Using Machine Learning for Predicting Interchanges

Leveraging Machine Learning Methods for Finding Equivalent Products Based on Competitor Products

Master's thesis in Engineering Mathematics and Computational Science

Jesper Wiström

DEPARTMENT OF MATHEMATICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Using Machine Learning for Predicting Interchanges

Leveraging Machine Learning Methods for Finding Equivalent Products Based on Competitor Products

Jesper Wiström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematics
Chalmers University of Technology
Gothenburg, Sweden 2025

Using Machine Learning for Predicting Interchanges
Leveraging Machine Learning Methods for Finding Equivalent Products Based on Competitor Products
Jesper Wiström

Supervisor: Hugo Carlen, Company Supervisor
Supervisor: Johan Jonasson, Department of Mathematics
Examiner: Johan Jonasson, Department of Mathematics

Master's Thesis 2025
Department of Mathematics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A picture of a bearing.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Using Machine Learning for Predicting Interchanges
Leveraging Machine Learning Methods for Finding Equivalent Products Based on Competitor Products
JESPER WISTRÖM
Department of Mathematics
Chalmers University of Technology

Abstract

In the bearing industry, identifying equivalent products across different manufacturers is a complex and time-consuming task, traditionally reliant on expert knowledge and manual cross-referencing. This thesis explores the feasibility of automating this process using machine learning techniques. The core challenge lies in predicting an equivalent bearing designation from a competitor's product code, which typically lacks structured attribute data and consists of a single, information-dense string. To address this, a comprehensive machine learning pipeline was developed, encompassing feature extraction, encoding, model training, and decoding. The problem was framed as a multi-label classification task, where suffixes representing product features were predicted and then used to reconstruct a valid target designation.
Several machine learning models were evaluated, including Random Forests, Support Vector Machines, k-Nearest Neighbors, and Neural Networks, using both problem transformation and method adaptation strategies. Random Forests with label powerset transformation emerged as the most effective approach, offering a strong balance between accuracy and computational efficiency. The pipeline was further optimized through domain-specific feature engineering, such as extracting product size and type indicators, which significantly improved model performance.
Despite inherent limitations in the data—such as inconsistent labeling and sparse representation of certain suffixes—the results demonstrate that machine learning can reliably assist in generating bearing interchanges. This approach has the potential to significantly reduce manual effort and scale the interchange process, making it a valuable tool for industry practitioners.

Keywords: Bearing Interchange, Machine Learning, Multi-label Classification, Feature Extraction, Random Forest, Product Designation, Suffix Prediction, Cross-reference Automation, Domain-specific Encoding, Industrial AI Applications

Contents

List of Figures

List of Tables

1
Introduction

This report investigates the application of machine learning methods in the bearing industry. A common problem is replacing a bearing from one manufacturer with a bearing from another manufacturer. This task requires deep industry knowledge, which few people possess, and is extremely time-consuming. To aid in this task and reduce the time spent, machine learning will be applied.
The goal is to train a model on references from one manufacturer to another. The aim is that, given a product designation from the first manufacturer, the model can predict the equivalent bearing from the other manufacturer. The difficulty lies in the lack of attributes in this data; it consists of only one string that needs to be converted to another string. The core of the problem is to extract enough features from the product designation and train the model on these relationships.

1.1 Background

In the bearing industry, product designations (or product names) carry extensive information about the dimensions, type, and features of the product. Some parts of the designation follow an ISO standard, while other parts, which describe the product's features, are created by each manufacturer. The naming system uses many codes and abbreviations, and the designation system provides the information needed to decode these codes into meaningful descriptions. This system allows you to compare products between brands and identify an equivalent product from another manufacturer. However, this requires knowledge of the designation systems of different manufacturers and an understanding of their relationships.
To assist users who are not familiar with the different designation systems, interchange tools are available. These tools allow you to enter the designation from one manufacturer and then show the equivalent product designation from another manufacturer. Such a reference between a product from one manufacturer and an equivalent product from another manufacturer is called an interchange.
The project originated from such an interchange tool. This tool allows distributors to find an interchange from a competitor's product designation. With approximately 10,000 searches each day, it is a valuable selling tool for the company and an important tool for distributors in their daily work.
The application is based on a large, manually created database of interchanges from competitor products. This database contains information spanning most product categories and many brands, with the majority of the data focused on the most important competitors. The company now recognizes the need to expand this database

to cover more of the assortment and to find references for frequently searched product designations that are not currently included.

The current process is manual and extremely time-consuming, taking 10-20 minutes to identify each individual interchange. First, you need to find the competitor product and understand its type and features. Then, you need to determine if you have a similar offering and, if so, identify the exact product. This requires extensive knowledge of the bearing domain and the products. To free up time and scale the service, the company now aims to leverage machine learning methods to create new interchanges.

1.2 Previous Attempts

Over the years, there have been several attempts to solve this problem, highlighting a clear business need for a faster and more efficient way to identify interchanges. Ideally, an AI model would be created that requires less expertise and can be used on a larger scale. Notably, there have been two significant attempts using different methods: one based on traditional machine learning and another utilizing Large Language Models to make these predictions. The main aspects of this work and their outcomes will be described below.

1.2.1 Internal Project

This internal project, driven by the company's research and development branch, was undertaken by an experienced AI & ML expert. The goal was to find a way to match competitor products to the company's products by exploring various machine learning methods to determine what works best and what is possible.

The project concluded that this task is impossible. It was determined that converting product designations requires rich attribute data from both manufacturers' products. Obtaining such data is challenging for the company's own products and even more difficult for competitor products. The conclusion was that a model needs this data to learn the relationships, and without it, the task is not feasible.

Another suggested solution was to manually build a decision tree along with a lookup table to convert the affixes from the competitor's products to the company's equivalent affixes and create rules based on this. However, this approach would result in an extremely large and complex decision tree, as suffixes have different meanings for different product types. Building such a tree would be extremely time-consuming, difficult, and expensive, making it unfeasible, even though it was considered the best possible approach.

Overall, the project's conclusion suggests that the task we are attempting in this master thesis is impossible or at least infeasible. However, the hope is that by transforming the problem in the right way and performing effective feature extraction, success can be achieved.

1.2.2 Previous Master Thesis

There has also been a previous attempt to complete a master thesis on this topic. The premise of the thesis was the same as this one; however, the students who took on this challenge came from a very different educational background.

Initially, they conducted extensive literature reviews to find if anyone had tried to solve a similar problem before. The only slightly similar case they found was the conversion of e-commerce descriptions, which are much longer and often contain product attributes. The next step was exploring methods such as mapping the products into a vector space. However, an encoder was never successfully developed and would need to be hand-created for each specific competitor.

Due to these challenges, it was decided to utilize Large Language Models (LLMs) to solve the task of identifying equivalent products. Initially, standard models were employed with little success. Consequently, they tried to fine-tune a model based on existing interchanges. This approach yielded better results but was very expensive, with costs both for training and for running the model when needed.

Although the fine-tuned model produced better results, they were still not good enough. An accuracy of 70% was achieved, which is decent, but this was on a filtered dataset that removed many OEM interchanges. Such interchanges are from a random number to an industrial product where there is no logical relationship or pattern, so removing them gives much higher accuracy. Another challenge was the lack of explainability; it was impossible to explain why a particular suggestion was made.

One major issue with this method was that the model always provided an answer, even if it had no idea about the relationship. This made it difficult to filter out good predictions from bad ones, making validation time-consuming. Another significant issue was that the predictions had no connection to actual product designations, so a prediction could look correct only to discover that the product doesn't exist and is a series that the company does not make. The final issue was the lack of consistency and explainability. Running the model several times on the same product could yield different results each time, making it non-deterministic. This is because of the nature of LLMs, which can produce different output given the same input. The predictions were also completely unexplainable, and you could not follow the reasoning. This inconsistency caused low trust in the model and led to doubts about all its predictions.

The conclusion from this project was that, at the time of evaluation, LLMs were not good enough and not suitable for the task. It could be that in the future, when they are easier to train and adapt to domain-specific areas and better at learning patterns, this methodology might be feasible. However, currently, this is not the case. A main takeaway is the importance of consistency and reliability, as trust in the model is crucial for its actual implementation. For these reasons, the model was never implemented or utilized again.

1.3 Aim

The aim of the project is to identify and leverage an appropriate machine learning method that can be trained on the existing database of interchanges. The trained model should be able to identify a correct interchange for a given competitor's product designation.

1.4 Limitations

Due to the large scale, varying designation systems, and number of interchanges in the database, the project will be limited to the most important competitor.
However, the aim is to develop a method that can be replicated for other manufacturers, with minor differences in data preprocessing.

1.5 Problem Specification

The input to the model will consist of only a single data point: the competitor's product designation. Because of this, the project will need to be split into four dependent parts: feature extraction, encoding, model fitting, and decoding.
The main difficulties in this project will be feature splitting, encoding, and decoding. With perfect data, fitting a model would be relatively easy. However, for a machine learning model to be effective, it needs features from which it can learn patterns. Therefore, feature extraction will be a core part of the project. If you were to split the designations by the delimiters you would get features that contain different types of information with different designations as sometimes suffixes come in different positions. This happens when a product is of the standard specification of an affix group, and therefore does not need to be specified. This makes it difficult to split the affixes into good features. To enable the model to understand the features and bypass difficulties, encoding will be central. If the affixes are encoded during training and applying the model, they need to be reinstated during the decoding step to produce a valid product designation. The difficulty here lies in making sure the affixes appear in the right order, which can be hard to achieve if the permutation of them has been removed in the encoding. Consequently, the core challenges will be those occurring before and after the actual machine learning model, as is often the case when applying such methods to domain-specific problems rather than theoretical ones.

1.6 Methodology

First, a broad analysis will be conducted to identify potential machine learning models and methods to apply.
Methods for feature extraction will be investigated first to understand how possible inputs could look for the models that will be applied later. This will be the most critical part of the project since, if the features are well extracted, most models

would likely perform well. It will become more a question of what feature extraction limitations and trade-offs can be made and later finding a machine learning model that supports these limitations.
After identifying several different methods with various trade-offs, corresponding models will be investigated and evaluated for feasibility. They will all have some requirements for success that need to be verified if they can be fulfilled.
After this study, a method will be selected, and a period of implementation will start. The first step will be to implement the feature extraction according to the selected method, as well as the encoding and decoding, to and from the model. The selected model will then be trained and evaluated on a training/test split of the data.
Evaluation is more complex. While a simple accuracy or F1 score provides a score from one perspective, it is more subjective from a business perspective. This is because you want to make references that have business relevance, which might often be to a different product than the current reference due to reasons such as availability, profitability, or strategy. For simplicity, a traditional score will be the primary metric, applied to a validation set that the model has not been trained on.
Finally, the whole process will be refined with modifications to the feature extraction and model to achieve better results. The cases where the model fails will be studied to identify the root cause, which will be mitigated if possible.

2
Theory

2.1 Designation system

In the bearing industry you have many domain specific terms. To describe the products and be specific in our language, some of these need to be understood.
The name of a product is known as the product *Designation*. This is more than just a name, it contains a lot of dense information about the dimensions and features of the bearing. The basics of this nomenclature is referred to as the *Basic Designation System*. [2]

2.1.1 Basic Designation System

The basic bearing designation system describes how to interpret the *Main Designation*, this is the part of the designation describing the product type and the main dimensions of the product. In a standard bearing the principal dimensions are the inner diameter (also referred to as bore diameter), the outer diameter, and the width. This naming convention comes from an ISO standard which is commonly used for bearings, in this standard it is clearly defined how to name products. This also ensures that if you buy a product with the same main designation, the principal dimensions are ensured to be the same. This provides a high degree of interchangeability of bearings for many of the simple use cases.
According to the standard the first number describes the *product type* or *product family*. This is the type of bearing which could be one of many different options, each type has specific use cases and features. Some accommodate radial load only, while others support axial load, some types even support loads in both directions. The next two numbers describe the width and outer diameter. These are not specified directly, you instead have different *dimensional series* of products where the ratio between the width and outer diameter is described relative to the bore diameter.
You often talk about series such as the *222* series of bearings which are *spherical roller bearings*, indicated by the first number, and of a specific outer diameter and width ratio. This bearing series is then available in many different *sizes*, here size refers to the inner diameter (commonly referred to as the bore diameter). The bore diameter is specified by the last 2 numbers of the main designation. If you multiply this number by 5 you get the bore diameter in millimetres.
Having a system like this allows you to communicate a lot of information in a very short product name. It does however require a lot of knowledge from the customer, which is not easily acquired. What also complicates things are the many exceptions, which deviate from this standard.

For example, the product type *Deep Groove Ball Bearing* is represented by the number 6, however in one specific dimension series you instead write 16. Many of the deep groove ball bearing dimension series are so common that you exclude numbers from the dimension series to make the designation shorter. For example, the common designation 6205 should really be called 60205 but the first 0 is omitted. This is typically the same between manufacturers, but not all opt to do this omission, while other manufacturers choose to do further omissions.
There are also product families where instead of being represented by a number it is represented by one or more letters. This is typical for the *Cylindrical Roller Bearings* where for example "N", "NU", or "NJ" is used depending on the design. [2]

2.1.2 Affixes

A *suffix* consists of blocks of letters and numbers that comes after the *main designation*. Each product family has a different designation system where the suffixes and their meaning are described. The suffixes provide information about the features of the product, a certain type in a certain size has many different variants distinguished by the suffixes. Sometimes a product has prefixes, this is similar to a suffix but before the main designation. Such prefixes could indicate the product type but also attributes of the product. It is therefore similar to a suffix in many ways, but not used as often. Suffixes and prefixes can collectively be called *affixes*.
The affixes do not follow any form of standard, and different manufacturers use different affixes to describe the same thing. For example, sealed on both sides can be described as 2HRS by one manufacturer and by 2RSH by another. To replace a product from one manufacturer with a bearing from another you therefore need to understand what the suffixes mean, and then also what the other manufacturer calls the same thing.
Suffixes describe features such as sealing, clearance, cage variants, etc. Many times, these features are vital to the application and cannot be exchanged for another variant. If possible, you always opt for standard variants as they are cheaper, therefore if you have purchased a bearing with certain features these must be matched when finding an equivalent product. [2]

2.2 Evaluation Metric

The accuracy score is a common metric within machine learning topics. This is a simplistic but crude measurement of classification accuracy. It is defined according to the following:

$$\frac{\text{No. Correct Classifications}}{\text{No. Correct Classifications} + \text{No. Incorrect Classifications}} \tag{2.1}$$

This formula shows that the output is simply the percentage of correct classifications from the model. In a multilabel classification model the prediction needs to predict all the same labels to be considered a correct classification, if one label is missing, incorrect, or additionally included then the classification will be considered incorrect.

2.3 Classification

Classification is the task of learning relationships to classify an instance based on its input data. There are several forms of classification which could be relevant such as binary classification, multi-class classification or multi-label classification.
You want to, from a set of data, learn to predict one or more labels λ from a set of disjoint (unique) labels L. We define M as the number of labels in L. [7]

2.3.1 Single label classification

In binary classification you want to classify an *instance* into one of two possible classes, in other words, $M = 2$ and you want to assign a single label λ to each instance.
Similarly, in multi-class classification you want to assign each instance to a single class out of multiple options of classes. This means that in multi-class classification $M > 2$ while you still aim to predict a single λ for each instance.
The difference between binary classification and multi-class classification is the number of possible classes. In binary it is one of 2 classes, while in multi-class its one out of many possible classes. Multi-class classification can be achieved by problem transformation into several binary classification problems. This means that a binary classifier can be used for multi-class classification tasks.
There are many ways to do this, but the most common and simple one is *one-vs-rest*. Here you turn several classes into binary decisions which are taken in series, each of the binary decisions are whether the instance belongs to the class or not. If you have 3 classes A, B, C you need 2 binary classifiers, one for A or not A, one for B or not B. If both classifier predictions are negative, then it is implied that the instance belongs to the class C.

2.3.2 Multi-label Classification

Multi-label classification is a subset of classification in the machine learning domain. Classic classification tries to assign each datapoint to a single class. In multi label classification you want to assign a set of labels $\lambda_j \in L$ of an unknown number $j \geq 0$ to an instance. It could be zero or more labels that should be assigned, and its a fully acceptable state that an instance has the empty set as its label. Most common is that the number of labels $M > 2$, while if $M = 1$ it transforms into binary classification since it becomes a decision between no label or a label. In more simple terms, you want to based on the input features, assign the correct tags out of multiple options to an instance. [7]
The difficulty partly comes from not knowing how many labels that should be assigned as this can vary between instances, but also from the very large output space. In Neural Networks, multi label classification is quite natural. You have an output node for each of the different labels, and using an activation function which decides if this label should be included in the prediction or not. This prediction is done via a single classifier which would not be possible with a method such as SVM. This is a form of *method adaption*.

If you are using classifiers such as SVM or Random Forest, which are capable of multi-class classification, you instead need a different approach for multi-label classification. Instead of adapting the method, you transform the problem into a multi-class classification task. [8]

2.3.2.1 Problem Transformation

Binary Relevance is a method that treats each label as a binary classification problem. You train M separate binary classifiers on each of the labels λ_j. This method does not take interdependencies between the labels into account as you train a separate classifier for each label which predicts if a label should be present or not. The approach is relatively simple, but the consequence is that you lose the inter-label dependencies which can negatively affect performance of the model if the labels have relationships between each other.
Classifier Chains is similar to the binary relevance method except here you build chains of binary classifiers where each build upon the decisions of the previous binary classifiers. This method takes interdependencies between labels into account somewhat, however it will still fail to capture trends which are permutation dependent. If for example you have a label which will depend on the existence of another label, if that label has not been classified yet then the dependent label will not have the information needed.
Label Powerset solves the issue in a slightly different way. Instead, you keep the problem as a single class classification problem by creating unique classes for each of the combinations of labels present in the training data. This can create a huge number of possible classes where some of them could have very little support. It also makes it impossible to make predictions of label combinations that are not present in the training data. The method can work well when you have a limited number of labels so the number of classes does not become huge, otherwise it suffers from performance issues and computational complexity.
As the problem gets broken down into a binary classification problem in most cases when dealing with multi label classification, you should consider this when choosing a classifier. Something well adapted for binary classification is likely to perform well. [8]

2.3.2.2 Method Adaption

Another alternative to transforming the problem into a domain which is suitable for existing implementations of classification methods, is to instead adapt the methods to suit multi label classification. This can be done in many ways and will be specific to the method which is being adapted. The aim is to customize the method to natively be able to handle multi label classification with a multi label output. Many gains can be achieved here, especially compared to the binary relevance problem transformation method which does not take the label interdependencies into account. Examples of such methods which have been adapted are *Multi-label k-Nearest Neighbours*, *Multi-label Decision Trees*, and *Neural Networks* with a multi label activation layer. [8]

2.4 RAKEL: Random k-Labelsets

RAKEL (Random k-Labelsets) is a machine learning method specifically designed for multi-label classification tasks as described in Section 2.3.2. Since this is not a separate method but instead a way to apply a normal multi-class classifier, it is a form of problem transformation.
RAKEL can in simple terms be explained as dividing all the labels into smaller sets, either overlapping or distinct, then applying label powerset on these smaller sets and training a classifier for each. The results of these are then aggregated through majority voting into a final prediction, making it an ensamble method.
RAKEL is an application of label powerset, which is described in Section 2.3.2.1. Compared to regular label powerset, RAKEL does not create classes out of all the possible combinations of labels. Instead, it divides the initial set of labels L, into random subsets, called labelsets. The labelsets are used to create classes out of all possible combinations of the labels, this turns the problem into a multi-class classification task. If a labelset contains all of L, you get the regular label powerset method. In RAKEL, the parameter k specifies the size of the labelsets, meaning how many labels are included in each labelset.
With these labelsets created the regular *Label Powerset(LP)* method is used on each of these subsets. This means training a given multi-class classifier on the new transformed task of predicting one of the classes which represent a combination of labels. The difference here from regular LP is that you do this on smaller labelsets and train multiple classifiers, instead of one on a huge set of classes.
RAKEL is an *ensemble* method, meaning that it trains several models and aggregates their output into a final prediction. There are two ways of dividing the labels into labelsets of size k, either disjoint (RAKEL$_d$) or overlapping (RAKEL$_o$):

2.4.1 RAKEL$_d$

Using the disjoint labelsets, you will not have any overlap in labels between the classifiers. Given the size of the labelsets k, you divide the set of labels L randomly into $m = M/k$ disjoint labelsets. When m is an integer, you get evenly sized labelsets, if m is a fraction you round it upwards and the final labelset is the size of $M \mod k$, meaning a smaller labelset with the remainder. Using LP on each of the labelsets, you transform the labelset into classes of all combinations of labels in the labelset. You then train m multi-label classifiers, one for each labelset. Each of the classifiers will then make a prediction of a class from their respective labelset. This will then, in turn, be aggregated by taking a union of these disjoint sets of predicted labels, returning a list of labels that makes the final prediction.

2.4.2 RAKEL$_o$

When using overlapping labelsets, the same label may occur in several labelsets. The set of possible labelsets of size k is defined by $\binom{M}{k}$. With overlapping labelsets m is an input parameter, signifying the number of labelsets, with the restriction $m < \binom{M}{k}$. The m labelsets are drawn from the set $\binom{M}{k}$ by random sampling without

replacement. This makes it so that the sets can overlap, while when $mk > M$ the overlap in the labelsets is certain. It is not guaranteed that all labels will be included in a labelset.

A multi-class classifier is trained on each of the m labelsets. As the labelsets may overlap it is not as simple as in RAKEL$_d$, the final prediction for each label is instead aggregated using majority voting. The decision is taken for each label separately, first you take all the labelsets where the label is present. After this you record the decision by that classifier to either include the label or exclude it, if included you assign a value of 1 otherwise 0. The mean of all these decisions is then calculated to give an average vote, if this is greater than 0.5 the majority voting decision is to predict the label, otherwise the prediction is that the label is not present.

RAKEL reduces the computational complexity dramatically when the number of labels M becomes large. The recommended value of m is $2M$, but even though there are many labelsets, due to combinatorics, it will still result in a lower computational complexity. It also partly solves the issue of low support for many of the labelsets, since there will be more occurrences of each of these sub-labelsets in the data. Another limitation of a regular label powerset method is that it can only predict labelsets that are seen in the training data. Using RAKEL however, new combination of labels can be created in the ensemble stage of the method.

RAKEL has several advantages over using the regular label powerset method.

This reduction in computational complexity is valid when using a multi-label classifier only capable of binary classification, and uses methods such as one-vs-rest as described in 2.3.1. However, decision tree classifiers are sublinear in complexity to the number of classes. Meaning that, the computational complexity scales better than linear to the number of classes ($\mathcal{O} < \mathcal{O}(n)$). If this is the case, then the computational cost of using RAKEL is likely higher compared to label powersets. This means that RAKEL should preferably be combined with a classifier such as SVM, and is not suitable for random forest.

Another big advantage to RAKEL compared to label powerset is that it can predict labels which are not present in the training data. If you have many labels it is very unlikely that all combinations are represented in the training data, while it is likely that unseen data should be tagged using new label combinations. RAKEL also solves the problem of low support in many of the classes, when having a huge number of classes representing each present label combination, there will be very few occurrences in many of the classes. However, when using RAKEL you get smaller labelsets which have a chance of having a much higher representation in the training data and therefore providing more support. [7]

2.5 kNN - k-Nearest Neighbors

kNN (k-Nearest Neighbours) is a common but relatively simple machine learning method. It is a classifier which naturally handles multi-class classification directly. The method places the instance in the feature space of the training data, and, as the name describes, the k nearest neighbours are used to predict the class of the instance. It works based on the idea that similar instances are likely to belong to the

same class. Out of these neighbours the class with the most occurrences is chosen for the instance as the prediction.
There can be several different measures of distance which determines which points are the k closest neighbours, the typical distance metric is the Euclidean distance, but Manhattan(Grid, L_1-Norm), or Minkowski(L_p-Norm) distance can also be used. Even though it is a simple method, it allows for very complex decision boundaries between classes which are non-linear, this can be useful if the relationships between features is known to be non-linear and complex.
The main parameter to the classifier is the value of k, this determines how many of the nearest neighbours of an instance should be found. A small value means that the classifier will be sensitive to noise in the data as very few points will be used for the decision. It also makes the model quite prone to overfitting as it will closely follow the pattern of the training data, on the other hand it also makes the classifier more sensitive to local patterns. On the other hand, a large k smooths out the decision boundary between classes making the classifier more robust. Because of this the hyperparameter k needs to be tuned against validation data to prevent overfitting and find a balanced k. This is often done through cross validation.
A big advantage of kNN is its simplicity, it is a method which is easy to understand and reason about, while also being very simple to implement yourself. It also does not require any training phase, however it does need to calculate distances to all points given an instance. This makes it computationally expensive in datasets with a large amount of data.
kNN can and has been applied in many domains, the performance if often good with regards to the simplicity of the method. Successful implementations have been seen in various domains such as: image recognition, text categorization and recommendation systems. [3]

2.6 MLkNN - Multi-Label k-Nearest Neighbors

MLkNN (Multi-Label k-Nearest Neighbours) is an adaptation of the traditional k-Nearest Neighbours (kNN) algorithm tailored for multi-label classification tasks. This method leverages the principles of kNN while incorporating mechanisms to handle multiple labels simultaneously. It is the first multi-label lazy learning algorithm, which means that the generalization of the training data is delayed until a request of a prediction is made, unlike eager learning methods where the model is trained on the training data first, and then predictions are made. This does however mean that the computational cost will be incurred at runtime instead of training time. Meaning that the user of the model will have a higher query time, compared to other models where most of the time is spent in training, but later querying of the model will be fast.
The MLkNN algorithm works similarly to regular kNN, it finds the k nearest neighbours to the instance in the same way, but since multiple labels needs to be predicted instead of assigning the instance to a single class, more needs to be done. The number of neighbours belonging to each label is used with the maximum a posteriori principle to determine the labelset of the instance.

2.6.1 Technical Details

To understand more precisely how the method works we need to dive into the more technical details of the method.
We will define an instance as $x \in \mathcal{X}$ where $\mathcal{X}$ is the domain of all instances, and $Y \subset \mathcal{Y}$ where Y is a labelset and $\mathcal{Y}$ is the set of all labels.
We then define y_x as the category vector of x with the length equal to the number of labels $|\mathcal{Y}|$, where each of the labels is represented as a 1 if the label is present and 0 if its not. We let $N(x)$ denote the set of k nearest neighbours of x in the training set. With this we can define a membership counting vector based on the labelsets of the neighbours as:

$$C_{x,l} = \sum_{a \in N(x)} y_{a,l}$$

This counts the number of neighbours of the instance x which has each label l.
We pick a test instance $t \in \mathcal{X}$. Then let H_1^l be the event that t has label l and H_0^l be the opposite that t does not have label l.
Then E_j^l be the event that, among the kNN of t, there are j instances which have the label l. Using the maximum a posteriori principle, the category vector of labels for the test instance, y_t can be determined as:

$$y_{t,l} = \arg\max_{b \in \{0,1\}} P\left(H_b^l | E_{C_{t,l}}^l\right)$$

$E_{C_{t,l}}^l$ is the event that we have the count of label l from the nearest neighbours that we have, so practically, it is the count of neighbours with the label. Given this we then get the conditional probability of the label l being b so either 1 or 0. We are then trying to find if either $b = 1$ or $b = 0$ maximizes the probability, which means, is it more probable that the label is present or not.
With the goal of the calculation clarified and the notation explained, Baye's rule will now be used to rewrite the equation into something we can calculate:

$$y_{t,l} = \arg\max_{b \in \{0,1\}} \frac{P\left(H_b^l\right) P\left(E_{C_{t,l}}^l | H_b^l\right)}{P\left(E_{C_{t,l}}^l\right)}$$

$$y_{t,l} = \arg\max_{b \in \{0,1\}} P\left(H_b^l\right) P\left(E_{C_{t,l}}^l | H_b^l\right)$$

This shows that all that is needed is the prior probabilities $P\left(H_b^l\right)$ and the posterior probabilities $P\left(E_{C_{t,l}}^l | H_b^l\right)$ which can be directly estimated from the training set by frequency counting. For the exact details around how this is done and the algorithm used refer to the paper introducing the method.
ML-kNN has 3 main hyperparameters number of neighbours k, and the distance metric. Exactly like in regular kNN, k is the number of neighbours of an instance to take into consideration when making a prediction. Also similar to kNN several distance metrics can be chosen from, but the most common is to use regular Euclidean distance. Also, here it is common practice to use cross validation to determine the most suitable k for the dataset

In the paper introducing the method it was compared to several other multi-label classifiers where it was shown to outperform them in 3 different classification tasks. It was also seen that the model was relatively insensitive to the parameter k. [12]

2.7 Random Forests

Random forest is a machine learning method that is commonly used in both classification and regression. It is an ensemble method which constructs multiple decision trees during training and combines their output to make a final prediction, which improves accuracy and reduces overfitting.
Random forest is based around binary decision trees. Such a tree is built up by several binary decisions, splitting each node into two branches. At each node the model is given a random subset of features which it can use for determining the split. The reason for splitting on a random subset of features is that if a feature is a very strong predictor for the target, the feature will be selected in many of the trees causing them to become correlated. By doing this random selection of a subset of features you avoid this and thereby get higher accuracy.
The number of features which are considered in the subset for each split is a parameter of the model, if there are p features input into the model, then for a classification task you usually use $\sqrt{p}$ randomly selected features in each subset. This hyperparameter can be tuned to become optimal for the given problem it is applied to.
The split itself in each of the nodes based on the random subset of features is based around either *information gain* or *Gini impurity* to determine the optimal cut point.
The whole idea of the random forest model is to build up several of these binary decision trees where each split is determined based on a random subset of features, this is what creates a forest of many trees. Random forest is an ensemble method using *Bagging (bootstrap aggregating)*, which means that you utilize bootstrapping of samples to generate several training sample sets for the different decision trees, that are then aggregated into a final decision.
Bootstrapping works by generating a new sample set based on the existing dataset by sampling with replacement. This means that you draw a given number of random samples, where it is possible to draw the same row multiple times. These samples which are taken from the original dataset is bootstrapped samples. The idea of this is to artificially generate more sample data than is originally available.
Each of these samples generated using bootstrapping is then used to train a decision tree classifier which can then classify an instance. The aggregation part comes from aggregating each of the classifier's decisions into a final decision. If you are doing classification this is done by taking the *Mode* of the outputs, meaning performing majority voting and taking the most commonly predicted class. If doing regression, you typically take the average of all the outputs from the trees in the forest. Figure 2.1 describes the Bagging process visually.
Random forest offers several advantages such as being resilient to overfitting compared to using a single decision tree, it's also relatively insensitive to hyperparameters. Additional benefits are that it can handle both classification and regression tasks as well as providing direct measures of feature importance. There are some drawbacks to the method such as the computational demand of training several trees

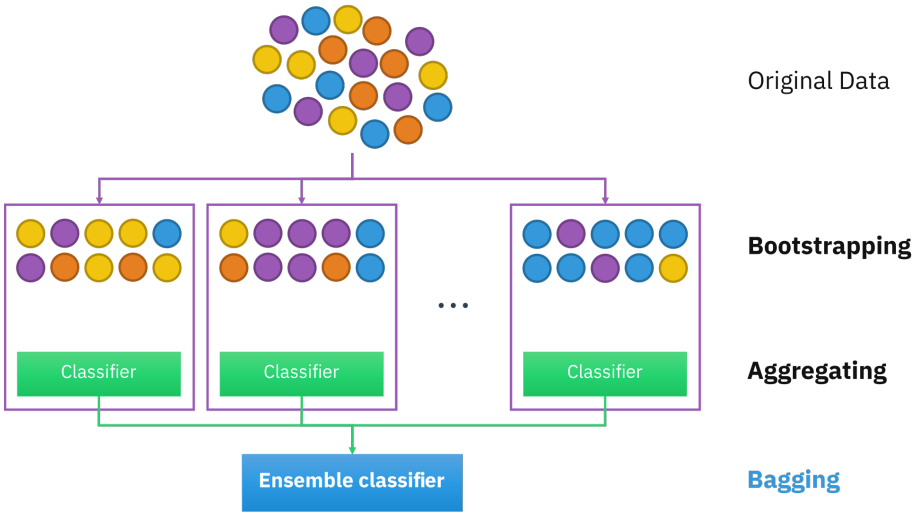

Figure 2.1: A simple illustration showing the basics of how random forest works.

instead of only one. You also lose some interpretability of the model when using an ensemble of many trees, but this can instead be provided by looking at the feature importance which gives some explanation.
In summary, Random Forest is a powerful and flexible machine learning method that leverages the strengths of multiple decision trees with a random subset of features, where each trees prediction is aggregated to make the final prediction. This in general provides robust and accurate predictions which are resilient against overfitting and not sensitive to its hyperparameters. [10]

2.8 Neural Networks

Neural networks are a type of machine learning methods which are inspired by how the human brain works. They consist of a large number of nodes which are connected, and organized in several layers. These neural networks can learn complex patterns from data and are nowadays widely used in many different applications.
A neural network consists of an input layer, one or more hidden layers and an output layer. Each neuron in these layers is connected to neurons in the previous and next layer, but not within its own layer. Each of these connections have a weight determining the transfer.
A network also uses an activation function, this function is applied at each node and, along with the weights, determines the output of that node given all the inputs from its connections from the previous layer. There are several common functions such as: sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). These functions help with introducing non-linearity into the system which allows the network to learn complex relationships which can be nonlinear.
During the training on data the network adjusts these weights based on the error of its predictions, this is what is called backpropagation. By adjusting the weights, the aim is to reduce the error as much as possible to increase the networks performance. If the weights are adjusted too much to the training data you can get issues with

overfitting, which is why you control this against unseen validation data.
There are several different kinds of neural networks which vary in architecture. The most common and standard one is *feed forward neural networks* which only has nodes that connect to nodes in the next layer, a variation of this is *convolutional neural networks*, which are common in analysing grid-based data such as images. Here a convolution using a kernel is utilized to combine each cell in the grid with the ones around it and summaries this into one output, where each convolution takes the dimensionality down. There are several other variations, but these are the most basic and commonly used.
Neural networks are very powerful and can fit to complex data, but they have certain limitations. They require very large amounts of data and are very computationally demanding to train. They can also be very prone to overfitting such that the model performs very well on the training data but not so much on new unseen data, but there are several techniques that can be utilized to mitigate this.
In summary, Neural networks are a common technique used in machine learning which can be applied to many different problems and often generate great results. However, it requires a lot of data to be successful and is very computationally demanding to train. [9]

2.9 Support Vector Machines (SVM)

Support Vector Machines (SVM) is a machine learning method that is widely used for both classification and regression tasks. SVM is built around finding the optimal hyperplane that separates data points of 2 classes with the maximum margin.
The main idea in SVM is to find a linear separator between two binary classes using a hyperplane that separates these classes. To do this, SVM maximizes the margin between the hyperplane and support vectors. The closest datapoints of the two classes to the decision boundary are what is known as support vectors, and the distance between these and the hyperplane is called margin. The method tries to place the hyperplane in a position so that it has the most margin to both classes support vectors
In a lot of datasets, the data is not possible to linearly separate with a hyperplane, therefore SVM uses kernels to transform the feature space into a space of higher dimensionality where the classes are hopefully linearly separable. This use of kernels allows the method to handle non-linear relationships since the hyperplane in the higher dimensionality space can be non-linear in the original feature space.
SVM does not handle multi-class classification natively, the hyperplane naturally will separate binary classes. This means that some strategy for multi-class classification needs to be used. The most common one is to use one-vs-rest as described in 2.3.1
The choice of kernel function is an important parameter in SVM. Some kernels that are common in SVM are: linear kernel, polynomial kernel, and radial basis function (RBF). When using a linear kernel, the feature space is not transformed into higher dimensionality and a hyperplane is found in the original feature space. This generally only works well when the data is linearly separable. However, the non-linear kernels allow SVM to learn non-linear relationships which makes it useful for a wide range of classification tasks

Fruit	Categorical value of fruit	Price
apple	1	5
mango	2	10
apple	1	15
orange	3	20

Table 2.1: Fruit, Categorical Value, and Price

Fruit_apple	Fruit_mango	Fruit_orange	Price
1	0	0	5
0	1	0	10
1	0	0	15
0	0	1	20

Table 2.2: One-Hot Encoded Fruits and Prices

Other than the choice of kernel there are other hyperparameters which can be tuned. The regularization parameter C controls the trade-off between maximizing the margin and minimizing the classification error. Depending on the kernel used there are hyperparameters for the kernel such as the degree of polynomial when using a polynomial kernel, or the parameter γ when using the RBF kernel.
In summary, Support Vector Machines is a good machine learning method for binary classification that finds the optimal hyperplane for separating data points of two classes. By using kernels, the method can transform the feature space into a higher dimension and thereby fit non-linear decision boundaries between the classes. [11]

2.10 One-Hot Encoding and Multi Label Binarizer

One-hot encoding is a method for preprocessing data to prepare it for a machine learning model to be trained on. The method converts categorical variables into a binary format through splitting the categorical value into several binary features. Each category is converted into an individual feature, this feature then takes a binary value of either 0, if it is not the class in the data record, or 1 if the class is present. In Table 2.1 we can see some example data, and in Table 2.2 the same data is one-hot encoded to demonstrate the method.
The benefit of one-hot encoding is that the method eliminates ordinality, meaning the order of categories. Many categories such as fruit have no inherent order and in the categorical encoding seen in Table 2.1 we see that the categorical value of fruit has an increasing series of numbers. A model does not know if a column represents numbers which are ordered or not, therefore this could be misinterpreted when training and bias the model on trends which do not exist. By one-hot encoding this into binary variable we can eliminate the ordinality at the sacrifice of getting higher dimensionality. [1]
One-hot encoding can be applied when you have a multiclass categorical variable,

Genre	Rating
{Sci-fi, Thriller}	6
Thriller	8
Sci-fi	7
{Thriller, Comedy}	6

Table 2.3: Genres and Ratings

Genre_scifi	Genre_thriller	Genre_comedy	Rating
1	1	0	6
0	1	0	8
1	0	0	7
0	1	1	6

Table 2.4: One-Hot Encoded Genres and Ratings

however if you have a multilabel categorical feature another method is required. A very similar method which is based on one-hot encoding is multi label binarization. In a similar way it finds all the unique classes and creates binary features from them, the only difference is that each record could have several labels or classes assigned. The result of this is that several of the columns can have a value of 1 whereas in one-hot encoding they are mutually exclusive, meaning that if one column has a value of 1 then you can deduce that all the others have the value 0. An example of data that is pre-processed using this method can be seen in Table 2.3 and then encoded in Table 2.4 [5]
What is important to note when using multi-label binarization that the order of categories is lost.

3
Implementation

The implementation will consist of two phases, first a phase where the pipeline is built up. This includes developing a good feature extractor as well as encoding of data to suit a machine learning model. It will also include implementing a basic model which can make affix predictions, as well as a decoder which makes the final prediction of an equivalent designation to the input product.
The second phase will involve the exploration and implementation of several machine learning models and methods which will be evaluated and compared to each other. The first phase will enable this since all the surrounding building blocks will be done. This means that different models and methods can be swapped in place since the rest of the infrastructure surrounding the model is set.

3.1 Overview of the Solution

An overview of the larger building blocks of the solution can be seen in Figure 3.1
The solution consists of several steps. First, a preprocessing step where features are extracted from the input string, which is the competitor designation. The similar feature extraction is performed to extract the labels of the target. This data is then encoded to make the format suitable for a machine learning model. The features are used to predict the affixes(labels) of the equivalent product, which is the task the machine learning model is trained to perform through multi-label classification. After the model has been trained and can predict affixes, these need to be decoded into a real designation of the equivalent product. This is done by parsing the main designation from the competitor and assuming it will be the same in the equivalent product. The predicted affixes are then utilized to find the right bearing, of the right size, with the right features. Through this decoding step, a final prediction of an equivalent product is determined, which is then validated against the true value in a test set to determine the performance of the overall process.
By studying the test cases and identifying patterns where the process fails, areas of improvement in the pre or postprocessing of the data are then identified. These will be investigated in the hope of improving the outcome of the pipeline.
After this initial phase of development of the pipeline, the model evaluation will begin. Initially the only model used will be random forest with the binary relevance method to handle the multi-label classification. This will be done for simplicity and to keep the results easy to compare when making pipeline alterations. As seen in Figure 3.1, the model is its own block, and is therefore replaceable. Meaning that different multi-label classifiers can be swapped in it's place. In this stage no

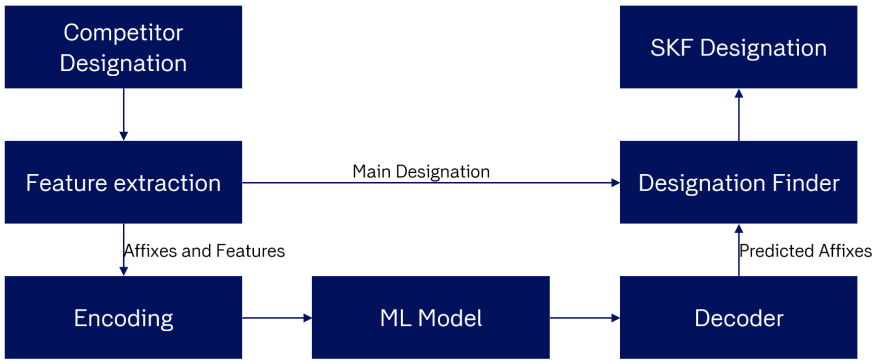

Figure 3.1: Schematic describing the larger building blocks of the pipeline

alterations will be made to the pipeline.

3.1.1 Mathematical Description of the Problem

To bring clarity and to clearly define the problem, a more mathematical definition will be presented here.
We define the competitor product as $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of all competitor products. We define the target product as $t \in \mathcal{T}$, where $\mathcal{T}$ is the set of all target products. Important to note here is that these sets will not be equal in size, some competitor products do not have a target equivalent and vice versa.
We define $P : \mathcal{S} \in \mathcal{C} \rightarrow \mathcal{T}$, where $\mathcal{S}$ is the subset where a target equivalent product exists. P is therefore the map between a competitor product and the equivalent target product, representing the pipeline for finding interchanges.

3.2 Pipeline Implementation

In this section the implementation and development of the pipeline for learning the function P (predicting interchanges) will be described. The focus will be on the pipeline itself and not the machine learning model, therefore the model will not be tweaked. The model that will be used in this development is Random Forest with Binary Relevance, through initial testing it has been seen that the model is quite capable and can be used with confidence.

3.2.1 Preprocessing, Feature Extraction, and Encoding

The structure of the given data is simple, as we have decided to handle a single brand we are left with two columns: the competitor's designation, and the corresponding target designation. While this is enough for a user it won't be possible to train a machine learning model on these direct relationships. Therefore, feature extraction needs to be performed as a first step to the process.

	Competitor	Target
Designation	23030-E1A-XL-K-M-C3	23030 CCK/C3W33

As described in Section 2.1, the product designation contains a lot of detail about the specification of the product and its features. These features need to be extracted so that the model can be trained on them. As the competitor and target manufacturer have different designation systems for the suffixes, the feature extraction will differ between them and needs to be treated separately
The feature extraction of the competitor designation is easy due to the nature of their designation system. Each prefix or suffix is split by a delimiter, this used to be "." but has now changed to "-". Both will be used as delimiters which splits the designation into parts. The main designation also needs to be extracted, this is done by extracting terms of more than 4 numbers in a row. With this logic the main designation can be extracted which leaves a list of suffixes delimited by "." or "-" which is used to split the suffixes into an array.
The feature extraction of the target designation is more complex. Since this designation system does not delimit each suffix by a character they come as a compressed string. From this compressed string each suffix needs to be extracted for maximal detail. To achieve this an already existing list of suffixes will be utilized, the designation is matched against this list of phrases. To ensure the right match, the opposite of a *greedy search* will be performed, that means we are from left to right searching for the longest matching phrase. Once this has been found the index is reset and the search continues along the rest of the string until the end is reached. Provided that the suffix phrase list is complete, this leaves an array of suffixes contained in the designation. The reason this approach works is because if it were not true that you could take the longest matching phrase, the designations could be interpreted in several ways which would cause issues and confusion. So, because of the underlying system this behaviour is guaranteed.

	Competitor	Target
Designation	23030-E1A-XL-K-M-C3	23030 CCK/C3W33
Suffix List	[E1A, XL, K, M, C3]	[CC, K, C3, W33]
Main Designation	23030	23030

With this feature extraction we can now begin the encoding of data. A big problem is the permutation of the features, the position of suffixes meaning the same thing can vary. If we were to split the suffix list into columns, we could get suffixes meaning the same thing in different columns, and matching this is extremely difficult as it would require that each column specifies each of the options that it could contain. This requires a huge amount of effort and is not desired. The main complexity of this project lies in how to handle the features in a good way which allows a model to be successfully trained on the data.
The key to solving this is to encode each suffix as its own binary categorical feature. The net list of suffix features can be extracted as all the suffixes occurring in the dataset, this can be done for both the competitor and target. Practically this is done by adding all suffixes to a set from each of the suffix lists, since the set guarantees

uniqueness we are left with a list of all suffixes present. This can be seen as a form of one-hot encoding.

The main gain of this is making the model permutation invariant, thereby ensuring that no matter the order and position the model will be able to separate the features. The drawback to this is that you lose the permutation of the prediction as well. With making a simplification in one place you almost always get a trade-off in another. In Section 3.2.4 the process for solving this loss of permutation will be described.

3.2.2 Evaluation

To evaluate the models, the data needs to be split into a training and test set. The chosen split is 70% training data and 30% validation data, which is a slightly larger validation set than what is common practice in the machine learning domain (8020). The training data is used to fit the model, and to reduce overfitting, the validation data is utilized to determine performance. This way, the model is tested on how well it generalizes to unseen data. Otherwise, it would be rewarded for overfitting, giving great accuracy but sacrificing performance on new, unseen data. The validation data is therefore used for testing the model and evaluating its performance. The model is first fitted to the training data, and then the validation input is used to get predictions from the model. These predictions are compared to the true value of the target.

3.2.2.1 Affix Conversion Accuracy

The machine learning models will output a set of affixes that it predicts are equivalent to the affixes from the competitor. Since this is the only conversion the model itself does, it can only be evaluated based on the accuracy of the affix predictions. A prediction is considered correct only if the predicted set and the true set are identical. If there is an affix missing or an additional affix, it is considered an incorrect prediction. This gives us a metric of how well the model performs.

3.2.2.2 Designation Conversion Accuracy

The true desired output is the equivalent designation to the competitor product. The prerequisite for this is a good affix conversion; if the affix conversion is not correct, you cannot expect a correct designation conversion. The affix accuracy will therefore place an upper limit on how good the designation accuracy can become.

The designation accuracy metric defines a correct conversion as one that predicts the exact same designation as listed in the true value of the validation data. Any deviation from this will be considered a false prediction. There are certainly cases where the existing reference is not optimal and there are better interchanges that provide a product that is a closer equivalent or equally as good. However, this is very hard to measure, so accuracy will be the metric of choice.

3.2.2.3 Incorrect Prediction Rate

When utilizing the model for generating interchanges, you will always need to validate the predictions manually. If you are validating predictions, you would rather like to see an empty output if the model is not certain of the prediction than having all lines populated and having to discern the correct from the incorrect. If most of the predictions are correct, then the validation becomes much easier, and you build more trust towards the model. Because of this, we want to minimize the rate of incorrect predictions and rather have more empty outputs.
To measure this, we introduce the incorrect prediction rate, which is the rate of non-blank predictions that are incorrect. There will of course be a balance between having high accuracy and a low incorrect prediction rate, since you do not want a model which never makes a prediction.

3.2.3 Initial Result

Now that the data is extracted into well-defined features, the model can be trained. Since we, at this stage, are not interested in comparing models, Random Forest using binary relevance will be implemented. This gives a base measure of performance that can be used to tweak the pipeline.
A random forest model was set up without any additional tuning or work; the default parameters were used except for setting a random state to ensure consistency between runs. Since random forest is robust to changes in its hyperparameters and configuration, and since the implementation in scikit-learn supports binary relevance by default, it is very easy to implement. The result from a random forest model without any tuning on the first try was a affix prediction accuracy of 0.69.

3.2.4 Decoding

The model predicts which affixes should be present in the conversion, but we need to transform this into an actual product designation. First, the encoded data needs to be decoded back into a set of affixes, which are then used to identify a product. This is done by searching a database of all existing products to find the best match.

3.2.4.1 Initial Search and Filtering

Given that the number of target products exceeds 700,000, the search must be conducted efficiently to reduce processing time. This ensures feasibility when processing and decoding many predictions. The final accuracy is based on identifying the correct product, not just the suffix list, even though the model itself is evaluated on suffix list conversion.
To address this computational complexity concern, a search is performed to find designations containing all the suffixes and the main designation of the competitor product. This introduces a limitation: only cases where the main designation is consistent between competitors and the target manufacturer will yield good results. While this generally holds true, there are exceptions. This trade-off enhances model consistency and reliability but limits the ability to predict certain product types.

3.2.4.2 Evaluation and Matching Algorithm

After filtering, the resulting list of possible target designations varies in length. For uncommon types, it is usually fewer than 10, while for common product types and sizes, it can be hundreds, especially when the affix list is short. These need to be evaluated to determine the final prediction. An algorithm is used to assess how much of the designation matches the predicted phrases.
The algorithm finds all matching designations and evaluates each to find a matching score. This score is calculated by summing the lengths of the affixes and main designation and dividing by the length of the designation candidates. A score for how good the match is to the main designation and the predicted affixes, this is called a match score. This is determined by the proportion of the designation that the main designation and affix list covers. The designation with the best match score is chosen. Initially, this process was slow, taking hours to complete, as the search was performed against the entire database for each prediction.

3.2.4.3 Algorithm Optimization

By investigating cases where the suffixes were correctly identified but the correct designation was not found, issues with the algorithm were identified. Many super precision bearings had issues due to having many short suffixes which are sometimes contained in each other. The problem was that the same suffix could match multiple times. For example, if the suffixes are *P4A* and *A*, *A* could match twice, theoretically resulting in a match exceeding 100%. To fix this, affixes are sorted in descending order of length to avoid matching a substring of an affix. Once a match is identified, the substring is removed from the designation being searched to prevent multiple matches. This resolved errors and significantly improved accuracy, approaching the upper limit set by affix conversion accuracy.

3.2.4.4 Speed Optimization

While effective, the method was slow. Validating the test set took hours, which was impractical for constant iteration and too slow for users converting long lists of competitor designations. Initial speed optimizations reduced runtime to 15 minutes, but this was still too slow.
To resolve this, the list of all designations was pre-processed to extract the main designation and a list of suffixes using the same preprocessing as the training data. The affix list was converted to a set, and the search for matching target designations was ran again. The input is a main designation from the competitor and a set of predicted affixes. First, the main designation is compared to all target main designations, retaining only matching lines. If the affix list is empty, only a designation without suffixes but with a matching main designation is returned, which can be only one product. If there are affixes, a set comparison checks if the predicted affix set is a subset of the target set, retaining only those matches.
This results in a list of designations where the main designation matches, and the predicted suffixes are a subset of the extracted suffixes. Using affix extraction into a set eliminates problems with multiple matches on the same string part. Since all

remaining designations contain the same affixes, the shortest remaining designation will always have the best *match ratio* according to the verified logic. This preprocessing and filtering dramatically reduce processing time and improve the logic's explainability and interpretability. This is the final logic used for finding a product designation from a competitor's main designation and a list of predicted affixes.

3.2.5 Pipeline Improvements

The initial performance of the random forest model was quite good, even without tuning and using the default hyperparameters. However, this is not the model's full potential, which is why fine-tuning was performed.
Studying the cases where the model failed to make the correct prediction provided insights into areas for improvement. From this, several fine-tuning attempts were made.

3.2.5.1 Cleaning of OEM Numbers

Original Equipment Manufacturer (OEM) interchanges are references from a part number which does not carry any meaning or feature descriptions, it is simply a random number that specifies a product. Therefore, there is no pattern that can be used to predict the correct target designation, so the OEM references are impossible to predict. The hypothesis is that the existence of these in the training data confuses the model and causes incorrect patterns to be learned. To explore the impact of these references, they were removed from the training set to prevent negative effects on the model. The OEM number references were filtered out by removing all references that do not have the same main designation between the competitor and target product. This also consequently removed some other references, such as housings, since the main designation does not always match. This somewhat negatively affects these types of products, but they are unlikely to match either way since our decoding of suffixes into a designation relies on the main designation matching as a prerequisite. The result was surprising. The training and validation sets remained the same, but the training set was filtered to remove non-matching main designations. This kept the results as comparable as possible. The accuracy dropped slightly, and false predictions increased slightly. Therefore, this idea was dropped as it was shown that OEM numbers were not negatively affecting the model. The impact on validation remains, as it will never be possible to reach 100% accuracy given that OEM numbers in the validation set are impossible to predict.

3.2.5.2 Adding Features Extracted from the Designation

The first improvement involved including the first three characters of the competitor designation as a new feature. In many cases, this provides valuable input to the model, as the first three characters usually specify the product family. This determines which suffixes to choose from and how they should be interpreted. For different product families, the same suffix could mean different things. Therefore, if the model knows the product family, it will have an easier time predicting the correct suffixes. However, sometimes a product has prefixes, or is of non-standard types.

The first 3 characters will then not indicate the family, however it still provides additional information. The first three characters strike a good balance between having many occurrences of each value while still providing enough detail.
In some product families, suffixes change depending on size. For example, in the Spherical Roller Bearings, a design by the target manufacturer indicated by the suffix *CC/W33* was replaced by a new product design labelled *E*. This means that the old design is no longer produced or sold and instead the newer design is preferred. However, this change was only for smaller sizes with an inner diameter under a certain threshold. In failed cases, several instances were identified where the suggestion was *CC/W33* but the true suffix was *E*, or vice versa.
The model should not be explicitly told this rule, but it can be given the information needed to learn it. Therefore, a new feature corresponding to the size of the bearing was added. As described in Section 2.1, the last two digits of the main designation describe the inner diameter of the bearing. By extracting these digits and making them a new feature, the model gains additional useful data.
This splits the main designation into semantically meaningful parts and decodes the domain semantics into features useful for the model. This is not valid in all references but provides enough information to be useful.
The outcome was a major leap in performance. Accuracy significantly increased first from adding the first three characters and then further by adding the size. Since the increase was significant, these two new features were added to the model. Features do not need to carry humanly meaningful information or be perfect and exact; they simply need to carry data that can separate classes. This explains why added features can still improve performance.

3.2.5.3 Refining the List of Suffixes

It was also observed that some target suffixes were not properly extracted from the designation. This indicated that either the parser was incorrect, or the proper term was missing from the list of suffixes. In all investigated cases, the root cause was the term missing from the list. Resolving this and finding a more complete list of prefixes and suffixes is likely to improve the model as it provides a more accurate representation of the features.
However, what resulted in a significant decrease in the model's affix conversion accuracy, caused a significant increase in designation conversion accuracy. This is not entirely unexpected, as the old table of affixes did not contain all affixes, leading to incomplete extraction and fewer possible target labels. For example, the suffixes *VA904* and *VA905* were not in the original list and were extracted as simply *V*. A model finds it easier to predict that the affix contains *V* than to predict the exact suffix. With several such cases, the suffix conversion accuracy decreases. However, the prediction of designations improves with a more complete list. If searching for something containing just *V*, either *VA904* or *VA905* could be possible matches, but only one is correct.
To summarize, with greater detail in affixes, it is easier to find the right designation given a correct affix conversion, but harder to make the right affix conversion. Predicting affixes is not enough if they are incomplete, which is why it is important to have good input data in the form of target affixes into the model.

3.3 Model Exploration and Comparison

Now that the pipeline is in place, the various options for machine learning models and methods will be explored. As explained in Section 2.3.2 there are a variety of options. Either adapting the machine learning model to be able to handle multi-label classification or as we have done up until now by transforming the problem into a multi-class classification problem. When evaluating models and methods, both of these will be explored.

The methods and models that will be explored are the following: **Problem Transformation**

- Random Forest with Binary Relevance (what has been used up until now)
- Random Forest with Label Powerset
- SVM with Binary Relevance
- Linear SVM with $RAKEL_o$
- Linear SVM with $RAKEL_d$
- Radial Basis Function(RBF) kernel SVM with $RAKEL_o$
- Radial Basis Function(RBF) kernel SVM with $RAKEL_d$
- kNN with Binary Relevance

Model Adaption

- Multi-Label k-Nearest Neighbors (ML-kNN)
- Neural Network with Sigmoid Activation Function

Each of the different models will need some basic hyperparameter tuning. A limitation in this exploration of methods will be the limited hyperparameter tuning for each of the models. This might result in all models not getting a fair chance but performing grid search for each method will be to computationally expensive without proper compute resources.

The basic hyperparameter tuning will be performed on the affix conversion accuracy only, but the final models will be compared based on both affix conversion accuracy and the final designation accuracy. The number of incorrect predictions will also be presented but might not be so relevant to compare between the models since it is mostly an effect of the decoding.

The test and validation split will be kept the same between the models to have as comparable output as possible. For utilization of the models themselves both sk-learn [4] and sk-multilearn [6] will be used which are python libraries supplying implementation of a wide variety of machine learning models and methods. While sk-learn is a very well developed and maintained library used by a lot of people daily, the same cannot be said for sk-multi-learn. This was developed as a tool to support research being done in the field, and was then released to the public for people to contribute and use. However recently it is not maintained, and in my limited usage I found a lot of bugs and incompatibilities with latest versions of libraries. This should be noted when judging the outcome of some of the methods, it could be that the method itself is good but that the implementation of it is poor or contains errors.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1150)	0
dense (Dense)	(None, 1500)	1,726,500
dropout (Dropout)	(None, 1500)	0
dense_1 (Dense)	(None, 1000)	1,501,000
dropout_1 (Dropout)	(None, 1000)	0
dense_2 (Dense)	(None, 500)	500,500
dropout_2 (Dropout)	(None, 500)	0
dense_3 (Dense)	(None, 247)	123,747

Table 3.1: Neural Network Architecture

3.3.1 Model Implementation

The different models will need to be configured and implemented in the pipeline. Since libraries are being used, the implementation of the models themselves will be left out, saving a lot of time and removing chances of mistakes. The challenges and discoveries in the implementation will be described here, but most of the details will be left out as the majority of it is repetitive work.

3.3.1.1 Neural Network Classifier

The first idea was to use a neural network, partly because of the buzz around this machine learning method, but also due to its ability to learn complex relationships. However, this approach was tried with some doubt due to the relatively small dataset, containing only 35,000 data points. The initial solution proposal was based on a neural network having each competitor suffix as a binary input node and each target suffix as a binary output node, as seen in Figure 3.2.
The hidden layers should then learn the relationships between these by tuning the weights and using non-linear functions between the layers. By including only the affixes present in the dataset for the competitor and corresponding target affixes, we end up with 1,150 input nodes and 250 output nodes. This shows how varied the inputs and outputs are.
Many different network configurations were tried. The smaller ones failed to pick up the underlying pattern, while more success was seen in the larger models. Through trial and error, the model that performed the best was the following:
The result of this network was still not satisfactory, as it did not generalize well to unseen data. The training was terminated early due to a lack of improvement in the validation loss for many iterations. The final accuracy score was 0.47, which will be compared to the results we can achieve with a random forest model.

3.3.1.2 Random Forest Classifier

A random forest model was set up without any additional tuning or work. The default parameters were used except for setting a random state to ensure consistency between runs. Since random forest is robust to changes in its hyperparameters and configuration, and since the implementation in scikit-learn can handle all kinds of

input and output formats, it is extremely easy to implement.

Some experiments were done with the hyperparameters of the model, but it was quickly realized that further search for hyperparameters would not lead to improvement. So, the working assumption is that the default parameters are close enough to being optimal.

The default solution for multi label classification when using random forest and scikit learn is binary relevance. To me this is a bit surprising, while random forest is great at binary classification, it is also a method which naturally handles multi class classification. The computational complexity of the model is sublinear in the number of classes, meaning that it scales more favourable than linearly to the number of classes. Rather than training n models where n is the number of output labels, you could instead use another method such as Label Powerset where you instead have many classes and leverage the strength of random forest, as described in Section 2.4.

Because of this, Label Powersets will also be explored. Here we instead make classes out of every combination of labels present in the training data, and train a single model on this multi-class classification problem. The performance of this will then be compared to random forest with binary relevance, and the rest of the models.

3.3.1.3 SVM

SVM is a very well-established model which can be used in many applications. Naturally it handles binary classification, which is why it comes to mind as a good classifier to try with the Binary Relevance method. It is also interesting to try out in combination with RAKEL as SVM was used in the paper where the method was suggested, especially since its described how the computational complexity is decreased dramatically when using RAKEL and SVM compared to SVM with Label Powerset. The reason RAKEL is not tried out with Random Forest is that this causes higher computational complexity compared to just using Random Forest with Label Powerset since the computational complexity of Random Forest is sublinear in the number of classes.

A big obstacle in using SVM is the computational performance. When using binary relevance, it takes an extremely long time to train classifiers for each of the output labels. The big blocker here is the number of input features, SVM does not scale well with increasing number of inputs. Initially the generic implementation which can handle both linear SVM and kernel based SVM was used. The performance was so bad that the training could not be completed. It was then discovered that an alternative implementation which is optimized for linear SVM only was available and that it scales better. This made the computational complexity feasible but still not good as will be seen in the results later on. Using kernels with the generic implementation on the other hand did not have the same issues with performance and managed much better.

However, in all cases where SVM was used the performance of the model itself was very disappointing.

3.3.1.4 RAKEL

The implementation from sk-multi-learn had several bugs and issues with the latest version of its dependencies, this caused runtime errors. The issues could be overcome in the case of RAKEL$_d$, but not for RAKEL$_o$. Therefore the performance of RAKEL$_o$ is not possible to investigate.

3.3.1.5 kNN Variations

The kNN variation implementation also contained issues. These could however be overcome, and the methods could be applied.

3.3.1.6 Notes on Computational Complexity

The remarks done on the computational complexity is based around the single threaded performance of the algorithm. Some methods such as Binary Relevance, does not have interdependencies between the classifiers. This would allow for parallel computations, which would greatly speed up the performance. Comparing the single threaded performance (running on a single computational thread) gives an equal base for the different classifiers. However, times for the models using multi-threading (running on multiple computational threads at the same time), and performance increases are made where possible will also be presented. These are not to be taken as the absolute optimal performance, as it is only an attempt at getting speed gains.

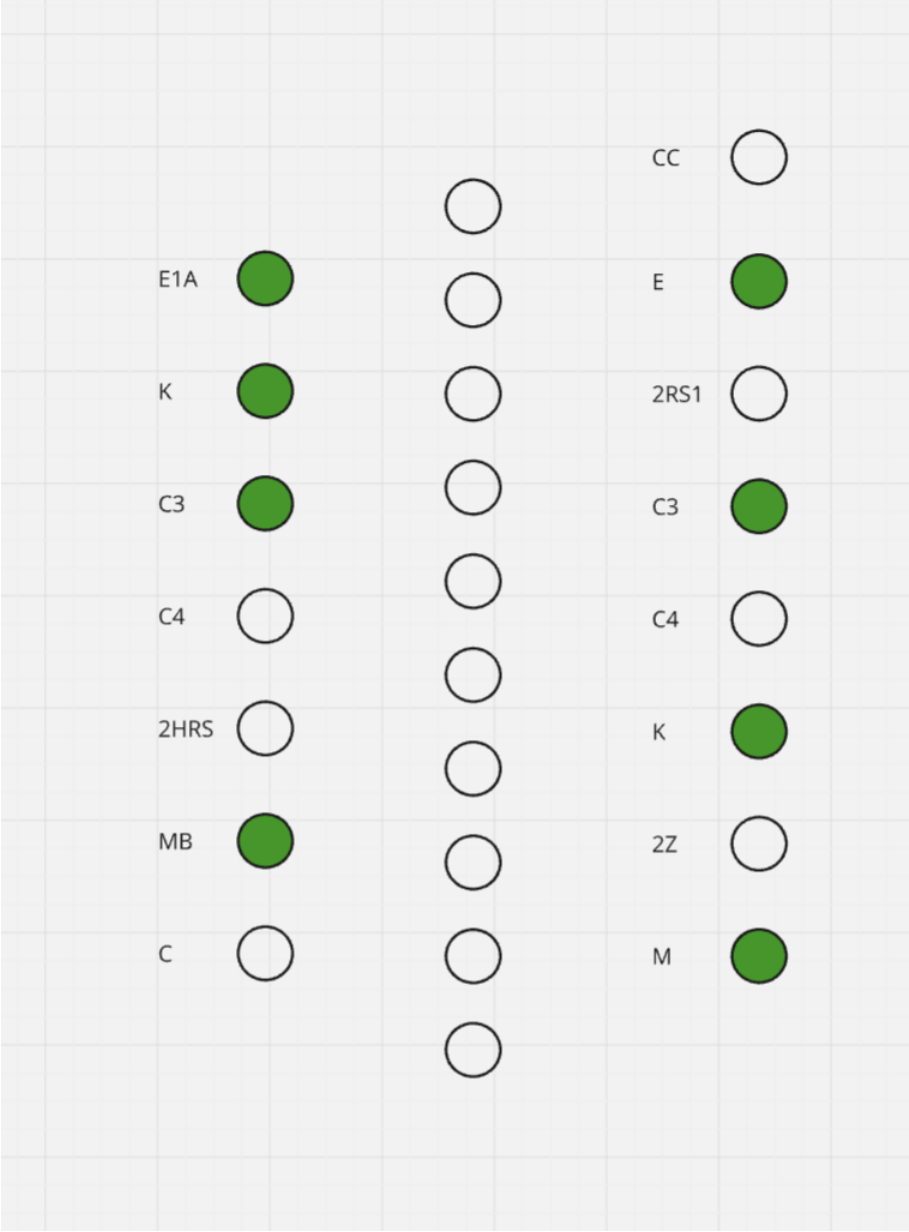

Figure 3.2: Illustrative drawing showing the input and output nodes with hidden layers in between.

4
Results

With the pipeline developed and several machine learning models evaluated, the results of this will now be presented here. First the results from the first phase consisting of pipeline development will be presented. Here alterations to the feature extraction and input data will be evaluated. The version that is determined to be the best is then what will be used when evaluating different machine learning models and strategies against each other to determine the best one. Lastly the input data will be studied to try to get a deeper understanding of why the performance looks like it does, and to get a sense of what accuracy is reasonable to aim for.

4.1 Pipeline Optimisation

With random forest as the preliminary model, the pipeline was evaluated. Several different test were ran to determine the best version of the pipeline, the results of these tests will be presented here.

4.1.1 Improved Affix List

When looking at cases where the affix prediction was correct, but the designation prediction was incorrect, it was seen that many of these cases were caused by an incomplete target affix list. For example, *W513* was extracted as just *W* from the true target designation, and this was successfully predicted by the model as *W*. However, this is only part of the affix, which is much easier to predict than the full affix. The consequence of this is that the designation search will not be performed based on the actual present affixes, leading to an incorrect prediction.
The hypothesis was that fixing this would decrease the affix conversion accuracy while increasing the designation conversion accuracy. Since the affixes will be more specific, it will become harder to predict the right ones. However, with the more specific affix predictions, it should lead to more accurate designation conversion. The results of this modification can be seen in Figure 4.1.
The results are somewhat surprising. The affix accuracy drops as expected; however, the designation accuracy remains almost the same throughout the results. It does seem like the false suggestions drop slightly with adding prefixes, which makes sense since with more specific affixes in the affix list, we are less likely to find something matching all of them. Even though the improvement is questionable, the change will be implemented as we are more accurately representing the affixes in the training data as well as the predictions, so it still makes sense. This also leads to the affix

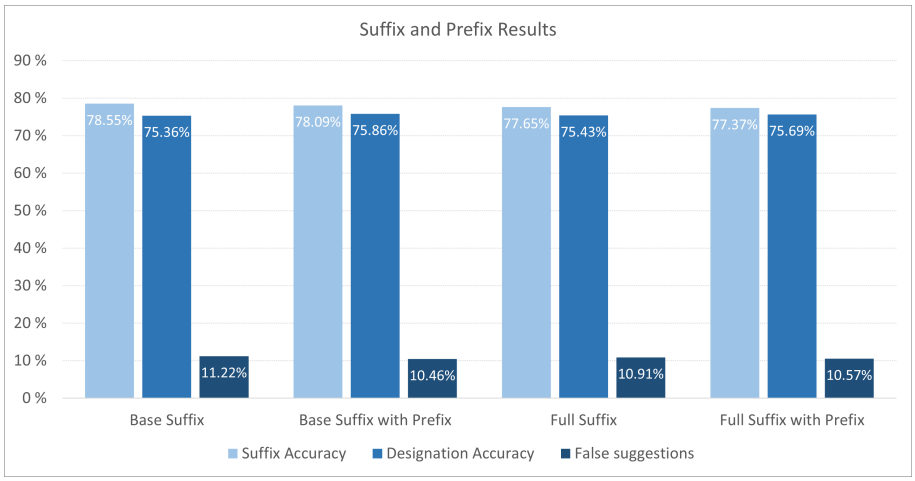

Figure 4.1: Results of suffix and designation accuracy as well as false suggstion rate based on changes to the target affix list

conversion accuracy being more representative since it is comparing to the true affix and not part of it.
Since the affix accuracy can be seen as the upper bound of the performance of the designation accuracy, it is better to have the affix accuracy be more correct. This way, it is easier to see how close to the upper bound we reach and how much is still possible to improve.

4.1.2 Adding Features for Size and Type

If you are familiar with the product ranges, you know that in many cases the suffixes change at a certain size threshold. In many cases, the larger bearings have another design, or another seal, or something else. This threshold could be different between different manufacturers, and some might have the same design throughout the size range. What this means is that the size of the bearing gives valuable information about which affixes to predict.
The same thing can be said about the range and type of product. Usually, the first three numbers tell you the range of bearing. For some ranges, you might have one design while for other ranges it could be another due to technical reasons. In other cases where the designation has a prefix, the first three characters tell you which product type it is or some other information. The conclusion is that the first three characters of the designation carry a lot of valuable information, regardless of exactly what it contains. By adding this as a feature (called f3), we get something that carries a lot of information even though the feature does not represent a single nicely extracted feature of the bearing.
The hypothesis is that improvement in accuracy will be seen by adding these two features, which carry a lot of information that is dense and not shown in the affix list. The results of this can be seen in Figure 4.2.
There is a significant improvement in accuracy when adding the first three characters and when adding the size independently. You get even further improvement when adding both features to the model, indicating that they are not especially correlated.

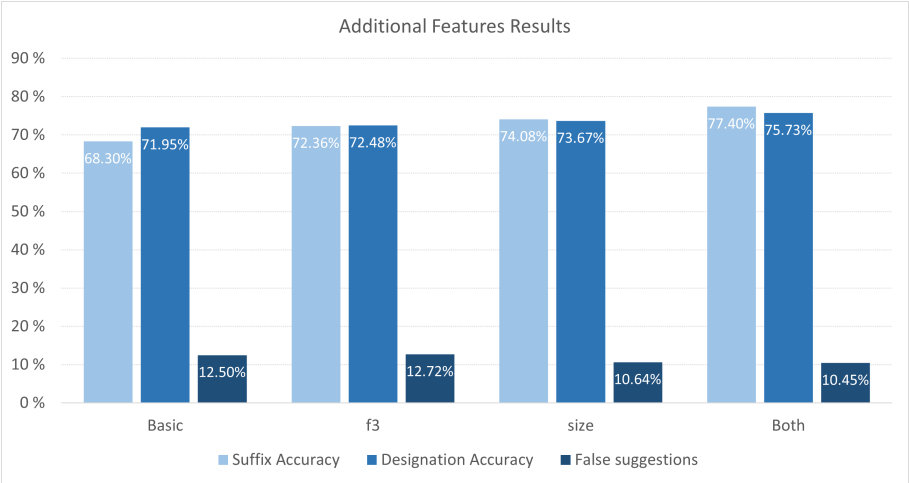

Figure 4.2: Results of adding features for size and first 3 characters(feature is called f3)

The outcome here is clear: the features bring great value and greatly improved accuracy and will therefore be kept in the model.

4.1.3 Training without OEM References

The data contains OEM references that the model, with its limitations, will never be able to predict. The hypothesis is that these references contribute negatively to the training of the model and filtering them out would give better accuracy.
The idea is to filter such references out from the training set but keep the validation set as it is without filtering. With this, we get results that are fully comparable and that are working on the same data. If you were to filter out the references from the validation set as well, you would, of course, get higher accuracy by default.
The results of this trial can be seen in Figure 4.3, where the results of training on a filtered set can be compared to that of a non-filtered training set.
The results show no improvement in the model which was trained on a filtered set of references; actually, a small decrease in performance can be seen. The differences are so minor that they are likely not significant.

4.2 Model Evaluation

To determine the most suitable machine learning model, they need to be evaluated against each other and compared. To compare them we will measure both affix conversion performance, and runtime. In Figure 4.4 the comparison between affix accuracy can be seen.
Here each bar represents a different model and strategy combination. You will see that for example random forest occurs several times, but with different problem transformation strategies. From the results it is clear that random forest outperforms the other methods, using either problem transformation strategy.
In Figure 4.5 you can see the resulting runtime of each of the models and strategies.

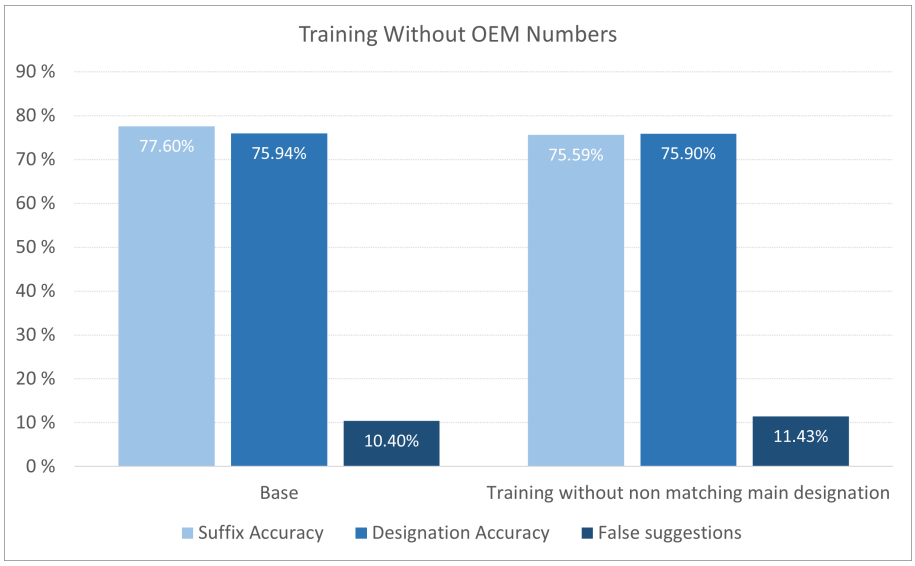

Figure 4.3: Results of training without OEM references

What is seen here is very varying results. Most notable is the difference in runtime for random forest using label powerset compared to binary relevance, especially when referencing Figure 4.4 where the performance difference is negligible. It is also notable how the multi-label methods seem to have a lot higher runtime. Likely due to a poor implementation, rather than the methods themselves, which makes the comparison difficult.

4.3 Data Limitations

To understand what constitutes a good accuracy level, it is essential to understand the underlying data, its trends, and patterns. If there are consistent patterns throughout the data, a very high accuracy level can be expected. On the other hand, if the data is very spread out with little support for many of the classes, great results cannot be expected. Ideally, the model should learn the well-represented relationships, but expectations should be low for those without sufficient support. Several plots have been created to analyse this.
In Figure 4.6, we observe the proportion of the data for each value of the feature f3(first 3 characters), with the values sorted in descending order of proportion.
It is clear that there are many feature values, with a few having significantly higher representation than others. This indicates that references matching these values will have much better support.
Figure 4.7 presents a similar figure to Figure 4.6, but with the feature values of the size attribute.
For the size feature, we observe fewer unique values and a more evenly distributed feature representation, with more sizes being well represented across the data.
Figure 4.8 displays a histogram. The measure is the proportion of references in which the suffix is present, and the histogram represents the number of features within a certain range. This figure shows the competitor affixes.

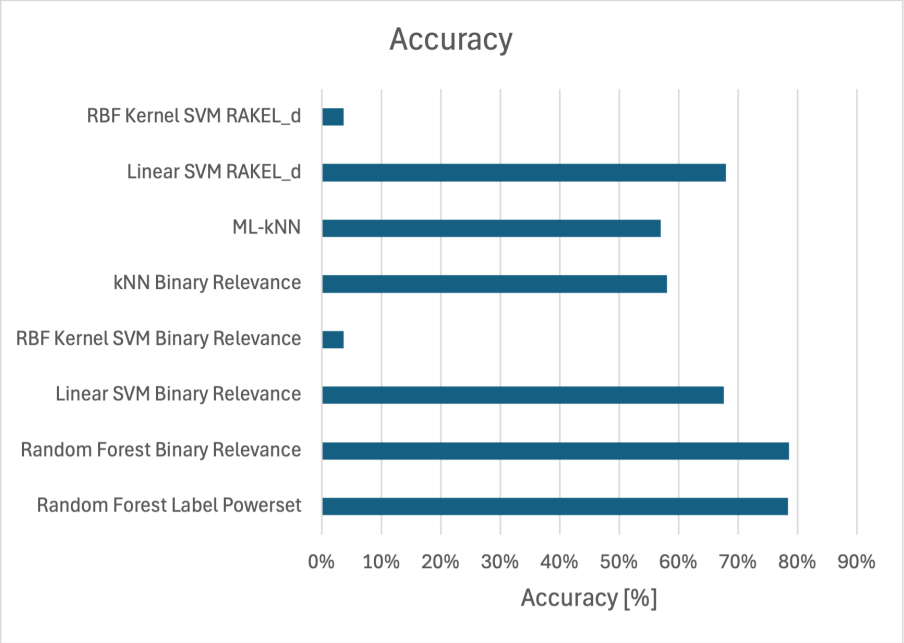

Figure 4.4: Comparison of the accuracy for different models and strategies. Random forest seems to perform the best.

The histogram shows that the data is very diverse, with many affixes having low representation in the number of interchanges. This makes it challenging to learn the patterns of these affixes due to their low recurrence. There is a high likelihood that references in the validation set will have unique suffixes not present in the training set, making it difficult to achieve high accuracy.
Figure 4.9 presents a histogram similar to Figure 4.8, but with the output affixes. This figure also shows that many of the affixes have very limited support in the existing interchanges. From this, we can conclude that the existing database of interchanges contains many references demonstrating the same pattern with the same suffixes. However, there is also a significant number of interchanges where the equivalent product has relatively unique affixes.

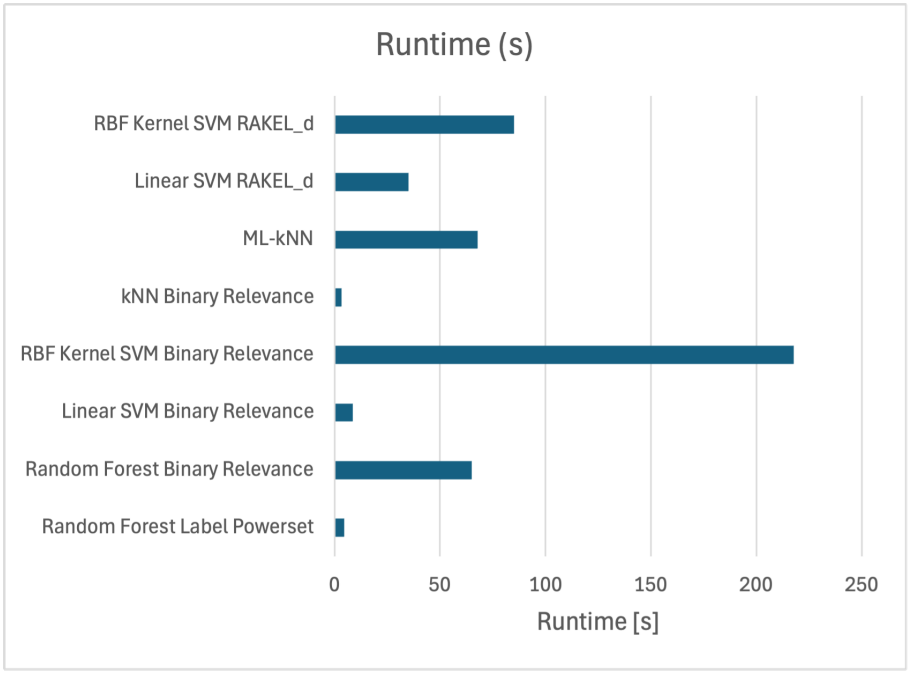

Figure 4.5: Comparison of the runtime for different models and strategies. Random forest has notable differences between label powerset and binary relevance

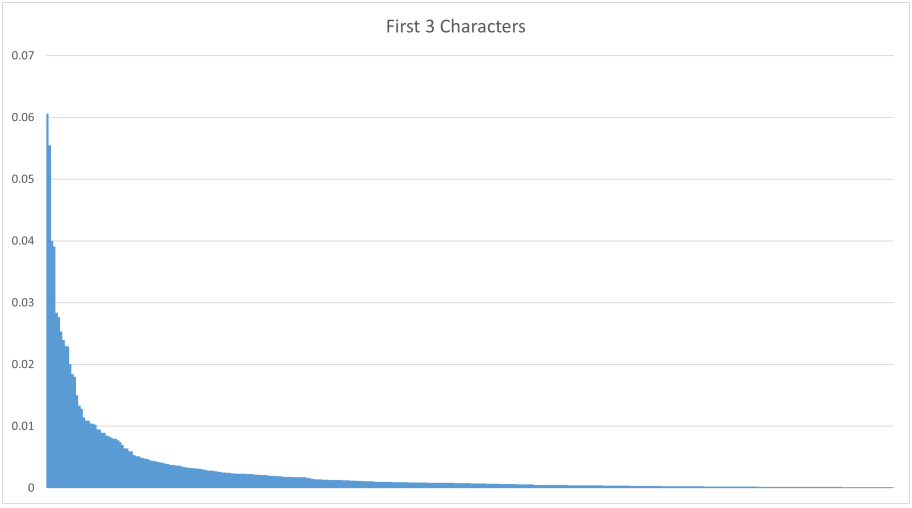

Figure 4.6: Proportion of data for each value of f3

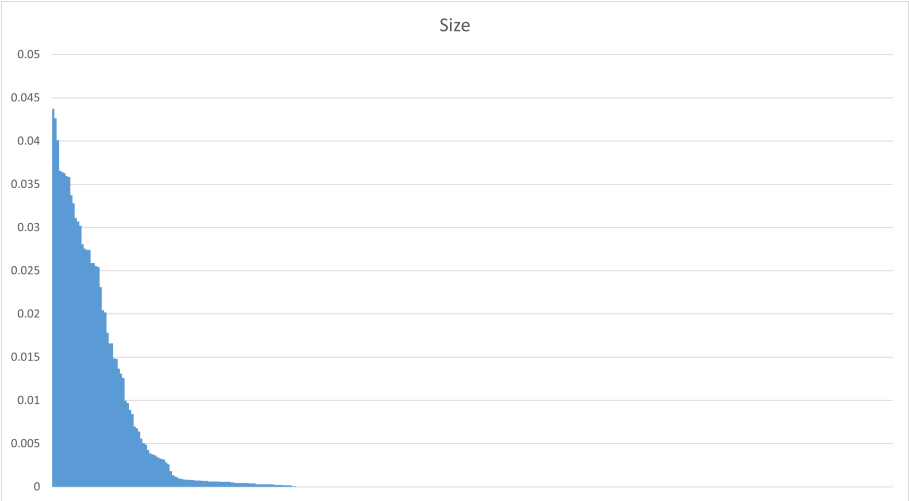

Figure 4.7: Proportion of data for each size value

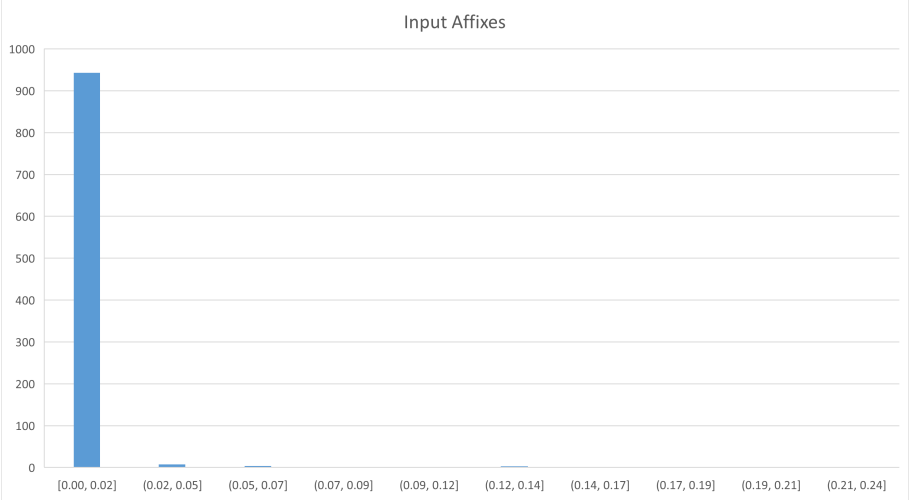

Figure 4.8: Histogram of competitor affixes

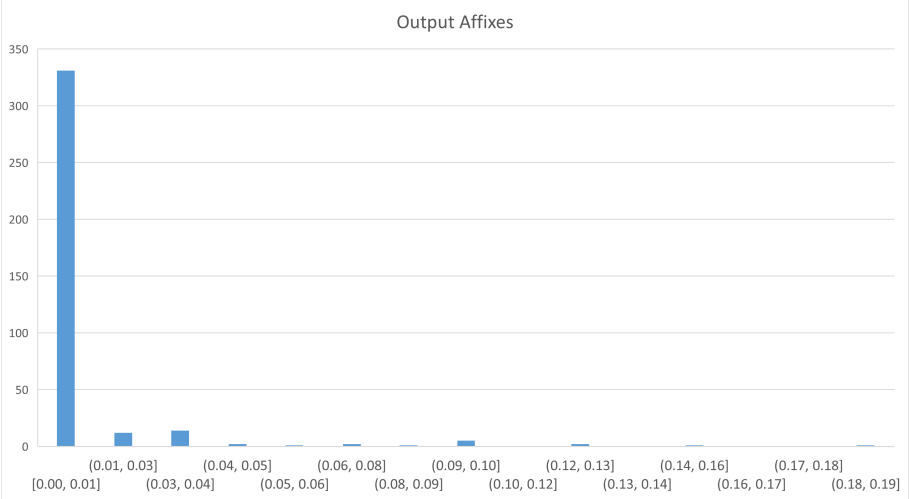

Figure 4.9: Histogram of output affixes

5
Conclusion

With the results presented, they will now be reviewed and evaluated to determine what conclusions we can draw. The results of the pipeline investigations will be judged to determine why improvement are seen and how the final version of the pipeline should look. The results of the machine learning models will be evaluated to determine which model is the most appropriate for predicting cross references using multi label classification. Finally, we will judge the feasibility of using machine learning to predict cross references between products of 2 different manufacturers.
After this, recommendations for what might be explored in the future will be presented. There are several opportunities to build upon the work that has been done, which might lead to either improved performance or more certainty in the solution.

5.1 Pipeline

The pipeline that pre- and post-processes the data is the core of the solution. This is what transforms the problem into a domain suitable for machine learning, and will therefore mainly set the limit to how well an ML model can perform. The main task of the pipeline is to extract features well enough for a model to be able to predict an interchange.
The chosen approach uses label extraction that extracts attributes of a product into a common label pool which then utilizes multi-label classification to predict the corresponding labels of the equivalent product. Multi-label classification is a much more difficult problem compared to multi-class classification.
By simply separating labels based on delimiters the results were already decent, however there was a wish to see if further improvements could be made. Therefore, additional feature extraction was performed to try to extract additional data from the competitor designation. Extraction of the type and size of the bearing was tried, both resulting in significant performance gains, both by themselves, and in combination.
Knowing the domain, this does not come as a surprise. It is known that certain products change their affixes after a certain size, this would be impossible for the model to capture without having this feature as input. The features and affixes also usually change between different series of the same size and bearing type, so it is no surprise either that including information about this boosts the performance significantly.
The list of affixes which was used to generate the target set of labels was found to be incomplete in some cases. To investigate the impact of this, a more complete list was

found and results were compared between using this and the original list. The results were initially a bit surprising, by using the more complete list, the performance of the model decreased. However, this is the performance of the affix conversion, the full end to end accuracy increased slightly. What could be seen was that some of the affix predictions was not on a complete affix but only part of it, therefore the task was greatly simplified, which of course grants better performance. It is likely that the reason for such a small impact in the results is due to very few references being impacted by this incorrect feature extraction. It is also the case that many of the affixes which where mislabelled, also where impossible to predict when correctly labelled, due to extremely low support in the data.
Regardless of performance impact, it was decided to continue with the more complete list of affixes. This makes the feature extraction more correct and gives the performance a more realistic result. This also allows future improvements to be made and evaluated on a more solid foundation. If there where to suddenly be more data around these uncommon affixes, the performance is likely to increase significantly with the complete list, whereas it would not with the original affix list.
A suspicion was that the occurrence of OEM references was causing issues with the performance. These are references pointing from a number without meaning to a designation with affixes and an ISO main designation. However, removing them from the training data had no significant impact on the overall performance of the model. This shows that the model used is robust to noise in the training data and manages to handle this in a good way.
There is a concern that the pipeline itself might be favoring random forest since this was the model used while developing. There is a risk of a kind of overfitting of the pipeline to the model that can happen, especially since we are measuring the improvement of changes in the pipeline with the improved accuracy using random forest. The reason this was done was to be time efficient and not over analyse the changes being made. In reality, the effect of this should be quite low since the addition of features is reasonable regardless of model, and the other changes cover the designation finder, which does not rely on the machine learning model at all. With this in mind the risk of overfitting the pipeline to random forest should be low, but can of course not be ruled out.

5.2 Machine Learning Model

With a developed pipeline, several different models and methods can be tested and evaluated to determine the most suitable one. These tests have been performed and evaluated on the multi-label classification task only, this means that it is only the affix prediction accuracy that is compared.
The model which was used during the development of the pipeline is Random Forest. This model, even when comparing to other options, proved to be highly capable. Whether using the label powerset method or binary relevance, the performance is largely unchanged and outstandingly good.
What is very interesting, but perhaps not so surprising is that the computational performance is significantly better when using label powerset compared to binary relevance. Random forest natively can handle multi class classification, and scales

sublinearly to the number of output classes. This means that it will remain computationally performant even when the target space contains many classes, which explains why it can be so quick in combination with label powerset for multi label classification. On the other hand, when using binary relevance, you do not leverage this scalability and instead train many models on only taking binary decisions. This takes much more time as your training several models instead of one.

What can be deduced from the similar performance between binary relevance and label powerset is that the labels are not correlated. This means that the knowledge that another label is present, does not help with the prediction of other labels.

The reason for random forests great performance is likely due to the non-linearity of its decisions, and the nature of how it does classification. The task of generating an interchange, when done manually, is very similar to a decision tree. Therefore, it is no surprise that a model that leverages and builds up decision trees is suitable for the task.

When thinking of alternative methods for binary relevance, SVM comes to mind immediately. It is a commonly used classifier for binary classification, and since binary relevance transforms the problem into several binary classification problems, it seems suitable. What can be seen when working with this method, however, is that it does not scale well with a large feature space. Since each input label is represented as a binary feature, this results in a very large number of features, which causes the computational performance to degrade. However, this should not impact the prediction performance of the model, which is still lower than that of random forest.

This was somewhat surprising as the decisions for each label should be straightforward. The hypothesis was that a simple classifier would perform very well, and that binary relevance would be a suitable method since there is little correlation between labels. What we can learn from this is that predicting if a suffix is present is not a simple task.

Using a method such as label powerset with a binary classifier such as SVM is not suitable. They scale extremely poorly with increasing number of classes, as many separate classifiers needs to be trained. This is due to only being able to handle binary classification and needing to use a strategy such as one-vs-all to transform the multi class classification into a single class classification task.

This leads us into the method RAKEL. The aim of this is to get the benefits of label powerset, but without the performance impact, and the promise is equal or better performance than label powerset. Unfortunately the overlapping labelsets was not able to be tested due to issues with the implementation. The overlapping labelsets is supposed to perform better than the disjoint sets. However, we can see in the results that $RAKEL_d$ grants similar performance to binary relevance. This is likely due to this application not benefiting much from label powerset, since the labels are not correlated.

Another classifier which is commonly used is kNN, this was used in combination with binary relevance. However, the results were not notably impressive. The main reason for trying this method was to make the comparison to the adapted version: Multi Label k Nearest Neighbours. This is a method which is based around the standard kNN algorithm, but that has been adapted to be able to make multi label

predictions. This version performs similarly to using kNN with binary relevance. The method is based around using Bayesian inference and the information from an instance's neighbours to predict which of the labels are most likely to occur in the instance. Both showed to have an optimum at $k = 3$.
This method had high expectations on it since being close in the feature space really should correspond to being close in the output space. The model's task is to map the product from one space to another, where similarity in one should equal similarity in the other. So, it is surprising that the performance is so relatively poor. You would expect that by looking at the affixes of a product's closest neighbours, you would be able to use the same for the instance. The issues likely come from scaling, since the input space is largely made up of binary features, then all of them have the same importance. In reality, some differences should result in larger or smaller distances. However, the proper scaling is hard to identify and is therefore left out in this examination.
My hypothesis is that the method struggles due to the neighbours not brining enough information, and that the model would require much more datapoints to make the right prediction. The classification is complicated enough that an instance nearest neighbours are not representative enough of the relationship. The relevant pattern might only be observable in instances which are far away, which causes poor performance. The model is likely to perform much better with a larger and more condensed dataset, but, as has been shown, the data is extremely spread out with low support for a large part of the labels.
Another method that was tried out, and which the research was initially centered around is Neural Networks. The initial hypothesis was that such a network could learn the complicated relationships and accurately estimate the probability of each label. However, it was quickly realized that the training data was far to limited and had to little support for a model like this to work. The performance is still respectable compared to some of the other options, but this approach does not suit this use case where limited training data will always be a fact, and large diversity without much support will always occur. The model is however a very capable multi-label classifier and can handle this task natively. Under different circumstances with a lot more training data, and more support for each affix, it would likely excel in its performance.

5.3 Cross Reference Prediction Feasability

When determining if the method is feasible for predicting interchanges, we must first determine what a good score is. The answer is of course always: it depends.
The patterns in this data are not always logical. All the references have been created manually by different people. This means that their individual preferences and knowledge impact what interchanges they make. In addition to this, it is also usually based on a customer request, where other unknown factors could lead to choosing one product over another. Another factor is the availability of products, the product lineup is not always so consistent since they are based around the actual demand. While something could be the theoretical equivalent, it might not exist for that specific size or series of bearing. This causes inconsistencies in the training

data which makes it hard for a model to perform well.
When determining if a prediction is correct, we must of course compare with the true label from the input data. There are however cases where the prediction made is correct, even though it is not the same as the true label. The extent of this is of course difficult to determine, since it would require an expert evaluating each prediction.
Since the aim for the model is not to make these predictions straight to a customer, the need for accuracy is not so high. Higher accuracy is of course always best, but the goal is for the model to be a tool for creating new references. Having a prediction which is correct in many cases saves a lot of time compared to doing the work by yourself. It is also usually easy to see if something is incorrect and then those cases need to be handled manually.
If the model can handle 75% of the cases correctly, then all the time that is saved can be put towards identifying the interchanges where the model fails. The overall time saved compared to doing it manually would still be big.
Since identifying incorrect predictions from the model is relatively easy and quick, the time saved will always be large compared to doing the work manually. Because of this any decent accuracy would still make the model useful. Additionally, since the goal is always for the prediction to be verified, the accuracy does not necessarily need to be that high.
With this in mind, it is determined that leveraging machine learning models as a tool for creating new interchanges is feasible.

5.4 Recommendations for Future Work

There are several ways you could go about continued work on this topic. While it is definitely possible to continue the exploration of machine learning models, this is not likely to give much improvement to the results. As we have seen the models are extremely limited with regards to the data that is available, it could be that in the future, with a larger dataset, that the method would automatically work even better.
On the other hand, where there are gains to be made is in the feature extraction. If features where to be extracted in a better way, it is very possible that much better results could be achieved. Due to the difficulty in feature extraction, this project has revolved around multi label classification. If an approach could be determined which separates the affixes, and thereby attributes and features of the bearing, into separate features, this would enable a very different type of learning.
As an example, if it was possible to identify all different affixes that describe the seal, and to extract this into one common feature, you would then have very different conditions to make your classification. In this case, instead of predicting each of the possible seal suffixes individually, you have a single multi class classification task.
You would then train separate multi class classifiers on each of the different affix types to make predictions. By taking the union of all their outputs you would get your final prediction of affixes.
This would enable a simpler classification task, as you only need multi class classification. It would also assist the model in understanding the relationships between

different affixes, as there are currently many relationships which are not modelled in the data.
The results would of course need to be validated in practice, but the hypothesis is that significant gains would be made if this was possible. The main difficulty would lie in how to extract such features. It is one thing to do it on the target affixes, which are fixed, but very different to do it on an unknown set of affixes from a competitor. It could be possible that such categorization and classification could be done using a model, but it might also be a labelling task that needs to be performed by a human.
Antoher topic one might investigate is the generalization of the method to other competitors. While it is obvious that it will work with manufacturers with similar delimitation of their designations, it will not work if the competitor writes their affixes without delimitors. There could be ways to solve this using similar methods as has been done for the target designation. There could also be possibilities to explore the training of a model to decipher the competitor designations into features. If the methodology is to remain as is, then a manual parser will need to be written for each competitor, but given this, the rest of the pipeline should work without further modification.

Bibliography

[1] GeeksforGeeks Contributors. One hot encoding in machine learning. https://www.geeksforgeeks.org/ml-one-hot-encoding/, n.d. Accessed: 2025-05-05.

[2] SKF Group. Basic bearing designation system. https://www.skf.com/group/products/rolling-bearings/principles-of-rolling-bearing-selection/general-bearing-knowledge/bearing-basics/basic-bearing-designation-system. Accessed: 2025-04-28.

[3] IBM Corporation. What is the k-nearest neighbors algorithm? https://www.ibm.com/think/topics/knn, n.d. Accessed: 2025-05-05.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] 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 Édouard Duchesnay. *sklearn.preprocessing.MultiLabelBinarizer*, 2011.

[6] P. Szymański and T. Kajdanowicz. A scikit-based Python environment for performing multi-label classification. *ArXiv e-prints*, February 2017.

[7] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089, 2011.

[8] Wikipedia contributors. Multi-label classification — Wikipedia, the free encyclopedia, 2025. [Online; accessed 5-May-2025].

[9] Wikipedia contributors. Neural network (machine learning) — Wikipedia, the free encyclopedia, 2025. [Online; accessed 5-May-2025].

[10] Wikipedia contributors. Random forest — Wikipedia, the free encyclopedia, 2025. [Online; accessed 5-May-2025].

[11] Wikipedia contributors. Support vector machine — Wikipedia, the free encyclopedia, 2025. [Online; accessed 5-May-2025].

[12] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

DEPARTMENT OF MATHEMATICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY