Rheinisch-Westfälische Technische Hochschule Aachen
Knowledge-Based Systems Group
Prof. G. Lakemeyer, Ph. D.

Understanding Hot Rolling Processes Using Machine Learning

Bachelor's Thesis

Johannes Gocke
Matriculation Number: 311979

Supervisor:	Prof. Gerhard Lakemeyer, Ph. D.
Second Supervisor:	Prof. Dr.-Ing. Tobias Meisen
Advisors:	Dipl. Inform. Martin Liebenberg
	Richard Meyes, M. Sc.

Contents

1 Introduction

In today's global economic system the markets around the world are mainly focused on a single goal: **growth**. To provide this growth, ever improving means of production are required. This connection lead to the first industrial revolution at the end of the 18th century, when coal became the most prominent driving force, especially in conjunction with steel production. In the early 20th century the second industrial revolution took place, when the importance of oil as a fossil combustible material surpassed coal and oil became the leading source of energy. Electricity found its way into factories and mass production inspired by Ford drastically increased output capacities. The digital revolution, the third industrial revolution, started in the 1970s, when computers started becoming one of the primary assets for automation, and information became a vital part of commerce. All this leads to Industry 4.0, or the proposed fourth industrial revolution. Industry 4.0 describes the trending process of collecting data and forwarding it to automate manufacturing, like in assembly lines using robots. Like its predecessors it creates the desire for more precise production at a lower cost (Wolter et al., 2015). The metal industry is one of the predominant fields supporting economic growth. Steel is widely used, for example, in construction, infrastructure development and assembly. China, the fastest growing economy, undertook enormous efforts to increase its steel production. The country increased its combined steel output from 37 million tons in 1980 to 100 million tons in 1996 (Tang et al., 2000).

This thesis targets improving the highly sought after metal working process of hot rolling. In hot rolling the height of a block of steel, called slab, is decreased. The slab is moved through an array of rolls, called rolling mills, and each iteration is referred to as a pass. To improve the process of hot rolling it is required to firmly understand the underlying mechanisms during the transformation of hot slabs into steel sheets, and to gain better knowledge and control of the parameters involved (Tsoi, 1991). Motivated by the possibility to make use of the vast amounts of data that is collected during the hot rolling process, the focus of this thesis lies on analyzing the benefits of using machine learning concepts on simulated hot rolling processes. One crucial quality of a steel block, and this thesis's primary optimization target, is its particle size, henceforth referred to as grain size. A steel block consists of tiny molecular grains, which determine qualities like hardness or stability. Since grain size has additional effects it is important to guarantee a specific grain size of the manufactured steel sheet, neither too high nor too low, so that it meets the industrial requirements, e.g. of the building industry.

Outlining the structure of this thesis, gives an overview over the basic concepts used and actions taken. First, the thesis problem, goals and planned work flow are detailed. Motivation and a look at similar research adds to this. Then hot rolling processes are illustrated, showing the metallurgic field dealt with. Following this, the Chapter Theoretical Background starts laying down the base concepts and techniques used in this thesis. Machine Learning gives an overview of the respective field and explains the connection to hot rolling processes through provided data. The two chapters after (Association Rule Mining and Dimensionality Reduction) illustrate the two steps that form the backbone of this thesis by providing explanations of basic ideas. Prediction is the first chapter to directly deal with the problem at hand. Basic concepts like classification and regression are explained. The Chapter Results now begins by showing the practical applications of the previous sections. This is followed by the second big section in Results; Improving Prediction. Here the techniques worked out in previous segments are brought together in order to find better and more satisfactory ways to solve this thesis problem. The Chapter Designing Pass Plans, forms a digression to show the actual applicability of the prediction models created in this thesis. This way it is shown how obtained results can be used to solve further problems. The final chapter then concludes with an overview over and evaluation of the results this thesis has produced. It also details what further research would be interesting or purposeful.

2 Problem Setting

As mentioned in Chapter 1, there is a strong desire to improve steel production and the hot rolling process. To calculate the result of a pass, various complex non-linear functions are required, e.g. for micro-structural changes or softening mechanisms (Rao et al., 1998). But knowledge of these is mainly expert-based, brought about by long time work, and less consolidated (Tang et al., 2000). Thus the whole process might not run optimally and waste energy, material, time and money. The rules between the calibration of rolling mills and the quality of the produced steel are interesting and complex, as they are based on a combination of highly complex and non-linear physical processes, which act together to shape the received block. Discovering these processes or rules may help deepen the understanding of the hot rolling process and metal-working in general. Or simply put, since an ever-growing amount of data becomes available, it is reasonable to assume that smart data analysis will manifest into a necessary leading role for technological progress in fields like this (Smola and Vishwanathan, 2010).

2.1 Motivation

Machine learning is a fast-growing field with a lot of ongoing research, from natural speech recognition to self-driving cars, which is underlined by success stories and improvements every day and illustrates the need for fast progress. This makes it an interesting and relevant research topic with future potential. Mainly founded in the 1950-1970s there still is a lot to discover and create in this field of research. There is a huge amount of machine learning techniques which all have different strengths and weaknesses and it is interesting to see how these differences apply to problems. Take for example, how the speed of linear tools gets outclassed by the easy applicability of neural networks on non-linear problems (Liu, 2001), or how Association Rule Mining shows relations unheard of. Furthermore it is amazing to see how these can be integrated into such a huge amount of common or household appliances, like smart homes (Teich et al., 2014), recommendation of Youtube videos (Davidson et al., 2010), which uses Association Rule Mining, or, more recently, AlphaGo beating professional Go players (Silver et al., 2016).

2.2 Thesis Goal and Methodology

The Establishing of rules, which the physical background of hot rolling is based on and the creation of a model of the underlying process that is able to predict future results is the subject of this thesis. The focus lies on the grain size of the resulting steel sheet. Furthermore there will be attempts to make sure the created model is at least close to a white-box model, which allows extracting gained knowledge, normally through additional steps. The difference between a black box and white box model is that while both may deliver a valid prediction it is possible to, at least partially, understand how the prediction of a white box model was formed, whereas a black box model does not allow the extraction of explicit knowledge. In a first step the rules governing the hot rolling process will be established using Association Rule Mining. A machine learning technique that aims to find relationships between items through statistical methods. Afterwards, a combination of dimensionality reduction methods will be used to add weights to the extracted rules. These determine the significance of each rule and its items for the hot rolling process. Dimensionality reduction refers to mathematical models that aim to map multi-dimensional data to a lower number of dimensions, usually two or three to enable visualization. Finally, a machine learning model of the hot rolling process will be implemented. Moreover, these three steps will be applied together to improve the prediction model and then evaluated concerning their contribution towards the former. All these steps are aimed at creating the following benefits: To start with, applications of the hot rolling process could become more efficient or effective, thanks to one's being able to better predict the result of a pass. This

may manifest either in less required time, energy consumption, result quality, stability or others. Furthermore, the detailed process of finding such a solution could be applied to comparable tasks in other fields of production as, for example, the design of rolling schedules.

In order to achieve the goal of predicting the grain size for the manufactured steel sheet, several machine learning techniques are used, this includes classification and regression analysis through decision making components. The task of implementing and improving these is data-driven, which means that conclusions on further steps should be derived from the provided data instead of external knowledge, this allows for analyzing a hot rolling process without any knowledge about the underlying physical relations.

Throughout the thesis, visualizations of the hot rolling process will stem from dimensionality reduction, which help demonstrate and understand relationships in the settings parameters of hot rolling and their governing rules and constraints. At the same time a documentation of the process used to achieve this will be delivered, allowing further research and understanding of the process followed in this thesis. At the end of this thesis the final chapter will discuss the relationship between settings for the hot rolling process and their results and will show how well a data driven prediction model is able to grasp these.

2.3 Related Work

Several articles and other scientific distributions deal with problems similar to this thesis. Beynon and Sellars (1992) collected the state of scientific knowledge behind microstructure changes in hot rolling. Their paper did not only lead to numerous studies by centralizing the knowledge of the changes in micro structure through hot rolling but the author Sellar is also known for a recrystallization model named after him. Data used in this thesis is based on his works, especially the formerly named recrystallization model.

Feldmann (1994) wrote about the possibility to use computers to analyze the required models. He came to the conclusion that computing has become a central component of developing models for rolling, but admits that there are many problems, especially considering the field of plastomechanic processes.

Using neural networks for predictions of hot rolling processes was researched by a number of people: P. Korczak and Łabuda (1998), attempted to predict mechanical properties of a slab after hot rolling, they came to the conclusion that neural networks deliver good results in predicting, but do not go into much detail about this. Furthermore, by choosing neural networks he limits himself to a black box model, which makes it impossible to obtain knowledge from their model. Their work might be the closest one to this thesis, with grain size being a focus point,

but branches off in depth and chosen techniques, primarily through the use of neural networks. Kusiak and Kuziak (2002) present a work very similar to the previous one, coming to the same result that neural networks deliver good results in predicting grain size. Lee and Lee (2002) focus on improving the accuracy of rolling models, using neural networks again. In the end they proposes the use of long-term learning neural networks to predict the qualities of produced slabs. Like in the two previous papers they come to the conclusion that traditional rolling models deliver very poor results, especially with focus on the first pass. In a second paper Lee and Choi (2004) evaluate the application of an on-line neural network on hot rolling processes.

Hong and Park (2003) analyzed processes for designing pass schedules to manipulate grain size. They find that while this is possible the result is still lacking in many areas, especially when it comes to height reductions above 20%. They also suggest that the history of temperature and strain have the biggest influence on grain size, meaning that initial temperature but also cooling time is very influential on grain size. This is a recurring subject in this thesis. Park (2001) tried predicting the grain size of steel, but with a more specialized focus on thick-plate rolling. He finds that controlled rolling, according to a prediction model, leads to better results, but must admit that the increase in force brought upon by this is detrimental to the process, making it unusable for many purposes.

3 Use Case: Hot Rolling

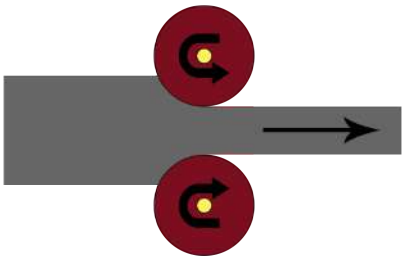


Figure 3.1: Diagram of a pass, in which hot steel is moved through 2 rolls to reduce its height, affecting many of the steels qualities like grain size.

Rolling is the process of decreasing the thickness of a metal block by passing it through an array of turning rolls, as seen in 3.1, normally one above and one below the processed block. The machine used for this is called a rolling mill, it consists of the rolls and their propulsion but also lifts that allow to increase or decrease the distance between its rolls.

The process of decreasing the thickness by a single pair of turning rolls is called pass. Rolling can be used to make a block uniform or increase other qualities like stability or hardness. Exemplary products would be steel sheets for construction or manufacturing but also bars, like rails or support beams. When rolling a steel slab, while the rolled material is above a temperature that allows recrystallization, it is considered hot rolling, below these temperatures as cold rolling (DeGarmo et al., 2003).

During rolling the molecular grain structure of the materials change, this is called Recrystallization. When a slab of steel is compressed, the grains that form the body get deformed and compressed. When cooling of after a pass these deformed grains now change their compromised shape to a more stable one via autonomous restructuring also called nucleation. In this process the orders between some grains weaken over time and they unify to bigger more stable grains. Hot rolling is often done at a slightly higher than required temperature, this ensures homogeneity in the resulting block or sheet (Roberts, 1983).

A mill is called reversing, when it is possible to pass steel through it in both directions, usually the same block going back and forth in quick succession. A mill not capable of this is called non-reversing. There is a huge amount of different rolling mill set-ups as can be seen in Figure 3.2. Each of the depicted roll arrangements exist and each comes with different benefits and trade-offs. A roll set-up as in E or F in the Figure allows for rolling of different structural shapes, they can be used for rolling curved sheets, these are also called shape mills (Grimble and Hearns, 1999).

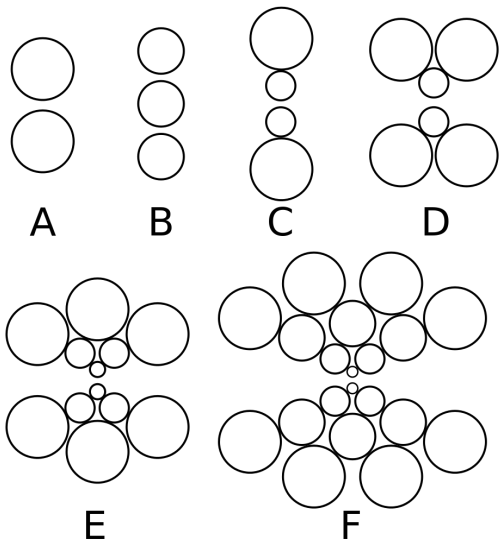


Figure 3.2: Diagram of various roll arrangements used in rolling mills.

The rolling mill that is to be emulated in this thesis is a so-called two-high reversing mill (number A in the Figure) which means only two rolls, but they are capable of reversing. This comes with the disadvantage that both rolls need to be halted, their rolling direction changed or reversed, and then accelerated to working speed again. Therefore what is called the non-reversing three-high mill was introduced (number B in the figure) which moves the slab alternating between the lower and upper gap, allowing for continuous rolling. This comes with the trade-off that lifts and their operation is required to move the block up and down between the gaps.

A hot rolling process has several different settings that can be adjusted and can drastically change the characteristics of the received slab or sheet. However, there are also some constraints. Several important ones are:

Settings:	Criteria:
• Temperature and cooling	• Height
• Initial grain size	• Resulting grain size
• Speed of a pass and time after	• Energy consumption
• Slab geometry	• Machine constraints
• Material composition	• Stability of results

4 Theoretical Background

This chapter's focus lies on establishing the theoretical background that allows for the obtaining of results in Chapter 5. First, the central mechanic of this thesis, machine learning, is introduced. Following this, association rule mining and dimensionality reduction are detailed, two techniques used to improve the results of prediction. Section 4.4 Prediction, then examines the methods used to model the hot rolling process and predict its results.

4.1 Machine Learning

Machine learning is a field focused on programs that can learn a parametrized function from examples in a training process. As described in Chapter 2.1, machine learning was extremely successful in recent years with applications in many fields like autonomous vehicles (Chen et al., 2015) or pattern recognition to detect credit card fraud (Brause et al., 1999). There have been advances of great significance in the algorithms and background theory that lay the foundation of machine learning (Mitchell, 1997). In practice, machine learning generally refers to having data and selecting or developing a model, which contains a function that is able to approximately predict a result based on input. The chosen model is then adapted so it fits the given data, which is called training or fitting. In the fitting process, the

parameters of a function that represents the model are adapted to data samples. For a linear function this would be done by changing the gradient or axis intercept so the function gets closer to data points. Afterwards, the adjusted model should be able to fulfill desired tasks like estimating the result of future unseen data, rating something, controlling a system or completing several other objectives.

4.1.1 Data

The most important aspect that determines whether machine learning is successful or not is the amount and quality of available data or the ability to create it. All concepts need samples, just like humans need experiences or a knowledge repository to learn new things. Several points affect the quality of data for machine learning (Fayyad et al., 1996).

- Amount of samples
- Amount and recognizability of invalid samples
- Measurement precision
- Representativeness and completeness
- Randomness and distribution of targets

The amount of samples is crucial for estimating how close a representing model can come to being the best possible model for a given procedure. But prediction quality does not scale linearly with the amount of samples. At a certain point the average precision gained per sample is not worth the extra time required for processing, as is visible in Figure 4.1. Here a huge increase in precision can be seen in the initial 40 samples, followed by a slower raise until around 100 samples, from then on growth in precision is extremely small in comparison to earlier results. This means that by adding more samples results do not improve much.

Invalid samples, are broken records, for example, by data loss or corruption during conversion, generation or transmission. This does not mean an unclean measured value but rather a completely wrong one. *Measurement precision*, on the other hand, is an unavoidable obstacle when samples are acquired. Assuming a sample describes the opacity of an image and the correct opacity is 0.5. Now the measuring tool is not very good and a value of 0.6 is determined; this shows a lack of measurement precision. A measured value of 10, because the tool had a malfunction, is an invalid sample. The main difference is that measurement precision errors can not be avoided, as all analog data always has a margin of error, but the measurement precision is estimable, therefore a certain amount of imprecision always needs to be factored in. On the other hand, a broken sample could be avoided, either by filtering samples or by ensuring the data is correct in the first place.

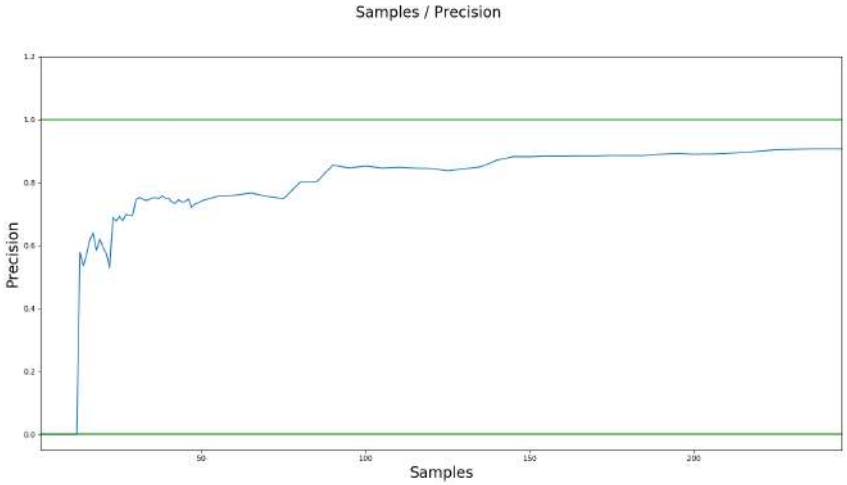


Figure 4.1: Precision versus number of samples for a random forest regressor. The blue line represents the model precision. The green lines represent the maximum and minimum of precision.

Representativeness and Completeness means whether samples represent reality. This depends on the conditions under which such samples are acquired. Maybe certain groups or aspects in samples are under- or overrepresented or even missing.

Randomness is less depending on the way samples are generated but on the way they are distributed into training and test sets. The target of learning is not that models are able to recite the training set but to generalize in order to predict from previously unseen data. A training set is the part of samples that is used to fit a machine learning procedure to data. A test set is used to validate the model. It is crucial to have a balanced split between training and test data. If, for example, a problem wants to differentiate between two values and all trainings sets belong to value 1 and all test sets to value 2, the results will probably be worse than random guessing. Thus it is possible that different separations of the same data create diverse models. Therefore it is recommended to train multiple instances with randomized separations, cross validation, the process of comparing models, can then be used evaluate their performance.

4.1.2 Hot Rolling Data

The relevant data for this thesis is provided via a database of records from a physically accurate simulation of the hot rolling process. To create a matrix the database is transformed into a 2 dimensional array with n columns. Columns $1...n-1$ each contain a different input parameter and column n the target value.

A function, given a list of parameter names, produces the 2D array in any combination of parameters. Thus different combinations can be tested quickly. As an example for hot rolling data the matrices 4.1 and 4.2 will be used repeatedly. The input columns are defined as: θ_{Init} for temperature, G_{Old} for the grain size before a pass, H_{Old} and H_{New} for old and new height and T_p for the cooling down time after the pass, afterwards grain size is measured. The target column is Δ_G for GrainSizeDelta, the change in grain size between two passes. From all available data association rules will be determined.

θ_{Init} [°C]	G_{Old} [µm]	H_{Old} [mm]	H_{New} [mm]	T_p [s]	Δ_G [µm]
1010	200	300	250	20	-30
1010	200	300	200	20	-30
1010	200	250	200	60	+30
1010	100	300	250	60	+30

Table 4.1: Raw Matrix (RM)

θ_{Init}	G_{Old}	H_{Old}	H_{New}	T_p	Δ_G
1	1	1	1	0	0
1	1	1	0	0	0
1	1	0	0	1	1
1	0	1	1	1	1

Table 4.2: Labeled Matrix (LM); Binary labels, 1/0, show the columns max/min

4.2 Association Rule Mining

Association Rule Mining (ARM) also known as Association Rule Learning is a machine learning concept that focuses on finding relationships between $1, ..., n$ items and target items, with n being the maximum amount of items available (Agrawal et al., 1993). Take, for example, the relationship between bought goods in a supermarket: someone buying bread might be more likely to also buy butter. A rule or relationship is defined as an increased probability of simultaneous occurrences or scale of several items compared to unrelated items. In simpler terms, a Rule $X_1 \Rightarrow X_2$ implies that the item(s) X_1 occurs more often at the same point as X_2 than is statistically likely for disjunct values that share no relation. This can either indicate a hidden connection between items or badly selected samples. This does not imply causality but a simultaneous occurrence that is more often than it statistically should be. Association rules do not tell anything about the weight or influence certain items have on target items or results, like small increases in temperature leading to huge decreases in grain size. Instead they represent what is called Confidence, or Lift, a metric that describes how much items influence or

relate to each other (Agrawal et al., 1993). Association Rule Mining uses different measurements or metrics to describe rules. The main use of Association Rule Mining in this thesis is to gather more information about the input parameters and possibly reduce the number of relevant dimensions.

4.2.1 Definitions

To understand how association rule mining works it is required to know the principles it relies on to extract rules from data, like a matrix of samples. Given a dense Matrix M, each row in M is a sample, which contains the values of all input and output parameters in a single run. We define X as a set of items of M and x as a single item, based on standard naming in machine learning. Furthermore, $X_1 \Rightarrow X_2$ defines an association rule between X_1 and X_2 and $D \subseteq M$ defines a collection of rows of M.

Support

Support defines the availability of an item set in M. This shows the frequency at which an item set, with certain values, will appear in relation to a target collection of samples. $|.|$ being element count.

$$supp(X) = \frac{|\{d \in D, X \subseteq d\}|}{|D|}$$

Example based on matrix 4.2:

$$supp(\Delta_G) = \frac{|\{d \in LM, \Delta_G \subseteq d\}|}{|\{LM\}|} = \frac{2}{4} = 0.5$$

or 50% of samples contain a Δ_G of 1

Confidence

Confidence defines the percentual correctness of a rule. E.g. if our item set X_1 appears three times, and in combination with X_2 one time, the Confidence of $X_1 \Rightarrow X_2$ will be 33% or $0.\overline{33}$.

$$conf(X_1 \Rightarrow X_2) = \frac{supp(X_1 \cap X_2)}{supp(X_1)}$$

Example based on matrix 4.2:

$$conf(H_{Old} \Rightarrow \neg\Delta_G) = \frac{supp(H_{Old} \cap \neg\Delta_G)}{supp(H_{Old})} = \frac{0.5}{0.75} = 0.\overline{66}$$

or in $66.\overline{66}\%$ of cases the rule $OldHeight \Rightarrow \neg\Delta_G$ is correct.

Lift

Lift defines the actual support of a rule $X_1 \Rightarrow X_2$ in comparison to the support that would be expected if X_1 and X_2 were independent. A Lift of 1.0 means that the rule has no basis. A higher Lift implies a stronger correlation between X_1 and X_2 , therefore the Lift can be considered as confidence of the rule.

$$lift(X_1 \Rightarrow X_2) = \frac{supp(X_1 \cap X_2)}{supp(X_1) * supp(X_2)}$$

Example based on matrix 4.2:

$$lift(H_{Old} \Rightarrow \neg\Delta_G) = \frac{supp(H_{Old} \cap \neg\Delta_G)}{supp(H_{Old}) * supp(\neg\Delta_G)} = \frac{0.5}{0.75 * 0.5} = 1.\overline{33}$$

this shows that the rule $OldHeight \Rightarrow \neg\Delta_G$ is more likely to fulfill than coincidence would suggest.

Based on these metrics it is possible to build association rules. For this, first basic, or elementary, rules are created, on top of these more complex ones. Graph 4.2 is an example of the process, rounded boxes symbolize rules, connecting arrows are rule forming steps. First, data is retrieved, in this case represented as a matrix. Then very basic rules, with one input variable only, are formed. Like the rule {Temperature ⇒ New Grain Size}, which links the simulated temperature and the new grain size. These are combined to more complex rules, by adding more items on either side. New rules are only formed if their confidence or lift, which is basically the statistical confidence that a rule is correct, satisfies a given threshold. No combination of rules can be more frequent than its component rules; this is called the downward-closure property (Agrawal et al., 1993). Thus rules can be created from the base and rules that are not a combination of already existing rules do not have to be examined.

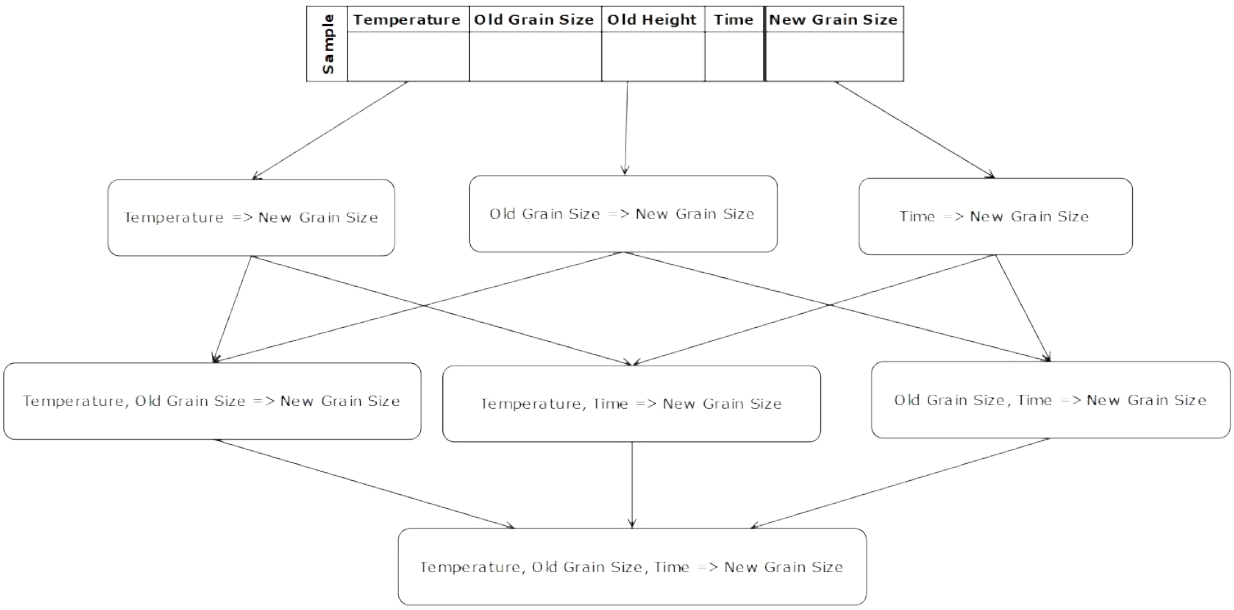


Figure 4.2: Example graph illustrating the process of association rule mining, going from an imaginary sample to association rules.

4.3 Dimensionality Reduction

Dimensionality reduction describes a group of mathematical algorithms and methods that allow for transferring high-dimensional data to a smaller space, usually 2 or 3 dimensions. Here dimensionality reduction is used to achieve two goals:

1. Make it possible to visualize items, their relationships and influence. Since displaying higher dimensional matrices is very difficult, by reducing the amount of dimensions to two or three, an easy-to-display version is achieved. But it is important not to miss necessary information. To guarantee visibility and readability figures created in this chapter, except Figure 4.3 will be in the appendix to allow for bigger image scale without disrupting text flow.

2. Calculate the importance of every parameter for the result. ARM extracted the underlying rules of the hot rolling process and provided which rules have a higher lift than average, thus being more probable. It also reduced the number of relevant input variables to only a few, namely those promising to contribute to qualities of the result. But it does not tell how strong each given input parameter affects the result, it only reveals how likely there is a relation at all between the parameter and the result as such. To fill this gap in knowledge dimensionality reductions is used. By comparing how much each input influenced the resulting 2 or 3 dimensional values weights can be extracted that represent the importance of each input parameter. These can be used to improve the machine learning model.

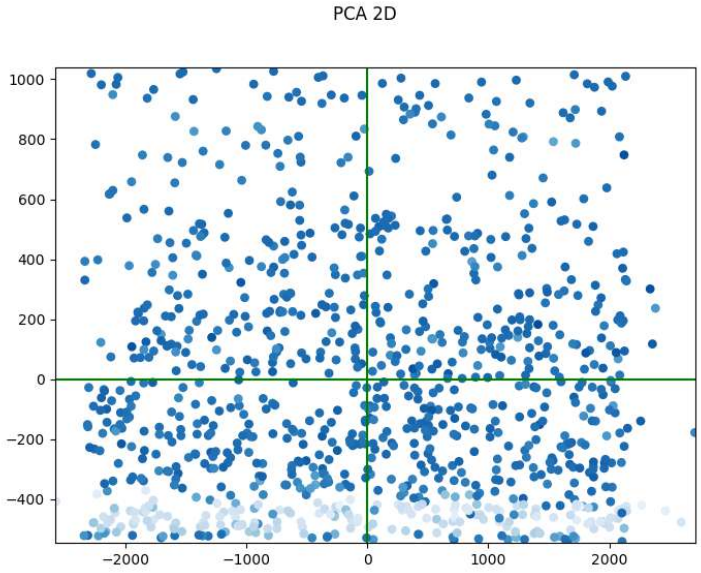


Figure 4.3: Example of Dimensionality Reduction using PCA,
77 Dimensions were reduced to 2 at 1000 samples.
Brightness shows change in grain size (Δ_G); clustering of passes with smaller grain size around the bottom is clearly visible

Two different methods will be used in this thesis: t-Distributed Stochastic Neighbor Embedding (t-SNE) and Principal Component Analysis (PCA). t-SNE is a non-linear dimensionality reduction technique, which is useful for transferring high-dimensional data into a space of two or three dimensions by representing similar values by near points and dissimilar values by distant points (van der Maaten and Hinton, 2008). PCA uses orthogonal transformation to transform samples into linearly uncorrelated variables, also known as principal components. The amount of principal components can be set and is lower or the same as the number of original values (Pearson, 1901). Figure 4.3 gives an example of dimensionality reduction using PCA, from 77 dimensions down to 2. Figure A.1 and A.2 are applications of dimensionality reduction using both PCA and t-SNE on a reduced hot rolling sample set, once down to two dimensions each and once to three. The resulting values are displayed as scatter plots. Color, ranging from a light to a dark blue, shows the difference in Δ_G, which is not known by or provided to t-SNE or PCA. The range between highest decrease and increase in Δ_G was normalized, higher values in Δ_G are represented by a darker blue. This shows that clusters or groups naturally occur, which means that these values must have had some precondition making them similar. The fact that these groups consist of primarily the same colors, depicts that the input data is somehow related to the result.

4.3.1 PCA and Bad Data

The first step to understanding the results produced by PCA and t-SNE is a sample run on a computable subset of data. The provided data, including computed items, contains 79 dimensions and around 1,600,000 samples, but this requires far too much memory to be practically useful at the given time. In a first test, PCA and t-SNE run on all 79 dimensions at 1000 samples, and points are color coded from light to dark blue, which reflects a change in grain size, which is the target size this paper wants to predict. Grain size is not given as a data point to either the PCA or the t-SNE model, which means grouped or clustered occurrences of similar Δ_G are based on unknown similarities between input values.

This first attempt, of PCA, resulting in Figure A.3 shows the limits of PCA, with 2 attributes in the input array having far higher values than all other items scaling becomes a problem. Those two dimensions were found out to be static computation variables and were removed from further calculations. As was to be expected, ARM, too, determines that these two attributes are not related to grain size. t-SNE, on the other hand, does not suffer as much from bad data points, which is a clear plus, but it suffers with regard to perplexity.

4.3.2 t-SNE and Perplexity

While t-SNE graphs give a great overview over problems, they suffer from 2 disadvantages. To start with, they are not stable, meaning running t-SNE twice on the same data may produce different scatter plots. While this is not very bad for visualization, as the graph can be computed as often as deemed necessary to create good visuals, it leads to a lack of certainty when analyzing its component buildup. Secondly, the hyper parameter perplexity, while being stable between 5 and 50, causes strong differences in the resulting data points, with no chance of knowing for sure which perplexity will lead to the best result; the one closest to the real influences of variables on the background process. To illustrate this point the same data was run through t-SNE several times with different perplexities shown in Figure A.5.

These images show the wildly different clustering that occurs on different perplexities, groups that are forming early seem to stay coherent, but spaces between groups seem to disappear while the whole scatter plot converges on a more unified shape.

4.3.3 From Visualizations to Knowledge

After having excluded PCA penalizing data, it is now possible to create better visualizations, that are human-readable. A new attempt at PCA and t-SNE with now 77 dimensions and 10,000 samples was then conducted and can be seen in Figure A.6.

The resulting Graphs A.6 show clustering. In the case of the PCA plot it appears that values that lead to bigger decreases in grain size gather around the bottom. On the other hand in the t-SNE scatter plots it appears that similar values form the outer borders of groups they adhere to.

Now all the dimensions are filtered to those which form association rules computed trough ARM, reducing the count from 79 to 14, which is far more manageable, but to be comparable to the previous experiments the same amount of 10,000 samples was be used.

Clear differences can be seen between the results of dimensionality reduction on all data and filtered data, in Figure A.8, with ARM filtered data showing more obvious clusters in both PCA and t-SNE, leading to the assumption that background processes are better represented. Based on the two still unresolved disadvantages in t-SNE, PCA is later selected for a more detailed weighting of the dimensions left by association rule mining, to determine which values have a higher influence on the resulting grain size.

4.4 Prediction

One of the goals of this thesis is to create a model that can predict the outcome of a pass, regardless of whether a relationship model using ARM has been created or not. The ability to predict the results of passes allows for a deeper understanding of hot rolling, if a white box model is used, by analyzing the way the prediction function is defined. It is also a huge help for designing pass plans. By making it possible to calculate the result of the next pass it is possible to plan several single passes ahead and link them together.

For this purpose, different machine learning techniques are used. Those will later be evaluated against each other. The process of predicting results from input data is neither trivial nor a perfectly precise one, since data and mill are prone to error. Thus it is only possible to approximate the result with some unavoidable uncertainty. The process of changing a machine learning tools' parameters, adapting it to the data, under consideration of a minimization function, is called fitting. Guessing a result based on data is called prediction analysis.

In machine learning algorithms or techniques are divided in classification and regression. Classification means the application of, sometimes predetermined, labels

to samples, in order to separate them by distinctive features or feature groupings. Regression, on the other hand, means predicting an exact result instead of a label. Both will be looked further into in their respective section, but first comes an overview over decision trees, which will be used in both classification and regression.

In this and following chapters modules from Scikit-learn will be used, especially for classification and regression models. Scikit-learn is a well documented python library for machine learning, it contains standard implementations for most machine learning techniques and auxiliary functions like grid search (Pedregosa et al., 2012).

4.4.1 Decision Tree and Random Forest

A decision tree is a very simple, usually binary, decision-making component that is used in various different classifiers as well as regressors, e.g. random forest or bagged random forest. A decision tree consists of several nodes, linked directly or indirectly to the trees' root. Each node is capable of making a simple decision based on attributes of the input it receives. That allows for inserting input in the root of a decision tree and traverse its branches until a leaf is reached. Every decision tree could be called a classifier, with each possible outcome being one label (Mohri et al., 2012). Figure 4.4 shows the predictions of two decision trees approximating a sine curve. The blue line shows the prediction of a decision tree with depth 2, the green line respectively with depth 5. The orange dots show, partly corrupt, samples of the curve. Depth defines how complex the paths in a tree may be, which is how many if/else decisions a path in a tree may have. Increased depth does not always lead to better predictions, since the tree may suffer from being too close to samples, especially erroneous ones, which is called over-fitting.

A random forest is a machine learning technique that uses multiple decision trees at the same time, building them when fitting to data. Small randomized variations are used in each tree and they collectively vote on a result. This allows for more variation in the learning process, as several instances or layers of mini-forest trees may be used. After a random forest has been created, it can be used to predict future inputs. A random forest can be improved in several ways; the most important one is the number of estimators, or trees, per layer.

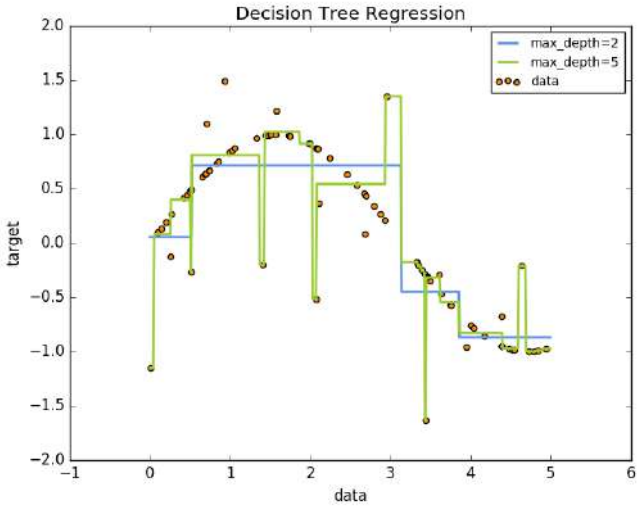


Figure 4.4: Exemplary decision tree regression using different depths.

4.4.2 Classification

Classification is a form of machine learning that takes data and tries to divide the samples into different classes called labels, according to the samples' content. A big distinction must be made between supervised learning and unsupervised learning. In supervised learning the labels into which data should be split are provided; take for example animals or flowers, which can be divided into their respective species. In unsupervised learning an algorithm creates its own labels. This thesis uses supervised learning, since result data exists, which makes it easy to define labels/classes. Exemplary labels would be *Height Delta*, Δ_G, Energy consumption in intervals or Machine constraints satisfaction.

4.4.3 Regression

A regressor is used to predict a continuous value instead of a label. A regression model can be seen as a function that approximates the resulting output for given input parameters. Regression is vastly more useful for predicting the results of a hot rolling process, since elementary qualities like grain size or electric consumption are not discrete values. Classification results can be evaluated quite easily, just by comparing whether the right or wrong class was selected. Regression, on the other hand, can only evaluate the accuracy of predictions. Multiple statistic measurements are used for this. Perfect prediction is nearly always unobtainable for big enough data samples. It is important to use a white box regressor, so that obtained results can lead to a deepened knowledge and understanding in the field they are applied to.

Metrics for Regression

To understand how good a regression prediction model is several metrics have to be established, which allow one to evaluate a model.

Setup Time: how long data preparation took, incl. ARM and dimensionality reduction.

Training Time: how long the fitting of a model to data took.

Total Time: Training Time + Setup Time

Explained variance Score: represents what is called the explained variance regression score. With Var for Variance, standard deviation squared, y being the desired output and $\hat{y}$ being the estimate output it is calculated by:

$$explained\ variance(y, \hat{y}) = 1 - \frac{Var(y - \hat{y})}{y}$$

Mean absolute error: represents l1-norm loss, absolute errors are averaged by division through the number of samples. n stands for the total number of samples.

$$MAE(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1} |y_i - \hat{y}_i|}{n}$$

Mean squared error: similar to the mean absolute error, but uses quadratic(squared) loss instead of absolute.

$$MAE(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2}{n}$$

Median absolute error: the median over all absolute error, particular interesting because of its robustness towards outliers.

$$MedAE(y, \hat{y}) = median(|y_1 - \hat{y}_1|,\ \ldots\ ,\ |y_n - \hat{y}_n|)$$

R^2 score: represents the coefficient of determination, normally denoted as R^2. R^2 measures the likelihood of future samples being predicted by the model. Ranging between 1.0 as best and -INF at worst, since it can be arbitrarily bad.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1} (y_i - \bar{y}_i)^2}$$

where

$$\bar{y} = \frac{\sum_{i=0}^{n-1} y_i}{n}$$

R^2 will be considered the main performance metric, shortened to performance, with others as support. Total time will be considered the main time metric, shortened to time.

Perceptrons and Multi-layer Perceptron Regressor

To deepen one's understanding of the differences between a black-box and white-box model, it is useful to understand the working of a black-box model, so it can be compared to the chosen white box models. For this purpose the very simple multi-layer perceptron (MLP) regressor was selected.

A perceptron is a function that takes vector input, attaches weights to it and calculates a binary output. This is normally done by setting a threshold that the scaled vector needs to cross.

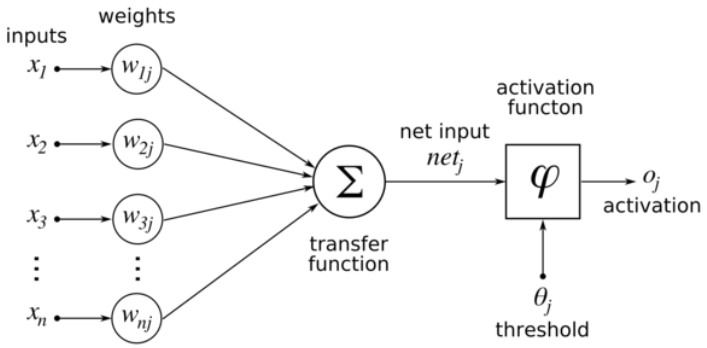


Figure 4.5: Example Perceptron.

$$f(x) = \begin{cases} 0 \text{ if } \sum_{i=1}^{n}(w_i * x_i) + b < \theta \\ 1 \text{ otherwise} \end{cases}$$

By combining Perceptrons it is possible to create a so-called neural network or, in this case, a multi-layer perceptron. This is commonly called a neural network because it is inspired by the neural networks in a human brain. Every perceptron or neuron only makes a very simple decision but, by linking them together, advanced functions can be approximated. The weights (w_i) of the perceptron get optimized using back propagation, which is minimizing a continuous error function, during training. This way the neural network slowly improves itself. The biggest problem of neural networks is that they are a black box approach. That means it is impossible to understand how the model works after fitting. This makes it useless for a further analysis of the process, which is why white box approaches like random forest are used in this thesis.

5 Results

Based on the theoretical background and methods presented in the previous chapter, results are collected and evaluated. Following the same order as Chapter 4, the results of association rule mining are presented first, showing the possibility to generate association rules from hot rolling data. Dimensionality reduction then is employed to improve these results through visualization and establishment of weights. Random forests are presented through exemplary classification and regression. This lays the foundation for the second main segment of this chapter: Improving Prediction Using ARM and Dimensionality Reduction. In this section the techniques extracted from previous results are combined to further improve the established prediction model. The improvements generated are then evaluated.

5.1 Association Rule Mining

This section describes processing of raw data into association rules that satisfy set requirements as detailed in Section 4.2. The used Database contains 68 different parameters/dimensions, with at least 10 more being easily computable. These parameters form the basis from which all rules will be calculated. Those initial 68 are:

Geometriefaktor	B1, B2, B3
Modellparameter	jelem, kelem
Hebelarmbeiwert	C1, C2, C3, C4
Walzgutgeometrie	hstart, bstart, lstart
Rekristallisationsmodul	dm, B, rx1, rx2, Qrx, n
Liesskurve	K, m1, m2, m3, m4, m5
Walzenparameter	mueroll, emodroll, nueroll
Temperaturen	thetainit, thetainf, thetaroll, thetah2o
Gef. < 1000	A_2, gg1_2, gg2_2, Qgg_2, D_2, gs1_2, gs2_2, Qd_2
Gef. > 1000	thetaswitch, dinit, A, gg1, gg2, Qgg, D, gs1, gs2, Qd
Therm. Rand.	rho, lambda, cp, alphaconv, alpharoll, alphah2o, epsilonrad, epsilondiss
Pass	Height, Width, Length, Speed, Rollradius, Starttime, Endtime, Scaling, Brake, Reel

From these values a few additional items can be calculated. There are also the output values, including grain size, whose prediction is the target of this thesis:

Computed Values	Output Values
Height/Width/Length Old [Previous]	Grainsize [µm]
Height/Width/Length Delta [Previous - New]	Force [MN]
Grain Size Old [Previous GS]	Energy Consumption [J]
Next Starttime	Δ_G
Cooldowntime [Endtime - Next Starttime]	
Pass count	

As this thesis has a data-driven focus, its goal is to work on the pure data without additional information, whenever possible, since it might not always be available. Thus the researched process, the result of this thesis, applies to a wider field of problems. It should be possible to exclude parameters that provide no value without background analysis based on their data's fingerprint.

With all necessary preparation complete, a schedule for the ARM process can be drafted.

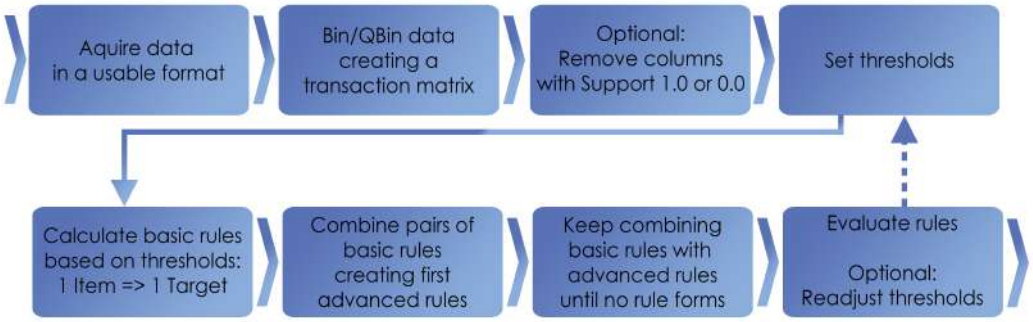


Figure 5.1: Flowchart of the ARM Process

5.1.1 Binning

Association Rule Mining works on discrete and especially binary values, which are not available in the given data; only continuous values are provided. Therefore, each parameter must be binned, that means n ranges, bins, are defined and the bin that contains the current sample is set to 1 or True and all others are set to 0 or False. Those new arrays are said to contain transaction lists, like a supermarket recipe. Here is a short example:

$$\text{Value1: } \{1.0,\ 2.5,\ 3.0,\ 1.7\ \}$$

For this 3 bins are selected, Bin1: (0, 1), Bin2: [1, 2), Bin3: [2, 3). Applying those bins to the data, a discrete / binary array is acquired.

Bin1	Bin2	Bin3
1	0	0
0	0	1
0	0	1
0	1	0

The resulting array has a size of the original array's size times the number of bins, that means too many bins make it uncomputable, while too few bins result in a lack of precision. When creating bins it is important to decide between bins of equal length, called Bins from now on, and bins that contain an equal amount of elements, called QBins. Two different binning techniques applied to the previous example create unequal bin contents. On the range 0-3 with a Bin size of 2 and equal length the following are created: Bin1: (0, 1.5), Bin2: [1.5, 3). QBins of equal content size would be: QBin1: (0, 2), QBin2: [2, 3)

Value1	Bin1	Bin2	QBin1	QBin2
1.0	1	0	1	0
2.5	0	1	0	1
3.0	0	1	0	1
1.7	0	1	1	0

Deciding between Bins and QBins is not trivial, since the sample arrangement changes, information in the data may be lost through bad simplification. But this can again be avoided by choosing a high enough amount of precision. Ideally each unique value would have its own bin, no data would be lost at all, but in this thesis 5 QBins and alternatively 10 Bins will be enough to give insight in the transactions of a rolling mill.

5.1.2 Setting Thresholds

Setting thresholds may be the most difficult task in the generation of association rules since thresholds are based on personal preference in risk or confidence. Different values can create vastly differently sized rule sets, with too low thresholds nearing exponential size.

There are two obvious ways to define thresholds for rules, either as Support and Confidence or as Lift, since Lift is basically just a combination of the other two. Using Support and Confidence gives better control over adjusting minimum thresholds since both values can be set independently, but it comes with the trade-off that a very high value in one of the values does not balance a low value in the other, since both must confirm to a minimum.

Support measures how often a rule appears in relation to its items' occurrences; for QBins each transaction has the same Support, since all bins are of the same size.

Bins, on the other hand, require a lower support since values may be arranged worse *. A good Support threshold function therefore would be:

$$Threshold := \frac{preference}{|bins|}$$

A *preference* of 0.5 ensures that for each Rule there is a Support of at least 0.5, meaning that the combination of Items and Target appears in at least 50% of cases.

Confidence measures how often a rule is found to be true and will be used as the second threshold for rules. One could argue that a rule only needs to more often be true than false meaning a Confidence bigger than 0.5, which obviously shows significance. When reliability is important, one has to incorporate the effect that randomness in data will produce imperfect sample representation Thus it is better to chose slightly higher values than 0.5 to reduce chance.

On the other hand a rule need not be correct more than half of the time to show relations, which do not need to mean significance. Even rules with lower confidence may signify some kind of connection, so it is important for the observer to decide what serves his use-case best.

For all further steps, a Support of 0.5 divided by the number of bins ($0.5/\#bins$) and a Confidence of 0.6 will be used, alternative Threshold will be a Lift of 4.

5.1.3 Building Basic Rules

A basic rule consists of one item and one target. As target for all rules the grain size is selected, binned in 3 bins with them representing, increasing, decreasing and staying relatively the same grain size. When given a Matrix X containing all input transactions, one per column and a Matrix Y containing the target transactions, it is necessary to define Support, Confidence and Lift Algorithms. Numpy, a python library for numerical operations is used for ease of use on array operations van der Walt et al. (2011).

```
import numpy as np
def Support(X, Y = none):
  supp = X[0]
  for i in range(1, len(X)):
    supp = supp * X[i]
    if Y is not None:
      for y in Y:
      supp = supp * y
  support = supp.sum() / supp.shape[0]
  return support
```

Listing 5.1: Support

Algorithm 5.1 takes 2 variables, both expected to be a matrix. First all vectors, which are the columns, of the X matrix are multiplied. Then, if a target matrix Y is given, all vectors of Y are multiplied on top of them. This is a very fast operation but it only works on binary vectors. In the end a vector *supp* exists that has a 1 at position $supp_i$ if every x_i in X and every y_i in Y are also 1, this way the overlapping rows of all vectors are calculated. The number of 1's in *supp* divided by the number of elements in *supp* gives the Support of $\{X \Rightarrow Y\}$, which is what is returned. Confidence and Lift get calculated according to 5.2.

```
def Confidence(X,Y):
  return self.Support(X,Y)/self.Support(Y)

def Lift(self, X,Y):
  return self.Support(X,Y)/(self.Support(X)*self.Support(Y))
```

Listing 5.2: Confidence and Support

These algorithms are identical to the definitions in the previous section. Finally it is possible to calculate all basic rules.

```
def GetBasicRules(xMatrix, yMatrix, sT, cT):
  rules = []
  for x in range(0, xMatrix.shape[1]):
    for y in range(0, yMatrix.shape[1]):
      xColumn = self.xMatrix[:,x]
      yColumn = self.yMatrix[:,y]
      conf = Confidence([xColumn], [yColumn])
      supp = Support([xColumn], [yColumn])
      if(supp > sT and conf > cT):
        rules.append([[(x)],y,[xColumn],[yColumn],supp,conf])
  return rules
```

Listing 5.3: Basic Rules

The GetBasicRules Algorithm, as in 5.3, needs 4 parameters, first the matrix of input transactions xMatrix and the matrix of target transactions yMatrix, followed by the thresholds for support and confidence. Now it loops through both of them and calculates the support and confidence for every combination of input and target columns, or more simply said, every item and every target. If the calculated support and confidence are both higher than their given global threshold, the rule is added to the list of Basic Rules. Every stored rule is defined according to the following format:

{Items, Target, Item Columns, Target Column, Support, Confidence}

Support and Confidence can also be replaced with Lift according to Section 5.4.

```
def GetBasicRulesLift(xMatrix, yMatrix, lT):
  rules = []
  for x in range(0, xMatrix.shape[1]):
    for y in range(0, yMatrix.shape[1]):
          xColumn = self.xMatrix[:,x]
          yColumn = self.yMatrix[:,y]
          lift = Lift([xColumn], [yColumn])
          if(lift > lT):
            rules.append([[(x)],y,[xColumn],[yColumn],lift])
  return rules
```

Listing 5.4: Basic Rules by Lift

Both Algorithms have equal runtime, since the computation of Lift is equal to the calculation of Confidence and an additional support. The algorithm itself has a runtime of

$GetBasicRules(xMatrix, yMatrix, lT) := |xMatrix| * |yMatrix| * Lift$

$= N * N * 1$

$= N^2$

which means at least N^2 operations are required to create the basic rules, which is still adequately computable.

5.1.4 Generating Advanced Rules

While calculating basic rules is very straight forward, the process of acquiring advanced rules is more complicated and allows for efficiency improvements. First, take a look at a dumb algorithm computing all rules, including advanced rules, using a brute force attempt:

```
def GetAllRules(xMatrix, yMatrix, lT):
  rules = []
  permutations = subLists(range(0, len(xMatrix))
  permutations.append(range(0, len(xMatrix))
  for x in permutations:
    for y in range(0, yMatrix.shape[1]):
      xColumns = self.xMatrix[:,x]
      yColumn = self.yMatrix[:,y]
      lift = Lift([xColumns], [yColumn])
      if(lift > lT):
        rules.append([[(x)],y,[xColumns],[yColumn],lift])
return rules
```

```
def subLists(list):
  sublist = []
  l = len(list)
  for x in range(0, l):
    row = []
    for y in range(0, l - 1):
      index = (x+y) % l
      row.append(list[index])
    sublist.append(row)
return sublist
```

Listing 5.5: Brute Force ARM

The Algorithm in 5.5 calculates all permutations of items, meaning all possible item combinations, of which there are $n!$ and calculates for each whether it is a rule or not. While being unfeasible this would be the required computation for all rules if the downward-closure property of ARM did not exist. Downward-closure defines that a rule can only be interesting, which means having a Lift or Confidence and Support above the threshold, if all of its sub-rules are interesting (Agrawal et al., 1993). Given this knowledge it is only required to calculate all basic rules, combine those to a set of new rules, and keep combining new rules with the basic rules until no new rule appears during a cycle as in 5.6. An index register helps calculating each rule only once, since there are always multiple ways to calculate a rule, depending on which basic rule is added first.

```
def GetAllRules(basicRules, sT, cT):
  all = []
  current = basicRules
  new = []
  ruleIndexList = []

  while(len(current) > 0):
    for rule in current:
      for ruleAdd in basicRules:
        if(ruleAdd[0] in rule[0]):
          continue

        com = list(set(rule[0] + ruleAdd[0]))
        target = rule[1]
        if([com, target] in ruleIndexList):
          continue

        xColumns = rule[2] + ruleAdd[2]
        yColumn = rule[3]
```

```
            supp = self.Support(xColumns, yColumn)
            conf = self.Confidence(xColumns, yColumn)
            if(supp > sT and conf > cT):
              newRule = [com, target, xColumns, yColumn, supp, conf]
              ruleIndexList.append([com, target])
              new.append(newRule)

        all = all + current
        current = new
        new = []

    return all
```

Listing 5.6: ARM using Downward Closure

GetAllRules takes 2-3 parameters, first the basic rules, then thresholds. Three lists are created, tiered by the time they will fill up, *all* will contain all rules that are confirmed and need no further expansion, *current* will contain the current working set of rules and *new* will hold rules just generated, that are not ready to be expanded. A forth list will hold the index of each created rule to enable fast lookup whether a rule already exists. Now looping on a working set called *current*, each a basic rule and a rule from *current* will be selected. First a check if the set *current* contains the basic rule is done, then a check if their sorted combination, *com*, is already registered. Afterwards support and confidence of the combined rule is computed and it is added to *new* if it passes the thresholds. After combining all *current* and basic rules *current* is added to *all* and *new* becomes *current*, and the process is repeated until no new rule is found and the working set is added to *all* which then gets returned.

This algorithm may still take $N!$ steps in the worst case, which would be the case of every rule being interesting, but in practice will resolve much faster, with its speed depending on the quality of threshold selection.

5.1.5 Visualizing Rules

To understand generated rules it is always helpful to generate visualization. Rules consist of smaller rules, down to basic rules, therefore graphs are great for connecting the associations they represent. Here are two possible graphs, based on real data gathered by ARM of the hot rolling process, that can be created from association rules that help to broaden their representation of knowledge.

1. Rule Interconnectivity Graph 5.2: A graph that features each rule as a node and connects nodes that contain at least one shared item with edges. Each node contains shortened item names, bins ⇒ grain size scale, and in a second and

third row the calculated confidence and support of the given rule. This graph gives a good overview over the complexity that rules reach, and shows how their confidence and support decreases when adding additional items. It is easy to recognize important advanced and basic rules, that are at the top and bottom.

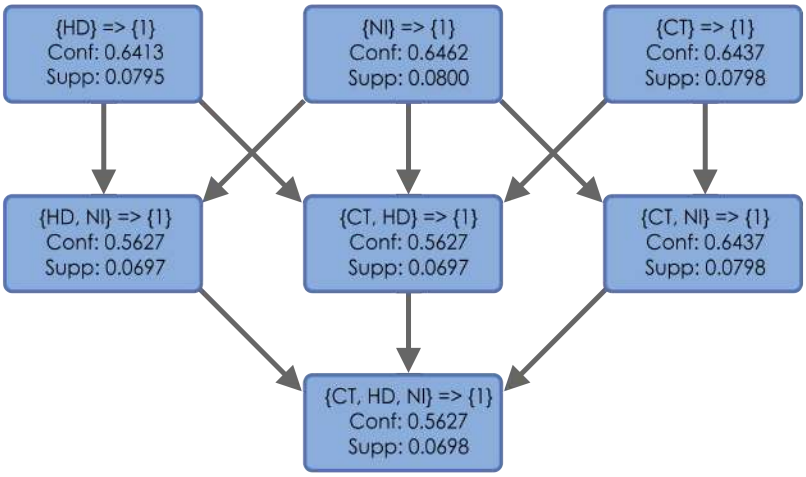


Figure 5.2: Rule Interconnectivity Graph

2. Item Interconnectivity Graph 5.3: A graph that consists of each binned item that is in at least one rule. Each bin of an item becomes a node. Basic rules are used as edges between item nodes. This graph shows which items correlate with each other and the degree of associations any item has.

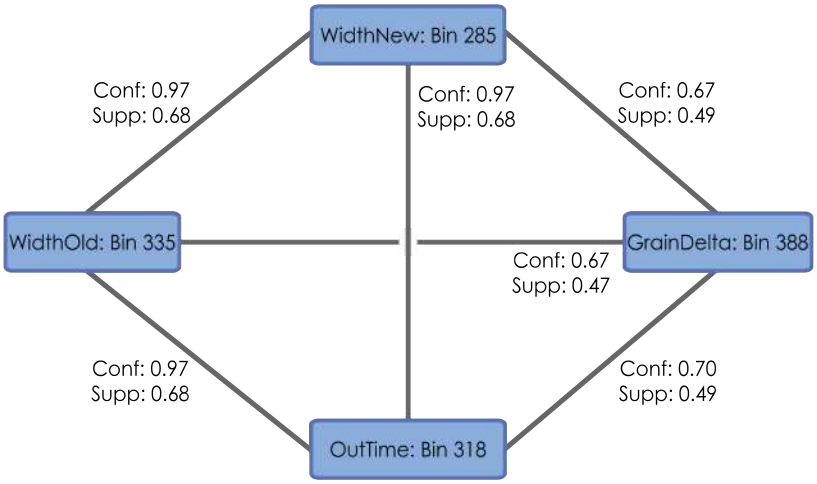


Figure 5.3: Item Interconnectivity Graph

5.1.6 Evaluation

This section gathers the results from previous sections and evaluates the ARM process. To evaluate ARM it is essential to test the performance of generating association rules. For this, rules were generated several times on randomized sample sizes and on 76 dimensions, with 2 different sets of thresholds. The results are shown in table 5.1.

Table 5.1: Applied ARM Results

Sample Size	Full(1,600,000)	1,000,000	100,000	10,000	1,000	100
Time	248.053 sec.	129.580	9.819	1.150	0.658	0.579
Basic Rules	69	69	69	71	77	72
Unique Items	17	17	17	17	18	17

Support of $0.25/\#Bins$ and a Confidence of 0.6, or a Lift of 4.

Sample Size	Full(1,600,000)	1,000,000	100.000	10.000	1.000	100
Time	231.758 sec.	137.558	11.077	1.166	0.660	0.586
Basic Rules	41	41	41	42	47	33
Unique Items	14	14	14	14	15	13

Support of $0.5/\#Bins$ and a Confidence of 0.7, or a Lift of 4.

Both tables detail multiple interesting properties of ARM. First of all, both show a linear increase in run time, in relation to sample size, as suggested by the code used. Thus ARM allows for calculation of rules on large scale projects. In both sets the generated rules for 1.6 million, 1 million and 100.000 are the same, this shows that even with smaller samples sizes ARM has a good chance at accurately representing the rule forming behavior of a larger sample size, given that its distribution of samples matches larger ones.

In the first test, rules generated in runs with smaller sample sizes contained nearly all the rules established when using more samples. But there are also additional rules, which where deemed unfeasible on larger sample sizes. These rules could become feasible again with lowered requirements on rules, which means one must decide whether a decrease in thresholds seems useful.

Remarkable is now that in the second set the 100 samples run had drastically less rules than all runs on larger samples, exactly the opposite of the first set. This illustrates the inaccuracy of this method on tiny sample sizes.

Furthermore, the items selected in the first table for rules in the first 4 iterations are equal. These are also the same ones as in the run on 100 samples. When computing rules for 1000 samples, ARM found a different rule, an outlier. Again it is shown that such small sample sizes are vulnerable to error, for example,

a bad random selection, which is why several runs, either on different random distributions or sample sizes are favorable.

To conclude, it can be said that ARM can efficiently compute rules. ARM is very unstable on small sample sizes, which is why it is not to be recommended for these. Robustness or stability then sets in quite fast on bigger sample sizes. The value of generated rules will be estimated in chapter 5.4.

100.000 samples appear to be a good trade-off between time and precision for the task at hand and therefore the results generated by runs on 100.000 samples will be used in this thesis. The 14 selected items from the second set will be used in later chapters, which are:

{*Temperature, HeightInit, WidthInit, LengthInit, HeightNew, WidthNew, InTime, OutTime, HeightOld, HeightDelta, WidthOld, LengthDelta, GrainSizeOld, NextInTime*}

5.2 Dimensionality Reduction

The theoretical background section on dimensionality reduction has shown interesting effects and concluded that PCA will be used to extract knowledge from hot rolling data. This is done in the following paragraphs. Furthermore PCA and t-SNE are compared and evaluated.

5.2.1 Knowledge Extraction trough PCA

This section illustrates the process of gaining knowledge from the result of running PCA with an example. PCA was run on the whole dataset but only with the 14 dimensions that ARM determined to be interesting, resulting in the first figure of figure A.8. One can clearly see that values are mainly divided by grain size along the x and y axis of the graph, with lighter blue towards the bottom left and darker blue to the right and top-right. Table 5.2 presents a sample data point:

The first column contains the name of each item, with the second containing its raw value. The third and forth column contain the components assigned to each item by PCA, once for the x dimension and once for the y dimension. A component is the value multiplied by an item transforming its value into PCA space. All component item multiplications are then added on top of each other to create the final value in the given dimension. For an easier understanding, the fifth and sixth column contain the calculated values of $item * component$. From here a first guess can be made about what values affect the resulting point most. At first sight *InTime*, *OutTime* and *NextInTime* have the biggest influence on both the x and y coordinate of the resulting point. All three are related to time

Item	Raw Values	X	Y	PCA X	PCA Y
Temperature	1190.4530	0.0002	-0.0051	0.2169	-6.0359
HeightInit	140.0000	0.0032	-0.0022	0.4477	-0.3031
WidthInit	100.0000	-0.0005	0.0005	-0.0537	0.0468
LengthInit	300.0000	0.0184	0.0313	5.5131	9.3959
HeightNew	121.6300	-0.0083	-0.0832	-1.0127	-10.1193
WidthNew	100.0000	-0.0005	0.0005	-0.0537	0.0468
InTime	266.3060	0.5816	0.3819	154.8743	101.6901
OutTime	266.5660	0.5819	0.3875	155.1266	103.3023
HeightOld	127.2500	-0.0151	-0.0961	-1.9203	-12.2327
HeightDelta	-5.6200	0.0068	0.0129	-0.0380	-0.0727
WidthOld	127.2500	-0.0151	-0.0961	-1.9203	-12.2327
LengthDelta	15.2400	-0.0254	0.1640	-0.3869	2.4990
KornOld	103.9800	-0.0428	-0.1051	-4.4502	-10.9256
NextInTime	354.3420	0.5654	-0.7997	200.3606	-283.3622

Table 5.2: Sample PCA data point.

and together accurately define the cool down time, which means the time it took for the slab to cool down before it got rolled again or finally measured. This comes as a mild surprise, since it implies that the change in cooling time of a block is the most important factor in grain size prediction, the same conclusion Hong and Park (2003) came to. Now a problem is that these values may vary wildly in average size, therefore it is recommended to use the mean as a way of calculating components' influence as in Table 5.3.

Item	Mean	X	Y	Mean * X	Mean * Y
Temperature	1100.2328	0.0002	-0.0051	0.2005	-5.5784
HeightInit	134.9321	0.0032	-0.0022	0.4315	-0.2921
WidthInit	100.9188	-0.0005	0.0005	-0.0542	0.0472
LengthInit	268.0752	0.0184	0.0313	4.9264	8.3960
HeightNew	68.4296	-0.0083	-0.0832	-0.5697	-5.6932
WidthNew	100.9188	-0.0005	0.0005	-0.0542	0.0472
InTime	211.1767	0.5816	0.3819	122.8130	80.6387
OutTime	211.5748	0.5819	0.3875	123.1248	81.9915
HeightOld	82.4388	-0.0151	-0.0961	-1.2441	-7.9250
HeightDelta	-14.0093	0.0068	0.0129	-0.0948	-0.1812
WidthOld	82.4388	-0.0151	-0.0961	-1.2441	-7.9250
LengthDelta	101.5815	-0.0254	0.1640	-2.5786	16.6567
KornOld	85.8721	-0.0428	-0.1051	-3.6752	-9.0229
NextInTime	206.6723	0.5654	-0.7997	116.8617	-165.2729

Table 5.3: Sample PCA data point, adjusted by means.

Again the time dependent values seem to dominate all other values. This leads to the assumption that the timing of a slab's rolling might be the best indicator to predict grain size. But a change in over 20% of the average determined height of some items shows the need to be careful and calculate means, especially with NextInTime having dropped by nearly 90 points, below the other two time related values. Items can now be ordered by importance but can also be assigned a quality indicator, a continuous score, with the biggest entry having a quality of 1.0 and others a matching percentage. These values are useful as weights, since they represent a scale based on the maximum value.

Item	X	Y
Temperature	0.0016	0.0338
HeightInit	0.0035	0.0018
WidthInit	0.0004	0.0003
LengthInit	0.0400	0.0508
HeightNew	0.0046	0.0344
WidthNew	0.0004	0.0003
InTime	0.9975	0.4879
OutTime	1.0000	0.4961
HeightOld	0.0101	0.0480
HeightDelta	0.0008	0.0011
WidthOld	0.0101	0.0480
LengthDelta	0.0209	0.1008
KornOld	0.0298	0.0546
NextInTime	0.9491	1.0000

5.2.2 Evaluation

Calculating PCA 2D for 1,000 passes in 14 dimensions took 0.007 seconds, comparable to this are PCA 3D at 0.006 seconds, t-SNE 2D at 7.642 seconds and t-SNE 3D at 13.857 seconds. Interestingly, PCA2D takes longer on average than PCA 3D while the opposite is the case for t-SNE. While these computation times are generally very small, there is a clear difference between PCA and t-SNE with a magnitude of 10^3. Now to put these values in relation all procedures are also run on a full set, all 76 dimensions. 1,600,000 passes on PCA 2D take 8.860 seconds and on PCA 3D take 8.998 seconds, which is still practically useful. t-SNE 2D and 3D were not computable at this sample size with 64 gigabytes of RAM and the standard t-SNE implementation provided by Scikit-learn (Pedregosa et al., 2012). More sample sizes and data are shown in table 5.4. With the second dimension being in thousands, E.g. 14x10 means 14 items, 10,000 samples.

With limited memory t-SNE could no longer be calculated after 25,000 samples. From this data the limitations, especially on size, of t-SNE can be seen. Computation time is also grows more rapidly for t-SNE than PCA. This is not useful for the analysis of huge datasets. The following observations can be made:

Items	14x10	76x10	14x25	76x25	14x1,600	76x1,600
PCA2D	0.02s	0.24s	0.05s	0.31s	3.78s	8.86s
PCA3D	0.02s	0.05s	0.05s	0.12s	3.96s	9s
t-SNE2D	224.84s	221.24s	778.26s	780.53s	-	-
t-SNE3D	149.10s	145.37s	975.27s	1025.89s	-	-

Table 5.4: Computing time of PCA and t-SNE.

1. Dimensionality reduction is cheap; in computation time. PCA allows for huge datasets to be reduced very fast, t-SNE is applicable for any non real time problem, given the amount of required RAM is reasonable.

2. Setting up PCA is easy, doing the same with t-SNE is hard. This does not mean that either results are superior at this point, t-SNE just is more volatile to adjustments of hyper-parameters, especially perplexity.

3. The components coming together in PCA can clearly be distinguished, showing that some items are more influential than others.

4. Initial dimensionality of items does not make much of a difference on t-SNE but a huge one on PCA. ARM seems like a very good initial step before computing PCA. ARM with stable results showed to take around 9-11 seconds in the previous evaluation. With a difference of over 5 seconds between 14 and 76 dimensions at 1.6 million samples it is indicated that at higher sample sizes ARM in combination with PCA will be faster than PCA alone on all dimensions.

5.3 Random Forest Results

Random forest is the predictor most used in this thesis, to clarify its usage this section presents an exemplary classification as well as regression using random forest. Usage of classifiers and regressors is detailed at the same time.

5.3.1 Exemplary Classification

This section deals with classification using a random forest. The goal of this classifier is to predict whether the grain size will shrink, grow or remain stable. Therefore 3 labels are required, which leads to using a function $Label(\gamma)$ which returns the appropriate label for a given Δ_G: γ. Δ_G describes the shift in grain size, γ is the Δ_G of a sample, and $\epsilon \in \mathbb{R}$ is a chosen threshold variable. 1.0 is set for a growing grain size, -1.0 for a shrinking grain size and 0.0 if γ is near 0, in the threshold ϵ.

$$Label(\gamma) = \begin{cases} \text{-1 if } -\epsilon > \gamma \\ 0 \text{ if } \epsilon > \gamma > -\epsilon \\ 1 \text{ if } \gamma > \epsilon \text{ otherwise} \end{cases}$$

Matrix features for classification of Δ_G are:
Temperature | Height | Result Height | Speed | Force | Grain size

First, data is acquired and translated into a useful data container, then a random forest classifier is built, with 400 trees or estimators, and fitted on the training samples. Finally a prediction is made and scored. A training size of 2188 passes and a test size of 547 passes is used.

Output for epsilon of 0 and therefore only 2 labels:

RandomForest Score: 0.8464
Classification report for classifier RandomForest:

label	precision	recall	f1-score	support
-1.0	0.87	0.95	0.91	441
1.0	0.67	0.41	0.51	106
avg / total	0.83	0.85	0.83	547

Training Time: 5.3783 sec.

Output for epsilon of 0.5 and 3 labels:

RandomForest Score: 0.8099
Classification report for classifier RandomForest:

label	precision	recall	f1-score	support
-1.0	0.83	0.95	0.89	418
0.0	0.50	0.03	0.06	31
1.0	0.67	0.44	0.53	98
avg / total	0.78	0.81	0.78	547

Training Time: 5.3087 sec.

The used sample size is comparatively low, as these values are only intended for orientation. In this example the introduction of an additional label decreases the precision of prediction by around 5%, which means the problem got harder.

5.3.2 Exemplary Regression

This section shows regression using a random forest regressor. A random forest regressor functions according to a random forest classifier but on continuous values as leafs instead of labels. To set these results in relation to other regressors, an MLP regressor is used. MLP regression is a neural network variation, which makes it a black box model not suitable for solving this thesis problem, but it is suitable for a quick performance comparison. Code similar to 5.3.1 is used to create a data container and both hyper parameter tuned regressors are fitted to the data.

Once unaltered data is used and once normalized data. The matrix and hyper parameter setup for the random forest regressor is identical to the one in 5.3.1. The MLP regressor is defined as:

```
MLPRegressor(hidden_layer_sizes=(200, ), activation='relu',
solver='lbfgs', learning_rate='constant',
shuffle=True, warm_start=False)
```

Results	Exp.Var.	Mean Abs.	Mean Sqr.	Med.Abs.	R^2	Time
RF Raw	0.9646	6.0768	82.5105	4.0553	0.9645	5.147s
RF Nor.	0.9647	0.0317	0.0022	0.0214	0.9646	5.684s
MLP Raw	0.8457	13.6859	358.2439	10.8017	0.8457	1.729s
MLP Nor.	0.9590	0.03272	0.0026	0.0228	0.9590	1.418s

Table 5.5: Results of random forest- and MLP regression on raw and normalized data.

One can see that even with a low sample size a very accurate prediction can be made using MLP as well as random forest. Predictions of MLP are improved significantly after normalizing the training samples, while random forest already produces better results on raw data. On the other hand, MLP is drastically faster than random forest, but is a black-box model. Furthermore, speed at such low datasets may be deceptive, since the training time of a neural network is mostly based on the time it takes to converge, which may be much easier with smaller sample sizes. One has to weigh the disadvantages and advantages of both and see how they affect the use case in question.

5.4 Improving Prediction Using ARM and Dimensionality Reduction

When looking at the results from the previous chapters, it is important to not only to look at what has been achieved, but what better results could be achieved by using the same data, a data-driven approach to improve the predictions made.

For example, with some machine learning tools better initial or first chunks of data lead to faster and better results. The first approximation steps of the prediction function are far closer to the desired result. Better data in this case means data that is close to the unknown "real" function that lies behind the data (Smola and Vishwanathan, 2010). This is an example where ARM could become useful, as ARM shows which values share a relationship. Dimensionality reduction may prove just as useful, making it possible to arrange selected items by importance for the result. Furthermore, the selection of the order of training data as well as the divide between training and test data, is not trivial. Averaging over all data

may lead to wrong results because not all data points are related to the result. The table below is a concise simplified example of the transactions in a hardware store, each column showing an item and each row showing how many items were sold:

Hammers	Nails	Gloves
1	100	1
0	100	0
0	100	0
0	100	0
0	100	0

When assuming the goal is to predict whether a pair of gloves is bought, averaging all data creates this result:

Average hammers	Average nails
0.2	100

From this data follows that the average transaction contains a lot of nails and a low number of hammers, so the neural net would be trained with samples containing many nails. If, instead, random samples are chosen there is a high chance that samples which do not contain hammers are used. Data points or values that are rare are easily missed or misrepresented using random selection. ARM now makes it possible to recognize that there is a relationship between hammers and gloves, the rule $\{Hammer \Rightarrow Gloves\}$ has the highest possible Lift at 1.0. This shows that samples containing hammers are more conclusive for a predictive for whether gloves are bought and should be used to train a regressor. But also samples that do not contain hammers need to be included; a balance between samples containing hammers and those that do not is to be sought.

5.4.1 Concepts for Combining Several Methods

For improving a prediction model first ideas on how to improve prediction must be collected, which will be run and tested in chapter 5.4.2. Some ideas seem obvious but should still be illustrated.

1. Using an ARM filtered set of input dimensions instead of a full set. The central idea of this thesis for improving predictions is using a more focused array of variables, in order to remove noise and increase speed. Less important dimensions may have wildly varying content which may throw our model off while providing little knowledge to the problem. Using less dimensions comes with the benefit of reduced memory consumption as less variables have to be stored, and may therefore enable the process to actually run on a machine that previously could not support it. Furthermore an improvement in speed is to be expected, but it is questionable if the calculation cost of ARM will be lower than the gained speed when creating a prediction model.

2. Using a Dimensionality Reduced set. While the memory consumption and training speed is certain to be minimal with this approach, it is uncertain if this will produce precise enough results since knowledge is certainly lost during dimensionality reduction, for a gain in visualization and information of input weighting.

3. ARM filtered and Dimensionality reduction weighted sets. For this dimensions will be filtered through ARM, then inputs will be ordered by their importance according to PCA, which means an attempt at selecting a more balanced set from the entirety. This way for example, it turns out that temperature is the most important factor, and 10% of samples should be used to train the model, now samples are ordered by temperature, and every 10th entry will be selected, thus a guaranteed range over all variations in temperature will be selected. Random selection may chose poorly with regards to focus on variance in temperature.

5.4.2 Detailed Process

Improving predictions is a multi-step process, several stages are brought together, with knowledge generating processes. Going from raw data to ARM filtered data enriches knowledge, revealing which items are favored by ARM. Going from either raw data or ARM processed data to dimensionality reduced data creates knowledge as well, determining which items seem more influential than others. Now these techniques can be used in a multitude of orders, with some more lending than others.

Raw Data

To create a reference point, it is advisable to look at the performance of simply running random forest regression on raw data. This means all dimensions and different sample sizes. As before, samples are expressed in thousands. A split of 0.8 between training and test set will be used. Meaning out of 100 samples 80 will be used for training and 20 for verification. The random forest was configured with 100 estimators and a depth of 4.

Samples	Exp.Var.	Mean Abs.	Mean Sqr.	Med.Abs.	R^2	Time
1500, 77	0.9945	1.5316	9.4457	0.5626	0.9945	922.055s
500, 77	0.9911	2.0872	15.3020	0.9171	0.9911	274.576s
10, 77	0.9699	3.9884	52.5586	2.0404	0.9699	3.075s
1, 77	0.9273	7.2094	133.7474	4.2940	0.9265	1.166s

Table 5.6: Results of random forest on raw data.

It can be seen that a random forest already produces very good results on this data set. It is questionable whether precision can be increased since this set contains all available data and already has a R^2 of 0.995 which is similar to 99.5% precision. Reductions in memory size or speed on the other hand may very well be reachable though.

Comparing Techniques

To be able to see the firmness of results produced in this chapter, it is elementary to compare multiple regressors to make sure that an adequate one was chosen. Therefore the a reduced sample set will be run on a multitude of regressors and a classifier. As classifier the random forest classifier was chosen, which directly shows the gap in performance of classification and regression on this problem. The chosen regressors were the random forest regressor but also:

Linear Regressor, a regressor based on Ordinary Least Squares, or OLS, a well known statistical algorithm for approximating unidentified parameters in a linear regression model (Pedregosa et al., 2012).

LinearSVR, or Linear Support Vector Regression, a support vector machine with associative learning algorithms; while being linear it can use what is known as a kernel-trick to compute non-linear functions by mapping items into high dimensional feature space (Pedregosa et al., 2012).

MLP Regressor, a black box regressor using neural networks, in this case a multi-layer perceptron (Pedregosa et al., 2012).

All models run on the same sample set, except for the classifier which will use a binned version of the grain size, since it can not work on continuous values and the MLP Regressor will run on normalized values, since these perform better for it.

Method	Exp.Var.	Mean Abs.	Mean Sqr.	Med.Abs.	R^2	Time
RF Class.	0.9515	0.1378	0.1719	0.0000	0.9514	34.551s
RF Reg.	0.9802	3.2641	34.1467	1.7572	0.9802	31.587s
Lin.Reg.	0.8053	12.9357	335.4751	9.5946	0.8053	0.090s
Lin.SVR	0.7823	18.0632	552.2803	15.5228	0.6795	21.784s
MLP.Reg.	0.9392	0.0305	0.0022	0.0209	0.9392	78.762s

Table 5.7: Results of an array of algorithms, 100,000 samples with 14 items, according to ARM, where used.

From Table 5.7 a number of observations can be made. First of all, random forest actually performs the best of the selected algorithms but not only the regression variant but also the classification version. This must be seen with a grain of salt,

as the classification is only using binned values. Filing an item in the correct bin is typically much easier than calculating a continuous value. The MLP regressor performs similarly to the random forest; a better hyper parameter tuning may even improve the result a bit. The linear regressor and support vector machine, on the other hand, perform vastly worse, dropping by more than 17 percentage points and with much higher error values. Secondly, linear SVR and especially linear regression were much faster than random forest, peaking in the linear regressor training in just 0.09 seconds instead of the 31.5 seconds of the random forest regressor. This may be a very good alternative for machines with very low computational power and less complex problems. The MLP regressor, on the other hand, took more than twice the time of the random forest while performing worse and being a black box model; this excludes it from practical application for this problem.

ARM Filtered Data

Now the same runs as in the chapter on raw data are done with ARM filtered Data, as detailed in the first improvement strategy:

Samples	Exp.Var.	Mean Abs.	Mean Sqr.	Med.Abs.	R^2	Time
1500, 14	0.9908	2.0054	15.7259	0.8421	0.9908	625.4363
500, 14	0.9858	2.6694	24.0719	1.3251	0.9858	154.4618
10, 14	0.9684	4.3295	55.1340	2.4523	0.9684	2.5947
1, 14	0.9619	5.7816	74.7008	3.7868	0.9617	1.1465

Table 5.8: Results of random forest on arm data.

These results are very similar in precision to the results of unfiltered data as in Table 5.6, with a difference of less than 0.004-0.005 in higher sample size categories. This is an amazing result, since the amount of data is way smaller but the precision is nearly equal. This proves that ARM recognized those dimensions that carry most knowledge very well. Even better is that on a small sample size of only 1000, ARM selected results outperform Raw by 0.04 or 4%. It is to be noted that, while sample selection was random, both use the exact same samples, just with most items filtered out. From this it can be concluded that the ARM filtered samples lead to slightly better precision. The increase in precision is probably based on a lower level of Noise, which means useless data. Which clogs the learning process, especially on small sample sizes, where it can not be learned to get ignored sufficiently.

Memory requirements have also been drastically reduced, from 75 to 14 dimensions, which is a reduction by 72%. This of course may vary wildly depending on how bloated data is.

Finally, and maybe most significant, is the reduction in time, by 50%-30% on higher sample sizes and smaller amounts on lower sizes, though this was to be expected based on the results in precision. This means a reduction in computational requirements; a less advanced machine, like in a production setting, may be able to perform just as well on ARM reduced data.

Dimensionality Reduced Data

Next in line is a look at the performance of random forest on data that has first been filtered with ARM, down to 14 dimensions, and then reduced by dimensionality reduction techniques. The previous results have illustrated that there is no reason for reducing from full data, since filtered data still seems to contain nearly all knowledge. Also, raw data requires vastly too much memory for a dimensionality reduction process like t-SNE.

Method	Exp.Var.	Mean Abs.	Mean Sqr.	Med.Abs.	R^2	Time
PCA2D 1500	0.7459	10.6999	434.6544	4.0237	0.7459	236.260s
PCA3D 1500	0.8611	7.6763	237.6386	3.0736	0.8611	267.226s
PCA2D 500	0.6660	12.9750	565.5122	5.5345	0.6660	58.471s
PCA3D 500	0.8277	9.1665	291.8002	4.3595	0.8277	67.944s
PCA2D 10	0.6102	14.9319	680.4091	6.5984	0.6097	1.492s
PCA3D 10	0.7904	10.9920	366.2787	5.7820	0.7899	1.680s
TSNE2D 10	0.7699	10.8508	401.2371	5.0677	0.7698	1.536s
TSNE3D 10	0.8822	7.9500	205.3981	4.3615	0.8822	1.634s

Table 5.9: Results of random forest on arm/dimensionality reduced data.

Results from the runs show three notable aspects. First of all, they are much faster, and, of course, memory efficient than their counterparts. Second of all, there is a huge drop in precision, which means quite a steep loss in Knowledge. The reduced data contains less information on the process, and its values correlate less with the grain size than hoped. Thirdly, and most interestingly, t-SNE seems to perform much better than PCA, even through its uncertainties like perplexity and unstable results. A t-SNE reduction on a bigger sample size may yield results actually close to ARM results.

Sorted Data

After having shown that dimensionality reduced data contains less information, it might be a good idea to attempt to use the order that dimensionality brought to prioritize data. Dimensionality reduction showed *InTime*, *OutTime* and *NextInTime* to be the dimensions of most importance. For this, two sorting techniques were used.

First, the data was sorted ascending by total value, of the most prominent features, according to dimensionality reduction.

Second, data was sorted again but this time according to the distance of each entry to the items mean; this way the distribution may be favorable.

Then every tenth element was chosen, thereby reducing the size by 90%.

To have a reference point, random shuffled data and badly sorted one will be listed. Data was badly sorted by choosing an item with high influence, then sorting according to it and selecting the lower 10 percent instead of a range over all samples. All sets are of equal sample size.

Sorted:	Exp.Var.	Mean Abs.	Mean Sqr.	Med.Abs.	R^2	Time
Random	0.9679	4.3605	56.0436	2.5346	0.9679	3.342s
Badly	0.1565	23.2209	1488.6399	10.9008	0.1336	2.800s
By Total	0.9695	4.5133	54.0417	2.6599	0.9695	3.251s
By Mean	0.9685	4.5714	55.8552	2.6905	0.9685	2.961s

Table 5.10: Results of random forest on arm/dimensionality reduced and sorted data.

The most outstanding result here is how bad the results for badly sorted data are. Scoring only an R^2 of 0.1336 and with a Mean Square Error of 1488.6399 these values are worse than blind guessing. This illustrates that at least random shuffling is a must. Sorted results on the other hand, prove to be better than random shuffled results, but only ever so slightly. Both perform a bit better and come with reduced training time, which is a trade-off to the insignificant time requirement for sorting. On much bigger sets, random shuffling may prove to be equally well performing, but only if the sample distribution is equal.

5.4.3 Evaluation

This chapter has proven that random forest is an adequate machine learning technique for the given problem, combining a way to handle the non linearity of the problem in a small amount of time. Linear regression models are less precise, and their faster speed is not a good enough trade-off with current computational power.

ARM was shown to be a great improvement. It reduced the computation time by a good margin with nearly no loss in precision. Memory usage has fallen sharply by the ability to exclude most dimensions. Furthermore, it allowed for the use of additional steps, like dimensionality reduction.

Dimensionality reduction has shown to be a great tool for visualization in Chapter 4.3. It allowed for weighting items and building a priority list which references the

actual influence of items quite well, to the point of reaching slightly better results than ARM on its own. Whether this makes enough of a difference to justify the additional cost of calculating these priorities depends on the given problem and can not be said for all cases. On big enough data sets with good random distribution, its effects are negligible.

To conclude, it can be stated that the results of machine learning techniques can certainly be improved, through data driven means. From this follows that a machine learning model, at least one like the ones used, is not able to gain all knowledge on its own and multiple angles of inspection may create preferable results. Therefore alternatives like ARM and dimensionality reduction should be considered when striving for improved precision or robustness.

6 Designing Pass Plans

Based on the results of previous chapters, the challenge of preemptively designing a pass plan can be tackled. From creating a reliable prediction model a lot can be learned about the hot rolling process, for example, how different temperatures change the variance in results or how cool down time affects the grain size. But it offers something more, the ability to see how changes in multiple inputs at the same time change results. Single input models can give insight but are not helpful in determining a pass plan in an adequate way.

A pass plan is a set of inputs for every step in successive hot rolling iterations. An example would be a very simple pass plan where only the height reduction for each step is defined, with all other inputs falling to default. From 400cm to 200cm, 100cm and 50cm. Now one can look at the resulting block after each step and see if a desired result was obtained, whether all constraints are kept and all desired qualities like grain size were achieved. This is a very limited approach since changing only the height reduction leaves out a lot of other inputs that may provide beneficial. Therefore it is better to change as many relevant inputs as possible while still being able to compute results. This way the created prediction models were all able to predict results with a diverse set of inputs. The effort to create a pass plan that satisfies requirements will now be called: calculating a pass plan.

There is a lot of constraints in calculating a pass plan, values must be realistic in the real world, financially as well as realizable. A huge steel slab might give great results, but if there is no machine which could fit it a pass plan incorporating it will be useless. Furthermore it is important that production constraints

aren't violated, rolling mills have limitations on the amount of applicable force or temperature.

So when calculating a pass plan, it is important to honor all constraints and based on those to get a satisfying result. This leads to the problem of optimizing or reaching multiple values at the same time. In the best case a function could be constructed that is fed the current materials attributes and a number of desired steps and qualities and returns a pass plan.

6.1 Satisfying Constraints, Optimizing Pass Plans

When given a good model of the hot rolling process, optimizing input parameters becomes an option. But while a model does tell what the result for any given input is, it does not tell what the best values are. Black box models will not even show how a change in any input parameter changes any variable in the result. For optimization to start, two things need to be clear. First, the goals of optimization and second, how each parameter affects the refined slab or rather how the process that is to be optimized works. The second point is the one where ARM becomes very useful.

In this case our goals will include several result qualities which all represent very different aspects. Both for fulfilling constraints as well as optimizing.

- Height, a given end parameter, it is a singular value that must be reached at the end; it is also a directly changeable input through roll distance.
- Grain size, a quality that may be reached anywhere in a range, in this case 20 µm - 40 µm
- Force (on the rolls), a constraint that may not be violated because it would break the rolling mill or but it under undesired stress.
- Energy Consumption, finally a parameter to be optimized, since lower energy consumption means lower production cost.

Therefore an optimization function has to make sure that the first 3 points are taken into account and the forth is as small as possible.

6.1.1 Evaluation Functions

To evaluate the points from the previous list it must be possible to obtain or calculate their values. But there is no given function for these, except for height which is an input setting, therefore all of them may be predicted, like grain size. 3 models are to be trained, one to predict each of the values.

Now a function must be defined that allows evaluating the results of these models. Given this is a minimization problem the function should return a lower value for better predictions. Starting height is given and therefore plays no role in the evaluation. A force too high must return infinity, since an answer that includes destroying or damaging the mill is not acceptable. A violation of grain size may either return infinity if its upkeep is of utmost importance or a penalty to the score. Since its derivation might be fixed in future steps making it an absolute might have negative impact on the entire calculation. Energy consumption may as well just be returned as is, smaller energy consumption is better. A python code example 6.1.

```
def EVAL(inputs):
force = ForceModel.predict(inputs) #1.0 for breaking
if(force == 1.0):
        return infinity

grainSize = GrainSizeModel.predict(inputs)
penalty = 1 #Penalty multiplier
if(UPPERBOUND < grainSize):
        penalty = (grainsize - UPPERBOUND) * 1000
if(LOWERBOUND > grainSize):
        penalty = (LOWERBOUND - grainsize) * 1000

energy = EnergyModel.predict(inputs) / 1000000000
return energy + penalty
```

Listing 6.1: Evaluation

This function is an attempt to represent the problem of creating a suitable pass plan by minimization. It returns infinity on a broken force constraint, which would be a violation of machine constraints, a pass that the rolling mill should not experience. This could be the attempt to decrease too steeply, or moving a slab too fast. If the reached grain size is outside of the defined boundaries, normally a grain size of 20-40 micrometer is desired, its minimal distance from the boundaries is calculated. From this a penalty modifier is created. Lastly the energy consumed is calculated, with the penalty modifier applied. This is then returned; ideally the produced slab would be within the desired grain size range and then have an energy consumption that is as small as possible. The energy consumption was shown to be of a size of around 1,000,000,000 joule, therefore it was reduced and a penalty factor of 1000 is applied.

6.2 The Minimization Problem

Minimization challenges have vastly different solutions, the staple: Linear Programming, a method for reaching the optimal output of a function, comes to mind, but does not mesh well with the non linearity of the hot rolling process (Dantzig and Thapa, 1997).

Grid Search seems like a simple yet promising method; it lends to this problem since ranges for all inputs are defined. In Grid search a range is given for every parameter and (all) combinations of parameters from these ranges are tested randomly, to search for the best combination (Bergstra and Bengio, 2012). This would certainly not be feasible on our entire dataset but ARM has shown that a way smaller number of parameters is required for accurate results, making it a possibility again.

Or a Neural Network with genetic improvement may solve this problem. Trying different combinations and learning whether a combination produces a better result than the previous one may lead to a fast solution (Liu, 2001).

Solving minimization problems is not within the scope of this thesis, but it was just shown that the problem of designing pass plans indeed leads to a minimization or that this is at least a valid attempt at solving it. Therefore SCIPY, a collection of standard scientific functions in python, will be used to attempt a solution. SCIPY contains a range of minimization and optimization functions, including ones that work with bonds, which makes it applicable for minimization on a limited dataset (Oliphant, 2007).

6.3 Detailed Process

The process of designing a pass plan based on the created and improved prediction models from previous chapters will be illustrated in this section. The goal is to minimize the function, EVAL, provided in the previous section.

First, prediction models for force, grain size and energy consumption must be constructed, this is fairly easy based on the outcome of the previous chapters. ARM was applied and prediction models were built: For grain size the previously established random forest was used:

```
RandomForestRegressor(n_estimators = 100, n_jobs = 6)
```

Now for predicting if a violation of the force constraint occurs a classifier is far more applicable. Either they were violated or not; the actual force applied does not matter in this special case.

```
RandomForestClassifier(n_estimators = 100, n_jobs = 6)
```

Again a random forest was chosen, since a random forest can be used both for regression as well as classification. Lastly, energy consumption must be predicted, a MLP regressor was chosen, as it is of no importance for this model to be white-box at the given time. Diversity seems more rewarding as in this case the MLP's precision has proven superior in test runs.

```
MLPRegressor(hidden_layer_sizes=(50,50,50), activation='relu',
             solver='adam', shuffle=True, warm_start=False)
```

The most notable hyper parameter here is hidden layer sizes=(50,50,50), which means that three layers of each 50 perceptrons were used. Now when training the models the following precisions are reached:

Target	R^2
Grain size	0.9844
Energy consumption	0.9815
Force constraint	0.9998

Table 6.1: Results of training models on arm reduced data.

From these results it can be deduced that the given minimization function will be highly accurate. Now, to test it, values are defined that should be probed on. *HeightDelta*, *Temperature*, *In Time*, *RunTime* and *CoolDownTime* were chosen based on the results about importance ranking from dimensionality reduction. *InTime* + *RunTime* is the *OutTime* and also allows for calculating *Speed*, which is *OutTime* - *InTime* with regards to the slabs *Length*. For all other values standard values or means were used. In a first trial it will be attempted to find the optimal settings for a single pass. Starting from a grain size of 140 µm, a grain size between 100 µm and 80 µm shall be reached.

Bounds must be defined for each of the parameters and, within these, a solution is searched for. These boundaries are based of the highest and lowest values encountered in the given data samples, with obvious outliers removed, because these are supposed to be all possible values.

Item	Height red.	Temperature	Insert time	Run time	Cool down
Range	(10.0,100.0)	(1000.0,1100.0)	(1.0,100.0)	(0.05,4.0)	(0.1,40.0)

Table 6.2: Parameter ranges for minimization.

Having defined the function to minimize as well as the applicable bonds minimizer must be selected to perform the actual minimization step. For this TNC from the previously mentioned SCIPY.Optimize package is chosen, it which performs a truncated Newton algorithm, which takes parameters defined by bounds and tries to minimize a matching function. The algorithm is also know as Newton Conjugate Gradient, as it uses gradient information (Nocedal and Wright, 2006).

The C in TNC indicates that the function is a wrapper for an implementation in C, which has better performance. Of course, any other minimization method may be freely used; they are not compared here.

```
bound = ((10,100), (1000,1100), (1,100), (0.05,4), (0.1,40))
optimum = minimize(EVAL, method='TNC', bounds=bound).x
```

After 3 iterations and a total of 94 function evaluations *optimum* contains the array [18.49, 1050, 1, 19.93, 10], which are the corresponding values for the parameters given above. When running these values in each of the three predictors the following results are receive:

1. A force of 1.0. This means all machine constraints where kept.

2. A grain size of 99.454 µm. From that it is clearly visible that the algorithm had no problem finding input that brings the grain size in the desired range but kept trying to optimize energy consumption, which is probably related to as little a reduction in grain size as possible.

3. An energy consumption of 199,283,632 joule or 199 MJ. This is a really small value, being just a fifth of the previously mentioned average of 1,000,000,000 joule.

This is a great result. It proves that it is possible to use a minimization algorithm to design a pass plan, and even optimize it with regard to energy consumption. Alterations to this can be used to optimize for different values or goals, like a set height reduction. Several minimizations back to back can be used to create an entire pass plan by calculating the optimum for each pass and then computing the next pass based on the results of the previous pass. Minimizing a function over all 76 variables would have required vastly more resources, but thanks to ARM and dimensionality reduction this was not needed!

7 Conclusion and Outlook

The main focus of this thesis lied on the metal working process of hot rolling and on how advances in machine learning can be used to understand it. This combines an interesting and complex problem, namely how the hot rolling process is definable through machine learning, with a challenge, the difficulty to determine the non-linear functions governing the hot rolling process. The problem is funded in real world mechanics; although the thesis uses just a simulated version of those mechanics, this makes it not only applicable but shows that it has a practicality and is not only of theoretical value. Machine learning, on the other hand, is a vastly growing field with a multitude of applications. From this stemmed the motivation to find a suitable solution.

To solve the given problem, first the theoretical backgrounds and concepts of ARM, dimensionality reduction and machine learning were presented to lay the foundation for further results. Then these results were detailed, which led to three steps being proposed to solve the problem:

- Step 1: Develop rules which represent the causalities in the hot rolling process through association rule mining. And use these rules to filter the data to make it usable for step two.
- Step 2: Reduce the dimensionality of input in order to make it possible to visualize the process and weigh its parameters.
- Step 3: Train a regression model to be able to predict the result of a pass. This allows to model the hot rolling process at very low cost, measured in time and effort.

The evaluations made in this paper have shown that, especially the first step, ARM, is a very efficient way of filtering data without losing much of the knowledge contained in the data at all. It allowed for nearly equally performing regression models in terms of precision and superior performing models in terms of speed and memory usage. Dimensionality reduction proved to be a good way of visualization and the weighting it made possible proved to be correct. Its applications for the regression model increased stability by reducing the probability of bad random selection. Regression models, especially random forest, have shown to be a great way for understanding the hot rolling process through machine learning tools. Their predictions turned out to be very precise and it was possible to create pass plans based on these prediction models.

Finally these three steps proved fruitful in Section 5.4, Improving Prediction. Here the thesis culminated when all steps were combined to create an improved model of the hot rolling process, which has proven superior to a raw model. Thus, it could be shown that ARM and dimensionality reduction lead to a faster and more stable prediction model, based on the random forest regressor.

The final Chapter, Designing Pass Plans, proved that the developed prediction model can be used to create pass plans according to requirements. This is a step in the direction of completely autonomous rolling mills, which adapt their settings in real time based on machine learning models.

From this result an array of prospective, promising future work can be derived:

- Re-evaluating the produced results against other tools for predicting pass plans, like RoCat of the IBF (Paesler and Hückl, 2012). This could lead to a mixed model with even better precision.
- Research into how ARM can be used on other industrial or metal working tasks, like casting, curing or bending.
- Extracting and detailing the underlying functions of the hot rolling process from the random forest model, for example, through combination of tree paths.

In summary, with an application of said techniques (ARM, dimensionality reduction and regression), very accurate prediction models of hot rolling can be made, leading to improvements in the process. Thus the demand for a more efficient and ultimately more cost-saving production method in the steel industry can be met, contributing to the growth of the industry and global economy.

Bibliography

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993. ISSN 0163-5808. doi: 10.1145/170036.170072.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012. ISSN 1532-4435.

John H. Beynon and C. Michael Sellars. Modelling microstructure and its effects during multipass hot rolling. *ISIJ International*, 32(3):359–367, 1992. doi: 10.2355/isijinternational.32.359.

R. Brause, T. Langsdorf, and M. Hepp. Neural data mining for credit card fraud detection. In *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '99, pages 103–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0456-6.

Chenyi Chen, Ari Seff, Alain L. Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. *CoRR*, abs/1505.00256, 2015. URL `http://arxiv.org/abs/1505.00256`.

George B. Dantzig and Mukund N. Thapa. *Linear Programming 1: Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. ISBN 0-387-94833-3.

James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

E. Paul (Ernest Paul) DeGarmo, J. Temple Black, and Ronald A Kohser. *Materials and processes in manufacturing*. New York ; Chichester : Wiley, 9th ed.,international ed edition, 2003. ISBN 0471033065. Previous ed.: London : Prentice-Hall International, 1997.

Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

F. Feldmann. Adaption von walzmodellen. *Aluminium*, 70(3–4):229–234, 1994.

M. J. Grimble and G. Hearns. *Advanced Control for Hot Rolling Mills*, pages 135–169. Springer London, London, 1999. ISBN 978-1-4471-0853-5. doi: 10.1007/978-1-4471-0853-5_5.

C.P Hong and J.J Park. Design of pass schedule for austenite grain refinement in plate rolling of a plain carbon steel. *Journal of Materials Processing Technology*, 143–144:758 – 763, 2003.

Jan Kusiak and Roman Kuziak. Modelling of microstructure and mechanical properties of steel using the artificial neural network. *Journal of Materials Processing Technology*, 127(1):115 – 121, 2002.

Duk Man Lee and S.G Choi. Application of on-line adaptable neural network for the rolling force set-up of a plate mill. *Engineering Applications of Artificial Intelligence*, 17(5):557 – 565, 2004.

Dukman Lee and Yongsug Lee. Application of neural-network for improving accuracy of roll-force model in hot-rolling mill. *Control Engineering Practice*, 10 (4):473 – 478, 2002.

G.P. Liu. *Nonlinear Identification and Control – A Neural Network Approach*. Springer Science and Business, 2001.

Tom M. Mitchell. *Machine Learning*. WCB McGraw-Hill, 1997.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

Travis E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007. doi: 10.1109/MCSE.2007.58.

H. Dyja P. Korczak and E. Łabuda. Using neural network models for predicting mechanical properties after hot plate rolling processes. *Journal of Materials Processing Technology*, 80–81:481–486, 1998.

P. Paesler and J. Hückl. Newsletter umformtechnik. Technical Report 31, ibf - Institut für Bildsame Formgebung, Institut für Bildsame Formgebung, Intzestraße 10, 52056 Aachen, 12 2012. Page 3.

Jong-Jin Park. Prediction of the flow stress and grain size of steel during thick-plate rolling. *Journal of Materials Processing Technology*, 113(1–3):581 – 586, 2001.

K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *CoRR*, abs/1201.0490, 2012. URL `http://arxiv.org/abs/1201.0490`.

K.P Rao, Y.K.D.V Prasad, and E.B Hawbolt. Study of fractional softening in multi-stage hot deformation. *Journal of Materials Processing Technology*, 77 (1–3):166–174, 1998.

William L Roberts. *Hot rolling of steel*. New York : M. Dekker, 1983. ISBN 0824713451. Companion vol. to: Cold rolling of steel. c1978.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Alex Smola and S.V.N. Vishwanathan. *Introduction to Machine Learning*. Cambridge University Press, 252 edition, 2010. ISBN 0521825830.

Lixin Tang, Jiyin Liu, Aiying Rong, and Zihou Yang. A multiple traveling salesman problem model for hot rolling scheduling in shanghai baoshan iron steel complex. *European Journal of Operational Research*, 124(2):267–282, 2000.

Tobias Teich, Falko Roessler, Daniel Kretz, and Susan Franke. Design of a prototype neural network for smart homes and energy efficiency. *Procedia Engineering*, 69:603–608, 2014.

Ah Chung Tsoi. Application of neural network methodology to the modelling of the yield strength in a steel rolling plate mill. In *NIPS*, pages 698–705, 1991.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *The Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. doi: 10.1109/MCSE.2011.37.

Marc Ingo Wolter, Anke Mönnig, Markus Hummel, Christian Schneemann, Enzo Weber, Gerd Zika, Robert Helmrich, Tobias Maier, and Caroline Neuber-Pohl. Industrie 4.0 und die folgen für arbeitsmarkt und wirtschaft: Szenariorechnungen im rahmen der bibb-iab-qualifikations- und berufsfeldprojektionen. IAB-Forschungsbericht 201508, Institut für Arbeitsmarkt- und Berufsforschung (IAB), Nürnberg [Institute for Employment Research, Nuremberg, Germany], 2015.

Appendix

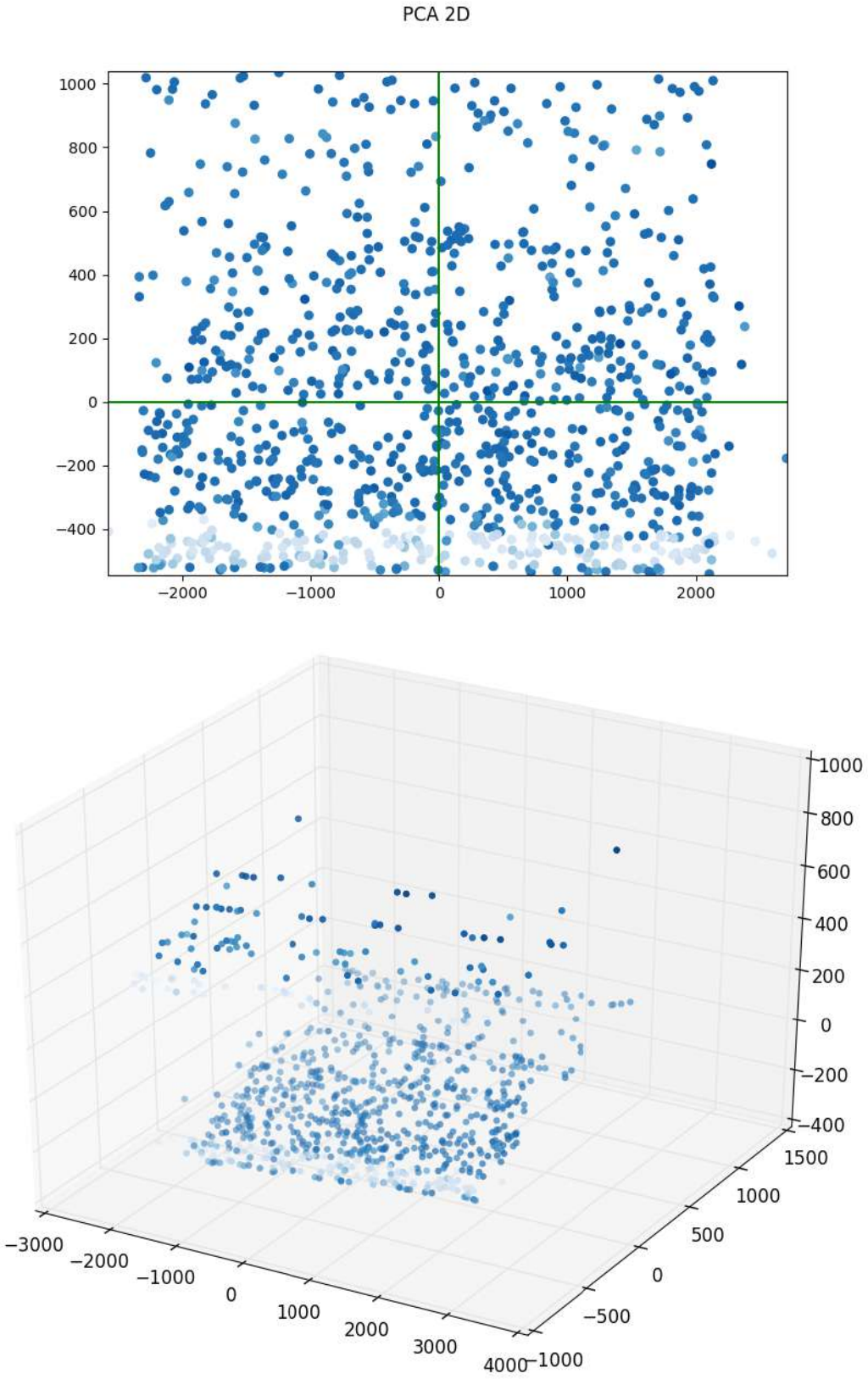


Figure A.1: Example of Dimensionality Reduction using PCA,
2D and 3D, 1000 samples.
Brightness shows change in grain size (Δ_G); clustering is clearly visible

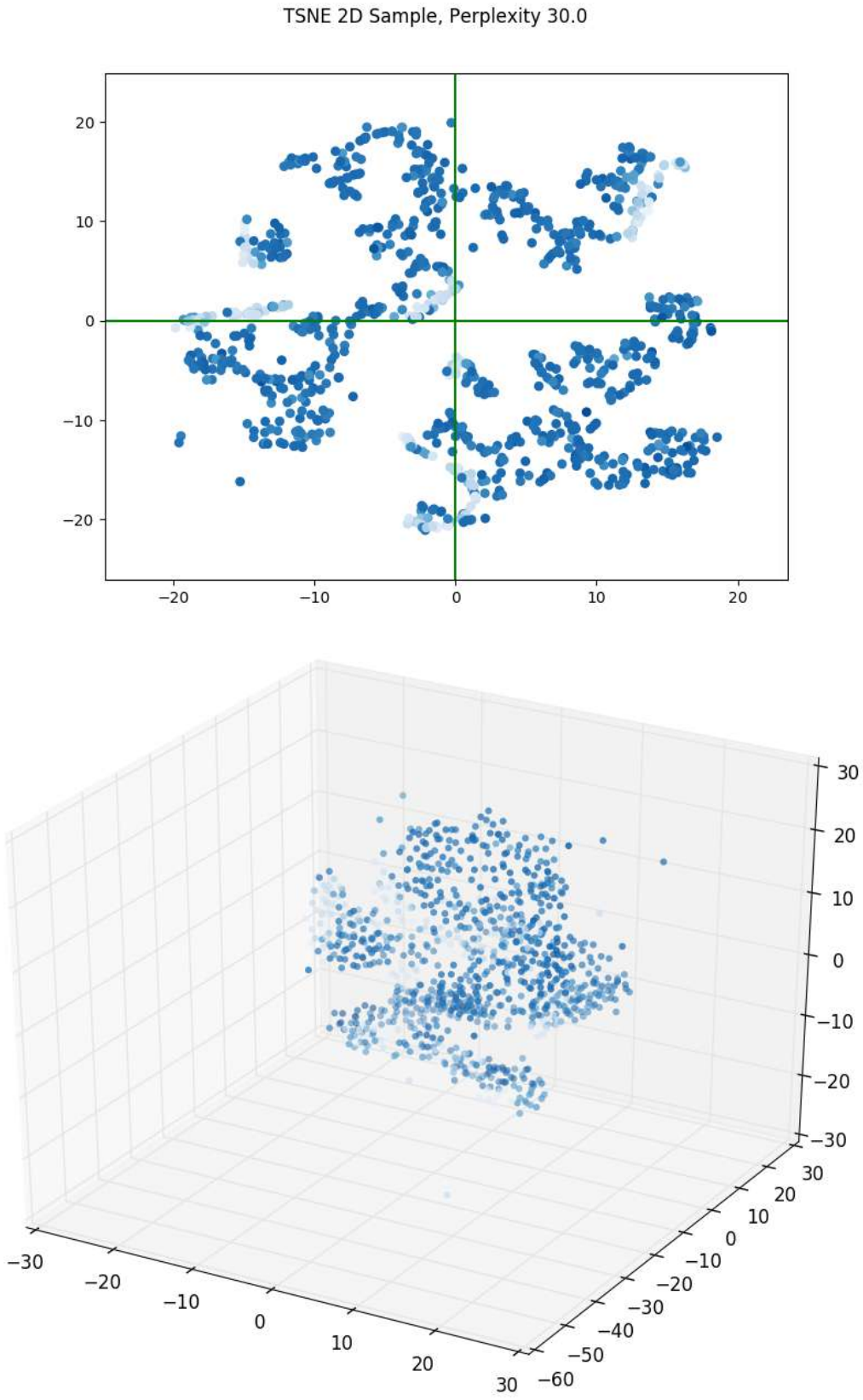


Figure A.2: Example of Dimensionality Reduction using t-SNE,
2D and 3D, 1000 samples.
Brightness shows change in grain size (Δ_G); clustering is clearly visible

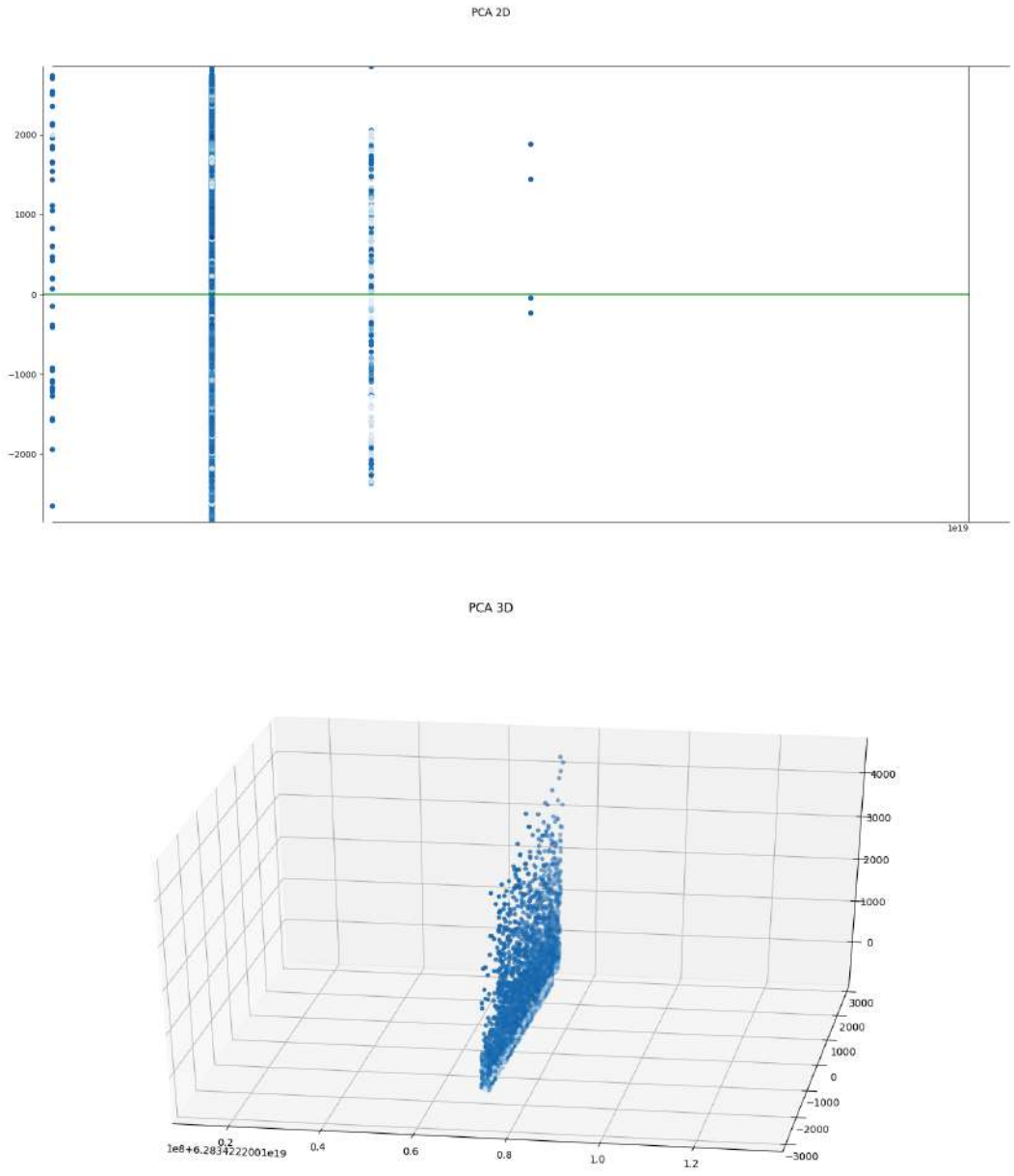


Figure A.3: PCA, 2D and 3D, 1000 samples at 79 dimensions.
Extreme data points are corrupting the visualization of PCA.

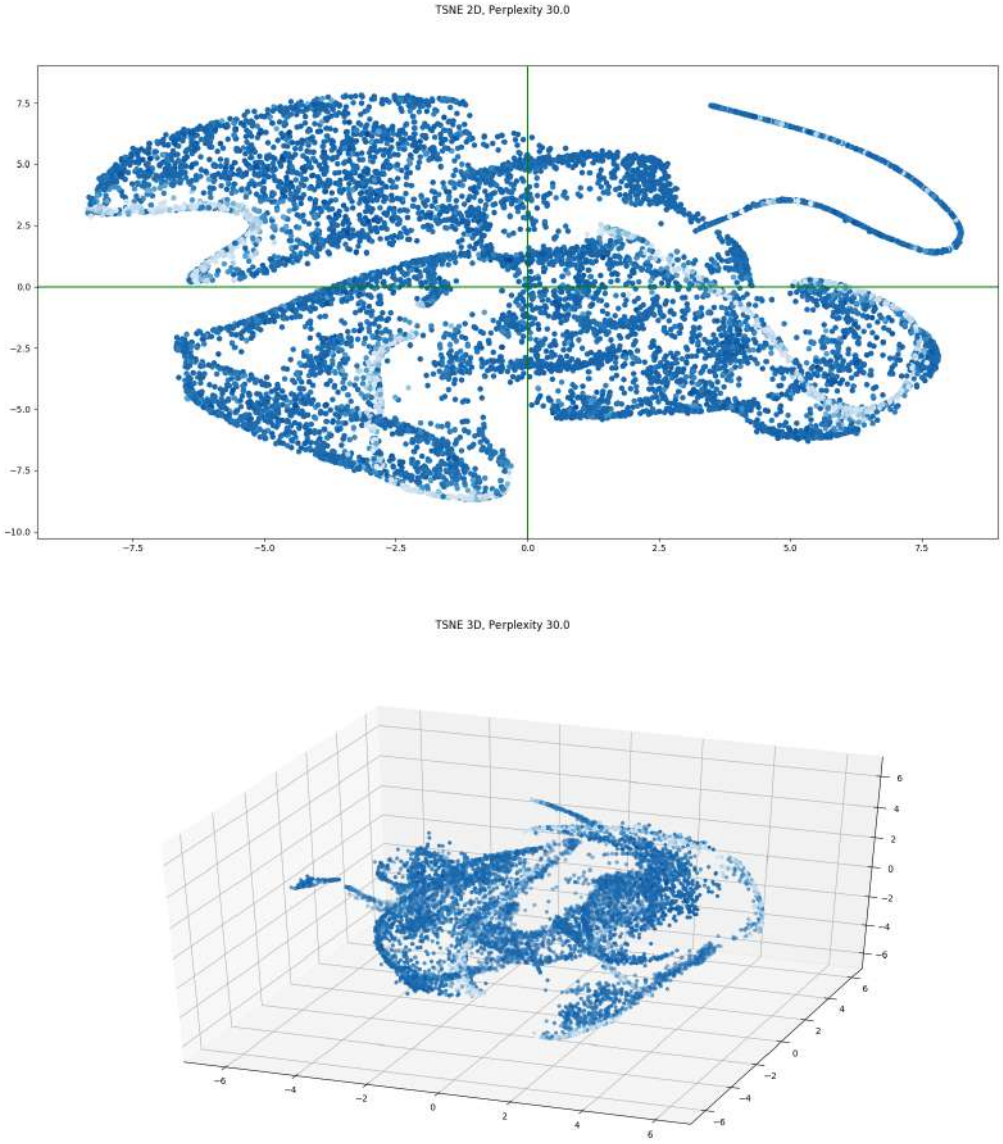


Figure A.4: t-SNE, 2D and 3D, 1000 samples at 79 dimensions.
Extreme data points do not disturb the visualization.

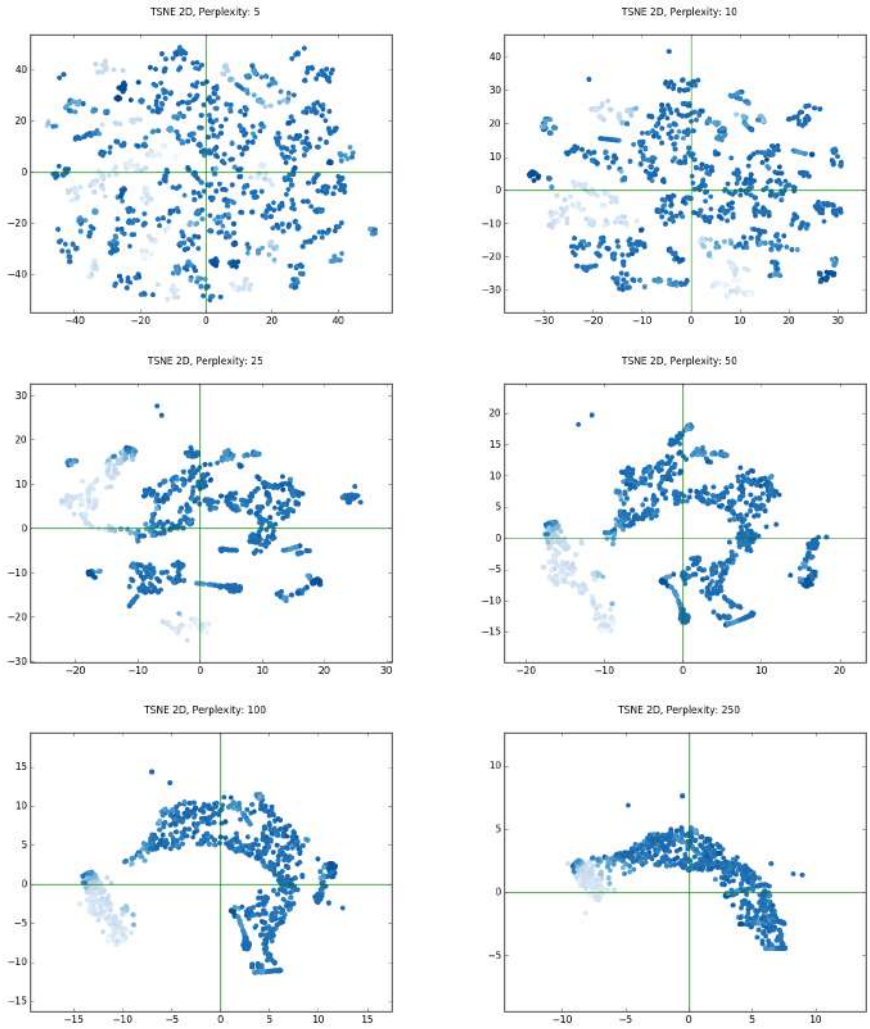


Figure A.5: t-SNE, 2D, 10.000 samples at 14 dimensions trough ARM. Different perplexities [5,10,25,50,100,250]

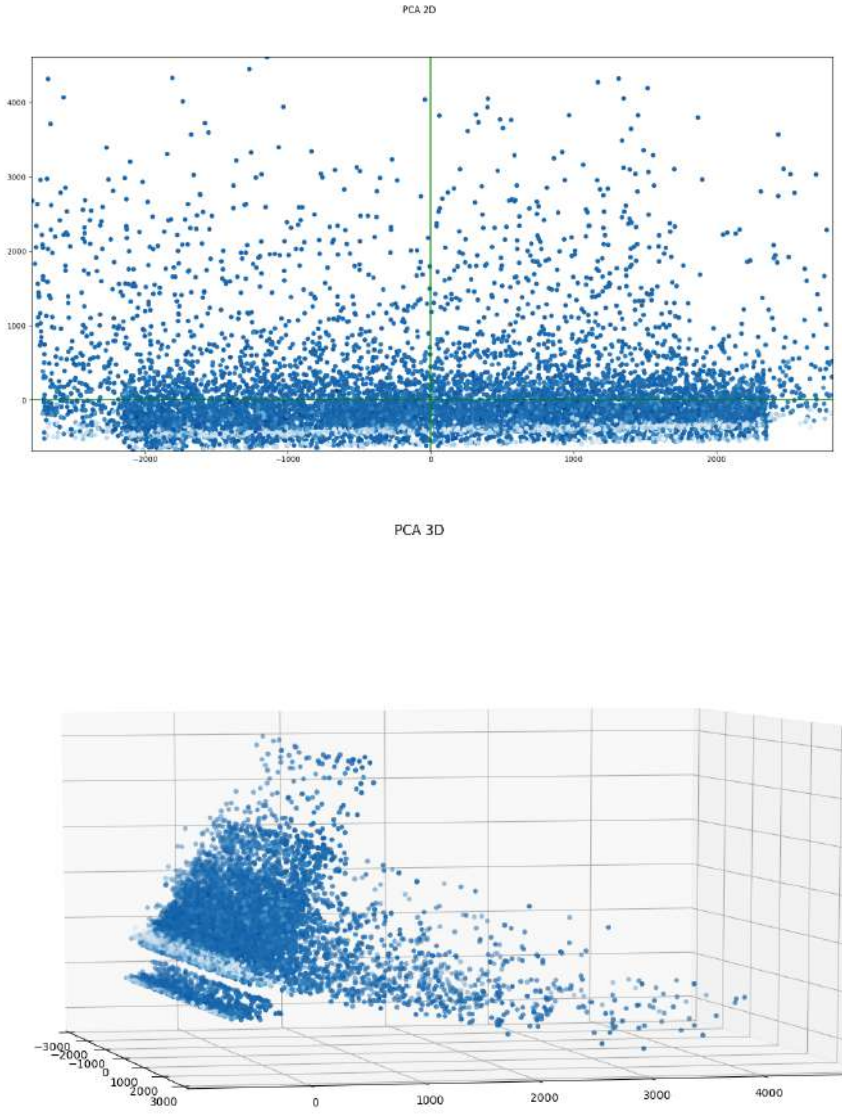


Figure A.6: PCA, 2D and 3D, 1.6 million samples at 77 dimensions.
2 dimensions have been filtered out.

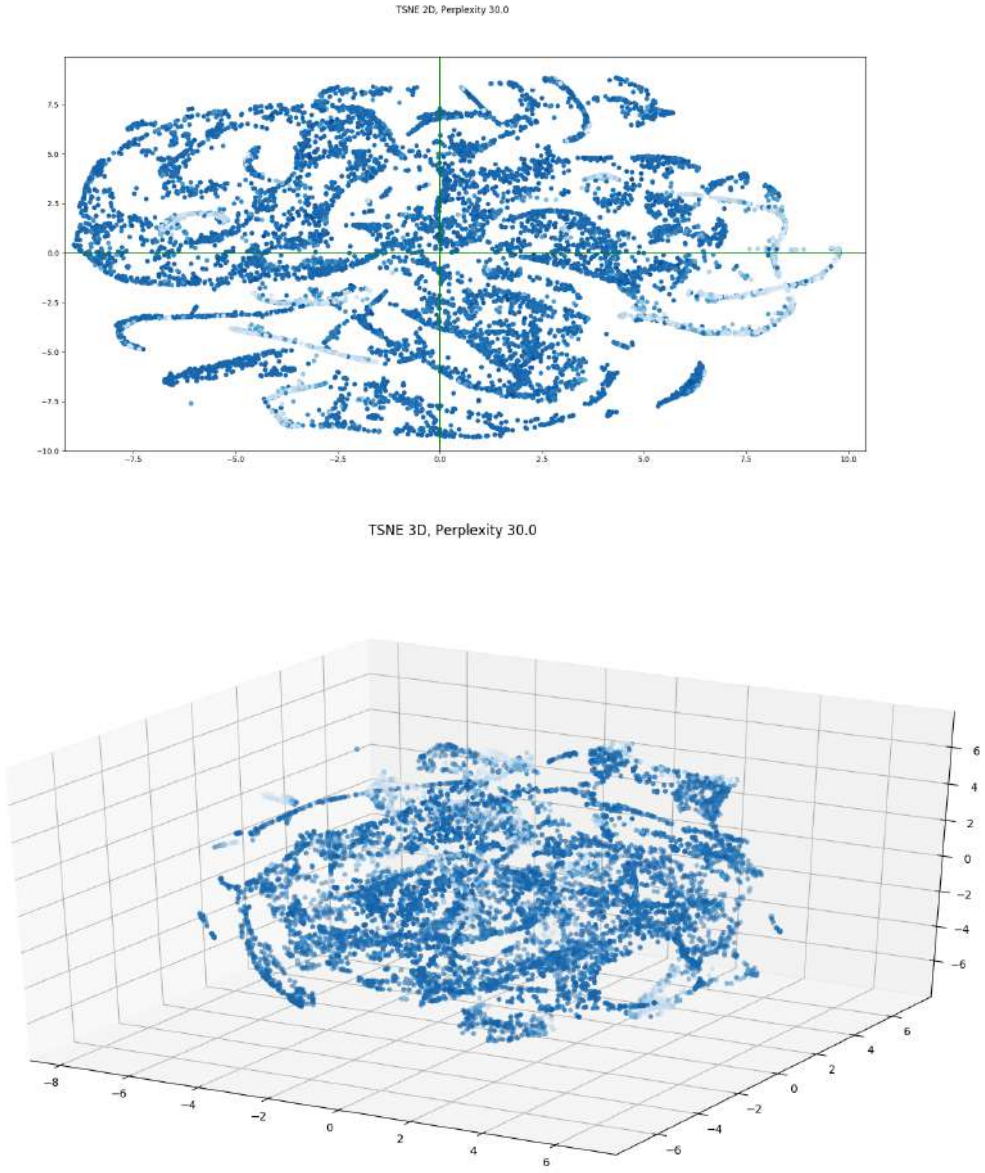
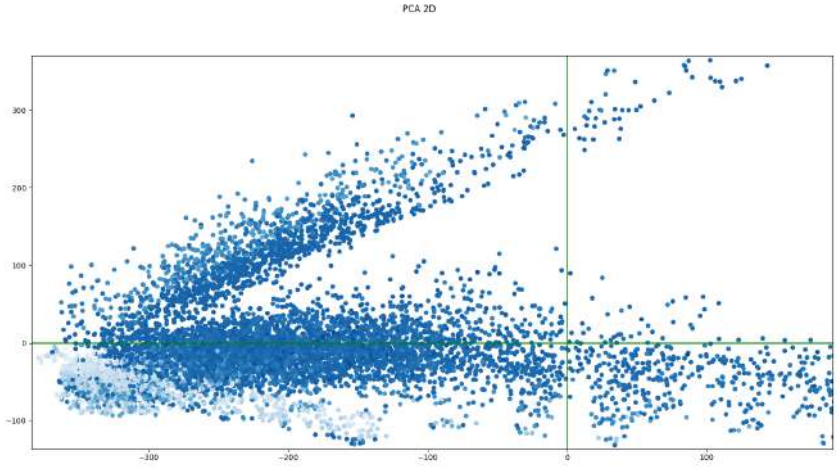


Figure A.7: t-SNE, 2D and 3D, 10.000 samples at 77 dimensions.
2 dimensions have been filtered out.



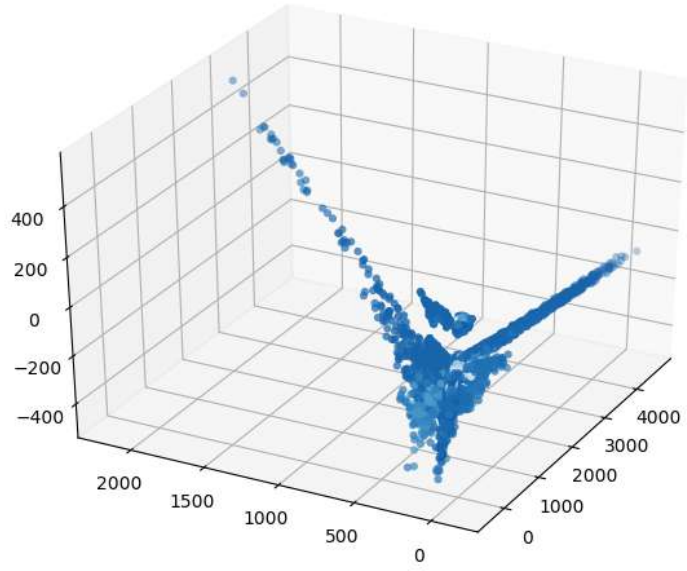


Figure A.8: PCA/t-SNE, 2D and 3D, 1.6 million samples. Dimensions have been filtered through ARM, down to 14

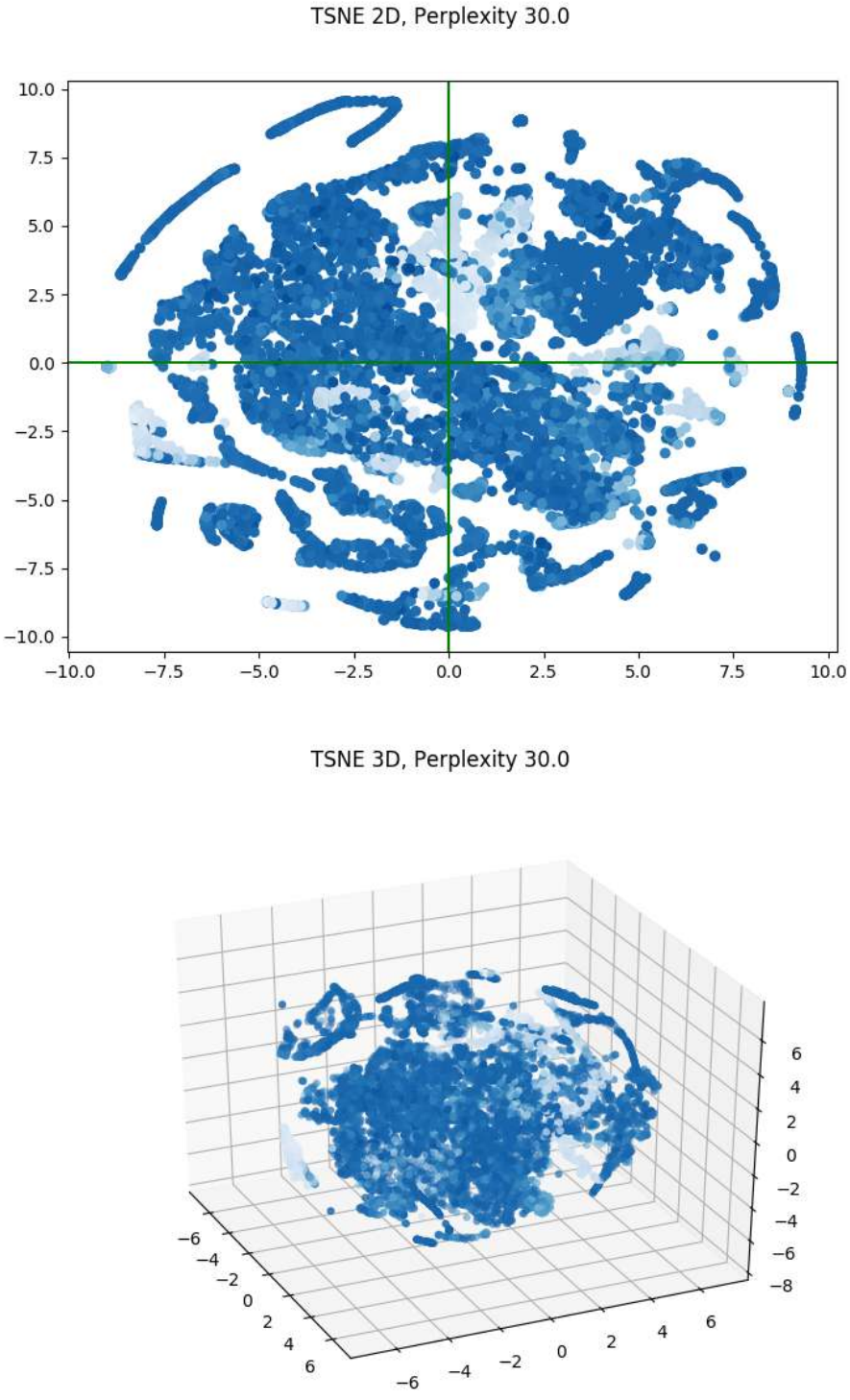


Figure A.9: t-SNE, 2D and 3D, 10.000 samples.
Dimensions have been filtered through ARM, down to 14