Master's Thesis

Text Classification of a Small and Imbalanced Dataset With Long Texts

MSc Artificial Intelligence
Graduate School of Natural Sciences
Utrecht University

By
Isa Apallius de Vos

student number: 5268664

July 14, 2022

Universiteit Utrecht

Movares

Submitted in partial fulfilment of the requirements for the degree of M. Sc.
First supervisor: Dr. Pablo Mosteiro Romero (Utrecht University)
Second supervisor: Dr. Meaghan Fowlie (Utrecht University)
Company supervisor: Jeroen van Boekel (Movares)

Abstract

Text classification describes the process of categorising documents into groups based on certain features in their content. Previous research on this topic has focused on testing how specific data attributes such as data size, class distribution, or document length influence the performance of certain types of classifier, but no research showed any findings on classifier performance if a dataset had many of these attributes at once. This thesis thus focuses on getting more insight into how using a dataset with multiple limiting attributes influences different classification models, and which type of model would work best on a dataset with multiple limiting attributes. To do this, a multi-label dataset of 403 labelled Dutch letters of objection with 24 different labels was created, after which three different simple models (Decision Tree, Naive Bayes, and SVM) and a pre-trained language model (BERTje) were trained on the dataset. After testing the models and comparing micro and macro F1-scores, it was found that the language model could not outperform the simpler models on the text classification task. The language of the texts and the class distribution in the dataset were not shown to greatly influence the models' performance, whereas the small data size was found to be the main data attribute limiting the performance of all models. Interestingly, some classifiers could obtain high F1-scores for some of the very small classes in the dataset, indicating that the documents do contain information on those subjects that could easily be extracted by a model if the right technique was used. It is thus proposed to further test the models on individual classes and to inspect a different, rule-based approach to classification to see whether model performance can be improved on this classification task.

Contents

1 Introduction

Given the ever-growing collection of data available to us, it can be a very time-consuming process to turn data into useful information. To make this problem more manageable, techniques have been developed that use statistics and artificial intelligence to automatically process and analyse large collections of data in order to get information out of them. This process, also known as data mining, can be used for many different types of data, including collections of texts. When working with language data, text mining is used, which is a process that uses natural language processing (NLP) to extract useful information from collections of unstructured texts. There are many NLP problems related to text mining, like sentiment analysis, text summarisation, and text segmentation. This thesis will focus on one specific problem, namely text classification.

1.1 Problem description

In text classification, the task is to analyse a collection of documents and to assign one or multiple categories or labels to each document in the collection based on its content. Text classification can be applied to a wide variety of document types and can be used for many different applications. To give a simple example, emails can be classified as either being spam or not based on their content. Many different approaches to text classification exist, ranging from simple rule-based classifiers to large and complicated neural models. Because each dataset of classifiable texts is different, the approaches that can be used to solve the classification task might be different as well. For example, simple models are explainable and can be trained quickly, but they can overfit quickly as well and cannot always handle small datasets without data or model enhancements, whereas more advanced classification techniques can find more complex patterns in the data and can sometimes learn patterns in the dataset even if less data is provided, while factors such as document language and document length can limit the amount of information that can be used as input for that type of classification model.

Although an attribute like having a small dataset impedes all types of models, the question remains which type of model could work best on a dataset that has multiple influencing attributes such as having multiple classes per document and an imbalanced class distribution, having documents in a language that has less resources than English, and having longer documents. For this type of situation, it can be insightful to see how different properties of a dataset influence the performance of different models, and whether one should decide to use simpler models with modifications to mitigate problems related to data size and distribution, or to use a more advanced classification technique that is modified to handle certain complex types of data. This thesis aims to give more insight into *the different aspects of a dataset that influence the results of different types of classifiers, and how the choice for a classification approach can be made given a set of data attributes*. These insights could be used in the future to determine whether, for example, enhancement of the data can be a remedy for simple models performing poorly on their classification task, or whether more advanced language processing techniques should be used to improve the performance on the classification task at hand. These questions will be studied with a case study, in which a dataset of texts with multiple limitations is presented and needs to be classified.

1.2 Case description

Although it is the case that processes that can be done easily by a machine or algorithm are still often done by hand, the number of companies pursuing automation is currently increasing (McKinsey, 2020). In the context of language processing and texts, automation is relevant when many documents that can easily be analysed and processed by a computer are being read by people. Although humans are generally seen as being good at understanding texts and extracting relevant information from them, the question is whether machines can do it more quickly but slightly worse, at least as well, or even better. Earlier research already found that machines and humans can both perform well in finding relevant parts of texts in indexing tasks (Anderson and Pérez-Carballo, 2001), and more recent research has even shown that some language models can outperform humans on different tasks related to language understanding (Nangia and Bowman, 2019). Computers then have an advantage over humans since they can process texts more quickly, with state-of-the-art models being able to handle 250 sentences per second (Sapunov, 2019), whereas the average human can read approximately 250 words per minute (Rayner et al., 2016). This shows that the automation of text processing can speed up a task that can be quite time-consuming for humans, without a large decrease in accuracy. This can be very useful for different fields that need to process large collections of longer texts, with two examples being the legal field (Wan et al., 2019) and the medical field (Su et al., 2021). Because of the need to automate this process, a specific automation case can be studied to see what can be learned from that for future cases. This thesis will thus focus on a specific case, provided by a company called Movares, in which a collection of texts that is now classified by hand needs to be classified automatically.

Movares is a Dutch consultancy and engineering company which, among other things, is involved with infrastructural projects related to highways and railways. Before the plans for a project are executed, a certain time-frame is given in which stakeholders, such as local residents and municipalities, can send in their objections and questions about the project in a letter of objection. The different points of objection made in such a letter can generally be assigned a certain category; for example, a point of objection can concern the impact of the planned project on the environment, the noise disturbance caused by the project, or the influence that the project will have on the landscape. After the submission period is over, the advisers and experts that work on the project receive the letters of objection, after which they are tasked with reading each letter, labelling which types of points of objection are made in the letter, and then responding to each of the objection letters. Since a project can receive anywhere from fifty to hundreds of letters of objection, this process of reading, labelling, and responding to letters can be relatively time-consuming work. Thus, the company is interested in finding a way to (partially) automate this process. The first step in the automation process is to find for each letter which types of points of objection are made in it, which will be the focus of this thesis. Implementing this first step would still result in people needing to read the points of objection in the letters. However, the labelling of the letters can serve as an indication for the project experts whether the letter can be answered with a standard response or whether it will need more thorough investigation. Additionally, the labelling can already indicate which expert is needed for answering a particular letter, which would omit the necessity to first read the letter to discover that. In both of these cases, some time is already saved by the implementation of the partial automation.

The problem described by the company can be framed as a multi-label text classification problem; given a collection of documents and a set of 24 category labels, all the category labels that are relevant need to be assigned to each document. Since Movares provided annotated letters from different projects they have worked on in the past, it was decided that a supervised learning approach for the text classification would be used. Many different approaches to supervised text classification exist, and the choices that need to be made about which algorithm is best suited for such a project depend on the shape and size of the provided data. The dataset provided for this project has a number of attributes that will influence the chance of success of the possible approaches, namely the small size of the dataset, the imbalanced frequency distribution of the category labels, the great length of some of the letters of objection, and the fact that the letters are in Dutch. This thesis will thus focus on finding out how these factors influence the model choice, and which techniques can be used to mitigate the different dataset limitations.

1.3 Research question

Given the problem description, the main research question in this thesis is defined as follows:

How can effective text classification be achieved for small and imbalanced datasets with long texts?

To find an answer to this question, the following sub-questions need to be answered:

- Which aspects of a dataset limit the performance of simple classification models, and how can these limitations be mitigated?
- Which aspects of a dataset limit the performance of classifiers based on language models, and how can these limitations be mitigated?
- To what extent does a classification approach using a language model yield better results than a simple text classification approach, given a dataset with different limitations influencing the performance of both classification types?
- To what extent do certain data aspects have a larger influence on the model performance than others, and can this be explained by inspecting the data and the functioning of the models?

The first two sub-questions are studied in the literature review, after which the third and fourth sub-questions are answered by implementing and comparing both types of classifiers and testing how the different aspects of the data influence their performance.

1.4 Thesis outline

The thesis will be structured as follows: in Chapter 2, the dataset used in this thesis will be discussed in more detail, and Chapter 3 will review the previous literature related to the thesis subject. Chapter 4 will describe the choices made for the implementation of the project, after which Chapter 5 will more concretely explain how the experiments to answer the research questions were executed. In Chapter 6, the experimental results will be presented, analysed, and discussed. Finally, Chapter 7 will conclude the thesis by answering the research questions, presenting the limitations of the research, and discussing possible future work.

2 Dataset

The dataset used in this thesis consists of a collection of letters of objection from different projects Movares was involved in. Firstly, in Section 2.1, some information about the origin of the letters will be given. After that, Section 2.2 will describe how the letters were collected and then Section 2.3 will explain how the data was prepared for the classification task. Finally, in Section 2.4, the size and class distribution of the dataset will be discussed.

2.1 Data background

The letters of objection in the dataset originate from three different projects in which Movares has been involved. The first project is related to the A4, a 125 kilometre long Dutch highway which connects some of the largest cities in the Netherlands. Because certain parts of this highway often cause traffic jams, it is important for the flow of the traffic and the accessibility of the cities that some changes are made to the highway. The plans for this project involve adding more lanes to a section of this highway, as well as changing the intersections of the state highway N14, connected to the A4, to grade-separated intersections. The second project concerns another Dutch highway, the A27. Similar to the other highway project, in this project the A27 is widened because there are now often traffic jams due to the large amounts of traffic passing this highway. Finally, the last project is related to the Maaslijn, a single-track railway line between Nijmegen and Roermond, two Dutch cities. A number of changes are going to be made to this railway line. Firstly, the railway will be electrified, meaning that the current diesel trains will be replaced by electric trains. Secondly, additional tracks will be constructed at four different parts of the railway line in order to make it easier for oncoming trains to pass each other. Finally, measures are being taken at various level crossings to improve level crossing safety.

2.2 Data acquisition

As was explained in the Introduction, anyone who has any objections or questions about a certain project can send in their objections in a ‘letter of objection’ during a given period of time, which is normally around six weeks. This can be done in any format, meaning that people are allowed to send in handwritten letters, printed letters, emails, online forms, or any other means of correspondence. However, the majority of the documents in the dataset are scans of typed letters, with handwritten letters being the exception. Because the complete collection of letters of objection needs to be available digitally for the experts on the project to analyse and reply to them, the physical letters are all scanned in, whereas the digital letters are sent directly, in most cases as a PDF or Word file. Because there are no regulations for the formatting of the letters of objection, things such as the length of the letters of objection and the structuring of the pages can vary greatly, with some letters being one page long and stating each point of objection in a single sentence, and other letters being ten pages long and using full paragraphs to explain each of the objection points thoroughly. After receiving the letters, the annotation of the given documents is done by hand by the project experts. This involves reading the letters of objection, determining which points of objection are made in the letters, extracting the texts of those separate points of objection, and labelling those points of objection with the relevant category label. This process results in a file containing, among other things, a unique ID for each letter of objection, the extracted texts of each point of objection, and a category label for each point of objection. This file can then be used to determine which expert should answer which point of objection, and to answer each letter of objection systematically.

2.3 Data preparation

Before the provided data could be used for the classification task, a number of changes needed to be made. These changes involved remapping the old category labels to new ones, matching the letters of objection to their category labels, and removing duplicate letters from the dataset.

2.3.1 Remapping category labels

As was explained, for each project, the points of objection in a letter of objection were assigned a category label describing its topic. One initial problem with the labelling of the documents was that the category labels for the three given projects differed slightly, and that some labels in the highway related projects were not used in the railway related project and vice versa. After some consultation with the project experts, it was decided by them that one general set of labels should be created for all of their future projects, and that the labels used in the three given projects should be mapped onto this new label set in order to create one dataset with clear labels. The Dutch names of the labels can be found in Table 1, as well as their English translations. These 26 categories were selected by the experts and encompass every type of objection that they have seen in previous projects and expect to see in future projects. It should be noted that only 24 of these classes are discussed in the following sections of the thesis because the labels *OO* (Explosive remnants of war) and *Overig* (Other) did not occur in the given dataset.

The mapping of the old labels to the set of new labels was relatively straight-forward for most of the labels in the projects because the old labels often only needed to be renamed or were mapped to a more general label. However, in some cases, an old label was split up into two separate new labels, which meant that the points of objection with those old labels needed to be checked manually to see to which new label they would map. These more complicated mappings were assigned and then verified with one of the project experts to ensure that the new mappings were done correctly.

2.3.2 Matching texts to category labels

Because this project focuses on classifying whole letters of objection based on their text, the input for the classifiers needs to be the text of a letter, as well as all the category labels associated with that text. Although the points of objection in the provided projects were saved with their corresponding labels, the original texts of the letters of objection were saved separately. This meant that one of the data preparation steps consisted of matching the original texts with their corresponding labels. For the Maaslijn project, the

Table 1: An overview of the categories in the dataset and their English translation.

Dutch category name	**English translation**
Geluid	Noise
Ontwerp	Design
Planschade en grondverwerving	Plan damage and land acquisition (Plan damage)
Luchtkwaliteit	Air quality
Verkeer en vervoer	Traffic and transport (Traffic)
Uitvoering	Execution
Participatie	Participation
Landschap, inpassing, en ruimtelijke kwaliteit	Landscape, integration, and spatial quality (Landscape)
Natuur	Nature
Trillingen	Vibrations
Water	Water
Besluit-plan	Decision-plan
Projectdoel	Project goal
MER	Environmental impact report (EIR)
Raakvlakprojecten en meekoppelkansen	Interface projects and opportunities (Interface projects)
Overwegen	Level crossings
Externe veiligheid	External safety
Archeologie en cultuurhistorie	Archaeology and cultural history (Archaeology)
Gezondheid	Health
Klimaat en duurzaamheid	Climate and sustainability (Climate)
Smart mobility en innovaties	Smart mobility and innovations (Smart mobility)
Ruimtegebruik	Use of space
Kabels en leidingen	Cables and pipes
Bodem	Soil
OO (Ontplofbare oorlogsresten)	Explosive remnants of war
Overig	Other

letters of objection were provided in the same format as when they were delivered to the project experts, meaning that the provided data was a collection of PDF and Word files. These files thus needed to be converted to texts, such that those texts could be used for classification. The conversion from PDF to text was done using the pytesseract Python library, while the conversion from Word document to text was done using the docx2txt Python library. Because the file containing the points of objection and their labels also provided the name of the original letter from which the points of objection originated, it was possible to then match the letters to their labels. For the A4 project, a similar process of matching the letters to their labels was possible. However, because the texts of the original letters of objection for the A4 project were already saved in a CSV file, it was not necessary to extract the texts from the letters. For the A27 project, the original letters of objection were available but could not be matched to their points of objection due to a mismatch between the file names of the letters and the names provided in the document with the points of objection. However, since the texts of the points of objection from each letter were still available, it was decided that the objection texts belonging to a single letter of objection would be appended to recreate the original A27 letters.

2.3.3 Removing duplicate letters

After collecting the texts for the letters of the different projects, the final preparation step was to remove very similar or even duplicate letters from the data. These types of letters appeared in the data because it sometimes occurs that multiple people who have the same points of objection use a letter template to each send a letter separately. In this case, either the same letter is sent multiple times, or the content of the letter is changed slightly for each individual to match their particular problem with the project. It should be noted that in the case that a template was used by different people to make different types of points of objection, the letters were not seen as duplicates or similar and were therefore not removed. It is essential in any task in which a model is trained and tested on a dataset that the data in the training set does not overlap with the data in the test set, because the model is tested on *unseen* data to see whether it can generalise what is has learned during training. If the same letter or very similar letters occur both in the train and test set, it cannot be guaranteed that the model's performance is caused by how well it learned the features in the data. Additionally, the inclusion of very similar letters with the same point of objection does not add any information to the model. Because of this, it was necessary to remove duplicate letters from each project. This was done by automatically detecting similar files in the data with a python script and manually checking whether a whole letter should be removed or whether parts of it could be used if the letters were partially similar. For calculating the similarity between two documents, a function from the *difflib* library called *SequenceMatcher* was used, which compares two strings and outputs a similarity score. Since two documents with completely different contents would have similarity scores under 0.05, a threshold of 0.35 was used to find documents that had enough in common to check their contents manually. If documents had a similarity score higher than 0.7, in almost all cases after manual inspection they were found to be so similar that one of the two letters could be discarded because it would not add any information to the dataset. For documents with a similarity score between 0.35 and 0.7, it was often the case that certain sections about specific points of objection were the same while the rest of the documents differed. In that case, only the overlapping section would be removed from one of the documents. If letters or sections were similar but were not about the same topic, they were not removed.

2.4 Data size and class distribution

After executing the different steps for the preparation of the data for the classification process, 403 labelled documents remained. Since these documents originated from three different projects, the number of letters per project can be found Table 2.

Table 2: The number of letters in the dataset per infrastructural project.

Project	Number of letters
A27	213
A4	148
Maaslijn	42
Total	403

The mean number of labels per document is 2.7, and the mean number of words per document is 803 words. However, the median number of words per document is lower than the mean number of words, which can be explained by looking at Figure 1. Although there are documents with more than 5000 words in them, more than half of the documents have less than 500 words.

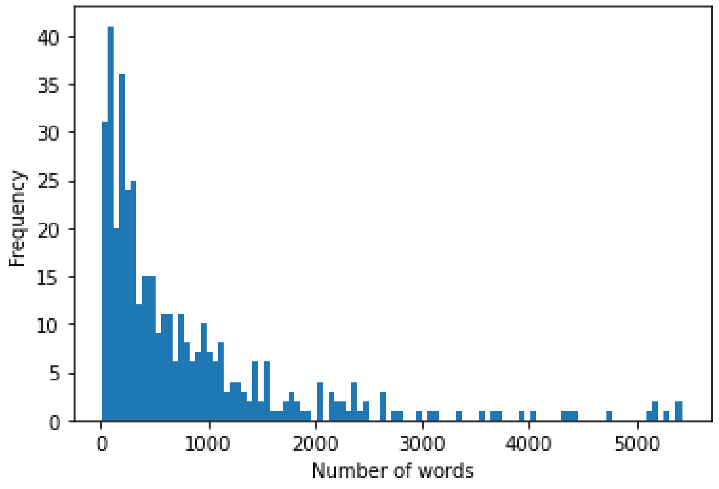


Figure 1: The frequency of the number of words per document.

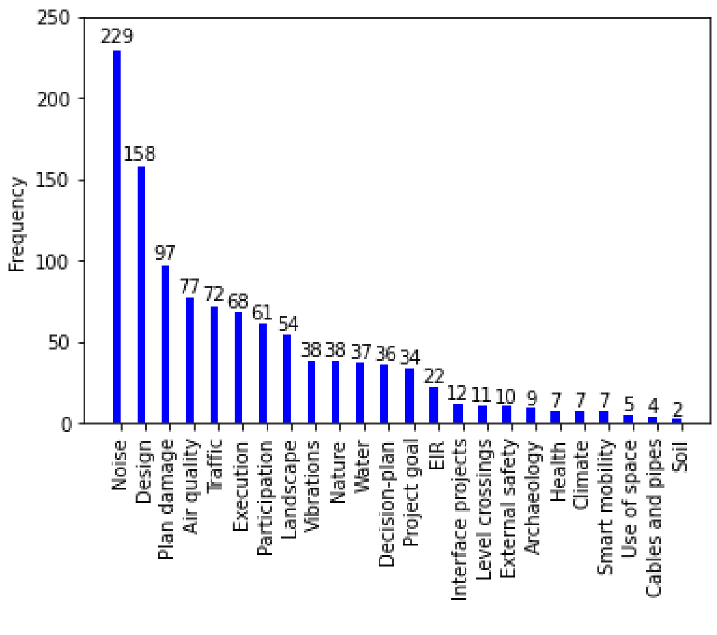


Figure 2: An overview of the frequency of each unique category label in the dataset.

Figure 2 shows in how many documents each unique label in the dataset occurs, where it can be seen that the most common label, *Geluid* (Noise), is found in over 50% of the documents. It can also be seen that the distribution of the classes is imbalanced since the number of samples per class differs quite a bit, and that two thirds of the classes have less than 40 samples. It should be noted here that since the dataset is multi-label, the sum of the frequencies of each class does not match the total number of documents in the data because documents can contain multiple classes at once.

3 Literature review

This thesis focuses on answering the questions of how text classification can successfully be implemented for a dataset with multiple limitations, how these limitations influence the different classification approaches, and how they can be overcome. In this chapter, the literature relevant for answering those questions will be discussed and reviewed. In Section 3.1, more general techniques for text classification will be discussed, after which Sections 3.2 and 3.3 will focus on what is known about supervised text classification using a simple classification approach and a language model based approach respectively. Section 3.4 will discuss different model validation and evaluation methods, and finally, Section 3.5 will summarise the findings described in this chapter.

3.1 Text classification and clustering methods

Because the problem of assigning labels to documents based on their content is a language problem that can be solved with different techniques, it will first be discussed more generally which approaches have been developed in the past.

To start off, a technique that was already popular in the '80s was the rule-based approach to text classification. For this approach, an elaborate collection of logical if-then rules is crafted to classify texts into categories based on the words that occur in them. To give an example, if a system needs to be built for classifying subjects of news articles, a rule could be that if a text contains the term 'football', 'team', or 'score', it should be classified into the category 'sports'. An interesting example of a rule-based system from the '90s is the Construe system by Hayes and Weinstein (1990), which was built to categorise news stories for the Reuters news agency. The rule-based approach can be quite effective if the rules used in the system are well-crafted and extensive, but there are some major drawbacks for this approach. As explained by Sebastiani (2002), the main problems are that both a knowledge engineer and an expert in the field are needed to handcraft each of the rules, and that if there are any changes to the set of categories, the rules need to be updated and tested to ensure that the system still works. More recent research has also focused on combining a rule-based approach with a probabilistic classification approach by using both associative classification rules and a Naive Bayes classifier to classify texts, which resulted in a better classification accuracy compared to purely rule-based algorithms and compared to only using Naive Bayes (Hadi et al., 2018). However, the performance was not compared to supervised classification techniques besides Naive Bayes, which means it is uncertain how much better this hybrid approach works compared to other non-rule-based approaches.

Another widely used approach for assigning categories to texts is topic modelling, which is an unsupervised learning approach that can be used on a collection of unlabelled documents to find latent topics. The idea behind it is that clusters of documents can be formed based on the words that occur in them, and that each cluster represents a topic that is defined by a list of found words. One of the most widely used techniques for topic modelling is Latent Dirichlet Allocation (LDA), which was developed by Blei et al. (2003). The LDA algorithm can be applied to many different problems, such as discovering the main subject of unlabelled research papers (Suominen and Toivanen, 2016), or finding topics in privacy policies (Sarne et al., 2019). There has also been research on how to implement topic modelling using LDA for datasets that grow over time (Canini et al., 2009), and how to effectively evaluate the topic modelling results (Nikolenko et al., 2017). Although the topic modelling approach can be effective for finding latent topics in texts, it is a less obvious choice when a dataset with labelled data is provided.

Finally, in the supervised approach to text classification, labelled documents and supervised learning are used to train a classifier to assign one or multiple category labels to a document. This is the approach that will be discussed more in depth in this thesis. Supervised text classification is generally done in two steps, where the first step consists of preprocessing the data and calculating a feature vector for each of the documents in the dataset, and the second step consists of using those vectors as input for a classifier and training that classifier on the labelled documents. This process can be done in different ways, but this thesis will focus on two main approaches: a simpler approach using word frequency based document vectors and simple machine learning classifiers, and a deep learning approach using a pre-trained language model.

3.2 Supervised text classification: baseline approach

The first approach that will be discussed is the simpler approach to text classification, in which a feature vector is calculated for each document, after which a classifier is trained and tested on the vectors. The first step, creating feature vectors for the texts in the dataset, is an essential step, because a classification algorithm cannot work with raw texts and needs a numerical input in order to learn the relation between the content of the documents and their assigned categories. There are a number of techniques to calculate the vectors of documents, but the simplest type is discussed here: calculating feature vectors based on word frequency counts.

3.2.1 Generating feature vectors

Bag-of-words
One of the most classic and widely used techniques for representing documents with feature vectors is the bag-of-words (BoW) approach. In this approach, the idea is to look at the complete set of words occurring in the available collection of documents, to make a list of every unique word in that collection, and to count for each document how often each unique word occurs in that document. In the BoW approach, it is assumed that the order of the words in the text is not relevant for the task at hand. When more information about word relations needs to be captured, it is also possible to count the frequency of sequences of words, called n-grams, where n stands for the number of words that are grouped together. Because the BoW approach often results in very sparse vectors with many zeroes for all the words that do not occur in a text, different methods like lemmatisation and stopword removal can be used to reduce the size of the vectors.

Tf-idf
Alternatively, a reweighting approach can be used for the BoW by not only considering

the frequency of the words but also the number of documents that the words occur in. The idea is that words that occur very frequently in a text are only interesting for the meaning of the text if the words do not also occur in every other document in the collection. This intuition is captured by the tf-idf statistic, which is defined as the term frequency (tf) multiplied by the inverse document frequency (idf). Here, the term frequency is the number of times a word occurs in a text, while the inverse document frequency is the inverse of the number of documents in which a word occurs. Often a log function is used over both the tf and idf to reduce the difference between the higher and lower frequencies (Jurafsky and Martin, 2009). Using the tf-idf scores for the vectorisation of documents results in a set of vectors similar to the BoW vectors, but with tf-idf scores instead of word counts. Because the tf-idf score captures a bit more information than the BoW approach without costing much more time or memory, this technique will be explored more in this thesis.

Review of techniques
Although relatively simple, the BoW and tf-idf are still widely used as part of the text classification process. Research topics about text classification using BoW and tf-idf for vectorisation include testing different modifications of tf-idf to better manage issues like uneven document lengths or class distributions (Roul et al., 2017), and comparing the performance of a classic BoW approach paired with a classifier to the classification of texts based on the fastText algorithm (Walkowiak et al., 2018) or to classification based on a word embedding technique paired with an LSTM (Semberecki and Maciejewski, 2017). These last two studies show that these simpler approaches can be used as a baseline to compare newer models to.

3.2.2 Classification

The final step in the simple text classification process is using machine learning to train and test a chosen classifier on the processed and vectorised data. Many different classification algorithms exist, so the discussion will be limited to some of the more common algorithms used for text classification in earlier research (Colas and Brazdil, 2006, Semberecki and Maciejewski, 2016, Roul et al., 2017, Short, 2019).

Basic classification techniques
Firstly, the Decision Tree (DT) classifier will be discussed. This tree-structured model predicts the label of a data point based on a set of decision rules. For each rule in the tree, a feature of the data point is evaluated, and based on its feature values, the data point moves down the tree until a leaf is reached, which represents a final class label. During the training of the model, the tree is recursively constructed by constantly splitting the dataset into smaller subsets based on specific distinguishing features in those sets. Advantages of using a DT classifier are the interpretability of the learned rules and the fact that little data preparation is needed. However, one drawback is that the model can quickly overfit, meaning that the model memorises the training data in such a way that it cannot correctly classify new, unseen data. Another drawback of this model type is that more complex tasks cannot be learned due to the low complexity of the model.

Another widely used classification approach is the probabilistic Naive Bayes (NB) classifier. This type of classifier is actually a collection of algorithms based on Bayes' theorem, where conditional independence is assumed; the assumption that every pair of features is independent of each other. The probability of a data point belonging to a certain class can thus be calculated by considering the different features in its feature vector without needing to take into account the complexity of different features influencing each other. Although this independence assumption is not realistic for language data, where words in texts are dependent on each other, Naive Bayes can still successfully be used for text classification (Nigam et al., 2000, Short, 2019). Multiple types of NB classifiers, such as the Gaussian NB, Multinominal NB, and Complement NB, exist. The Complement NB classifier is specifically suitable for imbalanced datasets, and is shown to perform better on text classification tasks than the Multinominal NB (Rennie et al., 2003).

Finally, classification can be done using a Support-Vector Machine (SVM), first introduced by Cortes and Vapnik (1995). This learning model focuses on two-class or binary classification, and is based on the idea that the feature vectors of data points can be mapped onto a high-dimension feature space with a mapping function such that the distance between the two classes is as large as possible. After training the model, new data points can then be classified based on which side of the separating hyperplane they are mapped. The SVM approach is quite robust and works very well in high dimensional spaces, but it does not perform well when the classes are overlapping too much instead of being clearly separated.

Multi-class and multi-label classification
In classification tasks, some distinctions can be made between the types of tasks that a classifier is given. Firstly, there is the distinction between binary classification and multi-class classification. In binary classification, the target variable can take on two possible values, for example when labelling whether an email is spam or not. Multi-class classification, on the other hand, is a classification task where the target variable may take on more than two values, and exactly one of them is assigned to each data point, for example when classifying whether a picture of a fruit is an apple, a pear, a banana, or an orange. Multi-label classification deals with multiple classes as well, but here each data point can have zero or more labels assigned to it. An example for this is a movie genre classification task where a movie can be assigned more than one genre.

Although many of the standard classification models are capable of doing classification tasks with more than two classes (with one exception being the SVM), it is often the case that they are not inherently able to assign multiple labels to data points. To remedy both the multi-class and the multi-label problem, the classification task can be transformed into a set of binary classification tasks. This means that instead of training one classifier to classify all classes, a separate binary classifier is trained for each of the classes. This approach is commonly done using the one-vs-rest or the one-vs-one approach. In the one-vs-rest approach, each classifier is fitted to one particular class and against the rest of the classes, whereas in the one-vs-one approach, a classifier is trained for each class pair. When using one of these approaches, it is now possible to assign multiple labels to a data point by testing that data point on each of the binary classifiers trained on a certain class. The data point can then be assigned a label for each class that it is classified as by the binary classifiers.

3.2.3 Limitations of basic classification techniques

One common problem that influences the performance of basic machine learning classifiers is the lack of a large collection of data. When training a classifying model on a task, the idea is that general patterns in the data are learned by the model in order to correctly classify new, unseen data. However, models trained on a small dataset are prone to overfitting, which results in a model that cannot generalise well when presented with new data. A number of general techniques are commonly used to counter overfitting: early stopping, alleviating a data imbalance, and using simple models. Early stopping is a relatively simple technique in which a model stops training as soon as the performance of a held-out part of the data, called the validation set, decreases, and will not need further explanation. Alleviating imbalanced data will be discussed more thoroughly, after which a number of comparative studies will be discussed in which simple classifiers were tested on small and imbalanced datasets.

Alleviating data imbalance
For imbalanced datasets, the distribution of the classes in the data can be changed in order to alleviate the imbalance and improve a model's performance on the smaller class or classes. One approach, called resampling, is generally used and can be divided into two categories: undersampling and oversampling. In undersampling, the majority class is reduced in order to get as much data as the minority class, while in oversampling, the minority class is copied to get the same amount of data as the majority class. Because this project focuses on small datasets, and removing data from the already sparse set could jeopardise the overall accuracy of the model, only oversampling will be considered further. The simplest way of oversampling is by randomly copying data points from the minority class, which can be a robust way of increasing the data, although a downside is that no extra information is added to the data (Ling and Li, 1998). Importantly, as noted by Santos et al. (2018), this oversampling of the data needs to be done at the right time during the training process in order to avoid the problem of making an overoptimistic model by having the same copies both in the train and test set. Alternatively, an approach called SMOTE (Synthetic Minority Oversampling TEchnique) can be used to create synthetic data based on near neighbours in the minority class (Chawla et al., 2002). Finally, instead of changing the distribution of the data, the calculation of the loss function of a model can be adjusted based on the class distribution in order to penalise the model more when misclassifying the minority class. This approach, called reweighting, is commonly used for classification with imbalanced data, but is shown to perform worse or similarly to resampling in different classification tasks (Seiffert et al., 2008). Research has also been done on reweighting and resampling an imbalanced dataset for the classification of texts specifically, where it was found that the success of these techniques depended, among other things, on the type of classifier used (Sun et al., 2009). Because both SMOTE and reweighting appear to be promising options for alleviating class imbalance, these two techniques will be investigated further in this thesis.

Comparative studies for classification of imbalanced datasets
Previous comparative studies between text classification techniques have been done, also focusing on smaller or imbalanced datasets. An early paper on this is the one by Yang and Liu (1999), in which five classification algorithms were compared with a multi-label text classification task for the Reuters-21578 dataset, which is an imbalanced dataset with 90 categories of which more than 80% has less than 100 data points. The results of the comparison showed that the SVM, k-nearest neighbours, and linear least-squares fit classifiers outperformed the neural network and NB classifiers when the number of data points in a category was relatively small (less than 10), but that all models performed similarly when more data was provided. Forman and Cohen (2004) looked at the interaction between different classification algorithms and feature selection techniques and tested which combination performed the best on a binary text classification task for different data distributions, where it was found that different combinations of techniques worked better for different data distributions. However, it was specifically stated that the NB classifier that they tested outperformed the SVM model when there was a 'shortage' of positive or negative samples, which appears to contradict the findings in the previously mentioned paper. This thesis will add to the knowledge gained by previous research by looking at both the NB and SVM model and their performance on classes with different amounts of data to find out which model works well on a smaller number of samples. Additionally, a baseline model not used in the other research, namely the Decision Tree model, will be trained on a text classification problem to see how it compares with the other classifiers. Finally, it is looked into whether certain techniques to alleviate the class imbalance can help the different model types.

3.3 Supervised text classification: language model approach

Although a basic approach to text classification can suffice for a simple classification task, more recent research has focused on building models that can predict a continuation of a text to solve different language processing tasks (Wang et al., 2018, Liu et al., 2019, Bender and Koller, 2020). Such models, called language models, can be used to represent language in such a way that they give a probability for how likely a sequence of words is, or how likely it is that a word follows another word or sequence of words. An advantage of such language models is that because they have a representation of how a certain language works, they can be used for many different language processing tasks.

3.3.1 Developments in language models

The simplest approach to a language model is a probabilistic approach, which looks at the frequencies of sequences of a predetermined number of words to compute how probable certain sequences are. However, this approach has two main drawbacks. Firstly, if the data size increases, the number of possible sequences can grow very quickly. Because the probabilities of all the possible sequences of a certain length are stored, a large part of the model will consist of sequences with a very low probability or a probability of zero, meaning that the model will be very sparse. Secondly, because the sequences of words are a predetermined length, the prediction of a next word given a sequence can only be based on a limited number of previous words, which means that a lot of the word's context is lost with this approach, due to long-range dependencies.

Instead of the probabilistic approach, another possible approach to language modelling is with the use of neural networks. By using a network to create vectors for each word based on their context, and by limiting the vectors to a predetermined size, the sparsity problem is tackled. One of the earlier techniques based on neural networks, word2vec, was groundbreaking because it could capture the contextual relation between words better than the probabilistic approach (Mikolov et al., 2013). However, because the model was trained to look only at a limited number of words around the target word, it could still only capture limited contextual information in the embeddings. Newer approaches, based on Recurrent Neural Networks (RNNs), were introduced to capture even more context by looking at the complete sequence of words occurring before choosing a certain word. To expand on that, such RNNs were made bidirectional to capture the context of a word based on the whole sequence before and after the word, with one example of a model based on this technique being ELMo (Peters et al., 2018). Although bidirectional RNNs were a good advancement for capturing context, a drawback was that the training could be slow, especially with longer sequences. It was thus needed that the process was parallelised, which was possible by using a deep learning model called a Transformer (Vaswani et al., 2017). This model uses an attention mechanism to determine the important context for each part of the sentence, thus not relying on the order of the sequence anymore. BERT and GPT-3, two of the more recent models based on this Transformer architecture, showed great improvements on different language processing benchmarks compared to older models (Devlin et al., 2019, Brown et al., 2020).

In summary, language models based on neural networks solve the sparsity problem and can capture much more of a word's context, thus performing better in language processing tasks than simpler models. Besides this, another advantage of using language models is that after pre-training the model on a large dataset to learn the general patterns in language, the model can be fine-tuned on a new dataset to be used for a specific language task. This process is called transfer learning, and can be specifically useful when there is not enough data available to train a model for a specific task.

3.3.2 Transfer learning

Often in machine learning tasks, one requirement for getting a robustly working model is that it is trained on a large amount of data. A common problem, both for the tasks of image recognition and natural language processing, is that the available dataset is relatively small, making it hard to train a model from scratch. Luckily, a new approach that has gained popularity in the last couple of years is the idea that a pre-trained model can be used as the basis for a language processing task. Instead of building a model for a specific task from scratch, one can take a general model that has been trained on a large collection of language data, replace the more general output layer of the model with a layer that fits the chosen task, freeze the top layers of the model to preserve the high level features learned on the large dataset, and then fine-tune the model on the lower layers based on the provided new dataset. For text classification, this would imply that the output layer should be replaced such that it can classify the texts into the classes defined beforehand.

Transfer learning has been a popular topic among researchers, again because it can be very useful when data is scarce. Earlier research already touched upon the idea that instead of learning a parameter function for every text classification model, knowledge from earlier models can be transferred to increase model performance (Do and Ng, 2005). There has also been earlier research on the selection of the data used for the pre-training of language models for text classification, where a framework was created that could automatically select relevant data for pre-training from an open knowledge space (Lu et al., 2014). More recent research focused on improving text classification results with more advanced models, with some examples using convolutional neural networks (Semwal et al., 2018), attention-based graph neural networks (Pal et al., 2020), and Transformer based models like BERT (Sun et al., 2019).

3.3.3 Limitations of the language modelling approach

Although language models can be seen as a great solution for reducing model sparsity, modelling word context, and handling smaller datasets, there are also some drawbacks that need to be discussed.

Long texts
One limitation of the use of state-of-the-art Transformer based language models is that the size of the input text is limited by these models. Because the attention mechanism in Transformers needs to do n^2 number of calculations for the n tokens in a sequence, the choice was made to set a maximum for the number of input tokens, which are either words, subwords, or characters. For example, the Transformer based model BERT has a maximum input length of 512 tokens, which still makes it possible to use the model for many different language processing tasks where shorter texts are used as input. However, if longer documents are used as input for a given task, this token maximum can pose a problem.

There are multiple ways of handling this issue. One approach used often for texts longer than the maximum number of tokens is to truncate the text in order to fit the text into the input length. One study interestingly showed that taking parts of the first and last text sections of a text yielded better classification results than truncating either the first or last part of a text (Sun et al., 2019). Another approach, used by Wan et al. (2019), consists of separating longer documents into chunks, using those chunks as input for doc2vec, a document vectorisation method based on word2vec, and then aggregating the document vectors using a bidirectional RNN. Alternatively, the longer texts can be summarised to reduce the text length. Some existing summarisation approaches include Summarunner (Nallapati et al., 2017) and TextRank (Mihalcea, 2005). However, this approach does not seem standard, as only one paper was found that discussed the use of summarisation to handle longer texts for language models (Mutasodirin and Prasojo, 2021). Finally, a paper by Beltagy et al. (2020) describes Longformer, a modified Transformer architecture that can process longer documents by changing the attention mechanism such that it scales linearly with sequence length instead of quadratically. This approach showed state-of-the-art results in a selection of language modelling tasks with long texts.

Dutch texts
Many of the larger pre-trained language models using architectures such as BERT and GPT-3 are trained on English texts (Devlin et al., 2019, Brown et al., 2020). However, if a language task needs to be done with a dataset in a different language, such an English language model might not give satisfactory results. Because the provided dataset for this research is in Dutch, the main focus will be on Dutch pre-trained models. Luckily, there are multiple pre-trained Dutch models available. For GPT, a Dutch GPT-2 model was found (de Vries and Nissim, 2020). For BERT, there are two main Dutch models available, namely robBERT (Delobelle et al., 2020) and BERTje (de Vries et al., 2019). BERTje has been successfully used previously for multi-label text classification (De Clercq et al., 2020). Alternatively, a more general multi-lingual pre-trained model like mBERT could be used as well (Devlin et al., 2019). However, this model does not appear to be performing better than BERTje on different language processing tasks (de Vries et al., 2021).

3.4 Testing the models

3.4.1 Validation

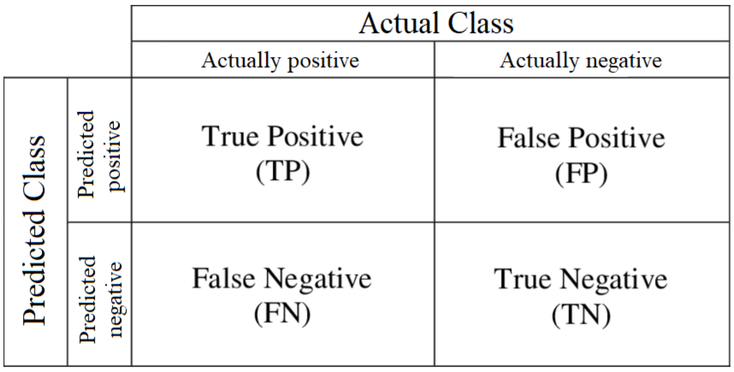
In general, after training a model on a dataset, it is important to test it to ensure that the model can generalise what it has learned during training to new data. The simplest approach to validating a model is to split the given data up in a train set and a test set, where the model is trained only on the train set and afterwards tested on the unseen test set. With this approach, you can see that the model is overfitting when its accuracy is high for the train set but low for the test set, which can be useful information. However, because the standard train-test validation approach only trains and tests the model on one data split, the possibility exists that this particular split happens to be more favourable or less favourable than other splits. To counter this problem, different cross-validation techniques have been developed where multiple data splits are created.

The most common cross-validation approach is called k-fold cross-validation. In this approach, the data is split into k equally sized folds, after which a separate model is trained on a set of k-1 folds and tested on the remaining fold. After training the different models, the mean accuracy of the models is calculated and used as the final overall model accuracy. One variant of the k-fold cross-validation is the leave-one-out cross-validation, in which k is equal to the number of data points. This means that each model is trained on nearly all of the data, and then tested on a single data point. This approach can lead to very unbiased accuracy estimates, but also unreliable estimates due to high variance (Efron, 1983). Nonetheless, it is often used in fields where data is very scarce. Another variant, called stratified k-fold, can be used when the distribution of the data classes is imbalanced. Instead of randomly creating k folds, the folds are created such that the class distribution in each fold is approximately equal to the class distribution in the remaining folds. This avoids the problem that some classes are overrepresented or underrepresented in some of the folds, which could, for example, lead to train or test splits where smaller classes are only present in the test set and not in the train set. This stratified k-fold approach is seen as the most promising one in this research due to its simplicity and the fact that it ensures that the data of all classes is distributed equally.

Other approaches to cross-validation are available as well. For example, a technique first introduced by Dietterich (1998) is 5 x 2 cross-validation, in which a two-fold cross-validation is applied five times. This type of validation was created because the standard k-fold approach was found to have a higher risk of type I error, which is falsely deeming one of two compared models to be better when they were actually comparable in performance. Although the 5 x 2 cross-validation approach reduced the risk of type I error, the k-fold approach had a higher power (Dietterich, 1998), and as noted by Refaeilzadeh et al. (2016), 5 x 2 cross-validation is not yet as popular in the data mining community as k-fold cross-validation. Finally, an approach called nested cross-validation was introduced to avoid the bias created by doing both hyperparameter tuning and model selection on the same dataset. In nested cross-validation, the inner cross-validation layer of the model is used for the hyperparameter tuning, while the outer loop is used for the testing of the model. However, a recent study showed that when few hyperparameters need to be optimised, using non-nested cross-validation will yield qualitatively similar results to nested cross-validation (Wainer and Cawley, 2021).

3.4.2 Evaluation

In classification tasks, the most standard way of evaluating the results of a model is by computing the precision, recall, accuracy, and F1-score of the resulting classifications. This is done by counting the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN), shown in Figure 3.



		Actual Class	
		Actually positive	Actually negative
Predicted Class	Predicted positive	True Positive (TP)	False Positive (FP)
	Predicted negative	False Negative (FN)	True Negative (TN)

Figure 3: Confusion matrix for binary classification.

These scores count the number of times the model correctly and incorrectly labels a data point with the target label and the number of times the model correctly and incorrectly labels a data point with the non-target label. The precision, recall, accuracy and F1-score can then be calculated as shown in Equations 1 to 4.

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \tag{1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \tag{2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \tag{3}$$

$$\text{F1-score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision+Recall}} \tag{4}$$

Here, the accuracy measures how often the model correctly makes a prediction. This is an intuitive measure, but a downside is that if the data is very imbalanced, a model can get a high accuracy score by assigning each data point to the majority class. In this case, the model does classify most of the data 'accurately', but it has not learned anything about distinguishing the data in the training process. The precision of a model measures the percentage of relevant results it found, whereas the recall measures how many of the total number of relevant results were found. Although studying the precision and recall of a model can already give some insight in the model performance, it should be noted that the individual scores might not say everything about a model. A very high precision score can be obtained by only giving a data point a label when the model is very sure about it, and a high recall can be the result of a model labelling all data points with a certain label. Because of this trade-off, the F1-score is an important metric for classification, because it is calculated by looking at both the precision and recall.

In multi-label classification, as explained by Sorower (2010), these standard evaluation metrics cannot be used directly because a classification with multiple labels can be fully correct, partially correct, or fully incorrect. To account for this, two evaluation metrics are used for multi-label classification: the macro F1-score and the micro F1-score. The macro F1-score is calculated by taking the average of the individual F1-score of each class, whereas the micro F1-score is calculated by globally summing up all the FP, FN, TP, and TN counts for each of the classes and then calculating the F1-score using those global scores. While the macro-average score gives an equal weight to all the classes, the micro score gives an equal weight to all the data points. This difference can influence the scores of a model trained on a dataset with an imbalanced class distribution. The macro score will be lower if the classification of the minority classes is done poorly because these classes have the same amount of influence on the score as the majority classes, whereas the micro score can still be high in this scenario because the majority classes have more influence on the score than the minority classes.

3.5 Summary

This chapter has given an overview of the different methods for text classification, with a focus on supervised learning approaches. It was found that both simpler and more complex techniques exist, each with their own advantages, drawbacks, and difficulties. For the simpler approaches, some drawbacks are that the simple feature vectors do not capture a lot of contextual information about the texts, and that the simpler classification techniques might perform badly when presented with small and imbalanced datasets. This last issue could be resolved by balancing out the dataset using either reweighting or resampling techniques. Alternatively, to avoid these problems, pre-trained language models can instead be used as a basis of a model, and can be fine-tuned on a given dataset to train for a specific task. This approach is used often because the state-of-the-art models have shown to work well on different language processing tasks, and because they can be fine-tuned successfully on smaller datasets. However, two drawbacks are that most language models are trained on English texts whereas a Dutch language model is needed here, and that the best performing language models often cannot directly take longer documents as input, thus making it necessary to adjust the documents or model in some way to make it possible to use the language models. Finally, different validation and evaluation techniques were discussed to see which approaches could be used for ensuring that the results of the models are valid and that the different models can be compared effectively and correctly.

Although much research has been done on the topic of text classification and many different techniques have been implemented and compared in the past, it is still not certain which text classification method will give the best results for a dataset with many different aspects that limit the performance of different models. Different comparative studies were found, but no earlier research was discovered in which simpler classifiers and more complex language model based classifiers were directly compared to each other in a multi-label classification task with a small and imbalanced dataset of longer texts. The rest of the thesis will thus focus on implementing these classifiers and testing which approach yields the best results, with the goal of learning more about which classification techniques are most suitable for such a specific language task and how the different data aspects influence the models.

4 Methods

This thesis is focused on implementing and comparing different text classification techniques for a case study with a small and imbalanced dataset of long Dutch letters of objection. The previous chapter reviewed the different classification techniques used for text classification and their drawbacks, where it was found that simpler classification methods can be limited by the data size and distribution, and that for more complex language models the resources for processing longer texts and texts in a language other than English are limited. The main questions now remain how these two types of models compare if a dataset has all these limitations, and how the different limitations of the data compare in their influence on the models' performance. In this chapter, the methodology for answering those questions will be explained concretely. Section 4.1 will explain how the baseline classification approach will be implemented and which important choices were made, and Section 4.2 will discuss the implementation of the language model approach. In Section 4.3, the chosen validation technique and evaluation metrics will be named, and finally, in Section 4.4 the decisions made in this chapter are summarised in a table.

4.1 Baseline text classification approach

For the baseline approach to text classification, rule-based classification, topic modelling, and supervised text classification were considered. It was concluded that rule-based classification and topic modelling were not appropriate for the project. As was explained in the previous chapter, the rule-based approach has the drawbacks of needing an expert to craft specific rules for every category and needing to be updated completely when new categories are introduced. Even though domain experts were available for this project, it would take a lot of their time to help craft all these rules. Additionally, it is not known whether this approach would outperform a supervised classification approach, which is a more common and faster approach to classification. Topic modelling is a widely used technique for finding topics in texts, but since the provided data are already labelled with categories for the different topics, a supervised learning approach is a more straightforward choice than an unsupervised approach. Moreover, given the small size of the dataset and the fact that most documents contain quite a number of different categories, it was not probable that the topic model could distinguish the different topics in the text accurately and robustly. Because of these drawbacks, the choice was made to focus on supervised text classification for this project.

As was explained in Section 3.2, for supervised text classification two main steps are needed, namely the transformation of documents into feature vectors, and the training and testing of a classification model on those vectors. For the creation of the feature vectors, the tf-idf statistic will be used. This approach was chosen because of its simplicity and robustness, along with the fact that it is commonly used for text classification tasks and can thus be seen as a good baseline. Although the bag-of-words approach was also considered, the tf-idf was preferred because it captures slightly more information than the bag-of-words approach without costing more memory or computational power. Before the documents are vectorised, a number of standard text preprocessing steps will be done to clean up the data and to reduce the vector size. These steps are explained in more detail in Section 5.1 of the next chapter.

After cleaning up the data and transforming the documents into vectors, the final step of the process is to train a classification model on the vectors to see whether it can learn to correctly assign the category labels to the documents. The Decision Tree classifier, Complement Naive Bayes classifier, and the SVM, which were the three classification models discussed in Section 3.2.2, are chosen as the classification models for the baseline classification approach. More than one baseline model was chosen because it was uncertain which technique would work best based on the previous literature. Although different comparative studies have been done in the past, most of them either had a different type of classification problem, did not consider the size or imbalance of the data, or did not compare the specific classifiers considered for this project. These three particular models were selected because they have been used successfully for text classification in the past, and are all based on different ideas and principles, which means the comparison between the three can also be an interesting analysis in itself for other future research with similar data.

Because the type of classification for the given dataset is multi-label classification, the baseline classifiers need to be adjusted to be able to handle assigning multiple labels to a document. To do this, the one-vs-rest approach is chosen. This approach is preferred over the one-vs-one approach for this project because the number of classes in the dataset is 24, which would result in more than 250 trained classifiers for the one-vs-one approach, whereas the one-vs-rest approach would need to train 24 classifiers.

Finally, to minimise the influence of the data imbalance on the classification performance, both the SMOTE resampling and the class reweighting techniques will be implemented. This was chosen because the literature was unclear about which technique would work best for the given task, and comparing them could give more insight into which technique could potentially benefit text classification tasks with similar datasets in future research.

4.2 Language model text classification approach

Due to the simplicity of the baseline classifiers and the complexity of the given dataset, it is uncertain whether these classifiers are able to successfully learn the patterns in the data such that they can distinguish the different possible classes that can be assigned to the documents. Because newer, more advanced approaches to text classification exist, the question is whether these could potentially yield better results for this task. As was discussed in Section 3.3.1, there have been multiple interesting developments in the field of language processing with language models, with the most recent language model types being based on the Transformer architecture. Because this type of model is currently one of the best performing model types on different language processing tasks, and because it can easily be fine-tuned for a new dataset and a new task, a Transformer based language model will be implemented to see whether it can outperform the baseline text classification approach. Different Transformer based language models exist, but in this thesis, BERT will be used for the task at hand. Although other models were considered as well, BERT was preferred mainly because of it being an open access language model with multiple options for pre-trained models specifically in Dutch. From the Dutch model options, BERTje will be used because it outperformed other Dutch and multi-lingual BERT versions on different language processing tasks and was shown to be usable for text classification in previous research (de Vries et al., 2021, De Clercq et al., 2020).

One of the limitations of a Transformer based language model such as BERT is that there is a limited input size of 512 tokens, which is a problem for the given dataset because more than half of the documents (52%) exceed that input length with input sizes ranging from 513 to around 5000 tokens. In the end it was decided that the simplest approach to handling longer texts would be used, meaning that during the tokenisation of the documents, each document was truncated after 512 tokens. Initially, after considering different approaches, implementing a Longformer version of the BERTje model was thought to be the most promising approach to getting the model to handle longer texts. As was discussed in the previous chapter in Section 3.3.3, compared to the truncation of the texts, this approach ensured that the most important information in the documents was not lost, and compared to the summarisation approach, which did not appear to be a common approach to handling long texts for BERT models, the Longformer approach was used more often in research and the positive results shown there indicated that the Longformer approach could be more promising. It also appeared to be more easily implemented than the chunking approach of Wan et al. (2019), since a notebook with a guide on how to implement the Longformer architecture for any kind of model was provided by the makers of the Longformer BERT model. However, due to time constraints, the implementation of the Longformer version of the BERTje model could not be finished in this project, which is why the simple truncation approach was used. Although it could have been decided that the baseline models should have also used truncated texts to make the comparison of the models more equal, this was not done in the end since this thesis focuses on looking at solutions to limitations of specific model types and having long texts is not a limitation of the baseline models.

To make the pre-trained BERTje model usable for the task at hand, the output layer of the model needed to be changed such that it could do multi-label classification. To do this, the *BertForSequenceClassification* model from the HuggingFace library (Wolf et al., 2019) was used, which is a BERT model type that has a classifier output layer which also has the option to do multi-label classification. However, because the BERT implementation using the multi-label data as input initially did not appear to work, it was decided that a one-vs-rest approach would also be implemented for the language model approach in order to test whether that could solve the issues with the language model performance. This resulted in an approach similar to the one used for the baseline models, in which a separate BERTje model was trained on each individual class.

4.3 Model validation and evaluation

To validate the results of the different baseline classifiers and the one-vs-rest language model approach, a stratified k-fold cross-validation will be used. The k-fold was preferred over the standard train-test split because it enlarges the chance that the quality of the results is consistent and not due to an accidental favourable or bad split. The stratified version of the k-fold was chosen for the model trained with the one-vs-rest method because it ensures that each of the classes in the data is divided equally over the train and test sets, which is important given that the class distribution of the data provided for the models is relatively skewed. For the multi-label language model, the normal k-fold cross-validation will be used, because the presence of multiple labels in one document complicates the process of splitting the data equally per label. Other cross-validation approaches were considered as well, but the k-fold was deemed the most fitting for this project given its relative simplicity and the fact that it is currently the most widely used option in the field.

The evaluation of the different models will be done by calculating their macro and micro F1-scores, since these are commonly used metrics for multi-label classification that can measure both the general performance of the models and how well the models can handle smaller classes. Because of the k-fold implementation, this will result in k macro F1-scores and k micro F1-scores for each classifier. The mean score of the classifiers can be calculated with these separate scores, after which a one-way analysis of variance (ANOVA) (Girden, 1992) can be done to test whether there is a significant difference between the results of the models.

4.4 Summary of decisions

To give a clear overview of the decisions described in this chapter, the choices that were made for both the baseline models and language models are shown concisely in Table 3.

Table 3: An overview of the decisions made for the methods of this thesis.

Choice	Baseline models	Language models
Vectoriser	Tf-idf	BERTje vectoriser
Classifiers	DT, NB, SVM	BERTje
Handling multi-label data	One-vs-rest	One-vs-rest and multi-label
Data attribute mitigation	Data imbalance: reweighting and resampling	Text length: truncation
Cross-validation technique	Stratified k-fold	Stratified k-fold and normal k-fold
Evaluation metrics	Macro and micro F1-score and F1-score per category	Macro and micro F1-score and F1-score per category

5 Experimental set-up

This section describes the practical set-up of the experiments that were run to answer the research questions discussed in Chapter 1. All models were trained on the multi-label dataset of 403 letters of objection, described in more detail in Chapter 2. This chapter is structured as follows: in Section 5.1, the set-up of the tf-idf vectorisation of the data and the training of the baseline models will be explained. After this, Section 5.2 describes the data preprocessing and model structure used for fine-tuning the BERTje language model. Section 5.3 explains the practical implementation of the validation and evaluation of the models, and finally, Section 5.4 gives an overview of the experiments and the parameters of each model.

5.1 Baseline models

5.1.1 Tf-idf set-up

As was described in previous sections, the first step in the baseline model training process is turning the documents in the dataset into vectors, which is done using the tf-idf statistic. In this project this is implemented using the *TfidfVectorizer* function provided by *scikit-learn*'s feature extraction library. This function takes a collection of documents and a number of parameters as input and outputs a matrix with n-grams and their computed tf-idf score per document. Most of the vectoriser's parameters were set on the default setting found in version 1.1.1 of *scikit-learn*, but some choices or decisions were made explicitly and will be discussed here.

Before calculating the tf-idf scores, it was decided that although the tf-idf score can be calculated for any type of n-gram, only unigram tf-idf scores would be needed. This was done because of the assumption that single words would already provide enough information for the model to distinguish document subjects from each other, which was based on the inspection of the data and the observation that many classes in the dataset had very clear words related to them. For the preprocessing of the data, a couple of decisions were made, such as the removal of stopwords, the removal of accents from letters, and the lowercasing of words in the texts. These were all implemented to reduce the number of dimensions in the document vectors. Additionally, the choice was made to not consider the tf-idf score for words that occur only in a single document in the dataset, which was done for two reasons. The first reason was that due to errors made by the text detection program that was used to read some of the letters of objection, some of the texts contained quite some noise. This led to many unique words in the dataset like 'st0p' instead of 'stop', or meaningless strings of characters like '1d4jc83jdyjhdd'. To filter these out of the dataset, the minimal document frequency of words considered for the tf-idf score was set to two. The second reason for this choice was that words that occur in a single document might not be informative enough to help the model to generalise well.

After calculating the tf-idf scores for the documents, words containing numbers or underscores were removed from the tf-idf matrix because they were often not informative and unnecessarily increased the size of the matrix. This might have led to the removal of some real words like ‘0-meting’ or ‘A27’, but they were believed to not add significant information to the documents. After this final step, the result was a tf-idf matrix containing 403 rows matching the number of documents in the dataset and 7465 columns for each unique word in the dataset and its tf-idf score for each document.

5.1.2 Model set-up

The three baseline models that have been chosen for this project, the Decision Tree, Complement Naive Bayes model, and Support-Vector Machine, have been implemented using the *sklearn* library. This library has pre-built classifier model architectures that can be used to easily define a model. Each model has a set of hyperparameters that can be tuned in order for the model to fit well on the data. The hyperparameters chosen for each model will be discussed in the next section. Due to the data being multi-labelled, a one-vs-rest approach needed to be implemented in order to train the baseline models on the data. Although a built-in one-vs-rest function exists, it was decided that a one-vs-rest approach would be implemented manually by initialising and training 24 separate models, with 1 model for each category. This was done to make the implementation of the cross-validation and calculation of the final F1-scores easier. The data was split into 80% train data and 20% test data, resulting in approximately 322 train samples and 81 test samples per model. This split size was used for each model to ensure that there would be enough training data given the small data size.

To see whether it was possible to counter the effect of the small size and imbalanced class distribution in the dataset, two techniques were implemented: reweighting of the classes and resampling of the data. The reweighting of the classes was done for the Decision Tree and SVM classifiers, which both have a parameter for reweighting when initialising the model. The weights of the classes were chosen to be ‘balanced’, which means they are inversely proportional to the class frequencies in the training data. For the Naive Bayes classifier, no class weighting was implemented because the model already considers each class probability separately and is thus not influenced by class imbalance in the way other models are. For the resampling of the data, the *SMOTE* function from the *imblearn* library was used. It was applied after splitting the train and test set and was only applied on the train set in order to avoid creating data that is very similar and occurs both in the train and test set. Because the number of samples for some of the classes in the dataset was rather small, the resampling was implemented in such a way that it was only used when there were at least two samples of the class in the train set. The number of considered neighbours for the resampling process was set to either the default number of 5, or to the number of samples minus 1 if there were 5 or less samples in the train set. After the resampling process, the number of positive and negative samples for a class in the training set were the same.

5.1.3 Hyperparameter selection

As was explained, each baseline model has its own hyperparameters than can be tuned to fit the data well on the model. In this project, the hyperparameters were tuned for each of the three baseline models using the original dataset, as well as for the three model versions that were trained on the resampled data and the Decision Tree and Support-Vector Machine models that used reweighting. This resulted in 8 hyperparameter sweeps.

For the Decision Tree model there were many hyperparameters that could be tuned, so to limit the search space, only the criterion, maximum tree depth and maximum number of considered features were tuned. After testing different hyperparameters, these hyperparameters appeared to be the most influential on the results of the model trained on this dataset. The functions that were considered to measure the quality of the split, called the criteria, were *gini*, which refers to the Gini impurity measure, and *entropy*, which refers to the Shannon entropy measure. Both are explained in more detail in the documentation of *sklearn*'s Decision Tree classifier. For the maximum depth of the tree, the values 10, 50, and 100 were tested, as well as an unlimited tree depth. For determining the maximum number of features considered for a split, the parameters *auto*, *sqrt*, and *log2* were tested. The hyperparameters for the Decision Tree models were tuned for the standard DT, the reweighted DT, and the DT trained with resampled data. For all three run types, the best hyperparameter combination was the Gini criterion with a maximum tree depth of 50 and a maximum number of considered features determined by taking the square root of the total number of features.

For the Complement Naive Bayes classifier, two parameters were tested: *alpha*, a smoothing parameter, and *norm*, which is the normalisation of weights. For alpha, the values 0, 0.01, 0.1, 1, and 5 were tested, whereas for the normalisation it was tested whether applying normalisation or not improved the model results. For the standard Naive Bayes model it was found that the best hyperparameters were an alpha of 0.01 with normalisation, while for the resampled Naive Bayes model the best combination was an alpha of 0.1 with no normalisation.

For the Support Vector Machine, the number of tested parameters was limited to the following three: C, which is a regularisation parameter, the kernel type, and gamma, which is the kernel coefficient for specific kernel types. For C, the considered values were 0, 0.01, 0.1, 1, and 5. The possible kernel types were *linear*, *poly*, *rbf*, and *sigmoid.* For gamma, the calculation options for the coefficient were *auto* and *scale.* It was found that the data was linearly separable, which meant the linear kernel type worked best for the model. For the linear kernel, the regularisation parameter and kernel coefficient have no influence on the model, so these parameters were set to default in subsequent runs. It was also found that the reweighting of classes is implemented in such a way for the SVM model that it influences the regularisation parameter C. Since C has no influence on the model when it uses a linear kernel, the reweighting of classes for the SVM model was thus excluded from the result.

5.2 BERT model

5.2.1 Preprocessing

For the implementation of the language model for the classification of the multi-label data, the Dutch BERTje model by de Vries et al. (2019) was chosen. Before the model could be pre-trained, the documents in the dataset first needed to be tokenised using the BERTje tokeniser. After the tokenisation, the texts were either truncated or padded to 512 tokens depending on their length, and a begin and end token were added. For the multi-label implementation of the BERT model, the provided labels were one-hot encoded to ensure that the size of the tensors would be consistent. For the one-vs-rest BERT model implementation, single values of either 0 or 1 were used for the labels. As with the baseline models, a split with 80% train data and 20% test data was used for the BERT models.

5.2.2 Model and parameters

For fine-tuning a pre-trained language model on the data, the BERTje model was used, which could be loaded using HuggingFace's Transformers library (Wolf et al., 2019). The model was fine-tuned on the data using the *AdamW* optimiser, and the learning rate scheduler provided by the Transformers library was used, where the initial learning rate value was set to 2×10^{-5}. These settings were based on the settings found in code used for other text classification tasks. For the training of the model, a batch size of 8 was used.

In the multi-label BERT classification implementation, the loss was calculated with a cross-entropy loss function, and the number of epochs was set to 20. This number of epochs was chosen because after preliminary testing it appeared that the validation loss was not decreasing as much and the F1-scores were not increasing anymore. Since the model outputs a non-normalised prediction score for each class, this score first needed to be converted to a normalised prediction, after which a threshold needed to be determined for deciding when the model was certain enough that a class belonged to an input text. Since a general threshold for all classes was most preferred, some preliminary testing was done to decide which threshold worked best overall, which resulted in a threshold of 0.25.

For the one-vs-rest BERT classification implementation, a Mean Squared Error loss function was used for the loss calculation, and 3 epochs were used to train the model on the data. The number of epochs was again determined by preliminary testing, where it was found that the decrease of the loss of the models for most classes started to stagnate after 2 or 3 epochs and that the model started to overfit after more epochs. For determining the decision threshold of the one-vs-rest BERT model, the receiver operating characteristic (ROC) curve was used, which uses the number of true positives and false positives of a class to calculate the best threshold.

5.2.3 MultiEURLEX dataset

Because the multi-label BERT implementation initially did not perform very well, it was decided that a test would be done to see whether the low performance of the model was caused purely by the small dataset of the model or whether other factors played a role as well. To test this, a dataset similar to the one used in this project was chosen, which was then trained on the same model with increasing amounts of data to check whether that would improve the final scores of the model. For this test, the MultiEURLEX dataset was used. This dataset was introduced by Chalkidis et al. (2021) and consists of a collection of

65000 European Union laws translated in different languages and annotated with multiple labels from a collection of concept labels. Only the Dutch part of the MultiEURLEX dataset was used for training the BERT model. For testing the influence of the data size on the performance of the multi-label BERT implementation, it was first checked whether the data in the MultiEURLEX dataset was similar enough to compare it to the data used in this project. Table 4 shows some of the attributes found in the two datasets, and what can be seen is that when taking a similar amount of data points from the MultiEURLEX dataset, a similar number of 'long' documents and a similar number of labels was found in both datasets. After establishing this, a number of pre-trained BERTje models, which used the same hyperparameters and truncation technique as the multi-label BERT model, was fine-tuned on increasing amounts of data from the MultiEURLEX dataset.

Table 4: An overview of the similarity in data properties between the dataset used in this thesis and the Dutch MultiEURLEX dataset.

	Given dataset	MultiEURLEX
Number of documents	403	400
Number of documents with > 512 tokens	211	273
Number of labels	24	21

5.3 Validation and evaluation

To validate the trained baseline and BERT models, a k-fold cross-validation was implemented. The number of folds was set to 5, resulting in 5 separate 80-20 train-test splits for each model. For the baseline models and the one-vs-rest BERT models, a stratified k-fold was used to ensure a balanced train-test sets, whereas for the multi-label BERT model a normal k-fold was used due to the complexity of equally splitting multi-label data. Because the one-vs-rest models train a separate model for each class, and because each of those models was trained 5 times on a separate data fold, the total number of trained models was 5x24=120 for the baseline and one-vs-rest BERT models.

For the evaluation of the models, the macro and micro F1-score of the baseline and BERT models were computed for each fold. For the one-vs-rest models, this was done by training each class on one of their folds and then using the confusion matrix and F1-scores of each class to calculate the macro and micro F1-score for that fold. For the multi-label BERT model, a similar process was used, but because the classes were not trained on separate models but all in the same model, the model was trained on one fold of multi-label data and the scores for each class were then computed for that fold. This process resulted in 5 macro and micro F1-scores for each model, which were then compared with a one-way ANOVA test. Additionally, for each model the F1-scores per category were computed to see how individual classes performed per model type.

5.4 Overview of the experiments and hyperparameters

To give a clear overview of the complete set-up of the different models and their inputs and outputs, a flowchart of the set-up can be found in Figure 4, while Tables 5 and 6 show the parameters of the baseline and language models respectively.

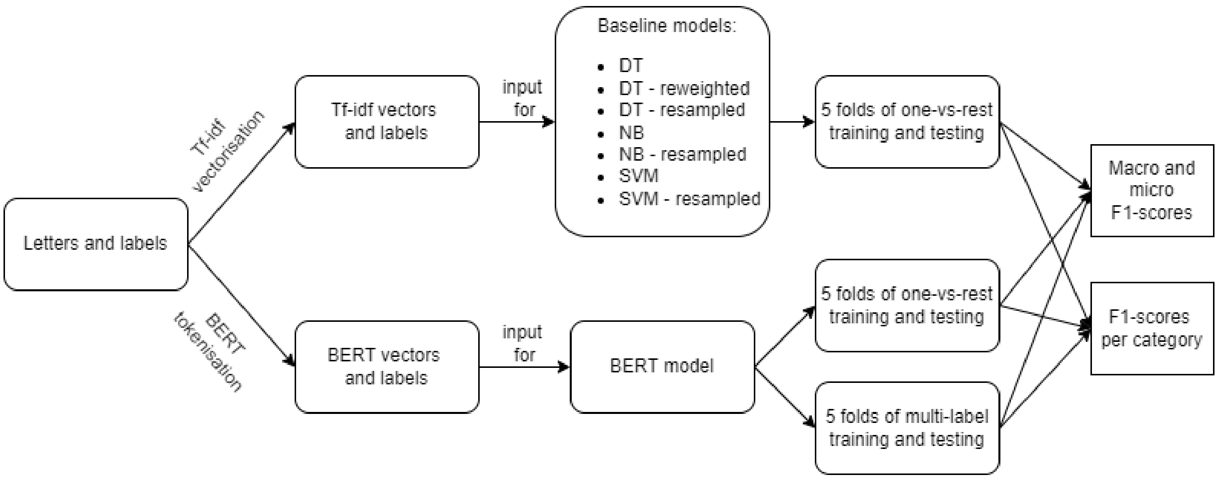


Figure 4: A flowchart of the experimental set-up, describing the inputs and outputs of the different models.

Table 5: An overview of the parameters and choices made for the implemented baseline models.

Choice	Decision Tree	Naive Bayes	Support-Vector Machine
Vectoriser	Tf-idf	Tf-idf	Tf-idf
Hyperparameters	Base DT, reweighted DT, resampled DT: criterion: 'gini' max tree depth: 50 max features: 'sqrt'	Base NB: alpha: 0.01 normalisation: True Resampled NB: alpha: 0.1 normalisation: False	Base SVM, resampled SVM: kernel: 'linear'
Train-test split	Stratified k-fold with 5 splits of 80-20 train-test	Stratified k-fold with 5 splits of 80-20 train-test	Stratified k-fold with 5 splits of 80-20 train-test

Table 6: An overview of the parameters and choices made for the implemented language models.

Choice	Multi-label BERT	One-vs-rest BERT
Vectoriser	BERTje vectoriser	BERTje vectoriser
Handling long documents	Truncation after 512 tokens	Truncation after 512 tokens
Optimiser	AdamW	AdamW
Initial learning rate	2×10^{-5}	2×10^{-5}
Batch size	8	8
Loss function	Cross-entropy loss	Mean Squared Error loss
Epochs	20	3 per category
Decision threshold	0.25	Calculated per category
Train-test split	K-fold with 5 splits of 80-20 train-test	Stratified k-fold with 5 splits of 80-20 train-test

6 Results

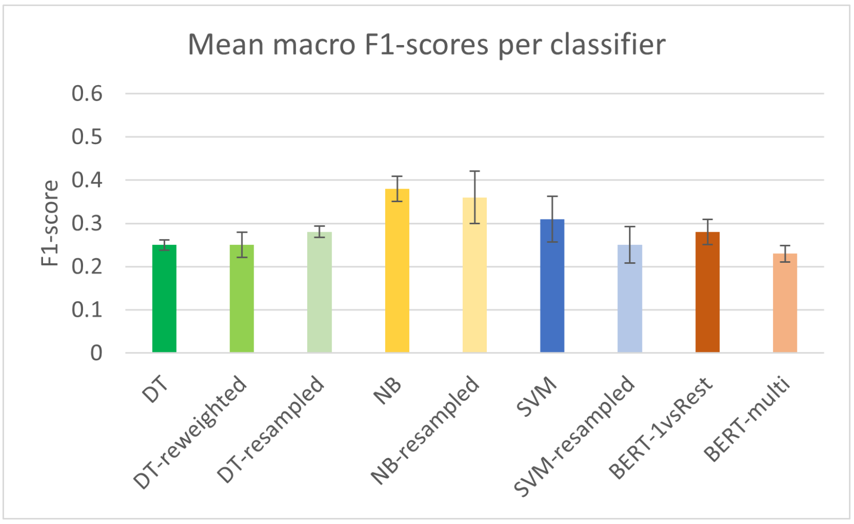
In this chapter, the results of the experiments discussed in the previous section are reported and analysed. These experiments consisted of training a selection of baseline models with or without reweighting of the classes or resampling of the data, as well as training a one-vs-rest implementation of a BERT model and a multi-label version of BERT. Section 6.1 will discuss the main findings regarding the difference between the baseline models and the BERT implementations. Section 6.2 will take a closer look at the baseline models to find out more about how they differentiate different classes and how reweighting and resampling influenced their performance. Finally, Section 6.3 will discuss the steps taken to learn more about the multi-label BERT model and why it did not outperform the baseline classifiers.

6.1 Model comparison: baseline vs BERT

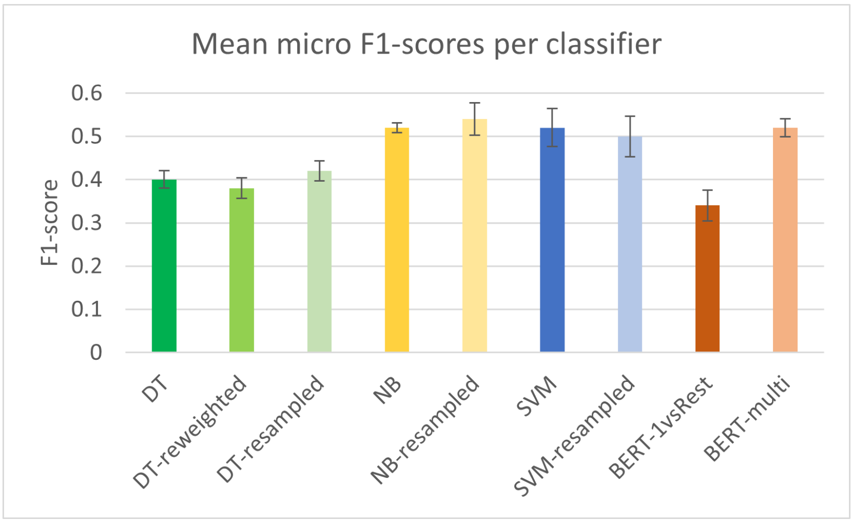
After training the baseline models and the one-vs-rest BERT model per category on 5 folds of data, the final results consisted of a collection of macro and micro F1-scores for each fold of each model as shown in Figure 5, as well as an average F1-score for each class in the dataset shown in Table 7. As was explained in Section 3.4.2, the F1-score is a metric that balances the recall and precision scores of a model, the macro F1-score of a model represents the average of the individual F1-score of each class, and the micro F1-score represents the score calculated by summing up the FP, FN, TP, and TN counts for each of the classes and using those sums for the F1-score calculation. Whereas the macro F1-score gives an equal weight to all classes, the micro F1-score gives an equal weight to all data samples.

6.1.1 Analysing macro and micro F1-scores per model

The results of the model training and testing can be seen in Figure 5. The exact macro and micro F1-scores per fold are not discussed in detail in this chapter, but can be found in Table 9 in the Appendix. A one-way ANOVA was conducted to compare the effect of the model type on the resulting macro F1-scores, and it was found that there was a significant effect of model type on model performance at the $p<.05$ level for the different models [$F(8,36) = 10.68$, $p = 1.55 \times 10^{-7}$]. More specific Tukey HSD tests were done to compare pairs of models, where a difference comparison with a p-value under 0.05 is considered significantly different. The Tukey HSD tests showed that of all the models, the base Naive Bayes model and Naive Bayes model with resampled data both had a significantly higher macro F1-score ($p<.05$) compared to the different Decision Tree models, the one-vs-rest and multi-label BERT model, and the SVM model with resampled data. No significant difference was found between the macro F1-scores of the base Naive Bayes model, the Naive Bayes model with resampled data, and the base SVM model. The multi-label BERT model had a significantly lower macro F1-score ($p<.05$) compared to the different Naive Bayes models and the base SVM model. No significant difference between the multi-label BERT and the different Decision Tree models, resampled SVM model, and one-vs-rest BERT model was found. The one-vs-rest BERT model was shown to not outperform any of the baseline models with respect to the macro F1-score, since it either scored a significantly lower macro F1-score, or no significant difference was found.



(a) Macro F1-scores.



(b) Micro F1-scores.

Figure 5: The mean macro and micro F1-score calculated for each classifier. The mean is taken over the 5 folds on which separate models were trained and tested. The error bars represent the standard deviation from the mean scores.

Similarly to the macro F1-score analysis, a one-way between subjects ANOVA was conducted to compare the effect of the model type on the resulting micro F1-scores, and it was found that there was a significant effect of model type on model performance at the $p<.05$ level for the different models [$F(8,36) = 28.52$, $p = 2.31 \times 10^{-13}$]. After comparing the different models with the Tukey HSD test, it was found that all versions of the Decision Tree model got significantly lower micro F1-scores ($p<.05$) than the different Naive Bayes and SVM models. Between the different Naive Bayes and SVM models, no significant difference between micro F1-scores was found. The micro F1-scores of the one-vs-rest BERT model were significantly lower compared to the different Naive Bayes models, SVM models, and the multi-label BERT model ($p<.001$) and compared to the Decision Tree with resampled data ($p<.05$), while no significant difference was found between one-vs-rest BERT and the base Decision Tree model and Decision Tree model with reweighted classes. The multi-label BERT model scored significantly higher ($p<.01$) than the different Decision Tree models, while no significant difference was found between the multi-label BERT model and the different Naive Bayes and SVM models.

After inspecting the differences in F1-scores, it could be concluded that there is no one model that clearly outperforms the rest of the models with respect to both the macro and micro F1-scores. The overall micro and macro scores are not very high, which shows that none of the models are capable of correctly classifying all of the classes consistently. Against expectations, both language model implementations were not able to outperform the baseline classifiers, which could indicate that for this small amount of data, a simpler model is the preferred choice for text classification. Interestingly, it can be seen that the type of F1-score that is reported (macro or micro) influences the view of how well the models perform. For example, the Naive Bays model outperforms most of the other models on the macro F1-score, indicating that the model can distinguish a large number of classes relatively well, while its micro F1-score is similar to a number of other models, indicating that these models all correctly identified a similar number of data points. Similarly, the multi-label BERT model appears to have trouble with getting a good score for a large number of classes given its macro F1-score, while its micro F1-score indicates that it can actually correctly classify quite a number of data points. To get more insight into why the models score relatively low and why there is a difference in micro and macro F1-scores, the F1-scores per category can be studied.

6.1.2 Comparing per-category results

In addition to the collection of macro and micro F1-scores per model, the F1-scores per category were collected for each model as well. The results can be found in Table 7, where each score represents the F1-score of the category trained on that model as the mean over the 5 folds, and where the bold values indicate per category which model had the highest mean F1-score. As can be seen in the table, the base Naive Bayes model scored the highest F1-score on the largest number of classes, which would explain the relatively high macro F1-score reported in the previous section. The results also show that there were quite a few categories, mainly the ones with little data, that could not be learned by most of the models and scored an F1-score of 0.0. This would explain the low macro F1-scores found for all of the models, since every class is weighted equally in the calculation of the macro score, thus resulting in the categories with little data influencing the final macro F1-score more negatively than the final micro F1-score, which is calculated based on the number of data points that were correctly classified.

What can be observed is that for categories with more than 30 samples in the data, the base Naive Bayes model and Naive Bayes model with resampled data got the highest F1-scores of all the models with the exception of one category, *Participation*, which was one of the only categories where both BERT models appeared to outperform all baseline models.

Table 7: Mean F1-scores of each label category per model type.

		Decision Tree			**Naive Bayes**		**SVM**		**BERT**	
Category	**Samples**	base	reweight	resample	base	resample	base	resample	1vsRest	multi
Noise	229	0.70	0.67	0.67	**0.82**	**0.82**	0.78	0.78	0.80	0.78
Design	158	0.48	0.43	0.57	0.71	**0.72**	0.65	0.66	0.70	0.68
Plan damage	97	0.38	0.35	0.39	**0.58**	0.55	0.39	0.41	0.57	0.53
Air quality	77	0.37	0.39	0.33	**0.47**	0.43	0.44	0.38	0.45	0.38
Traffic	72	0.25	0.38	0.34	0.49	**0.52**	0.47	0.35	0.50	0.50
Execution	68	0.29	0.23	0.37	**0.39**	0.42	0.37	0.29	0.34	0.33
Participation	61	0.29	0.21	0.28	0.35	0.32	0.32	0.20	0.38	**0.39**
Landscape	54	0.22	0.16	0.28	**0.32**	0.27	0.15	0.16	0.27	0.00
Nature	38	0.17	0.14	0.23	**0.40**	0.25	0.30	0.08	0.28	0.10
Vibrations	38	0.42	0.41	0.52	0.50	**0.62**	0.49	0.50	0.44	0.47
Water	37	0.35	0.38	0.27	0.42	**0.46**	0.44	0.44	0.29	0.33
Desicion-plan	36	0.21	0.24	0.25	**0.31**	0.26	0.26	0.09	0.25	0.11
Project goal	34	0.08	0.18	0.21	**0.46**	0.37	0.37	0.09	0.27	0.39
EIR	22	**0.36**	0.09	0.05	0.14	0.09	0.19	0.13	0.10	0.07
Interface projects	12	0.13	0.03	0.00	**0.20**	0.13	0.13	0.13	0.12	0.00
Level crossings	11	0.41	0.13	0.70	0.67	0.73	**0.86**	0.77	0.52	0.47
External safety	10	**0.18**	0.10	0.07	0.10	0.13	0.13	0.00	0.06	0.00
Archaeology	9	0.10	0.08	**0.20**	0.00	0.00	0.00	0.00	0.04	0.00
Health	7	0.46	0.23	0.33	**0.65**	**0.65**	0.33	0.33	0.18	0.00
Climate	7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.04**	0.00
Smart mobility	7	0.00	0.37	0.00	**0.59**	0.53	0.08	0.00	0.08	0.00
Use of space	5	0.00	**0.47**	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Cables and pipes	4	0.13	0.20	**0.60**	0.53	0.33	0.20	0.20	0.05	0.00
Soil	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

When comparing the BERT models with the baseline models, it can be seen that they almost never outperform the baseline models, with the two exceptions being the aforementioned *Participation* category and the *Climate* category. However, for the categories with more than 50 samples, the BERT models still score similar F1-scores to the best performing baselines, indicating that they are able to learn patterns in the dataset when enough data is provided. For the one-vs-rest BERT model, it should be noted that all categories were trained on 3 epochs because most categories started to overfit after more training, but that the categories with the most samples in the dataset could have been trained for longer. Although training the larger categories for more epochs was tested, it appeared that although the F1-scores ended up being slightly higher, they were still not higher than the baseline F1-scores.

For categories with less than 30 samples, it depended on the category which model could most successfully find a pattern in the data. One interesting thing to note is that for some of the categories with very little data (*Level crossings* with 11 samples, *Health* and *Smart mobility and innovation* with 7 samples, and *Cables and pipes* with 4 samples), it was still possible to get a relatively high F1-score of over 0.5, while other categories with a similar or even higher number of samples were harder for the models to distinguish. This could be explained by the fact that some categories have very clear terms that can be used to indicate whether a document mentions that specific subject or not, where those terms do not overlap with any of the other categories. For example, the terms ‘overweg’ and ‘overwegen’ (‘level crossing’ and ‘level crossings’) can clearly indicate whether a text is about level crossings or not, and since these terms are often not used in any of the other categories, they are very good unique indicators. Other categories, like *Landscape,* *integration, and spatial quality* or *Participation*, might be a bit more abstract, making it harder for the models to find the features in the data that can indicate that a text is about that subject. Finally, a comparison is done specifically between the Naive Bayes model and the SVM model because previous literature had found different results concerning which model could outperform the other when presented with a class with a very low number of samples. The F1-scores in Table 7 indicate that for classes with less than 10 samples, the Naive Bayes model consistently outperforms the SVM model, thus supporting the findings by Forman and Cohen (2004) and disproving the results of Yang and Liu (1999).

6.2 Inspecting the baseline models

6.2.1 What the models learn

To better understand how the baseline models classified the different categories, a closer look is taken at each of the base classifiers for a number of categories with different samples sizes. The selected categories for this inspection are *Noise*, *Vibrations*, and *Health*.

Decision Tree
For the Decision Tree, the classification decisions can be inspected by looking at the tree's feature splits that decide whether a data point is part of a specific class or not. These decisions can be visualised using a tree plotting function from the same Python library that provided the trainable Decision Tree model function. An example of such a tree can be seen in Figure 6. When looking at the trees of the *Noise* category, it can be seen that it depends on the tree which words are used to split the data, although terms like 'geluid' (noise), 'geluidsbelasting' (noise pollution), and 'geluidsoverlast' (noise disturbance) are often used as determining features higher up in the tree. Other models use seemingly arbitrary words like 'waarom' (why) or 'daarnaast' (beside) to split the data higher up, though these trees often score lower on the F1-metric than the trees that focused more noise-related words. For the category *Vibrations*, the visualisation of the Decision Tree shows that the models often overfit on the data, resulting in trees that often use more specific terms such as 'goederenvervoer' (freight transport) and 'overschrijding' (exceeding) to split the data, and sometimes even use names of provinces in the Netherlands or specific street names for the splits. The terms 'trilling' (vibration) and 'trillingen' (vibrations) turned out to occur in the trees less frequent than expected. For the *Health* category, most terms used for splits did not appear related to the category, with exceptions such as 'straling' (radiation) and 'riolering' (sewerage) for which a case can be made that they link to the subject of health. There was no use of the term 'gezondheid' (health) in any of the trees, which was surprising.

After inspecting the Decision Tree models further, it appeared that the hyperparameter that determines the maximum number of features considered when choosing the best split was quite influential in the performance of the model on different categories. The inspection showed that if there was no maximum number of considered features, the trees of the categories with more samples (*Noise* and *Vibrations*) were much more explainable, used more obvious features to determine the splits, and overall got a higher F1-score, whereas for a smaller category like *Health*, the model performed consistently worse when not using a maximum number of features. It should thus be noted that the results shown in Table 7 could have been different if different hyperparameters were chosen.

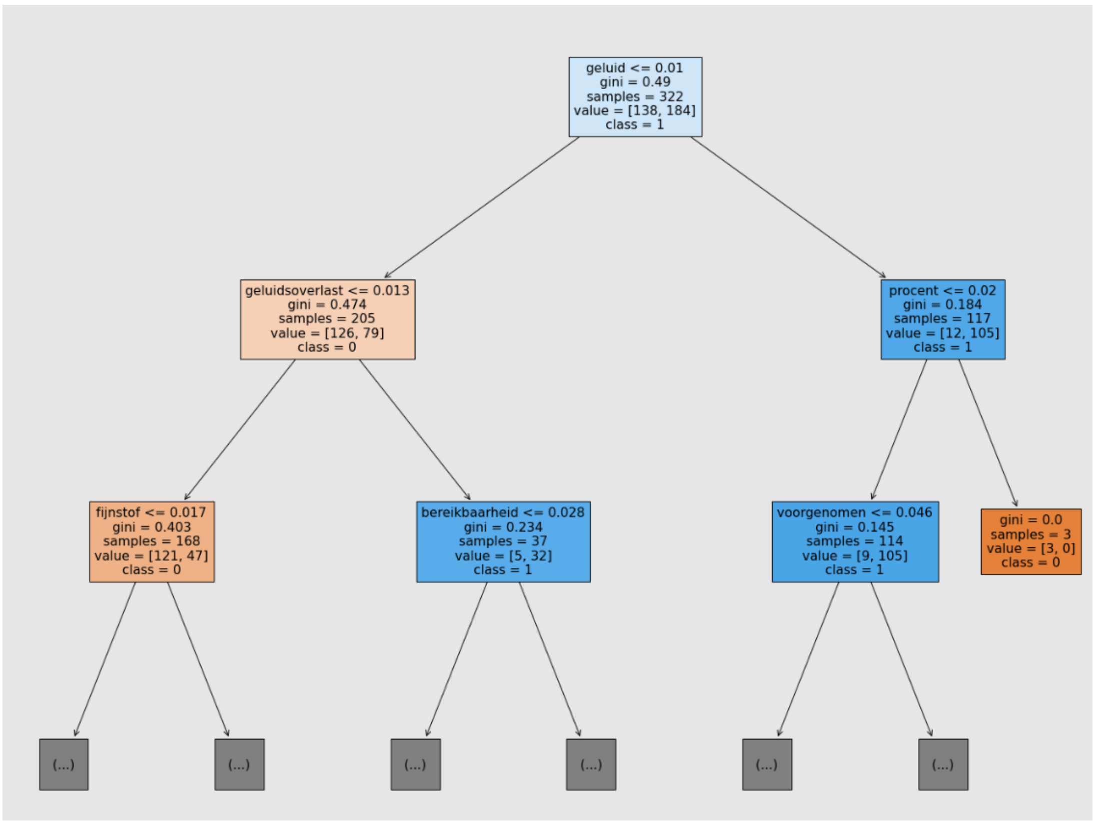
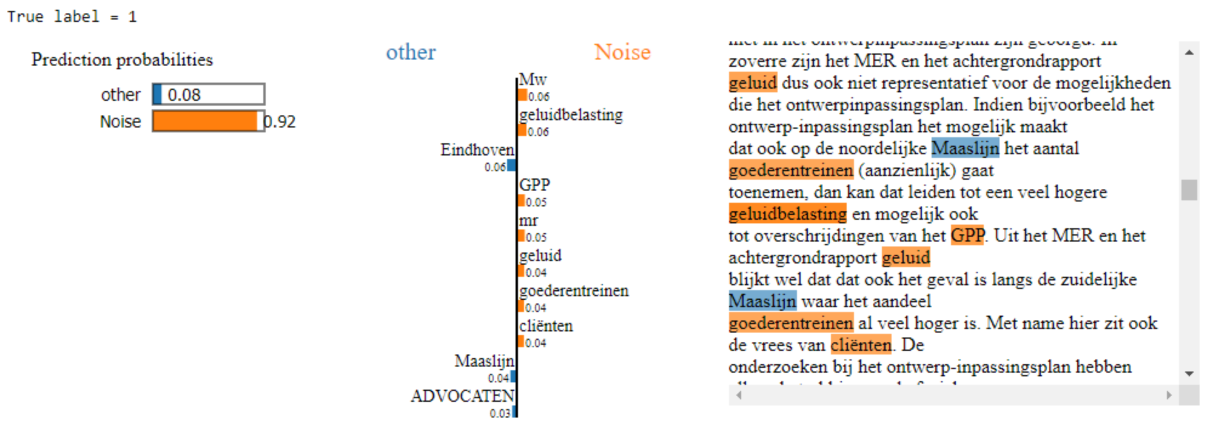


Figure 6: A snippet from a visualised Decision Tree model trained on the category *Noise*. The top row in each box represents a word in the dataset and the tf-idf score that a document needs to exceed for that word in order to be moved to the left node (more probable to be about noise) or right node (less probable to be about noise). After the feature word, the Gini impurity score and the number of samples at that node are shown. The *value* parameter shows the distribution of the classes in the sample, where the first value represents class 0 (not noise) and the second value represents class 1 (noise). Finally, the majority class of that node is named. The colour of the box represents whether the majority class of that node is not noise (orange) or noise (blue).

Naive Bayes

Unlike the Decision Tree classifier, the Naive Bayes classifier does not have a built-in way of visualising the decisions made by the model. To still be able to get some insights in the decisions made, a model visualisation tool called LIME was used. Originally proposed by Ribeiro et al. (2016), LIME is described by them as "an algorithm that can explain the predictions of any classifier or regressor in a faithful way, by approximating it locally with an interpretable model." (p. 1). For the text classification task at hand, LIME is used to visualise the decisions of the Naive Bayes model by looking at a single test case and showing the prediction probabilities of the classes, the terms used to decide which class is more likely, and the original text with the deciding terms highlighted in shades of orange for positive predictive words and blue for negative predictive words. An example of such a visualisation can be found in Figure 7.



zoverre zijn het MER en het achtergrondrapport
geluid dus ook niet representatief voor de mogelijkheden
die het ontwerpinpassingsplan. Indien bijvoorbeeld het
ontwerp-inpassingsplan het mogelijk maakt
dat ook op de noordelijke Maaslijn het aantal
goederentreinen (aanzienlijk) gaat
toenemen, dan kan dat leiden tot een veel hogere
geluidbelasting en mogelijk ook
tot overschrijdingen van het GPP. Uit het MER en het
achtergrondrapport geluid
blijkt wel dat dat ook het geval is langs de zuidelijke
Maaslijn waar het aandeel
goederentreinen al veel hoger is. Met name hier zit ook
de vrees van cliënten. De
onderzoeken bij het ontwerp-inpassingsplan hebben

Figure 7: A visualisation of the decision of a Naive Bayes model trained on the category *Noise* using LIME. The image shows the prediction probabilities for each class, the features that had the most influence on the probability score of the different classes, and the original text with the influential words highlighted in orange for the noise class and blue for the non-noise class.

When inspecting the model trained on the category *Noise*, it can be seen that the model uses direct noise-related terms like ‘geluid’ (noise), ‘geluidsscherm’ (noise barrier), ‘geluidswerend’ (soundproof), and ‘geluidshinder’ (noise pollution) as well indirectly related terms like ‘goederentreinen’ (freight trains) and ‘bouwterrein’ (construction site) to decide whether a document discusses the subject of noise. As with the Decision Tree model, the Naive Bayes model appears to use seemingly arbitrary words as well for the classification, although this does not seem to negatively influence the classification capabilities of the model enough to have other models outperform it on this category. For the *Vibrations* category, it is found that the documents about vibrations are often detected because the model uses terms such as ‘trillingen’ (vibrations) and ‘trillingshinder’ (hindrance caused by vibrations) for detection. Related words like ‘scheurvorming’ (cracking) and ‘goederentreinen’ (freight trains) were also used. Interestingly, it was found that for certain documents, the words ‘under’, ‘sleeper’, and ‘pads’ were used for detection. Since *under sleeper pads* are a type of pad used to absorb train vibrations with the goal of causing less hindrance for local residents, this indicates that the model can find more specific patterns in the data even though only unigrams are considered when creating the document vectors. When inspecting the *Health* category, it seems that although there are quite some texts about health classified correctly, the LIME visualistation does not show that terms like ‘gezondheid’ (health) and ‘gezondheidsproblemen’ (health problems) are indicative of the category *Health*, which seems remarkable. It is thus not clear from the selected indicative terms how the model classified the texts.

SVM

For the SVM model, the same approach as the one for the visualisation of the Naive Bayes model choices was used, namely the LIME algorithm. For the model trained on the *Noise* category, the model found the right classification often, but does not always seem to use terms directly related to noise to classify the documents. Although in one document the term 'geluidsbelasting' (noise pollution) appears to be used, the term 'geluidsscherm' (noise barrier) is not. Similarly, there was a document that contained the words for noise, noise barrier, and noise disturbance, but LIME indicated that terms like 'onjuist' (incorrect) and 'construction site' were the terms most indicative of the *Noise* category. Although this phenomenon occurred sometimes, more often the algorithm indicated that noise-related terms were used for the classification. When looking at the *Vibrations* category, it can be seen that terms like 'trillingen' (vibrations), 'trillingshinder' (hindrance caused by vibrations) and 'goederenvervoer' (freight transport) are used as important terms for classification. For texts where the classifier classified them as not being about the subject of vibrations when they actually were, it looks like the most indicating terms like 'trillingen' only occurred once, which would explain why the classifier was not confident enough to classify the text as being about vibrations. Finally, for the *Health* category, it is not clear how the model correctly classified some of the samples, since the terms that are deemed most important by the LIME algorithm are terms like 'horizonvervuiling' (visual pollution), 'voorgoed' (forever), 'trillingschade' (vibration damage), and 'pilaren' (pillars) , which do not seem to be related to health. This either indicates that the classifier finds patterns in the data that are not related to health due to the small data size, or it indicates that the LIME visualisation does not correctly show the most important terms used for classifying a data sample.

6.2.2 Reweighting and resampling

As was explained, the baseline models were trained without any changes as well as with modifications to the model or dataset in order to test whether reweighting of the classes and resampling of the data could help with alleviating the problems caused by having a small and imbalanced dataset. After inspecting the Tukey HSD tests done between the different versions of the Decision Tree model, it was found that there was no significant difference between the macro F1-scores of the base Decision Tree model, the Decision Tree model with reweighted classes, and the Decision Tree model trained on resampled data. For the micro F1-scores, a significant difference ($p<.05$) was only found between the reweighted and resampled Decision Tree models. For the Naive Bayes models and SVM models, no significant difference was found between their base version of the model and the model trained with resampled data. When inspecting the F1-scores per category found in Table 7, it indeed appears that most of the scores per model type are quite similar, with some more variation between model types for certain classes.

To get more insight into the effect of using reweighting and resampling on different amounts of data, a closer look will be taken at the results of classes with 0 to 30 samples, classes with 31 to 60 samples, and classes with more than 60 samples. This comparison will be done for each of the baseline classifier types. For the Decision Tree model, the base model will firstly be compared with the model with reweighted classes. For the categories with less than 30 samples, only a few categories got higher F1-scores after reweighting, but the scores for two of those categories, *Smart mobility and innovation* and *Use of space*, did increase from 0.0 to 0.37 and 0.47 respectively. For categories with 31 to 60 samples, both the base model and the reweighted model slightly outperformed the other in some of the categories, thus making it unclear which type of model would be preferred with that amount of data. For the categories with more than 60 samples, the base model more often scored a higher F1-score than the reweighted model, although the differences were again not that large. When comparing the base Decision Tree model with the model trained with resampled data, both the categories with less than 30 samples and more than 60 samples have very mixed results when it comes to which model performs better, while for the categories with 31 to 60 samples the model with resampled data appeared to outperform the base model on almost all categories. Although both techniques to alleviate class imbalance did not consistently outperform the base model, the fact that the model trained with resampled data was more effective than the reweighted model matches the findings by Seiffert et al. (2008). For the Naive Bayes models, the models performed similarly on categories with more than 60 samples, while for the categories with less than 60 samples, the base Naive Bayes model more often got higher F1-scores than the Naive Bayes model with resampled data. When comparing the base SVM model and the resampled SVM model, it can be seen that for all sample sizes, the base SVM model either outperforms the resampled SVM model or scores just below the resampled SVM model's F1-score.

When looking at the overall performance of the base models and their reweighted and resampled versions on the different categories, only the resampled Decision Tree model more often outperformed the base Decision Tree model, with the resampled Decision Tree scoring higher on 60% of the categories for which a score higher than 0.0 was obtained by one of the model versions. The results did not show that any of the reweighted or resampled models could completely or at least majorly (75%) outperform the base models, but it was also not found that the reweighted or resampled models never showed any improved performance compared to the base models. Because of this, it can be concluded that there is no conclusive evidence that shows that the reweighting of classes or the resampling of data are helpful tools for consistently overcoming the problem of having a small and imbalanced dataset for a dataset with many classes of different sizes. Since the effectiveness of the different techniques differed per category, the usefulness of the techniques should not be dismissed, and a practical recommendation would be to do more research on the subject of why the techniques work well or less well.

6.3 Inspecting the multi-label BERT model performance

Contrary to expectations, the multi-label BERT model did not outperform the baseline classifiers with respect to the macro and micro F1-scores, as was shown in the analysis of Section 6.1. Different steps were taken to test why the language model was not able to get higher F1-scores than the baselines, which will be discussed in this section.

6.3.1 Testing the model

Initially, the multi-label BERT model was trained on a small number of epochs, which resulted in F1-scores under the baseline scores. To ensure that the model did not need more epochs to eventually outperform the baseline models, the learning curve of the model was plotted to see whether the train and validation loss could still decrease more when the model was trained longer. Figure 8 shows that there is little decrease in the validation loss after the first 10 epochs, and when inspecting the F1-scores that the model got each epoch it appeared that after 15 epochs, the increase of the scores flattened.

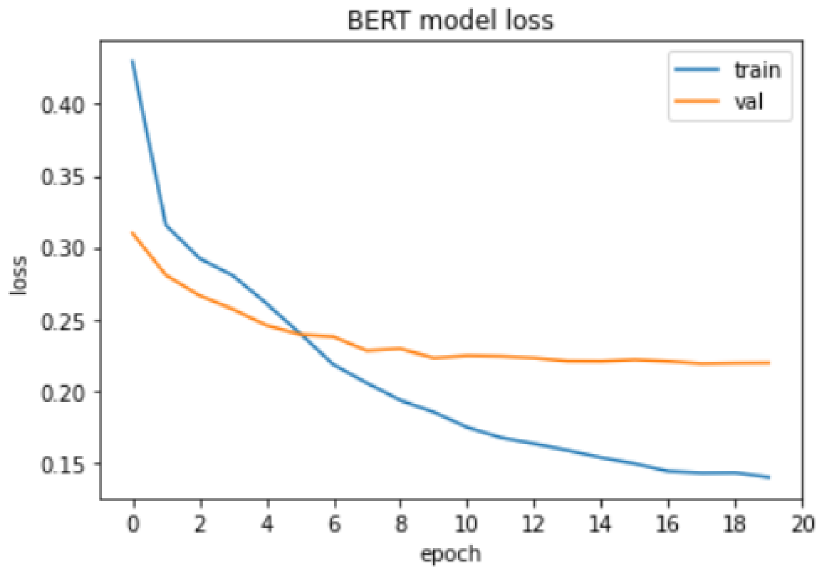


Figure 8: A plot of the training and validation loss showing the learning curve of the multi-label BERT model.

After this test, it was suspected that the multi-label BERT implementation could not outperform the baseline models due to the data being multi-labelled, thus making it too hard for the model to distinguish the different classes from one another and learn any pattern in the data. To test this, the one-vs-rest approach to fine-tuning BERT was implemented to see whether training individual classes on BERT models could simplify the classification problem, resulting in a higher model performance. The F1-scores presented in Figure 5 showed that although the one-vs-rest BERT model obtained a slightly higher mean macro F1-score than the multi-label BERT model, the mean micro F1-score of the one-vs-rest BERT model was actually significantly lower. Additionally, the one-vs-rest BERT model was not able to outperform any of the baseline models, and even had the lowest mean micro F1-score of all the trained models. This could have been caused by the low number of epochs that the models were trained on. The same number of epochs was used for all classes, which was determined by looking at the performance of classes with different amounts of data. Although this low number of epochs resulted in a better model performance for the smaller classes, it caused the larger classes to score lower, ultimately resulting in a low micro F1-score.

Finally, to test whether the relatively low F1-scores of the multi-label BERT model were caused by the small data size and not by some error in the implementation process, a test was done with a similar dataset. This dataset, the MultiEURLEX dataset, was trained on the same BERT model to see how it would perform on a multi-label dataset if more data was provided. The results in Table 8 show that after training the model for a single epoch, it is clear that more data already improves both the macro and micro F1-scores. An additional training run of the model on 6000 data samples showed that if the number of epochs was increased to 20, the model could reach a macro F1-score of 0.65 and a micro F1-score of 0.82. It can thus be concluded that the multi-label implementation of the BERT model can certainly work if enough data is provided and the model is trained for enough epochs.

Table 8: An overview of the macro and micro F1-scores of the models trained for 1 epoch on different amounts of data from the Dutch MultiEURLEX dataset.

Number of samples	Macro F1-score	Micro F1-score
400	0.15	0.48
6000	0.31	0.65
10000	0.41	0.70
30000	0.60	0.80
65000	0.71	0.84

6.3.2 Other explanations

The question remains why the baseline classifiers outperformed the multi-label BERT model. One explanation could be that the classification of texts in the categories of the given dataset is a type of problem that can be solved more easily when a simpler vectorisation technique and model are used. Given that quite some of the categories have very straightforward terms that can indicate whether the document is about that subject or not, the contextual information that the BERT model provides might not make the problem easier to solve and might even unnecessarily complicate the classification problem. Another explanation could be that the truncation of the documents before the training of the BERT models caused a significant loss in valuable information for the model, which made it impossible for the model to find a pattern in the features of the data. Although the training of the model on the MultiEURLEX dataset showed that increasing the amount of data could result in higher F1-scores even when the data was truncated, it cannot be said how much better the model would perform if a technique was implemented that would enable the model to use all the information in the longer documents. Since these explanations could not be tested in this thesis, it should thus be concluded that more research is needed to find out whether a language model can actually outperform baseline models on a text classification task.

7 Discussion

This thesis focused on the topic of text classification to get more insight into how effective text classification could be achieved for certain types of datasets, and in this final chapter, the insights that were gained in this project will be discussed, as well as the gaps in the knowledge that still remain. In Section 7.1, the main research question and its sub-questions will be answered, after which Section 7.2 will discuss the limitations of the research. Finally, Section 7.3 will suggest some future research options.

7.1 Conclusion

The main question posed in this thesis was how effective text classification could be achieved for small and imbalanced datasets with long texts, which was then divided into four sub-questions. These sub-questions will be answered and discussed, after which the answer to the main research question will be discussed.

Firstly, the question was raised *which aspects of a dataset limit the performance of simple classification models, and how these limitations can be mitigated.* After studying the literature on text classification using simple models such as the Decision Tree and Naive Bayes model, it was found that the two main limiting data aspects are the size of the dataset and an imbalance in the data classes. To mitigate the effect of these properties on the ability of the model to learn the right patterns in the data, two techniques were found: reweighting and resampling. The reweighting of the classes is a process that causes the model to be penalised more for misclassifying classes with less data, thus solving the issue of having an imbalanced dataset. The resampling of the data is a process in which data points from the minority class are used to create new, similar data points for that class such that the number of samples for the minority class is made to be equal to the number of samples in the majority class, which would have a positive effect on both the data size and the problem of the imbalanced data. Both techniques were implemented in this research to test whether they could improve the performance of different simple classification models. However, it was found that although the techniques caused minor improvements for some of the categories in the dataset, there was no consistently positive effect of the techniques on the performance of the different models.

Secondly, the question was raised *which aspects of a dataset limit the performance of classifiers based on language models, and how these limitations can be mitigated.* The main limitations found after the literature review were the resources available for training a model in a language other than English and the fact that some language models have a restriction on the input length of the documents, which would be an issue for the provided dataset since more than half of its documents exceeded the maximum document length of the chosen language model. For the issue of the language resources, it was found that there were multiple available models that performed well on a selection of language processing tasks and were trained on Dutch texts, which was the language of the dataset provided for this research. To solve the problem of having documents that were too long for the language model, different solutions were found in the literature, such as different truncation techniques, chunking the documents and using the separate chunks as input for a model, summarising the documents to shorten them, and using a Longformer architecture that modifies the language model in such a way that it can handle longer documents. Although the Longformer approach was deemed the most promising way of solving the issue of having long documents in the dataset, the simplest truncation technique was implemented in the end due to time constraints making it impossible to implement Longformer. Because of this, no clear conclusion can be drawn on which technique should be used to mitigate the limitations of having long documents when using a language model for text classification. However, what can be said is that the truncation of the texts worked well enough that the model was able to classify the larger classes almost as effectively as the baseline models.

Thirdly, the question was raised *to what extent a classification approach using a language model can yield better results than a simple text classification approach, given a dataset with different limitations influencing the performance of both classification types.* To answer this question, an experiment was set up to train a selection of different simple classifiers, as well as a language model, to see how they would handle the dataset provided for this research. After analysing the results, it was found that with the implementations of the models used in this research, the BERT language model was not able to outperform any of the simple baseline classifiers on the text classification task it was fine-tuned on. The Naive Bayes baseline model got the highest F1-scores on the most classes in the dataset and was one of the models that classified the most data points correctly. When inspecting the model performance per category in the dataset, it appeared that the language model performance was relatively close to the baseline models for the larger categories, whereas for the smallest categories the language model performed very poorly compared to the baseline models. One question that remains is whether the performance of the language model could be improved if the strategy for handling long documents was changed from truncation to using Longformer or a different approach.

Finally, the question was raised *to what extent certain data aspects have a larger influence on the model performance than others, and whether this can be explained by inspecting the data and the functioning of the models.* The different aspects that the given dataset had and thus the aspects that were inspected in this research were having a small dataset, having a class imbalance in the dataset, having multi-label data, having data in Dutch, and having data that partially consists of very long documents. From the results of the experiments executed in this project, it could be seen that for all of the models were able to consistently learn the patterns found in the classes with the largest amount of data, whereas the models had very mixed results when trained on classes with little data. It can thus be concluded that the data size has a large influence on the performance of both language models and simpler classification models. Although it was theorised that the language model could use its pre-trained language properties to more easily classify the texts, the size of the dataset appeared to be too large a factor to get the model to work effectively. The imbalance of the classes in the data appeared to be less of an explicit problem since the reweighting of classes, which is used to only alleviate the imbalance and not the data size, was not shown to improve the performance of the baseline models. The fact that the data was multi-label was more of a specific data feature that influenced the way the models had to be trained rather than an attribute that directly influenced the model performance. However, for the language models specifically, both a one-vs-rest approach and a direct multi-label approach were used for training the model, and from the results of those models it could be seen that the one-vs-rest implementation of the language model more often yielded slightly better results, indicating that having a model learn patterns in a multi-label dataset makes it slightly harder to differentiate the classes when data is scarce. The question remains whether this difference is also found when enough data is provided for all the classes. The data attribute of having texts in Dutch instead of English was not a problem in the end, since enough different resources for both the preprocessing of the data for the baseline models and the training of a Dutch language model existed. Finally, the influence of the text length of the documents in the dataset could not be tested for the language models, which means no clear conclusion can be drawn about how that influences the performance of that type of model. However, for the baseline models the text length was not a problem since only the word counts were used for the vectorisation of the documents.

Coming back to the main research question, it can be concluded that effective text classification can partially be achieved for small and imbalanced datasets with long texts by using a simple classification model, since it depends on the specific content of the class and its amount of data how well a model can perform. For the Naive Bayes model, the top performing model, a mean macro F1-score of 0.38 and a mean micro F1-score of 0.52 was found. These relatively low scores were explained by the fact that many of the small classes in the dataset could not be learned by the model, leading to F1-scores between 0.0 and 0.2 and thus lowering the overall model performance scores. However, even from the very small classes, certain baseline models were able to correctly classify multiple documents, indicating that if the data itself is simple enough, patterns can be found quickly by models and with enough tuning of the models, a good performance score is possible.

7.2 Limitations

To understand the significance of the results and conclusions found in this research, it is important to reflect on the limitations that different parts of the project execution have. These limitations will concern two main subjects, namely the creation of the dataset and the tuning of the models.

7.2.1 Dataset

A number of remarks about the quality of the dataset need to be made. Firstly, as was discussed in Section 2.3, the complete documents from one of the provided projects could not be matched to their labels, which meant that the separate texts of the points of objection belonging to one letter needed to be appended in order to recreate the original letters. This resulted in 213 appended letters, which did not contain the same noise present in the letters from the other projects. Additionally, for some of the letters in the Maaslijn project, the original documents were modified slightly by removing the pages that only contained images and no actual text because the used text detection algorithm could not handle images well and translated it to long strings of arbitrary characters.

This modification removed some of the noise in the dataset as well. Furthermore, for the labelling of the data, it was assumed that the labels provided in the documents with the letters of objection were all present and correct. Due to time constraints, this could not be checked manually, which means that there is no guarantee that the labels of every data point were correct. Moreover, during the remapping of the labelling of the letters, some remappings had to be done manually, and although the more uncertain mappings were checked with the domain experts, it could still have been the case that some errors were made. Finally, there was a large number of project-specific terms in the documents, for example terms related to a certain highway, a city, a neighbourhood, or a street name. Initially it was assumed that the classification models would not use those terms for determining whether a document was about a certain subject since these terms occurred in many documents, but after more closely inspecting the reasons behind the decisions made by specifically the baseline models, it could be seen that even very specific names and terms were used for the decision making. This could be prevented in the future by filtering out project-specific words. In conclusion, although ideally the models can still find the right patterns in the data when more noise is added, and although it should be possible to say things about the classification of letters of objection based on the created dataset, it should still be noted that the dataset might not be as representative of a collection of letters of objection as it could have been, and that classification of future documents using the trained classifiers could turn out not to be as reliable as desired.

7.2.2 Hyperparameter tuning

For the training of the models, the general limitation that can be named is that due to the large number of categories and thus the large number of models, there were many different parameters that were tuned per model type instead of per category. Because the different categories each had a specific number of samples and had their own data properties, the tuning per model type caused the performance of the models to vary greatly per category depending on which parameters were used. This could be seen in the baseline models when one model parameter of the Decision Tree classifier caused the score of a category with more data to increase greatly, while the score for a category with less data decreased instead. Another example is that for the training of the multi-label language model, it was decided that one decision threshold for the classification of the different classes would be used, while if the time had been available to implement it, a threshold per category could have resulted in higher performance scores overall. Similarly, for the training of the one-vs-rest language model implementation, a specific number of epochs was used to train each of the categories, while some categories performed better after more training and others performed better after less training. The decisions to use hyperparameters that were shown to work well in general was made with the interest of time in mind, since tuning the hyperparameters of every model for every category was just not feasible, but it still is a limitation since training the model with more specific hyperparameters could lead to different outcomes of the research.

7.3 Future research

Given that there are still some questions left unanswered by this thesis, a number of recommendations for future research can be listed. Firstly, a question that could not be answered in the research due to time restrictions was related to the fact that the documents in the dataset could be very long and that the language model used in this research had a limit on the input length of the documents. Different solutions to this problem were posed, with the implementation of the Longformer architecture being the most promising one. One line of research could focus on testing whether the implementation of this architecture or any of the other solutions could improve the performance of the language model used in this research, and whether this improvement would cause the language model to outperform any of the baseline models. Next to that, for the baseline models, there are still many factors such as the choice of vectorisation technique, the use of certain text preprocessing techniques such as lemmatisation, and the hyperparameter tuning of the models that can all be explored more to see whether making any changes to them influences the performance of the models. In general, more data collection is recommended, either by trying to recreate letters of objection or by collecting them from other infrastructural projects, and see how the increase in samples influences the models' performance in any way.

In a broader context, one aspect of the text classification problem that could be explored more is whether a different type of classifier, for example a rule-based classifier, could work better on the given dataset. Since many of the categories appeared to have very clear terms describing their subject, a rule-based approach could work especially well if well-written rules are defined, and given that such a classifier does not need to be trained on a dataset, this would circumvent the problems caused by having a small dataset. Additionally, an advantage of using a rule-based classifier would be that the outcomes are very explainable since the rules for each class are written explicitly and can be inspected if one wants to understand why a certain class was assigned to a certain document. One could also look into the possibility of training a language model on a large collection of more domain-specific documents, which in this case would mean a collection of letters of objection. Since many words specific to the subject of trains and railways were present in the dataset, it might have been the case that the vectorisation of the data by the used Dutch pre-trained language model caused some of the more domain-specific words to be left out, which would have been prevented if a more specifically trained vectoriser was used.

References

Anderson, J. D. and Pérez-Carballo, J. (2001). The nature of indexing: how humans and machines analyze messages and texts for retrieval. part i: Research, and the nature of human indexing, *Information processing & management* **37**(2): 231–254.

Beltagy, I., Peters, M. E. and Cohan, A. (2020). Longformer: The long-document transformer, *arXiv preprint arXiv:2004.05150* .

Bender, E. M. and Koller, A. (2020). Climbing towards NLU: On meaning, form, and understanding in the age of data, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Online, pp. 5185–5198.
URL: *https://aclanthology.org/2020.acl-main.463*

Blei, D. M., Ng, A. Y. and Jordan, M. I. (2003). Latent dirichlet allocation, *the Journal of machine Learning research* **3**: 993–1022.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D. (2020). Language models are few-shot learners.

Canini, K., Shi, L. and Griffiths, T. (2009). Online inference of topics with latent dirichlet allocation, *Artificial Intelligence and Statistics*, PMLR, pp. 65–72.

Chalkidis, I., Fergadiotis, M. and Androutsopoulos, I. (2021). Multieurlex–a multilingual and multi-label legal document classification dataset for zero-shot cross-lingual transfer, *arXiv preprint arXiv:2109.00904* .

Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique, *Journal of artificial intelligence research* **16**: 321–357.

Colas, F. and Brazdil, P. (2006). Comparison of svm and some older classification algorithms in text classification tasks, *IFIP International Conference on Artificial Intelligence in Theory and Practice*, Springer, pp. 169–178.

Cortes, C. and Vapnik, V. (1995). Support-vector networks, *Machine learning* **20**(3): 273–297.

De Clercq, O., De Bruyne, L. and Hoste, V. (2020). News topic classification as a first step towards diverse news recommendation, *Computational Linguistics in the Netherlands Journal* **10**: 37–55.

de Vries, W. and Nissim, M. (2020). As good as new. how to successfully recycle english gpt-2 to make models for other languages.

de Vries, W., van Cranenburgh, A., Bisazza, A., Caselli, T., van Noord, G. and Nissim, M. (2019). Bertje: A dutch bert model, *arXiv preprint arXiv:1912.09582* .

de Vries, W., van Cranenburgh, A., Bisazza, A., Caselli, T., van Noord, G. and Nissim, M. (2021). Gronlp/bert-base-dutch-cased · hugging face.
URL: *https://huggingface.co/GroNLP/bert-base-dutch-cased*

Delobelle, P., Winters, T. and Berendt, B. (2020). Robbert: a dutch roberta-based language model, *arXiv preprint arXiv:2001.06286* .

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Association for Computational Linguistics, Minneapolis, Minnesota, pp. 4171–4186.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms, *Neural computation* **10**(7): 1895–1923.

Do, C. B. and Ng, A. Y. (2005). Transfer learning for text classification, *Advances in neural information processing systems* **18**.

Efron, B. (1983). Estimating the error rate of a prediction rule: improvement on cross-validation, *Journal of the American statistical association* **78**(382): 316–331.

Forman, G. and Cohen, I. (2004). Learning from little: Comparison of classifiers given little training, *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, pp. 161–172.

Girden, E. R. (1992). *ANOVA: Repeated measures*, number 84, Sage.

Hadi, W., Al-Radaideh, Q. A. and Alhawari, S. (2018). Integrating associative rule-based classification with naïve bayes for text classification, *Applied Soft Computing* **69**: 344–356.

Hayes, P. J. and Weinstein, S. P. (1990). Construe/tis: A system for content-based indexing of a database of news stories., *IAAI*, Vol. 90, pp. 49–64.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (2nd Edition)*, Prentice-Hall, Inc., USA.

Ling, C. X. and Li, C. (1998). Data mining for direct marketing: Problems and solutions., *Kdd*, Vol. 98, pp. 73–79.

Liu, X., He, P., Chen, W. and Gao, J. (2019). Multi-task deep neural networks for natural language understanding, *arXiv preprint arXiv:1901.11504* .

Lu, Z., Zhu, Y., Pan, S., Xiang, E., Wang, Y. and Yang, Q. (2014). Source free transfer learning for text classification, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

McKinsey (2020). The imperatives for automation success.
URL: *https://www.mckinsey.com/business-functions/operations/our-insights/the-imperatives-for-automation-success*

Mihalcea, R. (2005). Language independent extractive summarization, *ACL*, Vol. 5, pp. 49–52.

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013). Efficient estimation of word representations in vector space.

Mutasodirin, M. A. and Prasojo, R. E. (2021). Investigating text shortening strategy in bert: Truncation vs summarization, *2021 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, IEEE, pp. 1–5.

Nallapati, R., Zhai, F. and Zhou, B. (2017). Summarunner: A recurrent neural network based sequence model for extractive summarization of documents, *Thirty-First AAAI Conference on Artificial Intelligence*.

Nangia, N. and Bowman, S. R. (2019). Human vs. muppet: A conservative estimate of human performance on the glue benchmark, *arXiv preprint arXiv:1905.10425* .

Nigam, K., McCallum, A. K., Thrun, S. and Mitchell, T. (2000). Text classification from labeled and unlabeled documents using em, *Machine learning* **39**(2): 103–134.

Nikolenko, S. I., Koltcov, S. and Koltsova, O. (2017). Topic modelling for qualitative studies, *Journal of Information Science* **43**(1): 88–102.

Pal, A., Selvakumar, M. and Sankarasubbu, M. (2020). Multi-label text classification using attention-based graph neural network, *arXiv preprint arXiv:2003.11644* .

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. and Zettlemoyer, L. (2018). Deep contextualized word representations.

Rayner, K., Schotter, E. R., Masson, M. E., Potter, M. C. and Treiman, R. (2016). So much to read, so little time: How do we read, and can speed reading help?, *Psychological Science in the Public Interest* **17**(1): 4–34.

Refaeilzadeh, P., Tang, L. and Liu, H. (2016). Cross-validation, encyclopedia of database systems (pp. 1–7).

Rennie, J. D., Shih, L., Teevan, J. and Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers, *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 616–623.

Ribeiro, M. T., Singh, S. and Guestrin, C. (2016). "why should i trust you?" explaining the predictions of any classifier, *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144.

Roul, R. K., Sahoo, J. K. and Arora, K. (2017). Modified tf-idf term weighting strategies for text categorization, *2017 14th IEEE India council international conference (INDICON)*, IEEE, pp. 1–6.

Santos, M. S., Soares, J. P., Abreu, P. H., Araujo, H. and Santos, J. (2018). Cross-validation for imbalanced datasets: avoiding overoptimistic and overfitting approaches [research frontier], *ieee ComputatioNal iNtelligeNCe magaziNe* **13**(4): 59–76.

Sapunov, G. (2019). Speeding up bert.
URL: *https://blog.inten.to/speeding-up-bert-5528e18bb4ea*

Sarne, D., Schler, J., Singer, A., Sela, A. and Bar Siman Tov, I. (2019). Unsupervised topic extraction from privacy policies, *Companion Proceedings of The 2019 World Wide Web Conference*, pp. 563–568.

Sebastiani, F. (2002). Machine learning in automated text categorization, *ACM computing surveys (CSUR)* **34**(1): 1–47.

Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J. and Napolitano, A. (2008). Resampling or reweighting: A comparison of boosting implementations, *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, Vol. 1, IEEE, pp. 445–451.

Semberecki, P. and Maciejewski, H. (2016). Distributed classification of text documents on apache spark platform, *International Conference on Artificial Intelligence and Soft Computing*, Springer, pp. 621–630.

Semberecki, P. and Maciejewski, H. (2017). Deep learning methods for subject text classification of articles, *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, IEEE, pp. 357–360.

Semwal, T., Yenigalla, P., Mathur, G. and Nair, S. B. (2018). A practitioners' guide to transfer learning for text classification using convolutional neural networks, *Proceedings of the 2018 SIAM international conference on data mining*, SIAM, pp. 513–521.

Short, M. (2019). Text mining and subject analysis for fiction; or, using machine learning and information extraction to assign subject headings to dime novels, *Cataloging & Classification Quarterly* **57**(5): 315–336.

Sorower, M. S. (2010). A literature survey on algorithms for multi-label learning.

Su, X., Miller, T., Ding, X., Afshar, M. and Dligach, D. (2021). Classifying long clinical documents with pre-trained transformers, *arXiv preprint arXiv:2105.06752* .

Sun, A., Lim, E.-P. and Liu, Y. (2009). On strategies for imbalanced text classification using svm: A comparative study, *Decision Support Systems* **48**(1): 191–201.

Sun, C., Qiu, X., Xu, Y. and Huang, X. (2019). How to fine-tune bert for text classification?, *China National Conference on Chinese Computational Linguistics*, Springer, pp. 194–206.

Suominen, A. and Toivanen, H. (2016). Map of science with topic modeling: Comparison of unsupervised learning and human-assigned subject classification, *Journal of the Association for Information Science and Technology* **67**(10): 2464–2476.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2017). Attention is all you need.

Wainer, J. and Cawley, G. (2021). Nested cross-validation when selecting classifiers is overzealous for most practical applications, *Expert Systems with Applications* **182**: 115222.

Walkowiak, T., Datko, S. and Maciejewski, H. (2018). Feature extraction in subject classification of text documents in polish, *International Conference on Artificial Intelligence and Soft Computing*, Springer, pp. 445–452.

Wan, L., Papageorgiou, G., Seddon, M. and Bernardoni, M. (2019). Long-length legal document classification, *arXiv preprint arXiv:1912.06905* .

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O. and Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding, *CoRR* .

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M. et al. (2019). Huggingface's transformers: State-of-the-art natural language processing, *arXiv preprint arXiv:1910.03771* .

Yang, Y. and Liu, X. (1999). A re-examination of text categorization methods, *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 42–49.

A Appendix

Table 9: Mean macro and micro F1-scores per model type.

(a) Macro F1-scores of each model type.

	Decision Tree			**Naive Bayes**		**SVM**		**BERT**	
Fold	base	reweight	resample	base	resample	base	resample	1-vs-rest	multi
1	0.26	0.24	0.28	0.39	0.32	0.27	0.23	0.26	0.22
2	0.25	0.24	0.30	0.38	0.35	0.36	0.27	0.26	0.22
3	0.23	0.21	0.27	0.37	0.35	0.26	0.20	0.33	0.26
4	0.25	0.29	0.27	0.34	0.31	0.28	0.24	0.27	0.22
5	0.26	0.25	0.27	0.42	0.46	0.37	0.31	0.28	0.25
Mean	0.25	0.25	0.28	0.38	0.36	0.31	0.25	0.28	0.23
Std	0.01	0.03	0.01	0.03	0.06	0.05	0.04	0.03	0.02

(b) Micro F1-scores of each model type.

	Decision Tree			**Naive Bayes**		**SVM**		**BERT**	
Fold	base	reweight	resample	base	resample	base	resample	1-vs-rest	multi
1	0.41	0.39	0.42	0.51	0.53	0.51	0.51	0.33	0.52
2	0.38	0.35	0.41	0.52	0.52	0.48	0.44	0.32	0.48
3	0.38	0.35	0.46	0.53	0.53	0.49	0.48	0.38	0.52
4	0.42	0.39	0.40	0.52	0.53	0.53	0.50	0.38	0.53
5	0.42	0.40	0.42	0.54	0.61	0.59	0.57	0.30	0.52
Average	0.40	0.38	0.42	0.52	0.54	0.52	0.50	0.34	0.52
Std	0.02	0.02	0.02	0.01	0.04	0.04	0.05	0.04	0.02