

Quantifying Underspecification in Machine Learning with Explainable AI

James Hinns

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Master of Science By Research



Swansea University
Prifysgol Abertawe

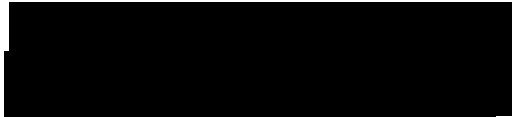
Department of Computer Science
Swansea University

June 2022

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

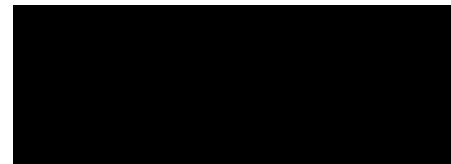
Signed



Date 08/06/22

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

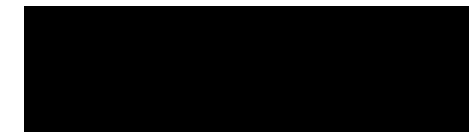
Signed



Date 08/06/22

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

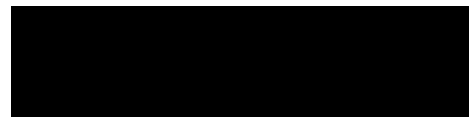
Signed



Date 08/06/22

The University's ethical procedures have been followed and, where appropriate, that ethical approval has been granted.

Signed



Date 08/06/22

Abstract

To evaluate a trained machine learning (ML) model’s performance, it is general practice to test its performance by predicting targets from a held-out testing set. For such a dataset, various models can be constructed with different reasoning that produce near-optimal test performance. However, due to this variance in reasoning some models can generalise, whilst some perform unexpectedly on further unseen data. The existence of multiple equally performing models exhibits underspecification of the ML pipeline used for producing such models. Underspecification poses challenges towards the credibility of such test performance evaluations and has been identified as a key reason why many models that perform well in testing, exhibit poor performance in deployment.

In this work, we propose identifying underspecification by estimating the variance of reasoning within a set of near-optimal models produced by a pipeline, also called a Rashomon set. We iteratively train models using the same pipeline to produce an empirical Rashomon set of a fixed size. In order to quantify the variation of models within this Rashomon set, we measure the variation of SHapley Additive exPlanations that the models produce using a variety of metrics. This provides us with an index representing the variation of reasoning within this Rashomon set, and thus the pipeline. This index therefore represents the extent of underspecification the pipeline exhibits.

We provide an implementation for this approach, and make it publicly available on github. We validate that this implementation shows the trends we expect using evaluation techniques previously used to prove the existence of underspecification. Furthermore, we demonstrate our approach on multiple datasets drawn from the literature, and in a COVID-19 virus transmission case study.

Acknowledgements

I would first like to thank both of my supervisors for their continued guidance and patience throughout this project.

Without Dr. Fan Xiuyi I would not have undertaken a masters of this kind, and possibly not at all. He has not only guided me through the technical aspects of my first project in XAI, first publication, and first research role, but also provided a calm stabilising figure throughout.

Additionally, I would like to thank my other supervisor, Prof. Markus Roggenbach, who generously supported me throughout and after Fans move. His experience, patience and understanding helped me with the final push I needed to finish this thesis.

I have learnt a great deal from both of them, and I am incredibly thankful, to once again, have such excellent guidance.

My thanks also extend to Orçun Yalçın and Raghav Kovvuri who helped me in my introduction to XAI, and particularly helped me get up to speed whilst I worked alongside them as a research assistant.

Finally, I would like to thank my mum. She helped to proofread and remove the continual random commas I add to most sentences, and did everything she was able to, so I could focus on getting this project finished.

Table of Contents

1	Introduction	1
1.1	Underspecification	2
1.2	Quantifying Underspecification with XAI	4
1.3	Aims and Contributions	5
1.4	Presented Material	6
1.5	Chapter Overview	6
I	Background Material	9
2	Supervised Machine Learning	11
2.1	Components of a Supervised Machine Learning System	11
2.2	Evaluation	12
2.3	Optimisation	18
3	Explainable Machine Learning	23
3.1	Introduction and Motivation	23
3.2	Post-hoc Interpretability	26
3.3	SHAP (SHapley Additive exPlanations)	28
4	Metrics	31
4.1	Distance, Similarity and Correlation Metrics	31
4.2	Euclidean Distance	32
4.3	Pearson Correlation Coefficient	34
4.4	Cosine Similarity	35
4.5	Kendall Rank Correlation Coefficient	37
5	Related Work	41
5.1	Stress Tests	41
5.2	Explanation Quantification	43
5.3	Explanation Influenced Optimisation	44
5.4	The Rashomon Effect	46
5.5	Prediction Variance	47

II Contributions	51
6 Measuring Underspecification with Underspecification Index	53
6.1 Quantifying Underspecification	53
6.2 Method Definition	57
7 Implementing Underspecification Index for Classification	61
7.1 Language and Library Considerations	61
7.2 Software Realisation	63
7.3 Experimental Setup	66
7.4 Experiment Results	69
8 Implementing Underspecification Index for Regression	77
8.1 Software Realization	77
8.2 Experimental Setup	78
8.3 Experiment Results	80
9 COVID-19 Case Study	85
9.1 Dataset Creation	85
9.2 Experimentation	88
III Conclusions	93
10 Conclusions and Future Work	95
10.1 Quantifying Underspecification with XAI	95
10.2 Future Work	98
Bibliography	101
A Classification Results	111
B Regression Results	119
B.1 Cosine Index Against Variance	120
B.2 Underspecification Index Variations against Accuracy	122

Chapter 1

Introduction

Contents

1.1	Underspecification	2
1.2	Quantifying Underspecification with XAI	4
1.3	Aims and Contributions	5
1.4	Presented Material	6
1.5	Chapter Overview	6

Often machine learning practitioners notice systems that performed well during testing, perform poorly when deployed in real world applications. There can be many reasons why these performance differences exist [QCSSL09, GJM⁺20, DYT⁺20], however it is generally suggested that having a sufficiently large training and testing set will mitigate this [DYT⁺20]. Unfortunately this is not the case in many situations [Hea20].

Underspecification is a key reason why these performance differences exist [DHM⁺20].

Definition 1.0.1 An underspecified machine learning problem is where there can exist multiple possible models that perform similarly well on the testing set.

Explanations can answer a why question about a single prediction given by a machine learning model [Mil19]. By generating an explanation about a prediction one can evaluate whether its decision process was reasonable and thus whether the prediction can be trusted [DVK17, Mol19]. An explanation with a reasonable decision process that performs well on the testing set (presuming the testing set effectively models the distribution of the real world population) should provide a model which performs well in the real world.

This thesis aims to establish areas in which underspecification exists, guided by and building upon, examples from the underspecification paper [DHM⁺20]. We do

this by computing the agreement of different possible explanations for a given machine learning problem. By quantifying the agreement between these explanations, we are able to quantitatively evaluate how credible test performance evaluations are.

1.1 Underspecification

A supervised learning problem aims to produce a predictor $f : X \rightarrow Y$ that maps input X to output Y . A model is specified by a model class \mathcal{F} from which a predictor $f(x)$ is chosen [DHM⁺20]. A machine learning pipeline involves everything to create a predictor, taking in data D from training distribution P and producing a predictor $f(x) \in \mathcal{F}$. The pipeline (generally) selects $f(x)$ from \mathcal{F} by maximising performance of some metric on a test set. However, there can exist many different predictors $f(x) \in \mathcal{F}$ that perform equally well on independent and identically distributed (i.i.d) evaluation, but perform differently on data further to that used in training.

Example 1.1.1 Consider the binary classification of five-bit binary strings into two classes; positive POS and negative NEG.

Consider the testing set D_{tst} such that each class has four examples:

$$\text{POS} : \{01101, 11101, 11111, 01111\}$$

$$\text{NEG} : \{00000, 00010, 10010, 10000\}$$

Consider three predictors $Predictor_1$, $Predictor_2$ and $Predictor_3$ that classify each example by examining only one-bit such that:

$$Predictor_1(x) = \begin{cases} \text{POS}, & \text{if } x_2 = 1 \\ \text{NEG}, & \text{otherwise} \end{cases}$$

$$Predictor_2(x) = \begin{cases} \text{POS}, & \text{if } x_3 = 1 \\ \text{NEG}, & \text{otherwise} \end{cases}$$

$$Predictor_3(x) = \begin{cases} \text{POS}, & \text{if } x_5 = 1 \\ \text{NEG}, & \text{otherwise} \end{cases}$$

All three predictors perfectly classify the ten examples in D_{tst} .

As shown in example 1.1.1 pipelines can produce multiple equally evaluated predictors with different internal methods. Pipelines such as these are called underspecified. More specifically, “An ML pipeline is underspecified when it can return many predictors with equivalently strong held-out performance in the training domain” [DHM⁺20]. Choosing between equivalently evaluated predictors of an underspecified pipeline such

as $Predictor_1$ and $Predictor_2$ are based on arbitrary factors. However, this same performance on the test set doesn't guarantee the same performance on further datasets.

As we will discuss in 2.3, the mapping learned by an ideal predictor should generalise to all datasets from the same true distribution. That is, an ideal predictor trained to classify cats and dogs should work on any picture of cats and dogs, not just those included in the training dataset. An underspecified pipeline is problematic in that equivalently evaluated predictors may perform differently on data further to the training and testing set.

Example 1.1.2 Consider again the pipeline that produces $Predictor_1$, $Predictor_2$ and $Predictor_3$ from example 1.1.1.

Consider the testing set \hat{D}_{tst} which is drawn from the same distribution as D_{tst} . \hat{D}_{tst} includes all examples from the previous testing set D_{tst} , and an additional instance for each class, such that:

$$\begin{aligned} \text{POS} &: \{01101, 11101, 11111, 01111, 01001\} \\ \text{NEG} &: \{00000, 00010, 10010, 10000, 00001\} \end{aligned}$$

$Predictor_1$ maintains its perfect performance on this new testing set \hat{D}_{tst} .

However $Predictor_2$ and $Predictor_3$ misclassify the newly added instances.

Of similarly evaluated predictors, some may generalise well, whilst others may not. This variation of supposedly well-behaving predictors presents challenges for credibility of test performance evaluations.

Although underspecification is not directly looking at state-space, these equivalently performing predictors can be seen as a state-space with many similarly valued minima.

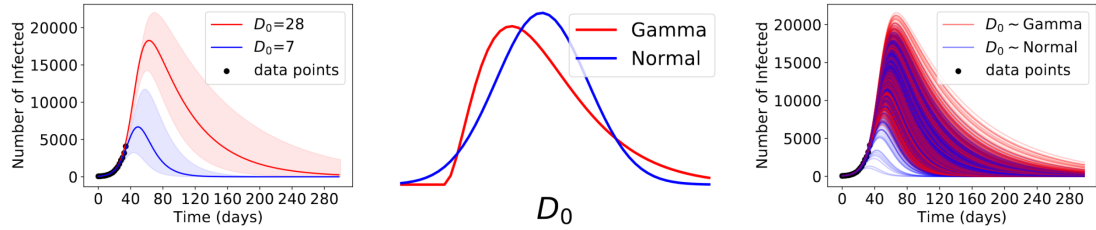


Figure 1.1.1: The different trajectories produced by different equivalently evaluated parameter sets using the SIR model as described in example 1.1.3 [DHM⁺20].

Example 1.1.3 The following example is summarised from [DHM⁺20]. Consider the Susceptible-Infected-Recovered (SIR) model used for epidemiological forecasting. The model enables the calculation of rates of susceptible (S), infected (I), and recovered (R) people.

$$\frac{dS}{dt} = -\beta \left(\frac{1}{N} \right) S, \quad \frac{dI}{dt} = -\frac{I}{D} + \beta \left(\frac{1}{N} \right) S, \quad \frac{dR}{dt} = \frac{I}{D},$$

where N is the population size, β the transmission rate and D the duration that someone remains infectious.

The specific machine learning problem is to learn the parameters β and D up to some time T_{obs} using gradient descent (a similar local search technique to hill climbing, discussed in chapter 2). With these learned parameters the full epidemiological trajectory up to time T is predicted, such that $T_{obs} < T$. When T_{obs} is small, that is the training data is small, the model is underspecified, as the number of susceptible is around constant to the population size (N), and infections grow approximately exponentially at the rate $\beta - 1/D$. There are many different parameter permutations of β and D that model this exponential growth $\beta - 1/D$. Because each of these parameter sets cannot be distinguished in this training setting, arbitrary choices determine which of these equivalently evaluated parameters are returned. The first image in figure 1.1.1 shows two different parameter sets that equivalently fit the given data (and are therefore equivalently evaluated) with orders of magnitude apart. The second image in figure 1.1.1 shows how the distribution used to draw D_0 (the point at which D is initialised) influences the predicted trajectories. Finally, the third image in figure 1.1.1 shows how changing D_0 produces a wide range of trajectories.

1.2 Quantifying Underspecification with XAI

As discussed above, an ML pipeline is underspecified when it is able to produce many different predictors with similar performance on the provided testing set. However, in the paper presenting underspecification as an issue in machine learning, [DHM⁺20], no formal definition is provided, and no way to quantify underspecification is presented. We therefore extend this definition to say pipelines are underspecified when we see variation between well performing predictors. Furthermore, we say pipelines with greater variation are more underspecified than those with less variation. Thus, we quantify underspecification as the variation between well performing predictors.

This definition allows us to quantify underspecification by examining the variation between well performing predictors produced by a pipeline. To do this, we use the pipeline to iteratively train predictors adding them to our ensemble only if they exceed a chosen performance threshold. If this value is chosen to be close to the optimal achievable performance of predictors for this pipeline, it ensures all predictors in the ensemble perform similarly well on available data. Comparing the variation of predictors within this ensemble should quantify the extent of underspecification for the given pipeline.

However, directly comparing predictors is not straightforward as they can contain a vast amount of parameters and different architectures will have different internal structures. In order to simplify this comparison, and to provide a model-agnostic approach for quantification, we compare explanations instead of predictors. As explanations model predictors for a given prediction (we discuss this in more detail in section 3.1), we can use a number of their produced explanations as a representation of the model itself. Our major assumption here is that the methods we use to generate such expla-

nations are consistent, that is, similar predictors should produce similar explanations.

Therefore, for each predictor in our similarly performing ensemble, we compute their explanations for a number of data instances, and compare the agreement of these explanations across predictors. The greater the agreement of these explanations, the more similar the predictors and therefore the less underspecified the pipeline is. Additionally, we say pipelines with little variation between explanations have more credibility in their test performance evaluations. We call the value representing agreement between explanations for a specific data instance a local underspecification index. We call the average of all local indices for the provided test set the set underspecification index. This set underspecification index is representative of the agreement of all similarly well performing predictors produced by our pipeline, and thus the extent of underspecification of the given pipeline.

1.3 Aims and Contributions

The major aim of this thesis is to investigate the possibility of underspecification quantification by quantifying explanation agreement. In order to investigate this, we aim to provide a basic framework which given a dataset constructs a Rashomon set for it and quantifies the agreement of explanations between each predictor in this Rashomon set. Using this framework we aim to produce repeatable results to enable researchers to continue our work, or utilise such evaluations to quantify if the quality of their dataset is sufficient for providing credible test performance evaluations.

The main contributions of this thesis are as follows.

Underspecification Indexes as a Measure of Underspecification: We formulate underspecification quantification as a problem of measuring similarity between explanations produced by predictors in a Rashomon set. We represent this similarity using our proposed underspecification indices, which can be used for both local and set-level evaluations.

Effect of Metric Choice on Underspecification Indexes: We quantify the similarity of explanations using four well studied metrics, Euclidean distance, Pearson correlation, Cosine similarity, and Kendall rank correlation. We discuss the implications of using each of these metrics to produce underspecification indices.

Framework for Computing Underspecification Indexes: We provide a Numba-optimised Python implementation of our proposed approach for both classification and regression problems. This implementation is publicly available in our repository.

<https://github.com/JamesHinns/Underspecification-Index>.

Relationship Between Underspecification Indexes and Evaluation Metrics: We demonstrate our approach for classification and regression on both existing datasets in the literature and a real-world COVID-19 dataset. We compare our produced underspecification indices to established performance and variation metrics for pipelines.

1.4 Presented Material

The work within this thesis has led to the following publication:

An Initial Study of Machine Learning Underspecification using Feature Attribution Explainable AI Algorithms: a COVID-19 Virus Transmission Case Study [HFL⁺21] (James Hinns, Xiuyi Fan, Siyuan Liu, Veera Raghava Reddy Kovvuri, Mehmet Orcun Yalcin and Markus Roggenbach PRICAI 2021)

This paper (and corresponding poster) presents an initial study of investigating explanation agreement as a form of underspecification quantification. Furthermore, it outlines the use of Kendall rank correlation coefficient as a metric for predictor variance across an ensemble. This paper only considers classification problems.

Additionally, the work within this thesis has been presented:

Quantifying Underspecification in Machine Learning using Explainable AI (James Hinns BCTCS 2022)

This presentation provided an overview of the work conducted in this thesis. We outlined our method of quantifying underspecification by explanation agreement. We briefly discussed the trends we find between our underspecification indices, prediction variance and prediction performance.

1.5 Chapter Overview

This dissertation is organised as follows:

We first introduce the background materials for this work:

- Chapter 2 introduces supervised learning problems. We give an overview of the basic components of a supervised machine learning system and the interactions between them. Particularly of focus in this chapter are the processes of evaluation and optimisation, in order to build upon in further chapters. In doing so we discuss a selection of convergence problems that may occur during optimisation, and how such problems may affect produced predictors.
- Chapter 3 presents an overview of interpretability in machine learning. We focus primarily on model-agnostic post-hoc interpretability methods as these are key to our method for interpreting underspecification in a machine learning pipeline. We discuss SHAP, the specific explanation method we use to model predictors produced by pipelines.
- Chapter 4 details metrics of distance, similarity and correlation. Specifically, we cover four metrics, Euclidean distance, Pearson correlation coefficient, Cosine similarity, Kendall rank correlation coefficient. For each of these metrics, we review definitions and provide examples calculating each.

- In chapter 5 we discuss some of the related work to our work. We examine the small amount of work that has been proposed for identifying underspecification directly. Additionally, we discuss methods that whilst not aimed at underspecification, have a potential for identification, via predictor variation quantification.

We then present our contributions for this work:

- Chapter 6 provides an overview to our proposed method for quantifying underspecification. We first review the definition of underspecification from this chapter, and follow how we can quantify its extent by examining variation between predictors. We then provide our approach for such quantification, giving pseudocode and component diagrams.
- Chapter 7 covers our implementation of the method proposed in chapter 6 for classification problems. We provide a naive implementation, followed by our subsequent optimisation of this implementation. We provide justifications for the technology choices we use to realise this implementation. Using this implementation, we experiment on a number of datasets from the literature in order to validate results shown the expected trends.
- Chapter 8 adapts the implementation provided in chapter 7 to work in both a regression and classification setting. Once again we experiment on a number of datasets from the literature to validate our approach.
- Chapter 9 presents a COVID-19 case study in both regression and classification for quantifying underspecification. Presenting a dataset used for our publication [HFL⁺21], we present a realistic version of the epidemiological example we show in example 1.1.3.

Finally, we conclude our work:

- Chapter 10 concludes this thesis by summarising our proposed method for quantification. Additionally, we present a selection of further research directions that could build upon the work presented in this thesis.

Part I

Background Material

Chapter 2

Supervised Machine Learning

Contents

2.1	Components of a Supervised Machine Learning System	11
2.2	Evaluation	12
2.3	Optimisation	18

In this chapter we cover the basic components of a supervised machine learning system, and the interactions between them. We focus on the processes of evaluation and optimisation as these are most relevant to our work. We consider the differences between evaluating regression and classification problems together with some of the nuances with such evaluations. Whilst discussing optimisation we briefly cover the basics of local search algorithms and a selection of the convergence problems these can face. We finally discuss the ramifications of these nuances and how selecting parameters based on such evaluations can lead to some of the issues seen in underspecified pipelines.

2.1 Components of a Supervised Machine Learning System

Traditionally, computers must receive explicit instructions in the form of an algorithm to achieve some goal. An algorithm is “a sequence of computational steps that transform a given input into the specified output” [CLRS09]. Machine learning differs from this approach in that provided data, machine learning methods decide the steps to the specified goal [Sam59].

Machine learning problems can be broadly split into four categories; supervised, unsupervised, semi-supervised, reinforcement learning [Sal18]. As our method is only aimed towards supervised learning problems we do not cover any other problem types. Supervised learning “observes some example input–output pairs and learns a function that maps from input to output” [RN09].

Following [RN09] recall that supervised learning problems can be generalised as:

Definition 2.1.1

Using an input training dataset \mathcal{D} of length n ; $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$

Predict output of prediction targets $Y = \{y_1, \dots, y_n\}$

by estimating a function f such that $f(x_i) \rightarrow y_i, i \leq n$

Each pair (x_i, y_i) in the training set $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ is called a training example. Each training example contains a collection of *features* x and a *prediction target* (label) y . *Features* are a measurable property of some phenomenon [Bis06], such as in the prediction of house prices, the number of rooms or interior volume. A *prediction target* is the feature that must be predicted by a given model.

Supervised learning algorithms can loosely be split into three sections[Tam21]:

Decision process: An algorithm that takes in training features x and returns predictions y by forming a model $f(x) \rightarrow y$ of the data it is given.

Model evaluation: A process measuring how effective the model produced by the decision process is by testing it on examples for which we possess the true outcome.

Model optimisation: Updating the decision process in such a way as to improve the prediction accuracy found in evaluation.

Model evaluation and optimisation are discussed in further detail in Section 2.2 and 2.3 respectively. These three sections (decision process, evaluation, optimisation) are repeated until the model reaches some evaluation optimum, at which point the training is complete. The model produced at the end of training, that is, the model with the final permutation of parameters, is called a predictor. This predictor, f , is chosen from all possible permutations of parameters (the model class \mathcal{F}) via training. The exact way the decision process works varies dependent on the type of model architecture that is used. Our method is model-agnostic (discussed in Section 3.2), treating all models as black boxes, we do not discuss any details of the decision process, only its interactions in the formation of the predictor.

2.2 Evaluation

Model evaluation should happen with respect to two independent and identically distributed (i.i.d) datasets; training and testing. During training models are evaluated using the training set that they learn from, whereas in testing they are evaluated against an unseen i.i.d test set. To accomplish this, datasets \mathcal{D} are often split into a training \mathcal{D}_{trn} and testing \mathcal{D}_{tst} set as shown in Figure 2.1.1. This i.i.d evaluation acts as validation or a ‘contract’ [JMMG20] that the predictor will behave the same on further data from the same distribution as the given dataset. It also helps to diagnose issues such as overfitting as we go on to discuss in 2.3.

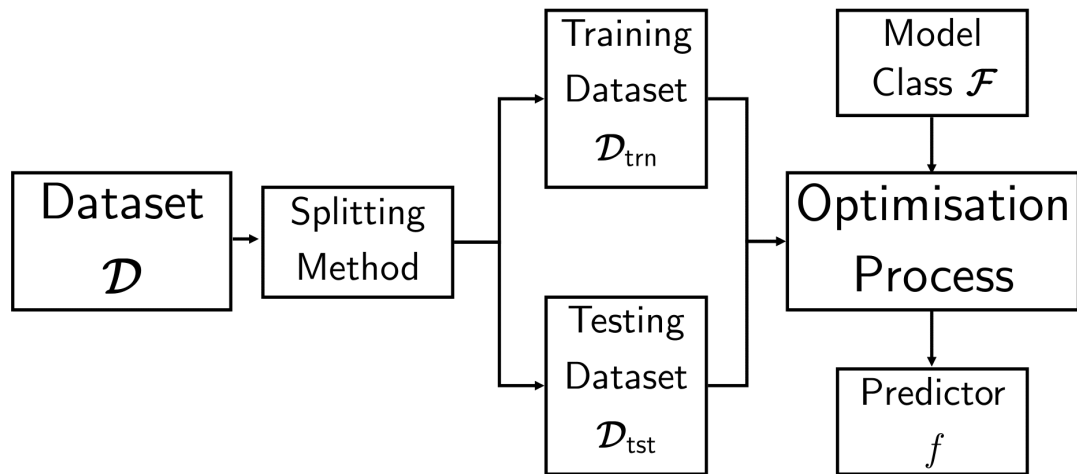


Figure 2.1.1: Example process of dataset to predictor

Supervised learning problems can be split into two main categories; classification and regression. The difference between classification and regression is the format of the prediction target. If the prediction target is a continuous value it is a regression problem. Whereas if the prediction target is discrete the problem is a classification [RN09]. Both regression and classification models are evaluated slightly differently as we go on to discuss.

2.2.1 Regression

Definition 2.2.1 *Regression* problems are supervised learning problems where the prediction target is a continuous value [AR18]. More specifically, *regression* problems aim to choose some mapping $f : X \rightarrow Y$ from model class \mathcal{F} that predicts the continuous output prediction target $Y \in \mathbb{R}$ from input features X .

An example of a regression problem could be to predict what the temperature would be on a given day. To calculate how effective a predictor is at a regression problem we can use a number of different error metrics. Error metrics aim to give an indication of how far away the predictions of a predictor are from ground truth results.

Example 2.2.1 One such error metric for regression is Mean Squared Error (MSE). MSE is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \quad (2.1)$$

where n is the number of values predicted, Y_i the true value, and \hat{Y}_i the predicted value.

Let f be a predictor that estimates stock prices in £(GBP), which has its performance evaluated using MSE.

Consider two sets such that

$v_t = \{1.2, 1.75, 2.05\}$ representing the price of three stocks at some time.

$v_p = \{1.2, 1.8, 1.95\}$ representing f 's predicted price of the same three stocks.

The MSE of these predictions is calculated as:

$$MSE = \frac{1}{3}[(1.2 - 1.2)^2 + (1.75 - 1.8)^2 + (2.05 - 1.95)^2] \approx 0.0042$$

2.2.2 Classification

Definition 2.2.2 *Classification* problems are supervised learning problems where the prediction target is a discrete value [AR18]. More specifically, *classification* problems aim to choose some mapping $f : X \rightarrow Y$ from model class \mathcal{F} that predicts the discrete prediction target $Y \in \{c_1, \dots, c_n\}, n \in \mathbb{N}$ from input features X .

The discrete outputs are called classes, where each possible unique discrete output $c_i, \forall i \in [1, n]$ is a class. If only two classes exist $n = 2$, that is, the prediction target is a Boolean, the problem is called a binary classification. An example of a binary classification problem could be to predict whether the temperature on a given day would be above 0°C or not.

Evaluating classification problems is different to regression, as the definition of error for regression problems is not directly translatable to classification. In regression problems, error is the true value minus the predicted value, which is not possible in classification if there is no order to classes. If there is order between classes, the distance between the classes may be uneven and make such an error unreliable. Classification evaluation often utilises a confusion matrix as shown in table 2.2.1. This confusion matrix shows evaluation results for a binary classification problem between two classes; negative and positive. True values are those that are predicted correctly, whilst false are those predicted incorrectly.

Table 2.2.1: A Binary Confusion Matrix, where TN is the number of True Negatives, FN, the number of False Negatives, FP, the number of False Positives, and TP is the number of True Positives.

		Predicted Class	
		Negative	Positive
Actual Class	Negative	TN	FN
	Positive	FP	TP

Definition 2.2.3 For binary classification, recall the definitions:

$$\textit{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\textit{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\textit{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total Predictions}}$$

Using such a confusion matrix enables the calculation of further metrics such as precision, recall, and accuracy [Faw06], which are defined in Definition 2.2.3. Each of these metrics paints a slightly different picture and as such all have slightly different use cases. Accuracy is simply the ratio of correct predictions to total predictions. It gives an overall view to the performance of the model, but can be misleading. Continuing with our temperature example, if 9 of 10 examples are positive, that is above 0° , even a predictor that predicts positive regardless of input would have 90% accuracy.

Precision is the ratio of correct positive predictions to total positive predictions. This can be a more insightful evaluation metric than accuracy if the avoiding false positives is important.

Example 2.2.2 An example of this could be email spam detection, if a legitimate email (negative) is predicted as spam (false positive) the user may lose the email, so a precise spam detector is important. In this domain, consider two predictors A and B , they each predict if an email is spam (positive) or is legitimate (negative). They use a dataset of 100 emails, 40 of which are spam and 60 of which are legitimate.

Predictor A correctly detects 30 emails as spam (True Positive) and lets 10 spam emails through (False Negative), whilst correctly predicting all 60 legitimate emails (True negative). This gives an accuracy of 0.9 (or 90%) and a precision of 1.0.

Predictor B correctly detects all 40 spam emails (True Positive), and correctly predicts 55 of the legitimate emails (True Negative), misclassifying 5 as spam (False Negative). This gives an accuracy of 0.95 and a precision of 0.8.

Although B has a higher accuracy, in this domain A may be the better choice. As B incorrectly classifies 10 emails as spam, the information on them may be lost. This is likely of more importance than the 10 emails incorrectly shown to the user as legitimate by predictor B , as the user can dismiss these emails as irrelevant.

Recall is the ratio of correct positive prediction to total positive examples. Recall is effective when avoiding false negatives is important.

Example 2.2.3 An example where avoiding false negatives is important, could be infectious disease detection, where if a patient is falsely cleared they could face severe consequences. Consider two predictors A and B that perform binary classification on a dataset of 100 examples, 30 of which are infected (positive) and 70 of which are not (negative).

Predictor A correctly predicts 25 of the 30 ill patients (True Positive), with 5 incorrectly classified as not infected (False Negative). A correctly predicts all 70 non-infected patients. This gives an accuracy of 0.95 (95%) and a recall of 0.83.

Predictor B correctly predicts all 30 infected patients (True Positive), whilst incorrectly predicting 10 non-infected as infected (False Positive). The other 60 non-infected patients are all correct classified (True Negative). Predictor B therefore has an accuracy of 0.9 (90%) and a recall of 1.0.

In this setting, B is likely the better choice. Regardless of the better accuracy A exhibits, it incorrectly classifies 5 infected patients as non-infected. These 5 patients will likely go on to spread the disease to more people and intensify the problem spreading the disease to more people. Minimising this risk is likely worth the trade-off of incorrectly classifying 10 healthy patients as infected, as they can then be manually cleared by medical professionals.

Example 2.2.4 Consider the binary classification problem of predicting whether a stocks price will rise (positive) or not (negative).

$v_t = [\text{POS}, \text{POS}, \text{POS}, \text{NEG}]$ the true classes, where POS = rise, and NEG = fall

$v_p = [\text{POS}, \text{POS}, \text{NEG}, \text{NEG}]$ the predicted classes

This results in the confusion matrix:

		Actual	
		NEG	POS
Predicted	NEG	TN:1	FN:1
	POS	FP:0	TP:2

Thus, accuracy = $\frac{3}{4} = 0.75$, precision = $\frac{2}{2} = 1$, recall = $\frac{2}{3}$

Example 2.2.5 Consider the multi-class classification of images of fruits. The dataset contains images of apples (a), oranges (o), pears (p), and bananas (b).

$v_t = [a, o, b, b, p, a]$ representing the true classes of examples.

$v_p = [a, a, b, o, p, a]$ representing the predicted classes of examples.

Giving the multi-class confusion matrix:

		Actual			
		a	o	p	b
Predicted	a	2	1	0	0
	o	0	0	0	1
	p	0	0	1	0
	b	0	0	0	1

In multi-class classification recall and precision are calculated per class.

Considering only those instances predicted as apples a , we can calculate the recall and precision using the confusion matrix.

From the confusion matrix we can see both apple instances are correctly classified as apples, and that one orange is falsely classified as an apple.

This means, for the apple class $TP = 2, FP = 1, TN = 2, FN = 0$.

Which in turn gives a precision of $\frac{2}{2+1} = \frac{2}{3}$ and recall of $\frac{2}{2+0} = 1$.

Continuing this for all classes results in:

Class Name	Precision	Recall
apples (a)	$\frac{2}{3}$	1
oranges (o)	0	0
pears (p)	1	1
bananas (b)	1	$\frac{1}{2}$

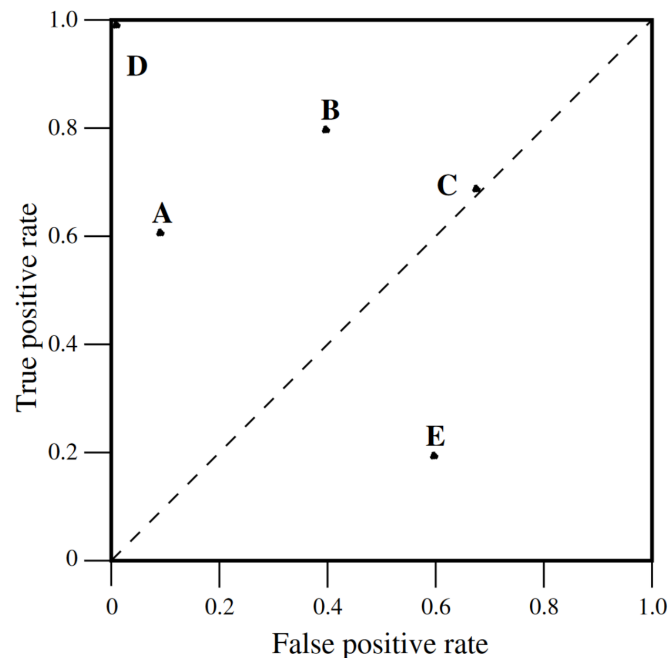


Figure 2.2.1: An example ROC graph with 5 predictors [Faw06].

Receiver operating characteristics (ROC) graphs display similar information to that

of a confusion matrix, with the distinct advantage of easily comparing many predictors [Faw06]. ROC graphs plot the true positive rate (precision) against false positive rate, where the false positive rate is the ratio of false positives to total negative class instances. Figure 2.2.1 shows such a ROC graph with 5 predictors, which could be produced by the same pipeline, simply changing parameters, or 5 completely different model architectures. The dotted line shows where the true positive rate and false positive rate are equal, and as such have a 50% accuracy. A predictor that randomly classified between positive and negative would place on this line (if using a balanced dataset), thus predictors such as E that fall below this line are worse than random. Generally ROC graphs can be interpreted as the closer a point to the upper left point (D), the better. Using this information, one can decide which models performance most fits the task dependent on how many false positives are acceptable. Similar to ROC graphs, Precision-Recall (PR) graphs replace the false positive rate with precision. Although ROC and PR graphs use a different metric for the x axis, they show the same patterns as each other, where a "curve only dominates in ROC space if and only if it dominates in PR space" [DG06].

2.3 Optimisation

As mentioned in Section 2.1, a key part of the supervised learning process is optimisation. Optimisation is the process by which model parameters are changed in such a way as to improve some objective function. An objective function is used to evaluate the performance of a system towards its intended task [RN09]. In supervised learning these objective functions most commonly come in the form of error metrics, such as those previously discussed. Objective functions are commonly referred to as loss functions, where a lower result (or loss) is preferred, or a reward function, where a higher result (or reward) is the objective.

Figure 2.3.1 shows an example of the state-space landscape. Such a landscape has a 'location' defined by the state (current choice of parameters), and an 'elevation' defined by the objective function. If using a reward function the aim of a supervised learning problem is to maximise this, or in other words look for the global maximum of the state-space. If using a loss function, the aim is to find the global minimum. To simplify this terminology we use the term optimum in this work to refer to the relevant maximum or minimum.

A simple example of a search algorithm used for such optimisation problems is hill-climbing search. Hill climbing (also sometimes called greedy search) aims to find an optimum by increasing its elevation. It simply loops until the next value is not higher than previous, so hits a peak. "This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia" [RN09]. A downside to basic hill climbing is that it will return the first optimum it hits, not necessarily the global optimum. If we consider the state-space shown in figure 2.3.1, starting at the marked current stated and travelling upwards in the direction of the arrow, we would reach a local maximum, not the global maximum. Reaching a local optimum rather than the global optimum

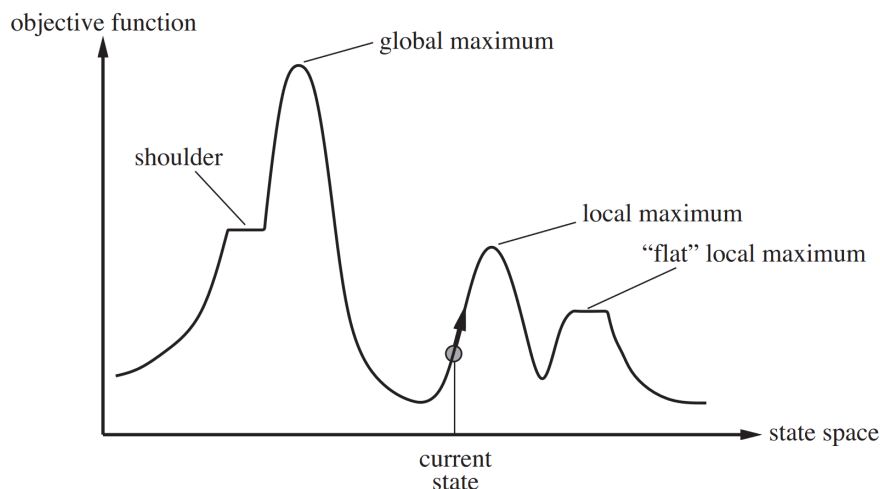


Figure 2.3.1: Simple state-space landscape, x axis represents parameter change, and y axis the result of a reward function [RN09].

means the optimal solution has not been found [GP71]. As the starting position of hill climbing is random, the algorithm can be ran multiple times, returning different optimum each time.

Many adaption to basic greedy search exist that are more resistant to such problems of returning a local optimum. The problem still remains, that there is no guarantee of finding a global optimum without searching the entire state-space [RN09]. This is why these search techniques are heuristics, the problem of finding a global optimum in a large high-dimensional state space is intractable, as the only way to do this is to evaluate every possible permutation of parameters. It should be noted that in a high dimensional space, problems of running into local optimum are less common. Instead, convergence issues such as saddle points are more common [DPG⁺14]. Saddle points seem similar to local optimum, in that the gradient in all directions is close to 0 (stationary point), but also they are a point of inflection. For this work, the effect these have can be seen as the same as local optimum, in that they return a result other than the global optimum, and are therefore sub-optimal.

Such search techniques are aiming for the global optimum as defined by the state-space and loss function, neither of which necessarily truthfully represent the real function. This means a model can perfectly fit the function of the training data, but perform poorly on test data, such a model is called overfitted. Figure 2.3.2c a simple example of overfitting a model to a binary classification. Overfitting is more likely in complex settings with more features, and is less likely with greater training instances [RN09]. Figure 2.3.2a shows underfitting, which is where the model fails to capture the relationship of the training data. This means, to find the optimal predictor balance must be found between underfitting and overfitting. This general trade-off between bias and variance can be summarised as: “The price to pay for achieving low bias is high variance” [GBD92]. As graphically represented in Figure 2.3.3, variance describes the spread of

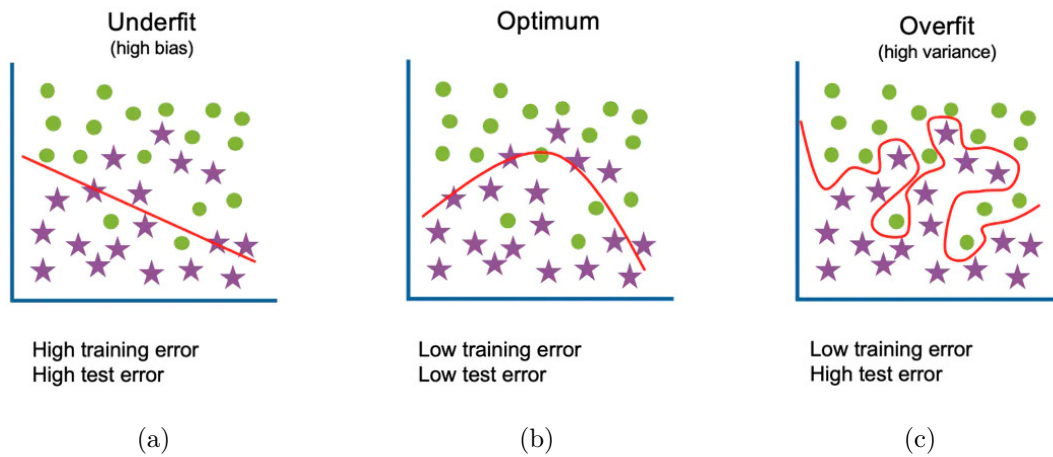


Figure 2.3.2: A simplified example of underfitting and overfitting in binary classification [Edu21]. One class is shown with purple stars and another shown by green dots, the red line shows the decision boundary.

predictions, whereas bias describes inherent error in predictions.

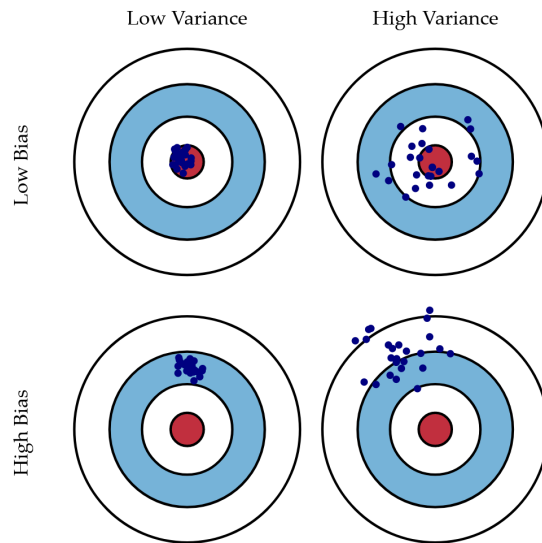


Figure 2.3.3: Graphical representation of bias and variance [FR].

Generally, it is considered that overfitted models do not generalise well and so is advantageous to avoid such situations. This view has recently become more challenged with the usage of extremely accurate complex models that perform well on both training and testing sets [BHMM19]. These models take advantage of large amounts of training data (amongst other things) and so are less likely to overfit.

As with local optimum, the specifics of overfitting are not important to this work,

simply the acknowledgment of further challenges to finding a perfect answer in complex settings.

Chapter 3

Explainable Machine Learning

Contents

3.1	Introduction and Motivation	23
3.2	Post-hoc Interpretability	26
3.3	SHAP (SHapley Additive exPlanations)	28

In the pursuit of greater performance, recent techniques such as Deep Neural Networks have increased in complexity by orders of magnitudes [BMR⁺20, ADRDS⁺20]. A simple example of this can be seen in the parameter count of each generation of the GPT model where the already complex Generative Pre-trained Transformer (GPT) model with 1.3 billion parameters was eclipsed just 2 years later by GPT-3 with 175 billion parameters. The downside of this increased complexity is generally the greater difficulty in interpreting reasoning from these complex models’ predictions [RSG16]. As such, the field of explainable machine learning has seen vastly increasing attention in recent years [ADRDS⁺20]. In this chapter we aim to briefly overview the field of explainable machine learning, discussing motivations for the field and defining key aims of explainable machine learning systems. We then focus on post-hoc interpretability methods, which increase the interpretability of a predictor, as they are of greater relevance to our work. Finally, we discuss SHAP, the specific interpretability method we utilise at the centre of our work. This chapter was influenced throughout by the book interpretable machine learning [Mol19].

3.1 Introduction and Motivation

As discussed in Chapter 2, machine learning models are standardly evaluated by assessing their performance on a test set using some error metric. Additionally, as discussed in section 1.1, standard test performance evaluations can provide misleading results. “The problem is that a single metric, such as classification accuracy, is an incomplete description of most real-world tasks” [DVK17]. In certain problems we wish not only to

to generalise than those that are right for the wrong reasons [RHDV17]. Greater interpretability makes it easier to understand the reasoning behind a prediction and as such whether the reason is right or wrong.

Trust: Trustworthy systems are those that humans are confident will not fail, such as air traffic control systems. Interpretable systems are easier to personify and as such have greater social acceptance [Mol19].

Interpretability can either be achieved intrinsically, where the model is interpretable by design, or by using post-hoc methods, where models are explained by external methods after they have been trained [WL20].

When creating a machine learning system a trade off between accuracy and interpretability must often be considered [CLG⁺15]. This is because as model complexity increases, prediction performance tends to increase whilst interpretability tends to decrease. Figure 3.1.2 shows the accuracy-interpretability trade-off, with complex models such as deep learning and ensembles boasting high accuracy whilst providing poor interpretability, and simple models such as linear regression and decision trees providing lower accuracy but higher interpretability.

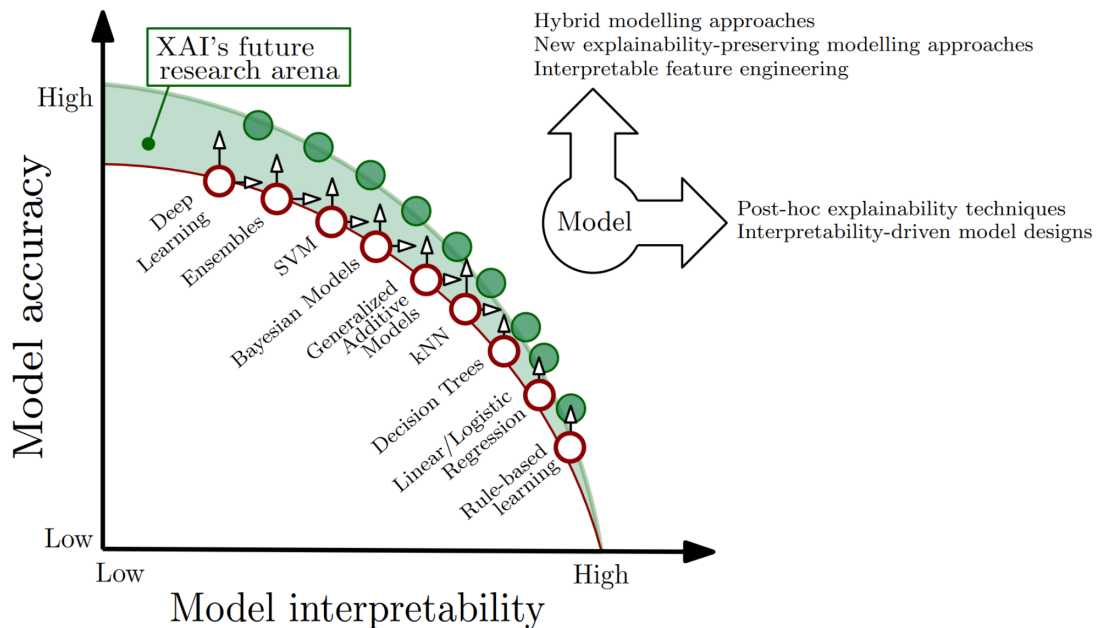


Figure 3.1.2: Trade-off between model accuracy and interpretability [ADRDS⁺20].

“An explanation usually relates the feature values of an instance to its model prediction in a humanly understandable way” [Mol19]. An explanation aims to answer (explain) why a prediction had the result it did [Mil19], in other words explanations are “the currency in which we exchanged beliefs” [Lom06]. Thus, explanations aim to increase the interpretability of a model by explaining its predictions. An explanation of a predictor’s prediction can itself be viewed as a model, and as such can be called an ex-

planation model [LL17]. For models of low complexity (and thus high interpretability) the model itself is the best explanation, as it is easy to understand and doesn't omit any information. Complex models are however, normally, poorly interpretable and as such the explanation model must be a simpler approximation of the original model [LL17].

3.2 Post-hoc Interpretability

As discussed in Section 3.1, post-hoc interpretability methods explain predictors externally from the model architecture itself. Before going into further detail of post-hoc interpretability, we take a high level view of the process. As shown in Figure 3.2.1, many levels of abstraction occur before an explanation is shown to a human. The first level is the world, or rather the ground truth. This is not necessarily the physical world but could be an abstract environment, such as in a targeted advertising system and as such we prefer the term ground truth to world. The second level is data, where we capture digital data from the ground truth. This data abstracts the world into a machine understandable format. Next, the black box model learns from the data. As discussed in Section 2.1 the model attempts to learn a mapping $f(x)$ that maps input x to output y . This process leaves us with a trained black box model (predictor) using which predictions can be made. Once the model has been trained, post-hoc interpretability methods can be used to explain the outputs of the predictor. These methods aim to somehow explain why certain predictions have the results they do. Finally, these methods generally help humans to interpret the results given by a predictor. Of course this is not always the case, sometimes humans are not involved in the process at all.

Post-hoc methods can either be model-specific, where they can only explain predictions from one model architecture, or model-agnostic, where they can explain any model architecture [AB18].

Benefits of model-agnostic methods over intrinsically interpretable models include [RSG16]:

Model Flexibility: Not held back to any singular architecture. In real-world applications, model architecture selection is often based on accuracy alone, without any consideration of interpretability. Model agnostic interpretability methods can help mitigate the effects of the complexity-interpretability trade-off [ADRDS⁺20]. These methods don't affect model accuracy, but do increase interpretability for complex models.

Explanation Flexibility: Not limited to any form of explanation. Explanations can come in many forms, each that may be more or less suited to any given problem. Further to this, different users have different thresholds for what they consider interpretable. A computer scientist may consider a Bayesian model interpretable, whereas a lay person may prefer simple decision trees. For example, a rule based system was considered uninterpretable by one user as it contained 41 rules, whereas another user analysed a rule based system containing 29,050 rules without issue [Fre14].

Representation Flexibility: Different feature representations can be used. Many models learn from uninterpretable features such as word embeddings to make their

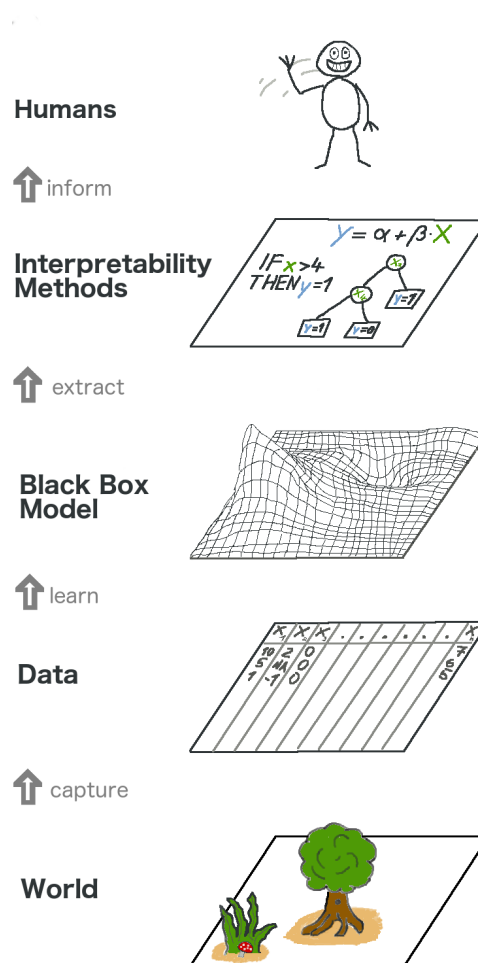


Figure 3.2.1: The 'big picture' of post hoc interpretability methods [Mol19]. Many levels of abstraction occur before an explanation is shown to a human.

predictions. Model agnostic methods allow for feature representations different to those used within the model [Mol19]. In the context of learning from word embeddings, a model agnostic method could use words rather than the vector embedding in the explanation.

Lower cost to switch: Switching or updating model is trivial. The model architecture used in a pipeline may need to be changed for a variety of reasons, for example a more accurate model is developed. In these cases model-agnostic methods allow users to understand the explanation of the new model without any further training.

Comparison of models with different architectures: As the same explanation method can be used for multiple different model architectures, comparisons between them are easier.

Model-agnostic methods can be local, where individual predictions are explained, or global, where the average affects of features on predictions are described [Mol19].

3.3 SHAP (SHapley Additive exPlanations)

Shapley Additive exPlanations (SHAP) is a feature attribution algorithm developed in explainable AI. It is based on the cooperative game theory concept of Shapley values, assigned to each feature of a data instance. Shapley values were introduced in the 1950s by Lloyd Shapley [Sha16] who later won a Nobel prize in economics [Rot88]. They are defined to answer the question: “What is the fairest way for a coalition to divide its payout amongst the players”. Shapley values assume players should receive payout proportional to their contribution.

A fair distribution of payout with Shapley values is defined as following three axioms, Symmetry, Null Player, Additivity.

Symmetry: Players that make equal contributions should receive equal payouts.

Null Player: A player that does not contribute towards the payout should receive 0 contribution.

Additivity: If a game is split into sub-games, the sum of all sub-games valuation should be the same as the original game.

Definition 3.3.1 Let N be a finite list of players indexed by i . Let $v : 2^N \rightarrow \mathbb{R}$ be a utility function that associates each coalition $S \subseteq N$ with a payout $v(S)$. We therefore represent a coalition game as the pair (N, v) . The *Shapley value* $\phi_i(N, v)$ of player i in a given coalition game (N, v) is calculated as [LL17]:

$$\phi_i(N, v) = \frac{1}{|N|!} \sum_{S \subseteq N \setminus \{i\}} |S|!(|N| - |S| - 1)! [v(S \cup \{i\}) - v(S)] \quad (3.1)$$

Example 3.3.1 Consider two business people A and B who decide to form a partnership. In order to calculate how their earnings should be divided between them, they decide to use Shapley values. Whilst in business by themselves A earnt £1000 per week, whereas B earnt £2000 per week. When they form their partnership, the new venture earns £4000 per week.

They use their earnings as their utility function v such that: $v(A) = 1000$, $v(B) = 2000$, $v(A, B) = 4000$. If neither of them are in business their earning will be 0, that is $v(\emptyset) = 0$. As $S \subseteq N = \{A, B\}$, it can only be of length 0 (when $S = \emptyset$), or 1 (when $S = A$ or $S = B$), $|S|!(|N| - |S| - 1)!$ will always equal 1 and so we omit it from the following calculations for the sake of readability.

Using equation 3.1 they calculate their Shapley values as:

$$\begin{aligned}
 \phi_A(N, v) &= \frac{1}{|\{A, B\}|!} ([v(\emptyset \cup \{A\}) - v(\emptyset)] + [v(\{B \cup A\}) - v(B)]) \\
 &= \frac{1}{2!} ([v(\{A\}) - 0] + [v(\{A, B\}) - 2000]) \\
 &= \frac{1}{2} ([1000 - 0] + [4000 - 2000]) \\
 &= 1500 \\
 \phi_B(N, v) &= \frac{1}{|\{A, B\}|!} ([v(\emptyset \cup \{B\}) - v(\emptyset)] + [v(\{A \cup B\}) - v(A)]) \\
 &= \frac{1}{2!} ([v(\{B\}) - 0] + [v(\{A, B\}) - 1000]) \\
 &= \frac{1}{2} ([2000 - 0] + [4000 - 1000]) \\
 &= 2500
 \end{aligned}$$

Meaning for a fair distribution based on Shapley values, of the £4000 earned, A should receive £1500, whilst B should receive £2500.

To use Shapley values in a machine learning context, SHAP considers features as players and the prediction outcome as the total payout. This allows a Shapley value to be assigned to each feature of a given prediction. In this setting, the Shapley value of a feature represents its contribution towards the prediction output. This list of Shapley values produced by SHAP explains an individual prediction, as such SHAP is considered a local explanation method. If we relate this back to example 3.3.1, where person A and B are feature values, and the prediction target is the earning of their combined venture. If we assume the base value (which we define below) is 0, then the SHAP explanation could be the list [1500, 2500]. This represents the contributions of A and B towards the prediction of 4000. SHAP is also model-agnostic as it does not rely on any internal mechanism of a model, treating them only as a black boxes, dealing with its inputs and outputs. For a data instance x , SHAP computes the marginal contribution of each feature x_m to the prediction of x . SHAP is an additive feature attribution method, as such the summation of a SHAP explanation model $\Pi(x)$ attributions approximates the output of the prediction it explains $f(x)$.

Definition 3.3.2 Additive feature attribution methods can be generalised as an explanation g model that is a linear function of binary variables z [LL17].

$$g(z) = \phi_0 + \sum_{j=1}^M \phi_j z_j \quad (3.2)$$

where $z \in \{0, 1\}^M$, M the number of features, and the attribution $\phi_i \in \mathbb{R}$.

Example 3.3.2 Consider the prediction problem shown in figure 3.3.1. Based on four features; Age=65, Sex=F, BP=180, and BMI=40 some black box model predicts an output of 0.4. To explain this prediction, we use SHAP to explain how each of these features contributes towards the prediction outcome. The base rate (value) is the value that would be predicted if none of the prediction values were known [LL17]. In other words, the base value is the mean prediction. Both BMI and BP have a positive contribution of 0.1, meaning they cumulatively increase the prediction by 0.2. Sex has a large negative contribution towards the prediction of -0.3 , decreasing our prediction output. Finally age has a large positive contribution towards the prediction, giving our final prediction output of 0.4. SHAP shows us both the magnitude of the contribution and whether it is a positive or negative.

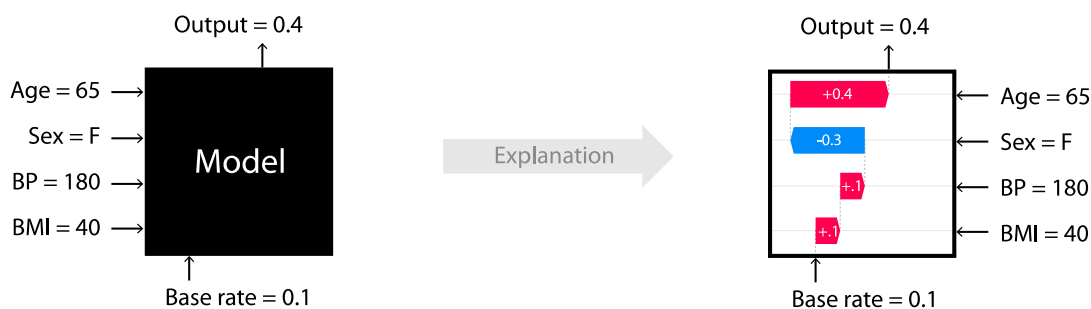


Figure 3.3.1: Example of SHAP output, explaining the contributions of four features towards a prediction [LL17].

SHAP itself can explain the output of any machine learning method, however different implementations of SHAP exist, some of which are model-specific to yield performance benefits.

Kernel SHAP: A model-agnostic method that can estimate the SHAP values of any model. As kernel SHAP is model agnostic it cannot leverage any model architecture detail for performance and as such is slower than model specific implementations.

Tree SHAP: An exact model-specific method that computes SHAP values for tree-based models (trees or tree ensembles) [LEC⁺20]. Tree SHAP benefits over model-agnostic implementations not only in that it is exact rather than an estimation, but is also faster.

Deep SHAP: A model-specific implementation of SHAP that estimates SHAP values for deep learning models. Its implementation is faster than model-agnostic implementations.

Chapter 4

Metrics

Contents

4.1	Distance, Similarity and Correlation Metrics	31
4.2	Euclidean Distance	32
4.3	Pearson Correlation Coefficient	34
4.4	Cosine Similarity	35
4.5	Kendall Rank Correlation Coefficient	37

This section will discuss different metrics of distance, similarity and correlation. Throughout our work we use metrics to quantitatively evaluate the spread of a set of explanations. We define our usage of the term metric, metric space and how we represent them in our work. After this, we detail four metrics; Euclidean, Pearson, Cosine and Kendall. We discuss the different relationships these metrics model, and give examples of each.

4.1 Distance, Similarity and Correlation Metrics

Generally a metric is “a standard of measurement“ [MW]. In this work the term metric is used to represent some function that quantitatively describes a relationship between vectors (which we generally represent as lists).

Definition 4.1.1 Let L be a given set. Let d be a finite function on the Cartesian product $d : L \times L \rightarrow [0, \infty)$ such that:

$$d(x, x) = 0; x \neq y \implies d(x, y) > 0; \tag{4.1}$$

$$d(x, y) = d(y, x); \tag{4.2}$$

$$d(x, y) + d(y, z) \geq d(x, z); \tag{4.3}$$

We say that d is a *distance function* (metric) in *metric space* L . The elements of a *metric space* are called *points*. These *points* can be of any dimensionality and so can take many forms such as, vectors and lists. The distance between two points a and b is $d(a, b)$.

These definitions are from [ČK69].

In this work the terms of distance and similarity are seen as opposites. Where a distance metric increases as two points are further from each other in a metric space, a similarity metric increases as two points are ‘closer’ in some way. If vectors are distant from each other they are very different. If vectors have high similarity they are similar.

The final type of metric we cover in this chapter is correlation. Correlation metrics do not have associated metric spaces as similarity and distance metrics do, as such they do not follow the axioms in equations (4.1, 4.2, 4.3). The correlation between two random variables “is a measure of the extent to which a change in one tends to correspond to a change in the other” [CC09]. Broadly, this means correlation is a measure of association or similarity between two variables. Generally in this work these variables come in the form of lists or vectors. Correlations between random variables are said to be stronger (or higher) when the variables are more associated, and weaker (or lower) when they are less associated.

4.2 Euclidean Distance

A common way to find the shortest distance between a pair of two-dimensional Cartesian coordinates is to construct a right angle triangle such that the hypotenuse is the distance. Solving for this distance is then just a matter of calculating the length of the two other sides of this triangle, which are just the differences between the x and y components of each point, and then applying the Pythagorean theorem.

Example 4.2.1 As shown in Figure 4.2.1, the Euclidean distance of two 2d points can be found by constructing a right angle triangle where the distance is the hypotenuse. Finding the length of hypotenuse (distance) is then simply achieved using the Pythagorean theorem.

Consider two points; $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$.

The horizontal side is the difference between x values; $x_2 - x_1$.

The vertical side is the difference between y values; $y_2 - y_1$.

Using the Pythagorean theorem $a^2 + b^2 = c^2$ we can substitute a and b as the two calculated sides to solve for the hypotenuse (in this case distance) c .

\therefore Let $a = x_2 - x_1$, $b = y_2 - y_1$, and $c = d(p_1, p_2)$

Thus, $d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

The Euclidean distance aims to find the distance between two objects in Euclidean space. It is a generalised version of finding distance using Pythagorean theorem. This means rather than points only in 2d, the distance between points of any dimensionality can be found.

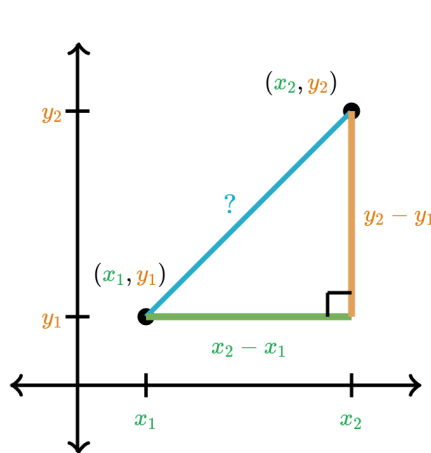


Figure 4.2.1: Finding the distance between two points [Kha16].

Definition 4.2.1 Consider two vectors x and y , which we represent as two lists such that $x = [x_1, \dots, x_n]$ and $y = [y_1, \dots, y_n]$. The *Euclidean distance* $d(x, y) \in [0, \infty)$ between x and y can be expressed as:

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (4.4)$$

Euclidean distance is not scale-invariant, as the Pythagorean theorem itself is not scale-invariant. This means that Euclidean distances are skewed if some vector components are far larger in magnitude than others [WFHP16, p. 135-136]. To counteract this effect, data can be normalised before calculating the metric.

The Euclidean distance also struggles more than other metrics in a setting of high dimensionality. This is because high dimensional data tends to become more sparse with a greater amount of data near the limits resulting in distances computed tending towards a constant and as such are less meaningful [AHK01].

Example 4.2.2 Consider a professor aiming to assign students to groups based upon similar module grades. From three students S_1 , S_2 , and S_3 , a professor aims to pick the pair of students with the lowest Euclidean distance of module grades from one another.

Consider three lists S_1 , S_2 , and S_3 such that:

$S_1 = [72, 67, 75, 83, 81]$, $S_2 = [64, 61, 63, 72, 68]$, and $S_3 = [84, 71, 79, 81, 86]$ are five module grades of the three students.

Three possible pairs exist between the three students, (S_1, S_2) , (S_1, S_3) , and S_2, S_3 .

Using Equation 4.4, $d(S_1, S_2) = \sqrt{\sum_{i=1}^n (S_{1_i} - S_{2_i})^2}$

$$d(S_1, S_2) = \sqrt{534} \approx 23.1$$

$$d(S_1, S_3) = \sqrt{205} \approx 14.3$$

$$d(S_2, S_3) = \sqrt{1161} \approx 34.1$$

It therefore makes sense to pick the pair of students S_1 and S_3 together, as their Euclidean distance is the smallest of the three possible pairs.

4.3 Pearson Correlation Coefficient

As mentioned in Section 4.1 the correlation between two random variables “is a measure of the extent to which a change in one tends to correspond to a change in the other” [CC09]. In a general statistical sense correlation is often thought of as a linear relationship between variables, represented by a metric such a Pearson correlation coefficient.

“Pearson’s correlation coefficient (r) is a measure of the linear association of two variables” [Kir08]. It is defined as the ratio between the covariance of two variables (lists) and the product of their standard deviations. The Pearson correlation coefficient is bounded between -1 and 1. A value of 1 means a perfect positive linear correlation, 0 is no linear correlation, and -1 is a perfect negative correlation.

Definition 4.3.1 Consider two lists X and Y , such that $X = [x_1, \dots, x_n]$ and $Y = [y_1, \dots, y_n]$. The *Pearson correlation coefficient* $r_{X,Y}$ of X and Y is defined as:

$$r_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \quad (4.5)$$

Where $\text{cov}(X, Y)$ is the covariance of lists X and Y , and σ_X, σ_Y the standard deviations of X and Y respectively. Thus, $r_{X,Y} \in [-1, 1]$ is defined as:

$$r_{X,Y} = \frac{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}} \quad (4.6)$$

where n is the sample size, \bar{X}, \bar{Y} are the mean of X and Y respectively.

Figure 4.3.1 shows examples of distributions with their Pearson correlation coefficient above. These distributions are formed by projecting the two lists onto each axis, making points $P = p_1, \dots, p_n$ where $p_i = (x_i, y_i)$ from lists x and y . The first row of the figure shows how r increases in magnitude the tighter the linear grouping of a distribution. As shown in the second row the magnitude (gradient in the image) of the values is not relevant to r , only the grouping. The third row illustrates how Pearson correlation only

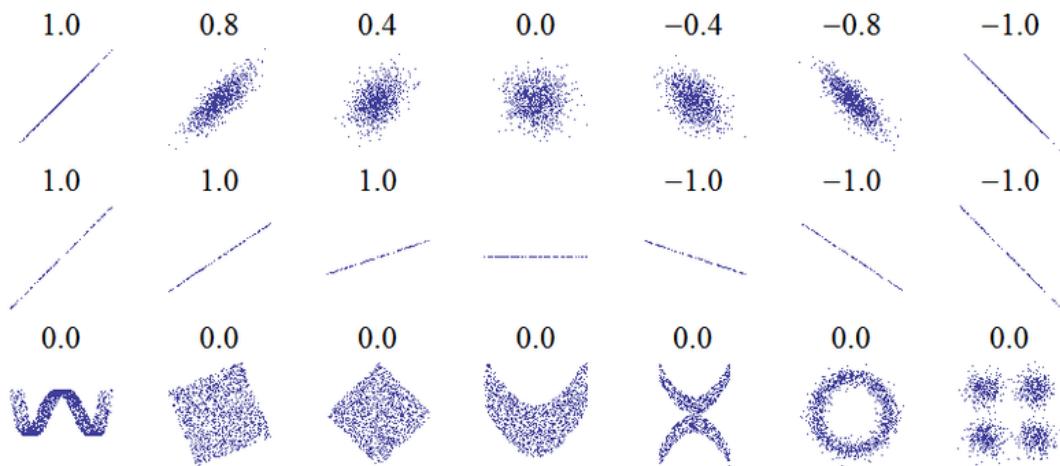


Figure 4.3.1: Example distributions and their Pearson correlation coefficient above [aEW07].

examines linear relationships. As such, non linear relationships that are clearly related can give $r = 0$ as there is no linear correlation.

Example 4.3.1 Consider calculating the correlation between the height and weight of three individuals.

Consider two lists H and M such that:

$H = [190, 162, 152]$, $M = [90, 60, 45]$ as heights and masses respectively.

Giving $\bar{H} = 168$, $\sigma_H \approx 19.7$ and $\bar{M} = 65$, $\sigma_M \approx 22.9$.

We know $cov(H, M) = \frac{1}{n-1} \sum_{i=1}^n (H_i - \bar{H})(M_i - \bar{M})$ from equation 4.6

So, $cov(H, M) = \frac{1}{2}(550 + 30 + 320) = 450$

Thus $r_{H,M} \approx \frac{450}{19.7 \times 22.9} \approx 0.997$ using equation 4.5.

A Pearson coefficient of 0.997 suggests a strong positive linear correlation, so as H tends to increase, so does M .

4.4 Cosine Similarity

Cosine similarity is the cosine of the angle between two vectors of inner product space [HPK11]. Its values are bound by cosine; $-1 \leq \cos(\theta) \leq 1$. Two similar vectors with $\theta = 0$ give a cosine similarity of 1. Two opposite vectors with $\theta = 180$ give a cosine similarity of -1. Cosine similarity is also sometimes written as a percentage scale of similarity $-100\% \leq d(x, y) \leq 100\%$.

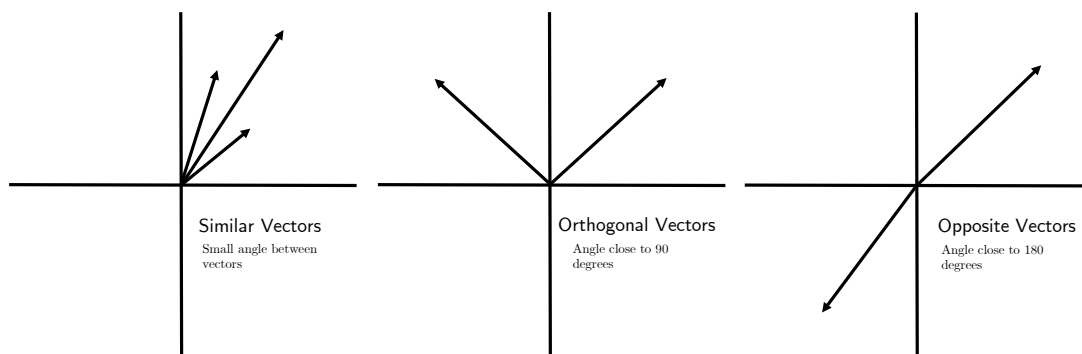


Figure 4.4.1: Example of how angle between vectors are the basis of cosine similarity, figure based on [TPM17].

Definition 4.4.1 The *cosine similarity* $d \in [-1, 1]$ of two vectors x and y at angle θ from each other can be calculated as the ratio between the dot product of the two vectors and the product of their magnitudes:

$$d(x, y) = \cos(\theta) = \frac{x \cdot y}{\|x\| \|y\|} \quad (4.7)$$

Or more specifically,

$$d(x, y) = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}} \quad (4.8)$$

where n is the cardinality of each vector, or in our representation of vectors as lists, the number of elements in each list.

Example 4.4.1 As shown in Figure 4.4.1, cosine similarity doesn't take the magnitude of vectors into account, only the angle between them.

Consider two lists x and y such that:

$$x = [1, 2, 3] \text{ and } y = [2, 4, 6]$$

$$\text{Thus, } x \cdot y = (1 \times 2) + (2 \times 4) + (3 \times 6) = 28$$

$$\text{and } \|x\| = \sqrt{1^2 + 2^2 + 3^2} = \sqrt{14}, \quad \|y\| = \sqrt{2^2 + 4^2 + 6^2} = \sqrt{56}$$

$$\text{So } d(x, y) = \frac{28}{\sqrt{14} \times \sqrt{56}} = 1$$

These two vectors are seen as similar to cosine similarity, even though the magnitude of y is twice that of x , as they are at the same angle in inner product space.

Example 4.4.2 Also shown in figure 4.4.1, vectors are orthogonal if their cosine similarity is close to 0 ($\theta \approx 90^\circ$) and opposite if their cosine similarity is close to -1 ($\theta \approx 180^\circ$).

Consider three lists X , Y and Z such that:

$$X = [4, 1, 3, 2, 3], Y = [-1, -2, -3, 0, 5] \text{ and } Z = [2, 4, 6, 0, -10]$$

$$\text{Thus, } X \cdot Y = 0, \|X\| = \sqrt{39}, \|Y\| = \sqrt{39}$$

So, $d(X, Y) = 0$ meaning X and Y are orthogonal

$$X \cdot Z = 0, \|X\| = \sqrt{39}, \|Z\| = \sqrt{156}$$

So, $d(X, Z) = 0$ meaning X and Z are also orthogonal

$$Y \cdot Z = -78, \|Y\| = \sqrt{39}, \|Z\| = \sqrt{156}$$

$$\text{So, } d(Y, Z) = \frac{-78}{\sqrt{39} \times \sqrt{156}} = -1 \text{ meaning } Y \text{ and } Z \text{ are opposite}$$

From these cosine similarities we can deduce that X bisects the angle between Y and Z , as Y and Z are both orthogonal to X whilst opposite to each other.

4.5 Kendall Rank Correlation Coefficient

Kendall rank correlation coefficient [Dod08] evaluates the degree of similarity between two lists of ranks.

Definition 4.5.1 Let $U = [u_1, \dots, u_k]$ be a collection of k objects $u_i, 1 \leq i \leq k, k \in \mathbb{N}$. Let $r : U \rightarrow O$ be a function that provides each object $U_i \in U$ some value in a totally ordered sequence O .

We order the objects $u_i = o_i$ into O :

$$O = [o_1, \dots, o_k]$$

Such that $o_i \geq o_j$ for $i < j$.

Note there be many such sequences O that can be created from U .

We define the *rank* r_i of an object u_i to be the function that provides its index in the totally ordered set O .

Example 4.5.1 Consider the list $U = [10, 3, 8, 5]$ from which we wish to produce the list of ranks R . To do so we first create the totally ordered sequence $O = [10, 8, 5, 3]$. Thus $R = [1, 4, 3, 2]$.

Kendall rank correlation coefficient, τ , considers how many inversions of pairs would be needed to transform one list of ranks into the other. The coefficient $\tau \in [-1, 1]$ can be interpreted in a similar manner to Pearson correlation; the greater the magnitude of the coefficient the greater the association between variables. The direction of

the relationship is indicated by the sign of the coefficient; a value of -1 meaning the two lists of ranks are perfectly inverted (discordant) and 1 meaning perfect agreement (concordant).

Definition 4.5.2 Consider an ordered set $O = [o_1, \dots, o_n]$ of N objects. O can be decomposed into the set δ_o containing $\frac{1}{2}N(N-1)$ ordered pairs, such that:

$$\delta_O = \{[o_1, o_2], [o_1, o_3], \dots, [o_{n-1}, o_n]\} \text{ (Assuming } N \geq 3)$$

Let $P = [p_1, \dots, p_n]$ be a similarly ordered set to O such that P imposes a different order onto the same objects as O . P can similarly be decomposed into a set of ordered pairs, such that:

$$\delta_P = \{[p_1, p_2], [p_1, p_3], \dots, [p_{n-1}, p_n]\} \text{ (Assuming } N \geq 3)$$

The *symmetric difference distance* is defined as the number of different pairs within these decompositions [Abd07]. The *symmetric rank distance* $d(O, P)$ of the sets O and P is the *symmetric rank difference* of the decompositions δ_O and δ_P , thus $d(O, P) = |\delta_O \setminus \delta_P|$.

Example 4.5.2 Let $X = [a, b, c, d]$ and $Y = [a, b, d, c]$.

To calculate the symmetric rank distance $d(X, Y)$ we first create the decompositions δ_X and δ_Y such that:

$$\delta_X = \{[a, b], [a, c], [a, d], [b, c], [b, d], [c, d]\}$$

$$\delta_Y = \{[a, b], [a, d], [a, c], [b, d], [b, c], [d, c]\}$$

$$\text{Thus, } \delta_X \setminus \delta_Y = \{[c, d], [d, c]\}$$

$$\therefore d(X, Y) = |\delta_X \setminus \delta_Y| = 2$$

Definition 4.5.3 Let $X = [x_1, \dots, x_n]$ and $Y = [y_1, \dots, y_n]$ be two lists of N ranks. Let δ_X and δ_Y be the ordered pair decompositions of X and Y as shown in definition 4.5.2. The *Kendall rank correlation coefficient*, τ , is calculated as:

$$\tau = d(X, Y) = 1 - \frac{2 \times d(\delta_X, \delta_Y)}{N(N-1)} \quad (4.9)$$

Example 4.5.3 Consider a competition where three dancers are ranked by two judges. Organisers of the competition are concerned that the judges have given conflicting ranks to contestants. To analyse these claims they employ Kendall rank correlation to the ranks produced by the judges.

Let r_1 and r_2 be the rankings produced by the two judges such that:

$$r_1 = [1, 2, 3] \text{ and } r_2 = [1, 3, 2]$$

Therefore $N = 3$

To form δ_1 and δ_2 the respective rankings must be decomposed into ordered pairs such that their length is $\frac{1}{2}N(N - 1)$ which in this case is 3.

$$\delta_1 = \{[1, 2], [1, 3], [2, 3]\}, \delta_2 = \{[1, 3], [1, 2], [3, 2]\}$$

The different pairs in δ_1 and δ_2 are $\delta_1 \setminus \delta_2 = \{[2, 3], [3, 2]\}$

So, $|\delta_1 \setminus \delta_2|$ and $\therefore d(\delta_1, \delta_2) = 2$

Thus using equation 4.9 $\tau = 1 - \frac{2 \times 2}{6} \approx 0.33$.

A Kendall coefficient of 0.33 suggests a slight agreement between the two ranks.

As with all of these metrics, the context of use is important for Kendall. The degree to which how similar lists should be (and therefore the threshold for an acceptable coefficient) will depend on the use case. In the case of classifying the likelihood of plagiarism (very naively), it may be the case that only lists that require very few inversions of pairs to translate to each other are considered. The lists would have a very high Kendall correlation coefficient, such as 0.95. On the other hand, consider the case of finding opposing judges for a competition. Here it may make sense to look for judges that have very different rankings and as such have a negative Kendall correlation coefficient. In this case, it may make sense to classify judges with a coefficient of less than -0.2 to strongly disagree with one another.

In this regard, metrics such as Cosine, Kendall and Pearson can also perform significance tests. These can help to show the statistical significance of the metric result, that is to quantify the likelihood of chance effecting these values. We do not discuss them within this chapter, as we aim only to compute metrics to quantify the differences between explanations and do not explore the significance of these metrics. Additionally, the sample size used for a significance test greatly effects the results [Ell10], generally a greater sample size allows for more confidence in statistics computed from it. This would mean each result of ours would need different analysis dependent on sample size, which is something we aim to avoid in our general approach.

Chapter 5

Related Work

Contents

5.1	Stress Tests	41
5.2	Explanation Quantification	43
5.3	Explanation Influenced Optimisation	44
5.4	The Rashomon Effect	46
5.5	Prediction Variance	47

In this section we cover some of the related work to the problem of quantifying underspecification. We begin this by discussing stress tests, the way in which the paper that initially presents underspecification [DHM⁺20] proves that underspecification is prevalent within many machine learning settings. We follow this with methods to quantify the quality of explanations against desiderata for a given problem. This is because our method can be seen as another such technique that evaluates against underspecification, rather than one of these desiderata groups. We then move into techniques which use explanation correctness in the choosing of predictors f from model class \mathcal{F} . Although there is no mention of underspecification in this literature, optimising based on explanation should, in theory, reduce the possible amount of similarly performing predictors with different explanations. Under the same assumption as we make, that two predictors give the similar explanations when they encode similar inductive biases, reducing the number of differing explanations should reduce the likelihood of underspecification. Following this, we discuss the Rashomon effect, as it is in line with the problem of underspecification, specifying multiple similarly performing predictors that we cannot effectively choose between. Finally, we discuss techniques that aim to quantify the variance between possible well-performing predictors for a given problem.

5.1 Stress Tests

As briefly discussed in section 1.1, D’Amour [DHM⁺20] focuses on the problem of underspecification affecting a predictors’ credibility. To show the prevalence of under-

specification in machine learning, they attempt to disprove the expected behaviour of predictors. To this end, stress tests, which are “evaluations that probe a predictor by observing its outputs on specifically designed inputs” [DHM⁺20] are used. This section follows [DHM⁺20] as the principal work in underspecification. As discussed in section 1.1 pipelines that evaluate their predictors based on test set performance can often fail to identify underspecification. As such, stress tests probe a predictor in more ways than simple test set performance evaluation.

By using stress tests to prove non expected behaviour of predictors, [DHM⁺20] shows underspecification is prevalent in many state of the art applications of machine learning. Below we cover the three classifications of stress tests utilised for their experiments.

Stratified Performance Evaluations Stratified performance evaluations split data into subgroups (strata) and then tests whether predictors perform similarly across all strata. Different values for the feature are then chosen, compared against others via their test performance. For example, facial analysis algorithms can be evaluated for fairness across different races by splitting images into subgroups of skin types and gender [BG18]. A fair predictor should perform equally well across all of these strata, as such performance differences across strata suggest a lack of fairness in the predictor.

Shifted Performance Evaluations Shifted performance evaluations use a label preserving function $s(D) \rightarrow D'$ to create a new data distribution D' from the original distribution D , such that $|D| = |D'| = N, L(D_i) = L(D'_i), \forall i \in [1, N]$ where $L(D_i)$ is the label of data instance i of D . The shifting function s should only edit the dataset such that no casual data is changed. An example of this is in the small noise added in adversarial examples as shown in Figure 3.1.1. The performance of a given predictor f is then tested on both D and D' , an ideal predictor should have a similar average performance across both datasets. The larger the difference in performance, the less the model generalises, and so has less credibility. These evaluations add different noise to a dataset, meaning that predictors that memorise the training data perform more poorly on D' than D . Such dataset shifts are often used in data augmentation and is common in complex machine learning pipelines [SK19], normally to improve the quality of the training dataset. For example, if we wanted to classify pictures of cats and dogs, we may shift our dataset by increasing the exposure of some images, randomly cropping some images, or adding noise. These changes are label preserving, that is, they shouldn't change whether any given image shows a cat or dog, so shouldn't affect the class (label). As such using a shifted test dataset, if a performance difference is observed between this shifted dataset and the original, the model may be basing its prediction on irrelevant details, thus may not effectively generalise. Evaluations of this kind are a major component of evaluating machine learning in the literature [HD19].

Contrastive Evaluations As shifted evaluations focus on the distribution level they can hide errors particularly in small groups of predictions. For example, if again we consider the classification of images of dogs and cats, it may be the case all images of

Chihuahuas are classified as cats. If only a few examples of Chihuahuas exist in a large dataset, this effect is unlikely to be large enough to be noticed at a distribution level. Contrastive evaluations support localised analysis meaning we can analyse inductive biases (the set of assumptions the model uses to predict) for any data instance. Evaluations of this type do not hide results of individual predictions as shifted performance evaluations do, as they are editing individual or small sets of data instances rather than the whole distribution. Contrastive evaluations aim to change an input observation and then calculate the significance of this effect on the output of a predictor.

5.2 Explanation Quantification

As discussed in Chapter 3 explanations have no firm accepted definition. As such, there is no accepted ‘correct’ explanation. Because of this lack of ground truth, we cannot standardly evaluate the correctness of explanations. With no accepted standard, researchers use a plethora of techniques to compare explanation methods, often reflecting performance against some desiderata [HWB⁺22], such as robustness [MSM18, YHS⁺19, AMJ18b] or faithfulness [BWM20, NM20, AMJ18a, BBM⁺15, RH20].

One such method to quantify the quality of explanations closely related to this work is explanation continuity. Explanation continuity is high when similar data points yield similar explanations [MSM18]. Explanation continuity can be seen as a measure of predictor robustness (discussed in Section 3.1). The explanation continuity Φ_f of the predictor f can be quantified by finding the largest variation between feature attribution explanations $\mathcal{E}(f, x)$ and $\mathcal{E}(f, x')$ for all data points $x, x' \in D$ in the input dataset D .

Specifically, Φ_f can be calculated as:

$$\Phi_f = \arg \max_{x \neq x'} \frac{|\mathcal{E}(f, x) - \mathcal{E}(f, x')|}{|x - x'|} \quad (5.1)$$

Another Explainable AI (XAI) quantification metric is faithfulness correlation, which aims to quantify how faithful a predictor is by studying the correlation between explanations and prediction differences after modifying input data [BWM20].

To achieve this a subset S of feature indices from the data instance x with d features are taken such that $S \subseteq \{1, 2, \dots, d\}$, $x_S = \{x_i, i \in S\}$ such that x is split into selected features x_S and non selected features x_c , $x = x_S \cup x_c$. $x_{[x_S=\bar{x}_S]}$ denotes a data instance x where each feature in x_S is set to the reference value \bar{x}_S , while the non selected indices x_c are not changed.

The faithfulness correlation for a data instance x is the correlation (*corr*) between the explanation model g and the difference in prediction output from f on x and the baseline data instance $x_{[x_i=\bar{x}_i]}$.

$$\mu_F(f, g; x) = \text{corr} \left(\sum_{S \in \binom{[d]}{|S|}} g(f, x)_i, f(x) - f(x_{[x_S=\bar{x}_S]}) \right) \quad (5.2)$$

However, as there is no accepted standard of these metrics, or a reliable way to compare one to another, it is difficult to choose between them. For this reason, Quan-

tus [HWB⁺22] bundles together many XAI quantification techniques organised by the primary desiderata they represent. This enables researchers to holistically quantify explanations against different desiderata using multiple techniques. An example of this can be seen in Figure 5.2.1. Figure 5.2.1a shows a standard example of a qualitative review, where a researcher must visually compare the images produced. Each highlighted area shows where each respective method identifies important features. Figure 5.2.1b shows a holistic evaluation in the form of a radar plot. Each of the five points represents how explanations perform to a respective desiderata group by aggregating many quantification techniques. This allows researchers to move easily compare explanations against different desiderata.

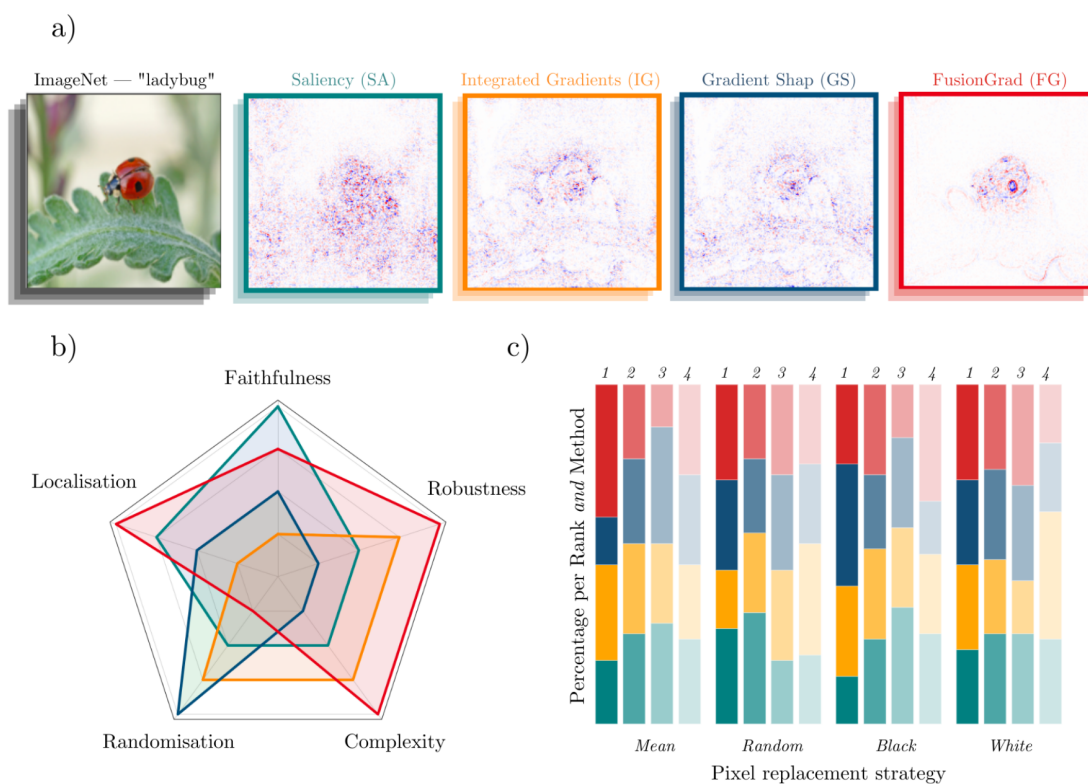


Figure 5.2.1: a) visual representations of gradient based methods, an example of a qualitative comparison. b) radar plot of aggregate quantification techniques sorted into their desiderata groups. [HWB⁺22]

5.3 Explanation Influenced Optimisation

“Right for the right reasons” [RHDV17] adds a component representing explanation accuracy to the loss function used during training. This means that during optimisation alongside consideration of test set performance, test set explanation performance is also considered. The aim of this is to produce predictors that are “right for the right reasons”,

Table 5.3.1: Gradient vs LIME runtime per explanation [RHDV17].

Dataset	LIME (s)	Gradients (s)	Dimension of x
Iris-Cancer	0.03	0.000019	34
Toy Colors	1.03	0.000013	75
Decoy MNIST	1.54	0.000045	784
20 Newsgroups	2.59	0.000520	5000

not just predictors that perform well on the test set. [RHDV17] uses input gradients as a form of explanation. Input gradients “characterize how a data point has to be moved to change its predicted label” [BSH⁺10]. This means, they can show how important certain aspects of data are to a particular prediction, in a similar (but less interpretable) fashion to feature attribution algorithms such as SHAP. This decreased interpretability is a trade-off to far greater computational efficiency, as shown in Table 5.3.1.

The accuracy of these input gradient explanations is evaluated with respect to human created annotations. These annotations are defined in an annotation matrix $A \in \{0, 1\}^{N \times D}$, where binary masks indicate whether dimension D should be irrelevant (1) or not (0) for the training instance N . These annotations are done at a per-instance (local) basis as global feature importance can be misleading. In situations where input features are less meaningful to users, such as word vector representation, A can be difficult to create as input gradients do not have explanation flexibility (discussed in Section 3.2).

The loss function shown in Equation 5.3 is the mechanism by which explanation accuracy is included in the optimisation process. The right reasons section contains a regularisation parameter λ_1 which aims to ensure right answers and right reasons have similar magnitudes, and so are similarly valued. As such, the optimisation process sees mappings $f(x)$ that provide gradients that disagree with A as less optimal. Thus, models trained with such a loss function are biased towards predictors that are right for the right reasons.

$$\begin{aligned}
 L(\theta, X, y, A) = & \underbrace{\sum_{n=1}^N \sum_{k=1}^K -y_{nk} \log(\hat{y}_{nk})}_{\text{Right Answers}} \\
 & + \lambda_1 \underbrace{\sum_{n=1}^N \sum_{d=1}^D \left(A_{nd} \frac{\partial}{\partial x_{nd}} \sum_{k=1}^K \log(\hat{y}_{nk}) \right)^2}_{\text{Right Reasons}} + \lambda_2 \underbrace{\sum_i \theta_i^2}_{\text{Regular}}
 \end{aligned} \tag{5.3}$$

Annotations are not needed for each training instance, as in the case that $A_n = 0, \forall d$ the explanation reasoning has no effect on the loss function. This means the annotations encourage small gradients where $A_{nd} = 1$, discouraging mappings that disagree with A . It is accepted A will be imprecise both by ambiguity of what the right answer is, and the variation across accepted reasoning by the humans who create A . Because of this,

the loss function makes predictors f that disagree with A less favourable rather than removing them from the model class entirely.

Although not discussed in the work itself, such a method may reduce the under-specification of a pipeline by reducing the variation of reasoning amongst produced predictors.

By iteratively adapting A , the loss function can encourage the creation of predictions with different reasoning. This means that in situations where A cannot be created easily, an expert can then investigate the pool of different predictors created and decide which has the best reasoning.

5.4 The Rashomon Effect

Similar to underspecification, the study of the Rashomon effect looks at different predictors that perform similarly well. The term ‘‘Rashomon effect’’ is based on a 1950 film of the same name in which four characters describe their different perspective of the same crime, leaving the viewer to wonder which is true. The Rashomon effect is when multiple different explanations exist for the same situation [SRP19], as in the film, we cannot tell which explanations (if any) are true. In a machine learning sense, the Rashomon effect applies to situations in which many models exist with near optimal test performance [Bre01].

To quantify the Rashomon effect, [FRD19] considers the set of near optimally performing predictors a given pipeline can create, which they call Rashomon sets. As discussed in Section 1.1, a pipeline \mathcal{P} specifies the model class \mathcal{F} , which is the set of all possible predictors produced by \mathcal{P} . They define a slight alteration of the Rashomon set, an ϵ -Rashomon set $\mathcal{R}(\epsilon)$ both for populations and samples. However, the concept is the same for both. An ϵ -Rashomon set $\mathcal{R}(\epsilon)$ is defined such that, $\mathcal{R}(\epsilon) = \{f_1, \dots, f_n\} \subseteq \mathcal{F}, n \in \mathbb{R}$, where each predictor $f_i \in \mathcal{R}(\epsilon)$ has at worst the accuracy of ϵ below a given reference predictor f_{ref} . To quantify the size of a Rashomon set, [FRD19] studies the range of variable importance within the set. They use their own metric ‘‘model reliance’’ (MR) to represent the variable importance of a given predictor, and ‘‘model class reliance’’ (MCR) to represent the range of predictor variable importance within a given model class. Figure 5.4.1 illustrates a Rashomon set, plotting accuracy against reliance on some feature X_1 , bounded between the range of model class reliance $MCR_-(\epsilon) \leq MR(f_i) \leq MCR_+(\epsilon), \forall f_i \in \mathcal{R}(\epsilon)$. It should be noted that the computational process for MCR is specific for different model classes and loss functions, as such this method is not model-agnostic.

[SRP19] quantifies the Rashomon effect by studying the Rashomon set, specifically via a metric they call Rashomon volume, from which they define further metrics. [SRP19] defines the empirical Rashomon set similarly to Fisher, with the key difference being, rather than using a reference predictor they use an empirical risk minimiser. Empirical risk minimisers work under the knowledge that using only a sample of data (a dataset) means we cannot calculate how well a predictor works in practice (its true

Illustrations of Rashomon Sets & Model Class Reliance

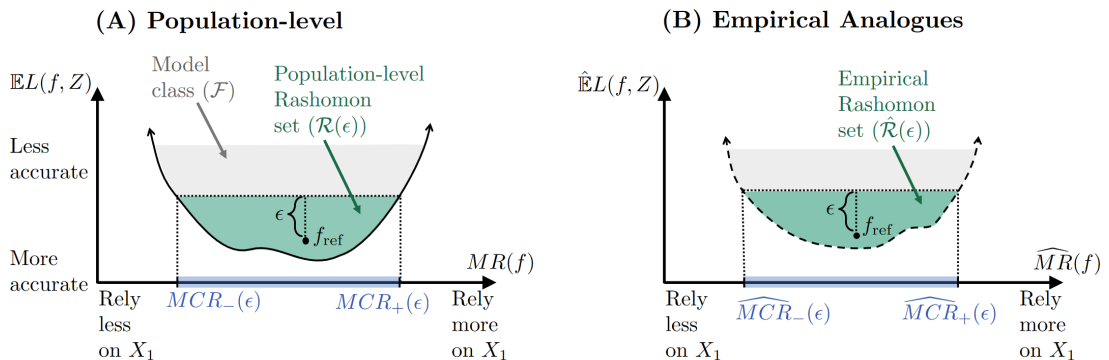


Figure 5.4.1: The empirical and population-level Rashomon sets ($\mathcal{R}(\epsilon)$) of model class (\mathcal{F}) [FRD19].

risk), but we can measure their performance on our training data (its empirical risk) [SSBD14].

The Rashomon volume is a measure of the volume of the Rashomon set in state space, defined specifically to a learning problem and state space. Using the same volume measure, the volume of the state space can be calculated. The Rashomon ratio is the ratio of the volume of predictors inside the Rashomon set (Rashomon volume) to the volume of predictors in the state space (state volume). The value of the Rashomon ratio is thus bounded between 0 and 1. As the volume of a Rashomon set relies on the Rashomon parameter ϵ , the interpretation of Rashomon ratios also relies on this chosen parameter. When ϵ is large enough, the Rashomon set $\mathcal{R}(\epsilon)$ contains all models in the model class \mathcal{F} . When ϵ is small, a larger ratio means that more predictors perform near optimally. Additionally, [SRP19] defines pattern Rashomon ratio, which considers the volume of predictions rather than predictors.

5.5 Prediction Variance

As with underspecification, predictive multiplicity examines prediction problems where multiple well performing predictors exist. As shown in figure 5.5.1, predictive multiplicity differs from the Rashomon effect in that similarly performing predictors must provide conflicting results. In line with our work, [MCU20] aims to quantify the extent to which predictive multiplicity affects a given problem, in the same way test performance is treated, from which researchers can make more informed design choices.

In order to achieve this, [MCU20] introduces two formal measures of predictive multiplicity, ambiguity and discrepancy. Ambiguity reflects the number of conflicting results that predictors can produce, whilst discrepancy reflects the number of predictions that change if a different well performing model $f \in \mathcal{F}$ was chosen.

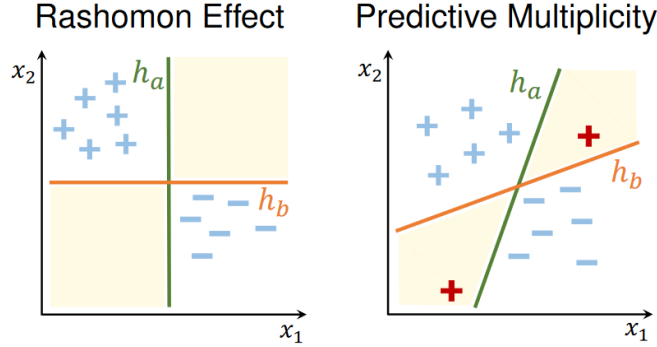


Figure 5.5.1: Examples of the Rashomon Effect and Predictive Multiplicity [MCU20]. Where two competing classifiers h_a and h_b classify two classes of $+$ and $-$. The points highlighted in red are those in conflict between the two classifiers.

[MCU20] produces an empirical Rashomon set (which they reference as an ϵ -level set) $\mathcal{R}_\epsilon(f_0) = \{f_1, \dots, f_k\}$ such that each predictor $f \in \mathcal{R}_\epsilon(f_0)$ performs equivalently or better than a given reference predictor f_0 add a chosen error tolerance ϵ .

Ambiguity is defined as the proportion of training instances that any predictor $f \in \mathcal{R}_\epsilon(f_0)$ differs in prediction than the reference predictor f_0 .

$$\alpha_\epsilon(f_0) := \frac{1}{n} \sum_{i=1}^n \max_{f \in \mathcal{R}_\epsilon(f_0)} \mathbb{1}[f(x_i) \neq f_0(x_i)] \quad (5.4)$$

Discrepancy is defined as the maximum proportion of different predictions between some predictor $f \in \mathcal{R}_\epsilon(f_0)$ and the reference predictor f_0 .

$$\delta_\epsilon(f_0) := \max_{f \in \mathcal{R}_\epsilon(f_0)} \frac{1}{n} \sum_{i=1}^n \mathbb{1}[f(x_i) \neq f_0(x_i)] \quad (5.5)$$

Where x_i is a training instance such that the training dataset $x_{trn} = \{x_1, \dots, x_n\}$.

In order to quantify variance between possible predictors for a given problem, prediction deviation can be used. Prediction deviation is a metric of uncertainty for predictors produced for a given problem [LLRB16]. To accomplish this, J predictors are created from which the empirical Rashomon set \mathcal{R}_{f^*} is defined as all predictors within the 95% confidence interval from the best performing predictor f^* on the available dataset D . Within the Rashomon set \mathcal{R}_{f^*} , the two predictors $\{f_1, f_2\} \subseteq \mathcal{R}_{f^*}$ with the maximum difference between predictions are selected. This difference (or deviation) between predictors f_1 and f_2 is the prediction deviation of the problem. If the prediction deviation of a problem is high, it means predictors chosen from \mathcal{F} can provide vastly different prediction results whilst still fitting the data well [LLRB16].

[MAD19] aims to identify extrapolation by examining the variance of predictors across a local ensemble. In this work, a prediction is underdetermined if there are many predictions that are equally evaluated. This can mean predictors base their predictions on arbitrary choices, as with underspecification.

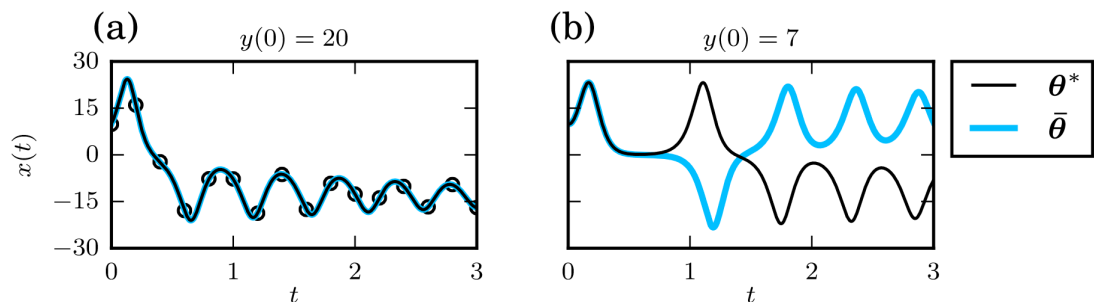


Figure 5.5.2: a) Circles show the data points of a given dataset, the best-fit predictor is shown in black and an alternative is shown in blue. b) Shows the same two models, using a different dataset drawn from the same distribution as in (a), performing far more differently. [LLRB16]

Given a predictor, they provide an extrapolation score which quantifies the extent to which a prediction for a particular input is underdetermined. This score aims to model the possible variance of predictions across a local ensemble, which is a set of similarly performing predictors to the given predictor. They prove that the extrapolation score correlates with the standard deviation of predictions on multiple datasets, as shown in table 5.5.1. Instead of forming an actual local ensemble of similarly performing predictors to the given reference predictor, they search for eigenvectors that form a Rashomon set around it. As eigenvectors with small eigenvalues represent directions with little curvature, predictors formed on these eigenvectors should be similarly evaluated. This Rashomon set is referred to as a loss-preserving ensemble subspace and is constructed as the orthogonal complement to the top m eigenvectors. To form the extrapolation score they project the test input gradient to the ensemble subspace.

Table 5.5.1: Pearson correlation between prediction standard deviation and extrapolation scores across four UCI datasets [MAD19].

Dataset	Pearson
Boston	0.76
Diabetes	0.50
Abalone	0.76
Wine	0.87

We were not aware of this method until after our experimentation for this work had completed, when in December of 2021 this paper was updated to reference underspecification rather than underdetermination. Despite the similarities between underdetermined and underspecified predictions; being published prior to the underspecification paper we discuss [DHM⁺20]; and sharing an author, there is no mention of it within the underspecification paper. This could be because of the pipeline focus of [DHM⁺20] is in contrast with the individual predictor focus of this work.

We mention it here for the sake of completeness and discuss it in its original form

5. *Related Work*

as that was its state at the time of publication.

Part II

Contributions

Chapter 6

Measuring Underspecification with Underspecification Index

Contents

6.1	Quantifying Underspecification	53
6.2	Method Definition	57

For a given machine learning problem, multiple predictors (trained models) may perform similarly on a testing set, whilst exhibiting different generalisation behaviour. Machine learning pipelines that can produce multiple different predictors that are similarly evaluated, exhibit *underspecification*. In this chapter we present our method for quantifying underspecification, using feature attribution methods from explainable machine learning. We first outline our description of underspecification, building from section 1.1. Presenting theoretical examples, we aim to illustrate how underspecification can affect explanations, and thus how by studying these effects one can identify underspecification. We conceptually present a method for such analysis, by generating a collection of similarly performing predictors from a given pipeline and measuring variation between explanations produced by these predictors.

6.1 Quantifying Underspecification

We consider underspecification in a supervised learning setting. A given ML pipeline \mathcal{P} produces a predictor $f \in \mathcal{F}$ where \mathcal{F} is the model class. The model class \mathcal{F} is the set of all possible different predictors produced by the pipeline \mathcal{P} . \mathcal{P} uses the dataset $\mathcal{D} \subset \mathcal{C}$ drawn from the distribution \mathcal{C} which is split into the i.i.d datasets, \mathcal{D}_{trn} the training dataset, and \mathcal{D}_{tst} the testing dataset. Regardless of the manner in which \mathcal{P} specifies \mathcal{F} , f is normally chosen from \mathcal{F} by optimising an evaluation metric reflecting the performance of f at predicting the held-out dataset \mathcal{D}_{tst} . An ML pipeline is underspecified if it can return multiple different predictors such that they give similar test evaluations, while

encoding substantially different inductive biases. Different inductive biases can result in different generalisation behaviours on further datasets drawn from the same distribution $\hat{\mathcal{D}} \subset \mathcal{C}$. This variation of generalisation behaviour poses issues for the credibility of test performance evaluations on underspecified pipelines. We see that predictors sometimes exhibit unexpectedly poor performance when used in real-world applications when such multi-predictor phenomenon occurs.

The first step of addressing underspecification is to identify it. As introduced in Section 5.1, stress tests measuring prediction performances have been reported in the literature [DHM⁺20]. Stress tests themselves need to be designed specifically for a given problem, and analysed against expectations. However, general i.i.d test performance evaluations often cannot fully identify underspecification, and as such the problem of underspecification challenges the credibility of such evaluations.

In this work, we present an alternative approach: identifying underspecification with explanations. In a nutshell, given a training dataset D_{trn} and a testing dataset D_{tst} we construct a set of well-performing predictors (similar to Rashomon sets discussed in Section 5.4) and study SHAP explanations generated from these predictors. We identify underspecification when observing “too many” different explanations from such predictors on the dataset. We observe: **if a dataset can be explained in multiple ways, then an ML pipeline built from it is likely underspecified.**

Since predictors can contain a vast amount of parameters and when generated with different model architectures have different internal structures, it is not straightforward to directly compare two predictors and determine how similar they are. Thus, to determine whether an ML pipeline is underspecified, we study explanations obtained from predictors produced by the ML pipeline, and use those as a proxy to estimate the differences between predictors. As discussed in section 3.1, explanations aim to model a given prediction, meaning we compare the explanation models rather than the predictors themselves.

To compare variation between a set of models, we could also study the variance of predictions as in the literature presented in section 5.5. However, particularly for binary classification, predictors can give the same output for some input, whilst having different reasoning. We show this theoretically in examples such as 6.1.1 and experimentally in chapter 7.

Our core assumption is that:

If two predictors give similar explanations to a prediction, then they encode the similar inductive biases; hence they should be considered similar.

Example 6.1.1 Our core idea can be illustrated with the following example, extended from example 1.1.1. Consider the binary classification problem of predicting whether binary strings belong the class POS or NEG. Let D_1 and D_2 be balanced datasets containing eight and twelve 5-bit binary strings respectively, where each bit is considered a feature.

Table 6.1.1: Two simple string datasets, D_1 , and D_2 for underspecification illustration.

	Data	POS Explanation
D_1	POS: 01101, 11101, 11111, 01111 NEG: 00000, 00010, 10010, 10000	$\cdot 1 \dots, \dots 1 \dots, \dots \dots 1$
D_2	POS: 01101, 11101, 11111, 01111, 01001, 11100 NEG: 00000, 00010, 10010, 10000, 00001, 10111	$\cdot 1 \dots$

As shown in Table 6.1.1, D_2 contains all strings of D_1 and four additional strings. It can be assumed D_2 and D_1 are both drawn from the same distribution \mathcal{C} . For the purpose of maintaining a simple example, we consider only predictors that classify each example based on a single bit. With these restrictions in mind, we can form three predictors that perfectly classifies D_1 . Let x be a binary string made of 5 bits $x = [x_1, x_2, x_3, x_4, x_5]$.

$$Predictor_1(x) = \begin{cases} \text{POS}, & \text{if } x_2 = 1 \\ \text{NEG}, & \text{otherwise} \end{cases}$$

$$Predictor_2(x) = \begin{cases} \text{POS}, & \text{if } x_3 = 1 \\ \text{NEG}, & \text{otherwise} \end{cases}$$

$$Predictor_3(x) = \begin{cases} \text{POS}, & \text{if } x_5 = 1 \\ \text{NEG}, & \text{otherwise} \end{cases}$$

As we are comparing explanations instead of predictors, we represent these three predictors as the following '1-bit explanations':

- $\cdot 1 \dots$: a string is POS because its second bit is 1,
- $\dots 1 \dots$: a string is POS because its third bit is 1, and
- $\dots \dots 1$: a string is POS because its fifth bit is 1.

Each of these explanations can belong to a predictor that has optimal performance on D_1 , and so there is no reason to prefer any. However, with the four additional strings introduced in D_2 , the latter two explanations no longer achieve optimal performance. This is because the two final NEG strings in D_2 , 00001 and 10111, would be incorrectly classified as POS by the second and third explanation respectively. There remains a single 1-bit explanation achieving optimal performance on D_2 :

- $\cdot 1 \dots$: a string is POS because its second bit is 1,

We observe that D_2 with more data yields fewer 1-bit explanations than D_1 and so specifies a more consistent model class. We say the model class is more consistent as fewer explanations that perfect classify these datasets, means explanations

are more consistently evaluated, that is, fewer explanations that are equivalently evaluated whilst performing differently on further data from the same distribution. Thus, pipelines built from D_2 are less likely to be underspecified than those built using D_1 .

We choose to use SHAP to produce our explanation models over other post-hoc interpretability methods for a number of reasons. Firstly, SHAP has been shown to be more consistent with human produced explanations than other methods such as LIME and DeepLIFT [LL17]. Additionally SHAP offers globally consistent explanations while methods such as LIME do not [Rat19, Mol19]. As SHAP fairly distributes the prediction amongst input features, even if the contribution of each feature is the same amongst two explanations, their explanations will be different if they have a different prediction output. This allows us to model not only reasoning with SHAP explanations, but also prediction variance. Compared to LIME, SHAP has been shown to produce a higher explanation accuracy as well as correlating to classification performance better [YF21]. Initial investigations have also suggested SHAP values are simpler to cluster than LIME [GG21]. As we aim to quantify differences within sets of explanations, having a more distinguishable explanation space is advantageous.

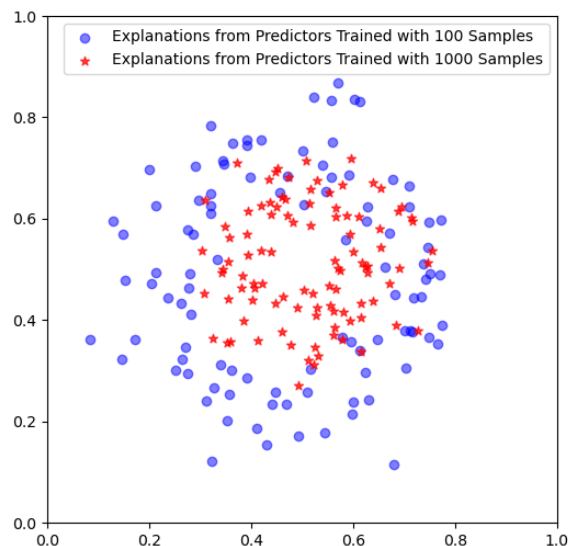


Figure 6.1.1: An illustration of explanations from predictors trained with different sample sizes¹. Predictors trained with more data - hence less underspecified ML pipelines - produce more agreeable explanations. Red stars are placed closer to each other than blue dots are. This work was completed by Dr. Fan Xiuyi, as a part of our published work [HFL⁺21].

6.2 Method Definition

Consider an ML pipeline \mathcal{P} with training dataset D_{trn} and testing dataset D_{tst} . We iteratively train predictors using D_{trn} to produce a set of predictors $\mathcal{M} = \{f_1, \dots, f_K\}$ that perform similarly on D_{tst} . Note that the number of predictors in \mathcal{M} , K , is a given parameter where we repeatedly train until K predictors that exceed the performance threshold θ have been produced. \mathcal{M} can be interpreted as an empirical Rashomon set (discussed in Section 5.4), where each predictor has performance better than θ , so could equivalently be expressed as $\mathcal{R}(\theta) = \{f_1, \dots, f_K\}$. For each predictor $f \in \mathcal{R}(\theta)$ we use the SHAP explainer Π to calculate the local explanations $\Pi(f, \mathbf{x})$ for each data instance $\mathbf{x} \in D_{tst}$.

For each test instance $\mathbf{x} \in D_{tst}$ we compute a local underspecification index $\mathcal{U}_{\mathbf{x}}$ which can be used for local analysis. This local index $\mathcal{U}_{\mathbf{x}}$ provides a measure of agreement between all predictor $f \in \mathcal{R}(\theta)$ explanations for a test instance \mathbf{x} . We experiment with a number of metrics (discussed in chapter 4) to compare how similar the explanations for a data instance \mathbf{x} are for all predictors $f \in \mathcal{R}(\theta)$. For the sake of simplicity we use $\lambda(x, y)$ to represent a metric between x and y , where λ is Euclidean distance, Pearson correlation, Cosine similarity or Kendall Rank Correlation. To compute our local underspecification index $\mathcal{U}_{\mathbf{x}}$, we compare each predictor $f \in \mathcal{R}(\theta)$ to each other predictor using some metric λ , such that:

$$\mathcal{U}_{\mathbf{x}} = \frac{2}{k(k-1)} \sum_{i=1}^k \sum_{j>i}^k \lambda(\Pi(f_i, \mathbf{x}), \Pi(f_j, \mathbf{x})) \quad (6.1)$$

where $\Pi(f_i, \mathbf{x})$ is the SHAP explanation of predictor f_i for test instance \mathbf{x} .

To compute our set underspecification index \mathcal{U} we simply take the mean of all the local underspecification indices calculated, such that:

$$\mathcal{U} = \frac{1}{n} \sum_{\mathbf{x}=1}^n \mathcal{U}_{\mathbf{x}} \quad (6.2)$$

where $n = |D_{tst}|$.

The component interactions of our method can be seen in Figure 6.2.1. In the blue box, we represent the given ML pipeline \mathcal{P} , that produces predictor f . Using the same training D_{trn} and testing datasets D_{tst} we repeat the training loop of \mathcal{P} , k times to produce the set of similarly-well performing predictors (the Rashomon set $\mathcal{R}(\theta)$) shown in green. For each model f in this set $\mathcal{R}(\theta)$ we calculate its explanations for

¹Blue dots and red stars represent explanations obtained from predictors trained with 100 and 1000 randomly selected samples in the COVID-19 dataset (introduced in chapter 9) respectively. Within each set, the coordinates \mathbf{x}_i are computed with a stochastic hill climbing algorithm that solves $\arg \min_{\mathbf{x}_i, \mathbf{x}_j} \sum |L_2(\mathbf{x}_i, \mathbf{x}_j) - D_\tau(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)|$, where L_2 is the L2 norm (the Euclidean distance to the origin), D_τ is the Kendall distance of each pair of explanations $(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)$.

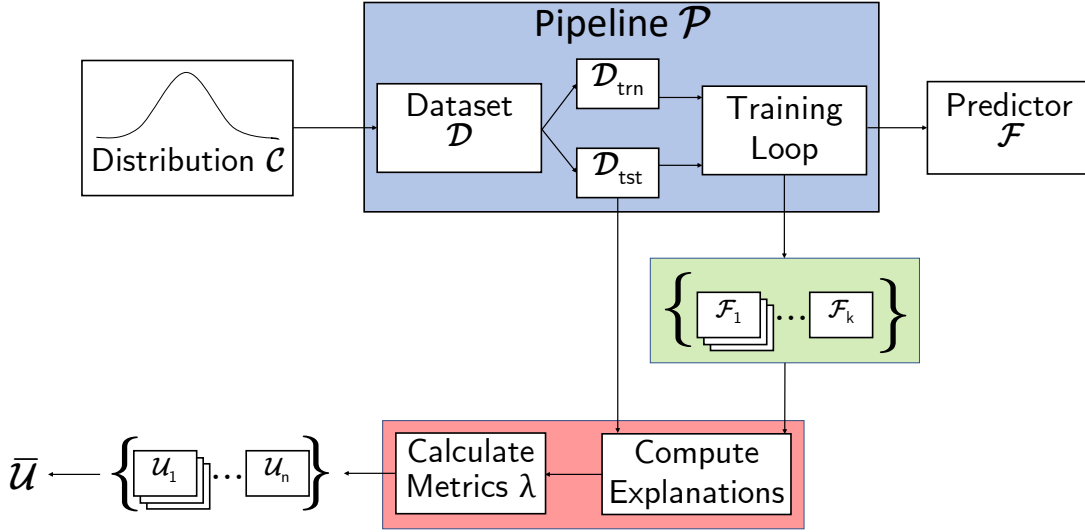


Figure 6.2.1: Simplistic component diagram of our method.

each instance $\mathbf{x} \in D_{tst}$. We use the chosen metrics λ to calculate how similar these computed explanations are to provide a local underspecification index for each data instance in the testing set. Finally, we take the mean of all local indexes to provide our set underspecification index \bar{u} .

Algorithm 1 GenExplanations($D_{trn}, D_{tst}, K, \theta$) **return** **E**

Input: The number of models K , Training Dataset D_{trn} , Testing Dataset D_{tst} , Prediction Performance Threshold θ

Output: **E** - Local explanations for each data instance $\mathbf{x} \in D_{tst}$ for K predictors

- 1: **E** = []
 - 2: **while** $|\mathbf{E}| < K$ **do**
 - 3: Train a predictor f with D_{trn}
 - 4: **if** the performance of f on $D_{tst} > \theta$ **then**
 - 5: **L** = []
 - 6: **for** each $\mathbf{x} \in D_{tst}$ **do**
 - 7: Append $\Pi(f, \mathbf{x})$ to **L**
 - 8: Append the local explanations **L** of f to **E**
 - 9: **return** **E**
-

Algorithm 1 shows how we form the explanation matrix \mathbf{E} that we study in order to quantify underspecification. First we must generate a set of well performing predictors. To accomplish this we iteratively train predictors, and only study them if their performance is better than a chosen threshold θ .

Once we have produced a predictor of sufficient performance, we compute its SHAP explanation for each data point $\mathbf{x} \in D_{tst}$, which we form into a list \mathbf{L} , where each element explains the predictor output for a matched data instance, such that $|\mathbf{L}| = |D_{tst}|$. Finally, we combine each predictor’s explanations \mathbf{L} to \mathbf{E} , such that $|\mathbf{E}| = K$.

Algorithm 2 CalcLocalIndex(x, \mathbf{E}, K) return $\mathcal{U}_{\mathbf{x}}$

Input: Index of test instance \mathbf{x} , The matrix of explanations \mathbf{E} , The number of models K ,

Output: Local underspecification index $\mathcal{U}_{\mathbf{x}}$ of data instance \mathbf{x}

```

1:  $\mathcal{U}_{\mathbf{x}} = 0$ 
2: for  $i \leftarrow 1$  to  $K$  do
3:   for  $j \leftarrow i$  to  $K$  do
4:      $\mathcal{U}_{\mathbf{x}} += \frac{2}{K(K-1)}\lambda(E_{i,x}, E_{j,x})$ 
5: return  $\mathcal{U}_{\mathbf{x}}$ 

```

Algorithm 2 depicts how we calculate the local underspecification index $\mathcal{U}_{\mathbf{x}}$ of a given test index \mathbf{x} . As stated previously, this is calculated by pairwise computing λ between each predictor to each other predictor. Each metric we calculate is transitive in nature, such that $\lambda(a, b) = \lambda(b, a)$. This means that we can save on computation by calculating the distance between only half the matrix, as a triangular matrix. As we are calculating only the inner triangular matrix values, we update the coefficient for the average, as to make the underspecification index of the same magnitude as λ . To calculate the set underspecification index $\bar{\mathcal{U}}$, we simply take the mean of all local indexes $\mathcal{U}_{\mathbf{x}}$ for each $\mathbf{x} \in D_{tst}$ for the set D_{tst} .

Overall, the proposed approach to identifying underspecification with explanations has the following advantages:

1. It is model-agnostic and applicable to any data types and ML models as long as such a model can be analysed with a model-agnostic explainer.
2. It is self-contained and does not require any additional information such as domain knowledge or human expert inputs.
3. It is simple and does not require any special treatment to the dataset, e.g., stratification or alteration, to estimate underspecification.

Over the next two chapters, we aim to investigate the following hypothesis by experimenting with our proposed approach:

Hypothesis 1 *As more examples are added to a dataset, the performance and explanation agreement should increase.*

Chapter 7

Implementing Underspecification Index for Classification

Contents

7.1	Language and Library Considerations	61
7.2	Software Realisation	63
7.3	Experimental Setup	66
7.4	Experiment Results	69

In this section we cover our implementation of our method detailed in Chapter 6. We first present our aims of this implementation and from this, justifications for our technology choices to realise our proposed algorithms. Afterwards, we detail our initial implementation and our successive optimisation, relating both back to our aims. In order to evaluate our implementation we design tests based on stratified performance evaluations and experiment these tests on multiple datasets from the literature. Finally, we discuss the results of these experiments and interpret them in order to evaluate our implementation.

7.1 Language and Library Considerations

Our method detailed in Chapter 6 could be implemented using a range of different technologies. In this section we detail the language and library choices we have made and present our reasoning for these decisions. Firstly, we will discuss our desirable aims for this implementation which we then use to justify our choices against. This work is aimed as an investigation of quantifying and identifying underspecification using feature attribution explanations. As such, the primary aims of our implementation are to verify the suitability of explanation analysis for underspecification quantification and to provide repeatable and easy to follow experiments.

We chose to implement our method using Python, for a number of reasons we discuss below.

Python has a large selection of relevant libraries for this project; by making use of such packages we can cut down on the time spent developing the implementation itself and focus on our experimentation. Of particular importance to us, the researchers who presented SHAP, developed a number of well optimised implementations already present in the literature [LL17, LEC⁺20, LNV⁺18]. Additionally, by using widely used and trusted packages it reduces the amount of code which reviewers need to evaluate for our project.

The second major reason we chose to implement using Python is readability both within our research group and throughout the wider machine learning community. Although this work itself is individually produced, there are collaborative aspects, such as in our published work [HFL⁺21], and our COVID-19 case study. In both of these aspects, work had been completed in Python before we started this project and to keep continuity we began this project in Python. For the wider scope outside our research group, Python is one of the most used programming languages in the world [Jet21, Sta21] and some of its most common uses are in data analysis and machine learning [Jet21]. Papers With Code, a website that organises published papers along with code that creates that papers findings, reported that in 2021 almost 60% of uploaded repositories used the Python machine learning package PyTorch [Pap22]. Furthermore, the usage of Python in data science has increased year-on-year as opposed to statistically focused languages such as *R* which has shown signs of decline [Gre17], or at least slower growth than Python [TIO22].

An often reported weakness of Python is its poor performance compared to efficiently optimised languages such as C++. Firstly, this is not a large concern in this project as we are not aiming to provide performant production code, but easily understandable repeatable experiments. Even with this considered, Python can be sped up by orders of magnitude using Just-In-Time (JIT) compilers such as Numba [DGBM] which make its performance comparable to languages such as C++. Furthermore, many of the machine learning and data science packages we use, such as SciPy, NumPy and SHAP use optimised code built on languages other than Python. With these packages being used for many of the computationally intensive tasks within our implementation, they dominate the total computation time more so than the less efficient code written in Python.

Following our decision to use Python as the language of our implementation, we must also decide which libraries we wish to use. The first and most obvious package decision is SHAP [LL17], we explain our reasoning for using SHAP in Section 6.1. For our experiments we use random forest models so that we can take advantage of the fast implementation of tree SHAP [LEC⁺20]. Tree SHAP supports a number of different random forest implementations, of which we decide to use SKLearn. One reason we decide to use SKLearn is its comparative ease of installation compared to XGBoost particularly on windows. Another is that we can use other components from SKLearn throughout our implementation, cutting down on the number of dependencies the implementation has. We can use SciPy and SKLearn (which is built on top of SciPy) to calculate all of our metrics detailed in Chapter 4. We use NumPy and Pandas to store and organise data throughout our implementation, as they are well-supported

by all other libraries we use, leverage performance benefits, and again simplify our codebase. Likewise, we use Numba to optimise our Python code, and drastically reduce run time. As previously discussed, Numba translates Python to optimised machine code and can easily be used to parallelise loops. Numba however, only supports a subset of Python and NumPy types, and enforces stricter standards than Python generally does.

7.2 Software Realisation

In this section we aim to implement our method to calculate a set underspecification index \bar{U} as specified in Section 6.2. We first show an initial Naïve implementation, which we compare against our optimised code we use for experimentation, a comparison between the runtime of our naïve and optimised implementations can be seen in Table 7.2.1. All of our code can be found at:

<https://github.com/JamesHinns/Underspecification-Index>

Specifically to support this section we created the `Software_Realisation.ipynb` notebook, which contains the code used for all figures within this section. Figures that make use of Numba within this notebook take time to compile, and as such these results are taken off of the second run. Numba can cache the compiled object code which saves compilation on the first time it is executed, we make use of this in our working implementation, which is found with the `underspecification_index` folder of the directory. To calculate the runtimes of the figures in this section we use the `default_timer` from the Python package `timeit`. The runtimes displayed within this chapter and the next are not deterministic due to the random nature of predictor training, as such they are only used for approximate comparisons. The experiments that produce these runtimes are calculated on an ubuntu machine with a 64-core AMD 3990x CPU, 250gb ram, and an NVIDIA GeForce RTX 3080 GPU.

Task	Naive Run Time (s)	Optimised Run Time (s)
Explanation Matrix	33.97	21.40
Local Index	2.44	0.30
Set Index	244.00	2.05

Table 7.2.1: Run time (in seconds) of naïve and optimised implementations of the tasks to produce Underspecification index \bar{U} for the Abalone dataset. 80% of the dataset randomly sampled as the training dataset, Rashomon performance threshold $\theta = 0.7$ with 80% training dataset, $\theta = 0.7$, and a fixed testing size of 100 randomly sampled instances from the held-out 20% of data. The code and results can be found in the `Software_Realisation.ipynb` example notebook.

We begin with our implementation of algorithm 1, which generates a 3-d matrix of explanations. This matrix, \mathbf{E} , contains a local explanation $L_{f,x} = \Pi(f, \mathbf{x}) = \{a_0, \dots, a_{k-1}\}$ for each data instance in a testing dataset D_{tst} for each predictor $f \in R(\theta)$.

To accomplish this, our naïve implementation takes an OOP view, splitting this into two functions, one to generate a Rashomon set $R(\theta)$, and another to generate the E from this Rashomon set. As stated previously we use SKLearn’s random forest classifier as our model for each predictor in the Rashomon set $f \in R(\theta)$, in our given examples we use ten trees per forest. The function `gen_rashomon_naive` takes five parameters, four representing the dataset split, and the accuracy threshold for the Rashomon set θ . Throughout this work we split our datasets D into training D_{trn} and D_{tst} , we then split each of these into their respective training features x and prediction target y resulting in the four arrays x_{trn} , y_{trn} , x_{tst} , y_{tst} . SKLearn’s model handles the training for us, only requiring x_{trn} , y_{trn} to produce our predictor f . We then check if f belongs to the Rashomon set $R(\theta)$ by having f predict y_{tst} from x_{tst} . We check the prediction performance again using SKLearn’s accuracy metrics and if this at least matches θ , f is added to $R(\theta)$. Furthermore, we repeat this process until the number of predictors in $R(\theta)$ matches our chosen predictor count parameter, which is fixed at 100 throughout our experiments.

We pass the test features x_{tst} and produced Rashomon set $R(\theta)$ to `gen_exp_matrix_naive` in order to produce \mathbf{E} . For each $f \in R(\theta)$ we generate the local SHAP explanation $\Pi(f_j, \mathbf{x}_i)$ to create the list of all predictors explanations for a data instance x_i . We repeat this for each data instance $x_i \in x_{tst}$ to create E .

Unfortunately, neither of these functions can make use of Numba for optimisation because the data type of SKLearn’s random forest is not supported. For this reason, we leverage the faster optimisation of tree SHAP, rather than looping through each data instance and computing its explanation, we pass the whole testing dataset x_{tst} which returns an array of local explanations for a given predictor. We repeat for each predictor $f \in R(\theta)$ setting each predictor f_i ’s local explanations to the i th index of an array. We transpose this array, swapping the first and second dimensions to allow us to pass a contiguous array for calculating each underspecification index. Furthermore, we merge `gen_rashomon_naive` and `gen_exp_matrix_naive`, meaning we don’t save the Rashomon set, simply calculate a predictors explanation only if it belongs to the Rashomon set, that is, at least matches the performance on x_{tst} of θ . As shown in Table 7.2.1, these improvements yield a noticeable performance increase.

Algorithm 2 aims to calculate a local index \mathcal{U}_x for a data instance x given a set of explanations for that instance \mathbf{E}_x and the metrics \mathcal{U}_x should be calculated with. To naïvely implement this, we pairwise compare each predictor to each other predictor via a nested loop, however to save on computation, we calculate this as a triangular matrix. This calculation is equivalent to the full matrix calculation due to the transitive nature of our metrics, $\lambda(f_1, f_2) = \lambda(f_2, f_1)$. For each pair of predictors we compare, we calculate each requested metric. In our Cosine, Pearson and Kendall Tau are all calculated using SciPy’s metrics, Euclidean distance is calculated using the L-2 norm implementation in NumPy. To save us from creating an array only to take the mean, we multiply each metric result by a coefficient $(\frac{2}{n(n-1)})$ instead of $(\frac{1}{n})$ to mitigate overflow errors in this regard. Although not an issue with our experiment datasets, this could be an issue for extremely large datasets, particularly when metrics can provide large results, such as Euclidean distance.

To optimise this, we employ Numba with the decorator `njit`, which allows no Python code to be executed, only the optimised machine code. We also use the `fastmath` tag, which if used with Intel SVML can provide significant performance benefits, and is negligible if not installed. However, SciPy's metrics aren't able to be used with Numba. For this reason, we implement these three metrics using NumPy. Cosine is simply calculated as shown in equation 4.7 as the dot product of both inputs over the product of the L-2 norm of each input. Both the dot product and L-2 norms are calculated using inbuilt NumPy functions. We make use of the NumPy Pearson correlation implementation which produces a correlation matrix, from which we can select the coefficient. Consistent with our other results, we find that these NumPy implementations outperform SciPy equivalents.

Metric	SciPy Run Time (s)	Optimised Run Time (s)
Pearson	0.628	0.156
Cosine	0.351	0.156

Table 7.2.2: Runtime (in seconds) of Numba optimised NumPy implementations vs SciPy implementation for input lengths of 1000, with 10,000 iterations

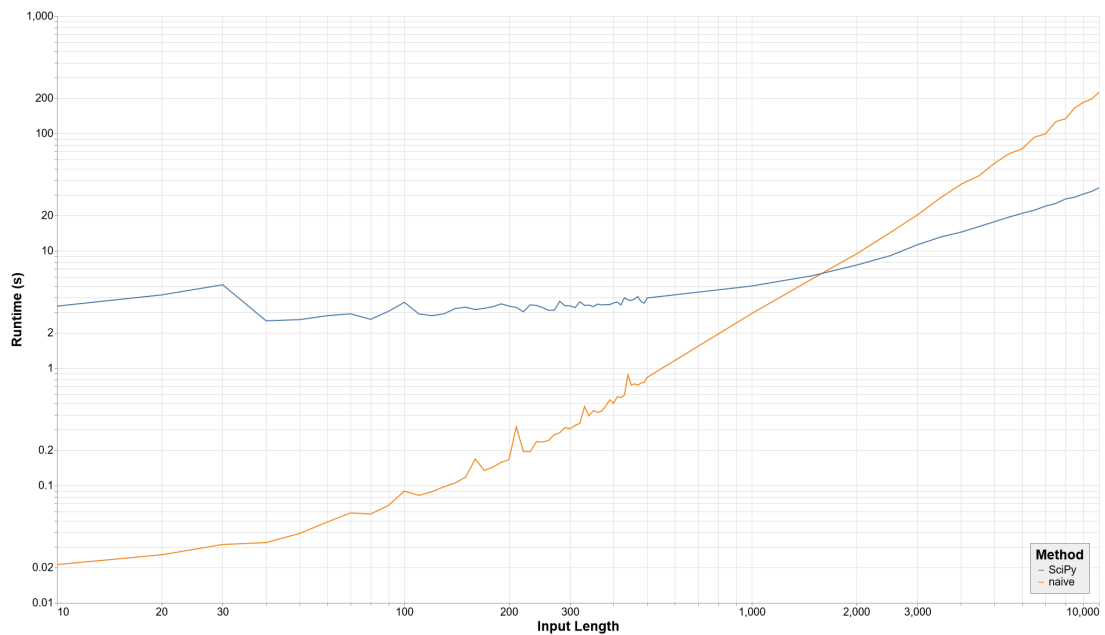


Figure 7.2.1: Implementation of naïve $\mathcal{O}(n^2)$ Numba optimised implementation vs SciPy $\mathcal{O}(n \log n)$ implementation for 10,000 iterations, plotted on a logarithmic scale.

Kendall Tau however is slightly more complex to calculate. There exists an $\mathcal{O}(n \log n)$ implementation of Kendall Tau, such as that used in SciPy. It is however not a straightforward implementation, basing part of its implementation on merge sort. To stay true to our objectives, providing relatively easy-to-follow implementations, we implement a

Table 7.2.3: The runtime (in seconds) of calculating the set index of the dataset used in Table 7.2.1 for a Python and SciPy implementation, a Numba njit optimised implementation, and Numba njit using parallelisation.

Method	Runtime (s)
Python	243.66
Njit	2.89
Parallel	2.05

$\mathcal{O}(n^2)$ algorithm. We do however, take leverage performance benefits where we can to still outperform the less complex algorithm in runtime, as shown in Figure 7.2.1. Firstly, we use NumPy’s argsort to produce our rankings, which does not produce tied rankings regardless of values. This means we can implement Tau_A as discussed in Section 4.5, saving us from calculating further components of the denominator. We also use Numba again with the decorator njit and tag fastmath. As shown in Figure 7.2.1 our ‘naïve’ $\mathcal{O}(n^2)$ implementation still outperforms SciPy’s $\mathcal{O}(n \log n)$ on all our tests of input lengths less than 1500. This could be an issue in complex domains with an extensive amount of features, however as all of our datasets have well below 100 features, our ‘naïve’ $\mathcal{O}(n^2)$ implementation has orders of magnitude less runtime than SciPy’s $\mathcal{O}(n \log n)$ implementation.

To calculate the set index for x_{tst} we compute the local index for each data instance in x_{tst} . We decide to return this array of local indices rather than the set index to allow for local analysis. The set index can simply be calculated using NumPys mean function for each metric. Alongside using the njit decorator here we use the parallel tag, which allows us to calculate local indices in parallel. We find the Numba optimised implementations are again orders of magnitude faster than our Python and SciPy based implementation, as shown in Table 7.2.3.

7.3 Experimental Setup

We experiment with our implementation on six datasets, summarised in Table 7.3.1. To investigate how underspecification changes with data set size we stratify each dataset into subsets of various lengths. We compute our set underspecification indices \mathcal{U}_x , average test performance, and prediction variance for each strata. We use the four metrics discussed in Chapter 4 to calculate \mathcal{U}_x and compare their results.

For each strata we train 100 random forest classifiers each with 10 trees. Choosing the number of predictors that the Rashomon set will contain is not straightforward. In choosing this parameter, we must handle the trade-off between the reliability of our set underspecification index and run-time. This is because a greater number of predictors allows us to model the true (population-level) Rashomon set more accurately, and thus place more credibility in our index which derived from this modelled Rashomon set. We choose a fixed number of models of 100 for all our experiments, justifying this decision experimentally in Section 7.4.

Table 7.3.1: Summary of datasets used for classification experiments, each available on the UCI ML repository [DG17].

Dataset	# of Samples	# of Features	Class Structure
Abalone [DG17]	4177	8	(2096,2081)
Adult [DG17]	48842	14	(37155,11687)
Bank [DG17, MCR14]	45211	16	(39922,5289)
Breast Cancer [DG17]	569	30	(212,357)
Mushroom [DG17]	8124	22	(4208,3916)
Wine Quality [DG17, CCA ⁺ 09]	1599	11	(744,855)

In most of our experiments, increasing the length of training data increases test performance. We are able to vary our performance threshold θ in accordance with the expected test performance. To enable this, we create a utility function to estimate how well our chosen model specification will perform on a given dataset. This utility function simply trains n predictors for each given strata and returns the average performance of each strata against a shared test set. We use these average performance accuracies as θ so that each predictor $f \in \mathcal{R}(\theta)$ matches or exceeds the average performance of the n predictors generated.

As in our explanation matrix implementation (algorithm 1) we do not save the predictors $f \in \mathcal{R}(\theta)$ we must make edits to conduct these experiments. To do so, we keep an array of the predictions and accuracy of each predictor we calculate explanations for ($\mathcal{R}(\theta)$). Along with returning the explanation matrix, the implementation also saves the variance of these predictions and the mean accuracy to a temporary file.

We also make a wrapper function to return a single csv for each dataset, recording the local indices, training strata, prediction accuracy and prediction variance. Given a set of training strata, a testing set, performance thresholds, and metrics to calculate this wrapper function, `compare_datasets` calculates local underspecification indices for each strata. For each strata this calls the functions to generate an explanation matrix, and then to calculate underspecification indices from this, combining these with the prediction variance and accuracy from the temporary file. This data is then saved as a csv which we can use to analyse the effect of underspecification against training length for each of our datasets.

This setup is similar to the stratified performance evaluations mentioned in Section 5.1. However, rather comparing test performance only, we additionally compare prediction variance and underspecification index.

An underspecified pipeline is one that can "return many predictors with equivalently strong held-out performance in the training domain" [DHM⁺20]. As we use SHAP explanations to model predictors, the greater variation between the explanations, the greater the variation between predictors. By quantifying the variation of these explanations we can quantify the extent of underspecification for a given pipeline.

We stratify each dataset to compare the relationship between dataset size, test accuracy, test variation and underspecification indices (explanation variance). We expect

to see an approximately exponential relationship between dataset size and test performance, plateauing towards the maximum achievable test performance [PKS15, SZ05, AAAB⁺21, BAL⁺19]. This relationship depends upon the complexity of the prediction problem, where more complex problems tend to take a greater number of samples to perform similarly [DP06]. Additionally, we expect that prediction variation will decrease as dataset size is increased [BW99], and that the variation of a predictors performance indicates its generalisability [PSM⁺07].

As prediction variation is directly related to how constrained a prediction problem is [LLRB16], and that this variance tends to decrease with dataset size, a problem is more constrained, and therefore has less underspecification as dataset length increases.

These stratified evaluations aim to support that our underspecification index is in agreement with these already identified relationships. By expanding on Hypothesis 1 (pg. 59) we propose the following Hypothesis for our classification experiments to represent this aim:

Hypothesis 2 *As more examples are added to a dataset, the classification accuracy and explanation agreement should increase. Specifically, we expect that underspecification indexes based on Pearson, Cosine, and Kendall should have a positive correlation with accuracy as dataset length is increased, whereas Euclidean-based indexes should have a negative correlation.*

Each of the datasets used can be found within the datasets' directory of the project repository (<https://github.com/JamesHinns/Underspecification-Index>). Prior to stratification we minimally process each dataset, removing rows with missing information and converting any non-numeric features to a numeric format. The exact code used to run each data experiment can be found with the relevant Python file in the experiments' directory.

Below we summarise the prediction problem we pose for each dataset:

Abalone: To create a binary classification problem from the Abalone dataset, we create a new binary feature that is whether a given abalone has at least ten rings. We chose this value as it makes the classes approximately even. To predict an Abalone's weight, predictors are provided eight features describing its physical measurements and sex.

Adult: A number of features extracted from 1994 American census data, such as education level, hours per week worked and occupation area are used to predict whether a given person makes under \$50,000 or not.

Bank: Input features such as job, education and the outcome of the previous marketing campaign are used to predict whether a person will subscribe to a term deposit or not.

Breast Cancer: Ten continuous features describe an image of a breast mass, from which we predict whether it is malignant or benign.

Mushroom: From features describing the appearance of a mushroom and its surroundings, we predict whether this mushroom is poisonous or not.

Wine Quality: The dataset has input variables that describe the chemical compositions of various red wines, and a prediction target of quality out of 10. To make this a binary classification problem we predict whether the quality of a given is greater than 5 or not, which creates to approximately balanced classes.

7.4 Experiment Results

For both regression and classification we compute the average prediction variance for each strata. For classification problems, this prediction variance is less meaningful than for regression. As we only experiment with binary classification problems the predictions can only be 0 or 1, and therefore little variance between them can occur. For this reason we don't discuss this relationship in this section.

To produce all the figures within this section and the matching section in regression (Section 8.3) we use altair to create graphing function, which take our saved underspecification indices and predictor performances as inputs and output created charts. We use these functions to produce and save a range of charts for each dataset. Further to the figures within this section we include charts of results from these experiments within the appendix Section A.

Table 7.4.1: The Pearson correlation between each different variation of set underspecification index and average prediction accuracy of all strata for each dataset.

Dataset	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
Abalone	-0.787	0.784	0.756	0.925
Adult	-0.986	0.987	0.990	0.868
Bank	-0.954	0.978	0.974	0.383
Breast Cancer	-0.856	0.989	0.987	-0.972
Mushroom	-0.895	0.960	0.951	-0.953
Wine Quality	-0.983	0.946	0.944	0.938

In order to assess the effect of the number of predictors we study in our empirical Rashomon set, we compare the Mushroom dataset experiment varying the number of predictors. Instead of the constant 100 predictors, we run the experiment with 10, 100 and 1000 predictors. We show the approximate effect of the predictor count on runtime in Table 7.4.2. The trade-off against this increasing runtime is the reliability of the underspecification index computed. We show this in Figure 7.4.1, showing all graphs display similar trends, but a decrease in variation can be seen with an increase in predictor count. In Table 7.4.3 we numerically represent this.

We show that the variation of explanations, represented by our underspecification indices, decreases as training dataset length and prediction accuracy increase. We

7. Implementing Underspecification Index for Classification

Table 7.4.2: Approximate runtime for the mushroom experiment, run with a different number of predictors in the studied Rashomon set $R(\theta)$.

Model Count	Runtime (s)
10	12.19
100	216.57
1000	17502.27

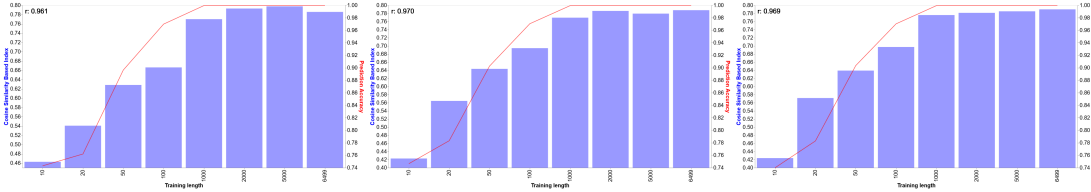


Figure 7.4.1: The Cosine-based set underspecification indices and prediction accuracy for the mushroom dataset for different strata of experiments run varying predictors studied. Left to right, the number of predictors studied is 10, 100, 1000.

Table 7.4.3: The sum of absolute differences between each of the 1625 local underspecification indices for each of the 8 strata of the mushroom dataset between the most reliable 1000 count experiment and the 10 (\mathcal{D}_{10}) and 100 (\mathcal{D}_{100}) experiments.

Metric	\mathcal{D}_{10}	\mathcal{D}_{100}
Euclidean	127.35	48.13
Cosine	382.76	132.94
Pearson	450.43	162.49
Kendall	311.30	67.77

find that this relationship is consistent across all datasets we test in, as shown in Table 7.4.1. To produce this table, we take the average prediction accuracy, average prediction variance, and set underspecification indices calculated by each of the metrics we use for each strata. This gives us a representation of explanation variance and performance for each strata of each dataset. Finally, we take the Pearson correlation between our set underspecification indices and prediction accuracy. This Pearson correlation coefficient gives us a simple singular value to evaluate against Hypothesis 2.

As expected in Hypothesis 2, we find that Cosine similarity and Pearson correlation based indices correlate similarly with classification accuracy. These metrics show a strong positive relationship between explanation agreement and prediction accuracy as training dataset length increases.

Generally we see a similar linear correlation between accuracy and Euclidean distance between explanation, only a negative relationship rather than positive, as expected by Hypothesis 2. This is expected as when Euclidean distance between explanations decreases, the explanations are more similar, as apposed to the opposite relationship with Cosine similarity and Pearson correlation. The result for the breast cancer dataset displayed in Table 7.4.1 is averaged from repeating the experiment 10

times. Because of the small size of the dataset, the random splitting of training and test sets can have larger effects on results than on larger sets. Equally, we see in Figure 7.4.2 that the dataset performs well even with the shortest dataset length selected, alongside this, we see that the Euclidean distance between explanations begins low and doesn't change. This means these random effects can be more influential than on longer, more difficult datasets, and so disrupts the trend of explanation constraint with dataset length. We show this effect in Table 7.4.4, where we see a much higher correlation in the average repeated results compared to an individual experiment.

Table 7.4.4: The correlation between each different variation of set underspecification index and average prediction accuracy of all strata for the breast cancer dataset, with a single experiment run vs the average from 10 experiments.

Repeats	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
1	-0.216	0.832	0.817	-0.798
10	-0.856	0.989	0.987	-0.972

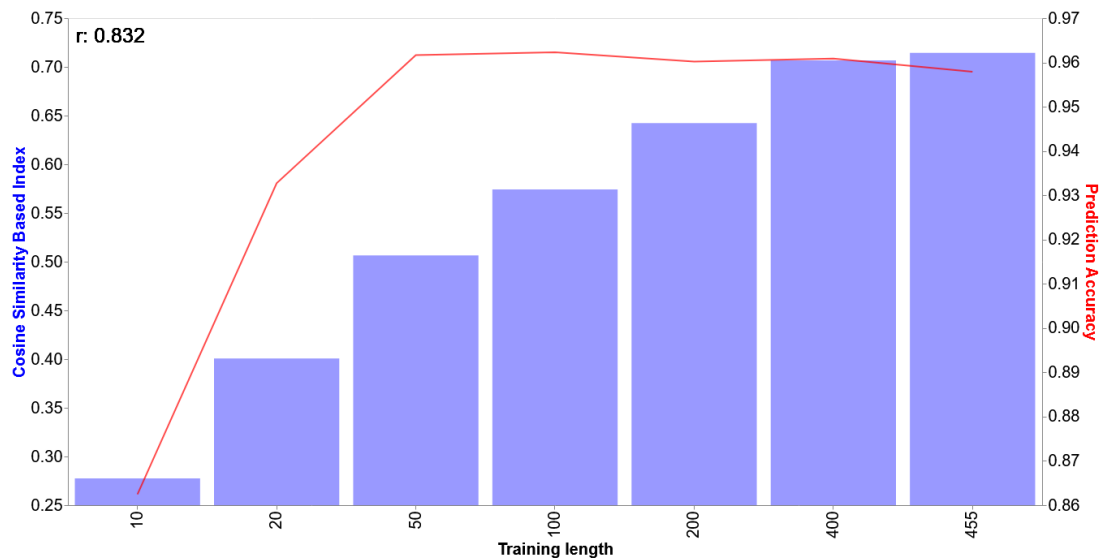


Figure 7.4.2: The Cosine-based set underspecification indices and prediction accuracy for the breast cancer dataset for different strata, produced by a single experiment run

We find far more mixed results from underspecification indices produced using Kendall rank correlation. Some results are in agreement with Hypothesis 2, whereas some did not. Demonstrated again by Table 7.4.1, we observe the expected strong positive linear relationship for the abalone, adult and wine quality datasets. We also observe three datasets where Kendall based underspecification indices do not follow the hypothesised trend, in the bank, breast cancer and mushroom datasets. Upon investigation, we observe that these datasets for which Kendall rank correlation based underspecifica-

tion indices disagree with Hypothesis 2 have the greatest number of features, as shown in Table 7.3.1. Taking the correlation between number of features and the correlation between Kendall underspecification indices and prediction accuracy (from Table 7.4.1), gives a Pearson coefficient of -0.92 . This shows that for our experiments, datasets with fewer features exhibit a relationship between Kendall underspecification indices and prediction accuracy closer to our expectation. This is likely due to the loss of the magnitude of contributions when converting explanations to rankings. Similar explanations can produce rankings in disagreement and different explanations may produce similar rankings. These interactions can have a greater effect if multiple features are assigned similar attributions, where small variations can drastically change a produced ranking. With a greater number of features, these small variations have a larger effect as they change a greater number of features in the ranking. This observed effect suggests that Kendall rank correlation based underspecification indices do not well represent agreement between explanations.

As well as using the plots shown in appendix Section A which show the relationship between prediction accuracy and underspecification indices against training length for each metric individually, we can also plot them all as a single scatter chart for each dataset. We show such plots in Figures 7.4.3 and 7.4.4 where we can see the expected patterns of Cosine, Pearson and Euclidean based indices, and also see the weaknesses of Kendall based indices.

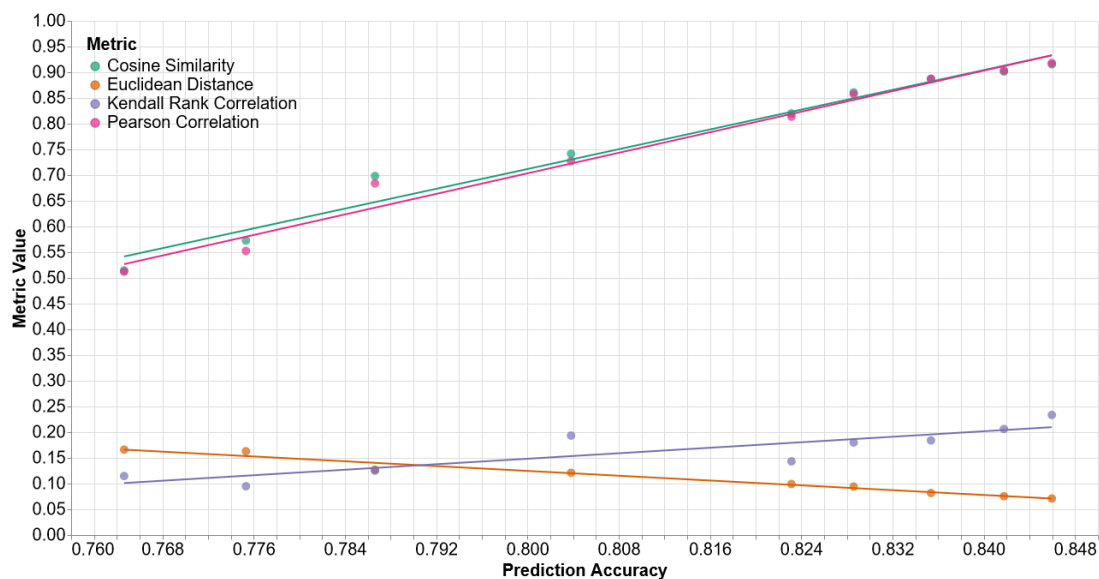


Figure 7.4.3: Different variations of underspecification indices plotted against their prediction accuracy, regression lines through each variation for all strata of the adult dataset.

To visually reinforce these findings, we include charts below demonstrating the expected pattern of increasing explanation agreement (represented by our Cosine-based

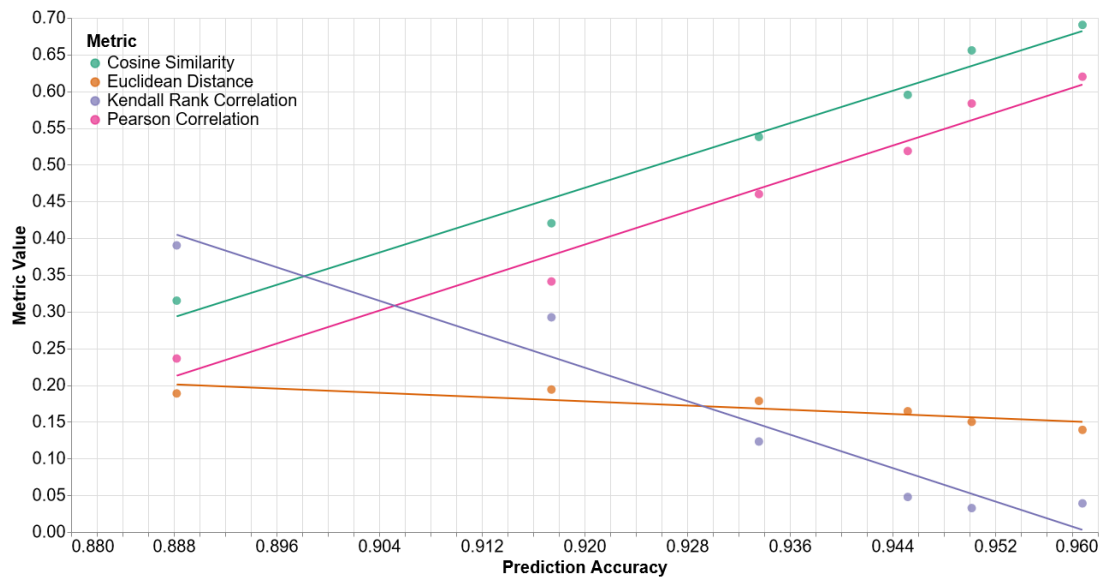


Figure 7.4.4: Different variations of underspecification indices plotted against their prediction accuracy, regression lines through each variation for all strata of the breast cancer dataset.

indices) and prediction accuracy with dataset length. In the mushroom dataset (shown in Figure 7.4.9) we see explanation agreement continues to increase even once prediction accuracy has reached 100%. This shows that our underspecification indices are able to represent how well a dataset constrains predictors more so than directly analysing predictor performance as all predictors in these cases perform equivalently.

7. Implementing Underspecification Index for Classification

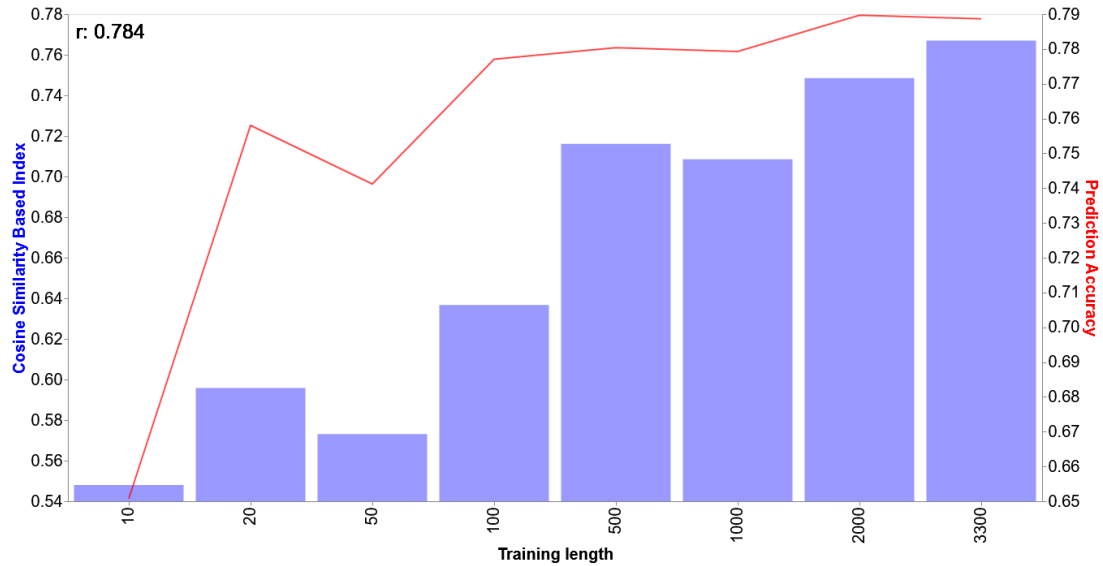


Figure 7.4.5: The Cosine-based set underspecification indices and prediction accuracy for the abalone dataset for different strata.

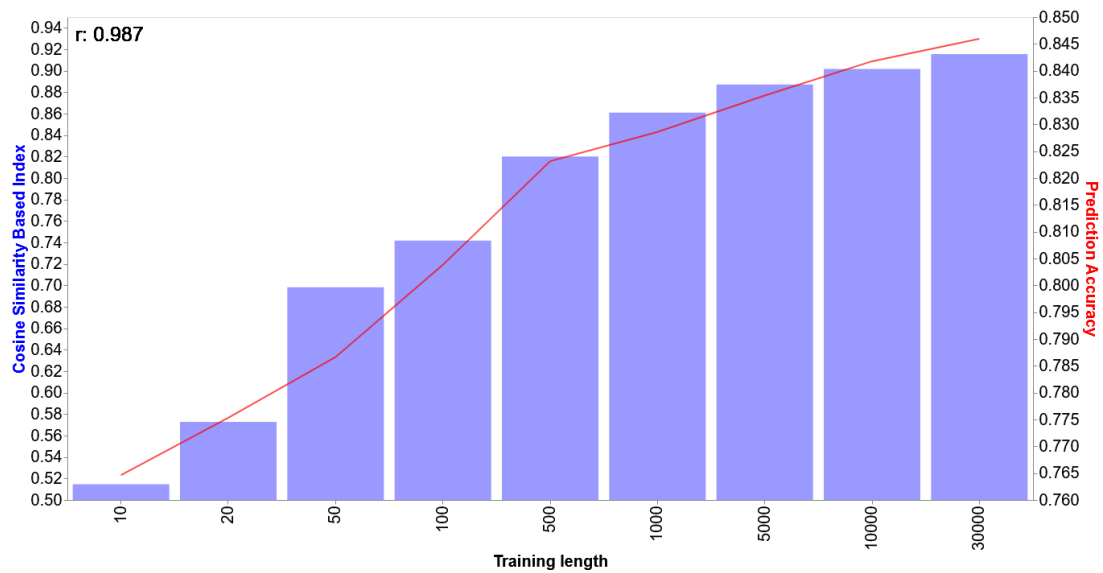


Figure 7.4.6: The Cosine-based set underspecification indices and prediction accuracy for the adult dataset for different strata.

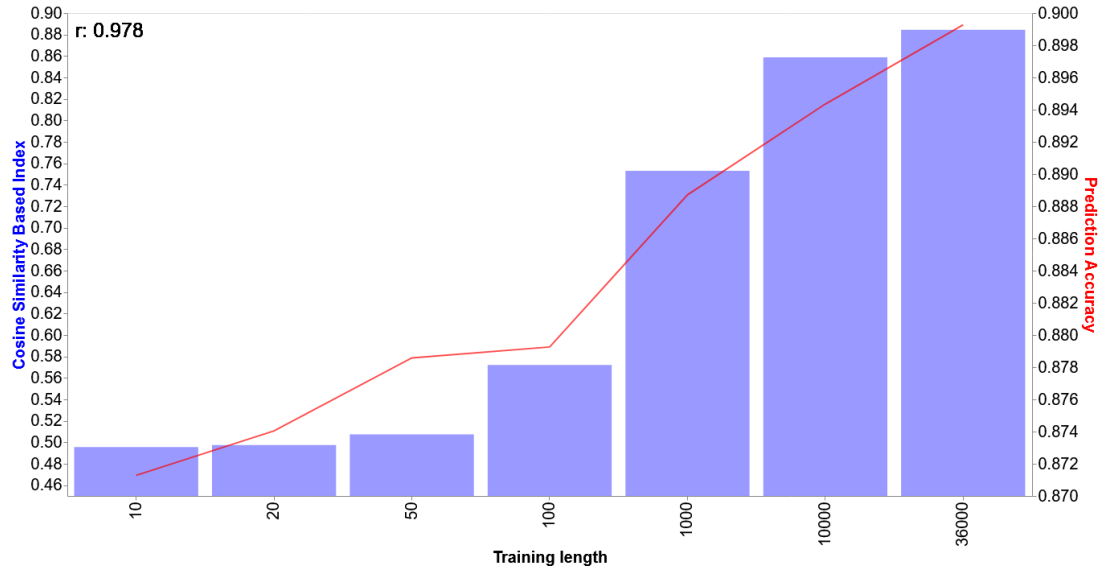


Figure 7.4.7: The Cosine-based set underspecification indices and prediction accuracy for the bank dataset for different strata.

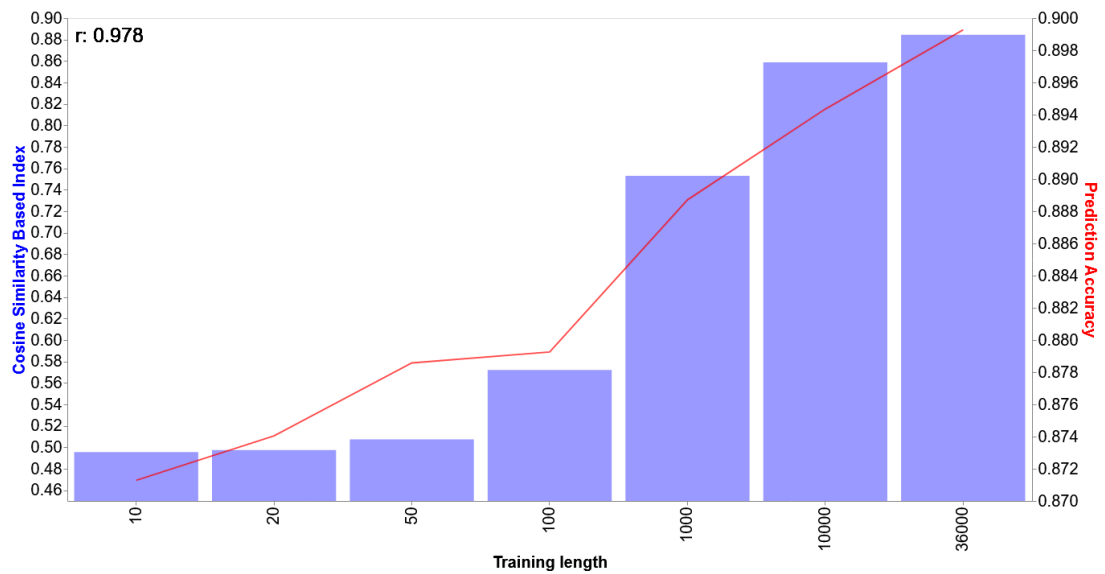


Figure 7.4.8: The average Cosine-based set underspecification indices and prediction accuracy for the breast cancer dataset for different strata, produced by repeating the experiment 10 times.

7. Implementing Underspecification Index for Classification

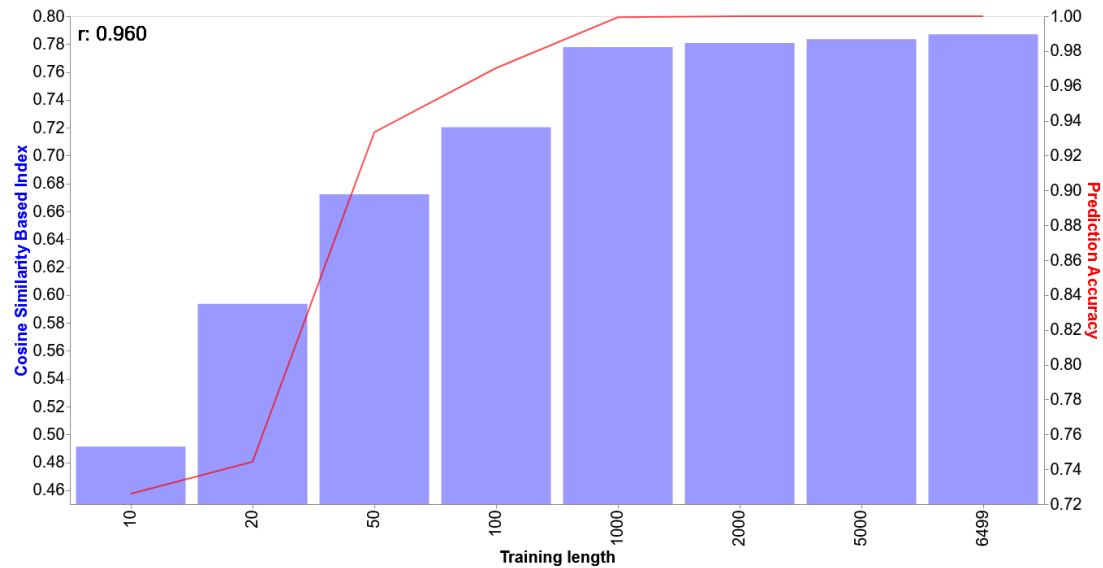


Figure 7.4.9: The Cosine-based set underspecification indices and prediction accuracy for the mushroom dataset for different strata.

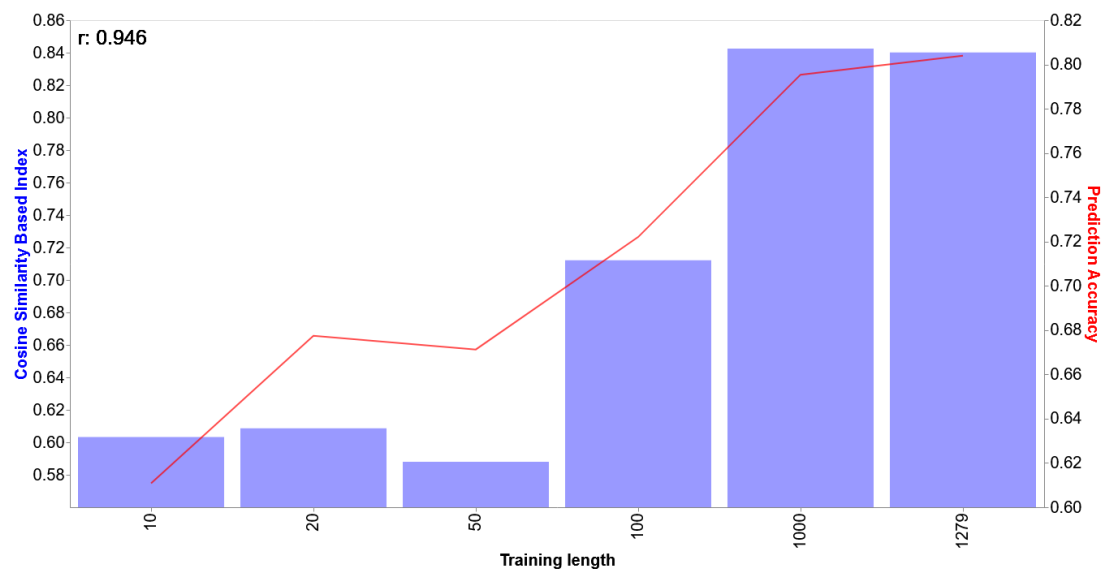


Figure 7.4.10: The Cosine-based set underspecification indices and prediction accuracy for the wine quality dataset for different strata.

Chapter 8

Implementing Underspecification Index for Regression

Contents

8.1	Software Realization	77
8.2	Experimental Setup	78
8.3	Experiment Results	80

This sections details implementing and testing our method for computing underspecification indices for regression problems. We build upon the work from the previous chapter, 7, to adjust the produced implementation to work in both classification and regression problems. As in the previous chapter, we discuss how we realise these changes in our python implementation. We then again test this implementation using stratified performance evaluations on datasets from the literature and discuss our interpretation of these results.

8.1 Software Realization

Our implementation for regression uses mostly the same code as for classification. Our implementation for Algorithm 2, which calculates local underspecification indices \mathcal{U}_x , and the function that calculates the set underspecification index $\bar{\mathcal{U}}$ see no change, as they are just passed explanations, which do not change between classification and regression. The changes are seen within our implementation for Algorithm 1 which generates explanation matrices. As multiple parts of this algorithm function differently for classification and regression, we create two sub-functions for classification and regression respectively. These functions generate a predictor, if it outperforms the given threshold θ it's SHAP explanation $\Pi(f, \mathbf{x})$ for each data instance $x \in D_{tst}$. The function that calls both of these sub-functions iteratively calls the relevant sub-function until the requested model count has been reached. The first difference between the regression and

classification sub-functions is simply in the model type of predictors. For classification, we use SKLearn random forest classifiers as opposed to random forest regressors for regression. Similarly, in both we train these models to produce predictors and then predict the target variable from input variables, before evaluating performance.

As discussed in Section 2.2, different metrics are used for classification and regression problems. Therefore, instead of maximising accuracy we minimise mean squared error (MSE). For regression problems we look for predictors with an MSE lower than θ whereas in classification problems we look for predictors with an accuracy greater than θ . Finally, when used for classification the SHAP library produces explanations for each class, where we only analyse the explanations for the first class. When used for regression, the SHAP library only provides one explanation for one data instance, so no filtering is needed.

The final difference for regression problems is that for each model we calculate the squared error for each test instance, whereas in classification we calculate the accuracy across the whole dataset. This is because classification can only be right or wrong, and local evaluation in this context doesn't provide meaningful results. By calculating the error at each instance we are able to calculate a more meaningful prediction variance across produced predictors in the Rashomon set. To calculate this squared error, we implement a simple function using Numba and NumPy that returns an array of the squared error for all test instances. In order to calculate the MSE, we simply take the mean of the returned array.

8.2 Experimental Setup

As in chapter 7, we experiment our implementation using a number of datasets from the literature. Other than passing an additional parameter to indicate we want to calculate underspecification indices for regression, our experimental control functions see no difference. We again stratify each dataset into a number of subsets of varying length, train a number of predictors on these strata and compute their set underspecification indices for a shared held-out testing dataset.

Within the project repository we include an example notebook, `Bikeshare_Example.ipynb`, which shows an example of how we ran these experiments in the regression setting using the Bike Share dataset [DG17, FTG13].

In agreement with our classification results and in line with the general Hypothesis 1 (pg. 59), we expect that prediction performance and explanation agreement should increase with dataset length. As with the previous chapter we expand Hypothesis 1 for our experiments and setting, in this case, regression problems. In parallel with Hypothesis 2 we give the following hypothesis relating performance (MSE) to explanation agreement:

Hypothesis 3 *As more examples are added to a dataset, the MSE and explanation agreement should increase. Specifically, we expect that underspecification indexes based on Pearson, Cosine, and Kendall should have a negative correlation with MSE as dataset*

length is increased, whereas Euclidean-based indexes should have a positive correlation.

In this chapter, we additionally discuss the relationship of prediction variance across strata as this is a meaningful reflection of performance variation in a regression setting. We foresee that prediction variance should decrease with dataset length as predictors are better constrained. We provide the following hypothesis to represent this expected trend:

Hypothesis 4 *As more examples are added to a dataset, the prediction variance should decrease, whilst explanation agreement should increase. Specifically, we expect that underspecification indexes based on Pearson, Cosine, and Kendall should have a negative correlation with variance as dataset length is increased, whereas Euclidean-based indexes should have a positive correlation.*

Table 8.2.1: Summary of datasets used for regression experiments, all but the house price dataset are available on the UCI ML repository [DG17].

Dataset [DG17]	# of Samples	# of Features
Auto MPG [DG17]	398	8
Bike Share [DG17, FTG13]	17379	16
House Price [Coc11]	1461	79
Student [DG17, CS08]	649	30
Wine Quality [DG17, CCA ⁺ 09]	1599	11

We summarise the five datasets we use in Table 8.2.1 by the number of samples and features. Below we summarise the prediction problem we pose for each dataset:

Auto MPG: Given 8 features that describe a given car such as number of cylinders, horsepower and weight, we aim to predict the average miles-per-gallon (MPG).

Bike Share: Input features describing the date, weather and total users are used to predict the amount of bikes rented for a given hour.

House Price: Input features characterise the location, size and amenities of a given house, from which we predict the sale price.

Student: Input features describe student demographic, as well as social and school background in secondary education of two Portuguese schools. We aim to predict a given student’s final grade in Portuguese. It should be noted this dataset includes intermediate test scores, which we exclude in order to make the prediction problem more difficult.

Wine Quality: The dataset has input variables that describe the chemical compositions of various red wines with a prediction target of quality out of 10. Rather than forming this as a binary classification as we did previously, or a multi-class classification, we pose predicting quality as a regression problem.

8.3 Experiment Results

We measure Pearson correlation between average prediction mean squared error and each of our underspecification indices produced by different metrics, as we did in the previous chapter. We show these correlation results in Table 8.3.1. Furthermore, we observe a strong positive correlation between error and Euclidean distance, following Hypothesis 3. Once again Pearson correlation and Cosine similarity based indices show an almost equivalent trend with test performance. We observe underspecification indices based on these metrics produce a strong negative Pearson correlation with prediction error, again supporting Hypothesis 3.

As in classification, we observe that Kendall rank correlation based underspecification indices produce mixed results. We compute the Pearson correlation between the results shown in Table 8.3.1 and the number of features shown in Table 8.2.1 and find a coefficient of 0.96. This means in agreement with our classification results, the greater number of features in a dataset the less Kendall based indices match the relevant hypothesis, in this case that is Hypothesis 3.

Table 8.3.1: The correlation between each different variation of underspecification index and prediction mean squared error of all strata for each dataset.

Dataset	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
Auto MPG	0.984	-0.943	-0.921	-0.890
Bike Share	0.899	-0.912	-0.919	-0.824
House Price	0.935	-0.973	-0.973	0.864
Student	0.968	-0.964	-0.965	-0.47
Wine Quality	0.893	-0.955	-0.955	-0.816

We find that the dataset with the fewest samples (data instances) we use, Auto MPG, produces less consistent results than the datasets with a greater number of samples. As we do for smaller datasets that have similarly varying results for classification, we repeat the experiment for this dataset ten times and report the average of these. We display the comparison of the repeated experiment and a single run in table 8.3.2, we use the repeated result in table 8.3.1 and 8.3.3. When repeating results and taking the average we observe greater correlation between our underspecification indices and prediction mean squared error. This is because of the randomness involved in the process, such as the random splitting of training and testing datasets. A dataset with fewer samples sees greater effects from this as they are likely less representative of their respective distribution.

As well as comparing the trend of underspecification indices to prediction accuracy as length increases, we also compare prediction variance. We summarise this relationship in Table 8.3.3, which is equivalent to Table 8.3.1 but replacing error with variance. A noticeable outlier in this table can be seen in the Wine Quality dataset. We have included this result still for the sake of completeness, however due to the nature of the prediction problem the prediction variance is not as meaningful nor equivalent to the

Table 8.3.2: The correlation between each different variation of underspecification index and prediction mean squared error of all strata for the Auto MPG dataset, with a single run compared to the average of ten repeated experiments.

Repeats	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
1	0.975	-0.692	-0.65	-0.683
10	0.984	-0.943	-0.921	-0.890

other datasets. This is because rather than a true continuous prediction target, we aim to predict an integer quality score between 1 and 10. The dataset also does not cover all values between 1 and 10, particularly on the lower end of scores. For this reason we will not discuss the variance results from this dataset. We equally will not discuss the Kendall based indices for the reasons detailed above and in the previous chapter.

We observe a strong positive correlation between Euclidean distance based indices and prediction variance. This is possibly the most expected result from our experiments, as SHAP explanations should model prediction variance. However, as SHAP distributes the difference from base value to prediction output, rather than the actual prediction output it is not a true measure of prediction variance. We see a strong negative correlation for both Pearson and Cosine-based indices with both indices producing similar results once again. These results support Hypothesis 4 that explanation agreement correlates strongly with prediction variance and as such is a good metric of how much a given dataset constrains predictors produced from it.

Table 8.3.3: The correlation between each different variation of underspecification index and prediction variance of all strata for each dataset.

Dataset	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
Auto MPG	0.977	-0.992	-0.985	-0.817
Bike Share	0.994	-0.933	-0.934	-0.783
House Price	0.894	-0.964	-0.963	0.896
Student	0.97	-0.95	-0.951	-0.548
Wine Quality	0.367	-0.0517	-0.0673	-0.46

As discussed above, we repeated the experiment for the Auto MPG dataset due to the instability of results based on the random train-test split. As in all summary tables, we include this repeated result for the prediction variance correlation table (table 8.3.3). We show the difference of explanation variance from a single run to the averaged results in table 8.3.4. In agreement with all other repeated experiments, we observe that the repeated average results are closer to the expected trend than a single run. We discuss reasoning for this above.

To further support these findings visually, we include the results of the cosine based indices and prediction error for each dataset below. We show further charts of these

8. Implementing Underspecification Index for Regression

Repeats	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
1	0.975	-0.692	-0.650	-0.683
10	0.977	-0.992	-0.985	-0.817

Table 8.3.4: The correlation between each different variation of underspecification index and prediction variance of all strata for the Auto MPG dataset, with a single run compared to the average of ten repeated experiments.

results for each metric in Appendix B.

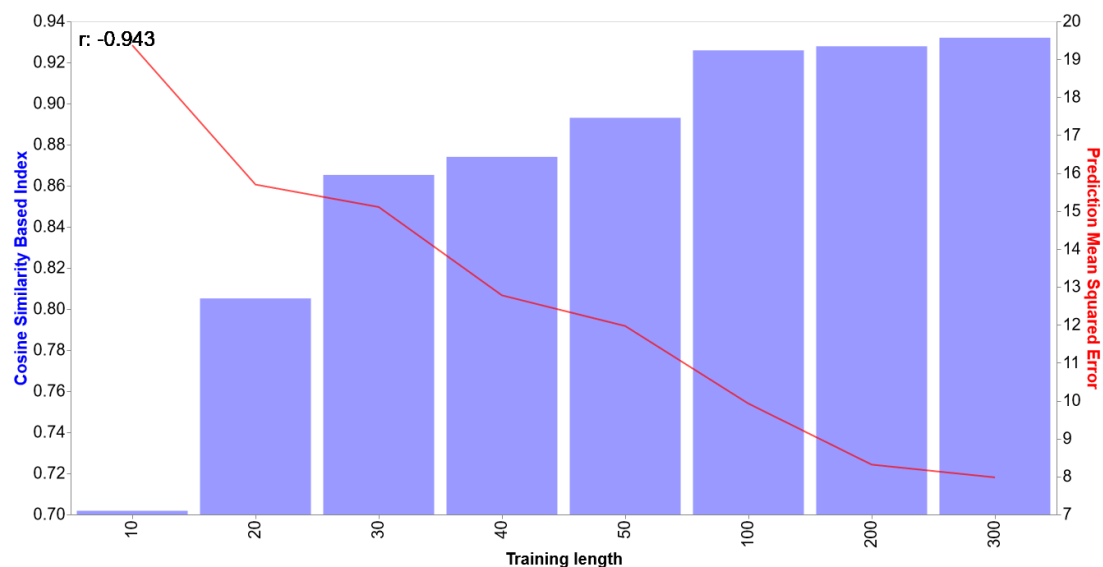


Figure 8.3.1: The Cosine-based set underspecification indices and prediction mean squared error for the Auto MPG dataset for different strata. Data is taken from the average of ten repeated experiments.

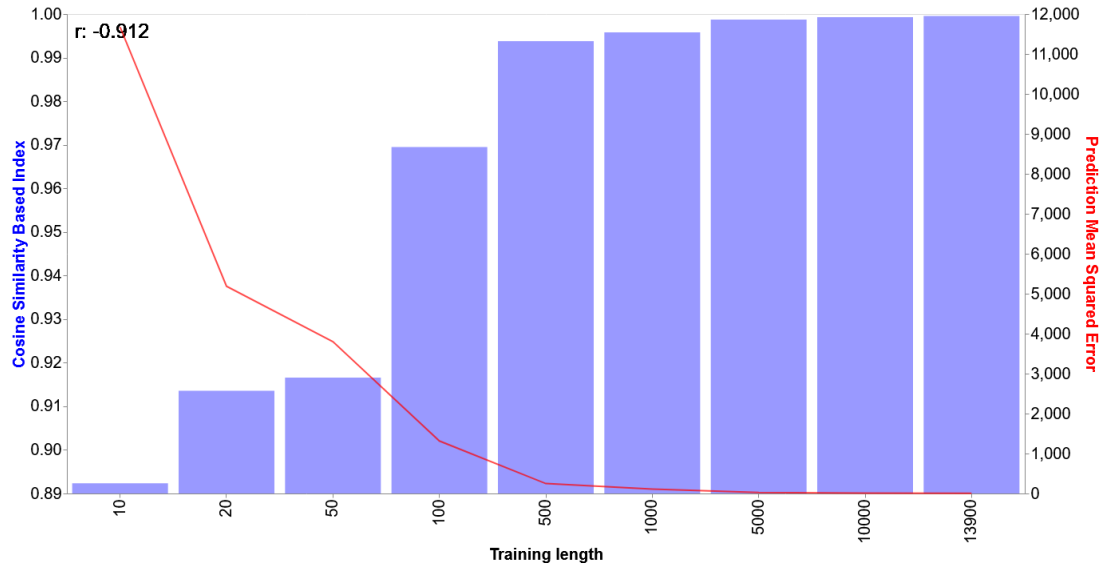


Figure 8.3.2: The Cosine-based set underspecification indices and prediction mean squared error for the bike share dataset for different strata.

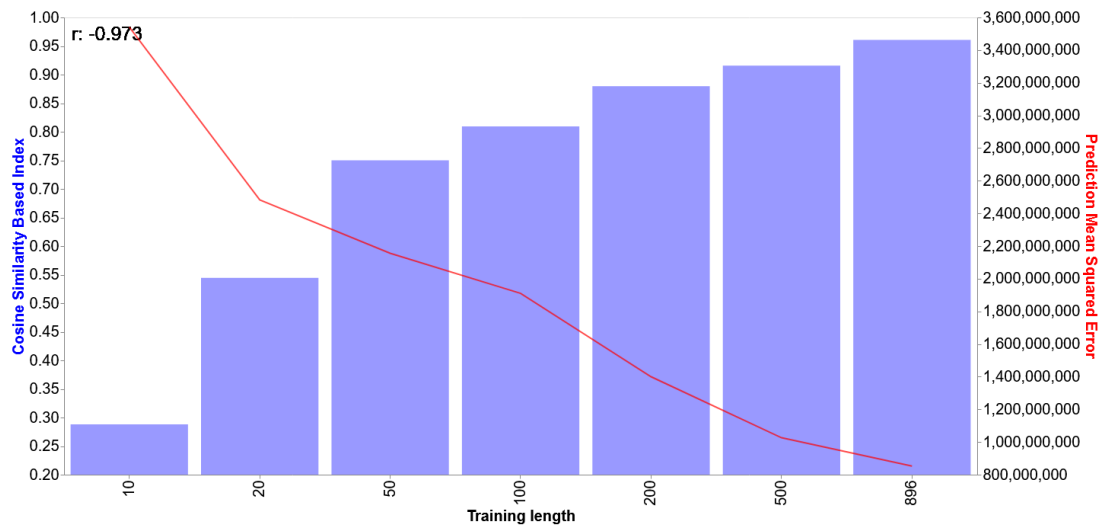


Figure 8.3.3: The Cosine-based set underspecification indices and prediction mean squared error for the house price dataset for different strata.

8. Implementing Underspecification Index for Regression

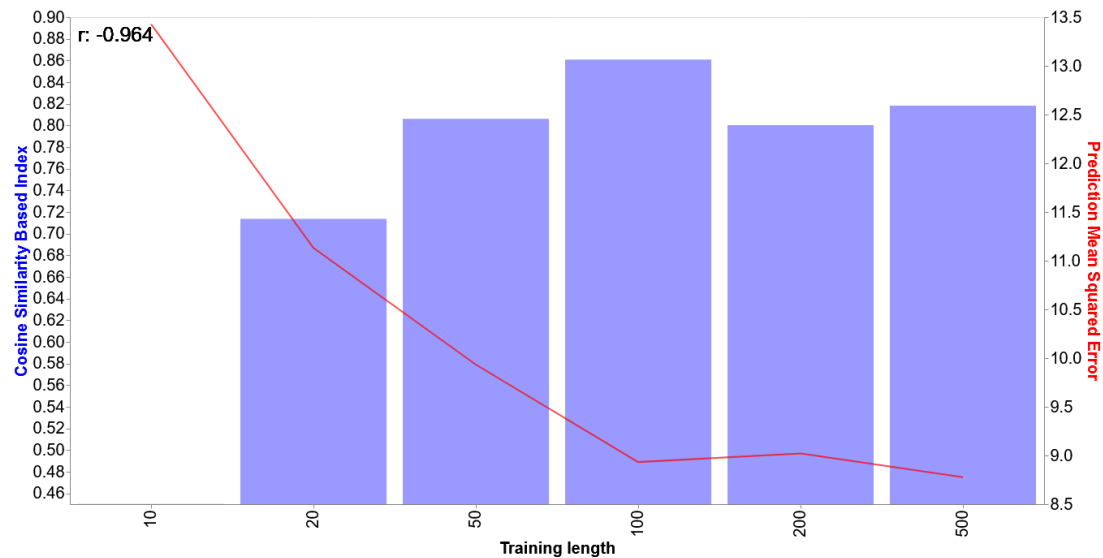


Figure 8.3.4: The Cosine-based set underspecification indices and prediction mean squared error for the student dataset for different strata.

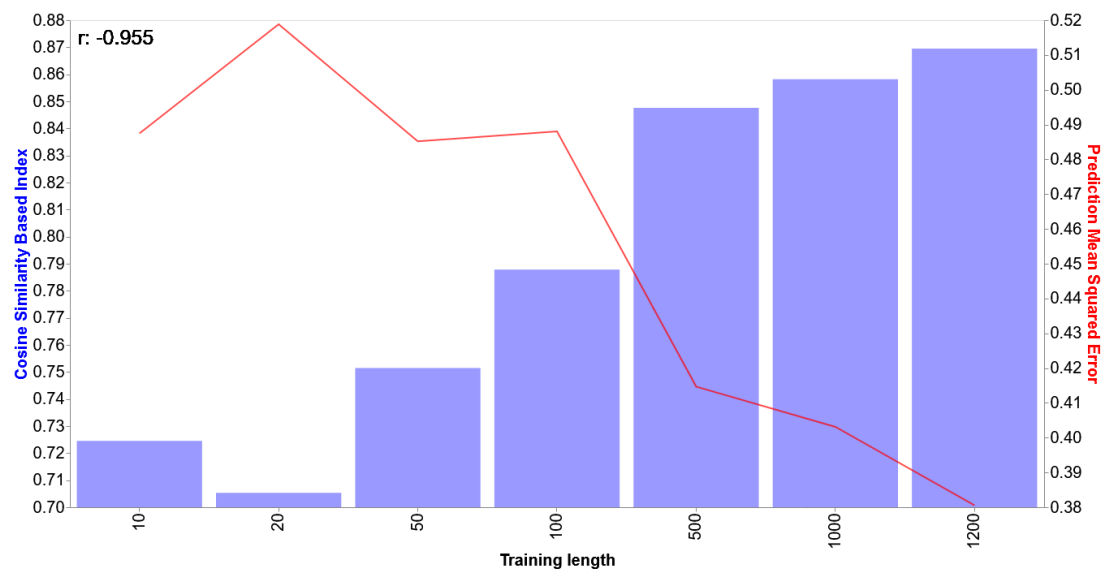


Figure 8.3.5: The Cosine-based set underspecification indices and prediction mean squared error for the wine quality for different strata.

Chapter 9

COVID-19 Case Study

Contents

9.1	Dataset Creation	85
9.2	Experimentation	88

In this section we apply our approach to a coronavirus virus transmission case study. This case study can be viewed as a realistic experiment modelled after the epidemiological model that demonstrates underspecification in [DHM⁺20]. We discuss the stages taken to create this dataset and its final format. Using this dataset, we pose multiple prediction problems from which we undertake experiments. Finally, we detail the setup and results of these experiments.

9.1 Dataset Creation

The COVID dataset we use in this section was made by Veera Raghava Reddy Kovvuri and Dr. Xiuyi Fan and published in our work [HFL⁺21] and external work [KED⁺21].

In order to create the dataset, we collected data from multiple sources. From the Public Health England website¹, we collected daily cases and deaths reported across 12 regions in UK: East Midlands, East of England, London, North East, North West, Northern Ireland, Scotland, South East, South West, Wales, West Midlands as well as Yorkshire and The Humber.

Non-pharmaceutical control measure data was composed based on UK's COVID policies as summarised in Table 9.1.1. Data was collected from various sources including Wikipedia and major news agencies. Control Measures were coded based on level of severity (e.g., 'High', 'Moderate', 'Low') for all control measures excluding non-essential shops and school closures, which are coded as binary choices ("Open" and "Closed"). In total 4,257 data instances were collected between February 2020 and February 2021.

¹<https://www.gov.uk/government/organisations/public-health-england>

Table 9.1.1: Non-pharmaceutical COVID Control Measures.

Meeting Friends / Family (Indoor)	Meeting Friends/Family (Outdoor)
Domestic Travel Control	International Travel Control
Cafes and Restaurants Control	Pubs and Bars Control
Sports and Leisure Closure	Hospitals / Care and Nursing Home Visits
Non-Essential Shops Closure	School Closure

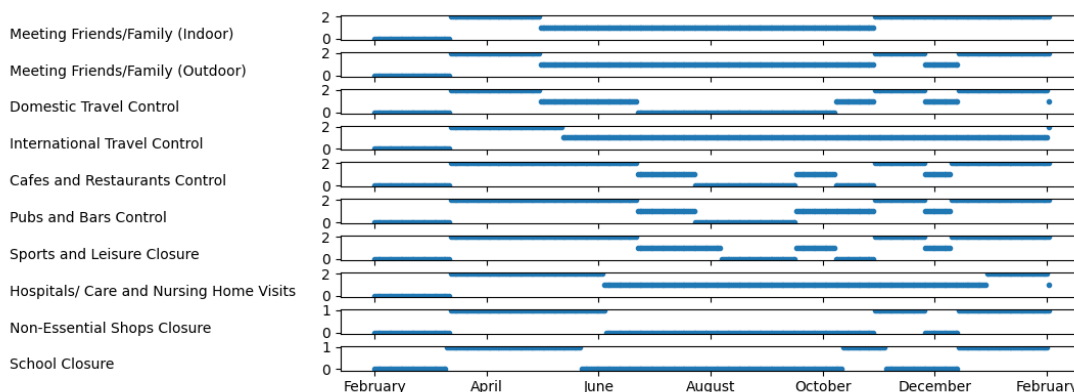


Figure 9.1.1: COVID control measures implemented in London from February 2020 to February 2021. Control measures with ternary values, 'High', 'Moderate' and 'Low' are represented with 2, 1 and 0, respectively. For binary control values, 'Open' and 'Closed' are represented with 0 and 1, respectively.

From daily infection numbers, we estimate the reproduction number R_t using the method reported in [FMG⁺20, WLB⁺20]. R_t is one of the most important quantities used to measure the spread of an epidemic. If $R_t > 1$, then the epidemic is expanding at time t , whereas if $R_t < 1$, then it is shrinking at time t . A *serial interval distribution*, which is a Gamma distribution $g(\tau)$ with mean 7 and standard deviation 4.5, is used to model the time between a person getting infected and then subsequently infecting another person on day τ . The number of new infections c_t on a day t is computed as:

$$c_t = R_t \sum_{\tau=0}^{t-1} c_\tau g_{t-\tau}, \quad (9.1)$$

where c_τ is the number of new infections on day τ ,

$$g_1 = \int_{\tau=0}^{1.5} g(\tau) d\tau,$$

and for $s = 2, 3, \dots$,

$$g_s = \int_{\tau=s-0.5}^{s+0.5} g(\tau) d\tau.$$

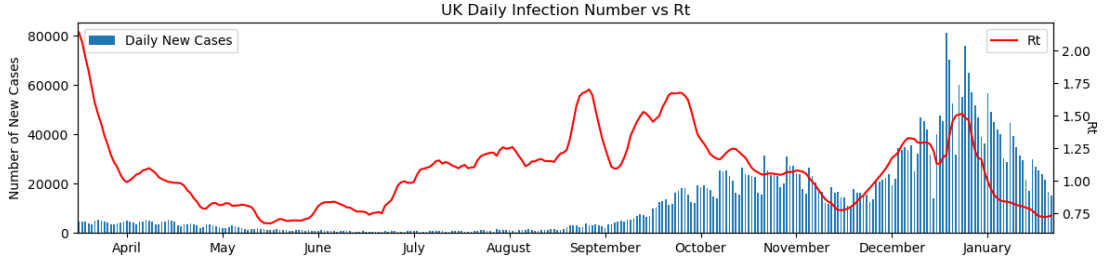


Figure 9.1.2: Daily Infection Cases vs Rt in UK.

From Equation 9.1, we have:

$$R_t = \frac{c_t}{\sum_{\tau=0}^{t-1} c_{\tau} g_{t-\tau}} \quad (9.2)$$

For $x = t$ and τ , c_x is the difference between the confirmed case on day x and the confirmed case on day $x - 1$, which is available from the dataset directly.

Figure 9.1.2 illustrates the relation between R_t and daily infection cases. Throughout, daily infection numbers are passed through a median filter with window size 7 to smooth out noises.

To account for the fact that control measures take time to affect the reproduction rate, we expand the dataset to include the duration of control measure implementation for all control measures. For example, “*Meeting Indoors (High) = 5*” means that “*it is the 5th day that meeting indoors has been banned completely*”. Similarly, *International Travel (Low) = 0* means that “*there is no restriction implemented on international travel*”.

Additionally, we drop instances before the 15th of March 2020 across all 12 regions in our dataset due to the low number of infections. As can be seen from Equation 9.2, when c_x is small, R_t can fluctuate in an unrealistically large range and generate noise in the dataset.

Finally, we discretise the values of deaths and cases to simplify the prediction problem. We do this by selecting intervals, whereby each number is sorted to a class based on these intervals. For example if the intervals we select for some feature were 0,10 and 100, a value of 0 will be assigned 0, all values between 0 and 10 would be assigned 1, all values between 10 and 100 would be 2, and all values greater than 100 would be 3. We use intervals of 0, 100, 250, 500, 750, 1000 and 2000 for cases and 0, 10, 20, 30, 40, 50, 60, 70 and 80 for deaths.

For a regression problem, we use these discretised features of deaths and cases, as well as non-pharmaceutical control measures to predict the estimated reproduction number R_t . Using this same data we also pose the classification problem of predicting whether $R_t \geq 1$. This trimmed dataset contains 3,948 instances between the 15th of March 2020 and the 6th of February 2021, using 23 input features. The dataset contains 2,280 positive instances, that is, where $R_t \geq 1$.

Table 9.1.2: Top 10 attributed features from absolute global SHAP values, normalised such that the sum of all attributions is 1.

Rank	Feature	Normalised Shap Value
1	Cafes and Restaurants High	0.192
2	Pubs and Bars High	0.168
3	Cases	0.104
4	Cafes and Restaurants Moderate	0.095
5	Deaths	0.080
6	Domestic Travel Moderate	0.058
7	Sports and Leisure High	0.050
8	Sports and Leisure Moderate	0.040
9	Hospitals/ Care and Nursing Home Visits Moderate	0.034
10	School Closure	0.033

We show the normalised SHAP values of the ten features with the highest average contribution in table 9.1.2.

9.2 Experimentation

For the experiments in this section we trim the data to only include instances prior to November 2020. We trim this data just to the increasing prevalence of the B.1.1.7 variant from this point onwards. It was estimated that the B.1.1.7 had a reproduction number of 40 - 70% higher than other previously identified variants [VMC⁺21]. Furthermore, the first COVID vaccination in the UK was in December 2020 which also increasingly effected the data. For these reasons we begin to observe dataset shift throughout this period, and as such experimentation using this data does not provide a fair representation of underspecification. This final trimmed dataset contains 2772 instances, with 1731 positive instances (where $R_t \geq 1$).

As with the datasets in Chapters 7 and 8 we perform a stratified performance evaluation on this dataset to examine the effects of training length on underspecification indexes and prediction performance. As in all previous experiments we train 100 random forests with 10 trees to establish the Rashomon set for each strata. From each Rashomon set we record prediction performance and underspecification indexes and analyse their relationships to increasing dataset length. We randomly split the data using an 80-20 training test split giving us 2217 training and 555 testing samples. We split this training data into strata of size $\{10, 20, 50, 100, 1000, 2000, 2200\}$ and evaluate them against the same test set. Our performance threshold θ for the Rashomon set is set as the average performance of 100 predictors on the test set trained on the respective strata.

Due to the random train and test split, we repeat each experiment 10 times and report the average result of these. This enables our results to show clearer trends by minimising the effects of this random sampling. We show how this effect is minimised

in Figure 9.2.1, where the repeated experiment shows a greater relationship between explanation agreement and prediction performance.

Our classification results are in agreement with the results presented in Chapter 7 in fitting to our expected trend. As shown in Figure 9.2.1 and Table 9.2.1 we see that as dataset length increases, prediction accuracy and explanation agreement increase and prediction variance decreases.

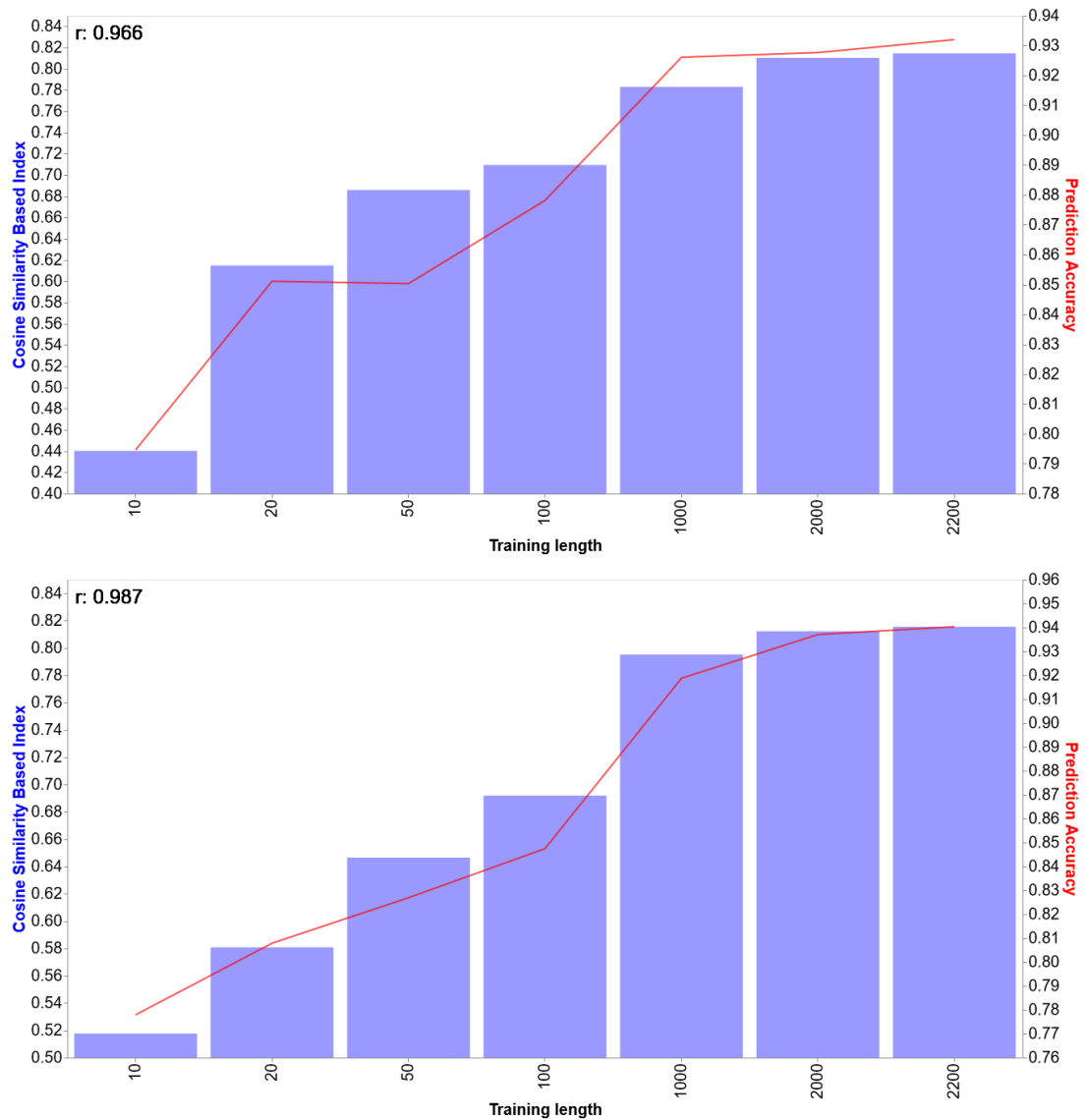


Figure 9.2.1: The Cosine-based set underspecification indexes and prediction accuracy for different strata of the COVID dataset predicting the binary classification $R_t \geq 1$. The top chart shows the result from a single experiment, the bottom shows the average results from ten experiments.

Table 9.2.1: The correlation between each different variation of set underspecification index and prediction accuracy and variance of all strata for binary classification on the COVID dataset. Data is taken from the average of ten repeated experiments.

Variable	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
Accuracy	-0.993	0.987	0.990	0.709
Variance	0.950	-0.938	-0.943	-0.724

Our regression results on the COVID case study support our results presented in Chapter 8. As shown in Figures 9.2.2 and 9.2.3, we observe decreasing prediction error and variance along with increasing explanation agreement as we increase sample size.

Table 9.2.2: The correlation between each different variation of set underspecification index and prediction accuracy and variance of all strata for regression on the COVID dataset. Data is taken from the average of ten repeated experiments.

Variable	Euclidean Distance	Cosine Similarity	Pearson Correlation	Kendall Rank Correlation
Accuracy	0.941	-0.982	-0.982	-0.926
Variance	0.992	-0.988	-0.986	-0.983

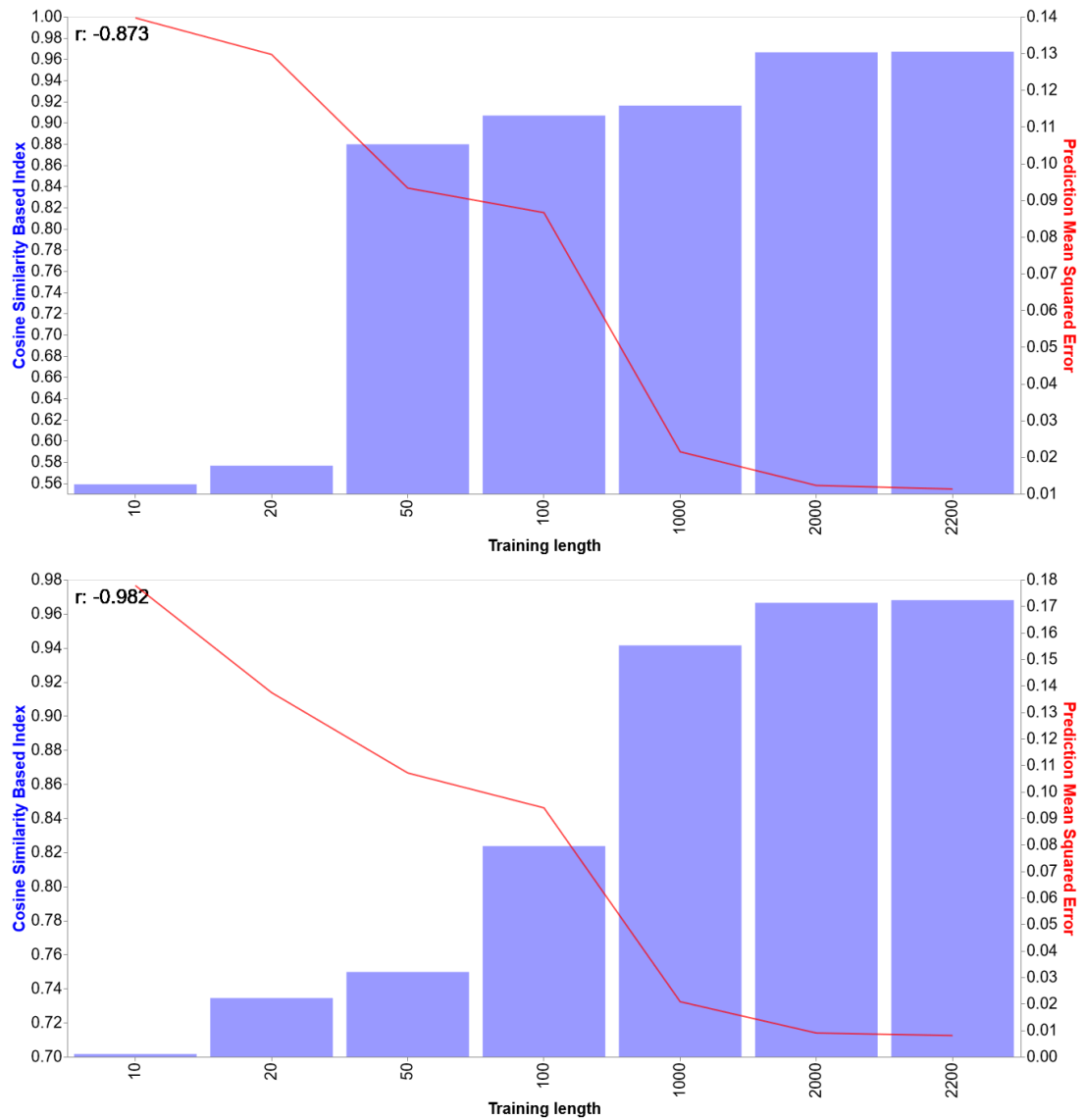


Figure 9.2.2: The Cosine-based set underspecification indexes and prediction accuracy for different strata of the COVID dataset predicting the reproduction number R_t . The top chart shows the result from a single experiment, the bottom shows the average results from ten experiments.

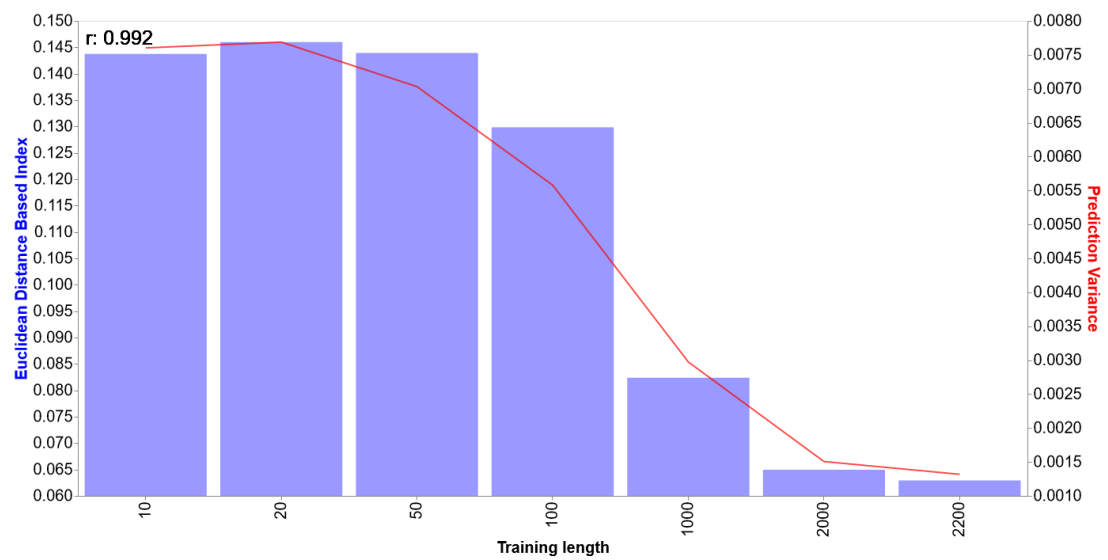


Figure 9.2.3: The Cosine-based set underspecification indexes and prediction variance for different strata of the COVID dataset predicting the reproduction number R_t .

Part III

Conclusions

Chapter 10

Conclusions and Future Work

Contents

10.1 Quantifying Underspecification with XAI	95
10.2 Future Work	98

In this final chapter, we present a summary of the work completed and review our contributions to the literature from this thesis. We follow this by discussing some limitations of our work, and how future research could address these.

10.1 Quantifying Underspecification with XAI

In this work, we presented a method for quantifying underspecification within a machine learning pipeline by examining variation between predictors.

As discussed throughout this thesis, a machine learning pipeline is underspecified if it can return many equally well performing predictors in the training domain.

As predictors are difficult to compare directly, we used SHAP explanations to model them. We then simply compared variation amongst these explanations in order to represent variation between predictors.

To quantify the likelihood that a pipeline is underspecified we repeatedly trained models that outperform a given performance threshold θ until we produced a set of similarly performing predictors of a desired size. For all predictors in this set $\mathcal{R}(\theta)$ we computed SHAP explanations for each data instance in the testing set. We then pair-wise computed the variation between each explanation to each other explanation at a particular data instance. Averaging this variation gives our local underspecification index, representing variation at that data instance. To compute our set underspecification index we took the mean of all local indexes calculated.

We created a Numba optimised Python implementation of this approach in order to run experiments to validate that these indexes behave as expectedly. This implementation was developed for both regression and classification settings. We populated the Rashomon set $\mathcal{R}(\theta)$ with random forests trained on a given training set x_{trn} , such

that all predictors in $\mathcal{R}(\theta)$ exceed the performance threshold θ on the testing set x_{tst} . We then computed an explanation matrix of explanations of each predictor for each data instance in x_{tst} . To quantify the similarity of explanations, we experimented with four pairwise metrics: Euclidean distance, Pearson correlation, Cosine similarity, and Kendall rank correlation. In our implementation we did not save the predictors themselves, only the test performance and explanation matrix produced by them. From this explanation matrix we computed our local underspecification indexes by a triangular matrix to save on computation. We calculated our set index by computing each local index in parallel, producing an array of local indexes.

We proposed a number of hypotheses to represent our expectations, and supported them via our experimentation.

Firstly, we proposed Hypothesis 1 on page 59, which expected:

As more examples are added to a dataset, the performance and explanation agreement should increase.

This is supported by the evidence for its two sub-hypotheses; Hypothesis 2 and 3.

Next we presented Hypothesis 2 on page 68 for the classification setting. This expected:

As more examples are added to a dataset, the classification accuracy and explanation agreement should increase. Specifically, we expect that underspecification indexes based on Pearson, Cosine, and Kendall should have a positive correlation with accuracy as dataset length is increased, whereas Euclidean-based indexes should have a negative correlation.

We found that this held true for indexes based on Pearson and Cosine, where, as we increased dataset length, accuracy and the indexes increased, with a high positive correlation across all datasets tested. We additionally saw this hypothesis was supported for Euclidean-based indexes, with a strong negative correlation between indexes and accuracy as dataset length was increased. However, we found that this hypothesis didn't hold for all cases with Kendall-based indexes. As we explained in section 7.4, this is due to small variations in explanations causing differences in the rankings produced, which in turn cause vastly different Kendall rank correlation coefficients. We found that the Kendall-based indexes strayed further from the hypothesis with a greater number of features in the dataset.

For the regression setting, we presented two hypotheses; Hypothesis 3 and Hypothesis 4 on page 79. Hypothesis 3 expected:

As more examples are added to a dataset, the MSE and explanation agreement should increase. Specifically, we expect that underspecification indexes based on Pearson, Cosine, and Kendall should have a negative correlation with MSE as dataset length is increased, whereas Euclidean-based indexes should have a positive correlation.

Similarly to Hypothesis 2 we found Hypothesis 3 was supported for all indexes other than those based on Kendall. Again for Kendall we saw the most similar results to the expected trend were for datasets with few features, and the results furthest from the expected trend had a greater number of features.

Hypothesis 4 does not include any mention of performance, and so is entirely separate to Hypothesis 1. It instead looks at the variation between predictions, and how this correlates to our underspecification indexes, specifically, it proposed:

As more examples are added to a dataset, the prediction variance should decrease, whilst explanation agreement should increase. Specifically, we expect that underspecification indexes based on Pearson, Cosine, and Kendall should have a negative correlation with variance as dataset length is increased, whereas Euclidean-based indexes should have a positive correlation.

As with the previous hypotheses, Kendall does not support this hypothesis, for the same reasoning. We do however, find that for all but one dataset our results strongly support this hypothesis. Our results show that Euclidean-based indexes have a strong positive correlation with prediction variance. Furthermore, our results support the prediction that Cosine and Pearson based indexes would have a strong negative correlation with prediction variance.

Further to these results, we experimented in both classification and regression with a real COVID-19 case study. The results from this case study were in agreement with our previously found results, with all indexes apart from Kendall supporting hypotheses 2, 3 and 4.

The main contributions of this thesis are as follows:

1. We formulated underspecification quantification as a problem of measuring similarity between explanations produced by predictors in a Rashomon set.
2. We quantified the similarity of explanations using four well studied metrics, Euclidean distance, Pearson correlation, Cosine similarity, and Kendall rank correlation.
3. We provided a Numba optimised Python implementation of our proposed approach for both classification and regression problems. This implementation can be found:

<https://github.com/JamesHinns/Underspecification-Index>.

4. We demonstrated our approach for classification and regression on both existing datasets in the literature and a real-world COVID-19 dataset.

10.2 Future Work

This work is an initial study into quantification of underspecification with explainable AI. As such it raises a number of further questions to be answered by further research. Below we outline some possible future research directions:

Consideration of other model architectures: All results within this work use random forests with the same number of trees. Further credibility to the results within this work could be added by considering further models. Particularly of interest would be complex models such as deep neural networks which allow a wider range of functions to be modelled, as such the potential for underspecification is greater.

Additional explanation generation techniques: In this work we only consider SHAP explanations as an explanation model for our produced predictors. We justify this decision theoretically in Chapter 7. However, we do not discuss further explanation methods such as counterfactuals which should be investigated in this domain. Additionally, although we avoided investigation of input gradients as explanations due to a lack of model agnostic methods, they do have some advantages in this domain that warrant further research. One such advantage we discuss within this work is computational efficiency, where gradient based methods can produce explanations in far less time than equivalent feature attribution methods. One downside of many explanation methods is a lack of robustness [AMJ18b]. As we use explanations to model their respective predictors, this lack of robustness means we cannot place complete confidence within our results. Although our results suggest this effect is negligible, evaluations specifically examining this should be conducted to place more credibility in underspecification indexes.

Consideration of other metrics: To produce the local underspecification indexes we take the mean of pairwise metric results, equally to produce set underspecification indexes from local indexes, we take the mean. By taking the mean, we see no notion of distribution of variation throughout indexes, instead metrics such as Gini index could be used to measure the distribution of variation across these sets of explanations and indexes. Furthermore, due to the lack of robustness of explanations as discussed, a more effective measure of underspecification may be achieved by analysing differences between small clusters of explanations rather than individuals. Analysis of this type may account for the slight variation of explanations between otherwise equivalent predictors.

Predictor-level credibility: Our method aims to quantify the extent to which a pipeline is underspecified. We therefore may only provide our indexes as a measure of credibility of test performance evaluations for predictor produced by the given pipeline. Ideally we would be able to identify which predictors in our Rashomon set experience the least underspecification and as such are the most credible.

More thorough definitions of underspecification: An issue within this work is the fact we cannot categorically say a given pipeline is underspecified, only how likely it is. This is due to the informal definition of underspecification, specifically that we may interpret equivalence of predictors in a number of ways. We take the approach of

equivalent predictors are those that produce equivalent explanation, however there is no guarantee this is correct. Formalisation of the problem of underspecification would allow us to draw more concrete conclusions.

Bibliography

- [AAAB⁺21] Alhanoof Althnian, Duaa AlSaeed, Heyam Al-Baity, Amani Samha, Alanoud Bin Dris, Najla Alzakari, Afnan Abou Elwafa, and Heba Kurdi. Impact of dataset size on classification performance: an empirical evaluation in the medical domain. *Applied Sciences*, 11(2):796, 2021.
- [AB18] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.
- [Abd07] Hervé Abdi. The kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics*. Sage, Thousand Oaks, CA, pages 508–510, 2007.
- [ADRDS⁺20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.
- [aEW07] Imagecreator at English Wikipedia. File:correlation examples.png. https://commons.wikimedia.org/wiki/File:Correlation_examples.png, 2007. [Online; Accessed 01/12/2021].
- [AHK01] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.
- [AMJ18a] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. *Advances in neural information processing systems*, 31, 2018.
- [AMJ18b] David Alvarez-Melis and Tommi S Jaakkola. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049*, 2018.
- [AR18] Mehryar Mohri Afshin Rostamizadeh, Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2018.

- [BAL⁺19] Indranil Balki, Afsaneh Amirabadi, Jacob Levman, Anne L Martel, Ziga Emersic, Blaz Meden, Angel Garcia-Pedrero, Saul C Ramirez, Dehan Kong, Alan R Moody, et al. Sample-size determination methodologies for machine learning in medical imaging research: a systematic review. *Canadian Association of Radiologists Journal*, 70(4):344–353, 2019.
- [BBM⁺15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.
- [BG18] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018.
- [BHMM19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.
- [Bis06] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128(9), 2006.
- [BMR⁺20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [Bre01] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.
- [BSH⁺10] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11:1803–1831, 2010.
- [BW99] Damien Brain and Geoffrey I Webb. On the effect of data set size on bias and variance in classification learning. In *Proceedings of the Fourth Australian Knowledge Acquisition Workshop, University of New South Wales*, pages 117–128, 1999.
- [BWM20] Umang Bhatt, Adrian Weller, and José MF Moura. Evaluating and aggregating feature-based model explanations. *arXiv preprint arXiv:2005.00631*, 2020.
- [CC09] James Nicholson Christopher Clapham. *The Concise Oxford Dictionary of Mathematics, Fourth Edition (Oxford Paperback Reference)*. Oxford Paperback Reference. Oxford University Press, USA, 4 edition, 2009.

-
- [CCA⁺09] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision support systems*, 47(4):547–553, 2009.
- [ČK69] Eduard Čech and M Katětov. Point sets. *Academia, Publishing House of the Czechoslovak Academy of Sciences (Praha)*, 1969.
- [CLG⁺15] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1721–1730, 2015.
- [CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
- [Coc11] Dean De Cock. Ames, iowa: Alternative to the boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19(3), 2011.
- [CS08] Paulo Cortez and Alice Maria Gonçalves Silva. Using data mining to predict secondary school student performance. *Proceedings of 5th Annual Future Business Technology Conference*, 2008.
- [DG06] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.
- [DG17] Dheeru Dua and Casey Graff. UCI machine learning repository. <http://archive.ics.uci.edu/ml>, 2017. University of California, Irvine, School of Information and Computer Sciences.
- [DGBM] Butts David, Dharuman Gautham, Punch Bill, and Murillo Micheal. Numba versus c++. <https://murillogroupmsu.com/numba-versus-c/>. [Online; Accessed 21/02/22].
- [DHM⁺20] Alexander D’Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*, 2020.
- [Dod08] Yadolah Dodge. *Kendall Rank Correlation Coefficient*, pages 278–281. Springer New York, New York, NY, 2008.
- [DP06] Robert PW Duin and Elżbieta Pełkalska. Object representation, sample size, and data set complexity. In *Data complexity in pattern recognition*, pages 25–58. Springer, 2006.

- [DPG⁺14] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv preprint arXiv:1406.2572*, 2014.
- [DVK17] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.
- [DYT⁺20] Josip Djolonga, Jessica Yung, Michael Tschannen, Rob Romijnders, Lucas Beyer, Alexander Kolesnikov, Joan Puigcerver, Matthias Minderer, Alexander D’Amour, Dan Moldovan, et al. On robustness and transferability of convolutional neural networks. *arXiv preprint arXiv:2007.08558*, 2020.
- [Edu21] IBM Cloud Education. Overfitting. <https://www.ibm.com/cloud/learn/overfitting>, 2021.
- [Ell10] Paul D Ellis. *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results*. Cambridge university press, 2010.
- [Faw06] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.
- [FMG⁺20] Seth Flaxman, Swapnil Mishra, Axel Gandy, H Unwin, H Coupland, T Mellan, H Zhu, T Berah, J Eaton, P Perez Guzman, et al. Report 13: Estimating the number of infections and the impact of non-pharmaceutical interventions on covid-19 in 11 european countries. Technical report, Imperial College London, 2020.
- [FR] Scott Fortmann-Roe. Bias and variance. <http://scott.fortmann-roe.com/docs/BiasVariance.html>. [Online; Accessed 17/10/2021].
- [FRD19] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable’s importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019.
- [Fre14] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.
- [FTG13] Hadi Fanaee-T and Joao Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pages 1–15, 2013.
- [GBD92] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

-
- [GG21] Alex Gramegna and Paolo Giudici. Shap and lime: An evaluation of discriminative power in credit risk. *Frontiers in Artificial Intelligence*, page 140, 2021.
- [GJM⁺20] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *arXiv preprint arXiv:2004.07780*, 2020.
- [GP71] AA Goldstein and JF Price. On descent from local minima. *Mathematics of Computation*, 25(115):569–574, 1971.
- [Gre17] Gregory Piatetsky. Python overtakes R, becomes the leader in Data Science, Machine Learning platforms. <https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html>, 2017. [Online; Accessed 21/02/22].
- [GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [HD19] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.
- [Hea20] Will Douglas Heaven. Google’s medical ai was super accurate in a lab. real life was a different story. <https://www.technologyreview.com/2020/04/27/1000658/google-medical-ai-accurate-lab-real-life-clinic-covid-diabetes-retina-dis> Dec 2020. [Online; Accessed 11/10/2021].
- [HFL⁺21] James Hinns, Xiuyi Fan, Siyuan Liu, Veera Raghava Reddy Kovvuri, Mehmet Orcun Yalcin, and Markus Roggenbach. An initial study of machine learning underspecification using feature attribution explainable ai algorithms: A covid-19 virus transmission case study. In *Pacific Rim International Conference on Artificial Intelligence*, pages 323–335. Springer, 2021.
- [HPK11] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.
- [HWB⁺22] Anna Hedström, Leander Weber, Dilyara Bareeva, Franz Motzkus, Wojciech Samek, Sebastian Lapuschkin, and Marina M.-C. Höhne. Quantus: An explainable ai toolkit for responsible evaluation of neural network explanations. 2022.

- [Jet21] Jet Brains. The State of Developer Ecosystem 2021. <https://www.jetbrains.com/lp/devecosystem-2021/>, 2021. [Online; Accessed 21/02/22].
- [JMMG20] Alon Jacovi, Ana Marasović, Tim Miller, and Yoav Goldberg. Formalizing trust in artificial intelligence: Prerequisites, causes and goals of human trust in ai. *arXiv preprint arXiv:2010.07487*, 2020.
- [KED⁺21] Marcin Kapcia, Hassan Eshkiki, Jamie Duell, Xiuyi Fan, Shangming Zhou, and Benjamin Mora. Exmed: An ai tool for experimenting explainable ai techniques on medical data analytics. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 841–845. IEEE, 2021.
- [Kha16] KhanAcademy. Distance formula. <https://www.khanacademy.org/math/geometry/hs-geo-analytic-geometry/hs-geo-distance-and-midpoints/a/distance-formula>, 2016. Online; Accessed: 01/12/2021.
- [Kir08] Wilhelm Kirch, editor. *Pearson’s Correlation Coefficient*, pages 1090–1091. Springer Netherlands, Dordrecht, 2008.
- [KMR16] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*, 2016.
- [LEC⁺20] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67, 2020.
- [LL17] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.
- [LLRB16] Benjamin Letham, Portia A Letham, Cynthia Rudin, and Edward P Browne. Prediction uncertainty and optimal experimental design for learning dynamical systems. *Chaos: An Interdisciplinary Journal of Non-linear Science*, 26(6):063110, 2016.
- [LNV⁺18] Scott M Lundberg, Bala Nair, Monica S Vavilala, Mayumi Horibe, Michael J Eisses, Trevor Adams, David E Liston, Daniel King-Wai Low, Shu-Fang Newman, Jerry Kim, et al. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature Biomedical Engineering*, 2(10):749, 2018.
- [Lom06] Tania Lombrozo. The structure and function of explanations. *Trends in cognitive sciences*, 10(10):464–470, 2006.

-
- [MAD19] David Madras, James Atwood, and Alex D’Amour. Detecting extrapolation with local ensembles. *arXiv preprint arXiv:1910.09573*, 2019.
- [MCR14] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.
- [MCU20] Charles Marx, Flavio Calmon, and Berk Ustun. Predictive multiplicity in classification. In *International Conference on Machine Learning*, pages 6765–6774. PMLR, 2020.
- [Mil19] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.
- [Mol19] Christoph Molnar. *Interpretable Machine Learning*. self-published, 2019. <https://christophm.github.io/interpretable-ml-book/>.
- [MSK⁺19] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592*, 2019.
- [MSM18] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.
- [MW] Merriam-Webster. Metric. <https://www.merriam-webster.com/dictionary/metric>. [Online; Accessed: 12/09/2021].
- [NM20] An-phi Nguyen and María Rodríguez Martínez. On quantitative aspects of model interpretability. *arXiv preprint arXiv:2007.07584*, 2020.
- [Pap22] Papers With Code. Trends. <https://paperswithcode.com/trends>, 2022. [Online; Accessed 21/02/22].
- [PKS15] Joseph Prusa, Taghi M Khoshgoftaar, and Naeem Seliya. The effect of dataset size on training tweet sentiment classifiers. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 96–102. IEEE, 2015.
- [PSM⁺07] Sang Cheol Park, Rahul Sukthankar, Lily Mummert, Mahadev Satyanarayanan, and Bin Zheng. Optimization of reference library used in content-based medical image retrieval scheme. *Medical Physics*, 34(11):4331–4339, 2007.
- [QCSSL09] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

- [Rat19] Shubham Rathi. Generating counterfactual and contrastive explanations using shap. *arXiv preprint arXiv:1906.09293*, 2019.
- [RH20] Laura Rieger and Lars Kai Hansen. Irof: a low resource evaluation metric for explanation methods. *arXiv preprint arXiv:2003.08747*, 2020.
- [RHDV17] Andrew Slavin Ross, Michael C Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. *arXiv preprint arXiv:1703.03717*, 2017.
- [RN09] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2009.
- [Rot88] Alvin E Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.
- [RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.
- [Sal18] Isha Salian. Supervize me: What’s the difference between supervised, unsupervised, semi-supervised and reinforcement learning? <https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/>, 2018.
- [Sam59] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3), 1959.
- [Sha16] Lloyd S Shapley. *A value for n-person games*. Princeton University Press, 2016.
- [SK19] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.
- [SRP19] Lesia Semenova, Cynthia Rudin, and Ronald Parr. A study in rashomon curves and volumes: A new perspective on generalization and model simplicity in machine learning. *arXiv preprint arXiv:1908.01755*, 2019.
- [SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.
- [Sta21] Stack Overflow. Stack Overflow Developer Survey 2021, 2021.
- [SZ05] Margarita Sordo and Qing Zeng. On sample size and classification accuracy: A performance comparison. In *International Symposium on Biological and Medical Data Analysis*, pages 193–201. Springer, 2005.
- [Tam21] Michael Tamir. What is machine learning? <https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/>, Accessed: 2021.

-
- [TIO22] TIOBE. TIOBE Index for March 2022. <https://www.tiobe.com/tiobe-index/>, 2022. [Online; Accessed 15/03/22].
- [TPM17] Alex Tellez, Max Pumperla, and Michal Malohlava. *Mastering Machine Learning with Spark 2.x*. O’Reilly, 2017.
- [VMC⁺21] Erik Volz, Swapnil Mishra, Meera Chand, Jeffrey C Barrett, Robert Johnson, Lily Geidelberg, Wes R Hinsley, Daniel J Laydon, Gavin Dabrera, Áine O’Toole, et al. Transmission of sars-cov-2 lineage b. 1.1. 7 in england: Insights from linking epidemiological and genetic data. *MedRxiv*, pages 2020–12, 2021.
- [WFHP16] Ian Witten, Eibe Frank, Mark Hall, and Christopher Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 4th edition, 2016.
- [WL20] Wei Wei and James Landay. Ml interpretability and intrinsic models. <https://hci.stanford.edu/courses/cs335/2020/sp/lec3.pdf>, 2020.
- [WLB⁺20] Joseph T Wu, Kathy Leung, Mary Bushman, Nishant Kishore, Rene Niehus, Pablo M de Salazar, Benjamin J Cowling, Marc Lipsitch, and Gabriel M Leung. Estimating clinical severity of covid-19 from the transmission dynamics in wuhan, china. *Nature Medicine*, pages 1–5, 2020.
- [WVP18] Christina Wadsworth, Francesca Vera, and Chris Piech. Achieving fairness through adversarial learning: an application to recidivism prediction. *arXiv preprint arXiv:1807.00199*, 2018.
- [YF21] Mehmet Orcun Yalcin and Xiuyi Fan. On Evaluating Correctness of Explainable AI Algorithms: an Empirical Study on Local Explanations for Classification. 2021.
- [YHS⁺19] Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Suggala, David I Inouye, and Pradeep K Ravikumar. On the (in) fidelity and sensitivity of explanations. *Advances in Neural Information Processing Systems*, 32, 2019.

Appendix A

Classification Results

In this chapter we include further charts from our classification results presented in section 7.4.

These figures include a chart representing the variation of underspecification indexes generated from each of the four metrics we experiment with (as outlined in Chapter 4 and 6) against prediction accuracy for different lengths of their respective dataset.

Each result shown within this chapter is generated from a single experiment and so is susceptible to the random variation highlighted in Chapter 7.

Figures continue on the next page...

A. Classification Results

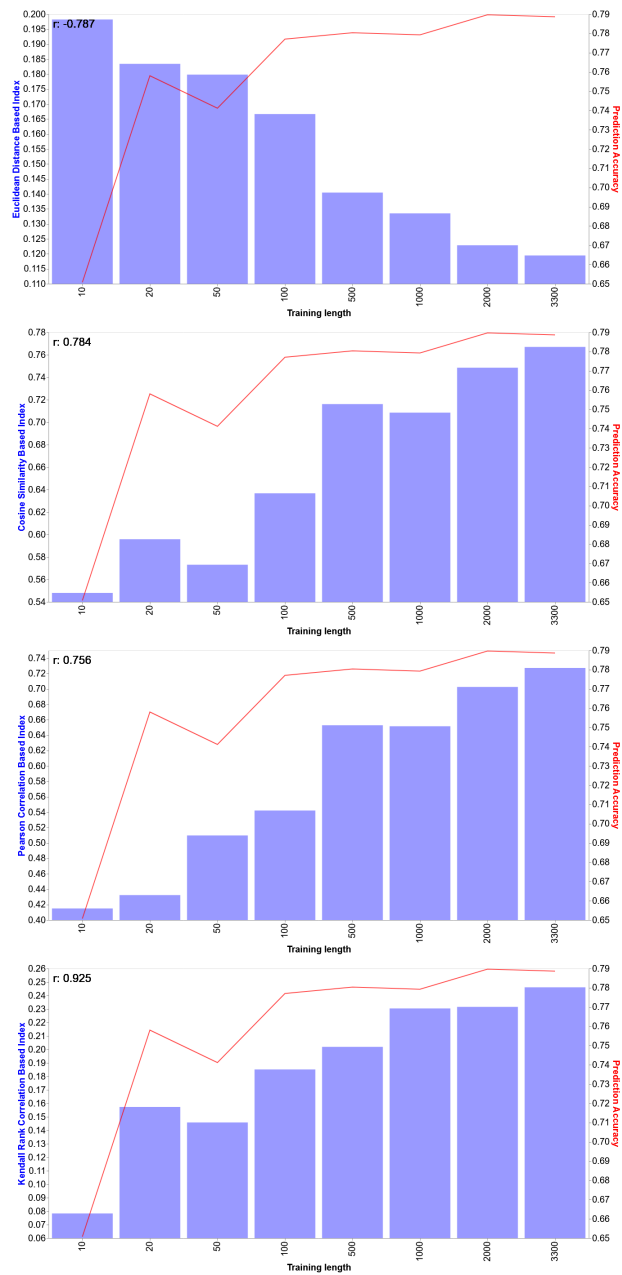


Figure A.0.1: Each variation of underspecification index against prediction accuracy for different strata of the Abalone dataset.

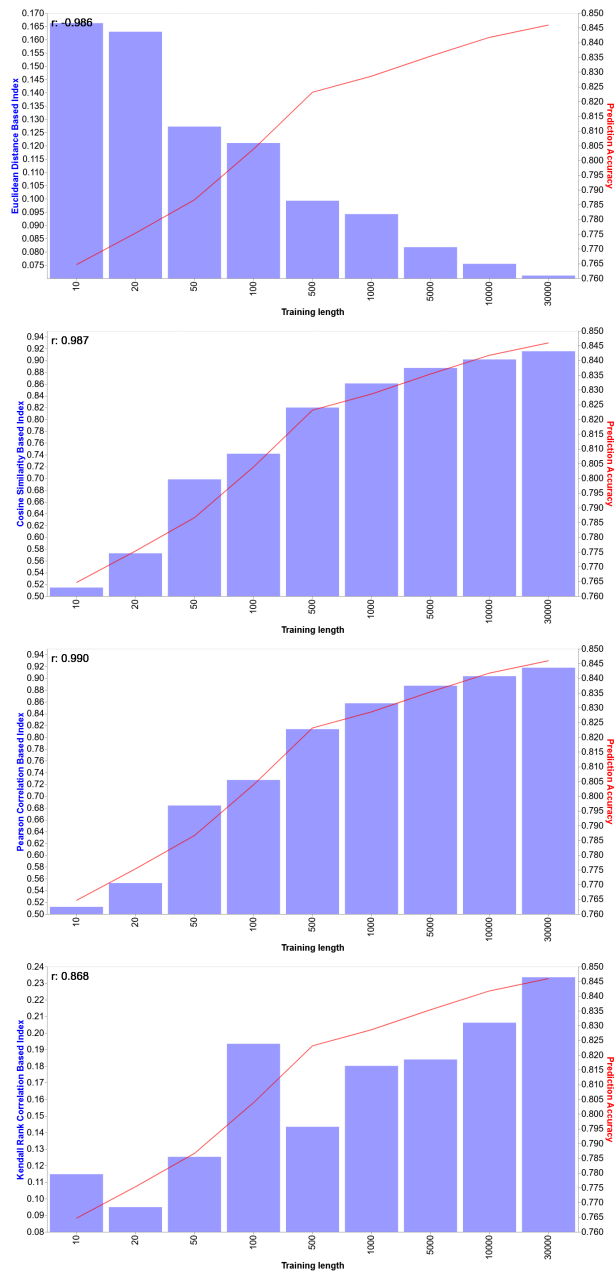


Figure A.0.2: Each variation of underspecification index against prediction accuracy for different strata of the Adult dataset.

A. Classification Results

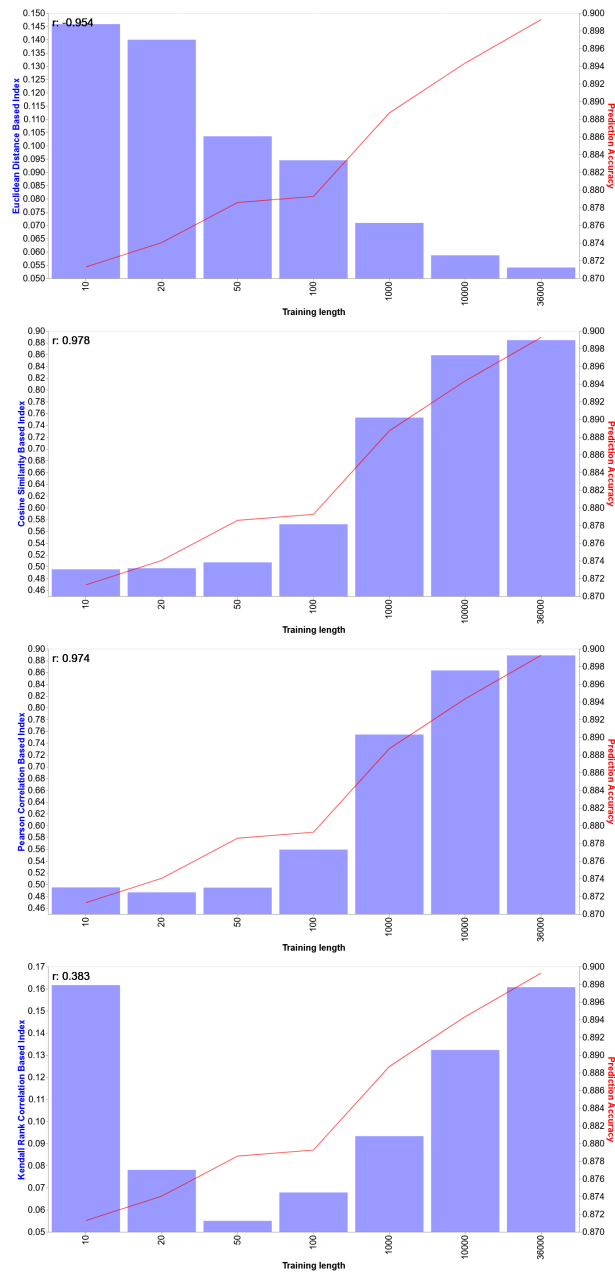


Figure A.0.3: Each variation of underspecification index against prediction accuracy for different strata of the Bank dataset.

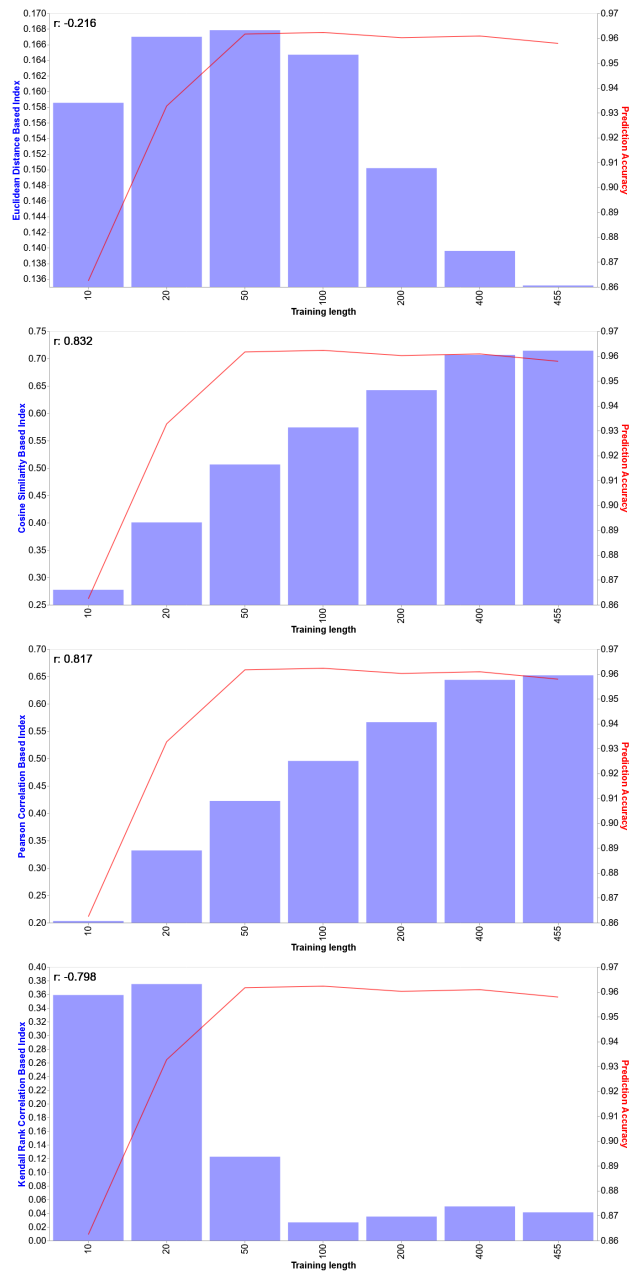


Figure A.0.4: Each variation of underspecification index against prediction accuracy for different strata of the Breast Cancer dataset.

A. Classification Results

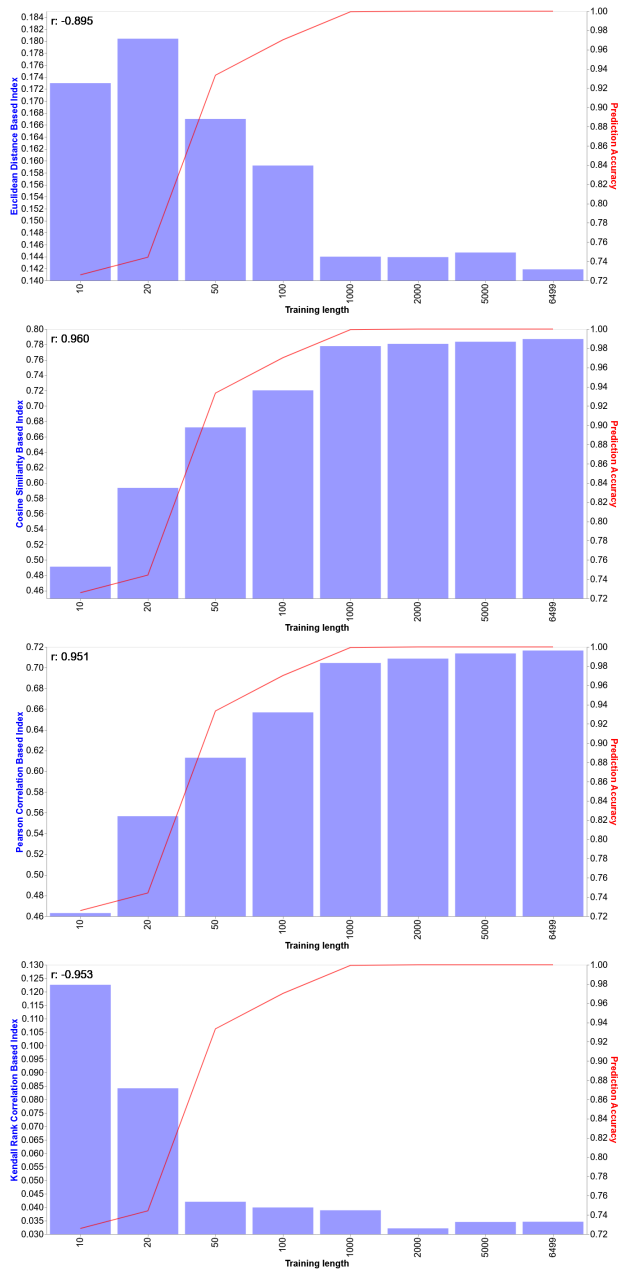


Figure A.0.5: Each variation of underspecification index against prediction accuracy for different strata of the Mushroom dataset.

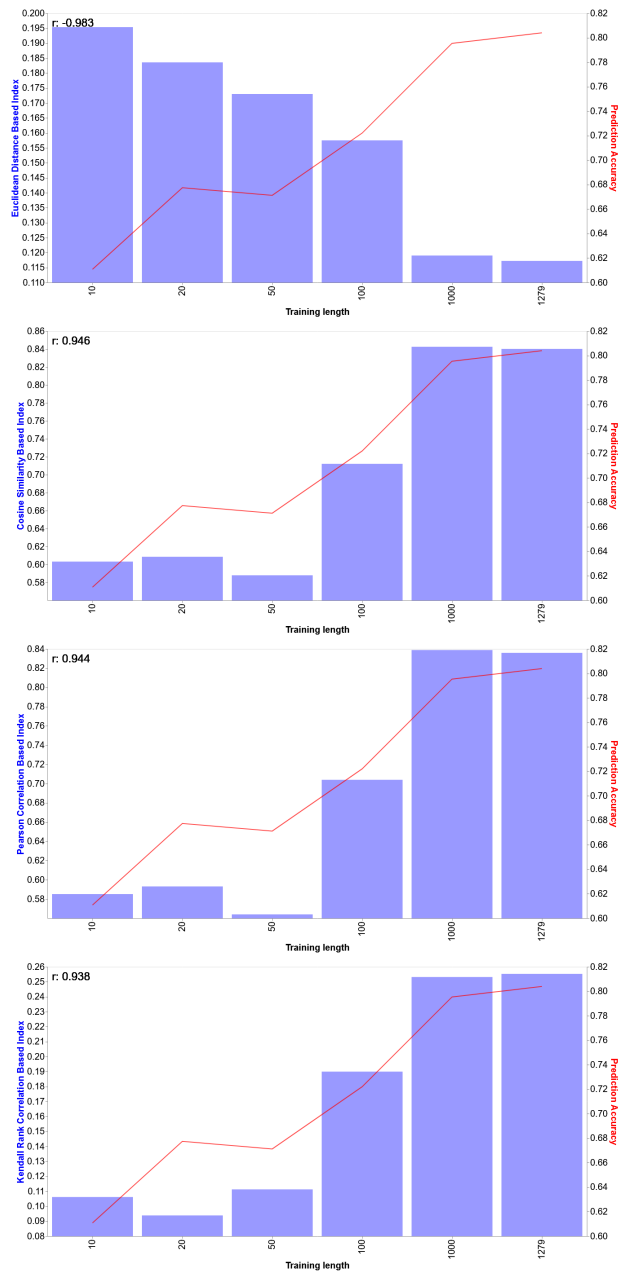


Figure A.0.6: Each variation of underspecification index against prediction accuracy for different strata of the Wine Quality dataset.

Appendix B

Regression Results

In this chapter we include further charts from our regression results presented in section 8.3.

We show two types of charts, the first, presented in section B.1, shows the relationship between the cosine based underspecification index and prediction variance for each strata of dataset length for each dataset.

The second, presented in section B.2, shows the relationship between prediction accuracy and each variation of underspecification index calculated using a different metric (as outlined in Chapter 4 and 6) for varying length of the respective dataset.

Each result shown within this chapter is generated from a single experiment and so is susceptible to the random variation highlighted in Chapter 8.

Figures continue on the next page...

B.1 Cosine Index Against Variance

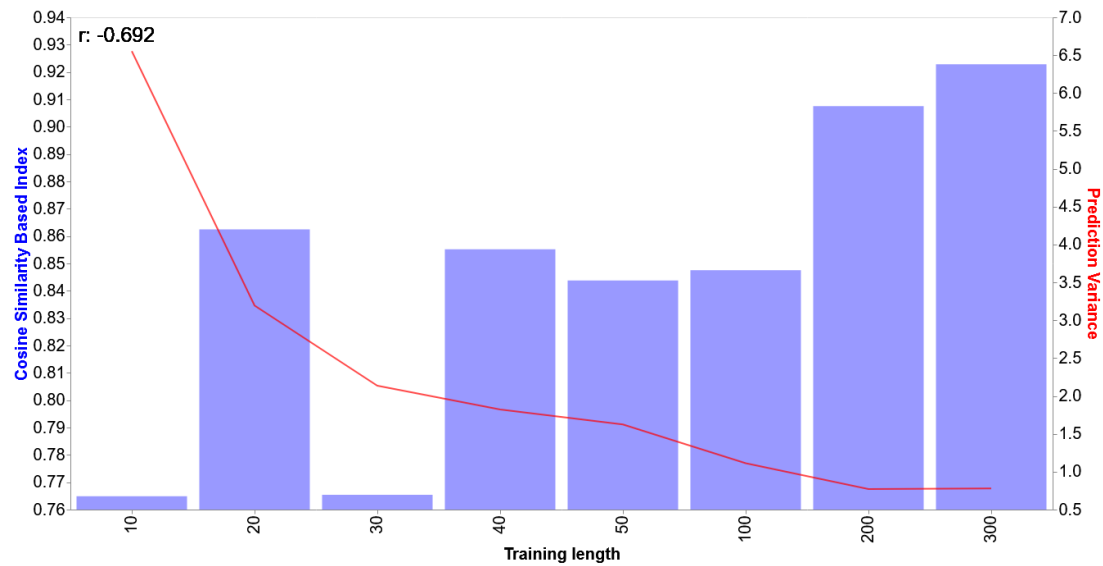


Figure B.1.1: The Cosine based set underspecification index and prediction variance for the Auto MPG dataset for different strata.

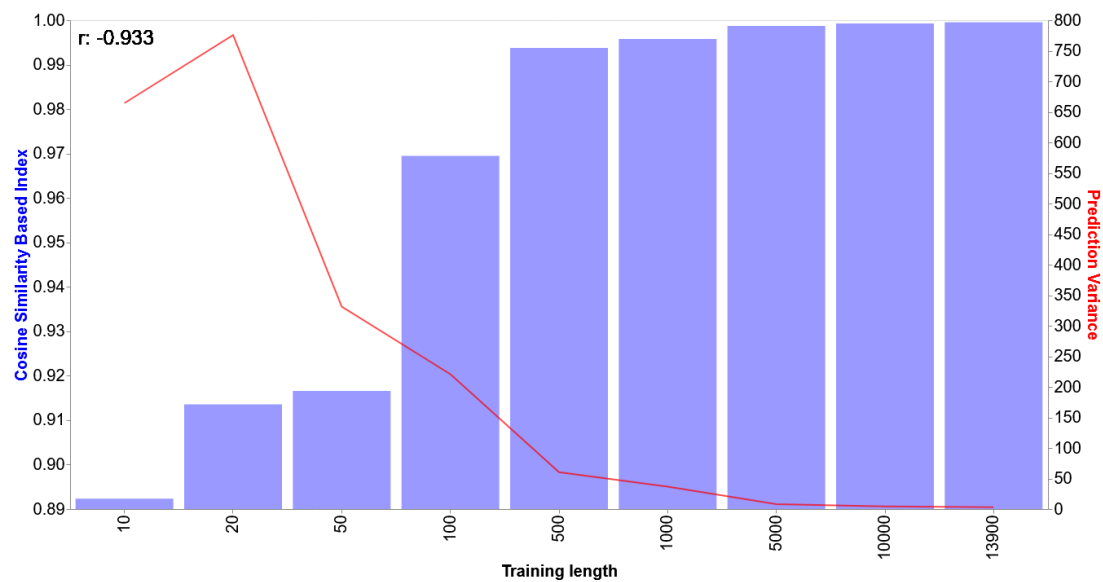


Figure B.1.2: The Cosine based set underspecification index and prediction variance for the Bike Share dataset for different strata.

B.1. Cosine Index Against Variance

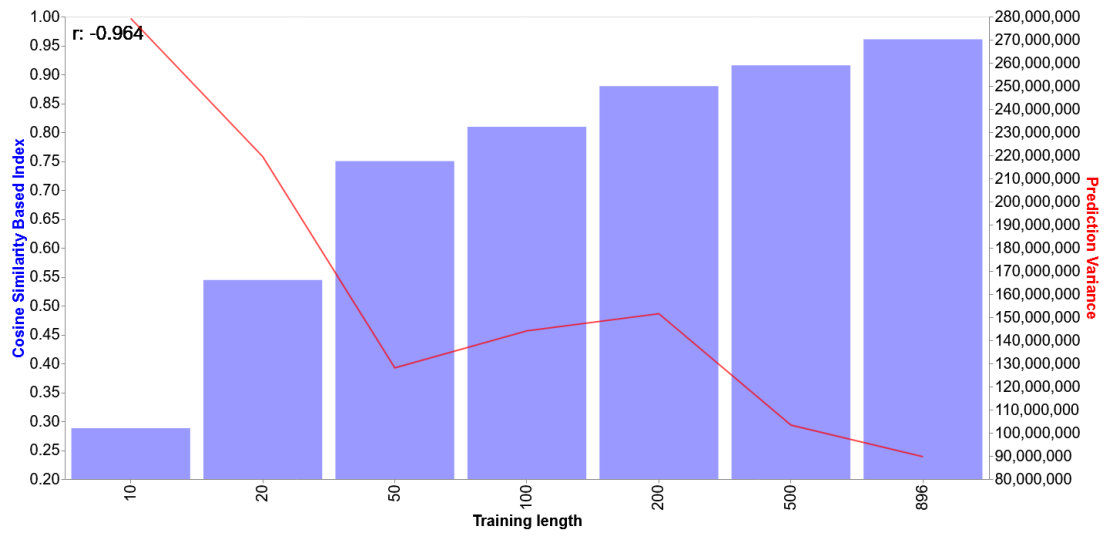


Figure B.1.3: The Cosine based set underspecification index and prediction variance for the House Price dataset for different strata.

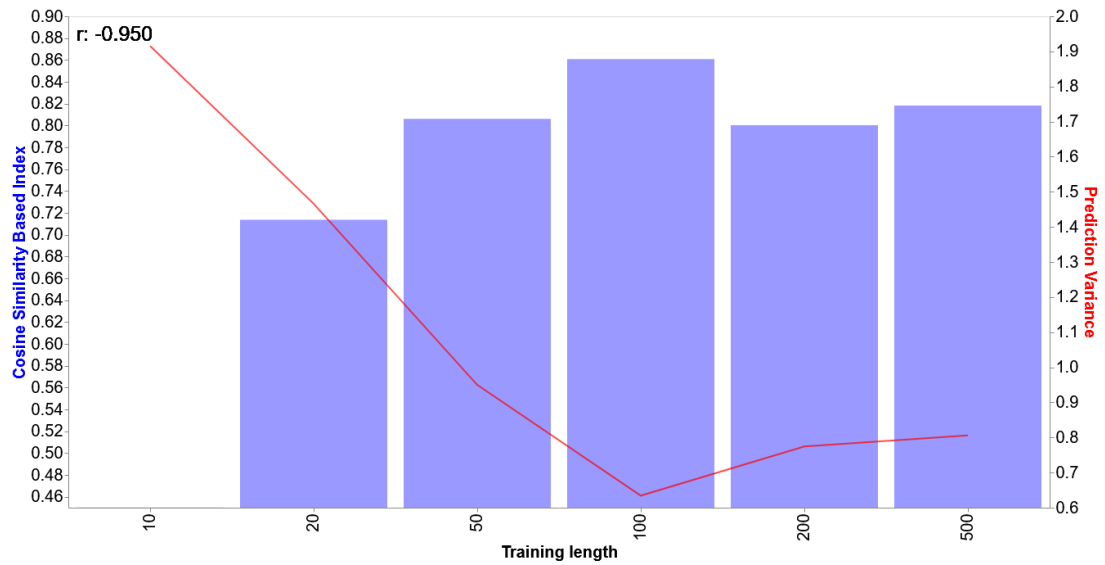


Figure B.1.4: The Cosine based set underspecification index and prediction variance for the Student dataset for different strata.

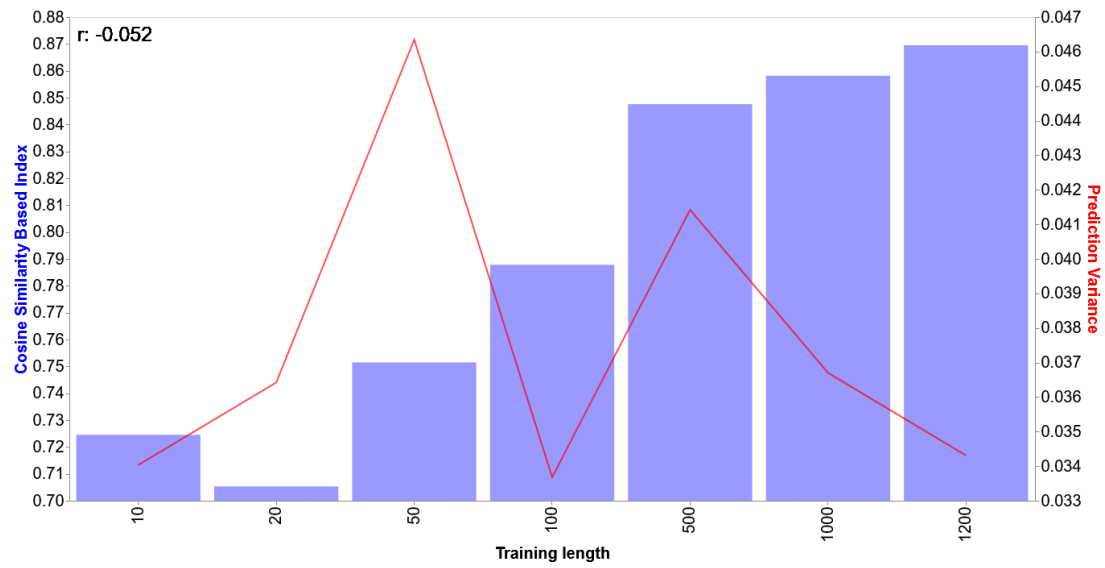


Figure B.1.5: The Cosine based set underspecification index and prediction variance for the Wine Quality dataset for different strata.

B.2 Underspecification Index Variations against Accuracy

Figures continue on the next page...

B.2. Underspecification Index Variations against Accuracy

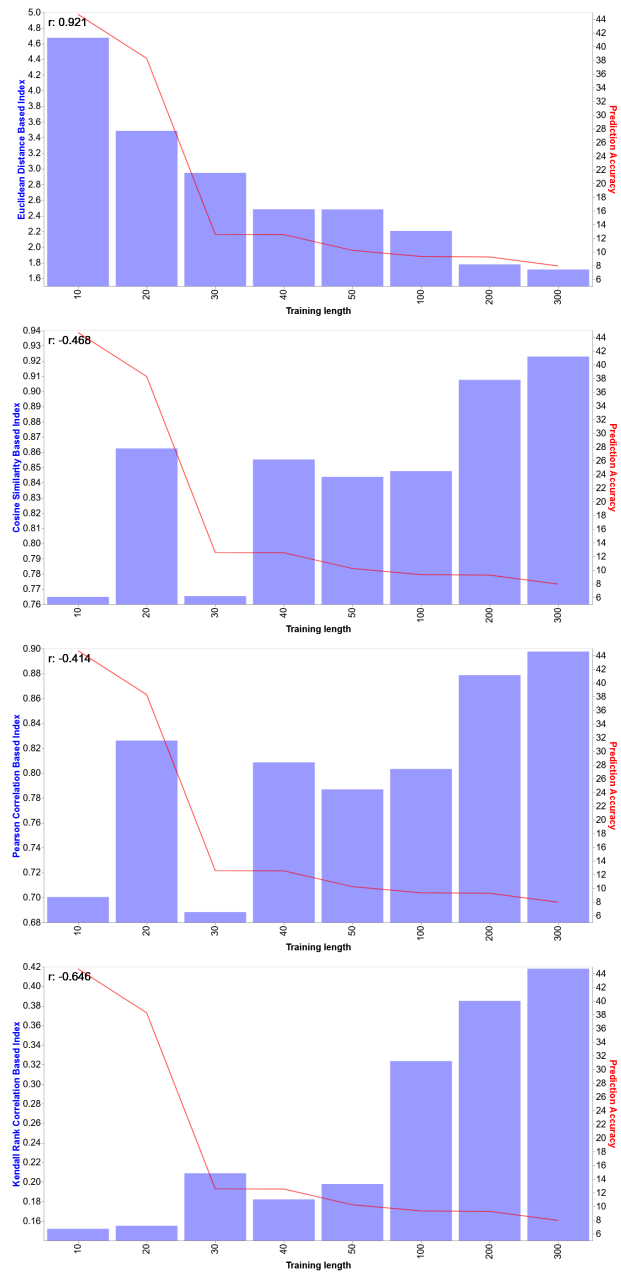


Figure B.2.1: Each variation of underspecification index against prediction mean squared error for different strata of the Auto MPG dataset.

B. Regression Results

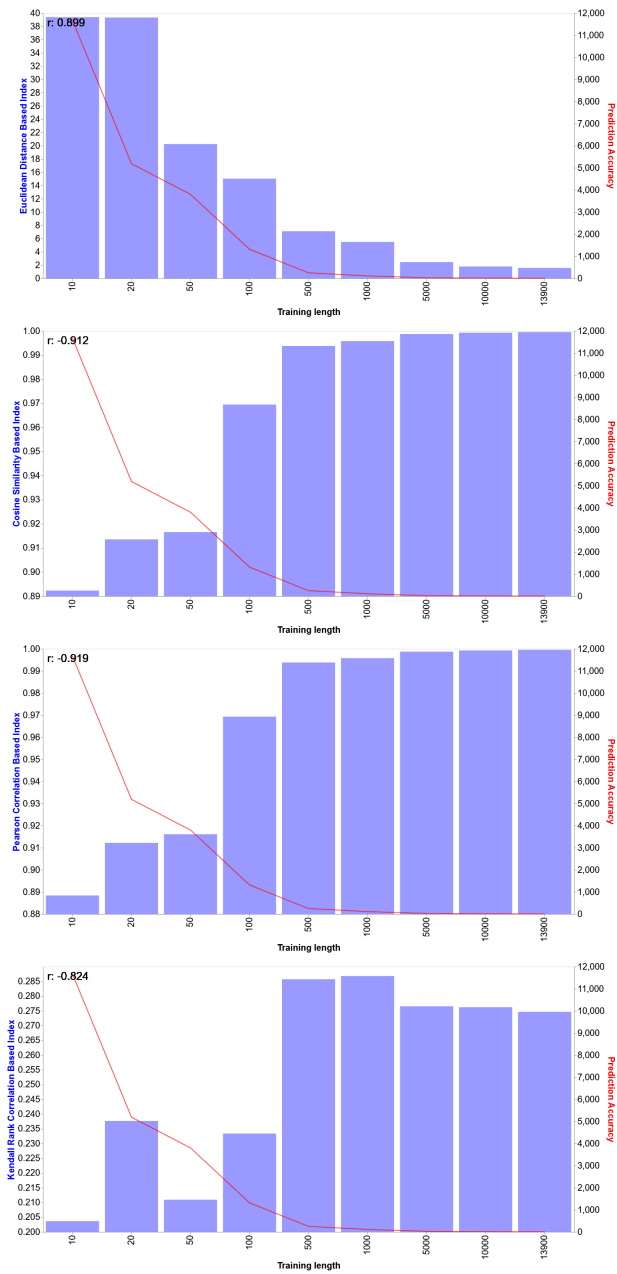


Figure B.2.2: Each variation of underspecification index against prediction mean squared error for different strata of the Bike Share dataset.

B.2. Underspecification Index Variations against Accuracy

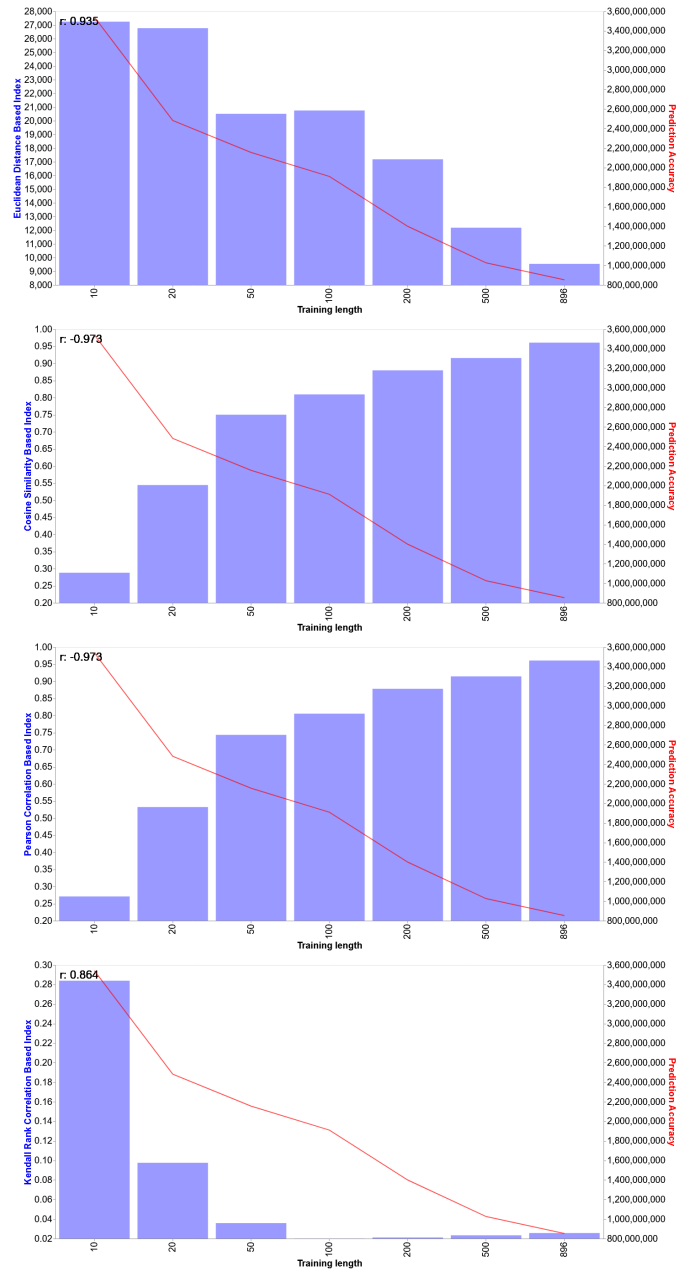


Figure B.2.3: Each variation of underspecification index against prediction mean squared error for different strata of the House Price dataset.

B. Regression Results

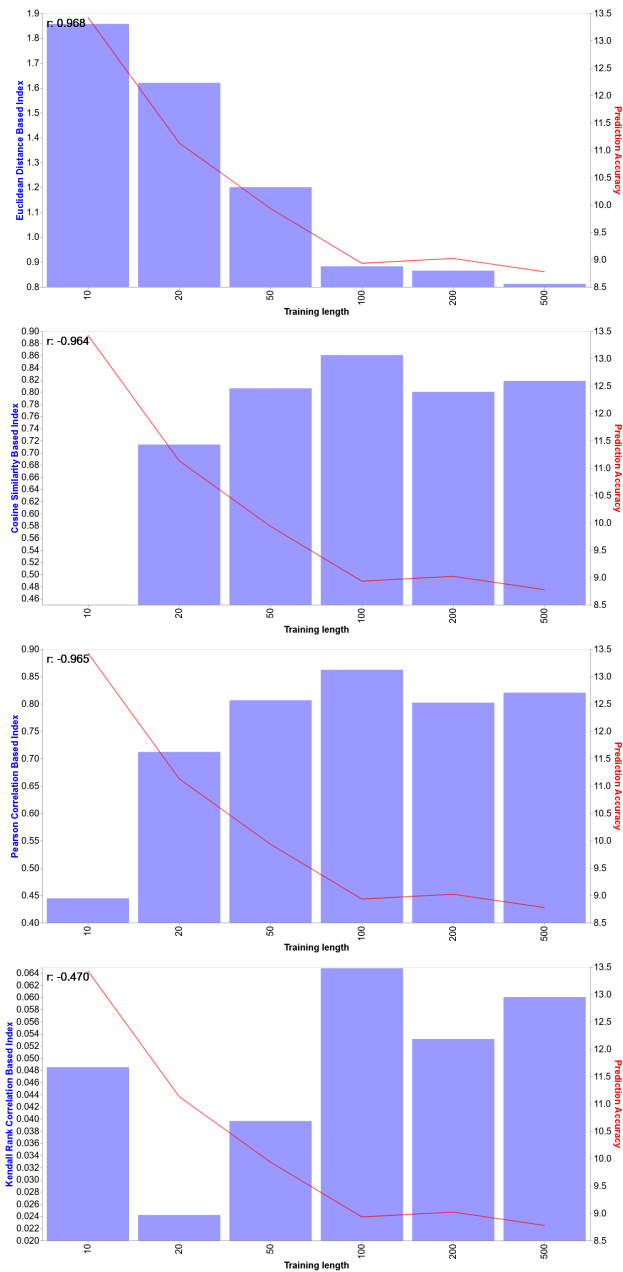


Figure B.2.4: Each variation of underspecification index against prediction mean squared error for different strata of the Student dataset.

B.2. Underspecification Index Variations against Accuracy

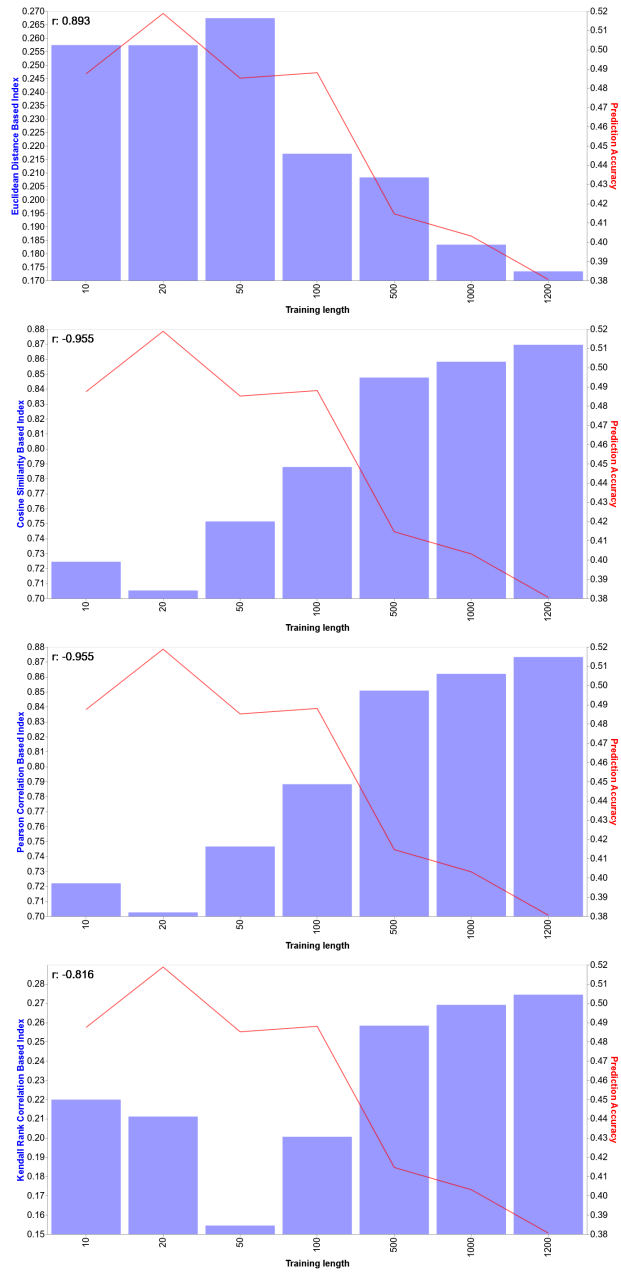


Figure B.2.5: Each variation of underspecification index against prediction mean squared error for different strata of the Wine Quality dataset.