Universiteit Hasselt

Masterproef voorgedragen tot het behalen van de graad van master in de informatica

Many Minds, One Decision: Leveraging Multiple AI Models for Informed Decision-Making

Author:

Gilles Eerlings

Promotor:

prof. dr. Kris Luyten

Co-promotor:

prof. dr. Jori Liesenborgs

Supervisor:

prof. dr. Gustavo Rovelo Ruiz

Academic year 2023-2024

Acknowledgements

First and foremost, I want to thank Prof. Dr. Kris Luyten for their invaluable support, insightful advice, and interesting ideas. I am grateful for the opportunity to work on this fascinating topic and to expand my knowledge of machine learning under your guidance. I also want to thank Prof. Dr. Jori Liesenborgs for the support regarding machine learning and working with computationally intensive tasks on the VSC. Additionally, I want to thank Prof. Dr. Gustavo Rovelo Ruiz for the additional guidance regarding the visualization of the end results. Finally, I want to thank my close friends and parents for their mental support and encouragement when progress was made..

Samenvatting

Kunstmatige intelligentie (AI) speelt een cruciale rol in gebieden zoals gezondheidszorg, financiën en justitie, waar machine learning (ML) modellen privégegevens analyseren om significante voorspellingen te doen. Echter, biases in ML modellen kunnen leiden tot onrechtvaardige uitkomsten, wat zorgen oproept over algoritmische transparantie en verantwoording. Complexe AI modellen, vaak aangeduid als black-box modellen, nemen beslissingen zonder duidelijke uitleg, waardoor gebruikers deze systemen overmatig vertrouwen of juist wantrouwen. Dit is vooral problematisch voor niet-experts, aangezien overmatig vertrouwen kan leiden tot gevaarlijke fouten terwijl wantrouwen de adoptie van AI kan belemmeren.

Om deze problemen aan te pakken, is er een streven naar algoritmische verantwoording en het "recht op uitlegbaarheid" nodig, met regelgeving zoals de GDPR van de EU die uitlegbare AI (XAI) bevordert. XAI streeft ernaar AI systemen transparanter te maken, wat zowel gebruikers als ontwikkelaars ten goede komt door foutdetectie en continue verbetering mogelijk te maken. Het creëren van uitlegbare AI modellen, met name deep neural networks (DNNs), blijft echter een uitdaging vanwege hun complexiteit. Hoewel er vaak een afweging is tussen nauwkeurigheid en uitlegbaarheid, suggereert recent onderzoek dat uitlegbare modellen een nauwkeurigheid kunnen bereiken die vergelijkbaar is met die van black-box modellen.

Deze scriptie onderzoekt de uitlegbaarheid van DNNs door veranderingen in hun data en architectuur te vergelijken. Door talloze licht verschillende modellen te trainen, beoogt het onderzoek verschillen in hun voorspellingen te koppelen aan veranderingen in hun interne logica. De studie richt zich op neurale netwerken die zijn getraind op de MNIST dataset van handgeschreven cijfers, gekozen vanwege de eenvoud en relevantie voor classificatietaken. Deze scriptie is onderverdeeld in meerdere hoofdstukken, die elk afzonderlijk worden besproken.

Het eerste hoofdstuk verdiept zich in het concept van model multiplicity, met een primaire focus op classificatiemodellen. Het begint met het definiëren van model multiplicity door de lens van het Rashomon effect, waarbij meerdere modellen dezelfde nauwkeurigheid bereiken maar verschillen in hun interne werking of besluitvormingsgrenzen. Dit fenomeen wordt procedurele multiplicity genoemd wanneer modellen intern verschillen maar dezelfde uitkomsten produceren, en predictieve multiplicity wanneer modellen verschillende voorspellingen opleveren ondanks vergelijkbare nauwkeurigheid.

Het hoofdstuk bespreekt de implicaties van model multiplicity op model evaluatie, waarbij wordt betoogd dat nauwkeurigheid alleen onvoldoende is voor het selecteren van het beste model. In plaats daarvan zouden andere wenselijke eigenschappen zoals uitlegbaarheid, eerlijkheid en robuustheid de modelselectie moeten leiden. Het hoofdstuk benadrukt ook het belang van juridische en ethische overwegingen, vooral in risicovolle gebieden zoals recidivevoorspelling, waar eerlijkheid van groot belang is.

Het concept van underspecification wordt geïntroduceerd, waarbij de noodzaak wordt benadrukt van een duidelijke specificatie van wenselijke eigenschappen om problemen zoals proxy discriminatie te voorkomen. Daarnaast wordt de uitdaging van rechtvaardiging aangepakt, waarbij de logica achter het kiezen van het ene model boven het andere in vraag wordt gesteld wanneer meerdere modellen even nauwkeurige maar uiteenlopende voorspellingen produceren.

In het volgende hoofdstuk wordt onze aanpak voor het genereren van een diverse set modellen op basis van de MNIST dataset gedetailleerd beschreven, met de focus op het bereiken van "model multiplicity", waarbij modellen een vergelijkbare nauwkeurigheid hebben maar verschillen in besluitvormingsprocessen. Dit omvat het gebruik van multi-class classifiers en het creëren van variaties door veranderingen in trainingsdata en machine learning algoritmeparameters.

Om dit te vergemakkelijken, hebben we een softwarearchitectuur ontwikkeld die de generatie van talloze modellen met verschillende eigenschappen automatiseert, waardoor de bias en tijdsbeperkingen van handmatige modelcreatie worden vermeden. De architectuur bestaat uit verschillende componenten, waaronder een variatiegenerator, dataloader, dataprocessor, modeltrainer en logger. Deze componenten werken samen om willekeurige combinaties van data en modelvariaties te genereren, datasets te laden, variaties toe te passen, modellen te trainen en alle relevante details vast te leggen. Een overzicht van de softwarearchitectuur wordt getoond in figuur 1.

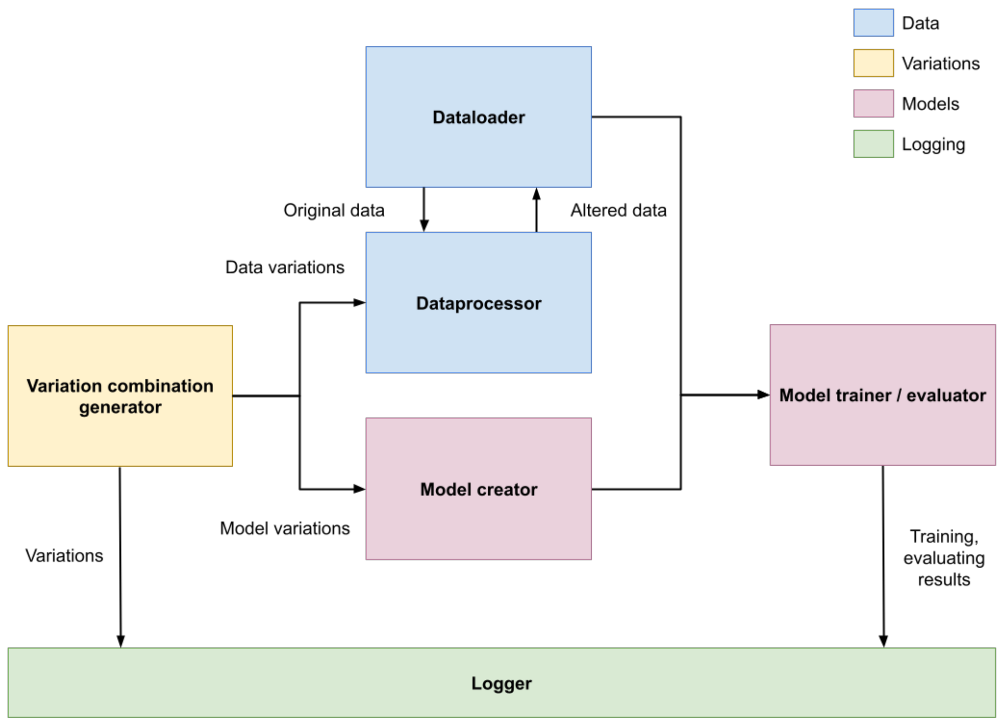


Figure 1: Een overzicht van de verschillende componenten en hun interacties binnen de softwarearchitectuur. Het proces begint met de willekeurige generatie van combinaties van data- en modelvariaties. Deze variaties worden vervolgens verdeeld: datavariaties worden toegepast op de originele data in de dataprocessor, terwijl modelvariaties worden gebruikt om het model te creëren. De modeltrainer gebruikt de verwerkte data om het model te trainen en evalueert de resultaten op de testset. Een logger houdt de gegenereerde variaties bij en registreert de trainings- en evaluatieresultaten.

De variatiegenerator creëert twee soorten variaties: datavariaties (bijv. kleurinversie, contrastaanpassingen) en modelvariaties (bijv. batchgrootte, optimizer keuze). We gebruiken equivalence partitioning om een uitgebreide dekking van parameterwaarden te waarborgen, waarbij problematische combinaties worden vermeden door middel van beperkingen.

Datavariaties worden toegepast met behulp van twee strategieën: in-place en appending. In-place wijzigt bestaande data, waardoor gelaagde augmentaties mogelijk zijn, terwijl appending gewijzigde data toevoegt aan een kopie van de originele dataset, waardoor de integriteit ervan behouden blijft. Outlier detectie, door middel van isolation forests, identificeert anomalieën binnen elk cijferlabel, wat resulteert in een gevarieerde dataset die de robuustheid van het model verbetert.

De architectuur ondersteunt verschillende data-augmentaties, zoals kleurinversie, contrastaanpassingen, rotaties en translatie. Deze augmentaties worden in een specifieke volgorde toegepast om inefficiënties te vermijden en reproduceerbaarheid te waarborgen. Daarnaast verkennen we de effecten van het al dan niet opnemen van outliers en het aanpassen van de hoeveelheid geaugmenteerde data.

Modelvariaties omvatten het aanpassen van hyperparameters zoals het aantal verborgen lagen, trainingsepochen, batchgrootte, dropout lagen, optimizer types en activatiefuncties. We introduceren ook het gebruik van validatiesets om overfitting te voorkomen.

Door deze parameters systematisch te variëren en een gestructureerde experimentele aanpak te gebruiken, genereren we een diverse set modellen, elk met unieke besluitvormingsprocessen, die bijdragen aan ons onderzoek naar model multiplicity.

In het daaropvolgende hoofdstuk onderzoeken we de gegenereerde modellen om degenen te identificeren die multiplicity vertonen. Dit omvat het evalueren van de prestaties van elk model met behulp van een testset en het groeperen van vergelijkbare modellen in Rashomon sets. Aanvankelijk worden de modellen beoordeeld op basis van hun nauwkeurigheid op afzonderlijk gemaakte handgeschreven cijfermonsters, waarbij modellen met slechte prestaties of laag vertrouwen worden gefilterd. Modellen die deze evaluatie doorstaan, worden gegroepeerd op basis van hun voorspellingsovereenkomsten voor elk testmonster.

Het hoofdstuk introduceert het concept van frequent itemsets om sets van modellen te identificeren die vaak dezelfde labels voorspellen. Deze aanpak is vergelijkbaar met het Market-Basket model dat in data mining wordt gebruikt, waarbij modellen worden behandeld als items en predictiegroepen als baskets. We passen het A-Priori algoritme toe om frequent itemsets te bepalen, met een hoge supportdrempel om ervoor te zorgen dat alleen zeer vergelijkbare modellen worden overwogen. Dit resulteert in de identificatie van paren, triplets en grotere sets modellen met vergelijkbaar voorspellend gedrag.

Het hoofdstuk verkent ook de kenmerken van deze frequent itemsets door de variaties in hun trainingsdata en hyperparameters te analyseren. Met behulp van metrics zoals Manhattan en Cosine afstanden meten we de overeenkomst tussen modellen binnen elke itemset. De resultaten tonen aan dat modellen in frequent itemsets niet alleen vergelijkbaar zijn in hun voorspellingen, maar ook in de variaties die ze ondergingen tijdens de training, wat suggereert dat het mogelijk is Rashomon sets te genereren door modellen te creëren met vergelijkbare variaties.

In het laatste hoofdstuk richten we ons op het uitleggen van model multiplicity. In gebruikersgerichte AI toepassingen kan het vertrouwen op een enkel model leiden tot problemen van overmatig of onvoldoende vertrouwen. Het introduceren van multiplicitous modellen, die een vergelijkbare voorspellende nauwkeurigheid hebben, biedt een oplossing door gebruikers in staat te stellen een set modellen, bekend als een Rashomon set, te raadplegen in plaats van een enkel model. Deze collectieve aanpak kan de besluitvorming in kritieke systemen zoals medische of juridische adviesplatforms verbeteren.

Om de verschillen in voorspellingen tussen modellen in een Rashomon set uit te leggen, visualiseren we hun voorspellende gedrag en analyseren we de variaties in hun hyperparameters en gegevenssamenstellingen. Bijvoorbeeld, we onderzochten drie modellen uit een Rashomon set die verschillende niveaus van vertrouwen toonden bij het voorspellen van een monster gelabeld als "5." Analyse toonde aan dat verschillen in data-augmentaties, zoals horizontale translatie en contrastaanpassingen, hun niveaus van vertrouwen significant beïnvloedden.

Naast de gegevenssamenstelling spelen model hyperparameters ook een cruciale rol, hoewel het interpreteren van hun impact deskundige kennis van neurale netwerken vereist. Door deze verschillen te begrijpen, kunnen gebruikers biases toewijzen aan bepaalde modellen op basis van de specifieke kenmerken van het monster dat ze evalueren.

Summary

Artificial intelligence (AI) plays a crucial role in areas such as healthcare, finance, and justice, where machine learning (ML) models analyze private data to make significant predictions. However, biases in ML models can lead to unjust outcomes, highlighting concerns about algorithmic transparency and accountability. Complex AI models, often referred to as black-box models, make decisions without clear explanations, causing users to overtrust or undertrust these systems. This is particularly problematic for non-experts, as overtrust can lead to dangerous errors while undertrust can impede AI adoption.

To address these issues, there is a push for algorithmic accountability and the "right to explainability", with regulations like the EU's GDPR promoting explainable AI (XAI). XAI aims to make AI systems more transparent, benefiting both users and developers by enabling error detection and continuous improvement. However, creating interpretable AI models, especially deep neural networks (DNNs), remains challenging due to their complexity. While there is often a trade-off between accuracy and interpretability, recent research suggests that interpretable models can achieve accuracy comparable to that of black-box models.

This thesis investigates the explainability of DNNs by comparing changes in their data and architecture. By training numerous slightly varied models, the research aims to link differences in their predictions to changes in their internal logic. The study focuses on neural networks trained on the MNIST dataset of handwritten digits, chosen for its simplicity and relevance to classification tasks. This thesis is divided into multiple chapters, each individually discussed below.

The first chapter delves into the concept of model multiplicity, primarily focusing on classification models. It begins by defining model multiplicity through the lens of the Rashomon effect, where multiple models achieve the same accuracy but differ in their internal workings or decision boundaries. This phenomenon is termed procedural multiplicity when models differ internally but produce the same outcomes, and predictive multiplicity when models yield different predictions despite similar accuracy.

The chapter discusses the implications of model multiplicity on model evaluation, arguing that accuracy alone is insufficient for selecting the best model. Instead, other desirable properties such as interpretability, fairness, and robustness should guide model selection. The chapter also underscores the importance of legal and ethical considerations, especially in high-stakes areas like recidivism prediction, where fairness is paramount.

The concept of underspecification is introduced, emphasizing the need for clear specification of desirable properties to avoid issues like proxy discrimination. Additionally, the challenge of justification is addressed, questioning the rationale behind choosing one model over another when multiple models produce equally accurate but divergent predictions.

In the following chapter, we detail our approach to generating a diverse set of models based on the MNIST dataset, focusing on achieving "model multiplicity", where models have similar accuracy but differ in decision-making processes. This involves using multi-class classifiers and creating variations through changes in training data and machine learning algorithm parameters.

To facilitate this, we developed a software architecture that automates the generation of numerous models with varying properties, avoiding the bias and time constraints of manual model creation. The architecture comprises several components, including a variation generator, data loader, data processor, model trainer, and logger. These components work together to generate random combinations of data and model variations, load datasets, apply variations, train models, and log all relevant details. An high level overview of the software architecture is shown in figure 2.

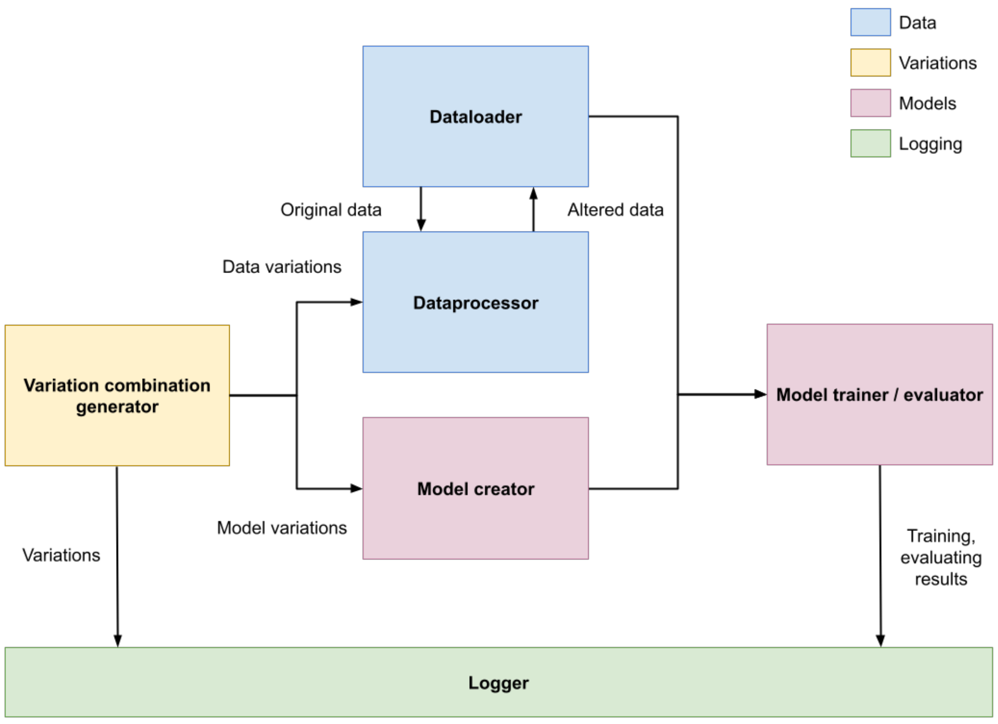


Figure 2: A high-level overview of the different components and their interactions within the software architecture. The process begins with the random generation of combinations of data and model variations. These variations are then divided: data variations are applied to the original data in the data processor, while model variations are used to create the model. The model trainer uses the processed data to train the model and evaluates the results on the test set. A logger keeps track of the generated variations and records the training and evaluation results.

The variation generator creates two types of variations: data variations (e.g., color inversion, contrast adjustments) and model variations (e.g., batch size, optimizer choice). We use equivalence partitioning to ensure comprehensive coverage of parameter values, avoiding problematic combinations through constraints.

Data variations are applied using two strategies: in-place and appending. In-place modifies existing data, allowing for layered augmentations, while appending adds modified data to a copy of the original dataset, maintaining its integrity. Outlier detection, through isolation forests, identifies anomalies within each digit label, resulting in a varied dataset that enhances model robustness.

The architecture supports various data augmentations, including color inversion, contrast adjustments, rotations, and translations. These augmentations are applied in a specific sequence to avoid inefficiencies and ensure reproducibility. Additionally, we explore the effects of including or excluding outliers and adjusting the amount of augmented data.

Model variations involve adjusting hyperparameters such as the number of hidden layers, training epochs, batch size, dropout layers, optimizer types, and activation functions. We also introduce the use of validation sets to prevent overfitting.

By systematically varying these parameters and using a structured experimental approach, we generate a diverse set of models, each with unique decision-making processes, contributing to our study of model multiplicity.

In the subsequent chapter, we examine the generated models to identify those that exhibit multiplicity. This involves evaluating each model's performance using a test set and grouping similar models into Rashomon sets. Initially, the models are assessed based on their accuracy on individually created handwritten digit samples, filtering out models with poor performance or low confidence. Models that pass this evaluation are grouped according to their prediction similarities for each test sample.

The chapter introduces the concept of frequent itemsets to identify sets of models that frequently predict the same labels. This approach is similar to the Market-Basket model used in data mining, where models are treated as items and prediction groups as baskets. We apply the A-Priori algorithm to determine frequent itemsets, setting a high support threshold to ensure that only highly similar models are considered. This results in the identification of pairs, triplets, and larger sets of models with similar predictive behaviors.

The chapter also explores the characteristics of these frequent itemsets by analyzing the variations in their training data and hyperparameters. Using metrics like Manhattan and Cosine distances, we measure the similarity between models within each itemset. The results show that models in frequent itemsets are not only similar in their predictions but also in the variations they underwent during training, suggesting the potential for generating Rashomon sets by creating models with similar variations.

In the final chapter, we focus on explaining model multiplicity. In user-centered AI applications, relying on a single model can lead to overtrust or undertrust issues. Introducing multiplicitous models, which have similar predictive accuracy, offers a solution by allowing users to consult a set of models, known as a Rashomon set, instead of a single model. This collective approach can enhance decision-making in critical systems like medical or legal advice platforms.

To explain the differences in predictions among models in a Rashomon set, we visualize their predictive behaviors and analyze the variations in their hyperparameters and data compositions. For instance, we examined three models from a Rashomon set that showed different confidence levels in predicting a sample labeled "5". Analysis revealed that differences in data augmentations, such as horizontal translations and contrast adjustments, significantly influenced their confidence levels.

In addition to dataset composition, model hyperparameters also play a crucial role, though interpreting their impact requires expert knowledge in neural networks. By understanding these differences, users can assign biases to certain models based on the specific characteristics of the sample they are evaluating.

Contents

Chapter 1

Introduction

As artificial intelligence (AI) becomes more prevalent in our daily lives [AB18], it extends its reach into more sensitive domains of life, such as healthcare [YBK18], finances and the justice system [Oli+22]. People start using machine learning (ML) models that generate predictions by analyzing private data, such as their medical or criminal history. The predictions made in these critical fields can become life altering, e.g. COMPAS determining the risk of recidivism in prisons [Far+23], bank loan qualification or university admission [DF18a]. Designers of machine learning models have the ability to introduce certain biases into the data which the algorithms learn from, this to influence the resulting predictions. Using the example of COMPAS, the data can be altered in a way so that prisoners having a certain gender or race are more frequently labeled as a higher risk of recidivism, to promote a specific agenda or policy. In these high-stake situations, the need for gaining insights into how these algorithms make their conclusions becomes fully apparent. If this process is neglected and the models become so complex that its decision-making process cannot be easily interpreted, they are referred to as black-box models, as shown in figure 1.1. Users of these systems will find themselves in a position where they either have to blindly trust the model (overtrust) or may be hesitant to depend on the predictions (undertrust). Non-expert users of these models with very little knowledge of the technical capabilities of the system find themselves quite often in this position. Overtrusting or overrelying on the AI can have dangerous consequences when the AI makes a mistake or the data is manipulted as in the COMPAS example. While undertrust hinders the adoptation of systems using artificial intelligence in society [Lar23].

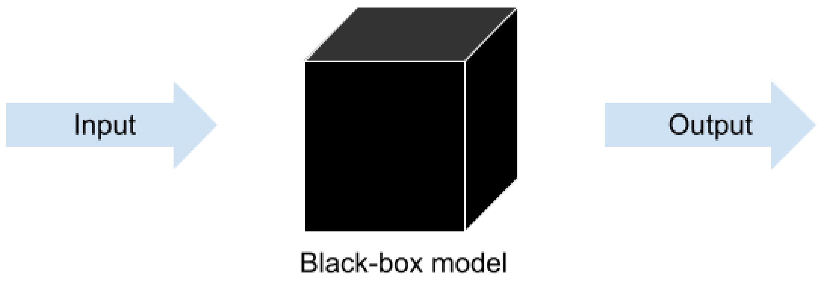


Figure 1.1: A schematic overview of a black-box model that computes an output based on an input, without explaining the inner logic.

Researchers and governments have argued for algorithmic accountability or a "right for explainability". Which resulted in laws in the EU General Data Protection Regulation (GDPR) that establish a certain amount of transparency around automated decision-making [Kam19]. These laws ensure that users receive meaningful information about the inner workings of these systems and they can be referred to as white-box models. However, as artificial intelligence takes on increasingly advanced tasks in society, its inner workings become more complex as well. The explanations provided for these complex models are often based on their algorithmic decision-making, which non-expert users struggle to understand. This literacy gap hinders their understanding on how and why their input leads to a certain outcome [Che+19]. This problem, in combination with the inappropriate amount of trust in these systems, led to the emergence of a subfield of AI called Explainable Artificial Intelligence (XAI).

XAI aims to transition the development of black-box models towards creating more transparent AI. It is essential to provide users with insight into these systems and give them the information to point out errors or flawed systems. Transparent AI does not only benefit end users; developers of these systems can proactively avoid issues by understanding the details on how the system behaves. Which can also be used to continuously improve them more easily. Learning new facts from interpretable models, such as new chess strategies from AlphaZero, can be desirable [AB18]. When the tools and technologies from XAI are correctly applied, users' over- and undertrust will be transformed into an appropriate amount of reliance in AI [FW23].

Despite the numerous benefits XAI brings, there are multiple reasons why it is not systematically used. The biggest reason being, the complexity of creating interpretable AI, especially the widely-used deep neural networks (DNNs). These models consist of multiple hidden layers each having numerous neurons. The collaboration of these neurons often result in a high accuracy, but interpreting such a nested network is a difficult task. There is often a trade-off between accuracy and model interpretability, where the most accurate models are often less explainable and the more interpretable models less accurate [AB18]. However, recent research [Che+18], has shown that interpretable models can be as accurate as other black-box models. These findings can be based on the "multiplicity of good models" phenomenon [HG18]. It says that multiple models can have the same amount of accuracy for a certain task, but can be completely different in their internal decison making process [BRB22].

This has sparked interest in the way models come to their internal decision process, based on their architecture and the data they are trained on. If we make a small change in the model's dataset or architecture, its internal decision making should change aswell, but can we link the made differences to the changes in the decision process? The question we search to answer in this thesis is, can the decision process of a model be explained by comparing the changes in its trained data and architecture with a large amount of similar but slightly varried models. To do this we train an extreme amount of redundant models and compare their predictions and differences to try explain the inner logic of these models.

In this thesis, the choice is made to focus on neural networks trained on the MNIST dataset, introduced in [Lec+98]. The rationale for choosing neural networks is that these models are not inherently interpretable and that they are modular. The latter characteristic means that they can be easily customised, for example, by adding a layer or changing activation functions. This flexibility will enable and simplify the process of creating varried models. The MNIST dataset consists of 70000 28x28 grayscale images of the ten handwritten digits, with ten examples of each digit shown in figure 1.2. The dataset is originally derived from the larger NIST Special Database 19, which also contains handwritten letters. The MNIST dataset has become a standard benchmark for classification tasks in machine learning and other computer vision applications [Coh+17]. The fact that these images are grayscale and the small dimensions of the images make the dataset easy to work with and perhaps expand. Another more personal reason to validate the choice of the dataset, is that in the course "Human-AI Interaction", the very basis of this study was laid. During an assignment in this course I studied the impact on the accuracy of neural networks when the digits in the MNIST dataset were slightly altered. During this introduction, it was made clear that when a model trained on the original dataset was given a handwritten sample, that the accuracy of the model plummeted. This occurrence intrigued me to further research how this could happen, which has lead to the development of this thesis.

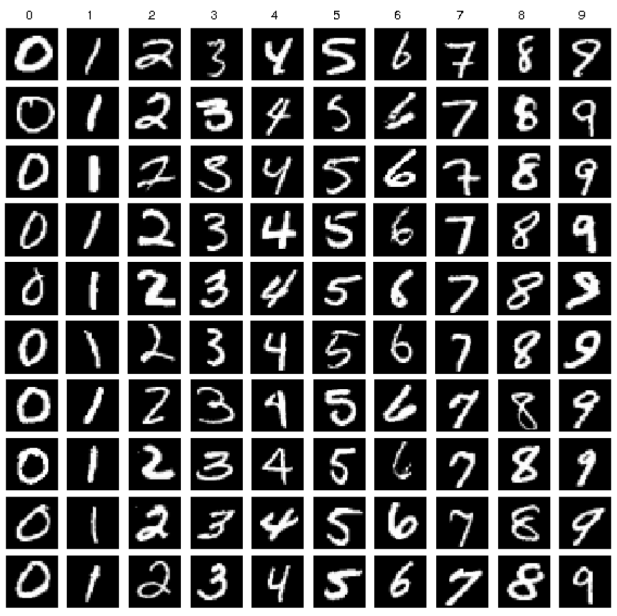


Figure 1.2: The MNIST dataset, numbers zero up until nine are the labels which are pictified as the columns, of each label there are ten shown examples. The examples are already diversified, e.g. a seven with and without a horizontal stroke, which helps in the further creation of a diverse dataset. [LYP16]

This thesis is divided into multiple chapters: "Model Multiplicity", "Generating many Models", "Searching for Multiplicity in Many Models" and "Explaining Model Multiplicity". The first chapter delves into the topic of model multiplicity, explaining its meaning and how it can be used to increase trustworthiness. The second chapter discusses the method introduced to generate large numbers of models that vary on different parameters. The third chapter covers the process of searching through these generated models and grouping similarly behaving models after a filtering process. The final chapter explains how the different parameters used to generate these models account for the varied predicting behaviors of models with similar accuracy.

Chapter 2

Model multiplicity

The first chapter delves into the concept of model multiplicity and its formulation in a general context. It begins with a definition of model multiplicity, followed by a discussion and comparison of various types of multiplicity. Additionally, the chapter explores the impact of and challenges associated with multiplicity and analyzes related work.

It is important to note that this section will primarily focus on model multiplicity in classification models. While this phenomenon also occurs with regression models, the decision boundaries used by classifiers often provide clearer examples for illustration. It's worth noting that much of the discussion here draws on insights from the seminal work "Model Multiplicity: Opportunities, Concerns, and Solutions" by Black et al. [BRB22], one of the few high-quality sources available on this emerging topic

2.1 Defining model multiplicity

Model multiplicity was first introduced as "the Rashomon effect", which is based on a Japanese movie. In this movie, four people spectate the same criminal incident from four different vantage points. Whenever the case came to court, all four witnesses stated the same facts but their stories of what happened are very different because of the different vantage points. The Rashomon effect signifies that within a set of functions, there exists a multitude of different elements capable of producing the same minimum error rate [Bre01]. What this means is that when you have a collection of functions fitted on the same dataset, a certain amount of functions will have the same accuracy while having different parameters. The set of functions that exhibit this behavior is often referred to as the Rashomon set [DAm21].

When the term "functions" in the definition is substituted with "models", it can be understood as follows: "models achieving comparable accuracy for a specific prediction task while differing in terms of their internals". Here, "internals" refer to the internal processes of a model that dictate how it makes a certain decision [BRB22]. The concept defined by using the term "models" instead of "functions' is referred to as "Model Multiplicity". To further clarify this definition, an example is provided in figure 2.1. This figure illustrates two binary classification examples, each depicting two examples of Rashomon sets and thus, model multiplicity.

2.2 Different ways of multiplicity

As defined in section 2.1, model multiplicity describes how two or more models can be internally different yet yield the same accuracy for a certain prediction task. In this section, we explore two ways in which equally accurate models can differ: either in their internals or in their predictions.

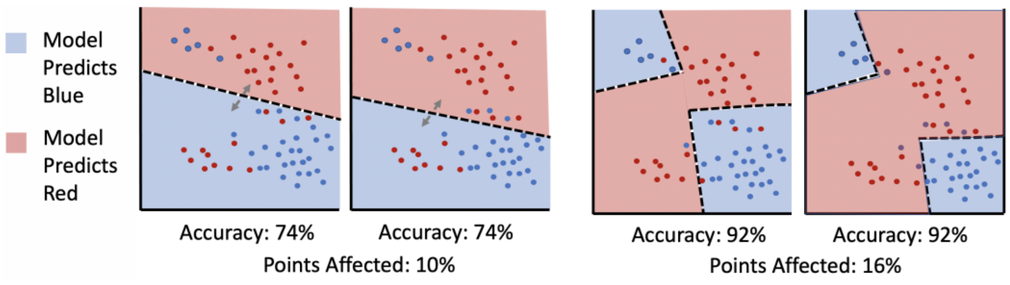


Figure 2.1: Two independent cases of model multiplicity, a simpler linear model (on the left) and a more complex non-linear model (on the right). Both cases involve the same binary classification task, which is to label the dots with the corresponding color. Despite the two models in both examples achieving the same accuracy, the models have a diferent decision boundaries, illustrating their distinct decision-making processes [BRB22]. Image from [BRB22].

2.2.1 Procedural multiplicity

When two or more models achieve the same accuracy for a prediction task but differ in their internal logic, this multiplicity is referred to as procedural multiplicity [BRB22]. The internal logic of a machine learning model can be represented as a decision boundary. The decision boundary is a hyperplane that separates different classes in a specific feature space. It serves as the representation of a line or plane that indicates where the model's predictions transition from one class to another. Essentially, the decision boundary is a visual representation of how a model determines which label to predict based on the features along the axes.

For instance, in figure 2.2, two examples of decision boundaries in the same two-dimensional feature space are illustrated. These decision boundaries vary in complexity; example A depicts a linear decision boundary, while example B represents a non-linear decision boundary. Despite their different complexities, both models in this example achieve the same accuracy, namely 100%, in labeling each sample with the corresponding label.

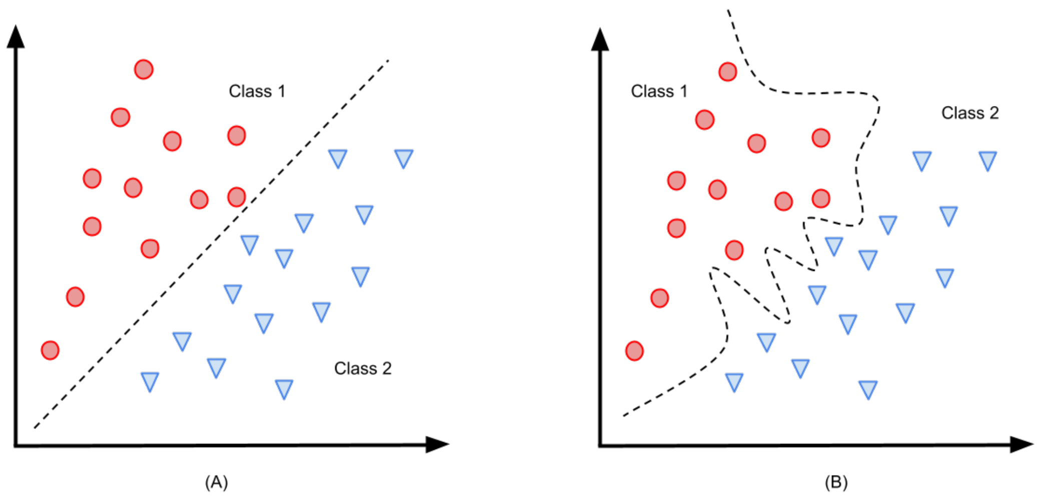


Figure 2.2: Two representations of decision boundaries are depicted, each in its own two-dimensional feature space. In subfigure A, a simple straight line serves as the border between both classes. In contrast, subfigure B features a curved line that delineates the boundary between the two classes.

As exemplified above, procedural multiplicity can occur when different but equivalent decision boundaries are utilized. Decision boundaries are learned during the training phase of a machine learning model. The complexity of the boundary varies based on multiple factors:

1. The dataset: if the dataset is imbalanced, the boundary can be biased toward the majority class.
2. The weight initializer, if the weights of a neural network are initialized randomly or according to a certain algorithm [Meh+20].
3. The machine learning algorithm: simple models like linear models produce linear decision boundaries, whereas deep neural networks can produce complex decision boundaries ranging from linear to curved, depending on factors such as the number of hidden layers [BRB22].
4. The amount of learning: overfitting and underfitting can affect the complexity of the decision boundary.

Another way to achieve procedural multiplicity is by employing multiple models with the same accuracy but utilizing different features from the data. This means that one model may process a specific subset of the data to predict the label, while another model utilizes a different subset. Consequently, the decision boundaries of both models are entirely different, as the axes representing the features are not the same [BRB22].

To provide a concrete example, consider two machine learning models designed to score a person's creditworthiness. One model bases its decision on the user's gender, while the other model relies on their income and tax information. Despite predicting the same label, the models' decision-making processes are fundamentally different. However, users will remain unaware of any differences between them, as both models will behave identically and predict the same labels [And+20]. This introduces the need for explanations in multiplicity, allowing users to gain insights into how these models differ in their operations.

2.2.2 Predictive multiplicity

In addition to procedural multiplicity, there is predictive multiplicity. Predictive multiplicity occurs when two or more models have the same accuracy but differ in at least one prediction. This means that models with the same accuracy may predict different labels for some datapoints in the same input set [BRB22] [MCU20].

Different model predictions on a certain input indicate differences in their decision boundaries. Consequently, predictive multiplicity can be considered a special case of procedural multiplicity [BRB22]. However, the challenge lies in discovering predictive multiplicity. Whenever procedural multiplicity is found, the process of searching for at least one data point where the models in a Rashomon set disagree can become endless.

2.3 Impact on model evaluation

When training a model, various choices must be made, including what type of model to use, how to format the input data, which data attributes are important for the model, the batch size, the number of epochs, and whether the model should see a validation set. The most common approach to addressing these questions is by training models with different configurations and selecting the model with the highest accuracy. However, in the presence of model multiplicity, multiple models may achieve the same accuracy [BRB22]. This raises an important question: which is the best model if multiple ones have the same accuracy?

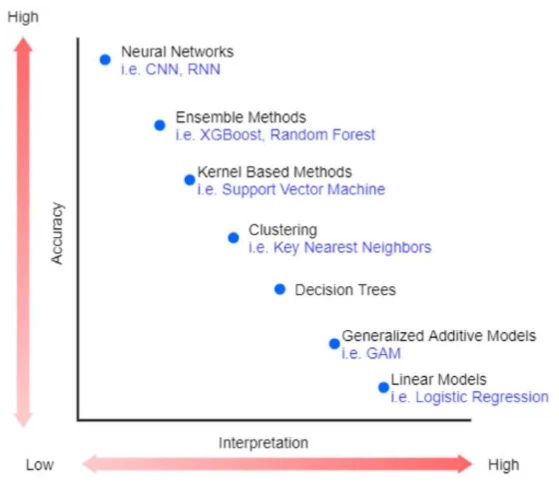


Figure 2.3: The accuracy-interpretability trade-off between the most common machine learning algorithms. Models with high inherent interpretability often have a lower predictive power and consequently may achieve lower accuracy. Conversely, models with high predictive power may sacrifice interpretability.[AZA21]. Image from [AZA21].

2.3.1 Occam's razor

One possible approach is to apply Occam's razor principle, often interpreted as favoring simpler solutions. In the context of model multiplicity, this principle suggests that when multiple models achieve the same accuracy, the simplest should be chosen. Simplicity in machine learning is often equated with the interpretability of the underlying algorithm [Bre01]. For instance, models employing inherently interpretable algorithms such as decision trees or linear models are considered simple, whereas deep neural networks are often perceived as more complex. To illustrate, when both a decision tree and a deep neural network achieve the same accuracy for a prediction task, the decision tree should be favored due to its simpler architecture.

However, this solution falls short when multiple models possess the same accuracy and architectural complexity. This represents the dilemma that cannot be resolved solely by adhering to Occam's razor. Consequently, when there are multiple models in a Rashomon set, there is no single "best" model if only accuracy is being accounted for.

2.3.2 Other desirable properties

Thus, determining the best model solely based on accuracy is no longer feasible. At first glance, this may appear to be a problem. However, it actually offers more flexibility to optimize models for other desirable properties. Properties such as interpretability, fairness, and robustness were previously viewed as trade-offs between having these and achieving the highest accuracy [Ber+19]. In figure 2.3 the trade-off between inherent interpretability and accuracy for the most common model architectures is depicted. Now, thanks to model multiplicity, these desiderata can guide the model selection process in addition to maximizing predictive accuracy.

For example, in life-altering situations such as the COMPAS case introduced in chapter 1, the interpretability of the model is crucial. Therefore, when two or more models form a Rashomon set, the model with the highest interpretability should be prioritized. If other desirable properties are equally important, the choice should be made to select the model that best fits these desiderata. Thus, the model evaluation process shifts to prioritize selecting the model that best fits the problem, rather than solely focusing on accuracy.

2.3.3 Impact of the law

Thus, model multiplicity provides the freedom to prioritize other desiderata in the model selection process without sacrificing accuracy. However, when focusing on the model's fairness, the freedom that allows model developers to focus on different features of the data can have legal implications.

In high-stakes scenarios, machine learning models may be trained on data containing sensitive personal information such as gender, ethnicity, sex and race [RHN23]. These features, often termed as normatively objectionable, are considered morally unacceptable as a basis for decision-making. Models that utilize these attributes or a combination of them to make decisions can lead to discrimination. The discrimination law, is the law that prohibits model developers to create these models in certain domains. With model multiplicity, developers can create models that abstain from using these attributes while maintaining equivalent accuracy to models that do [BRB22].

To exemplify this, let's consider the COMPAS example. The original COMPAS model aimed to predict recidivism to assess a person's likelihood of committing a crime. The COMPAS dataset consists of 137 features about an individual and their past criminal record. COMPAS has been frequently accused of racial bias, allegedly negatively impacting black defendants more than white defendants [DF18b]. Assuming these accusations are true and that the model uses features related to the defendant's race, model multiplicity demonstrates that it's possible to train a model that is equally accurate without using racially biased features.

2.4 The implications of model multiplicity

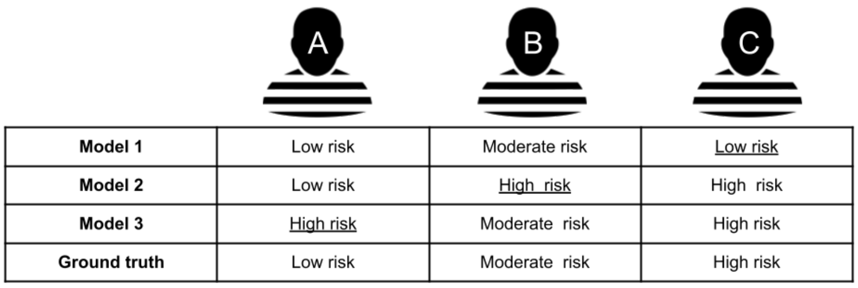
Multiplicity provides model developers with increased flexibility to prioritize desiderata such as interpretability, fairness, and robustness alongside accuracy. However, this flexibility can also lead to significant problems. In this section, two of these problems will be discussed: underspecification and justification.

2.4.1 Underspecification

Continuing to choose a model on the basis of acuracy alone, disregards other important factors, such as interpretability. When evaluating different models, as discussed in Section 2.3.2, it becomes necessary to choose the model that best fits the desired properties, or desiderata. This underscores the need to specify these properties upfront, as without explicit definition, it is unlikely that a model will naturally exhibit them [BRB22]. This challenge is known in literature as underspecification, emphasizing the importance of defining properties in advance to guide model development [DAm+20]. The following

As discussed in section 2.3.3, discrimination laws impose strict prohibitions on the use of certain characteristics in decision-making across a range of high-stakes domains, such as recidivism and lending. Model multiplicity demonstrates to developers that models can be trained without relying on normatively objectionable features. However, a challenge arises when removing proscribed features does not necessarily eliminate disparities. This can occur when the unfavorable predictions made by the model are indirectly influenced by other features. For example, in section 2.3.3, we assumed that the COMPAS model used features directly linked to the defendant's race. The solution was to train multiple equally accurate models that do not use these features. However, what if these newly generated models use other features that indirectly correlate with the defendant's skin color? This could result in the same discrimination as the model that directly used the defendant's race [BRB22].

Furthermore, model developers can exploit models that do not directly rely on controversial features while remaining discriminatory. This phenomenon, known as proxy discrimination, occurs when neutral traits are used as proxies for prohibited traits [DCF17]. It allows the behavior of the model to be justified by explaining that it does not use legally proscribed



Model 1	Low risk	Moderate risk	Low risk
Model 2	Low risk	High risk	High risk
Model 3	High risk	Moderate risk	High risk
Ground truth	Low risk	Moderate risk	High risk

Figure 2.4: Three distinct defendants undergo assessment by three unique COMPAS-like models. All models demonstrate identical accuracy rates of 66.6%, as verified by the ground truth depicted in the final row. Because of predictive multiplicity all three inmates are treated differently by at least one model. For example, when evaluated by model 3, defendant A is assigned a high risk of recidivism despite having a low actual risk. In certain scenarios, such as with defendant C and model 1, unjustifiable positive outcomes may arise. The lack of justification beyond accuracy for these predictions underscores the ethical dilemmas they present.

features. Proxy discrimination is a prime example of a result caused by underspecification; models are less likely to discriminate against users if that goal was explicitly defined during the model development process.

2.4.2 Justification

Multiplicity of models not only introduces the challenge of underspecification but also raises concerns about justification. When developers create multiple models leading to predictive multiplicity, these models may yield different labels for the same data point. The issue arises: what rationale can be provided when one model predicts unfavorably for an individual while another equally precise model predicts favorably [BRB22]?

To illustrate, consider a scenario where a defendant assessed by the COMPAS model is deemed to have a high risk of recidivism, while another model with comparable accuracy assigns a low risk label. Without considering any additional parameters beyond accuracy, there is no clear basis for favoring one model over the other This implies that due to model multiplicity, there will always exist an alternative model capable of altering an individual's prediction. In figure 2.4, this concept is elaborated upon with three different inmates (A, B, and C) being assessed by three distinct yet equally accurate models. Consequently, model developers are deprived of a solid justification, particularly in high-stakes scenarios where additional criteria are essential to justify predictions [BRB22].

2.5 Related work

As model multiplicity is a rather new topic, there has not been done a lot of research on it. In this section the research work around model multiplicity will be discussed and what their findings where.

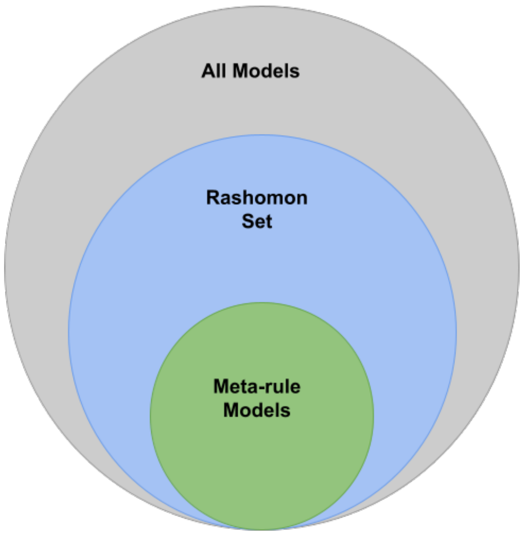


Figure 2.5: The complete set of models (grey) encompasses those with the same accuracy for a given prediction task (blue), which, in turn, includes models adhering to the meta-rules (green). As more rules are defined, the number of candidate models for a specific task decreases.

2.5.1 Meta-rules

In section 2.4.1, the issue of underspecification is introduced, which arises when model developers fail to specify the desired behavior of models upfront. Black et al. proposed the concept of meta-rules as a solution to the underspecification problem. Meta-rules entail a document outlining a set of criteria for acceptable procedural and predictive behavior beyond mere accuracy [BRB22].

Meta-rules serve as a valuable tool for clarifying to model developers how to optimize for desired properties, as discussed in section 2.3.2, during the model building process. These criteria also function as a model evaluation tool; all models achieving similar levels of accuracy on a prediction task (Rashomon set) must adhere to these meta-rules and exhibit behavior consistent with them [BRB22]. This approach yields a reduced set of candidate models, as illustrated in figure 2.5.

When a set of models meets the criteria outlined in the meta-rules, these rules can be used to justify why a particular model is chosen. Instead of selecting models arbitrarily, developers explicitly evaluate the differences between Rashomon models. However, it becomes challenging to explore all possible models that satisfy these rules when strict constraints are imposed. This challenge is aggravated by limitations such as the developer's ability to conduct experiments and factors like time and budget constraints. To address this, Black et al. suggest explicitly defining all the constraints developers may encounter, including factors like available funding, as these constraints influence the breadth of the search for multiplicitous models [BRB22].

Following the identification of the meta-rule-compliant model set, a crucial question arises: how do we select a model from this final set? Black et al. propose three aggregation techniques to distill a single model from a collection of equally "good" models. These techniques aim to restore justifiability by elucidating the process and criteria for model selection. To illustrate these proposals, let's first define the group of models that satisfy the meta-rules as $\mathcal{M}$ [BRB22].

The first technique is called mode aggregation. When this technique is applied, a voting system is established. Each model in $\mathcal{M}$ predicts a label for a certain sample x. Each model "votes" for its predicted label, and the label with the highest number of votes is chosen as the correct label for that sample. The mode aggregator $\overline{m}$ is the result of this voting system and harnesses the collective knowledge embedded within all voting models. By having this collective knowledge, the aggregator minimises the chance for a disagreement between its prediction and any individual chosen model from $\mathcal{M}$ [BRB22].

Randomized predictions and random model selection are the second and third technique proposed by Black et al. and are based on the principle of randomness. With randomized predictions, a random model from $\mathcal{M}$ is chosen for each sample x, and that model decides the label for that sample. On the other hand, random model selection involves arbitrarily choosing one model $\mathcal{M}$ from and using it to predict labels for all samples. The distinction lies in the fact that randomized predictions involve selecting a new model for each sample, while random model selection chooses a model once for all samples [BRB22].

Each of these techniques, when combined with the documented meta-rules, offers a means of justifying why a particular model is chosen over another. However, the context in which the model operates often dictates the preferred technique. To exemplify, consider models operating in the legal domain, such as the COMPAS model. In such domains, the government bears a legal burden to ensure consistency in decision-making, rendering randomized predictions ill-advised. Moreover, in legal cases, preference is often given to stable model explanations, making reliance on a single model inadvisable. Consequently, mode aggregation emerges as the recommended technique to address multiplicity in these scenarios. Conversely, in low-stake scenarios like advertising, where consistency is less critical, and variety in exposure is desirable, randomized predictions prove to be the optimal technique. This approach ensures that all models have an impact in this low-stakes environment, allowing for a diverse range of ads to be shown to users [BRB22].

2.5.2 Metrics to justify multiplicity

As discussed in Section 2.4.2, in addition to underspecification, a lack of justification presents a challenge stemming from predictive multiplicity. To elucidate this issue further, Marx et al. propose that developers of multiplicitous models should quantify and disclose multiplicity to stakeholders, akin to how we quantify and disclose test error. This approach would necessitate changes in how models are developed and deployed in applications involving human interaction [MCU20]. Marx, Calmon and Ustun investigate how stakeholders could be informed by introducing two metrics: ambiguity and discrepancy.

Before defining these metrics, the authors establish the concept of a Rashomon set as an ϵ-level set, with ϵ denoting the maximum acceptable error tolerance in accuracy among models to be included in the ϵ-level set.

Ambiguity

Marx et al. introduce the metric ambiguity over the ϵ-level set $S_\epsilon(h_0)$. This metric measures the proportion of data points in a dataset (n) that receive conflicting labels when classified by another model $h \in S_\epsilon(h_0)$. The definition of ambiguity (α) involves a baseline classifier (h_0), which represents the model typically deployed, and a dataset characterized by features denoted by x. The formal depiction of ambiguity is provided in equation 2.1.

$$\alpha_\epsilon(h_0) := \frac{1}{n} \sum_{i=1}^{n} \max_{h \in S_\epsilon(h_0)} \mathbb{1}[h(x_i) \neq h_0(x_i)] \tag{2.1}$$

Essentially, ambiguity quantifies the extent to which predictions can vary when using an alternative model with comparable accuracy. This metric indicates the proportion of individuals whose predictions are influenced by the choice of model from a Rashomon set and, consequently, the proportion of individuals subject to potential discrimination [MCU20].

Discrepancy

Discrepancy (δ) over the ϵ-level set $S_\epsilon(h_0)$, is the maximum proportion of inconsistent predictions between the baseline classifier and another classifier $h \in S_\epsilon(h_0)$, which is shown in equation 2.2

$$\delta_\epsilon(h_0) := \max_{h \in S_\epsilon(h_0)} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[h(x_i) \neq h_0(x_i)] \tag{2.2}$$

In essence, discrepancy quantifies the maximum number of predictions that could change if the baseline model were replaced by an equally accurate alternative.

When these metrics are given to the stakeholders, the model evaluation process discussed in section 2.3 becomes more fine cut. As stakeholders gain insight into the potential ramifications of changing the deployed model and whether deployment is warranted in the first place [MCU20].

Chapter 3

Generating many models

As outlined in the introduction (see chapter 1), our objective is to create a large set of models based on the MNIST dataset. These models will have similar accuracy levels but will differ in how they arrive at their decisions, achieving what we term 'model multiplicity' (discussed in chapter 2). Unlike the examples in the previous chapter, our experiments use multi-class classifiers instead of binary ones. Our first step involves generating a significant number of models that vary in their internal decision-making processes. This section details our experimental approach to accomplish this.

To induce variations in the models, we will adjust several aspects, such as the training data and the parameters of the machine learning algorithms (explored further in sections 3.2 and 3.3). Subsequently, we will describe the structure of our experiment, including the key components and how they interact to produce models with diverse decision-making processes.

3.1 The architecture

The objective of this experiment is to generate models that vary along different dimensions, including model hyperparameters (see section 3.2) and data variations (see section 3.3). To achieve this, we require a method capable of automatically generating a large quantity of models based on random combinations of these hyperparameters and data variations. This automation is essential to avoid the bias and significant time investment associated with manual model creation. In this section we introduce a software architecture that facilitates the generation of numerous neural networks with varying properties.

This architecture consists of several components, each performing a specific task to generate a fully trained neural network on the MNIST dataset. However, the system is designed to work with any dataset and can incorporate new variations as needed. The components handle tasks such as generating random combinations of variations, loading the dataset, applying data variations, creating and training the model, and logging. Figure 3.1 shows how these components work together to achieve the final goal.

3.1.1 Variation generator

The equivalence partitioner is responsible for generating combinations of variations. This component creates two types of variations: model variations and data variations, each consisting of different sets of parameters.

Data variations are composed of parameters, such as color inversion or contrast adjustments, which will be further discussed in section 3.3. These parameters generally cover three aspects: the extent of data to be affected, the intensity of the data manipulation, and whether it should

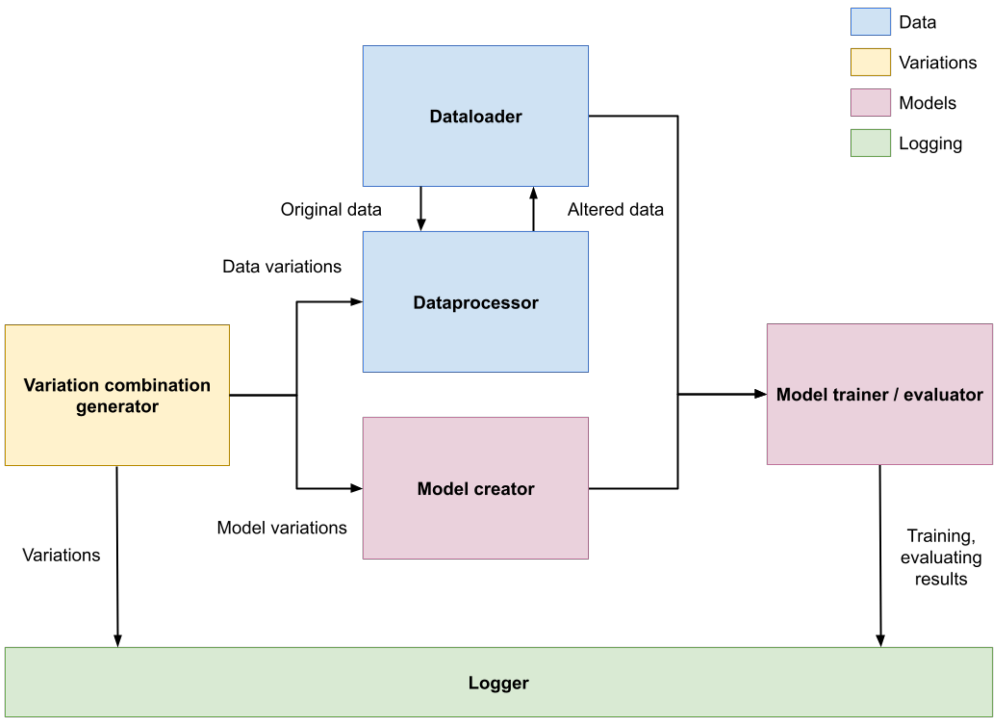


Figure 3.1: A high-level overview of the different components and their interactions within the software architecture. The process begins with the random generation of combinations of data and model variations. These variations are then divided: data variations are applied to the original data in the data processor, while model variations are used to create the model. The model trainer uses the processed data to train the model and evaluates the results on the test set. A logger keeps track of the generated variations and records the training and evaluation results.

Parameter	Description	Parameter partitions
Amount typicals	Percentage of typicals allowed	0, 0.2, 0.4, 0.6, 0.8, 1
Amount outliers	Percentage of outliers allowed	0, 0.2, 0.4, 0.6, 0.8, 1
Data variation amount	Percentage of data to apply augmentation on	0, 0.2, 0.4, 0.6, 0.8, 1
Data variation strategy	In-place or append augmentation strategy	in-place, appending
Contrast factor	Adjusts contrast	0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8
Rotation degrees	Amount of degrees to turn	-20, -15, -10, -5, 0, 5, 10, 15, 20
Translation factor	Amount of pixels to translate	-2, -1, 0, 1, 2
Hidden layers	Amount of hidden layers	1, 2, 3
Dropout	Include or exclude dropout layers	true, false
Optimizer	Optimizer for the model	adadelta, adagrad, adam, adamax, ftrl, nadam, rmsprop, sgd
Activation function	Activation functions for the hidden layers	elu, exponential, gelu, hard sigmoid, linear, relu, sigmoid, softmax, softplus, swish, tanh
Batch size	The batch size for training	32, 64, 128, 256, 512

Table 3.1: A detailed overview of the various data and model augmentations used in the experiment. The upper section of the table lists the data variations, while the lower sections covers model variations, detailing the structure of the neural network.

be applied in-place or appending. For instance, consider the augmentation technique of contrast adjustments. The associated parameters include the amount of data to be contrast-adjusted, whether the contrast should be increased or decreased, and whether the adjustments should be made to the working dataset or the original dataset. Exceptions to these three parameters include outlier detection and color inversion augmentations. Outlier detection involves two parameters: the amount of typicals and the amount of outliers. Similarly, color inversion involves two parameters since the strength of inversion cannot vary.

Model-based variations are hyperparameters such as batch size or choice of optimizer, which are determined before the training process, these are described in section 3.2. Unlike data variations, these parameters are individual and not grouped.

Randomness in creating different parameter combinations is a core aspect of this experiment. However, pure randomness could result in incomplete coverage of cases or specific scenarios. To address this, we implement equivalence partitioning. All parameters are divided into partitions considered equivalent. For example, the percentage of outliers is divided into five partitions: 0, 0.2, 0.4, 0.6, 0.8, and 1. These values represent five equivalence classes: the lower boundary (0), low (0.2), lower middle (0.4), upper middle (0.6), high (0.8), and the upper boundary (1). Table 3.1 shows all parameters and their corresponding partitions.

Some combinations of partitions can cause issues in the final result. For example, if both the amounts of typicals and outliers are set to zero, no data will be present in the dataset, causing obvious problems. To avoid such problematic combinations, constraints are applied. If a combination of parameters is problematic, it is regarded as invalid, and a new combination is generated.

3.1.2 Data loader

The data loader is responsible for loading the original data and storing both the working data and the original dataset. This makes it a crucial component of the experiment. It ensures that other components can access the correct amount of data from the appropriate subset, allowing augmentations to be applied accurately.

3.1.3 Dataprocessor

The data processor is the component responsible for executing all data variations provided by the equivalence partitioner on the dataset. Upon receiving these variations, it reorders them in the correct sequence to avoid undesired variations, which will be further discussed in section 3.3.5 and executes them accordingly. The data processor works closely with the data loader, taking data from it before applying the variations and storing the results back into the data loader.

3.1.4 Model creation, training and evaluating

The creation of models based on the hyperparameters based on the random partition given by the equivalence partitioner is done in the model trainer component. It first generates the model by building the different layers prescribed by the partition, then it compiles it with correct optimizer and loss function. After these two steps the model is ready to receive the augmented data from the data loader which it can train on. Finally the test set is given to evaluate the generalization capabilities of the model on unseen data.

3.1.5 Logging

Since the process of random model generation is automated, we cannot know which variations are applied to a specific model. This information is crucial for comparing multiple models with the same accuracy, based on the variations applied to the training data and the model architecture. To address this, we need to log the augmentations applied to the dataset and the hyperparameters introduced to the model. This logging is managed by the delta component, which is linked to both a model and a dataset, representing how each model differs from the default dataset and model.

The delta component is integrated into various components such as the equivalence partitioner, data processor, and model trainer. Each time a new variation partition is created, a new delta is generated to log all the variations. When the data processor applies these variations, it logs the amount of data affected and the label distribution. Similarly, when a model is trained and evaluated by the model trainer, the results are logged in the delta. The data logged in a delta include:

1. **Random seed**: The seed used to ensure reproducibility, used in methods like the splitting of the training dataset.
2. **ID**: A unique identifier for the delta, used to track and link it with the corresponding model.
3. **Model layers**: Configuration details of the model layers, including the number of neurons and activation functions.
4. **Loss function**: The loss function used, always set to sparse categorical cross entropy.
5. **Optimizer**: The optimization algorithm used to minimize the loss function during training.
6. **Batch size**: The number of data samples processed in each training iteration.
7. **Dataset**: Description of the dataset composition, including applied variations and data distribution across training and validation sets.

Entry	Description	Value Ranges
Amount of layers	Number of hidden layers	1, 2, 3
Dropout	Inclusion or exclusion of dropout layers	0, 1
Activation function	One-hot encoded representation of the activation functions	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Batch size	One-hot encoded representation of the batch size	0, 1, 2, 3, 4
Validation set	Integer indicating the presence of a validation set	0, 1
Outliers	Float representing the percentage of outliers in the dataset	$[0, 1]$
Typicals	Float representing the percentage of typical data points in the dataset	$[0, 1]$
Invert	Float representing the percentage of inverted data in the dataset	$[0, 1]$
Horizontal translation	Float representing the percentage of horizontally translated data in the dataset	$[0, 1]$
Amount of horizontal translation	Number of pixels translated horizontally	-2, -1, 0, 1, 2
Vertical translation	Float representing the percentage of vertically translated data in the dataset	$[0, 1]$
Amount of vertical translation	Number of pixels translated vertically	-2, -1, 0, 1, 2
Rotate	Float representing the percentage of rotated data in the dataset	$[0, 1]$
Amount of rotation	One-hot encoded representation of degrees rotated	0, 1, 2, 3, 4, 5, 6, 7
Contrast	Float representing the percentage of contrast-adjusted data in the dataset	$[0, 1]$
Amount of contrast adjustment	Strength of contrast adjustment	0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8

Table 3.2: Detailed overview of the various entries in a compressed delta, including descriptions and value ranges for each parameter related to data and model augmentations.

8. **Has validation set**: A boolean indicating whether a validation set was used during training.

9. **Train results**: Loss and accuracy metrics from the last epoch of training, including validation results if applicable.

10. **Evaluation metrics**: Final loss and accuracy metrics on the test set to evaluate performance.

Once a model is fully generated, trained on the augmented dataset, and evaluated on the test set, the delta is exported to a JSON file. JSON is chosen because it can be easily converted to a Python dictionary, supports complex nested data structures, and remains human-readable.

While the delta component is primarily used for logging, it can also serve as a blueprint to reproduce the same or slightly altered models. For example, a delta for a random model can be imported for a new model with modified activation functions, creating a new model. A compressed version of the delta, containing core information in a more compact format, is used for comparisons. Some entries are one-hot encoded to make distance-based comparisons more meaningful, all aspects of the compressed delta are depicted in table 3.2.

3.2 Variation on parameters

The objective of this experiment is to construct a diverse ensemble of neural networks, each characterized by distinct decision-making processes. To achieve this, we adjust various hyperparameters of the models. Hyperparameters are settings determined before training starts, and they significantly affect both the training behavior and the final performance of the models. In our experiment, we modify the following hyperparameters: the number of hidden layers, the number of training epochs, the batch size, whether to include dropout layers, the choice of optimizers, and the type of activation functions used. The loss functions are not varied across the models; due to the requirement for multi-class classification capabilities, and the incompatibility of certain loss functions with this criterion, sparse categorical cross-entropy has been uniformly selected as the loss function for all models.

3.2.1 Amount of hidden layers

As introduced in chapter 1, neural networks are a fundamental concept of deep learning, which make use of layers built out of interconnected neurons to process data and analyze complex patterns in data. These layers are categorized into three types:

1. **Input layer:** The first layer of the network that receives the input data. The number of neurons equals the number of features in the data. For instance, the MNIST dataset's 784-dimensional data means the input layer has 784 neurons.

2. **Hidden layers**: These layers are sandwiched between the input and output layers and are crucial for learning. Varying the number and size of these layers alters the model's predictive behavior.

3. **Output layer:** The last layer of a network produces the output or the predictions that the model makes. In this experiment this layer will be of size ten, as there are ten possible labels that could be predicted.

The role of hidden layers is to analyze the data's embedded patterns; an increase in the number of these layers enables the learning of more complex patterns. This capability is often correlated with a deeper understanding of the problem, leading to enhanced predictive accuracy. However, too many layers can lead to overfitting, where the model fits the training data too closely and performs poorly on new data. This happens when the model's complexity exceeds the problem's needs [UJ20; Haw03]. Additionally, an excessive count of hidden layers can precipitate the vanishing gradient issue during backpropagation, where the gradients shrink, impeding weight updates in earlier layers [GBC16]. A high number of hidden layers also increases the time complexity, slowing down training and prediction [UJ20]. On the contrary, too few layers can lead to underfitting, where the model is too simple to capture the data's complexity [UJ20].Selecting an optimal number of hidden layers is thus critical and remains a subject of ongoing research.

In this experiment, we aim to generate a diverse set of models with varying decision-making processes. To ensure the training duration remains feasible, we confine the number of hidden layers to between one and three, with a consistent count of 128 neurons per layer.

3.2.2 Number of epochs

The training of a machine learning model consists of two main phases: the forward pass and the backward pass. During the forward pass, the model processes the input data to compute its output. This computation involves passing the data through various layers of the model. In the backward pass, the model adjusts the weights and biases of its neurons, starting from the output layer and moving towards the input layer. This adjustment is how the model learns how to identify patterns in the data.

An "epoch" is a term used to describe one complete cycle of passing the training data through the model both the forward and backward passes. Therefore, if a neural network processes all the training data once, that counts as one epoch. The total number of epochs indicates how many times the neural network will see and learn from the entire dataset. Exposing the neural network to more training data enhances its ability to discern patterns within the data. Training for an insufficient number of epochs leads to underfitting, where the network fails to capture essential patterns. Conversely, excessive training can result in overfitting, where the network begins to memorize the noise rather than the underlying data patterns.

The ideal number of epochs allows the model to "converge." This means the model no longer improves significantly with further training and has captured the primary patterns in the data. The convergence point depends on other model parameters, such as the number and size of the layers. Generally, models with more layers or neurons need more training to converge [Sid+18].

In this experiment, a large set of models, each characterized by a unique parameter configuration, must be trained until they converge. Due to the variability in architecture, these models are expected to require different training durations to achieve optimal performance.

To facilitate this process and mitigate overfitting risks, the experiment employs the "early stopping" regularization technique. This approach terminates the training when the performance on a validation set ceases to improve and instead starts to decline, indicating that the model has started to learn noise rather than useful data patterns. The "patience" parameter within this technique determines the number of epochs to continue training after no further improvements are observed. This helps to avoid stopping too early before the model has fully learned from the data. Figure 3.2 illustrates the early stopping technique and shows why monitoring the loss or error is crucial. Without early stopping, the model would overfit the training data and perform poorly on new, unseen data.

To manage the training time across many models, we set a hard limit of 100 epochs. If a model's validation loss continues to decrease even after this limit, training will stop regardless. This approach helps make the training of numerous models less time-consuming while ensuring they are adequately trained.

3.2.3 Batch size

During training, if the total amount of data is too large to fit into memory, an epoch can be divided into several smaller segments called "batches". These batches determine the number of training samples used to compute the gradient in a single iteration of model training. This means that after processing each batch (or iteration), the network updates its weights and biases, facilitating learning. The "batch size" is a hyperparameter that specifies the number of training samples in each batch. Figure 3.3 illustrates how an epoch is divided into several batches of a specific size.

There are several techniques that model developers can use to train their models efficiently by varying the batch size. These include:

1. **Batch Gradient Descent:** Here, the batch size equals the entire training set size, and the network updates after processing all data. This approach provides stable convergence due to the exact gradient calculation from the entire dataset [LeC+12]. However, this can require significant memory, especially for large datasets and when training on GPUs.
2. **Stochastic Gradient Descent:** Here, the batch size is one, meaning the network's parameters are updated after each sample is processed. Although the high noise level introduced by this approach can aid the model in escaping local minima, it can render the convergence path highly erratic and slow [LeC+12].
3. **Mini-batch Gradient Descent:** This method strikes a balance by setting the batch size between one and the total size of the training set. It merges the computational benefits of batch processing with the advantageous stochastic noise in gradient estimation. This

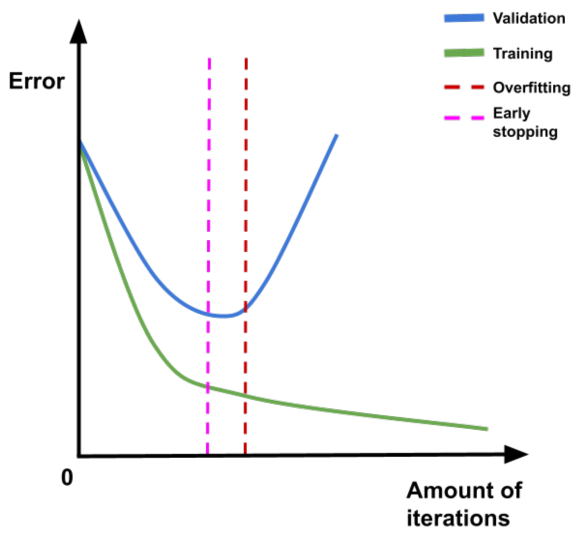


Figure 3.2: The typical error trajectories for both training and validation sets as a function of the number of training iterations. Initially, the error for both sets decreases as the number of iterations increases, indicating learning and improvement in model accuracy. The training error continues to decrease throughout, showing the model's increasing fit to the training data. Conversely, the validation error decreases to a point before starting to rise again, suggesting the onset of overfitting. The vertical dashed line marks the early stopping point, which is determined when the validation error begins to increase, indicating the optimal stopping point to prevent overfitting and ensure the model generalizes well to unseen data. [GQ01]. Replicated from [GQ01].

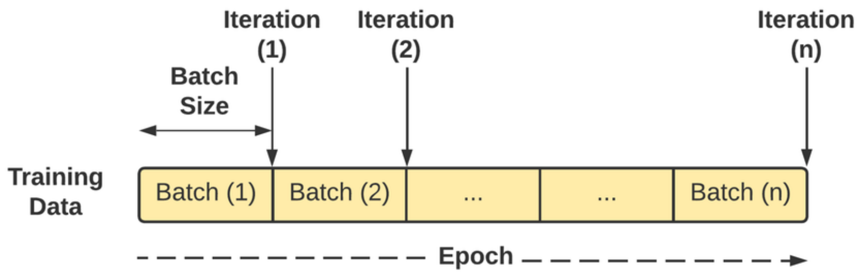


Figure 3.3: The training data segmented into multiple batches of a specific batch size during a single epoch. The number of batches (n) is determined by dividing the total amount of training samples by the batch size [Tha+24]. Imaage from [Tha+24].

balance facilitates more stable and accelerated convergence compared to pure SGD and avoids the memory constraints of full batch gradient descent. The main challenge here is selecting an optimal batch size; overly large batches diminish the stochastic noise's benefits, whereas excessively small batches decrease training efficiency [Ben12].

For this experiment, mini-batch gradient descent was selected due to its advantages in speeding up training and the flexibility it offers in adjusting batch sizes. Literature indicates that the optimal batch size is within the range of 32 to 512 [Kes+16; Ben12]. Therefore, we allowed the models in this experiment to train with batch sizes of 32, 64, 128, 256, and 512, aiming to cover a broad spectrum of sizes to enhance model diversity.

3.2.4 Dropout

As we discussed in Section 3.2.1, large neural networks are powerful and can accurately predict complex tasks. However, these larger networks are more prone to overfitting. Dropout is a technique used to prevent overfitting by randomly disabling neurons in the input and hidden layers during training [BS13]. When a neuron is "dropped out," it doesn't contribute to the network during that training epoch. The selection of neurons for dropout is governed by a predetermined probability, as specified by the model's developer [Sri+14].

Dropout prevents the network from memorizing the data by ensuring that each training sample is processed by a different subset of neurons [Sri+14]. This randomness means that no neuron can depend on the presence of another, forcing the network to learn more robust features. This approach spreads the learning across a broader set of connections. An example of dropout in a neural network with two hidden layers is shown in figure 3.4.

Incorporating dropout layers leads to model multiplicity in our experiments. By inducing variations in neuron participation, it effectively creates distinct models at each training iteration, even when all other parameters remain constant. This contributes to the extensive diversity of models examined in this experiment. To explore this, some models will include dropout layers while others will not.

3.2.5 Optimizer

The efficiency and effectiveness of a neural network's learning process are greatly influenced by the choice of optimizer. An optimizer helps adjusting the neural network's weights and learning rate to minimize the loss function, helping the model learn from the training data effectively..

The learning rate (γ) is a crucial hyperparameter that determines the size of the steps the model takes in updating its weights at each iteration. It directly controls how fast the model learns. A very high learning rate may lead the model to converge too quickly to a suboptimal solution, while a very low learning rate can slow down the whole learning process significantly. The impact of different learning rates is shown in figure 3.5.

In this experiment, we explore four categories of optimizers: basic gradient descent variants, adaptive learning rate optimizers, adaptive moment estimation optimizers, and hybrid optimizers.

Basic gradient descent variants

Basic gradient descent optimizers update the weights of the neural network by solely following the gradient of the loss function. This category includes the Stochastic Gradient Descent (SGD) optimizer, which adjusts the model's weights using a fixed learning rate based on one or a few data points at a time, as explained in section 3.2.3.

Using a fixed learning rate makes these algorithms straightforward to implement. However, this simplicity can lead to issues such as slow convergence, where the optimizer takes a long

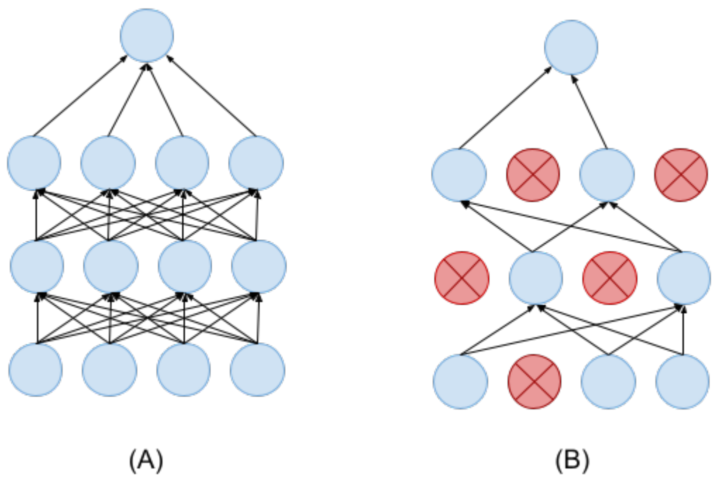


Figure 3.4: The effect of dropout in a neural network. Subfigure (A) depicts a standard neural network with two hidden layers, where all neurons are active during training. Subfigure (B) illustrates the network after applying dropout. Here, some neurons are randomly deactivated (marked with a cross). This process ensures that the active neurons do not depend solely on the full set of neurons but only on those that are active. This reduces the network's reliance on any individual neuron, helping to prevent overfitting. Image replicated from [Sri+14].

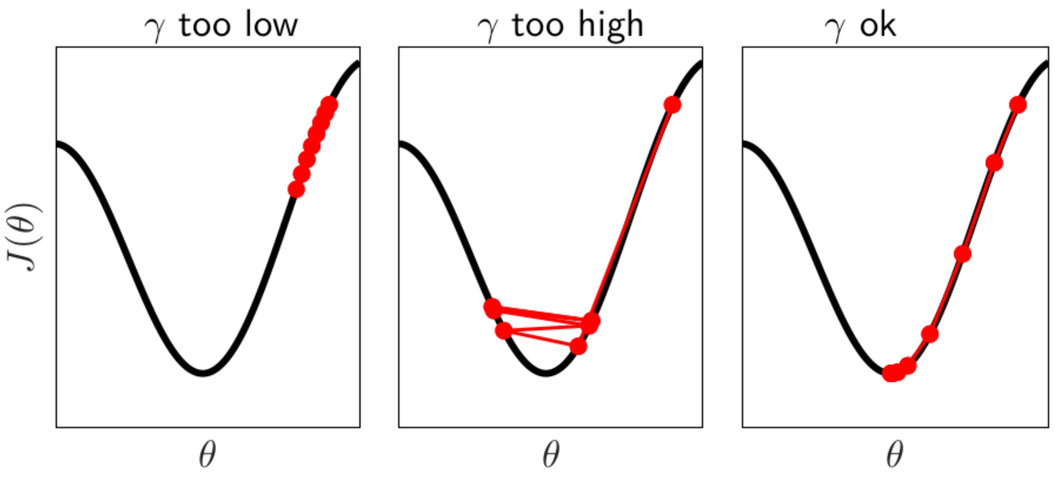


Figure 3.5: Impact of learning rate (γ) in optimizer algorithms and how it progresses the updates on a loss function $J(\theta)$ with respect to a parameter θ. In the first subfigure a low learning rate is shown which results in a small steps towards the minimum, which will take a lot of time. The second subfigure shows a high learning rate, which causes very large steps, this can cause the updates to overshoot the minimum, making it difficult for the model to converge. The right subfigure shows an appropriate learning rate, which is dynamic as it starts with big steps and decreases as it approaches the minimum. Image from [BN21].

time to find the minimum, or getting stuck in local minima, where the optimizer settles in a suboptimal point and misses the global minimum.

Adaptive learning rate optimizers

Adaptive learning rate optimizers are designed to overcome the challenges of using a fixed learning rate. Unlike basic methods that use the same rate for all updates, these advanced optimizers adjust the learning rate based on how the training is progressing. As the training continues, they typically reduce the learning rate, helping the model to gradually settle into the best solution without overshooting.

Another important feature of these optimizers is their ability to set different learning rates for each parameter of the model [Des20]. This means that features that occur more frequently can have smaller learning rates, preventing them from dominating the learning process, while less frequent features can have larger rates to ensure they are not ignored. This approach is especially beneficial for working with sparse data, where it's crucial to learn effectively from less common features.

In this experiment, we used Adagrad, Adadelta, and RMSprop as examples of adaptive learning rate optimizers.

Adaptive moment estimation optimizers

Adaptive moment estimation optimizers are a type of optimization algorithm that improves learning by using both past gradient information and adaptive learning rates. By calculating the momentum, which is the moving average of past gradients, these optimizers help smooth the learning path towards the minimum. This smoother path comes from incorporating information from previous steps, which helps overcome some of the erratic movements that can occur in optimization.

These optimizers also adjust the learning rate as the training progresses, speeding up convergence towards the best solution [Des20]. In this experiment, we used Adam, Adamax, and Nadam as examples of adaptive moment estimation optimizers.

Hybrid optimizers

Hybrid optimizers combine strategies from various optimization methods to enhance their overall performance. By blending the strengths of different approaches, these optimizers create a more robust and efficient way to find optimal solutions. They are designed to compensate for the weaknesses of individual methods, ensuring more reliable results across different types of problems.

In our experiments, we focused on the Follow-the-Regularized-Leader (FTRL) optimizer. FTRL uses a mix of L1 and L2 regularization to prevent overfitting, which is crucial for maintaining the model's ability to generalize to new data. This combination is particularly effective in online optimization scenarios, where data arrives in a stream, and the optimizer needs to update the model in real-time.

FTRL is especially useful for situations where many features in the data are irrelevant. By regularizing the model, it helps to focus on the most important features, ignoring the noise and preventing the model from overfitting to the less relevant data [Des20].

3.2.6 Activation function

Each node in a neural network layer uses an activation function to determine if the neuron should be activated. This decision affects whether the input is passed on to the next layer. Activation functions are crucial because they allow the network to model complex relationships and learn efficiently [SSA20]. These functions introduce non-linearity into the network, enabling it to approximate any continuous function and solve complex real-world problems [RAS20].

By using different activation functions, we can observe various learning dynamics and performance levels in the model. This is particularly relevant in studying model multiplicity, where diverse activation functions can lead to different model behaviors and generalization abilities. It's important to note that only the hidden layers use variable activation functions in this experiment. The output layer consistently uses the softmax function to ensure that models are comparable in their predictive results. Moreover, all hidden layers in a network use the same activation function to constrain the variation.

In this experiment, we explore three categories of activation functions: linear, non-linear, and specialized.

Linear activation functions

Linear functions pass the input directly through to the output. These are often used in the output layer for continuous value predictions or simple tasks [SSA20]. They are computationally efficient and stable but are not suitable for modeling complex patterns in hidden layers.

Non-linear activation functions

Non-linear activation functions form the cornerstone of contemporary neural networks by facilitating the capture of complex and high-dimensional data patterns, surpassing linear model capabilities. The experiment utilizes several non-linear activation functions, including ELU, GELU, hard sigmoid, sigmoid, softplus, swish, tanh, and ReLU, each contributing distinctively to the network's performance and functional capabilities [SSA20].

Specialized activation functions

This category includes functions typically employed under specific circumstances or in specialized network layers. In this experiment, the softmax function is used in the output layer to derive probabilities for each class label [SSA20]. The exponential function, which returns a positive number that can grow exponentially, is prone to numerical instability due to potential large magnitude values.

3.2.7 Presence of a validation set

The use of a validation set is a fundamental aspect of training neural networks. This subset of data, which is not used for training, serves to evaluate the model's performance on unseen data. Incorporating a validation set enhances generalization capabilities, mitigating the risk of overfitting. The validation set is instrumental in determining the optimal number of training epochs, as it enables performance monitoring for early stopping, as detailed in section 3.2.2.

However, using a validation set has its drawbacks. Primarily, it reduces the amount of data available for training the model. In our experiments, we create the validation set by reserving a portion of the training data for this purpose. This reduction can lead to a scarcity of training data, especially in datasets that are not very large to begin with.

For example, the original MNIST dataset has 60,000 images. While this number may seem large, the modifications and variations we discuss in section 3.3 can significantly reduce this size. Since the volume of training data is crucial for the performance of machine learning models [UL24], this reduction can make the model less effective, as it has less data to learn from.

Given these considerations, we allow models in this experiment to use a validation set or not. This choice introduces an interesting variable in studying model multiplicity, as it shows how different training strategies can lead to different model behaviors.

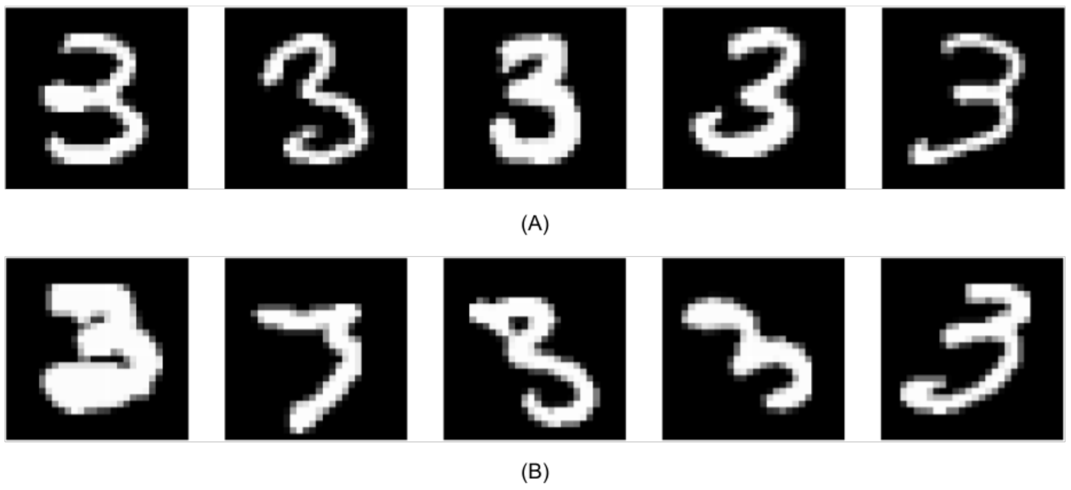


Figure 3.6: In this illustration, the top row (A) shows a collection of five typical samples of the number "3" from the dataset. These samples are very similar and represent the standard appearance of this digit. The bottom row (B) displays five outliers of the number "3", which are noticeably different from the norm, showing variations that set them apart from the typical examples.

3.3 Variation on data

As previously mentioned in the introduction (chapter 1), we are working with the MNIST dataset, which contains 70000 28x28 grayscale images of ten different handwritten digits. This section aims to discuss the various modifications we apply to the dataset's images. We will address three key aspects for each type of variation: why we apply these variations, how we implement them, and the effects they have on the outcomes

The specific variations we will discuss include separating outliers from typical images, inverting colors, adjusting contrast, applying rotations and translations. Additionally, this thesis introduces two methods of making these adjustments: appending changes to the original images and making changes directly in-place. To provide clear examples, our discussion will focus on images labeled as "3" from the MNIST training dataset.

3.3.1 Splitting data into outliers and typicals

The first variation we make to the dataset is dividing it into two groups: outliers and typicals. This process is often referred to as anomaly detection in literature, which involves identifying and isolating anomalies, or abnormal patterns, from normal samples. Anomalies are often defined as a pattern that does not conform to expected normal behavior [CBK09]. In the context of the MNIST dataset, we look for images of digits that differ significantly from the norm, such as those with unusually thick or thin strokes, or digits that are rotated. The norm here being the generic representation of the label "3" in the dataset. Figure 3.6 displays five examples each of outliers and typicals from the dataset.

Isolation forest

Anomaly detection has been widely studied for decades, with various methods developed for identifying these unusual patterns. Machine learning techniques, which can be either supervised or unsupervised, are among the most used methods.

Supervised anomaly detection expects the training data to be labeled as normal or anomalous. This approach poses challenges such as the difficulty of obtaining correctly labeled data—particularly for the MNIST dataset, where such labels do not preexist. If we were to label the data ourselves, it could introduce bias, which is undesirable. Additionally, anomalies are typically rare compared to normal data, leading to imbalanced datasets that can result in poor model performance [CBK09].

Unsupervised anomaly detection, on the other hand does not require labeled data for training, which simplifies its use. Instead, it relies on the assumption that normal samples outnumber anomalies, which is usually the case with the MNIST dataset, we opt for unsupervised techniques. The common unsupervised approach is clustering-based, where data is grouped into clusters. This method presupposes that normal data forms large, dense clusters, while anomalies are found in smaller or sparse clusters [CBK09]. However, due to the computational demands of clustering algorithms, especially with large, high-dimensional data like MNIST (784 dimensions per sample), this method becomes impractical for large-scale experiments due to its slow processing speed. Because of this, another method has to be chosen and the choice has become an isolation forest.

An isolation forest is a unsupervised method that works based on the principle of isolation. Isolation means that anomalies will be separated from all the normal samples. To do this isolation forest do not use distance or density measures as clustering methods do. This omission of density calculations significantly accelerates the algorithm's execution time, making it well-suited for high-dimensional datasets such as MNIST. The effectiveness of isolation forests stems from their utilization of the inherent sparsity and distinctiveness of anomalies within the dataset [LTZ09].

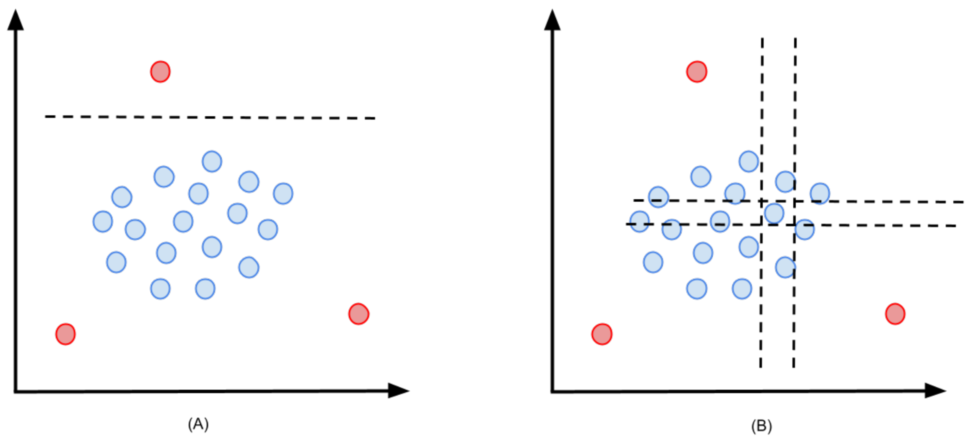


Figure 3.7: Subfigure A demonstrates the Isolation Forest algorithm applied to a small dataset containing three anomalies, marked by red dots. In this figure, a single split isolates one anomaly, illustrating the algorithm's sensitivity to unusual data points. In contrast, subfigure B shows the isolation of a normal data point, which requires multiple splits to be isolated. This distinction is reflected in the tree structure: the depth for the anomaly in subfigure A is one, whereas the depth for the normal data point in subfigure B is four. Image replicated from [LTZ09].

The algorithm constructs a series of isolation trees, each initiated from a randomly selected feature and a corresponding split value, which is determined randomly between the minimum and maximum values of that feature. This splits up the data into two groups, which will be initial child nodes of the isolation tree. This random selection of feature and feature value is repeated until all data points are isolated into individual leaf nodes. The path length from the root to a leaf indicates the number of splits required to isolate a sample, serves as a metric for anomaly detection. Anomalies, being fewer and inherently different, typically demonstrate shorter path lengths, indicating fewer partitions necessary for isolation. This principle can be seen in figure 3.7, where in the left graph the isolation of an anomaly is shown and on the right the isolation of a normal data sample. As shown the isolation of the anomaly takes one step, which means in the isolation tree, the anomaly will have a path of length one from the root. When multiple trees collectively produce a shorter path for specific nodes, they are likely to be anomalies [LTZ09].

A key aspect of the isolation forest algorithm is its random selection of features and split values. This randomness means that the algorithm might produce different results each time it's run on the same data. However, in our experiments, this variability is actually advantageous. Our objective is to generate a wide variety of datasets, and the inherent randomness of the algorithm helps achieve this by mixing up the data in different ways each time.

Benefits and risks

The debate on whether outliers should be removed from datasets is ongoing, with valid arguments both for and against their removal. The presence or absence of these anomalies can significantly impact a model's decision-making process. There are key benefits to removing outliers from a dataset:

1. **Improved Data Quality:** Anomalies are abnormal data points. Removing these can significantly enhance the dataset's quality. The absence of outliers reduces noise and errors, which enhances the dataset's representation of the true underlying patterns, thus improving its overall quality [Lar+19].
2. **Enhanced Model Accuracy:** The performance of machine learning algorithms heavily relies on the quality of the data. High-quality data leads to more reliable decision-making by the model. Anomalies can cause a model to learn from noise rather than the true underlying patterns, leading to an inaccurate representation of data. This often results in a model that overfits noise, failing to perform well on new, unseen data [RB21].
3. **Simpler Models:** Datasets filled with anomalies often require more complex models to accurately classify these unusual points. Removing outliers can simplify the decision boundaries, leading to simpler, more straightforward models. These models are easier to understand, and often more generalizable.

While removing outliers from datasets can improve the simplicity and accuracy of models, it's important to consider the potential downsides:

1. **Loss of Realistic Extremes:** Sometimes, what appears as an outlier might actually represent extreme but realistic variations within the subject matter. Removing these points could make the dataset less representative of all possible scenarios, potentially omitting crucial aspects needed for a full analysis.
2. **Introduction of Bias:** Eliminating outliers can inadvertently introduce bias. This happens especially if these outliers are systematically linked to specific features or outcomes. Such bias can distort the model's interpretation of the data, leading to inaccurate predictions that don't reflect real-world diversity.
3. **Poor Performance on Unseen Data:** If a model is trained on a dataset that excludes outliers, it may struggle to perform well on new datasets that include such extremes. This lack of preparedness can lead to errors when the model is applied in real-world conditions that differ from the training data.

Outliers in the MNIST dataset

Our experiment explores the effects of including different amounts of outliers in the training data, with settings that vary from 0% to 100% of outliers and typical data points. This approach allows us to see how models behave under various conditions: some models are trained with many outliers, others with none, and some with a mix of both typicals and outliers. The outcome is a diverse set of models. Some perform well on normal instances of the digit “3”, while they may struggle with nearly unreadable versions of it. This creates a broad spectrum of model performance, with some excelling with outliers and others doing better with typical data points.

An essential aspect of our approach is the use of an isolation forest for each digit label, which helps us identify anomalies within each specific group of digits. This method is more efficient than applying the isolation forest across all data, which could slow down the process and make it difficult to pinpoint all anomalies. For example, a “3” that closely resembles a “9” might be mistakenly identified as an anomaly if we considered the entire dataset, but when analyzing only the “3”s, it’s correctly recognized as typical for that group.

To understand how many outliers exist in the dataset, a model was developed using only the outlier data extracted from the entire dataset. The results indicated that 13% of the dataset consisted of outliers. Additionally, these outliers were not uniformly distributed across the various categories or labels. This information is detailed in figure 3.8, which provides a visual summary of how many outliers there are and how they are spread across different labels. The uneven distribution could be due to the inherent randomness of the isolation forest method or from the unique ways numbers are written. For example, the number “1” is often just a simple straight or slightly curved line, less affected by individual handwriting styles. In contrast, more complex numbers like “0” or “8” might vary more significantly with handwriting, making them more susceptible to being classified as outliers.

3.3.2 Color augmentation

Color augmentation is a method that enhances image data by modifying pixel values to create varied data instances. This process helps models become more resilient to changes in lighting or color balance. In our experiments, we applied two specific techniques: color inversion and contrast adjustments. Techniques like color channel swapping, which might reorder the channels in a color image from RGB to BGR, are irrelevant to our grayscale images and were not considered.

Color inversion

Color inversion is a simple yet effective augmentation technique. In this method, each color in an image is swapped with its opposite within the color model used. Color models are the numeric representation of colors, such as RGB and grayscale. What color inversion essentially means is that light colors will become dark and dark colors become lighter.

In a grayscale color model like the MNIST dataset, colors are limited to one dimension, where each pixel ranges from 0 (black) to 255 (white). Inverting grayscale images is done by subtracting the current pixel value from 255, this transforms light colors to dark and vice versa. For example, in the MNIST dataset, which consists of white digits on black backgrounds, inversion changes this to black digits on white backgrounds. The effect of this transformation is illustrated in figure 3.9, showing a side-by-side comparison of an original and inverted MNIST sample.

In our experiments, we vary the proportion of inverted images in the dataset from 0% to 100% in 20% increments. This variation allows some models to train with predominantly original images while others see more inverted images, enhancing their ability to recognize digits across different color schemes.

Contrast adjustments

Contrast adjustment is another color augmentation technique used alongside color inversion. It adjusts how distinctly the light and dark colors stand out in an image. Increasing the contrast makes the light and dark colors more distinct by brightening the colors, whereas decreasing it makes these differences subtle, resulting in a more uniform and faded image.

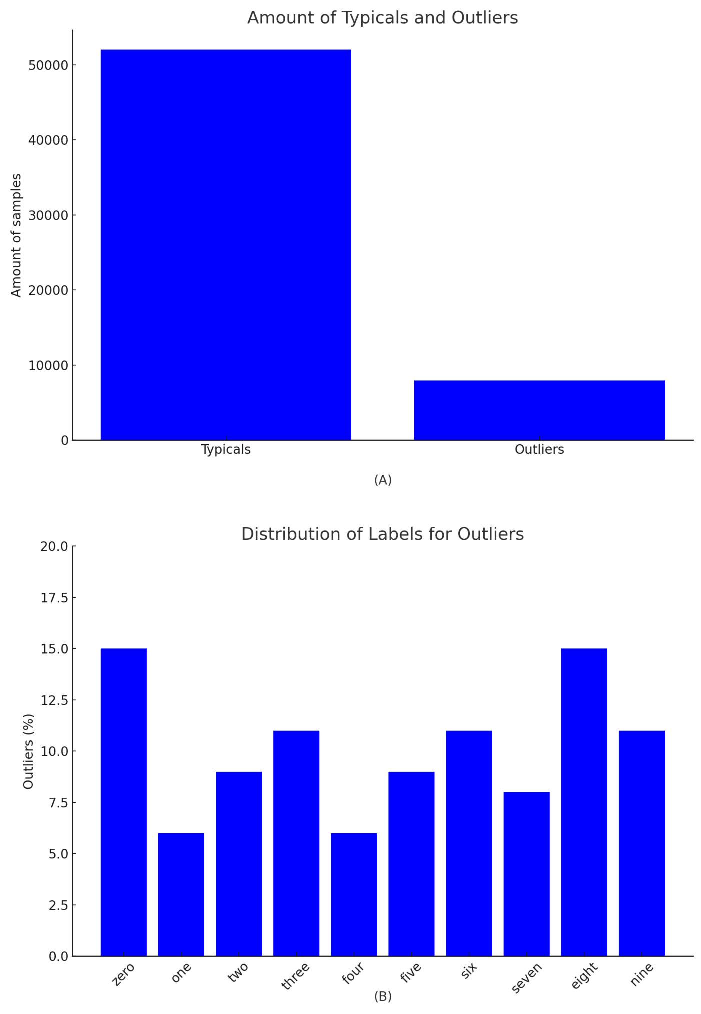


Figure 3.8: This figure provides a visual summary of the distribution of typical data points and outliers in the MNIST dataset. Subfigure A shows that typicals constitute the majority, numbering 52,095, while there are 7,941 outliers, making up approximately 13% of the dataset. Subfigure B further explores the outliers, illustrating how the labels are distributed unevenly among them. Notably, the count of label “1” is almost one-third that of label “0”, indicating a significant imbalance in representation among the outlier labels.

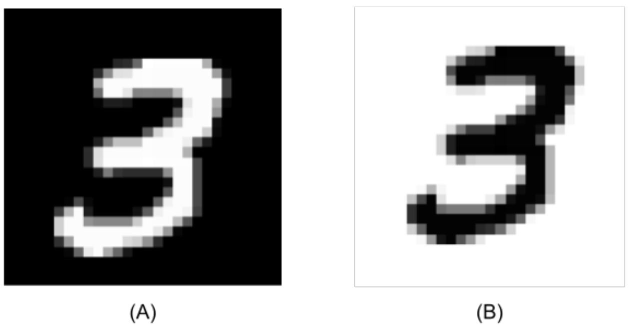


Figure 3.9: The impact of color inversion on a sample from the MNIST dataset. In subfigure A the original image with the white number and black background is shown. Subfigure B represents the inverted version of the same image, where the color values are reversed, resulting in a white digit on a black background.

For grayscale images like those in the MNIST dataset, contrast can be adjusted by multiplying pixel values by a contrast factor. This factor is a float value between zero and two, if the value is one, the contrast will stay the same. A factor of one maintains the original contrast, values below one reduce the contrast—potentially to a minimum threshold of 0.2 to prevent all pixels from becoming uniformly black—and values above one enhance the contrast up to a maximum of 1.8 to avoid excessive whitening that could obscure critical image features. The effects of contrast adjustments on an MNIST sample are depicted in figure 3.10: decreased contrast on the left, original in the middle, and increased contrast on the right.

As with the color inversion, we vary the proportion of contrast adjusted images in the dataset from 0% to 100% in 20% increments. Which allows some models to train with mostly increased or decreased contrasted images, while others see a smaller amount of these, resulting in different model behaviour.

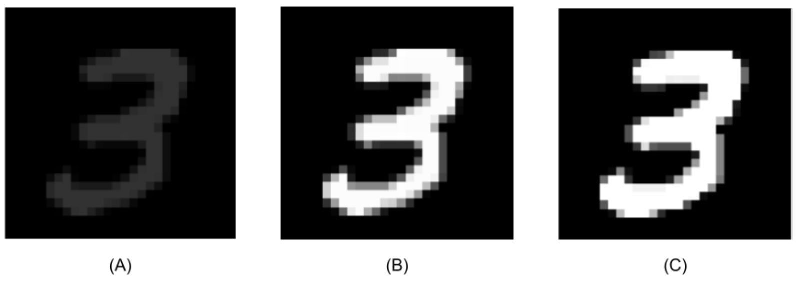


Figure 3.10: The effect of changing the contrast of a sample out of the MNIST dataset, decreased contrast (A), the original (B) and increased contrast (C). Subfigure A shows the effect of setting the contrast factor to 0.2, which reduces the difference between light and dark areas to a minimum, resulting in an almost uniformly gray image. In subfigure C the contrast factor is set to 1.8, which enhances the distinction between the digits and their background. However, this high contrast setting slightly erases the gray areas along the digit edges.

3.3.3 Position augmentation

Position augmentation is another valuable data augmentation technique used to enhance the training of machine learning models. Unlike color augmentation, which alters pixel values, position augmentation changes where pixels are located in an image. This technique includes rotating or shifting the positions of objects within images to help models better understand and predict different spatial arrangements. Such training can improve model performance in

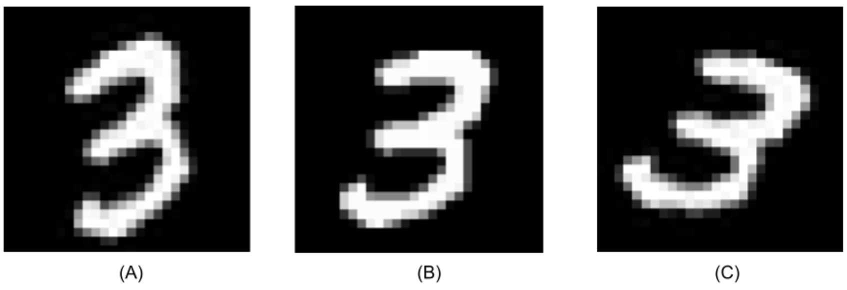


Figure 3.11: The effect of rotating a sample out of the MNIST dataset, left rotation (A), the original (B) and right rotation (C). Subfigure A shows a rotation of 20 degrees, which rotates the number to the left. In subfigure C the number is rotated -20 degrees, resulting in a rotation to the right.

real-world situations where objects may not be perfectly aligned or oriented. In this thesis, we focus on using rotations and translations, while other methods like shearing and flipping were not included.

Rotating

Rotation or image rotation, as a form of geometric transformation, involves repositioning image pixels based on a rotation matrix that applies to each pixel's coordinates, facilitating the counterclockwise rotation by a specified angle [Van22]. In equation 3.1, this rotation matrix $\begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}$ is applied on the matrix $\begin{bmatrix}x\\ y\end{bmatrix}$, which represents a pixel's x and y value of the original image [Wei03]. The result of this matrix multiplication is the new matrix $\begin{bmatrix}x'\\ y'\end{bmatrix}$, which is the pixel's new position. This calculation is then applied to each pixel in the image. An important note is that the angle is defined in radians. Figure 3.11 illustrates this by showing a sample from the MNIST dataset rotated to both the left and the right in addition to the original configuration.

$$\begin{bmatrix}x'\\ y'\end{bmatrix} = \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix} \cdot \begin{bmatrix}x\\ y\end{bmatrix} \tag{3.1}$$

When rotating an image, parts of it might extend beyond the original frame. To manage this, our thesis opts to fill these out-of-bounds areas with the same value, without any interpolation, preserving the image's original dimensions. Additionally, to avoid misinterpretation of numbers, the experiment restricts rotations between -20 to 20 degrees in 5-degree increments, to prevent misidentification of numerals, a critical consideration illustrated in figure 3.12. Here, an excessively rotated "6" misleadingly appears similar to a "9". This precaution ensures the integrity of the dataset's real-world applicability in handwritten digit recognition. Rotational variation within this experiment ranges from 0% to 100% in 20% increments, enabling an extensive evaluation of model adaptability across different rotational contexts.

Translating

Translation refers to the shifting of the entire image by a certain amount of pixels in one or more directions. In image processing, such as in this experiment this means moving the image up, down, left or right. In practice, this is done by moving each pixel $P(x, y)$ over a certain distance d_x for horizontal and d_y for vertical translations. The pixel position after translation is defined as $P(x', y')$. A translation for a single pixel is defined in equation 3.2, important to note that if only a horizontal translation is needed d_y is zero and if only a vertical translation

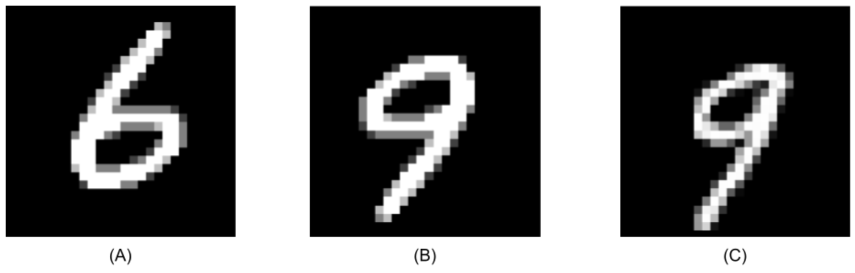


Figure 3.12: A representation of problematic rotations in the MNIST dataset. Subfigure A shows the original image, and Subfigure B shows the same image rotated 180 degrees. This rotation makes Subfigure B look nearly identical to another dataset sample labeled as "9", shown in subfigure C. Due to this similarity, models may incorrectly identify both images as the same digit, leading to undesirable errors in digit recognition.

is needed d_x is zero. A negative distance will result into a movement to the left or upward, a positive distance will result into a movement to the right or downward [Van22].

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{3.2}$$

When translating images, we must consider what happens to pixels that shift outside the original boundaries and those that shift into the frame. Pixels that move outside the image borders are removed, which can be problematic as significant parts of the image, like parts of digits, might be lost. This issue is illustrated in figure 3.12, where essential features of digits are cut off, making it hard to distinguish the digit. To prevent this, we limit translations to a maximum of two pixels in any direction. Conversely, pixels that enter the image frame due to translation are set to black, under the assumption that these areas represent non-informative background rather than digit defining content.

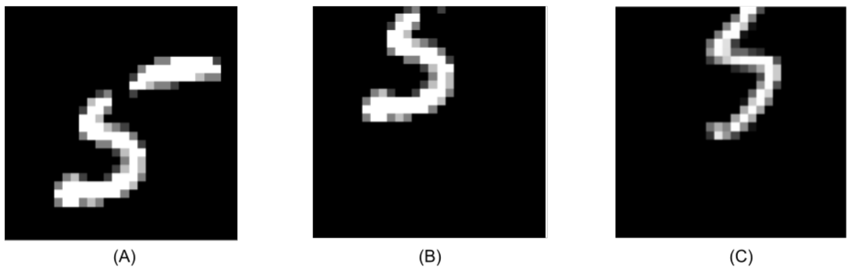


Figure 3.13: The effects of problematic translations in the MNIST dataset. Subfigure A displays the original image, while subfigure B shows the image after a vertical translation. Subfigure C presents a vertically translated image of a different sample representing the label "3". Both subfigures B and C appear almost identical, making it extremely difficult for models to distinguish between them. This similarity poses significant challenges in accurately identifying the images after translation.

In our experiments, we treat horizontal and vertical translations as separate processes to provide a diverse range of image movements. This method allows us to move images just horizontally, just vertically, or in both directions but to varying degrees. By not always translating in both directions simultaneously, we create a dataset with a wide variety of image positions. Like with other types of data augmentation, we adjust the frequency of these translations from 0% to 100% in 20% increments. This variability ensures that different models are trained on different types of data, to have that variety in model behaviour.

Others

In our experiments, we evaluated additional position augmentation techniques like flipping and shearing alongside the primary methods. Flipping images horizontally or vertically can increase data variability by mirroring images, which is useful when the object's orientation can vary without altering its fundamental characteristics. However, applying this to digits, such as those in the MNIST dataset, can change the digit's meaning turning a "6" into a "9", for example or result in a unrealistic representation of the digit. These inaccuracies could cause a machine learning model to learn incorrect patterns that are not representative of real-world scenarios. Therefore, we prioritized maintaining accurate digit representation over increasing model robustness with these techniques.

Image shearing involves skewing an image along one axis, which distorts its geometry. Like flipping, shearing can create unrealistic representations of numbers. Since our goal is to accurately represent handwritten digits, we found that shearing disrupts this objective.

3.3.4 In-place and appending

In this experiment, all data augmentation techniques, with the exception of outlier detection, can be categorized into two implementation strategies: in-place or by appending to the dataset. These strategies necessitate distinguishing between two datasets: the "working dataset" and the "original dataset". The working dataset is dynamic; it is where data is augmented and reinserted, growing and diversifying with each applied technique. In contrast, the original dataset remains static, preserving the initial data state, exemplified by white digits on a black background.

In-place augmentation directly modifies a portion of the working dataset. For example, if 50% of the data is selected for augmentation, only that subset is altered and replaced. This method allows for layered augmentations, where newly applied techniques can modify previously augmented data. This can result in complex data transformations, such as inverted samples with a decreased contrast as shown in figure 3.14

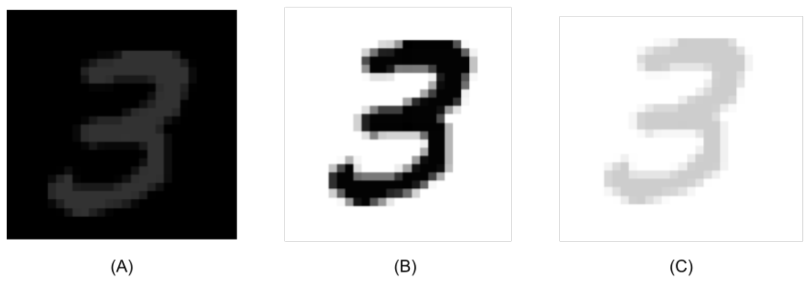


Figure 3.14: Layered Augmentations Demonstrated on an MNIST Sample. Subfigure A illustrates the effect of decreasing contrast, making the digit less distinct against its background. Subfigure B shows the same sample with its colors inverted, creating a visual contrast. Subfigure C combines the techniques shown in A and B, resulting in a digit that is both color-inverted and has reduced contrast, demonstrating the combined impact of multiple augmentations.

A critical limitation of this approach is the potential for complete data replacement if augmentations are applied to 100% of the working dataset, thereby risking the loss of the original data's representational fidelity. To prevent this, appending augmentation was introduced. This method involves applying a technique to the original dataset and then adding the augmented data to the working dataset. This approach maintains a balance, preserving original data while introducing variety through augmented data. For example, applying an inversion to 100% of the data in an appending manner results in a dataset containing both original and inverted

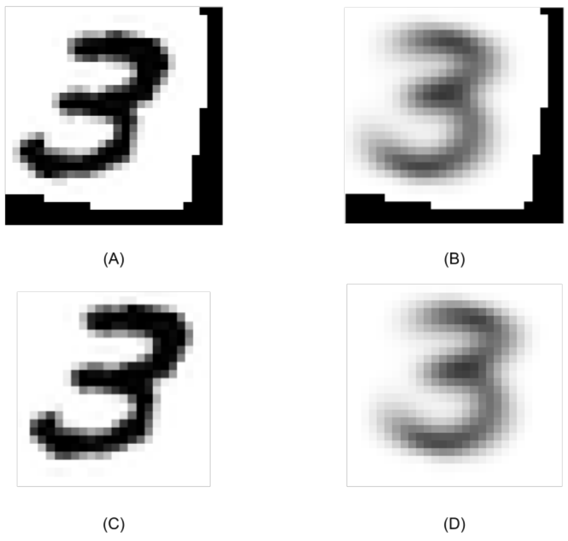


Figure 3.15: The impact of different augmentation sequences on the dataset. Subfigures A and B show results when inverting is performed first, followed by other augmentations such as translation and rotation. In contrast, subfigures C and D demonstrate a consistent augmentation order: starting with translation, followed by rotation, and concluding with inversion. Both B and D depict the average appearance of the digit “3” after these processes, highlighting how the order of operations influences the final image.

samples. Unlike in-place methods, appending does not support layered augmentations but ensures continuous growth of the dataset. Thus all augmentations except outlier detection have the possibility to be in-place or appending so both advantages can be possible.

Outlier detection is exclusively an in-place augmentation technique because it involves categorizing the data into “typicals” and “outliers”. If these categories were simply appended back to the dataset, it would result in duplicates, which are undesirable in machine learning because they can skew the model’s learning process. Once the data is split during outlier detection, both the working and the original datasets are divided accordingly. This means that any subsequent augmentations that use the appending method will not reintroduce data that has been separated out, maintaining the integrity of the learning process.

3.3.5 The order of operations

When numerous augmentations are applied to a dataset and can be layered by applying them in-place, the need arises to consider whether there should be a specific order to these operations. Without an established sequence, augmentations would be applied randomly, leading to several issues: inefficiency in time usage, challenges in reproducibility, and difficulties in comparing results across different models.

The time-related inefficiency is evident when considering the sequencing of specific augmentations, like outlier detection. Outlier detection reduces the dataset’s size, which subsequently decreases the processing time for any augmentations that follow. If outlier detection is applied later, it has to deal with altered data, which complicates the detection process and slows it down.

Reproducibility is jeopardized when augmentations are applied without a consistent order. This randomness precludes the possibility of replicating a specific augmentation sequence, crucial for recreating the same model outputs or implementing controlled variations.

The comparability of different models is hindered when the order of augmentations varies. As illustrated in figure 3.15, identical augmentations applied in different sequences yield distinct results. I In this figure, subfigures A and B show black borders when the sample is inverted because the inversion happens before positional augmentations. As explained in section 3.3.3, new pixels introduced during these augmentations are made black. In inverted images, these black zones are critical as they outline the digit and significantly affect the model's learning process, as the model may mistakenly learn these background patterns as part of the digits. This variability in results, despite identical processing steps, highlights the need for a standardized sequence of operations to ensure reliable model comparisons.

To establish a consistent and efficient process, the following order of operations is applied:

1. **Outlier detection** To minimize time-related inefficiencies, outlier detection is performed first.
2. **Position augmentations** To prevent the introduction of black borders, positional augmentations are carried out before any color augmentations. Translations are done before rotations, but this specific order does not impact the final result.
3. **Color augmentations** To further avoid black borders, color augmentations are the final step, with contrast adjustments happening before inversion.

Chapter 4

Searching for multiplicity in many models

In this part of the experiment, we analyze the models generated in chapter 3 to identify those that are multiplicitous. This chapter describes the process, which includes two main tasks: evaluating the generated models and grouping similar models into Rashomon sets. Figure 4.1 provides an overview of this process, illustrating each component and their connections in this experiment.

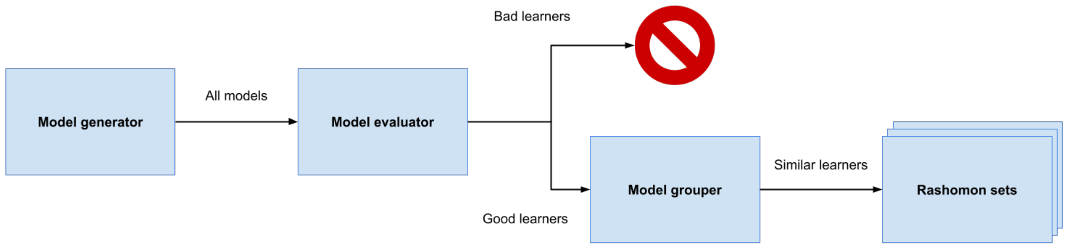


Figure 4.1: An overview of the experiment. It shows the model generator (chapter 3), and the model evaluator and grouper (discussed in this chapter). The evaluator assesses the generated models on a test set, filtering out poor performers. The remaining good learners are grouped into Rashomon sets based on the similarity of their predictions.

4.1 Evaluating the generated models

As specified in Chapter 2, multiplicitous models exhibit comparable predictive accuracy. To quantify this accuracy, the models must be assessed using a test set. By evaluating each model's predictive performance on each individual sample within the test set, we can ascertain similarities. This section details the composition of the test set, the evaluation methodology, and the process for filtering out inadequate learners.

4.1.1 The test set

As described in Chapter 2, multiplicitous models achieve similar accuracy on a test set. To identify these model sets, we must first evaluate their performance. The initial step is to create an appropriate test set.

The data used for evaluation must be distinct from the training data. While the MNIST dataset's predefined test set could be used, we chose to create our own handwritten digit samples

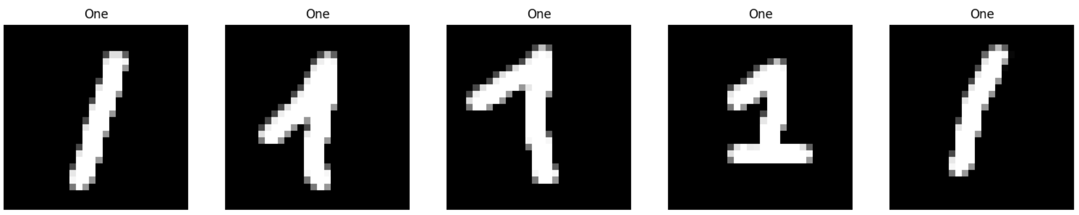


Figure 4.2: Samples of the digit "1" from the test set, demonstrating variety in the data. Some samples are straight lines, similar to the training set. Others have a second diagonal line, which is rare in the training set, and one sample includes a horizontal line, which is never seen in the training set. This variety reflects real-world scenarios where people write numbers differently.

to ensure greater variety. For each digit, we created ten samples, aiming to capture diverse handwriting styles. Figure 4.2 shows examples of the digit "1" to illustrate this variety.

4.1.2 The evaluation approach

A straightforward approach would be to let all generated models evaluate the entire test set and calculate an overall accuracy. However, accuracy only indicates how often the model correctly classifies labels relative to the total predictions. When grouping similar models, relying solely on overall accuracy can be misleading since equal accuracies do not guarantee identical sample classifications. Therefore, we evaluate models based on their individual predictions.

The goal of this method is to group models that predict the correct label for a set of samples, resulting in collections of models that "think" alike. To form these groups, each sample is evaluated by all models one at a time. After predicting a label with an associated accuracy, a filtering process eliminates "bad learners".

All generated models use sparse categorical cross-entropy as their loss function and softmax as the activation function for the output layer, which produces a probability for each label representing confidence. "Bad learners" are models that are unsure of their predictions and essentially guessing, meaning their confidence levels for the predicted label and another label are close. In this experiment, the confidence difference (epsilon) between the predicted label and any other label must be at least 0.1 (or 10%). This ensures that only "good learners" are grouped together.

The evaluation process, including the filtering steps, is described in figure 4.3. This figure presents a flow chart where tasks are depicted as squares, and decision points, such as filtering based on whether the top label's confidence is close to the second, are represented as yellow diamonds.

Excluding bad learners is crucial because their uncertain predictions could distort the results, making it difficult to identify models that genuinely think similarly. This uncertainty might arise because the models did not converge to a minimum or the training data was too diverse for them to learn clear patterns. If the model is a "good learner" and predicts correctly, its ID is stored in a bucket that holds models that predicted the same label for that sample. After each model has been evaluated on each sample, the result is a collection of buckets containing models that confidently predicted the same correct label. The way these buckets are created for an arbitrary sample is shown in figure 4.4.

4.2 Grouping of similar answers

These buckets of similar acting models are based on sample specific predictions and are not a sign of model multiplicity. For this to be true, they have to be in a large amount of buckets as

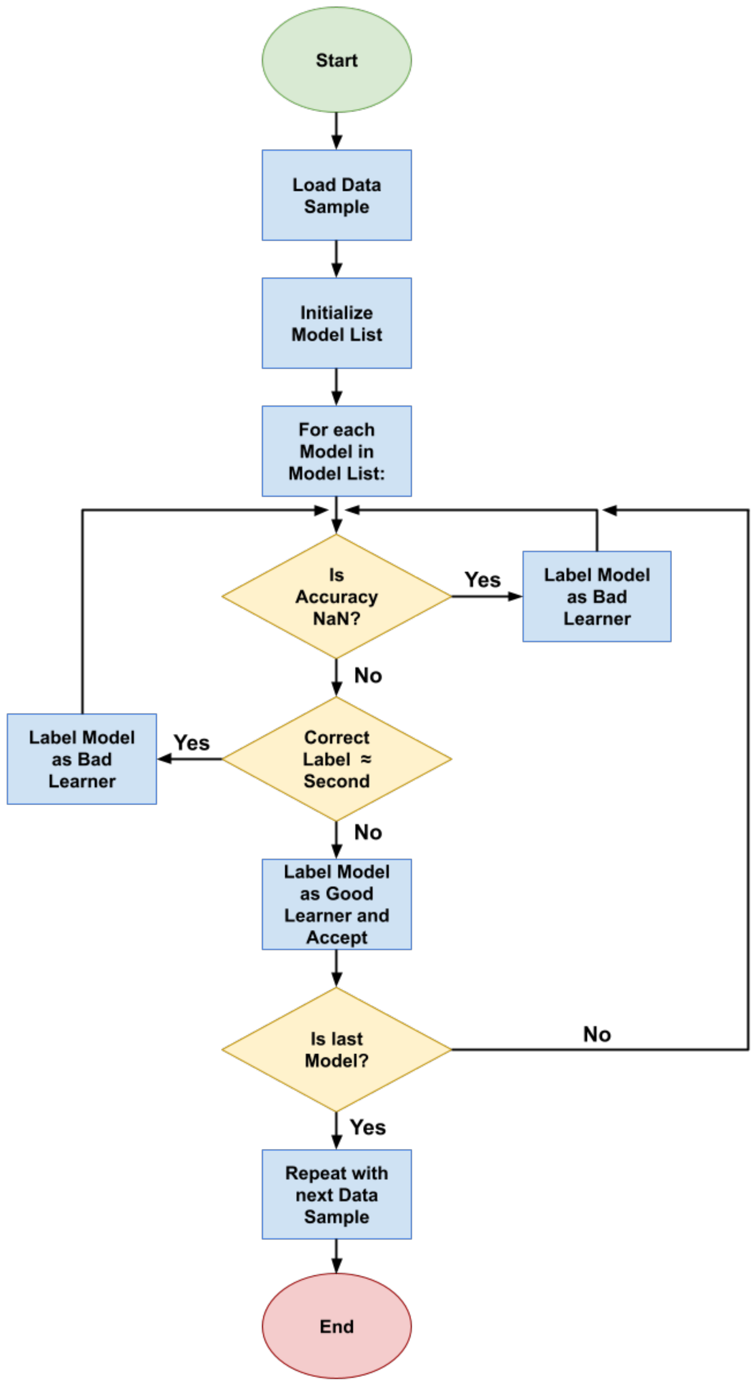

Start
Load Data Sample
Initialize Model List
For each Model in Model List:
Is Accuracy NaN?
Yes
Label Model as Bad Learner
No
Label Model as Bad Learner
Yes
Correct Label ≈ Second
No
Label Model as Good Learner and Accept
Is last Model?
No
Yes
Repeat with next Data Sample
End

Figure 4.3: Flowchart depicting the model evaluation process. The process starts by loading a data sample and the generated models into memory. Each model is evaluated in sequence to check if the accuracy is NaN (Not a Number), in which case the model is labeled as a bad learner. If the probability of the most confident label is close to the second most confident label, the model is also labeled as a bad learner. Otherwise, the model is labeled as a good learner and the predicted label is accepted. This means the model is added to a bucket containing all models that predicted that label for the specific sample. This process is repeated for each model and each new data sample.

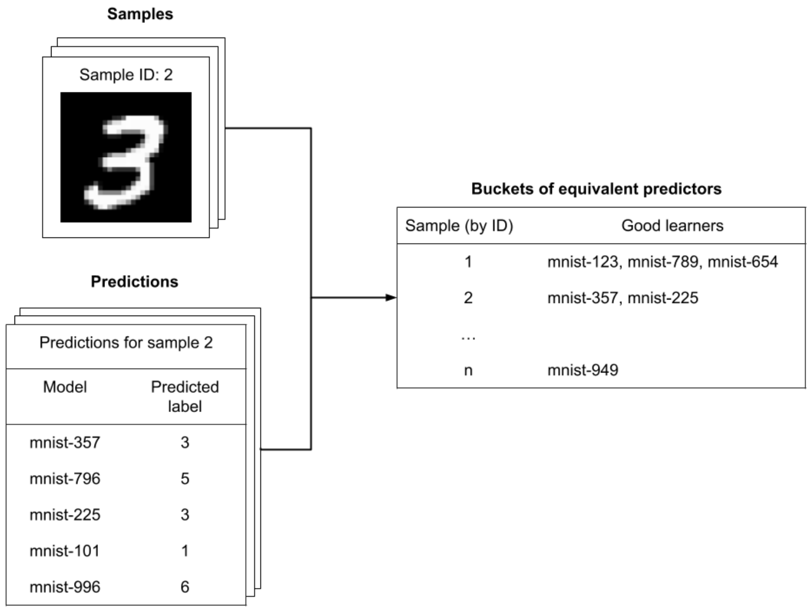


Figure 4.4: The creation of buckets that contain “good leaners” that give a sample the same correct label. In this example a sample with label “3” is being predicted by five models in which two of them give it the correct label and have a high enough confidence to be “good learners” are grouped in the bucket of that sample.

will be the same as similar predictive reasoning. To find if these buckets are often or frequent paired with each other we have to determine the amount of times they are grouped and form a baseline threshold to be viewed as frequent pairs. This is essentially an example of the mining of frequent itemsets problem often found in data mining [Bor12].

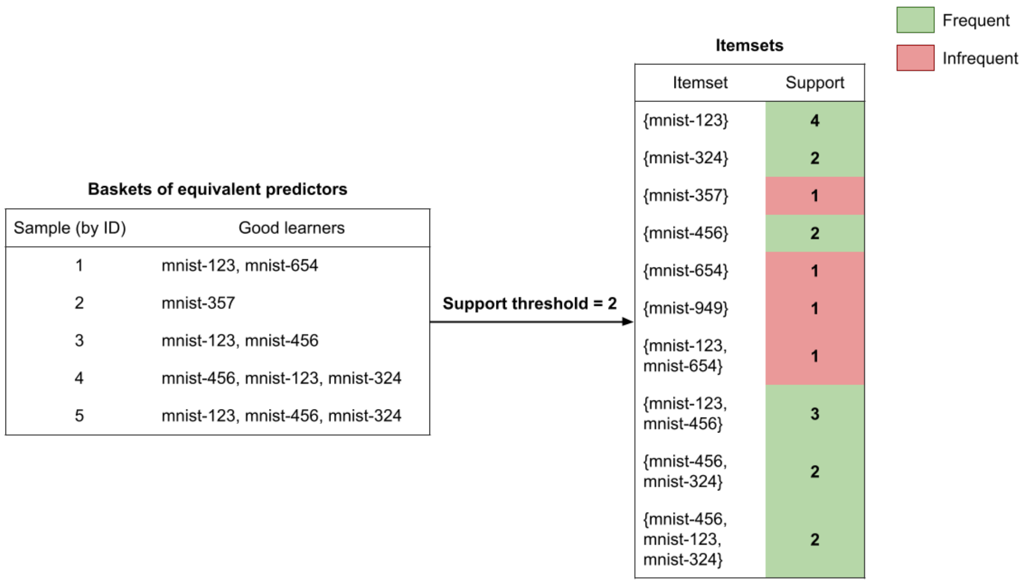
4.2.1 The frequent itemsets problem

Before we can define the frequent itemset problem, we need to explain the Market-Basket model. This data model comes from supermarket shelf management and defines relationships between two kinds of objects. First, there is a large set of items (e.g., products sold in a supermarket). Second, there are baskets (e.g., items bought by one customer in the store), which are small subsets of items called itemsets [LRU20].

Frequent itemsets are sets of items that appear in many baskets. To define ”many,” we introduce a support threshold (s). The support for an itemset is the number of baskets in which that itemset appears. An itemset is considered frequent if its support is at least the support threshold [LRU20].

In our experiment, we apply the Market-Basket model by treating the generated models as items and the buckets of equivalent predictors created during the evaluation process as baskets. To clarify, figure 4.5 shows an example of how equivalent predictors can be classified as frequent or infrequent based on a support threshold of two.

In our experiment, baskets are stored in files. To find the support of all possible itemsets, we need to count occurrences as the data is read. A naive approach is to make one pass over the basket data, creating all subsets for every item and increasing the corresponding count. Since we want to find all frequent itemsets of all sizes, we also generate pairs of itemsets. However, this approach leads to a memory problem, as it creates 2^{items} subsets. For a large number



Baskets of equivalent predictors

Sample (by ID)	Good learners
1	mnist-123, mnist-654
2	mnist-357
3	mnist-123, mnist-456
4	mnist-456, mnist-123, mnist-324
5	mnist-123, mnist-456, mnist-324

Itemsets

Itemset	Support
{mnist-123}	🟩 4
{mnist-324}	🟩 2
{mnist-357}	🟥 1
{mnist-456}	🟩 2
{mnist-654}	🟥 1
{mnist-949}	🟥 1
{mnist-123, mnist-654}	🟥 1
{mnist-123, mnist-456}	🟩 3
{mnist-456, mnist-324}	🟩 2
{mnist-456, mnist-123, mnist-324}	🟩 2

Figure 4.5: The formation of itemsets involves creating combinations of items and counting how often they appear in the baskets of equivalent predictors. The support threshold is set to two, which means that itemsets appearing in at least two baskets are considered frequent. In this example, six itemsets are labeled as frequent (green), and four are labeled as infrequent (red).

of items, this won't fit into main memory, resulting in disk thrashing and slower run times [LRU20].

To solve this issue, algorithms use the concept of "monotonicity". This means that if a set of items is frequent, then every subset of that itemset is also frequent. For example, let's define two itemsets, J and I, where $J \subseteq I$. This means every basket containing I must also contain all items in J, so the support of J is at least the support of I. If I meets the support threshold s, then J will also meet it, making both I and J frequent [LRU20]. In practice, monotonicity reduces the need to generate candidate pairs exhaustively because if an item doesn't meet the support threshold, no superset of that item can be frequent. The algorithm to find frequent pairs used in this thesis is the popular A-Priori algorithm, which is discussed in the next section.

4.2.2 A-Priori algorithm

The A-Priori algorithm is a two-pass algorithm designed to reduce the number of item pairs to count, though it requires making two passes over the data. It leverages the concept of monotonicity to minimize the number of pairs that need to be generated. Before running the algorithm, the support threshold must be set. This threshold is domain-specific and can vary, but for our purposes, we aim for it to be relatively high. The outcomes of applying different thresholds are discussed in section 4.3.

Finding frequent pairs

In the first pass of the A-Priori algorithm, two tables are created. The first table is a map that translates model names to integers, reducing the memory required to store strings. The second table keeps the counts for all individual items, using the integers from the first table as indices. As baskets are read, each time an item is encountered, its counter is incremented by one.

After the first pass, we can examine the support for each individual item (singleton), identifying many infrequent singletons. The results are then transferred to a new table, which retains the unique integer mapping for each item but sets the count for each infrequent item to zero while preserving the support for frequent items [LRU20].

The second pass of the A-Priori algorithm involves looping over the baskets again, but this time it only counts pairs composed of two frequent items. First, pairs are generated by looking in each basket and creating all possible pairs of frequent singletons found in that basket. This process is repeated for each basket [LRU20].

During the loop over all baskets, the frequency of all generated pairs is counted. Because of the principle of monotonicity, no frequent pairs will be missed, as a pair cannot be frequent if one of its items is infrequent. By pruning infrequent items, the memory required to store the item pairs is reduced since only frequent items need to be stored. At the end of this pass, the pairs are evaluated based on their support to determine if they meet the support threshold to be considered frequent itemsets of size two [LRU20].

Finding all frequent itemsets

The same process used for finding itemsets of size two can be expanded to find itemsets of a larger size (k). In the A-Priori algorithm, there will be a pass for each set size k. This means that to find triplets instead of pairs, three passes are needed instead of two. This small adjustment allows the algorithm to search for frequent itemsets of all sizes, stopping when there are no frequent itemsets of a certain size. Due to monotonicity, if there are no frequent itemsets of size k, there will be no frequent itemsets of size $k + 1$, ensuring that no frequent itemsets are missed [LRU20].

The process of changing from size k to $k+1$ in the A-Priori algorithm involves two steps. First, a set of candidate itemsets of size k is created. These itemsets need their support counted to determine if they are frequent. If they are frequent, they are added to another set that represents the frequent itemsets.

In the second step, these frequent itemsets of size k are used to generate candidate itemsets of size $k + 1$. These candidate itemsets are then evaluated in the next pass to check if they are frequent. This process is repeated until the value of k is so high that no frequent itemsets of size k are found [LRU20]. The complete process of finding itemsets of size k using the A-Priori algorithm is illustrated in a flowchart in figure 4.6.

4.3 The results

Using the methods outlined in chapter 3, we generated 500 random instances of neural networks trained on the MNIST dataset. These models differ in their training data and hyperparameters, creating a diverse sample. In this section, we discuss the results of the evaluation process and the A-Priori algorithm applied to these 500 models.

4.3.1 The evaluation process

As discussed in section 4.1, the models are evaluated on a handwritten MNIST-like test set containing ten samples for each digit. The evaluation results in baskets of equivalent predictors, which are models that predict the same label for a given sample, excluding bad learners. Bad learners are models that are not confident in their predictions, indicated by the confidence level of the predicted label compared to other labels. If the confidence difference between the predicted label and any other label is less than 0.1, the prediction is filtered out. Table 4.1 shows the frequency of models being unsure about their predictions. The first column lists ranges of “bad learner” occurrences, indicating how often a model is unsure. The second column shows the number of models in each range, and the third column shows the percentage of models in each range. For example, the second row indicates that 255 models made between 1 and 25

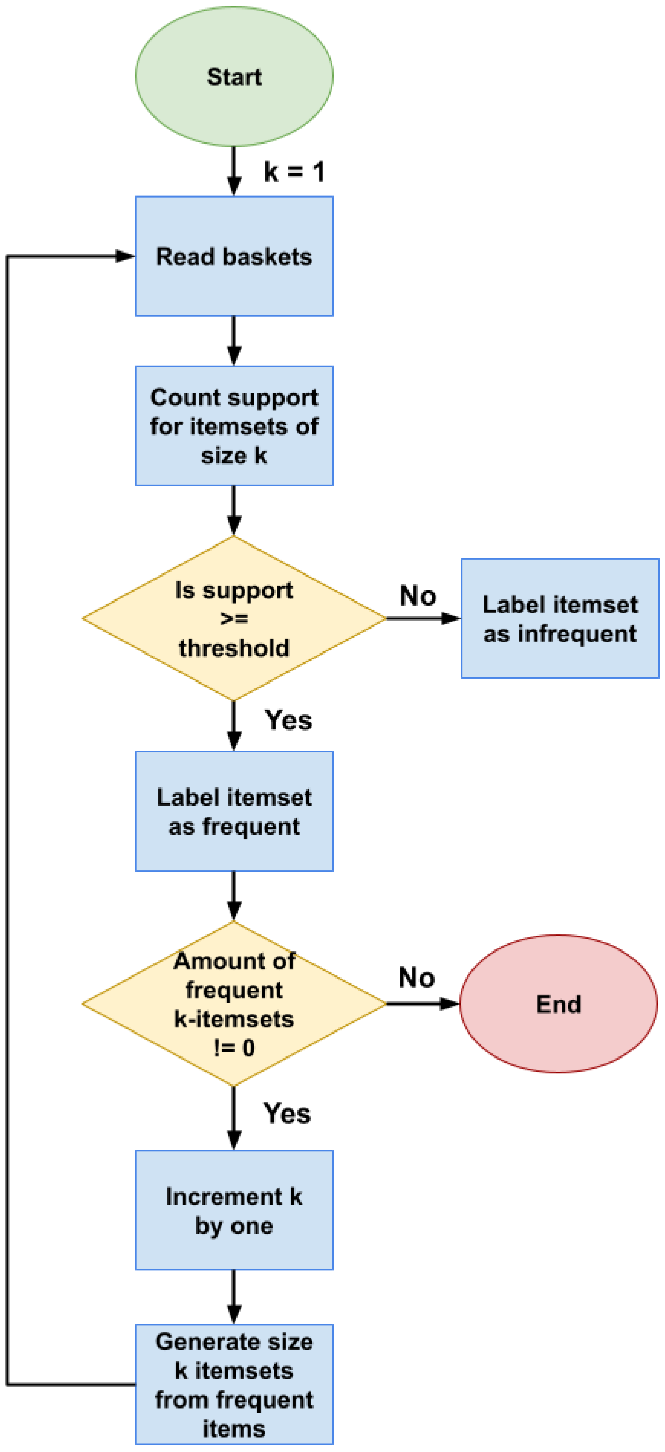

Start
k = 1
Read baskets
Count support for itemsets of size k
Is support >= threshold
No
Label itemset as infrequent
Yes
Label itemset as frequent
Amount of frequent k-itemsets != 0
No
End
Yes
Increment k by one
Generate size k itemsets from frequent items

Figure 4.6: Flowchart depicting the A-Priori algorithm. The algorithm begins by counting the support of itemsets of size k. Itemsets that do not meet the support threshold are labeled as infrequent and, due to monotonicity, are disregarded for the rest of the algorithm. Itemsets with support greater than or equal to the threshold are labeled as frequent. If no frequent itemsets exist, the algorithm stops, as there can be no frequent itemsets of a larger k due to monotonicity. If frequent itemsets exist, k is incremented by one, and candidate itemsets of size k are generated. The process then repeats, with the support of these new itemsets being counted. This continues until there are no frequent itemsets.

Amount of excluded predictions	Amount of models	Percentage of models
0	79	15.8%
1-25	255	51%
26-50	27	5.4%
51-75	22	4.4%
76-99	13	2.6%
100	104	20.8%
Total	500	100%

Table 4.1: Table of bad learner occasions. The first column indicates the frequency ranges for excluded predictions, starting from no uncertain predictions and increasing in steps of 25 up to 100 (all) samples. The second column shows the number of models that fall within these ranges. The third column displays the percentage of these models relative to the total number of models, providing a clear view of how many models fall into each frequency range compared to the entire collection.

unsure predictions across the entire test set, meaning they are usually very confident in their predictions.

These results indicate a clear difference in model quality. Most models are quite confident in their predictions, suggesting they converged well. However, 20.8% of the models are completely unsure of their predictions, effectively guessing 100% of the time.

The length of the baskets of equivalent predictors represents the number of models that predicted the same correct label for a sample. Larger baskets indicate greater agreement among the models. Figure 4.7 shows a boxplot illustrating the distribution of basket lengths. The average basket length is 156, meaning that on average, 30% of the models predict the same label for a sample. It is important to note that only good learners contribute to these baskets. Considering the results from table 4.1, 104 models are always bad learners, increasing the 30% agreement rate to around 40%.

However, some samples have small basket lengths, indicating that most models make mistakes on these samples. This could be due to these samples having features that are underrepresented during training or resembling other labels. Figure 4.8 shows a confusion matrix that depicts the frequency of misclassifications, including the frequency of bad learners per label. An interesting misclassification is for label “7”, as these samples are more commonly labeled as the digit “3”. This makes sense as the models might see the second horizontal stripe as a deciding feature for both labels. Another notable aspect of this confusion matrix is that samples labeled “7” and “9” are more often misclassified than correctly classified.

4.3.2 The grouping process

The baskets generated by the evaluation process is not a direct sign of model multiplicity, to accomplish this frequent itemsets have to be found. To accomplish this the A-Priori algorithm is applied to the baskets. As we are looking for models models that act as similar as possible, the support threshold should be a relative high value. To experiment what different thresholds result in multiple thresholds have been tried out. As discussed in section 4.2.2, the algorithm will find itemsets of different sizes which means that multiple models have acted similar on the test set, as we are looking for multiple models we are not interested in frequent singletons. In table 4.2, the amount of pairs, triplets, quadruplets, quintuplets and sextuplets found by A-Priori using different support thesholds. The lower the support threshold the more itemsets of larger sizes are labeled as frequent, if the threshold is higher the amount of frequent itemsets of all sizes decline. As we are looking for very similar models lower thresholds such as 60% or 65% are not good enough to filter out the less similar models. The threshold of 80% returns no results, which indicates that no models are labeling the same samples correctly with confidence

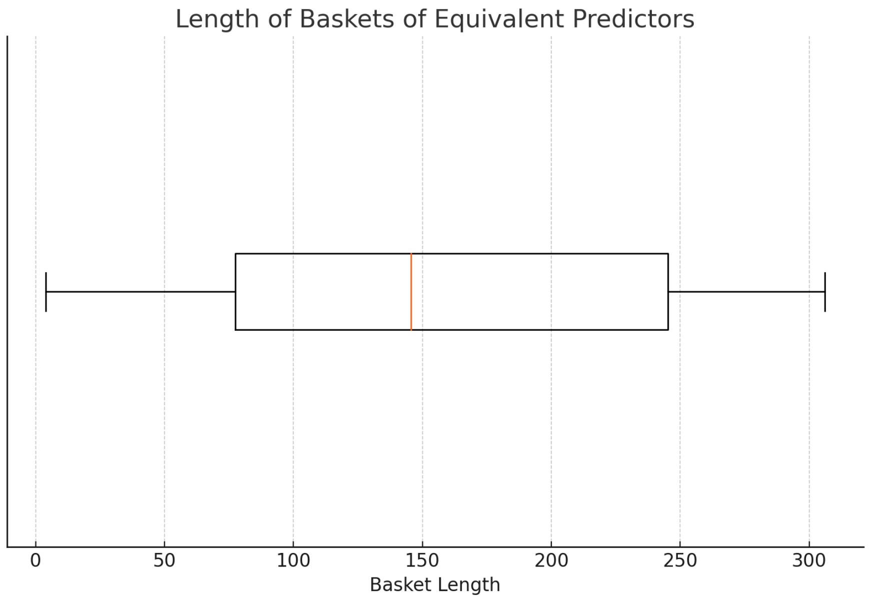


Figure 4.7: Box plot illustrating the lengths of 100 baskets which are the results of the evaluation step. Each basket represents a collection of models that has predicted the same label for a specific sample of the test set. The x-axis represents the basket length , indicating the number of models within each basket.

80% of the time, thus this threshold is also not considered any further. The results from 70% and 75% tresholds are interesting as they do not return large amounts of frequent itemsets but only a select few, which have to further explored.

The baskets generated by the evaluation process are not a direct indication of model multiplicity. To achieve this, frequent itemsets must be identified using the A-Priori algorithm. Since we aim to find models that behave as similarly as possible, the support threshold should be relatively high. To explore the effects of different thresholds, various values were tested. As discussed in section 4.2.2, the algorithm finds itemsets of different sizes, meaning multiple models have similar behavior on the test set. We are not interested in frequent singletons. Table 4.2 shows the number of pairs, triplets, quadruplets, quintuplets, and sextuplets found by A-Priori using different support thresholds. Lower support thresholds result in more frequent larger itemsets, while higher thresholds reduce the number of frequent itemsets of all sizes. Lower thresholds, such as 60% or 65%, are insufficient to filter out less similar models. The threshold of 80% returns no results, which indicates that no models are labeling the same samples correctly with confidence 80% of the time, making this threshold unsuitable. The results from the 70% and 75% thresholds are notable as they return only a select few frequent itemsets. The groups of models in each of the itemsets with a high support threshold are comparable in their predictive behavior and could be viewed as Rashomon sets.

4.3.3 Are models in frequent itemsets similar?

The A-Priori algorithm identifies frequent itemsets that contain models exhibiting similar predictive behavior on the test data. These models, generated using the method described in chapter 3, vary in their data composition and hyperparameters used for training. These differences are logged in the delta component (section 3.1.5) and a compact version called the compressed delta. This compressed delta, with one-hot encoded parameters, allows us to compare the differences between models within the same frequent itemset. In this experiment, we compare models by calculating the distance between their corresponding compressed delta vectors. By analyzing the distances between compressed deltas of similar models, we can po-

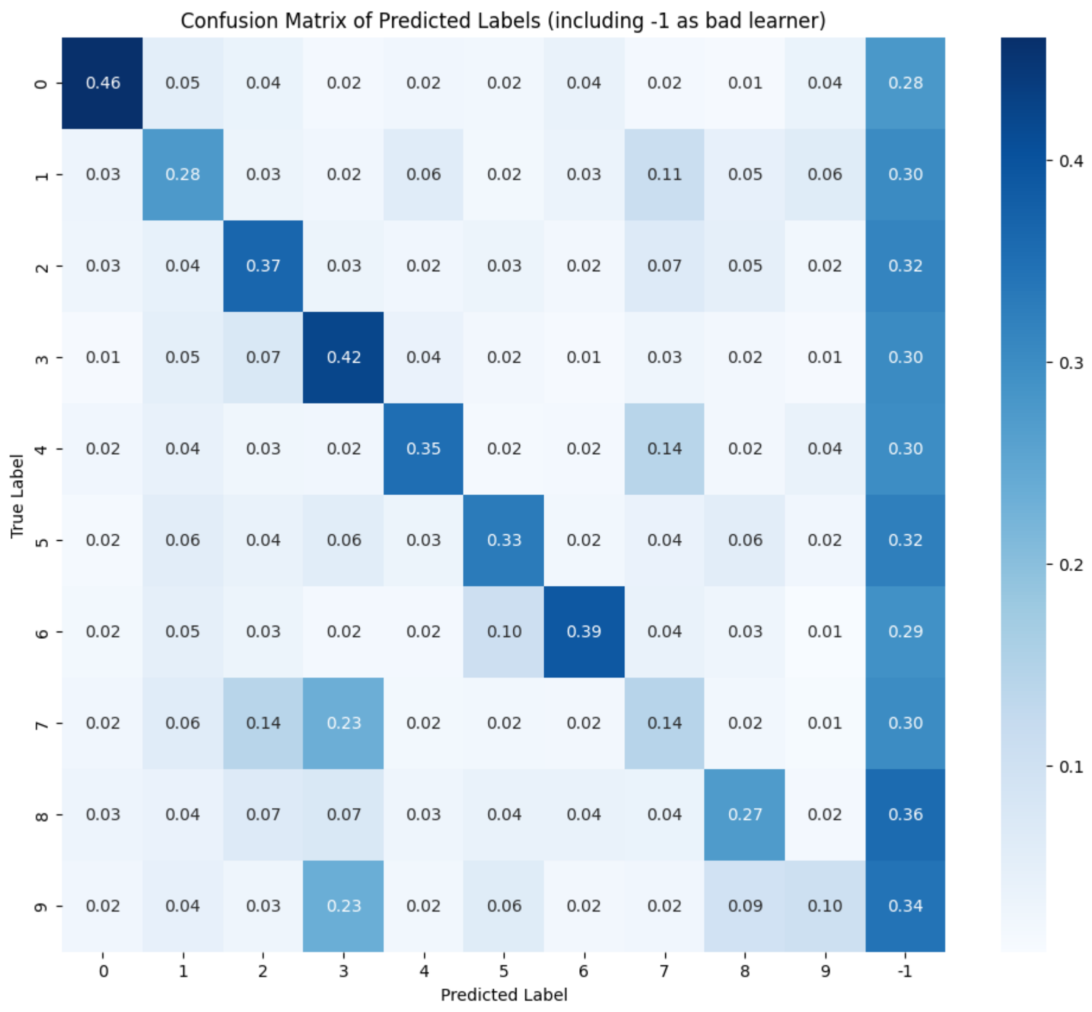


Figure 4.8: Normalized confusion matrix for the test dataset. Each cell represents the proportion of instances where the true label (shown on the y-axis) was predicted as the corresponding label (shown on the x-axis). The label -1 is included as an indicator for bad learner occasions. The diagonal elements represent the correctly classified instances, while the off-diagonal elements show the misclassifications. This matrix helps in understanding the relative frequency of misclassifications, highlighting which labels are often confused with each other.

Support threshold	Pairs	Triplets	Quadruplets	Quintuplets	Sextuplets
60%	692	1055	477	77	3
65%	147	72	3	0	0
70%	27	2	0	0	0
75%	1	0	0	0	0
80%	0	0	0	0	0

Table 4.2: Number of frequent itemsets of different sizes identified by the A-Priori algorithm at various support thresholds. Lower support thresholds result in more frequent larger itemsets, while higher thresholds reduce the number of frequent itemsets. The table shows the count of pairs, triplets, quadruplets, quintuplets, and sextuplets for each threshold.

tentially link parameter variations to specific predictive behaviors. A low distance between two models in a frequent itemset indicates that they underwent similar variations, while a high distance should indicate different variations.

There are various mathematical methods to calculate the distance between vectors, and the choice of distance metric depends on the vector's composition. Since we are using one-hot encoded values, the vectors will be sparse, containing many zeros. The best metric for sparse vectors is cosine similarity, which measures the angle between two vectors, focusing on their orientation rather than their magnitude. This makes it effective when vectors consist of many zeros. To convert cosine similarity to cosine distance, we subtract the cosine similarity value from 1. Another distance function that handles sparsity well is the Manhattan distance, or L1 Norm, which sums the absolute differences between two vectors.

Before the distance can be calculated, an adjustment must be made to the compressed deltas: the application of a scaling factor to the following parameters: outliers, typicals, invert, horizontal translation, vertical translation, rotate, and contrast. These parameters are float values between 0 and 1, representing the percentage of the dataset influenced by that variation. For example, if one model's dataset consists of 10% inverted digits and another has 70%, the difference between them is only 0.6, which does not significantly affect the distance calculation. Because these parameters are crucial in defining the data composition, their values are multiplied by 10. This adjustment increases the difference between 10% and 70% inverted images to 6, reflecting the significance of this variation more accurately.

Since the A-Priori algorithm returns frequent itemsets larger than pairs, we need to calculate the differences between multiple vectors. This is done by performing a pair-wise distance calculation for all vectors in each frequent itemset. Essentially, all combinations of size two are generated from the models in that itemset. After examining all pairs, the average distance for that itemset is calculated. This average distance represents how different the deltas are among the vectors.

In table 4.3, the median distances (Manhattan and Cosine) of frequent itemsets of different sizes with their corresponding itemset support are shown. The maximum distance between two compressed deltas is 97.6 for Manhattan and 1 for Cosine, indicating that the models in the itemsets are quite comparable. Additionally, the table reveals a trend: the higher the itemset support, the lower the distances between the compressed deltas of the models in those itemsets. This suggests that models with similar predictive behavior are also similar in the variations they have undergone, which implies the possibility of generating Rashomon sets by creating models with similar variations. However, it is important to note that the results for the 75% support threshold could be misleading, as they are based on only one sample.

Itemset Support	Itemset length	Manhattan Distance	Cosine Distance
60%	2	24.700	0.279
	3	24.133	0.295
	4	23.633	0.285
	5	23.540	0.277
	6	22.940	0.264
65%	2	23.400	0.279
	3	23.733	0.266
	4	23.700	0.260
70%	2	21.100	0.201
	3	21.067	0.205
75%	2	18.500	0.103

Table 4.3: The Manhattan and Cosine distances between the compressed deltas of models within the same frequent itemset are presented. Frequent itemsets with higher support thresholds (greater than 70%) exhibit lower distances compared to those with lower support thresholds.

Chapter 5

Explaining model multiplicity

When artificial intelligence is applied in user-centered applications, typically only one model is used. This can lead to overtrust if the user blindly trusts the model, or undertrust if the user does not trust the model and chooses not to use the application. By introducing a method to find multiplicitous models, we have identified models with similar predictive accuracy, making them equivalent advisors for a given use case. Instead of using these models individually, we can use models from a Rashomon set as a council of expert advisors. This approach means the user does not have to rely on a single model that may make mistakes, but can instead consider the advice of multiple equivalent models before making a decision. This can be especially relevant in critical decision support systems, such as those providing medical or legal advice, including systems like COMPAS.

5.1 Explaining by differences

In this section, we introduce a basic method for visualizing the different predictions of models in a Rashomon set. We do this by examining the predictions of similarly behaving models and comparing the differences in their hyperparameters and data composition. This information is logged in the models' compressed delta, which has been previously used to compare how similar Rashomon sets are across different support thresholds (section 4.3.3). The difference with this approach is that we now focus solely on the models from a specific Rashomon set, allowing them to predict a specific sample. They will provide their predicted labels, which may differ since they are not exact copies. We will then explain the differences in the predicted labels by analyzing their internal differences.

We tested a Rashomon set of three models that came from a frequent itemset with a support threshold of 70%. The sample tested is an inverted version of a sample labeled "5" from the test set. The three models show very different confidence levels in their predicted labels, as shown in figure 5.1. The first model is uncertain whether the correct label is five or eight, but slightly favors five. The second model has the same uncertainty but leans towards eight. The third model is very confident that the sample is labeled five and has no doubts.

To explain the large differences in confidence, we examine the differences in data composition of the three models. First, we look at the data augmentation distributions for the labels five and eight. It is possible that one of the labels underwent more augmentations compared to the other. Figure 5.2 shows a grouped bar chart for the label five (A) and eight (B), depicting the amount of data for a label that underwent a certain data augmentation. A side note is that the augmentation type "amount" shows the percentage of the total dataset that the label represents. As the bar charts indicate, both labels underwent the same amount of augmentations. This implies that the datasets for both labels are balanced, with no label being overrepresented, which could have led to a preference for a certain label. Since the sample these models are

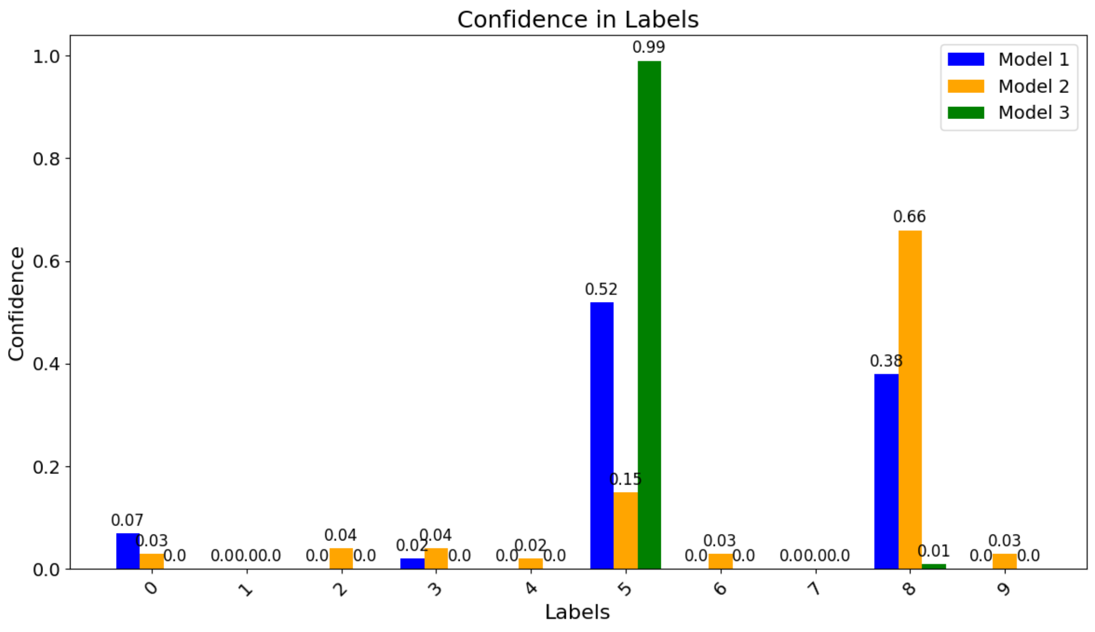


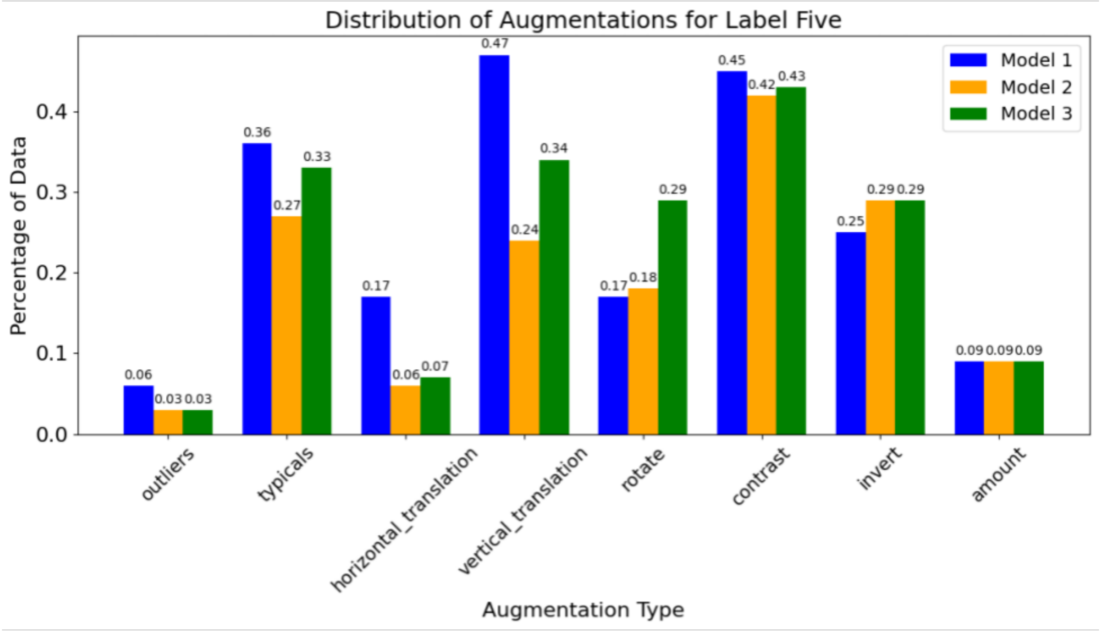
Figure 5.1: The different confidence levels for each label predicted by three multiplicitous models on a sample with the true label of five. The first and second models are uncertain between the labels five and eight, with the first model leaning towards five and the second towards eight. The third model, however, is the only one confident that the correct label is five.

evaluated on is an inverted digit, a lack of inverted images during training could have led to poor predictions. However, all three models have seen almost the same amount of inverted images.

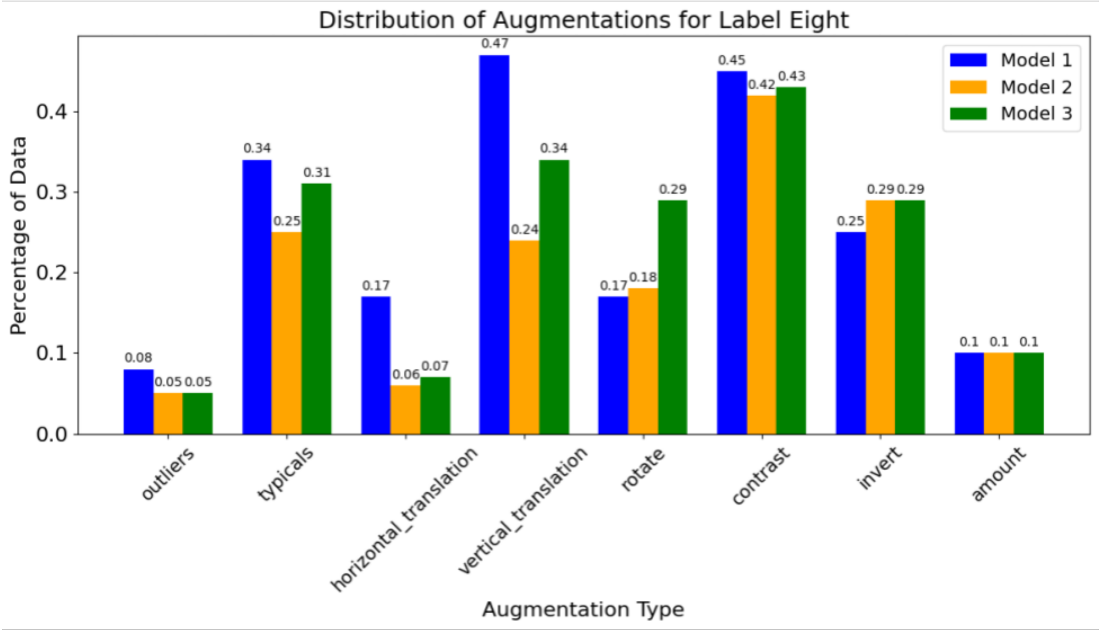
However, the percentage of data that underwent augmentation does not indicate the strength of each augmentation. For example, a contrast augmentation can either decrease or increase the contrast, which is crucial for augmentations such as contrast, rotations, and translations. The differences in augmentation strengths are visualized in figure 5.3. The first subfigure shows that the models that did not predict the correct label underwent positive horizontal translations, meaning the digits they learned from were slightly moved to the right, while the model that is confident in the correct prediction had its digits moved slightly to the left. In subfigure B, the differences in rotations are shown: the unsure models underwent positive rotations (rotated to the left), while the confident model's data was rotated to the right. Subfigure C shows that both unsure models experienced increased contrast, while the confident model had a very strong decreased contrast. These differences in augmentation strengths could be linked to the variations in predicted labels and their corresponding confidence levels.

In addition to differences in dataset composition, differences in model hyperparameters can also explain variations in predictive behavior for a certain sample. However, interpreting the impact of specific hyperparameters requires deep knowledge of neural networks and their learning processes. Therefore, these parameters are more suitable for expert users. The differences are shown in Table 5.1. While an immediate conclusion cannot be drawn, the difference in batch size is notable and could lead to different predictive behavior.

By considering the differences among the various advisors, a bias can be assigned to a certain model. For example, when predicting a sample with decreased contrast, an end user can examine the differences in contrast within the models' datasets and place more trust in the third model, which has been trained on similar data. However, this method is not final and will require further refinements to improve the ability to visualize differences in a more user-centered

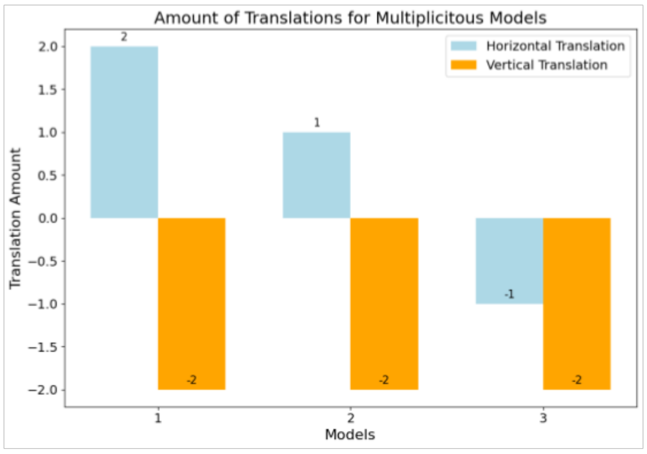


(A)

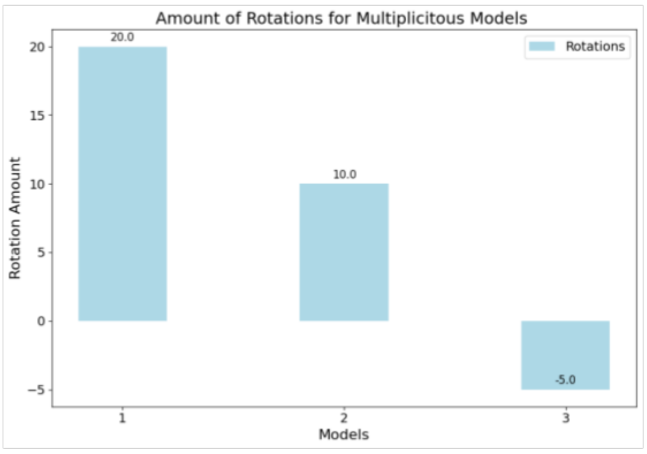


(B)

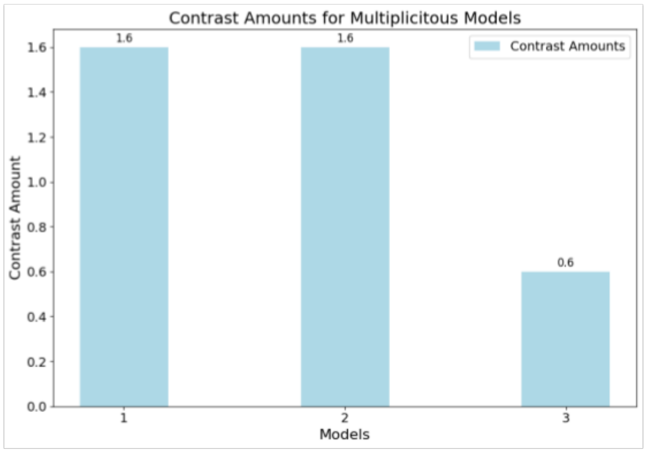
Figure 5.2: Two grouped bar charts depicting the percentage of data for a label that underwent dataset augmentations. The "amount" label visualizes the percentage of the total dataset with the corresponding label. Subfigure A represents the distribution of augmentations for label "five" and subfigure B for label "eight." The differences between the two labels are negligible. However, the differences between the three models are noticeable for the horizontal translation augmentation. For the other augmentations, the distribution is quite similar, indicating that the amount of augmented data is nearly the same.



(A)



(B)



(C)

Figure 5.3: Three bar charts depict the strengths of horizontal and vertical translations, rotations, and contrast adjustments, revealing differences between the models. The first and second models both have positive horizontal translations, while the best-performing model has a negative horizontal translation. For rotations, the best model has negative rotations, whereas the other two have positive rotations. The first and second models increase contrast, while the third model decreases it. This suggests the third model performs better with decreased contrast, right tilts, or leftward movements.

Model	Hidden Layers	Dropout	Act.Fun.	Optimizer	Batch size	Validation set
1	1	No	ReLu	RMSProp	512	Yes
2	2	Yes	GeLu	AdaMax	32	Yes
3	3	No	GeLu	FTRL	256	Yes

Table 5.1: The differences in hyperparameters for models in a Rashomon set require domain knowledge in machine learning and cannot be easily interpreted by end-users, making them more suitable for expert users.

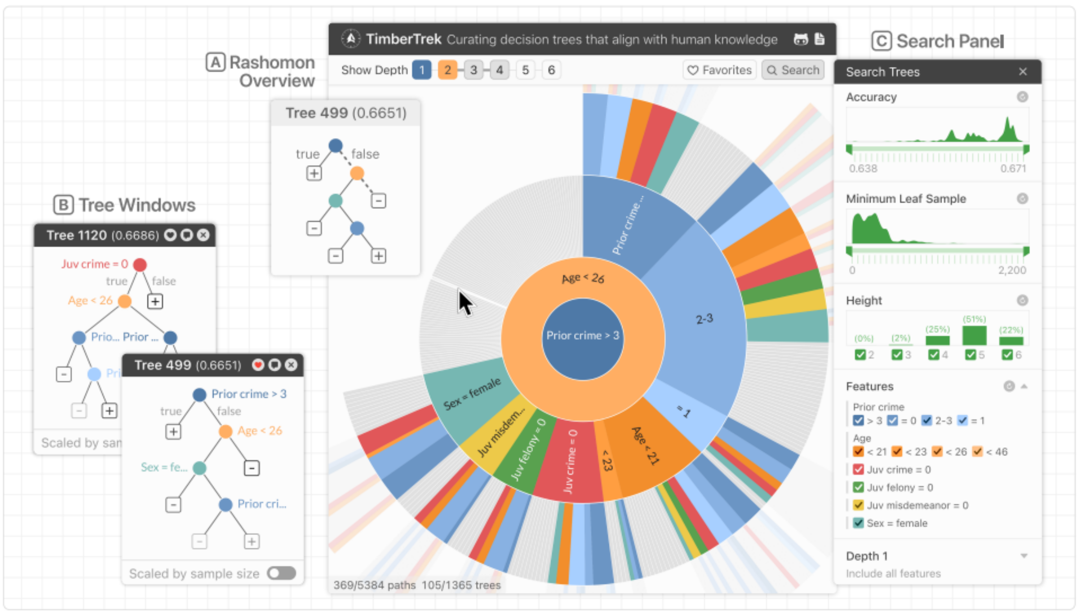


Figure 5.4: The TimberTrek sunburst chart incorporates additional features such as filtering decision trees based on various parameters, including accuracy and height. In the Tree Windows, the trees created by the user are displayed. However, as the decision space becomes more diverse, the window can become cluttered quickly, making it increasingly difficult for users to explore all possible paths [Wan+22].

approach.

5.2 Related work

As model multiplicity is a relatively new research area, there has been limited research on explaining the different models within a Rashomon set. However, one visualization tool for Rashomon sets has been identified: TimberTrek. TimberTrek employs a sunburst chart, enabling users to explore the different decision paths taken by various models within a Rashomon set. This allows users to examine all decision trees and select the one that best aligns with their decision-making process [Wan+22]. While TimberTrek is an effective tool for navigating the decision spaces of multiplicitous models, it can become cluttered as the decision space expands, as illustrated in figure 5.4.

Since TimberTrek only works with decision trees, it cannot be used with systems that do not use this specific model architecture. As we are using neural networks, which have a non-linear decision boundary, visualizing the different decisions is not possible. However, the idea of allowing users to create their own decision paths is interesting. We could implement this by letting users explore the different dataset augmentations of multiplicitous models and curate a model with their selected augmentations. This approach would enable users to learn how different augmentations impact the predictions made by the created model.

Chapter 6

Conclusions

6.1 Conclusions and future work

This thesis aimed to investigate the phenomenon of the "multiplicity of good models" and develop a method for identifying these models. Additionally, we sought to explain the different predictive behaviors of these models by linking them to variations in data composition and network architecture, focusing on neural networks that are multi-class predictors trained on the MNIST dataset. Our study demonstrated that a large number of models, varying in their training data and model hyperparameters, can be automatically generated using equivalence partitioning and controlled randomness. This method allows for the efficient generation of diverse models, facilitating the identification of multiplicitous models by evaluating them on a test set.

Our findings indicate that models exhibiting similar predictive behaviors tend to have undergone similar variations, thus confirming our research question. The method to generate multiplicitous models introduced in this thesis can be used in future research to further investigate how model multiplicity can change the model evaluation process and the future of user-centered applications that utilize neural networks. The presence of multiplicitous models in these use cases can reduce overtrust and undertrust by providing a council of expert advisors instead of relying on a single model for critical decisions.

While the study provides valuable insights, it is important to note the limitations. One major limitation is the use of the A-Priori algorithm to find frequent itemsets of equivalent predictors. This algorithm uses a lot of memory because it has to generate all possible combinations of frequent items of different sizes, which can result in disk thrashing when memory capacity is reached, significantly slowing down the algorithm. In this thesis, we evaluated 500 models on a small test set of 100 samples. To increase the variety of the test set, we introduced variations such as inverting, rotations, and translations, resulting in a larger test set of 1000 samples. This increase led to a higher number of baskets, quickly reaching memory capacity and making it unable to run on my hardware. The same problem occurred when we tried to increase the number of models for the evaluation process.

Another limitation is the lack of a user study. One of the goals of this thesis was to reduce overtrust and undertrust in decision support systems by implementing multiple expert advisors instead of relying on a single model. The impact of this council could have been better evaluated if the council and the internal differences between the models were shown to a set of test users. Without this study, we miss out on real-world feedback that could validate the increase in appropriate levels of trust.

These limitations have identified several key areas for future work on this topic. First, research alternative methods to better visualize the differences between multiple models of the same Rashomon set. These methods should then be implemented on a user-centered platform to conduct a user study and effectively gather insights on how this affects trust in artificial intelligence. Next, develop a scalable algorithm that finds frequent itemsets more efficiently, allowing for the expansion of the dataset and the number of models. Finally, apply the method introduced in this study to datasets other than the MNIST dataset, incorporating more complex data such as time-series data and implementing more complex neural networks, such as convolutional neural networks or recurrent neural networks.

6.2 Personal reflection

Embarking on this research journey has been both challenging and rewarding, particularly as this thesis originated from a hand-drawn sketch and has evolved into a fully functioning proof of concept that utilizes new phenomena. The process of effectively executing an idea originating from a brainstorming session fills me with pride.

Throughout this process, I encountered numerous technical challenges, particularly with the computational limitations of the A-Priori algorithm and the parallel generation of neural networks. These hurdles pushed me to think creatively and adapt my approaches to achieve meaningful results despite hardware constraints. This experience has significantly enhanced my problem-solving skills and deepened my understanding of the complexities involved in machine learning research.

One of the most enlightening aspects of this project was realizing the potential impact of my work on real-world applications. The idea that a council of models could provide more balanced and trustworthy decision support systems is something I find particularly inspiring. This has strengthened my belief in the importance of transparency and diversity in AI systems, and I am excited about the possibilities for future research in this area.

In conclusion, this thesis has been a significant learning experience, shaping my perspective on both the technical and human aspects of AI. I look forward to continuing this journey, building on the foundation laid by this research, and contributing to the development of more robust and trustworthy AI systems.

Bibliography

[Lec+98] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[Bre01] Leo Breiman. “Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)”. In: *Statistical Science* 16.3 (Aug. 2001). ISSN: 0883-4237. DOI: 10.1214/ss/1009213726. URL: http://dx.doi.org/10.1214/ss/1009213726.

[GQ01] Ramazan Gençay and Min Qi. “Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging”. In: *Neural Networks, IEEE Transactions on* 12 (Aug. 2001), pp. 726–734. DOI: 10.1109/72.935086.

[Haw03] Douglas M. Hawkins. “The Problem of Overfitting”. In: *Journal of Chemical Information and Computer Sciences* 44.1 (Dec. 2003), pp. 1–12. ISSN: 0095-2338. DOI: 10.1021/ci0342472. URL: http://dx.doi.org/10.1021/ci0342472.

[Wei03] Eric W Weisstein. “Rotation matrix”. In: *https://mathworld. wolfram. com/* (2003).

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey”. In: *ACM Comput. Surv.* 41 (July 2009). DOI: 10.1145/1541880.1541882.

[LTZ09] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. “Isolation Forest”. In: Jan. 2009, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[Ben12] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures”. In: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533. URL: http://arxiv.org/abs/1206.5533.

[Bor12] Christian Borgelt. “Frequent item set mining”. In: *WIREs Data Mining and Knowledge Discovery* 2.6 (Oct. 2012), pp. 437–456. ISSN: 1942-4795. DOI: 10.1002/widm.1074. URL: http://dx.doi.org/10.1002/widm.1074.

[LeC+12] Yann A. LeCun et al. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade: Second Edition.* Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_3. URL: https://doi.org/10.1007/978-3-642-35289-8_3.

[BS13] Pierre Baldi and Peter J Sadowski. “Understanding Dropout”. In: *Advances in Neural Information Processing Systems.* Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf.

[Sri+14] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[Kes+16] Nitish Shirish Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

[LYP16] Seung-Hwan Lim, Steven Young, and Robert Patton. “An analysis of image storage systems for scalable training of deep neural networks”. In: Apr. 2016.

[Coh+17] Gregory Cohen et al. “EMNIST: Extending MNIST to handwritten letters”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2921–2926. DOI: 10.1109/IJCNN.2017.7966217.

[DCF17] Anupam Datta, Cmu, and Matt Fredrikson. “Proxy Discrimination in Data-Driven Systems Theory and Experiments with Machine Learnt Programs”. In: 2017. URL: https://api.semanticscholar.org/CorpusID:204906842.

[AB18] Amina Adadi and Mohammed Berrada. “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)”. In: *IEEE Access* 6 (2018), pp. 52138–52160. ISSN: 2169-3536. DOI: 10.1109/access.2018.2870052. URL: http://dx.doi.org/10.1109/ACCESS.2018.2870052.

[Che+18] Chaofan Chen et al. *An Interpretable Model with Globally Consistent Explanations for Credit Risk*. 2018. arXiv: 1811.12615 [cs.LG].

[DF18a] Julia Dressel and Hany Farid. “The accuracy, fairness, and limits of predicting recidivism”. In: *Science Advances* 4.1 (Jan. 2018). ISSN: 2375-2548. DOI: 10.1126/sciadv.aao5580. URL: http://dx.doi.org/10.1126/sciadv.aao5580.

[DF18b] Julia Dressel and Hany Farid. “The accuracy, fairness, and limits of predicting recidivism”. In: *Science Advances* 4.1 (Jan. 2018). ISSN: 2375-2548. DOI: 10.1126/sciadv.aao5580. URL: http://dx.doi.org/10.1126/sciadv.aao5580.

[HG18] Patrick Hall and Navdeep Gill. *An introduction to machine learning interpretability: An applied perspective on fairness, accountability, transparency, and explainable AI*. en. 2018.

[Sid+18] Md. Abu Bakr Siddique et al. “Study and Observation of the Variations of Accuracies for Handwritten Digits Recognition with Various Hidden Layers and Epochs using Neural Network Algorithm”. In: *2018 4th International Conference on Electrical Engineering and Information and Communication Technology (iCEEiCT)*. 2018, pp. 118–123. DOI: 10.1109/CEEICT.2018.8628144.

[YBK18] Kun-Hsing Yu, Andrew L. Beam, and Isaac S. Kohane. “Artificial intelligence in healthcare”. In: *Nature Biomedical Engineering* 2.10 (Oct. 2018), pp. 719–731. ISSN: 2157-846X. DOI: 10.1038/s41551-018-0305-z. URL: http://dx.doi.org/10.1038/s41551-018-0305-z.

[Ber+19] Dimitris Bertsimas et al. “The Price of Interpretability”. In: (2019). DOI: 10.48550/ARXIV.1907.03419. URL: https://arxiv.org/abs/1907.03419.

[Che+19] Hao-Fei Cheng et al. “Explaining Decision-Making Algorithms through UI: Strategies to Help Non-Expert Stakeholders”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI ’19. ACM, May 2019. DOI: 10.1145/3290605.3300789. URL: http://dx.doi.org/10.1145/3290605.3300789.

[Kam19] Margot E. Kaminski. “”The Right to Explanation, Explained””. In: (2019). DOI: 10.15779/Z38TD9N83H. URL: https://lawcat.berkeley.edu/record/1128984.

[Lar+19] Stefan Larson et al. “Outlier Detection for Improved Data Quality and Diversity in Dialog Systems”. In: (2019). DOI: 10.48550/ARXIV.1904.03122. URL: https://arxiv.org/abs/1904.03122.

[And+20] Christopher Anders et al. “Fairwashing explanations with off-manifold detergent”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 314–323. URL: https://proceedings.mlr.press/v119/anders20a.html.

[DAm+20] Alexander D’Amour et al. “Underspecification Presents Challenges for Credibility in Modern Machine Learning”. In: *CoRR* abs/2011.03395 (2020). arXiv: 2011.03395. URL: https://arxiv.org/abs/2011.03395.

[Des20] Chitra Desai. “Comparative Analysis of Optimizers in Deep Neural Networks”. In: (Oct. 2020).

[LRU20] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. en. 3rd ed. Cambridge, England: Cambridge University Press, Jan. 2020.

[MCU20] Charles Marx, Flavio Calmon, and Berk Ustun. “Predictive Multiplicity in Classification”. In: *Proceedings of the 37th International Conference on Machine Learning*.

Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 6765–6774. URL: `https://proceedings.mlr.press/v119/marx20a.html`.

[Meh+20] Johannes Mehrer et al. "Individual differences among deep neural network models". In: *Nature Communications* 11 (Nov. 2020). DOI: `10.1038/s41467-020-19632-w`.

[RAS20] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. "A Review of Activation Function for Artificial Neural Network". In: *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. 2020, pp. 281–286. DOI: `10.1109/SAMI48414.2020.9108717`.

[SSA20] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. "ACTIVATION FUNCTIONS IN NEURAL NETWORKS". In: *International Journal of Engineering Applied Sciences and Technology* 04 (May 2020), pp. 310–316. DOI: `10.33564/IJEAST.2020.v04i12.054`.

[UJ20] Muhammad Uzair and Noreen Jamil. "Effects of Hidden Layers on the Efficiency of Neural networks". In: *2020 IEEE 23rd International Multitopic Conference (INMIC)* (2020), pp. 1–6. URL: `https://api.semanticscholar.org/CorpusID:231682198`.

[AZA21] Talal A. A. Abdullah, Mohd Soperi Mohd Zahid, and Waleed Ali. "A Review of Interpretable ML in Healthcare: Taxonomy, Applications, Challenges, and Future Directions". In: *Symmetry* 13.12 (Dec. 2021), p. 2439. ISSN: 2073-8994. DOI: `10.3390/sym13122439`. URL: `http://dx.doi.org/10.3390/sym13122439`.

[BN21] Sarat Moka Benoit Liquet and Yoni Nazarathy. *Mathematical Engineering of Deep Learning*. CHAPMAN and HALL CRC, 2021.

[DAm21] Alexander D'Amour. "Revisiting Rashomon: A Comment on "The Two Cultures"". In: *Observational Studies* 7.1 (2021), pp. 59–63. ISSN: 2767-3324. DOI: `10.1353/obs.2021.0022`. URL: `http://dx.doi.org/10.1353/obs.2021.0022`.

[RB21] Atiq ur Rehman and Samir Brahim Belhaouari. "Unsupervised outlier detection in multidimensional data". In: *Journal of Big Data* 8.1 (June 2021). ISSN: 2196-1115. DOI: `10.1186/s40537-021-00469-z`. URL: `http://dx.doi.org/10.1186/s40537-021-00469-z`.

[BRB22] Emily Black, Manish Raghavan, and Solon Barocas. "Model Multiplicity: Opportunities, Concerns, and Solutions". In: *2022 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '22. ACM, June 2022. DOI: `10.1145/3531146.3533149`. URL: `http://dx.doi.org/10.1145/3531146.3533149`.

[Oli+22] Leonardo Ferreira de Oliveira et al. "Path and future of artificial intelligence in the field of justice: a systematic literature review and a research agenda". In: *SN Social Sciences* 2.9 (Aug. 2022). ISSN: 2662-9283. DOI: `10.1007/s43545-022-00482-w`. URL: `http://dx.doi.org/10.1007/s43545-022-00482-w`.

[Van22] Frank Van Reeth. *Computer graphics Cursustekst*. Universiteit Hasselt, Faculteit Wetenschappen, 2022.

[Wan+22] Zijie J. Wang et al. "TimberTrek: Exploring and Curating Sparse Decision Trees with Interactive Visualization". In: *2022 IEEE Visualization and Visual Analytics (VIS)*. 2022, pp. 60–64. DOI: `10.1109/VIS54862.2022.00021`.

[Far+23] Michael Mayowa Farayola et al. "Fairness of AI in Predicting the Risk of Recidivism: Review and Phase Mapping of AI Fairness Techniques". In: *Proceedings of the 18th International Conference on Availability, Reliability and Security*. ARES 2023. ACM, Aug. 2023. DOI: `10.1145/3600160.3605033`. URL: `http://dx.doi.org/10.1145/3600160.3605033`.

[FW23] Raymond Fok and Daniel S. Weld. *In Search of Verifiability: Explanations Rarely Enable Complementary Performance in AI-Advised Decision Making*. 2023. DOI: `10.48550/ARXIV.2305.07722`. URL: `https://arxiv.org/abs/2305.07722`.

[Lar23] Retno Larasati. "Trust and Explanation in Artificial Intelligence Systems: A Healthcare Application in Disease Detection and Preliminary Diagnosis". In: (2023). DOI: `10.21954/OU.RO.00015ACA`. URL: `https://oro.open.ac.uk/id/eprint/88778`.

[RHN23] Arjun Roy, Jan Horstmann, and Eirini Ntoutsi. *Multi-dimensional discrimination in Law and Machine Learning – A comparative overview.* 2023. DOI: `10.48550/ARXIV.2302.05995`. URL: `https://arxiv.org/abs/2302.05995`.

[Tha+24] Panissara Thanapol et al. "Round-Based Mechanism and Job Packing with Model-Similarity-Based Policy for Scheduling DL Training in GPU Cluster". In: *Applied Sciences* 14 (Mar. 2024), p. 2349. DOI: `10.3390/app14062349`.

[UL24] Shahadat Uddin and Haohui Lu. "Dataset meta-level and statistical features affect machine learning performance". In: *Scientific Reports* 14.1 (Jan. 2024). ISSN: 2045-2322. DOI: `10.1038/s41598-024-51825-x`. URL: `http://dx.doi.org/10.1038/s41598-024-51825-x`.