f_μ and μ are most likely unknown, $\mu(y|x)$ is known in some situations, more specifically remember the set S.

2.1.6 Bias Variance Trade off

Before going into a definition of generalization let's first analyse the particular case where the loss function is the mean square error (MSE) the expected value as in equation (2.7):

$$\mathbb{E}_\mu[l(f(X), Y)] = \int_X \left(\int_Y (y - f(x))^2\, d\mu(y|x) \right) d\mu_X \tag{2.12}$$

Calculations presented in section (A.0.1) of Appendix 1 lead to:

$$= \left(\int_X f(x) d\mu_X - \left(\int_Y y\, d\mu(y|x) \right) \right)^2 + \sigma_f^2 + \sigma_Y^2 \tag{2.13}$$

Where,

$$\sigma_f^2 = \int_X f(x)^2 d\mu_X - \left(\int_X f(x) d\mu_X \right)^2 \tag{2.14}$$

And,

$$\sigma_Y^2 = \int_Y y^2\, d\mu(y|x) - \left(\int_Y y\, d\mu(y|x) \right)^2 \tag{2.15}$$

Remember the definition of *regression function* in equation (2.9), we can also write as:

$$= \left(\mathbb{E}_X[f] - f_\mu \right)^2 + \sigma_f^2 + \sigma_Y^2 \tag{2.16}$$

Before we interpret term wise the above equation let's first introduce the notion of noise [Has+01] given by:

$$y = f_\mu(x) + e \tag{2.17}$$

Under the assumption that the noise $e \sim \mathbb{N}(0, \epsilon)$ and is independent. Intuitively this

means that the sampling methods used have some interference or error leading to an imperfect sampling.

Calculations presented in section (A.0.2) of Appendix 1 lead to:

$$\mathbb{E}_{\mu}[(f_{\mu}(X)+e-f(X))^2] = \left(\mathbb{E}_X[f]-f_{\mu}\right)^2 + \sigma_f^2 + \sigma_e^2 \tag{2.18}$$

As one can see the presence of noise only affects the last term of the equation (previously σ_Y^2) which is incredibly relevant as it does not necessarily affect the error associated with f maintaining the same type of relationship, so according to some papers [Nea19][Nea+18] we can conceptually define:

$$\left(\underbrace{\mathbb{E}_X[f]-f_{\mu}}_{\text{bias}}\right)^2 + \underbrace{\sigma_f^2}_{\text{variance}} + \underbrace{\sigma_e^2}_{\text{Irreducible error or Noise}} \tag{2.19}$$

1. *Error due to Bias*: The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict. Take into consideration that although we are only trying to produce a single model when referring expected or average prediction values we are talking about the average on the hypothesis space, for example imagine you could redo the learning algorithm more than once, each time you sample new data and run a new optimizer and producing a new predictive function f, due to variability in the underlying data, the resulting function will have a range of predictions. Bias measures how "wrong"in general these models' predictions are from the correct value. What is the inherent error that you obtain from choosing some hypothesis space even on the whole product space D? This is due to your hypothesis space being "biased"to a particular kind of solution.

2. *Error due to Variance*: The error due to variance is taken as the variability of a model prediction for a given pair (x, y). Again, imagine you can redo the learning algorithm more than once, the variance is how much the predictions for a given pair (x, y) vary between different functions in in hypothesis space. Captures how much your predictive function alters if you use a different training sample, in other words how 'over-specialized' is the predictive function to a particular random sampling..

3. *Error due to Noise or Irreducible error*: How big is the data-intrinsic error induced from bad sampling or pure "natural"noise? As the construction and name implies there no way to reduce this error it is an aspect of the sample.

Ideally the final function should be error free in any given situation, but as we can see in this section that is highly improbable, knowing that the error is most likely not null for any f_λ will play a big part in the next section.

2.2 Generalization Error

Now that we have gone through the basics of SLT we can now provide a solid definition of the classical notion of *generalization error*.

Remember that S is a set of pairs (x, y) of features and targets respectively and ideally our objective would be to find a function f, such that:

$$f : X \to Y \tag{2.20}$$

Where f minimizes $\mathbb{E}_\mu[l(f)]$ over $X \times Y$, nevertheless due to the impossibility of knowing the underlying distribution in D, we cannot calculate its expected value of f with respect to μ.

However from (2.11) we've seen that solving the expected value over Y also solves this problem so we minimize:

$$f = \underset{c \in \mathcal{F}}{\operatorname{argmin}} \int_Y l(c, y)\, d\mu(y|x) \tag{2.21}$$

Although it is sufficient to minimize the problem over Y the measure $d\mu(y|x)$ is only known for some pairs (x, y). From this we define the estimate *empirical loss* using our sample S:

$$L_S(f) := \frac{1}{n} \sum_{k=1}^{n} l(f(x_k), y_k) \tag{2.22}$$

Unlike before, by the error point wise we can calculate an *empirical loss* to infer a function f such that:

$$f = \underset{c \in \mathcal{F}}{\operatorname{argmin}}\ L_S(c) \tag{2.23}$$

Naturally since we are using an estimation we cannot guarantee that for every f we have $L_S(f) = L_D(f)$.

Due to this issue, classically the *generalization error* G of f is defined as the difference between the *empirical loss* and the loss over the underlying distribution D.

$$G(f) := L_D(f) - L_S(f) \tag{2.24}$$

An immediate problem arises since again, we have no way of calculating our loss over D, again we rely on another random sample S_{test} of D of unseen pairs (x,y) for an estimator loss called *estimation error*, so the calculable *generalization error* is given by:

$$\hat{G}(f) = L_{S_{test}}(f) - L_{S_{train}}(f) \tag{2.25}$$

We've changed the denomination of S to S_{train} to facilitate the distinction between both samples and also to align the names with the literature denominations.

2.3 Learning Models

In section (2.1.2) we've defined *hypothesis space* as usually being constrained to given structure, in this next section we will go through a couple of well known 'constrains' of hypothesis spaces, which we will call as *learning models* and some of their possible learning algorithms.

2.3.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a model that tries to approximate the structure and behaviour of the human brain cells. It is formed of connected artificial neurons, also known as nodes which can be understood as a graph.

The structure is composed of artificial neurons, each one can have several inputs but returns a single output which can be transmitted to other neurons. Neurons are sequentially clustered together in layers, where neurons in the same layer are not connected with each other, the cluster of inputs of the network is called *Input Layer*, while the outputs of the final cluster of output neurons of the architecture is called *Output Layer*, and all layers between the input and output layers are called *Hidden Layers*.

In particular each neuron takes the weighted sum of all of its inputs, in other words each input has a particular weight or trainable variable associated to it. There can be a bias term added to this sum. This weighted sum is then passed through a (usually nonlinear) activation function [Nwa+18] to produce the single output.

As stated in the book "Information Theory, Inference, and Learning Algorithms"[Mac03], in each time we describe a NN learning model we need to further specify at least two things:

- **Architecture**. The architecture specifies the structure, what variables are involved in the network and their topological relationships. In other words it defines a *hypothesis space* where the λ parameter is the vector of all trainable weights in the network.

- **Learning rule**. The learning rule specifies in which way the neural network's weights change. To put it formally a learning rule is the learning algorithm that chooses different weight configurations out of a fixed architecture. There are several learning algorithms commonly used to optimize NN architectures such as the gradient gradient descent [Mac03] based "sgd"[Zho+19] and "rmsprop"[Rud16].

As important remark is that the ANN model's by themselves only define a constrained set of possible *hypothesis spaces* (architectures), we define *hyperparameters* as the parameters that given a learning model define a hypothesis space (in this case an architecture). Naturally these are user defined and are unchanged during the learning algorithm which is often called optimizer.

Bellow is an example of an architecture:

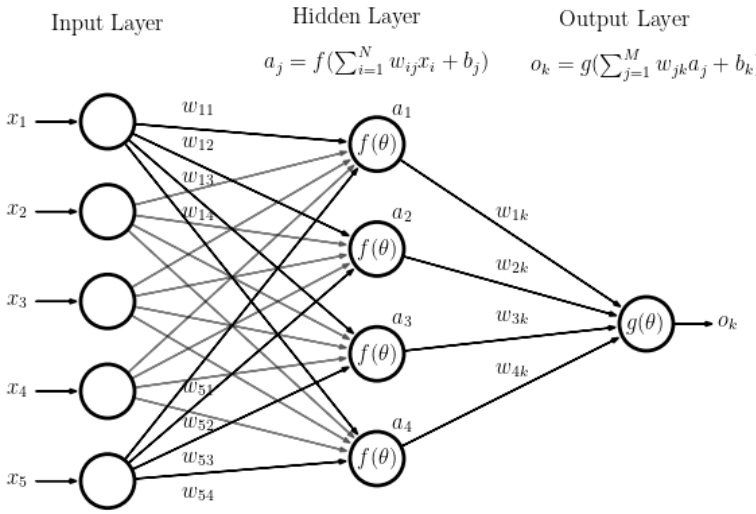


Figure 2.1: Example of a NN architecture

Online Learning [Mac03] is the procedure in which we predict, evaluate a pair $(x, y) \in S_{train}$ and proceed to apply the learning algorithm changing the weights vector.
Batch learning [Mac03], instead applying the learning algorithm for each pair's $(x, y) \in S_{train}$ evaluation, we cluster them with a fixed size (batch size), evaluate and only then apply the learning algorithm over an average evaluation.

ANN's are iterative, which means they execute their learning algorithm multiple times, we define an *epoch* when either online or batch learning as gone over all pairs in the training sample. So a common terminal condition is the number of epochs.

2.3.2 Genetic Programming

Genetic Programming GP is a sub-field in Evolutionary Computation. Its purpose if to evolve a set of individuals in a population-based procedure that follows the principles of Darwin's Theory of Evolution [Dar59].Each individual represents a program or tree-based structure [Wil+97] as shown in figure 2.2 bellow.

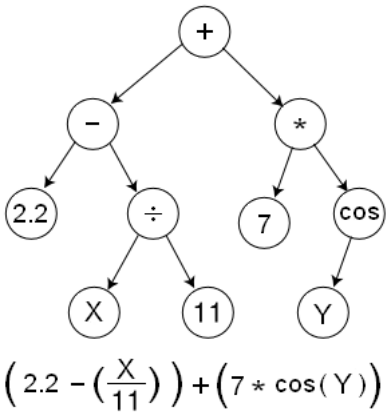


Figure 2.2: Example of a tree-based representation of a GP individual

In the particular case (determined from the figure), an individual is a tree with terminals= $\{X, Y\}$ (inputs) and $functions$ = {sum, difference, product, division, cosine, sum constant}. So the *hypothesis space* is the set of all possible combinations of trees constrained by the above defined terminals and functions. The parameter λ is the structure itself, where the hyperparameter as defined in the section above is the set $functions$.

The procedure can be summarized by the following steps:

1. Create Initial population (Initialization);
2. Calculate *Loss* or Fitness of each individual;
3. Until some termination condition is met repeat:
    - Select fitter individuals for reproduction (Competition);
    - Recombine between individuals (reproduction);
    - Variation of individuals (mutation);
    - Reevaluate loss;
4. Return fittest individual in the population;

Each iterative step or generation in point 3. is the learning algorithm and in the particular case of genetic programming has many variations and possibilities [CM19],

but intuitively evolution of the population begins with a set of random individuals from the hypothesis space. Then, by applying a selection operator we define the 'parent' individuals, offspring is generated through a recombination of the parents and after variation or mutation, transited to the next generation. This process iterates until reaching certain terminal condition (commonly a given number of iterations or generations).

2.4 Overfitting

If a learning model performs well on the training set $L_{S_{train}}(f) \simeq 0$ but generalizes badly $\hat{G}(f) >> 0$, we say it is *overfitting* [Has+01]. A learning model might overfit if the training/test sets are not well sampled, in other words S_{train} and S_{test} do not fully represent the same distribution. Another common reason is when a model only memorizes S_{train}, which means that if the model's number of trainable parameters (dimension of λ) is higher than the number of pairs (x, y) in S_{train} then there is a possibility that it stores the prediction values in those parameters instead of learning the problem, this particular issue is prone to happen if a learning model has a high number of iterations or epochs.

In the next subsections we will go through some commonly used methods to avoid *overfitting*.

2.4.1 Regularization

Regularization is a method to constrain our learning algorithm from producing an f that is too complex [LR15], which may therefore overfit, according to [Information Theory, Inference, and Learning Algorithms] it involves modifying the loss function in such a way as to incorporate a bias [Mac03] against the sorts of solutions which are undesirable.
Commonly complexity is associated with a large λ (for example a high sum of the weights in a neural network).

Take the L1 and L2 *regularizers*, they use the L1 and L2 metrics [NM18] respectively to enforce lower values of λ in the loss function, in particular using the MSE a L2 regularizer would alter the empirical error:

$$L_S(f) := \frac{1}{n}\left[\sum_{k=1}^{n}(y_k - f(x_k))^2 + \delta \sum_{b\in\lambda} b^2\right] \tag{2.26}$$

Where δ is called the *learning rate* [LR15] and the b's are the trainable parameters.

Another commonly used regularizer, for neural networks is the use of dropout [Sri+14]

in a particular layer, which ignores a subset of neurons in that layer with a set probability. In a way it forces the model to expand reducing interdependent learning among units, which may have led to overfitting.

2.4.2 Data Augmentation

The more pairs (x, y) the random sample S_{train} has, the higher the probability it fully represents the underlying space D. Which naturally would help the learning algorithm to better generalize [Won+16].

In some situations there is no possibility of gathering more data being constrained to the random sample S_{train}, although it is more common for classification problems, more in particular images, we can apply data augmentation to artificially increase the size of our random sample. Using the image classification example, we can perform various image transformations to each image creating "new"ones (e.g., flipping, rotating, re scaling, shifting).

2.4.3 Validation Set

Validation Set is a sub partition of the test set usually used to fine-tune the hyperparameters of a learning model (change the hypothesis set), there are some applications that use this sub partition to either prevent or reduce overfitting, some of them are presented bellow.

2.4.4 Early Stopping

In subsection (2.1.4) we stated that a learning algorithm has a terminal condition, normally this terminal condition is given by a predefined value on the training loss or due to computational limitation a certain number of iterations or epochs. In these situation there is no control or knowledge about the models generalization error.

Early Stopping can be understood as method to act as a terminal condition or an overwrite of the previous existing terminal condition in a learning model according to some particular measure (does not need to be the definition of generalization) over a different random sampling commonly called as *validation sample*.
Naturally the *early stopping* method cannot base itself on the test sample, in order to use the generalization error one of the requirements is a sample of unseen pairs which would have been corrupted if used in this method. A common implementation is to terminate training in cases where the training error is reducing while the validation's error increases (which suggests overfitting) with a given patience level to mitigate any natural fluctuation of the errors [Car+00].

2.4.5 Cross Validation

Cross-validation [Rod+10] is a widely used method to choose the value of a hyperparameter redefining an hypothesis space, that might be less prone to overfit. The method splits the sample S_{train} into k sets (k-fold cross-validation), for each k-fold each one of the sets will be the test sample while the others together make the train sample. The result is an average error over all k-folds using all of sample S creating a more robust model which should be less prone to overfitting, an immediate minus is that it is k times more computationally expensive.

Regarding the hyperparameter tuning, according to [A Bias Correction for the Minimum Error Rate in Cross-validation], by training and testing the model on separate subsets of the sample S, we achieve a notion of the model's prediction error as a function of the tuning parameter, a better method to choose a different parameter.

Chapter 3

Related Work

Throughout the years there have been several studies conducted to both grasp the essence of what it means to have a generalized f and also also to improve the generalization of learning algorithms on existing *hypothesis spaces*. Although most of the studies where based on classification problems, one can summarize them in the sections outlined below.

3.1 Train Test Distribution Disparity

As some papers state [Chu+18], most machine learning (ML) learning models assume that the random sampling for training has the same distribution as the test data in which the generalization error will be calculated. However, due to sample selection bias [LZ14] or, more broadly, covariate shift [IS15], there exist potential training examples that are completely unknown to the learning model. This discrepancy between training and testing samples leads to a low generalization performance of the learning algorithms and hence biased predictions (remember section 2.1.6).

3.2 Problem Complexity

Some studies [Ney+17] try to understand a models generalization capacity trough a study of the models robustness [Zha+18] and sensitivity [Set15].
While others [Gó+14] define a measure (originally defined in a Boolean target space) to quantify the complexity of data in relationship to the prediction accuracy that can be expected when using a supervised classifier like a neural network.

3.3 Over-parameterization Paradox

Both papers [Nov+18][BG18] explore the phenomenon, that in practice it is often found that complex over-parameterized neural networks architectures have a lower generalization error when compared with their smaller counterparts, an observation that appears to conflict with classical notions of function complexity, which typically favor smaller models.

Chapter 4

A Generalization Metric

The initial conjecture in section (2.1.1) assumes that S is an i.i.d sample of an underlying distribution defined by $D = X \times Y$, from that a definition of *generalization* is constructed on the difference between the training sample's with an unseen sample's S_{test} empirical loss.

Assuming that S_{test} was sampled from the same exact conditions as S is in many situations is not true and raises two immediate problems:

1. Difficulty of interpretation, using the common examples of underfit and overfit in both states of poor *generalization* one cannot accurately understand the difference in metric between them;

2. The above definition dissolves all the training information and does not capture the relation between the sample used for training and the one used for testing. If the distribution of both is sufficiently diverse, the model will always under perform on S_{test} although this shouldn't be seen as an error in generalization but as a result of sub optimal sampling. Meaning S_{test} is not a good representation of D.

These assumptions are both hindering, since an interpretable metric of a models generalization given its training is more than desirable, and also wrong, since more often than not real world data is full of noise and wrong observations.

We propose that it is more reasonable to weight the empirical loss over S_{test} with regard to the probability of each $(x_i, y_i) \in S_{test}$ belonging to D. In other words, we are not proposing a new *loss function* as defined in section 2.3, we are simply stating

that the definition of *generalization* should **not** be the difference between the empirical losses of two samples.

4.1 Probability Score

As stated in (chapter 2), μ or in this case $p(x,y)$ cannot be known, but under the assumption that a learning algorithm can also learn the underlying distribution of the training set S [Kim+19] we can infer p based on a kernel density estimation (KDE) [Weg18]. Thus for each $(x,y) \in S_{test}$ we approximate $p(x,y)$ by $p^*(x,y)$:

$$p^*(x_0,y_0) = \sum_{(x,y)\in S} \exp\left(-\frac{|l(f(x_0),y_0) - l(f(x),y)|}{1+\|(x_0,y_0)-(x,y)\|}\right) \tag{4.1}$$

Note that basing this probability estimate on the assumption that a learning algorithm can learn the underlying distribution of the training set denies the possibility of using it during any learning step.

We define *Probability Score* as:

$$P_f(S_{test}) = \frac{1}{n} \sum_{(x,y)\in S_{test}} p^*(x_0,y_0) \tag{4.2}$$

Through some initial empirical experiments we have reason to believe that this score can be used as a learning algorithm's *stopping condition* in order to prevent overfit.

4.2 Generalization Score

We define a new error over S_{test} as:

$$L^*_{S_{test}}(f) = \frac{1}{|S_{test}|} \sum_{(x,y)\in S_{test}} l(f(x),y)p^*(x,y) \tag{4.3}$$

An immediate result that comes from (4.1) is the comparability it offers with the empirical loss over S, ideally if L_{test} was well sampled and f 'understood' the underlying distribution of S_{train} then:

$$\forall (x,y) \in S_{test} : \quad p^*(x,y) = 1 \tag{4.4}$$

Where $L_{S_{test}}(f)$ becomes the classical empirical error over S_{test}. The new generalization metric is:

$$G(f) = L_{S_{train}}(f) - L^*_{S_{test}}(f) \tag{4.5}$$

Where $L_{S_{train}}(f)$ is the usual empirical error of f over S_{train}

Chapter 5

A New Loss Function

The construction of the new loss function l^* is based on the assumption that given some random sample of pairs $(x, y) \in S_{train}$ that come from an underlying distribution in D, it is always possible to fit a function f such that $L_{S_{train}}(f) = 0$.

But, in the presence of noise or bias we have:

$$y = f(x) + \epsilon \quad : \quad (x, y) \in S \wedge \epsilon \in \mathbb{R} \tag{5.1}$$

Under the assumption that the noise $e \sim \mathbb{N}(0, \epsilon)$ and is independent.

So a direct approximation metric, such as the mean absolute error (MAE) is not optimal as it would always have error induced by the existence of noise (section 2.1.6) leading to a poor performance on unseen instances.

Intuitively the idea is that under the assumption that the set of $\xi = \{\epsilon_i\}_{i=1,..,n}$ comes from a underlying distribution independent to Y of expected value 0, by sub sampling S_{train} we can infer for each pair $(x, y) \in S_{train}$ a *correction ratio* in order to mitigate the error induced be noise.

5.1 Nearest Neighbors

For each pair $(x, y) \in S_{train}$ we use a k-nearest neighbors algorithm [AL16] a simple method that returns the set of k closest pairs in S_{train} denoted as $N_\alpha(x, y)$ with respect to the euclidean distance. So the only **user defined parameter** is the number of nearest neighbors to use.

Intuitively the set $\{(x,y), N_\alpha(x,y)\}$ defines a sub distribution centered around an unbiased pair, remember that $\mathbb{E}(\xi) = 0$ and is independent.

5.2 Density Centroids

A centroid is defined as the arithmetic mean of the points but can also be seen as a minimizer of the average distance of a center object to a given collection of objects.

For each pair $(x,y) \in S_{train}$ we find its respective set of α nearest neighbors. We then proceed to calculate it's centroid defined as:

$$centroid_\alpha(x,y) = \frac{1}{\alpha} \sum_{(x_k,y_k)\in N_\alpha(x,y)} (x_k, y_k) \tag{5.2}$$

Intuitively each pair (x,y) defines a ball centered around $centroid_\alpha(x,y)$ with radius $\|(x,y) - centroid_\alpha(x,y)\|$ [Xin+03] denoted as $B_{centroid_\alpha(x,y)}(x,y)$.

5.3 Correction Ratio

Now that we have a ball $B_{centroid_\alpha(x,y)}(x,y)$ for each $(x,y) \in S_{train}$ given each prediction $f(x)$ there is the need to correct an unbiased position of the pair $(x, f(x))$.
We define the *correction ratio* as :

$$cr(x,y) = \frac{\|(x,f(x)) - centroid_\alpha(x,y)\|}{\|(x,y) - centroid_\alpha(x,y)\|} \tag{5.3}$$

Intuitively the closest the pair $(x, f(x))$ is to the centroid center the more unbiased it will be, on the other hand it also accounts the proximity to the intended target.

5.4 Loss Function

The evaluation of f for each point $(x,y) \in S_{train}$ will be it's previous loss function (for example the Mean Absolute Error) multiplied by the *correction ratio*:

$$l^*(f(x),y) = l(f(x),y) \times cr(x,y) \tag{5.4}$$

Intuitively we are correcting each loss values respectively in order to mitigate the effect of noise in the training sample S_{train} .

Finally the empirical error of S_{train} is given by:

$$L^*_{S_{train}}(f) = \frac{1}{|S_{train}|} \sum_{(x,y)\in S_{train}} l^*(f(x),y) \tag{5.5}$$

5.5 An Example

S_{train} is the set of pairs (x, y) where $x \in \mathcal{X}$ is a set of 100 points uniformly distributed in $[0, 2\pi]$ and $y = sin(x)$, we proceed to add some noise from a normal distribution $\mathcal{N}(0, 0.1)$.

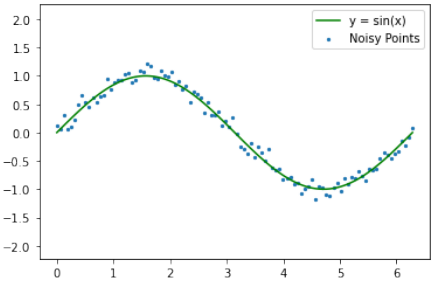


Figure 5.1: True f and noisy points

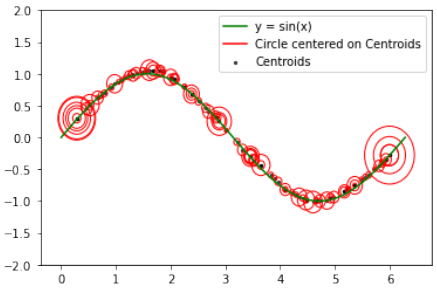


Figure 5.2: Visual Construction of the loss function

As one can see in the figure predictions $(x, f(x))$ such that:

$$(x, f(x)) \in B_{centroid_\alpha(x,y)}(x, y) \tag{5.6}$$

Then,

$$\frac{\|(x_k, f(x_k)) - centroid_\alpha(x_k, y_k)\|}{\|(x_k, y_k) - centroid_\alpha(x_k, y_k)\|} < 1 \tag{5.7}$$

Which means it will have a lower evaluation when compared with the absolute difference between y and $f(x)$, analogy if the pair does not belong to said ball it will have comparatively a higher evaluation.

Chapter 6

Methodology

The objective of this experiment is to understand if the use of the proposed loss function with a given number of nearest neighbors as introduced in the previous chapter, increases generalization. In order to achieve fair results a broad benchmark on several problems, with different learning models needs to be performed in comparison to a standard or control loss function, naturally in the same conditions. Variation for statistical validation will be introduced through the repetition of benchmarks using 30 different seeds in the partition of the problem set into Train and Test.

6.1 Proposed Approach

6.1.1 Control

In supervised regression problems the standardly used loss function is either the (Mean Square Error) or the Absolute Square Error (MAE) [WM05]. Taking into consideration the definition of our custom loss function, we chose that the control loss function to use as comparison should be the MAE. So for this study we will compare the use of the loss functions bellow defined:

$$l_{custom}(f(x), y) = |y - f(x)| \times cr(x, y) \tag{6.1}$$

$$l_{control}(f(x), y) = |y - f(x)| \tag{6.2}$$

6.1.2 Problem Set

For this experiment, nine real-life symbolic regression problems or datasets were used. The Bioavailability, Plasma Protein Binding level (PPB), and Toxicity are problems from the drug discovery area and their objective is to predict the value of a pharmacokinetic parameter, as a function of a set of molecular descriptors of potential new

drugs. The Concrete datasets, objective is to predict the concrete strength as a function of some observable characteristics of the material, the Energy dataset objective is to predict the energy consumption in particular geographic areas and in particular days, as a function of some observable features of those days, including meteorologic data, the Istanbul dataset objective is to predict returns of the Istanbul Stock Exchange as a function of seven other international indexes, the Park Motor dataset objective is to predict the severity of the motor symptoms of Parkinson's Disease patients, according to the Unified Parkinson's Disease Rating Scale assessment, the Park Total dataset objective is to predict the severity of the general Parkinson's Disease symptoms, according to the Unified Parkinson's Disease Rating Scale assessment and the Residential Build Sale Price (Residential) dataset objective is to predict the sale price of a set of residential buildings, using data that include construction cost, project variables, and economic variables.

Table (6.1) reports, for each one of these problems, the number of features (variables) and the number of data points (observations) in the respective datasets. The table also reports a bibliographic reference for most of the datasets, where the reader can find a more detailed description of these problems.

Dataset	**# Features**	**# Data Points**
Bioavailability [Arc+07]	241	206
Concrete [MCS13]	8	1029
Energy [MCP15]	8	768
Istanbul [OAB14]	7	536
Protein Plasma Binding Level (PPB) [Arc+07]	626	131
Park Motor	18	5875
RESID BUILD SALE PRICE (Residential)	107	372
Toxicity [Arc+07]	626	234
Park Total	18	5875

Table 6.1: Problem Set Description

6.1.3 Learning Models

Under the natural assumption that the proposed method should be *learning model agnostic* we will produce results using two very different learning models:

1. **Artificial Neural Networks** for their complexity but also the highly analytical learning algorithms that can be used.

2. **Genetic Programming** as a highly stochastic learning model it does not use in any form *gradient descent* which might provide some fresh insights on comparison with ANN's.

6.1.4 Evaluation Metrics

The objective of this experiment is to test how well our proposed loss function generalizes in comparison with commonly used metrics. Due to the definition of the loss function itself we cannot use the generally accept difference between train and test to compare between the control and custom loss functions.

Having this in mind we will evaluate of both control and custom models, at each generation or epoch the test samples using the standard Mean Absolute Error metric on the Test or unseen sample:

$$MAE_f(S_{test}) = \frac{1}{|S|} \sum_{(x,y) \in S_{test}} |y - f(x)| \tag{6.3}$$

To compare results between the control and custom models.

6.2 Pipeline

In order to provide standard conditions for all benchmarks, we define the following ordered pipeline. All of the benchmarks were coded in Python [VRD09].

6.2.1 Train Test Split

Naturally we only have one sample which is the problem set, a partition of the sample into a S_{train} and S_{test} is required, bellow is the code that executes such partition with a given random state or seed of the dataset:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
            train_test_split(X, y, test_size=0.3, random_state=seed)
```

We will be using the standard test size of 0.3, which represents a 70% to 30% ratio between the two Train and Test samples. The random state or seed ensures that at each run the random samples will be different, but reproducible.

6.2.2 Normalization

After the partition, even though it is common practice to normalize the data independently, in our particular case it is absolutely necessary since our loss function is based on distances, so we will be using the classical normalization $N(S)$:

$$\forall s \in S : N(s) = \frac{s - E[S]}{STD[S]} \tag{6.4}$$

Where,

$$E[S] = \frac{1}{|S|} \sum_{s \in S} s \tag{6.5}$$

And,

$$STD[S] = \left(\frac{1}{|S|} \sum_{s \in S} (s - E[S])^2 \right)^{\frac{1}{2}} \tag{6.6}$$

Below is the python code used to normalize both samples:

```
from sklearn.preprocessing import StandardScaler

X_train = StandardScaler().fit_transform(X_train)
y_train = StandardScaler().fit_transform(y_train.reshape(-1, 1))

X_test = StandardScaler().fit_transform(X_test)
y_test = StandardScaler().fit_transform(y_test.reshape(-1, 1))
```

6.2.3 Hypothesis Set Definition

6.2.3.1 Artificial Neural Networks using Keras Tensorflow

Keras [Cho+15] is a deep learning API written in Python, running on top of the machine learning platform TensorFlow.

As for the Hypothesis Set definition since Neural Networks architecture, learning rules, etc... can vary a lot depending on the problem set, to provide several levels of complexity we will use the following architectures to benchmark:

- **Hypothesis Space 1** : Input layer will naturally depend on the problem set, as for the **hidden layers we'll use 4 with 4 neurons each**, with activation function relu [Aga18] in all of them, the output layer is a single neuron with no activation function (linear).
- **Hypothesis Space 2** : Input layer will naturally depend on the problem set, as for the **hidden layers we'll use 8 with 8 neurons each**, with activation function relu in all of them, the output layer is a single neuron with no activation function (linear).
- **Hypothesis Space 3** : Input layer will naturally depend on the problem set, as for the **hidden layers we'll use 8 with 16 neurons each**, with activation function relu in all of them, the output layer is a single neuron with no activation function (linear).

As for the learning rule or optimizer we will use the standard 'sgd' [Zho+19], also with a batch size [GG18] of 32 units for all of the spaces.

6.2.3.2 Genetic Programming in Python

We will use the gplearn [Wwwa] python library SymbolicRegressor, the standard gp function for regression problems.

For the hypothesis set definition or hyperarameterization we will use the **default parameters** of the function itself stated in the documentation.

6.2.4 Terminal Condition

As you might have noticed we have not defined any terminal condition or early stopping yet. Overfitting can be understood as the degeneration of the test error, having that in mind we'll run each benchmark for **500 epochs** on the ANN and **500 generations** for the GP in order to evaluate the error progression or landscape as well.

6.3 Results Structure

For both control and custom loss functions, on each problem set, on each learning model (Neural Networks define various Hypothesis Set) for 500 iterations with two different numbers of nearest neighbors (20 and 40) on 30 different seeds or Train-Test samplings each. For comparative purposes we will use the median over unseen data (Test sampling) of each of the 30 seeds, later performing statistical validation tests.

Chapter 7

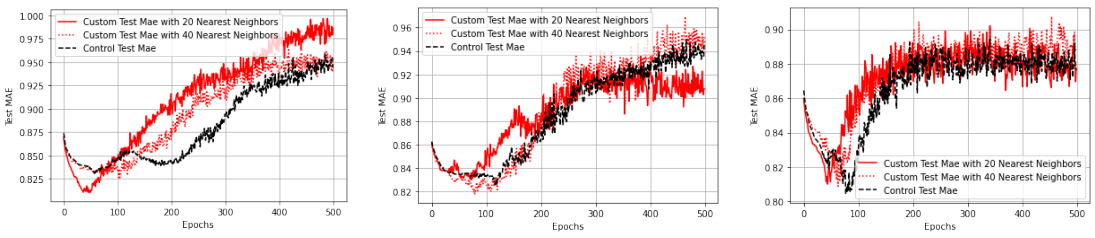
Experimental Results

7.1 Overview

In this section we will present the results obtained from the experimental pipeline defined in the previous chapter.

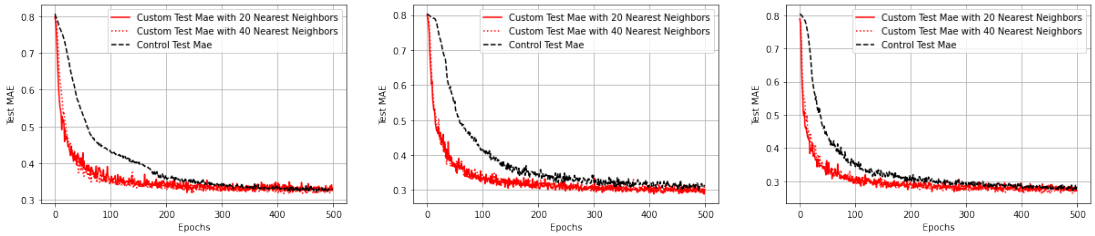
7.1.1 Artificial Neural Networks Benchmarks

The obtained results from the Artificial Neural Networks benchmarks are presented in Figures 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9 (respectively for the Bioavailability, Concrete, Energy and Istanbul, PPB, Park Motor, Residential, Toxicity and Park Total datasets). Each one of the sub figures represents a different Hypothesis Space showing the Mean Absolute Error of the test sample over all 500 epochs for the control loss function and both custom loss functions with 20 and 40 nearest neighbors.



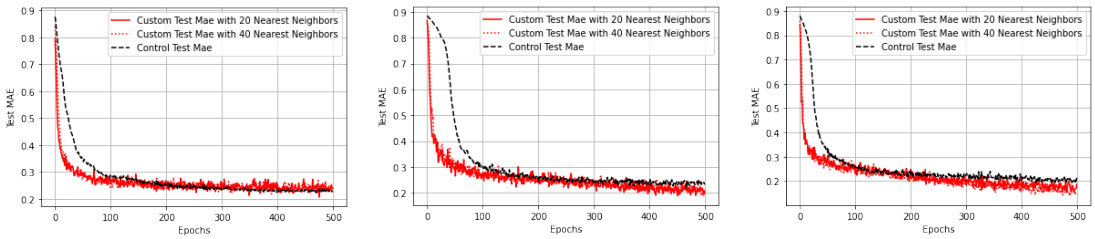
NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.1: Median Test Mae over 30 seeds on Bioavailability



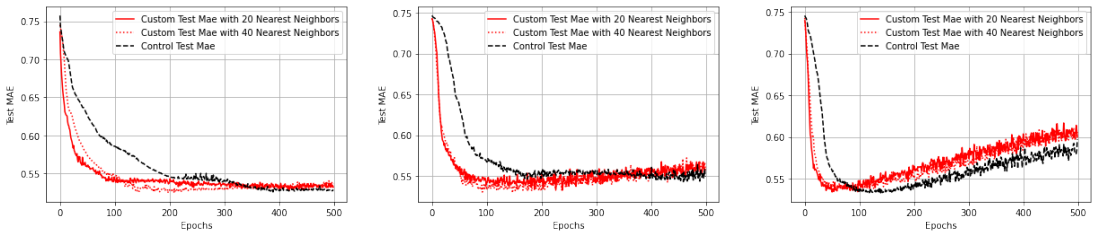
NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.2: Median Test Mae over 30 seeds on Concrete



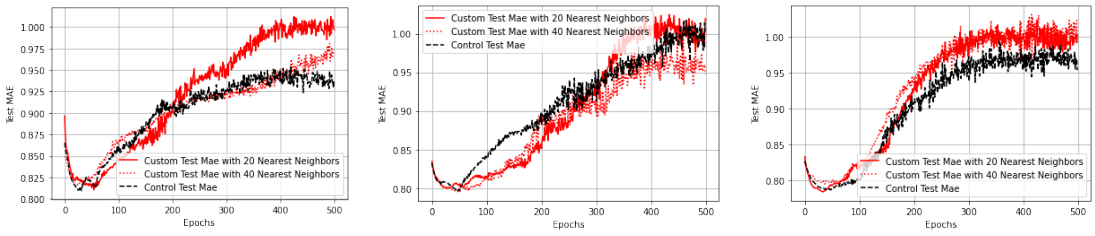
NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.3: Median Test Mae over 30 seeds on Energy



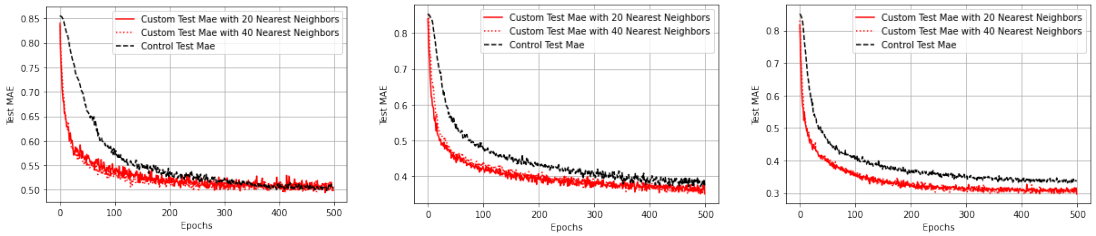
NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.4: Median Test Mae over 30 seeds on Instanbul



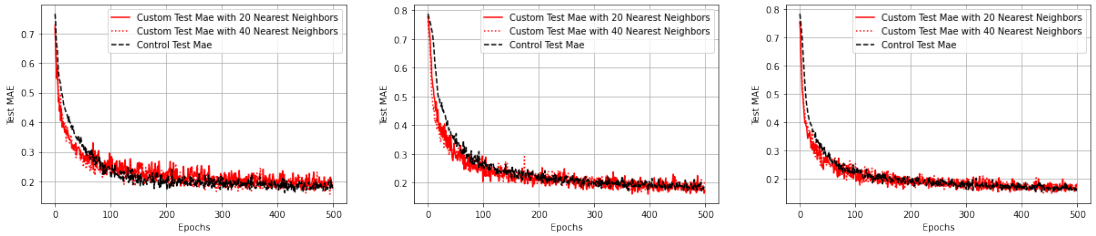
NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.5: Median Test Mae over 30 seeds on PPB



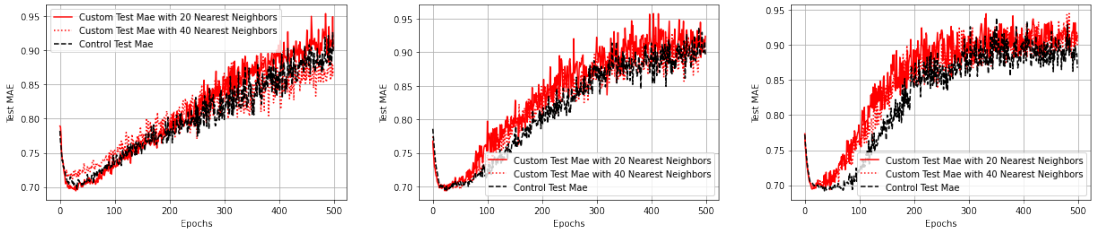
NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.6: Median Test Mae over 30 seeds on Park Motor



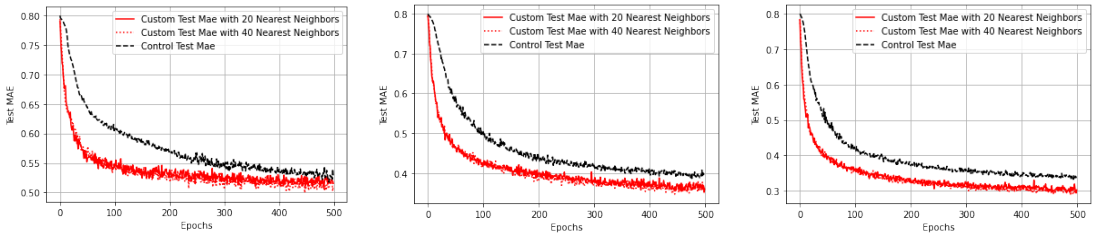
NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.7: Median Test Mae over 30 seeds on Residential



NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.8: Median Test Mae over 30 seeds on Toxicity



NN Hypothesis Space 1 NN Hypothesis Space 2 NN Hypothesis Space 3

Figure 7.9: Median Test Mae over 30 seeds on Park Total

7.1.2 Genetic Programming Benchmarks

The obtained results from the genetic programming benchmarks are presented in Figures 7.10, 7.11, 7.12, 7.13, 7.14, 7.15, 7.16, 7.17, 7.18 (respectively for the Bioavailability, Concrete, Energy and Istanbul, PPB, Park Motor, Residential, Toxicity and Park

Total datasets). Showing the Mean Absolute Error of the test sample over all 500 generations for the control loss function and both custom loss functions with 20 and 40 nearest neighbors.

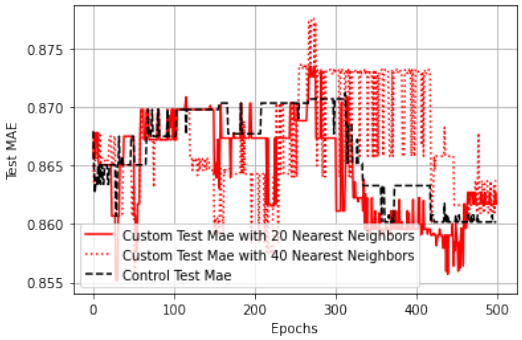


Figure 7.10: GP Median Test Mae over 30 seeds on Bioavailability

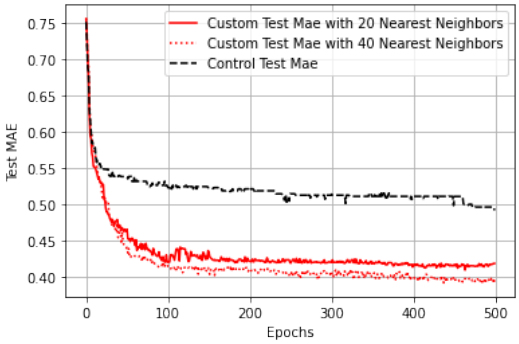


Figure 7.11: GP Median Test Mae over 30 seeds on Concrete

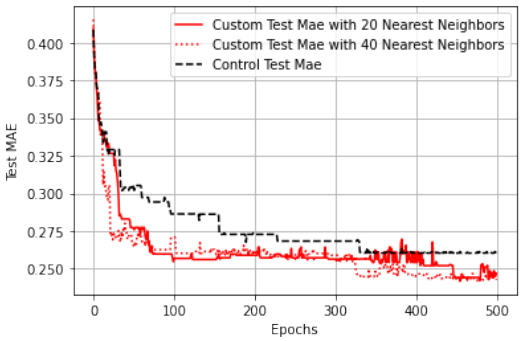


Figure 7.12: GP Median Test Mae over 30 seeds on Energy

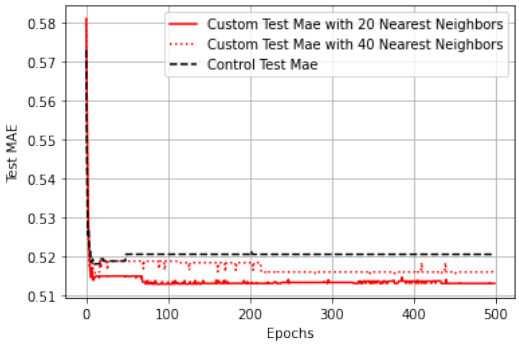


Figure 7.13: GP Median Test Mae over 30 seeds on Instanbul

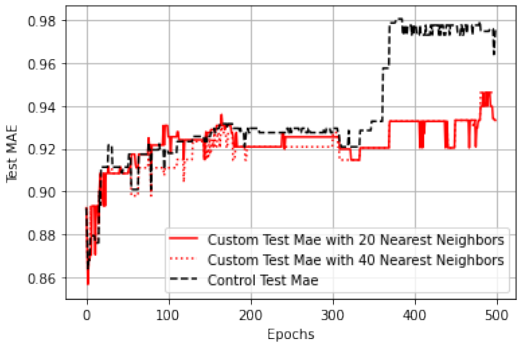


Figure 7.14: GP Median Test Mae over 30 seeds on PPB

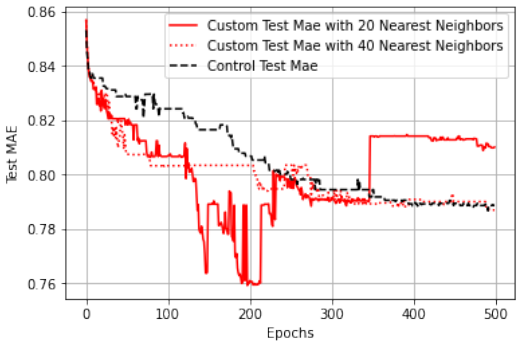


Figure 7.15: GP Median Test Mae over 30 seeds on Park Motor

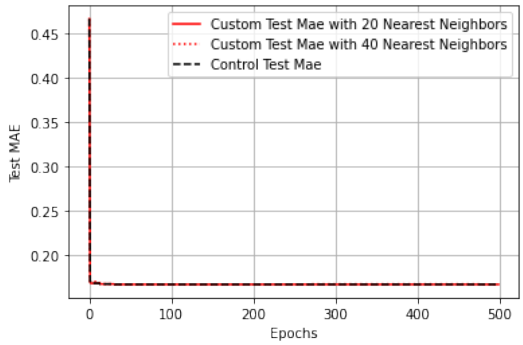


Figure 7.16: GP Median Test Mae over 30 seeds on Residential

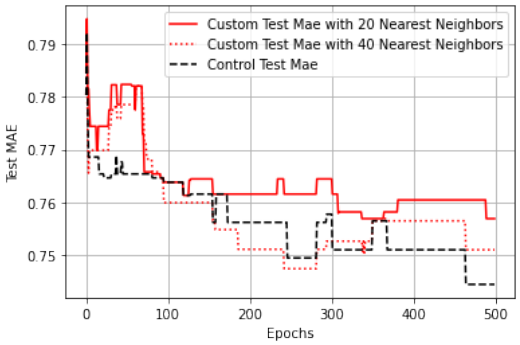


Figure 7.17: GP Median Test Mae over 30 seeds on Toxicity

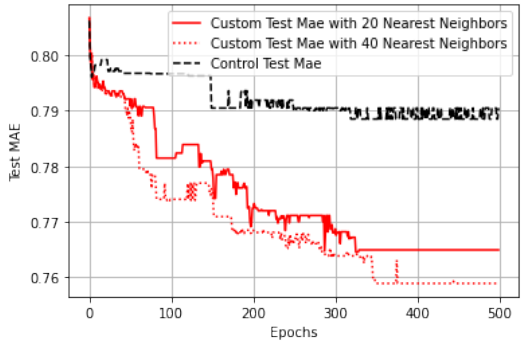


Figure 7.18: GP Median Test Mae over 30 seeds on Park Total

7.2 Statistical Validation

Since the intended purpose was analyse the behaviour of our custom loss function over the training procedure we ended up not using any stopping condition besides epochs or generation number, naturally to proceed with any statistical testing we need to choose a particular iteration.

For the ANN benchmarks, as one can clearly see the main difference between custom and control seems to be its *convergence rate* (number of iterations required to stabilize), with that in mind we'll perform statistical tests for the epochs **50; 100; 150; 200; 500** in order to better understand their validity. Regarding the GP benchmarks this trend is not so clear so we will only test for the last generation **500**.

A Kolmogorov-Smirnov test [Kol] has shown that the results do not come from a normal distribution and thus a rank-based statistic [RN11] has to be used. Bellow there a tables showing the results of the Wilcoxon rank-sum test for pairwise data [Wwwb] comparison under the null hypothesis that the medians between custom and

control results are equal. In the tables below we show the difference in the median MAE values of the test samples between custom and control where entries are in bold if their respective p-values are below .05, in other words they reject the null hypothesis with 95 % degree of confidence.

	20 Nearest Neighbors			40 Nearest Neighbors		
Dataset	H. 1	H. 2	H. 3	H. 1	H. 2	H. 3
Bioavailability	-0.0263	0.0023	-0.0146	-0.0017	-0.003	0.0035
Concrete	**-0.1297**	**-0.172**	**-0.0871**	**-0.1478**	**-0.1827**	**-0.1134**
Energy	**-0.052**	**-0.1946**	**-0.0528**	**-0.0794**	**-0.1794**	**-0.0713**
Instanbul	**-0.0692**	**-0.0835**	**-0.0229**	**-0.0528**	**-0.0856**	**-0.021**
PPB	-0.0013	0.0063	0.0055	0.005	0.001	0.0108
Park Motor	**-0.0888**	**-0.0918**	**-0.0696**	**-0.1126**	**-0.0848**	**-0.0655**
Residential	-0.0254	-0.0381	-0.0053	-0.0374	**-0.0389**	-0.0313
Toxicity	-0.0019	0.0111	0.0105	0.0177	-0.004	0.0157
Park Total	**-0.0933**	**-0.1142**	**-0.1104**	**-0.0787**	**-0.1165**	**-0.1064**

Table 7.1: Difference between Custom and Control Test MAE with Wilcox Rank-Sum test p-values on ANN Results on Epoch **50**

	20 Nearest Neighbors			40 Nearest Neighbors		
Dataset	H. 1	H. 2	H. 3	H. 1	H. 2	H. 3
Bioavailability	-0.0009	0.0112	0.0387	-0.0079	-0.005	0.0308
Concrete	**-0.0762**	**-0.0849**	**-0.0563**	**-0.0829**	**-0.0761**	**-0.0539**
Energy	**-0.0256**	**-0.0348**	-0.0091	-0.0448	**-0.0302**	-0.0141
Instanbul	**-0.0408**	**-0.029**	0.01	**-0.0404**	**-0.0293**	0.0097
PPB	-0.0081	-0.0241	0.0151	0.0094	-0.0272	0.0241
Park Motor	**-0.0488**	**-0.0601**	**-0.0438**	**-0.0458**	**-0.061**	**-0.0454**
Residential	0.0237	**-0.0679**	0.008	-0.0078	-0.0229	-0.0077
Toxicity	0.0012	0.037	**0.0675**	0.0176	0.0252	**0.0438**
Park Total	**-0.0567**	**-0.0797**	**-0.0514**	**-0.0581**	**-0.0736**	**-0.0585**

Table 7.2: Difference between Custom and Control Test MAE with Wilcox Rank-Sum test p-values on ANN Results on Epoch **100**

	20 Nearest Neighbors			40 Nearest Neighbors		
Dataset	H. 1	H. 2	H. 3	H. 1	H. 2	H. 3
Bioavailability	0.031	0.0293	-0.007	0.0051	-0.0043	0.0006
Concrete	**-0.0505**	**-0.0405**	**-0.0277**	**-0.05**	**-0.0241**	-0.0171
Energy	0.0007	-0.005	-0.021	-0.0168	**-0.0279**	-0.0031
Instanbul	**-0.0225**	-0.0136	0.0121	**-0.0345**	-0.0191	0.0107
PPB	-0.0151	-0.0375	-0.0259	-0.0016	-0.0355	0.0083
Park Motor	**-0.0326**	**-0.0399**	**-0.0361**	**-0.0344**	**-0.0281**	**-0.0415**
Residential	**0.0516**	-0.0099	**0.0345**	0.0387	-0.0393	**0.0424**
Toxicity	0.0134	**0.0253**	**0.0655**	0.0274	**0.041**	**0.0582**
Park Total	**-0.0521**	**-0.0571**	**-0.062**	**-0.0648**	**-0.0556**	**-0.0666**

Table 7.3: Difference between Custom and Control Test MAE with Wilcox Rank-Sum test p-values on ANN Results on Epoch **150**

	20 Nearest Neighbors			40 Nearest Neighbors		
Dataset	H. 1	H. 2	H. 3	H. 1	H. 2	H. 3
Bioavailability	**0.0731**	0.0219	0.0175	0.024	0.0106	0.0019
Concrete	**-0.0298**	**-0.0229**	-0.0071	**-0.0266**	**-0.0335**	-0.0133
Energy	0.019	-0.0098	-0.0273	0.0043	-0.0164	-0.019
Instanbul	-0.0086	-0.0032	**0.0193**	**-0.0195**	-0.0138	0.0095
PPB	-0.007	-0.0416	0.0167	-0.0029	-0.0079	0.03
Park Motor	0.0002	**-0.0274**	**-0.0445**	-0.0082	**-0.0291**	**-0.0435**
Residential	0.0346	-0.0232	0.0044	0.0172	-0.017	0.0065
Toxicity	-0.0034	0.0264	0.0509	0.0084	0.0429	0.0221
Park Total	**-0.044**	**-0.045**	**-0.0486**	**-0.0404**	**-0.0549**	**-0.046**

Table 7.4: Difference between Custom and Control Test MAE with Wilcox Rank-Sum test p-values on ANN Results on Epoch **200**

	20 Nearest Neighbors			40 Nearest Neighbors		
Dataset	H. 1	H. 2	H. 3	H. 1	H. 2	H. 3
Bioavailability	0.0425	-0.0404	0.0007	-1e-04	0.0038	0.0118
Concrete	0.0004	-0.013	**-0.0114**	0.0031	-0.0201	-0.0084
Energy	0.0129	-0.024	-0.0267	0.0058	**-0.0273**	**-0.0562**
Instanbul	0.0032	0.0089	0.0028	0.0074	0.0109	0.0089
PPB	0.0601	0.02	0.0489	0.0359	-0.0515	0.0353
Park Motor	-0.0071	**-0.0238**	**-0.0392**	0.0005	-0.0044	**-0.0396**
Residential	0.0105	-0.0093	0.0163	0.0065	0.0199	-0.0071
Toxicity	0.0042	0.0269	0.0462	-0.0354	0.0077	0.0306
Park Total	-0.0122	**-0.0334**	**-0.035**	**-0.016**	**-0.0272**	**-0.0394**

Table 7.5: Difference between Custom and Control Test MAE with Wilcox Rank-Sum test p-values on ANN Results on Epoch **500**

And now for the GP results:

Dataset	20 Nearest Neighbors	40 Nearest Neighbors
Bioavailability	0.0015	0.0035
Concrete	**-0.0729**	**-0.0951**
Energy	-0.0132	-0.0174
Instanbul	-0.0075	-0.0045
PPB	-0.0427	-0.0297
Park Motor	0.0215	-0.0017
Residential	0.0	0.0
Toxicity	0.0124	0.0065
Park Total	-0.0255	-0.0316

Table 7.6: Difference between Custom and Control Test MAE with Wilcox Rank-Sum test p-values on GP Results on Epoch **500**

Chapter 8

Discussion

8.1 Interpretation

The observed results are very satisfactory. Indeed both custom and control loss functions generally converge to the same loss values, nevertheless the fact the custom loss function converges much faster is by itself a significant victory in terms of generalization.

Remember section (2.1.6), we can see a function's error lower boundary solely dependent on the chosen hypothesis space and the irreducible error separably. These particular lower boundaries cannot be overcome by a new loss function, which means that converging faster translates in a better generalization.

The following section provides an interpretation of the previously observed results from chapter 7, in order to explain said results, we will analyse the main difference between both learning models, their hypothesis sets as well as the difference between using a different number of nearest neighbors for the custom loss function.

As one can see in figures (7.1), (7.5) and (7.8) the ANN model almost immediately started to degenerate it's test loss in the Bioavalability, PPB and Toxicity datasets for all three hypothesis spaces where we can't infer much about the degeneration itself. Looking at the same datasets on the GP learning model, there is hardly any degeneration but again we cannot infer anything about these benchmarks. Difficulty in learning these datasets is no surprise, in table 6.1 one can see that they are the only problem sets that have more features than instances naturally making the problem much harder to solve.

The ANN model on the Instanbul dataset on figure (7.4) provides a very good example of the possible problems that can arise from increasing the complexity of a hypothesis space, notice how changing to more complex spaces actually translates in the models test error degeneration, nevertheless in the first two hypothesis spaces both parameterizations of the custom loss function have much higher and statistically significant convergence rate compared with control, regarding hypothesis space 3 we have some very curious results, where if we were to align the three training procedures test MAE minimum in the same epoch they would practically match in the remaining of the training procedure, but again the custom loss is much faster. Regarding the GP benchmark on Instanbul in figure (7.13) we can actually see some small differences but they are not statistically significant.

In the remaining datasets (Concrete, Energy, Park Motor, Residual and Park Total) the ANN model actually learned consistently, the results overall converged to similar error comparatively, with the exception of the two 'Park' datasets which curiously were the ones that had a much higher instances by features ratio (5875 by 18) where a higher complexity meant a even higher negative difference between terminal median values. However in all of them there is a consistent and very significantly higher convergence rate when compared to control where a higher complexity translated in a less accentuated convergence rate.

Regarding the GP model there was only one statistically significant benchmark in figure 7.11 on the Concrete Dataset, where the custom loss in both parameterizations over performed against control by a wide margin. Regarding the remaining datasets on GP we can see some differences but again not statistically significant.

The differences in parameterizations between loss functions with 20 and 40 nearest neighbors were hardly ever noticeable but again only two were used.

8.2 Advantages and Shortcomings

As we have seen there are very strong evidences that this new loss function can actually perform with a higher convergence ratio against a comparable control loss and in some cases even outperform its convergence error, however the situations where it happens are still unknown. Additionally due to its definition it is slightly more computationally costly, although not much it has its effects.

Chapter 9

Conclusion

9.1 Summary

We started this thesis with the purpose of further understanding the notion of generalization in supervised machine learning, providing a better generalization metric and a new loss function to improve generalization.

Going into the basic definitions of Statistical Learning Theory allowed us to understand the formal background behind the classical definition of generalization error which enabled us to define a generalization score based on an estimation of the unknown underlying probability measure μ which given its definition is more suited to quantify generalization in sub optimally sampled S_{train} and S_{test} samples.

All of these notions served as inspiration for a new loss function intended to better perform given the underlying error induced by a noisy sampling of S, an error formally defined in the Bias Variance Trade Off section (2.1.6).

The results are indeed very promising and confirms that this new loss function correction improves a learning model's generalization in Artificial Neural Networks, in certain conditions.

9.2 Limitations

Empirically proving that a new loss function improves generalization is very challenging, as it is expected that it will not succeed in every problem set and/or learning

model. Although in the methodology we tried to cover a wide range of different problem sets and two learning models it is still insufficient to fully understand in which scope should we use our correction, so a broader study should be made.

9.3 Future Work

This thesis highlights many opportunities for future work, which we outline below.

- **Broader Scope** : In this experiment we only used two different numbers for the nearest neighbor parameter in the custom function, in the future a sensitivity study should be done to verify how they affect the training of the learning mode; In the particular case of the ANN model, try to understand if the use of different optimizers lead or not to similar results and naturally, using different hypothesis spaces or even alternative learning models.
- **Correct a different Loss**: We've chose to apply our correction loss function on the MAE loss but a study on the MSE or others would also be very pertinent.
- **Generalization Score** In this thesis we only enunciated and gave some formal background of the generalization score in chapter 4, nevertheless an experiment should be constructed in order to verify if the definition is empirically well founded.
- **P Score Early Stopping**: Also in chapter 4 we had some initial insights that the P Score alone could be used as an indicator for an early stopping method, it would be very interesting to conduct a study to verify if it the insights are valid valid.

Bibliography

[Moh+12] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN: 026201825X.

[Lan93] S. Lang. *Real and Functional Analysis*. Graduate texts in mathematics. Springer-Verlag, 1993. ISBN: 9783540940012. URL: https://books.google.pt/books?id=IVtHcAAACAAJ.

[Evg+00] T. Evgeniou, M. Pontil, and T. Poggio. "Statistical Learning Theory: A Primer." In: 2000. DOI: 10.1023/A:1008110632619.

[Gol10] S. A. Goldman. "Computational Learning Theory." In: *Algorithms and Theory of Computation Handbook: General Concepts and Techniques*. 2nd ed. Chapman Hall/CRC, 2010, p. 26. ISBN: 9781584888222.

[Tah16] H. Taherdoost. "Sampling Methods in Research Methodology; How to Choose a Sampling Technique for Research." In: *International Journal of Academic Research in Management* 5 (Jan. 2016), pp. 18–27. DOI: 10.2139/ssrn.3205035.

[KBH19] J. K. Blitzstein and J. Hwang. *Introduction to Probability Second Edition*. 2019.

[BS51] S. Bergman and M. Schiffer. "Kernel functions and conformal mapping." en. In: *Compositio Mathematica* 8 (1951), pp. 205–249. URL: http://www.numdam.org/item/CM_1951__8__205_0.

[CC12] Z. Cao and H. Cao. "On Fast Division Algorithm for Polynomials Using Newton Iteration." In: (2012). Ed. by B. Liu, M. Ma, and J. Chang, pp. 175–180.

[Sem65] Z. Semadeni. "Spaces of continuous functions on compact sets." In: *Advances in Mathematics* 1.3 (1965), pp. 319 –382. ISSN: 0001-8708. DOI: https://doi.org/10.1016/0001-8708(65)90041-1. URL: http://www.sciencedirect.com/science/article/pii/0001870865900411.

[Ghr14] R. Ghrist. *Elementary Applied Topology*. Createspace, 2014.

[Nel15] G. Nelson. *A User-Friendly Introduction to Lebesgue Measure and Integration*. Student Mathematical Library. American Mathematical Society, 2015. ISBN: 9781470421991. URL: https://books.google.pt/books?id=3fjkCgAAQBAJ.

[CS] F. Cucker and S. Smale. "On The Mathematical Foundations of Learning." In: Hong Kong: Departement of Mathematics City University of Hong Kong.

[Has+01] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[Nea19] B. Neal. "On the Bias-Variance Tradeoff: Textbooks Need an Update." In: *ArXiv* abs/1912.08286 (2019).

[Nea+18] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas. "A Modern Take on the Bias-Variance Tradeoff in Neural Networks." In: *CoRR* abs/1810.08591 (2018). arXiv: 1810.08591. URL: http://arxiv.org/abs/1810.08591.

[Nwa+18] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning." In: *CoRR* abs/1811.03378 (2018). arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378.

[Mac03] D. J. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[Zho+19] Y. Zhou, J. Yang, H. Zhang, Y. Liang, and V. Tarokh. "SGD Converges to Global Minimum in Deep Learning via Star-convex Path." In: *CoRR* abs/1901.00451 (2019). arXiv: 1901.00451. URL: http://arxiv.org/abs/1901.00451.

[Rud16] S. Ruder. "An overview of gradient descent optimization algorithms." In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[Dar59] C. Darwin. "On the Origin of Species by Means of Natural Selection." In: (1859).

[Wil+97] M. Willis, H. Hiden, P. Marenbach, B. McKay, and G. Montague. "Genetic programming: An introduction and survey of applications." In: Oct. 1997, pp. 314 –319. ISBN: 0-85296-693-8. DOI: 10.1049/cp:19971199.

[CM19] W. G. L. Cava and J. H. Moore. "Semantic variation operators for multidimensional genetic programming." In: *CoRR* abs/1904.08577 (2019). arXiv: 1904.08577. URL: http://arxiv.org/abs/1904.08577.

[LR15] T. P. L. Rosasco. *Machine Learning: a Regularization Approach*. MIT-9.520 Lectures Notes, Manuscript, 2015.

[NM18] M. A. Nabian and H. Meidani. "Physics-Informed Regularization of Deep Neural Networks." In: *CoRR* abs/1810.05547 (2018). arXiv: 1810.05547. URL: http://arxiv.org/abs/1810.05547.

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[Won+16] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell. "Understanding data augmentation for classification: when to warp?" In: *CoRR* (2016). arXiv: 1609.08764.

[Car+00] R. Caruana, S. Lawrence, and C. Giles. "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping." In: vol. 13. Jan. 2000, pp. 402–408.

[Rod+10] J. Rodríguez, A. Pérez, and J. Lozano. "Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation." In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 32 (Apr. 2010), pp. 569 –575.

[Chu+18] Y. Chung, P. J. Haas, E. Upfal, and T. Kraska. "Unknown Examples & Machine Learning Model Generalization." In: *CoRR* abs/1808.08294 (2018). arXiv: 1808.08294. URL: http://arxiv.org/abs/1808.08294.

[LZ14] A. Liu and B. Ziebart. "Robust classification under sample selection bias." In: *Advances in Neural Information Processing Systems* 1 (Jan. 2014), pp. 37–45.

[IS15] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *CoRR* (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[Ney+17] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. "Exploring Generalization in Deep Learning." In: *CoRR* abs/1706.08947 (2017). arXiv: 1706.08947. URL: http://arxiv.org/abs/1706.08947.

[Zha+18] H. Zhang, T. Weng, P. Chen, C. Hsieh, and L. Daniel. "Efficient Neural Network Robustness Certification with General Activation Functions." In: *CoRR* abs/1811.00866 (2018). arXiv: 1811.00866. URL: http://arxiv.org/abs/1811.00866.

[Set15] A. Seth. "Network Sensitivity Analysis: Invited Paper." In: *IETE Technical Review* 2 (June 2015), pp. 399–407. DOI: 10.1080/02564602.1985.11437861.

[Gó+14] I. Gómez, S. Cannas, O. Osenda, J. Jerez, and L. Franco. "The Generalization Complexity Measure for Continuous Input Data." In: *TheScientificWorldJournal* 2014 (Apr. 2014), p. 815156. DOI: 10.1155/2014/815156.

[Nov+18] R. Novak, Y. Bahri, D. Abolafia, J. Pennington, and J. Sohl-Dickstein. "Sensitivity and Generalization in Neural Networks: an Empirical Study." In: *ArXiv* abs/1802.08760 (2018).

[BG18] A. Brutzkus and A. Globerson. "Over-parameterization Improves Generalization in the XOR Detection Problem." In: *CoRR* abs/1810.03037 (2018). arXiv: 1810.03037. URL: http://arxiv.org/abs/1810.03037.

[Kim+19] B. Kim, H. Kim, K. Kim, S. Kim, and J. Kim. "Learning Not to Learn: Training Deep Neural Networks With Biased Data." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 9004–9012.

[Weg18] S. Weglarczyk. "Kernel density estimation and its application." In: *ITM Web of Conferences* 23 (Jan. 2018), p. 00037. DOI: 10.1051/itmconf/20182300037.

[AL16] O. Anava and K. Y. Levy. "k*-Nearest Neighbors: From Global to Local." In: *NIPS*. 2016.

[Xin+03] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng. "Distance Metric Learning with Application to Clustering with Side-Information." In: *Advances in Neural Information Processing Systems 15*. Ed. by S. Becker, S. Thrun, and K. Obermayer. MIT Press, 2003, pp. 521–528. URL: http://papers.nips.cc/paper/2164-distance-metric-learning-with-application-to-clustering-with-side-information.pdf.

[WM05] C. Willmott and K Matsuura. "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance." In: *Climate Research* 30 (2005), pp. 79–82. DOI: 10.3354/cr030079. URL: https://doi.org/10.3354%2Fcr030079.

[Arc+07] F. Archetti, S. Lanzeni, V. Messina, and L. Vanneschi. "Genetic programming for computational pharmacokinetics in drug discovery and development." In: *Genetic Programming and Evolvable Machines* 8 (Nov. 2007), pp. 413–432. DOI: 10.1007/s10710-007-9040-z.

[MCS13] L. V. M. Castelli and S. Silva. "Prediction of high performance concrete strength using Genetic Programming with geometric semantic genetic operators." In: *Expert Systems with Applications* (2013).

[MCP15] L. V. M. Castelli L. Trujillo and A. Popovic. "Prediction of energy performance of residential buildings: A genetic programming approach." In: (2015).

[OAB14] H. B. O. Akbilgic and M. E. Balaban. "A novel Hybrid RBF Neural Networks model as a forecaster." In: (2014).

[VRD09] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[Cho+15] F. Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[Aga18] A. F. Agarap. "Deep Learning using Rectified Linear Units (ReLU)." In: *CoRR* abs/1803.08375 (2018). arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375.

[GG18] E. Goceri and A. Gooya. "On The Importance of Batch Size for Deep Learning." In: July 2018.

[Wwwa] *GPLearn: A Genetic Programming library in Python*. https://gplearn.readthedocs.io/en/stable/. Accessed: 2020-08-14.

[Kol] "Kolmogorov–Smirnov Test." In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 283–287. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_214. URL: https://doi.org/10.1007/978-0-387-32833-1_214.

[RN11] D. Rey and M. Neuhäuser. "Wilcoxon-Signed-Rank Test." In: *International Encyclopedia of Statistical Science*. Ed. by M. Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1658–1659. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_616. URL: https://doi.org/10.1007/978-3-642-04898-2_616.

[Wwwb] *Scipy.stats: Wilcoxon's signed-rank test*. https://docs.scipy.org/doc/scipy//reference/generated/scipy.stats.wilcoxon.html/, note = Accessed: 2020-08-14.

Appendix A

Appendix

A.0.1 Bias Variance Tradeoff Part 1

Continuation of calculus from section (2.1.6) equation (2.12):

$$= \int_Y f(x)^2 - 2yf(x) + y^2 \, d\mu(y|x) \tag{A.1}$$

$$= \int_Y f(x)^2 \, d\mu(y|x) - 2\int_Y yf(x) \, d\mu(y|x) + \int_Y y^2 \, d\mu(y|x) \tag{A.2}$$

$$= \left(f(x) - \int_Y y \, d\mu(y|x)\right)^2 - \left(\int_Y y \, d\mu(y|x)\right)^2 + \int_Y y^2 \, d\mu(y|x) \tag{A.3}$$

$$= \left(f(x) - \int_Y y \, d\mu(y|x)\right)^2 + \int_Y y^2 \, d\mu(y|x) - \left(\int_Y y \, d\mu(y|x)\right)^2 \tag{A.4}$$

The second and third term by definition are the variance [citar] of Y denoted as σ_Y^2, which means:

$$\int_Y (f(x) - y)^2 \, d\mu(y|x) = \left(f(x) - \int_Y y \, d\mu(y|x)\right)^2 + \sigma_Y^2 \tag{A.5}$$

Now let us:

$$\int_X \int_Y (f(x) - y)^2 \, d\mu(y|x) d\mu_X = \int_X \left[\left(f(x) - \int_Y y \, d\mu(y|x)\right)^2 + \sigma_Y^2\right] d\mu_X \tag{A.6}$$

$$= \int_X \left[\left(f(x) - \int_Y y \, d\mu(y|x)\right)^2\right] d\mu_X + \sigma_Y^2 \tag{A.7}$$

$$= \int_X \left[f(x)^2 - 2f(x)\left(\int_Y y \, d\mu(y|x)\right) + \left(\int_Y y \, d\mu(y|x)\right)^2\right] d\mu_X + \sigma_Y^2 \tag{A.8}$$

$$= \int_X f(x)^2 d\mu_X - 2\int_X \left[f(x) \left(\int_Y y \; d\mu(y|x) \right) \right] d\mu_X + \int_X \left(\int_Y y \; d\mu(y|x) \right)^2 d\mu_X + \sigma_Y^2 \tag{A.9}$$

$$= \int_X f(x)^2 d\mu_X - 2\left(\int_Y y \; d\mu(y|x) \right) \int_X f(x) d\mu_X + \left(\int_Y y \; d\mu(y|x) \right)^2 + \sigma_Y^2 \tag{A.10}$$

$$= \int_X f(x)^2 d\mu_X + \left(\int_X f(x) d\mu_X - \left(\int_Y y \; d\mu(y|x) \right) \right)^2 - \left(\int_X f(x) d\mu_X \right)^2 + \sigma_Y^2 \tag{A.11}$$

$$= \int_X f(x)^2 d\mu_X - \left(\int_X f(x) d\mu_X \right)^2 + \left(\int_X f(x) d\mu_X - \left(\int_Y y \; d\mu(y|x) \right) \right)^2 + \sigma_Y^2 \tag{A.12}$$

The first and second term by definition are the variance of $f(x)$ denoted as σ_f^2, which means:

$$= \sigma_f^2 + \left(\int_X f(x) d\mu_X - \left(\int_Y y \; d\mu(y|x) \right) \right)^2 + \sigma_Y^2$$

A.0.2 Bias Variance Tradeoff Part 2

Continuation of calculus from section (2.1.6) equation (2.16):
Consider the equation (2.7):

$$\mathbb{E}_\mu[l(f(X), Y)] = \int_{X \times Y} l(f(x), y) d\mu = \int_X \left(\int_Y l(f(x), y) \; d\mu(y|x) \right) d\mu_X \tag{A.13}$$

Assume $l(f(x), y) = (f(x) - y)^2$ (MSE),

$$\int_Y l(f(x), y) \; d\mu(y|x) = \int_Y (y - f(x))^2 \; d\mu(y|x) \tag{A.14}$$

$$\sigma_Y^2 = \int_Y y^2 \; d\mu(y|x) - \left(\int_Y y \; d\mu(y|x) \right)^2 \tag{A.15}$$

$$= \int_Y (f_\mu(x) + e)^2 \; d\mu(y|x) - \left(\int_Y (f_\mu(x) + e) \; d\mu(y|x) \right)^2 \tag{A.16}$$

$$= \int_Y (f_\mu(x)^2 + 2e f_\mu(x) + e^2) \; d\mu(y|x) - \left(\int_Y f_\mu(x) \; d\mu(y|x) + \int_Y e \; d\mu(y|x) \right)^2 \tag{A.17}$$

$$= \int_Y f_\mu(x)^2 \; d\mu(y|x) + 2\int_Y e f_\mu(x) \; d\mu(y|x) + \int_Y e^2 \; d\mu(y|x) - \left(\int_Y f_\mu(x) \; d\mu(y|x) \right)^2 \tag{A.18}$$

$$= f_\mu(x)^2 + 2 f_\mu(x) \int_Y e \; d\mu(y|x) + \int_Y e^2 \; d\mu(y|x) - f_\mu(x)^2 \tag{A.19}$$

$$= \int_Y e^2 \; d\mu(y|x) = \sigma_e^2 \tag{A.20}$$

Also when dealing with the Bias:

$$\left(\underbrace{\int_X f(x) d\mu_X - \left(\int_Y y \; d\mu(y|x) \right)}_{\text{bias}} \right)^2 = \left(\int_X f(x) d\mu_X - \left(\int_Y (f_\mu(x) + e) \; d\mu(y|x) \right) \right)^2 \tag{A.21}$$

$$= \left(\int_X f(x) d\mu_X - \left(\int_Y (f_\mu(x)\, d\mu(y|x) + \int_Y e\, d\mu(y|x) \right) \right)^2 \tag{A.22}$$

$$= \left(\int_X f(x) d\mu_X - f_\mu(x) \right)^2 \tag{A.23}$$