MASTER'S THESIS 2022

# Multilingual Large Scale Text Classification for Automotive Troubleshooting Management

Alv Romell, Jacob Curman


ISSN 1650-2884
LU-CS-EX: 2022-07

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-07

# Multilingual Large Scale Text Classification for Automotive Troubleshooting Management

**Alv Romell, Jacob Curman**

# Multilingual Large Scale Text Classification for Automotive Troubleshooting Management

Alv Romell
al6515ro-s@student.lu.se

Jacob Curman
ja2085cu-s@student.lu.se

March 7, 2022

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Olof Steinert, olof.steinert@scania.com
Markus Borg, markus.borg@cs.lth.se

Examiner: Pierre Nugues, pierre.nugues@cs.lth.se

**Abstract**

This master's thesis explores the possibility of using pre-trained transformer-based language models to predict the malfunctioning part on trucks based on human-generated text descriptions from workshop work orders. It tackles a large-scale text classification problem with heavy data imbalance and multiple languages based on data from a Swedish truck manufacturer. Data analysis was done to understand the domain and generate hypotheses for methods of increasing predictive performance, with a special focus on underrepresented classes and languages. Experiments were set up to test the pre-trained models *DistilBERT* and *XLM-RoBERTa*'s predictive performance in both a monolingual and a multilingual domain, based on different techniques to reduce the data complexity, sampling techniques and augmentation through unidirectional translation. Findings show that basic methods of up-sampling infrequent classes or languages improve performance on the underrepresented segments and that monolingual models trained on translated data can perform equally well as multilingual models trained on data in its original language, and even show that monolingual models perform remarkably well on multilingual data.

**Keywords**: Machine Learning, Deep Learning, Natural Language Processing, Multilingual Text Classification, Class Imbalance, Transformers

# Acknowledgements

We would like to express our gratitude to Scania, for providing us with an interesting topic to our thesis and help and guidance with the work. First, we would like to thank Olof Steinert, our supervisor at Scania, for his dedicated work and time put in to discussing the work conducted in this project, and for always being a source of help and reflection. We would also like to thank Prashant Maheshwari and Kristoffer Bernhem for coming to our rescue when technical difficulties arose, and John Paylopoylos for the detailed and interesting insights into the world of NLP research.

We would also like to extend a warm thank you to Markus Borg, our supervisor at the Department of Computer Science at LTH, for his help and guidance along the project, and for being a great source of inspiration.

Finally, a massive thank you to our family and friends for the support along the way, and for five incredible years in Lund.

# Contents

# Chapter 1
# Introduction

"[0, 15270, 964, 320, 369, 22, 13736, 66805, 174, 19777, 964, 320, 369, 67428, 7, 123476, 38, 2]"

*Leif Östling*

## 1.1 Background

This master's thesis project was done in collaboration with a large Swedish provider of transport solutions with a global presence (hereafter referred to as "the company"). They manufacture heavy-duty trucks and busses and are a leading provider of industrial and marine engines. Headquartered in Södertälje, south of Stockholm, they have over 50,000 employees in around 100 countries. A core element of the company's business model is the provision of aftermarket services, such as repairs and maintenance of vehicles. Most vehicles sold by the company also come with service agreements, as efficient handling of truck malfunctions are critical to ensure high levels of uptime for the company's customers.

Every day, large amounts of text data are generated through fault claims from the network of workshops and dealerships. Among these claims, valuable information can be found regarding the observed issues and how customers become aware of these. As the flow of information can be ineffective and lead to loss of vital information between customer and service technician, improvements in this area could bring several benefits for both the company and its customers.

Just as in many other industries, the potential for using machine learning is growing in the transport solutions sector. Recent developments in the area of natural language processing (NLP) have enabled the automatic extraction of information from text. The company has started to investigate how this technology could be leveraged to make the process of troubleshooting more efficient by analyzing text data generated from customers. This could turn

descriptions of symptoms into a valuable source of information in the diagnostics process. One potential application is building tools to quickly provide diagnostics for the representatives who come in contact with customers, as well as efficiently provide scheduling and planning of services, and check spare part availability at an early stage. These tools could help streamline the flow of information and make the process of handling customer claims and repairs more efficient.

### Problem Description

Due to being a global actor, the customer claims that reach the company come in many different languages, and describe a wide variety of issues. Supervised-learning-based text classifiers like recurrent- or convolutional neural networks can be trained to predict the fault based on descriptions of the symptoms observed by the customer. Currently, the way to deal with the multilingual aspect is for all text to be translated into English before the models are trained. Given the recent development in NLP, the company now wants to examine the possibility of using pre-trained, transformer-based multilingual models for the troubleshooting management to get around the translation step and to investigate possible increases in performance. Using automated services for translations of large amounts of text data can be costly, hence there is also a business incentive for exploring the field of multilingual models.

### Project goals

This thesis is part of a larger project regarding advanced analytics and troubleshooting management automation. Previous research at the company has involved traditional machine learning models such as decision tree-classifiers, and deep learning models including recurrent- and convolutional neural networks. The use of deep learning models has shown promising results, but the area of transfer learning with large, pre-trained, transformer-based language models is relatively unexplored at the company, although it has shown great potential for similar problems in other contexts. The goal of this thesis is therefore to investigate whether such models can be used to predict fault classes based on human-generated text descriptions in multiple languages, and whether they can be improved with respect to under-represented classes and languages, while yielding highly interpretable models.

## 1.2 Research Questions

The research questions to be investigated in this project are the following.

- RQ1: Can existing pre-trained, multilingual, transformer-based language models yield high predictive performance* and interpretability in the context of large scale text classification for troubleshooting management?
- RQ2: How can the models' accuracy of under-represented languages and infrequent classes be improved?
- RQ3: Other than predictive performance, what are some critical aspects (such as inference time and training time) for the company to consider regarding further implementation of text classification models supporting different languages?

## 1.3 Delimitations

To narrow the scope of this project, it is confined to the context of labeled data generated from workshop work orders. The data is limited to text descriptions that are generated during the troubleshooting of trucks in workshops and contains descriptions of symptoms experienced by customers as well as findings from the troubleshooting process. While there are many sources of data that could bring valuable information to a machine learning model, such as age, mileage and error codes of a given vehicle, we consider text data as the only input to the model and hence do not consider how other data could be used or combined. This makes it easier to draw conclusions from the experiments and makes the problem more tangible. All pre-trained transformer-based models used in this project are accessed from the open-source library *HuggingFace* [1].

Regarding the use of models, one monolingual and one multilingual transformer-based model will be used to be able to study these in more detail. While it is possible to extend the project to involve several models, the primary purpose is to explore the possibility of using these, and to later optimize them for the given task and understand them better in the given context.

*The term *performance* can cover a number of factors in a computer science context. Throughout this thesis project, however, we limit the use of the term 'predictive performance' to signify a classifier's ability to correctly label samples through the use of the metrics described in Section 2.5

# Chapter 2
# Theory and related work

This chapter gives a theoretical background and provides an overview of the current state of the field in academia and industry for deep learning, natural language processing and imbalanced data. We conclude the chapter with an overview of related work on application-specific improvements to text classification.

## 2.1 Machine Learning

Machine Learning is a sub-field of artificial intelligence, which has evolved from fields including applied mathematics, statistics, computer science, control theory, signal processing and biomedical modeling. It is the study of algorithms that learn from experience and observations, much like how a human learns from observing the world with our senses. In traditional programming, the goal is to determine pre-defined program rules including model structures and parameters to generate a certain output from a given input. In contrast, Machine Learning is focused on training computers to learn to solve specific tasks without being explicitly programmed to do so. Tom Mitchell in 1997 provided a more formal definition of Machine Learning: A computer program is said to learn from experience $E$ with respect to some class of task $T$ and performance measure $P$ if its performance measure at tasks in $T$, as measured by $P$, improves with experience $E$ [2]. In today's age of increasing access to data, Machine Learning involves methods that can detect patterns in large datasets and use these uncovered patterns to predict future data or perform other kinds of decision-making under uncertainty [3].

The field of Machine Learning can be further divided into categories based on the system for which learning is conducted, and a common breakdown is into the fields of supervised learning, unsupervised learning and reinforcement learning [3]. This thesis is centered around supervised learning and the task of classification, where the goal is to learn a mapping from inputs $x$ to outputs $y$, where the outputs belong to a set of classes, $y \in \{1, \ldots, C\}$, given a set of labeled input-output pairs.

## 2.2 Deep Learning

The field of Machine Learning has seen impressive development and growth in recent decades, both in terms of new techniques and areas of application. Deep learning provides a powerful framework for supervised learning through the possibility to represent functions of increasing complexities. One of the key terms in the area is *neural networks*, which has evolved to encompass a large class of models and learning methods. Neural networks, or *multilayered perceptrons* are nonlinear statistical models with the goal of approximating some function $f^*$, which in the example of a classifier could be a function mapping an input $\boldsymbol{x}$ (vector-valued) to an output class or category $y$, through $y = f^*(\boldsymbol{x})$. A neural network would approximate this by defining a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learning the values of the parameters $\boldsymbol{\theta}$ that result in the best function approximation [4]. The output $\boldsymbol{y}$ could be a vector of values representing the predicted probabilities of the input belonging to each class.

A feedforward neural network is one where the information flows through the model and its intermediary computations without any feedback connections (where model outputs are fed back into itself). They are often represented by a *network diagram*, an acyclic graph describing the chain of functions [4]. As can be seen in the example in figure 2.1, the output layer $o$ would be a combination (chain) of the functions, or layers, $\boldsymbol{h}_1$, $\boldsymbol{h}_2$ and $\boldsymbol{h}_3$ on the input $i$. The overall length of the chain gives the *depth* of the model, and this is the origin of the term *deep learning*. A network with only a few layers of trainable weights is generally referred to as *shallow*. Deep models are those with many hidden layers, and often have a massive number of trainable parameters, reaching from tens of millions up to billions [3]. Each hidden layer of the network is typically vector-valued, where each element of the vector may be interpreted as playing the role analogous to a neuron. Rather than thinking of the layer as representing a single vector-to-vector function, one can also think of the layer as consisting of many *units* that act in parallel, each representing a vector-to-scalar operation. Each unit represents a neuron in the sense that it receives inputs from many other units in the previous layer and computes its own activation value [4].

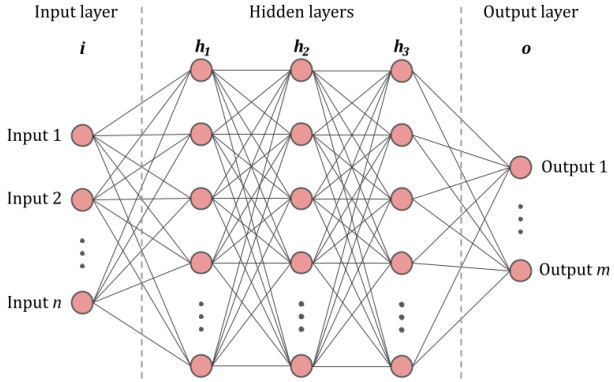



**Figure 2.1:** Structure of a Feedforward Neural Network

The motivation for using deep learning models over traditional, linear models is that many machine learning problems become exceedingly difficult when the number of dimensions in the data is high, a phenomenon known as the *curse of dimensionality* [4]. It is also the case that the linearity is limiting when it comes to modelling complex relationships between

variables in the data. Linear models can be extended to represent non-linear functions of $\boldsymbol{x}$, by applying the linear model to a transformation of the input, $\boldsymbol{\phi}(\boldsymbol{x})$, where $\boldsymbol{\phi}$ is a non-linear transformation which can be thought of as providing a new representation of $\boldsymbol{x}$. The question then is how to chose this mapping $\boldsymbol{\phi}$, and the strategy of deep learning is to learn it from data. By parameterizing the representation as $\boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\theta})$ and using an optimization algorithm one can find the $\boldsymbol{\theta}$ that corresponds to a good representation. Using a loss function, which provides feedback on how close the model output is to the expected output, or true label, and a process called back-propagation the parameters of the model are adjusted until the loss function reaches a low value, at which point the model has hopefully learnt how to accurately represent the mapping from input to output. As the non-linearity of neural networks causes the loss functions to become non-convex, most neural networks are trained by using iterative, gradient-based optimizers [4]. Gradient based optimizers decrease the value of a function $f(x)$ using *gradient descent*, updating $x$ with small steps with opposite sign of the derivative. The gradient is a generalization of the derivative when it is calculated with respect to a vector, containing all its partial derivatives. A common optimization technique is *Stochastic Gradient Descent*, which is an extension of gradient descent that alleviates the computational expensiveness of large training sets and the computational costs associated with calculating the gradient step for each of the many training samples. It is done by approximating the gradient using only a small set of samples [4].

### 2.2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specialized kind of neural networks for processing data that has a known, grid-like topology such as time-series data (a 1D-grid of time steps) and images (a 2D-grid of pixels). CNNs have been widely used in image classification, and make use of a mathematical operation called *convolution* [4]. The convolution is an integral of two functions after one has been reversed and shifted, which is evaluated at all values for the shift and expresses how one function is modified by the other. Convolutional neural networks are neural networks that use convolution instead of regular matrix multiplication in one or more layers. This process can in simple terms be described as "sliding" a filter over the grid which can detect the presence of features. The filters are also referred to as "kernels" and are normally multidimensional arrays of parameters that are adapted by a learning algorithm [4]. CNNs are great at capturing spacial dependencies and patterns, and leverage important concepts which can improve a machine learning system. These are for example *sparse interactions* (reducing the connections needed in a model, in turn reducing the parameter count), and *parameter sharing* (using the same parameter for more than one function). Convolution also allows for working with inputs of variable size [4].

## 2.3 Natural Language Processing

Natural Language Processing (NLP) revolves around linguistics, and how computers can comprehend, produce and derive meaning from human communication in the form of text and speech. It is a cross-disciplinary field that draws from computational linguistics and computer science, and the challenge within natural language processing lies within the fact that computers are unable to understand human language in the form of words and sentences,

and are only able to comprehend numbers and vectors. Also, a large amount of the complexity of a given NLP problem comes from the diversity of information each written or spoken sentence can contain, whether it is in the form of choice of words, tone, or punctuation. Humans do not have a problem processing this information, and we are also able to process things like irony and different kinds of negations, which a computer might be unable to do. Common tasks in the field of NLP are sentiment analysis (e.g. understanding if online reviews are positive or negative), text summarization, question-answering, and machine translation [5, 6].

The idea of computers being able to communicate with human-like language has been around for a long time. Already in the 1950s, Alan Turing proposed a game-like idea called *The Imitation Game* in which a human judge communicates with another human and a computer, using written communication. Both the computer and the human tries to convince the judge that they are human, and if the judge cannot consistently tell which is which, the computer wins the game [7]. When this idea was proposed it was merely a thought experiment, as the computational capabilities of that time were unable to really put this question to the test. However, as technology improved, the field of NLP has grown immensely, with the state-of-the-art performances within different tasks constantly being pushed as new models enter the scene.

### 2.3.1 Tokenization

When dealing with any machine learning task revolving around language, it is important to realize that no machine learning model takes in raw text data as its input - instead they work with numerical vectors. Thus, it is important to find suitable ways to convert this text data into a numeric representation. The process of transforming text into numerical vectors is called *vectorizing*, and it is a process needed to prepare the text data before using it in a machine learning model. The first step in this process is tokenization, which is a method performed to break down sentences into smaller segments, such as words, punctuation, or subwords. The standard and most straightforward way of tokenizing a sentence is simply splitting at every space. Using this method, the sentence *Let's learn natural language processing!* would be tokenized as ["Let's", "learn", "natural", "language", "processing!"]. An improvement to this method would be to not only split at spaces but also at punctuation, as the token "processing!" is hardly a token we would like in the model vocabulary. Adding this step to the tokenizer would make the sentence tokenized as ["Let", "'", "s", "learn", "natural", "language", "processing", "!"].

Both of these rule-based tokenization methods are very computationally inexpensive and easy to understand, but they are both likely to result in a large model vocabulary, especially when given a large corpus as input, as every single word, punctuation mark, and number in the corpus needs to be given its own token. To deal with this problem, there are a number of more sophisticated tokenization methods that use subword tokenization. The idea behind subword tokenization is to give frequently occurring words their own tokens in the vocabulary, while less frequent words are divided into more frequently occurring subwords. This way, the meaning of a word can be kept intact, but the word does not need to be given its own token. Using this method, it is also possible to specify the desired vocabulary size as a hyperparameter to the tokenization algorithm.

Two commonly used tokenizers are the WordPiece tokenizer, first outlined by Schuster and Nakajima in the paper *Japanese and Korean voice search* [8] in 2012, and the SentencePiece tokenizer, introduced by Kudo and Richardson in 2019 [9]. Both the WordPiece tokenizer and the SentencePiece tokenizer are sub-word tokenizers, and attracted a lot of attention after being the tokenizers used in the BERT model and the XLM-R model respectively (as described in more detail in sections 2.4.1 and 2.4.3). The difference between the two is in how they break down words into sub-word tokens. Where WordPiece uses double octothorps (commonly known as hashtag-symbols) to denote sub-word elements which are not the beginning of words, SentencePiece does the reverse. An element in the start of a word is prefaced by an underscore, and this in essence provides a good way to capture white spaces and reconstruct full sentences. Figure 2.2 and Figure 2.3 show examples of a text being tokenized. As seen in the figures, the tokenizers also use special tokens to denote the beginning and end of a sentence. The WordPiece tokenizer uses [CLS]-tokens (classification) and [SEP]-tokens (separation), whereas the SentencePiece tokenizer uses <s> and </s>.

**Figure 2.2:** Example of tokenization with the WordPiece tokenizer

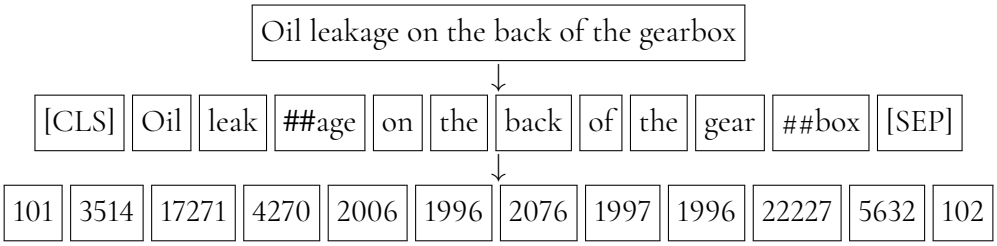
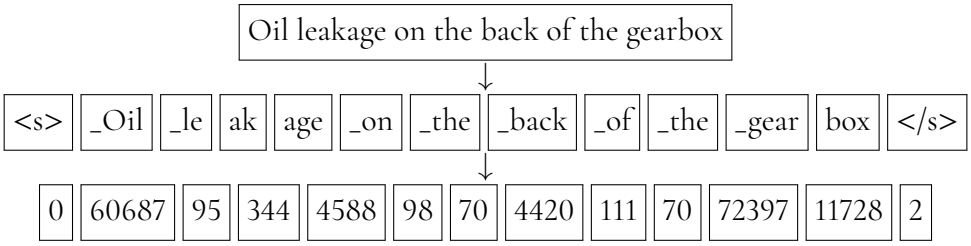



**Figure 2.3:** Example of tokenization with the SentencePiece tokenizer




## 2.3.2 Word Representations

After performing the chosen tokenization procedure on the text data, numeric vectors are associated with the generated tokens, and it is these vectors that are fed into the model. There are a variety of ways to associate each token with a vector, but two frequently used techniques are *One Hot Encoding* and *Word Embeddings* [10].

### One Hot Encoding

One Hot Encoding is a common and one of the most basic ways to convert a token into a vector representation. The technique simply assigns an integer value $i$ to each token in

the vocabulary and thereafter represents that specific token with a vector of size $N$ (size of the vocabulary) with a 1 on the $i$:th position and zeroes elsewhere. One Hot Encoded word vectors are sparse, hard-coded, and often high-dimensional (same dimensionality as words in the vocabulary), as opposed to the other commonly used technique, Word Embeddings [10].

## Word Embeddings

The term word embedding describes a mapping of a particular word or token to a vector representation, where the vector representation can capture the semantic meaning of the word. The vectors are typically real-valued, and the vectors encode the meaning of the words such that words that have similar semantic meaning are likely to have vector representations that are located close to each other in the vector space. Word embeddings can be learned from the data, where the word vectors are initialized randomly and then altered using back-propagation in a similar manner that a neural network learns its weights. However, if the available amount of data is small, another approach is to use already existing word embeddings that have been pre-trained on another task. When using word embeddings, the semantic differences between words can be represented as geometric relationships between the numeric vectors. This can be exemplified by performing basic vector operations such as addition and subtraction. For instance, taking the embedding vectors for the words "Sweden", "Stockholm" and "Portugal" and performing the operations **Stockholm – Sweden + Portugal** gives a vector that is very close to the vector representing the word "Lisbon". In the same way, it is possible to find vectors that represent different word operations. For instance, the vector that allows us to go from the word "son" to the word "daughter" should be similar to the vector that allows us to go from the word "king" to the word "queen". Similarly, the vector that allows us to go from "king" to "kings" should also be similar to the vector that allows us to go from "queen" to "queens". These vector representations are visualized in a lower-dimensional space in Figure 2.4. As opposed to one-hot encoded word vectors, word embeddings are dense, lower-dimensional, and learned from the data [10]. Visualizations of both one-hot word vectors and word embeddings are shown in Figure 2.5.

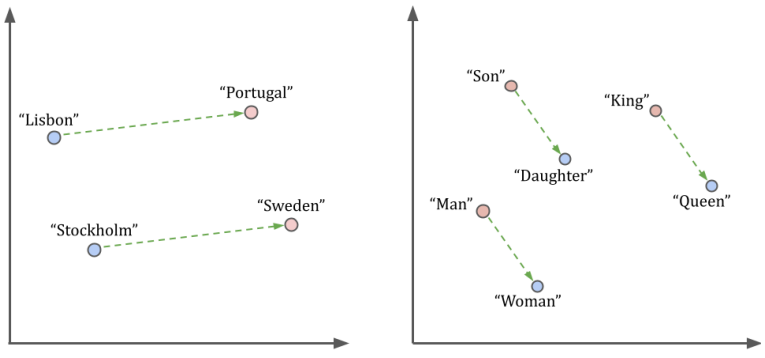



**Figure 2.4:** Illustration of relationships between word embedding vectors in two dimensions.

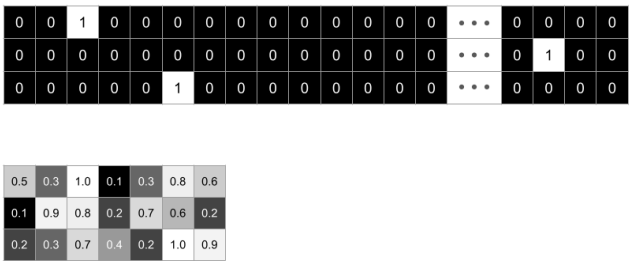



**Figure 2.5:** Visualization of one-hot word vectors (top) showing large, sparse vectors with black representing zeros and white representing ones, and word embeddings (bottom) showing lower dimensional dense vectors. Adapted from [10]

## Word2Vec & FastText Embeddings

Word2Vec is one of the most commonly used algorithms for creating word embedding representations from a large corpora of unstructured data. Introduced by Mikolov et al. in 2013 [11], the Word2Vec algorithm uses a neural network to learn the embedding representations of words using two different learning techniques - continuous bag-of-words (CBOW) and skip-gram. When using the continuous bag-of-words method, the task of the neural network is to predict a word given its surrounding words, using a certain window size. When using skip-gram, the training technique is the exact opposite. Here, the task of the network is to predict the neighboring words given the center word. The two techniques are visualized in Figure 2.6 and Figure 2.7 using a window length of three, where the red words are the target words that are to be predicted, and the blue words are context words.

**Figure 2.6:** CBOW

I like natural language processing
I like natural language processing
I like natural language processing

**Figure 2.7:** Skip-gram

I like natural language processing
I like natural language processing
I like natural language processing

Even though Word2Vec has proven to be a very efficient method to generate word embeddings learned from data, there are drawbacks associated with the method. Firstly, the method is unable to handle out-of-vocabulary words. When training, the Word2Vec algorithm learns a vector representation for every word, but it cannot handle words it has not encountered in the training data. The second drawback is related to the morphology of the words. As Word2Vec learns vector representations for each word individually based on the context the words appear in, there are no automatic similarities between the vector representations of words that share the same lemma or stem, thus ignoring the internal structure of the words [12].

FastText is an extension of the Word2Vec-model, which was introduced by Bojanowski et al. in 2017 [12]. Instead of learning vector representations for individual words, the FastText algorithm breaks words down into smaller parts, so called character level n-grams.

**Table 2.1:** Character level n-grams for different values of n

| Word | n | n-grams |
|---|---|---|
| vector | 3 | vec, ect, cto, tor |
| vector | 4 | vect, ecto, ctor |
| vector | 5 | vecto, ector |

The algorithm learns a vector representation for each of these n-grams using the skip-gram technique, and the final vector representation for the word becomes the sum of the vector representations for the word's n-grams. With this approach, the model will end up with vector representations for all n-grams obtainable from the training corpus. This means that if the algorithm encounters a word that it has not seen during training, it is in many cases able to assign a vector representation to the word since it is likely to have a representation for one of the word's n-grams.

### 2.3.3 Transformer Architecture

The transformer architecture was first introduced in 2017 [13], and has since then been used in many of the models that today give state-of-the-art performance on many NLP-tasks. Transformers make use of the *Attention Mechanism*, which allow models to make connections between words in sequential text data. This section will briefly describe how both the attention mechanism and Transformers work.

#### Attention

Attention is a mechanism used in neural architectures that allows for placing of special focus on certain parts or features of neural network inputs [14]. In NLP, it helps models to distinguish connections between different inputs, or different parts in an input sequence by weighting them based on how important they are to one another. For instance, if a model is fed the input sentence "The dog didn't cross the street, because it was scared", it could prove difficult for the model to distinguish if the word "it" refers to the dog or the street. In this case, the attention mechanism helps the model clear the ambiguity by putting more weight on "the dog" when encoding the word "it". In short, the attention mechanism can be described as weighting the relevance of input elements, and then taking these weights into consideration when performing the model's main task [14]. In the case of Natural Language Processing, weighting the relevance of input elements refers to quantifying how relevant each word in the input sentence is to every other word in the same sentence. This mechanism is of great importance when dealing with models that handle sequential inputs, such as text input in NLP. In this case, attention enables models to take in long input sequences without forgetting information early in the sequence when it completes processing the whole input.

## Transformers

The transformer architecture was introduced in 2017 in the now acclaimed paper *Attention is all you need* [13]. Before the release of this paper, the attention mechanism had almost exclusively been used in conjunction with Recurrent Neural Networks, which are neural networks well suited to handle sequential data, but for which the possibilities of parallel computations are severely limited due to the sequential nature of the network. The transformer, however, was the first model architecture to rely entirely on attention to compute representations of its input and output data, without using Recurrent Neural Networks.

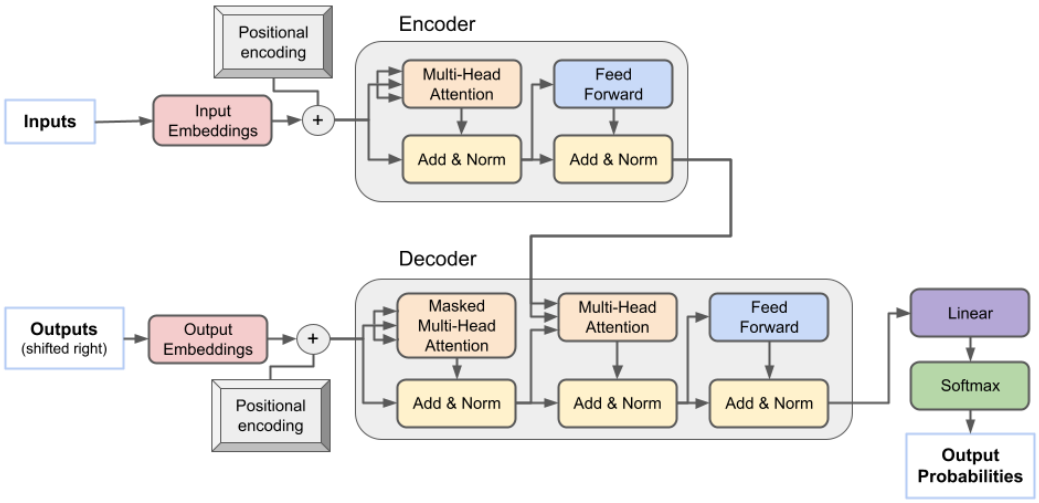



**Figure 2.8:** The architecture of the Transformer. Adapted from Vaswani et al. [15]

Figure 2.8 describes the overall architecture of the Transformer. The main components of the Transformer is the encoder shown in the top of the figure, and the decoder shown below. The encoders encode the input to some hidden representation, and the decoders decode this hidden representation into a new output. In for example machine translation, this decoder output can be a new representation of the input text in another language. Within both the encoder and the decoder there are several *attention layers* (colored orange in Figure 2.8), making use of the previously mentioned attention mechanism, followed by a feedforward (often fully connected) neural network. More specifically, the central part of the transformer is "self-attention", meaning that the attention is calculated between words in a given sentence internally, rather than calculating attention between words in different sequences. The encoder maps the input sequence to a sequence of representations, which are the outputs of the encoder's attention layer. Given these representations, the decoder then generates an output sequence of symbols, one element at a time.

When passing information into the transformer model you start by representing words in the input $\boldsymbol{X}$ with token embeddings, as numerical vectors, meaning word $\boldsymbol{i}$ is represented by the vector $x_i$. Starting with this word embedding, it is then computed how it is related to all other words using three types of vectors, called queries, keys, and values ($\boldsymbol{Q}, \boldsymbol{K and V}$). To compute these there are three types of weight matrices, $W_q, W_k, W_v$. The parameters of these

matrices are optimized during the training of the neural network. The input $x_i$ is multiplied by the $W$-matrices to generate $q_i$, $k_i$ and $v_i$ for each token respectively. After computing $Q$, $K$, and $V$ a dot product is computed between the queries and the keys which gives a similarity metric. The results are scaled by dividing with the square root of the dimension (to improve stability of model during training), and the results are scaled again by being passed through a softmax layer, to between 0 and 1. This is called scaled dot product attention, and its formula is shown in the following equation:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where $d_k$ is the dimensions of the queries and keys. The softmax-operation normalizes a vector of real numbers, and turns it into (what can be interpreted as) a probability distribution of values in the interval of (0, 1), all adding up to 1.

One of the great benefits of the Transformers lies within the multi-head attention blocks described in Figure 2.8. These blocks allow the Transformer to focus on different positions in the input data by calculating attention multiple times with different sets of queries, keys, and values, and then average the outputs to get the final representations. Thus, one multi-head attention layer consists of several attention layers running in parallel, enabling the model to perform parallelized computations. This results in Transformer architecture models requiring significantly less time to train compared to other models that also use attention [13], while also delivering state-of-the-art results on many NLP tasks.

### 2.3.4 Transfer learning in NLP

Deep learning is notorious for its dependence on massive amounts of data for training, in comparison to traditional machine learning methods [16]. This creates difficulties in producing efficient machine learning models for many applications due to the fact that labeled data is often hard to come by for specific tasks. The concept of transfer learning has become an important tool in deep learning to solve the problems that arise due to data scarcity. The goal is to leverage knowledge from a source task to improve learning in a target task [17]. The idea is that a model can be trained in a domain separate from that of the final task, where training data is abundant, to avoid the labor-intensive and costly task of labeling data in the target task domain. As phrased by Tan et al., transfer learning relaxes the hypothesis that the training data must be independent and identically distributed (i.i.d.) with the test data [16], which opens up for the possibility described above. Definition 1 from Yang and Pan [18] gives a formal description.

**Definition 1** (Transfer Learning). *Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$*

In natural language processing with deep learning, for a model to gain a rich language understanding there is a need for extensive training, and thereby for large amounts of training data. It is also a time-consuming and complex process. Being able to use previously learned language understanding from earlier models is therefore a very attractive idea when creating task specific models. Some of the earliest forms of transfer learning in NLP was the use of pre-trained word embeddings, which is a field that has evolved through the years with some of

the most impressive and successful examples mentioned in section 2.3.2. A significant breakthrough in the field was due to the approach of using unlabeled data to induce word features, as it brought with it the possibility of utilizing large, unlabeled corpora which were easily accessible on the internet [19]. The *semi-supervised* approach of word prediction produced great results in deep sequence models for NLP tasks. The learned parameters in the networks can be used as a starting point in other models, which are adapted for specific downstream tasks, and the parameters are fine-tuned through training on supervised data [20].

In recent years, substantial gains on many NLP tasks have been shown through the process of *pre-training* language models on large corpora of texts, followed by *fine-tuning* the models on specific downstream tasks. This pre-training can be seen as giving the model a general language understanding, where the semantic meaning of words and context can be captured. It has been shown that scaling up language models (increasing the capacity of the models through an increased number of parameters) greatly improves task-agnostic performance [21], although the scale of today's state-of-the-art models is on a level where only a few actors have the resources to produce and train such models. Many models have, however, been released to the public and are available as open-source, including the ones described later on in section 2.4. These are increasingly flexible and apt for downstream transfer, and the ability to fine-tune pre-trained language models based on the recent transformer structure has all but removed the need for task-specific model architectures [21].

### Fine-Tuning of Transformer Models

In the paper *Universal Language Model Fine-tuning for Text Classification*. The authors present a novel method for fine-tuning language models on downstream tasks, to avoid "catastrophic forgetting" of semantic understanding learned during pre-training. The method includes *discriminative fine-tuning*, *slanted triangular learning rates*, and *gradual unfreezing* and is referred to as ULMFiT [22].

The fine-tuning of pre-trained transformer-based language models is an unstable process, and training the same model with random initialization can result in large variance of task performance [23]. While it is generally accepted that two main reasons for this instability are small fine-tuning datasets and catastrophic forgetting, it has also been attributed to optimization difficulties that lead to vanishing gradients [24].

## 2.4 Models

This section introduces the models BERT, DistilBERT, and XLM-R, of which the two latter are the deep learning models used throughout this thesis, which are both based on the progress made by the team behind the original BERT model. The models are available open-source, and the DistilBERT and XLM-R models were accessed through the Transformers library from *Huggingface*.

### 2.4.1 BERT

BERT (Bidirectional Encoder Representation from Transformers) is a state-of-the-art language model developed by Devlin et al., released in 2018, which has achieved impressive re-

sults in an array of language understanding tasks. The novelty of BERT lies in its bidirectional training, using the transformer architecture. Previous models relied on processing sequential data when analyzing words in a sentence in relation to each other, to capture context. This process of only looking from left-to-right, or vice versa, limits the model's ability to gain a full language understanding, as the context of a word in a sentence depends on other words both to the left and to the right of itself [25].

BERT alleviates the unidirectionality constraint through the use of a pre-training method called Masked Language Modelling (MLM). In MLM a percentage of the input words (15% in the original BERT-paper) are masked with a special token. The goal is then for the model to predict the masked word, based only on the context of the surrounding words in the sentence. This allows for the pre-training of a deep bidirectional transformer. The training of BERT also features a second parallel training method called Next Sentence Prediction (NSP), where the model is given two sentences, A and B, and tries to decide whether B is actually a sentence that proceeds sentence A, or a random sentence. This lets the model understand semantic relationships between different sentences. BERT is pre-trained on two large English text sets, the BooksCorpus which contains 800 million words, and text passages from the English Wikipedia containing 2,500 million words [25].

Comparing BERT to previous word embedding models, like *Word2Vec* mentioned in 2.3.2, BERT can capture the contextual meaning of homonyms, or multiple-meaning words. An example is the word "bark", which can mean both the outer layer of a tree and the sound made by a dog. As two homonym words often have completely different meanings, not being able to distinguish between the two severely limits a model's ability to correctly identify the semantic meaning and context of a sentence. The BERT "base" model has 12 transformer block layers, an embedding dimension of 768, and 110 million parameters for the base model [25].

BERT pushed the state-of-the-art within the field of NLP far forward. As the inherent language understanding of the trained BERT model is captured in these parameters, the models can be used for a wide variety of different tasks once it has been trained on a large corpus, as discussed earlier in the concept of transfer learning.

### 2.4.2 DistilBERT

In machine learning, *knowledge distillation* is a process by which it is possible to transfer knowledge and learned predictive capability from a large and cumbersome model to a smaller, more lightweight model, which might be more suited to deploy and use in production. Bucilua et. al [26] present a method for "compressing" large, complex ensembles of models into smaller models and thereby gaining computational speed without a great loss in performance.

A distillation approach that has proved effective in transferring knowledge from a larger to a smaller model is by training the smaller model to replicate the target distribution of the larger model, using the class probabilities of the large model as so-called "soft targets" [27]. The idea is to teach a small model to mimic the target distribution of a larger model with many more parameters, thereby providing it with the ability to generalize on new data better than if it had only been trained directly on the datasets used to train the larger model. It has been shown to allow for training with less data and larger learning rates [27]. Learning rate is a hyperparameter that decides how much the weights of the model are updated at every iteration in response to the estimated error during prediction.

In 2019 a *distilled* version of BERT was released by Sanh et al. [28] through the use of the

previously mentioned methods, aptly named DistilBERT. The authors created a model with the same general structure as the original BERT model, modified to reduce its size through the removal of layers, resulting in a parameter count of 66 million. Compared to the original model this is a 40% reduction in model size, which is 60% faster to train yet retains 97% of the language understanding of BERT [28].

### 2.4.3 XLM-RoBERTa

In November 2019 Facebook released a robustly optimized recreation of the original BERT, named RoBERTa, in which they experimented with the pre-training procedure without making significant changes to the general model structure originally presented by the Google research team [29]. In this implementation, key hyperparameters were modified, the next sentence prediction objection is removed, and batch sizes are changed.
Subsequent work from Facebook led to the development of a multilingual version of RoBERTa, which came to be known as XLM-RoBERTa (XLM-R) [30]. XLM-R uses the tokenization technique SentencePiece, instead of BERT's WordPiece. The base model has a similar structure to that of BERT with 12 attention block layers and an embedding dimension of 768, but has a larger capacity of 270 million parameters (for the base model) [30].

## 2.5 Precision Metrics

In binary classification tasks the performance of a classifier can be expressed in a variety of metrics, calculated by using the terms *true positive*, *false positive*, *true negative* and *false negative*. Positive and negative refer to how a sample has been classified, as in either belonging to a target class or not. True and false in turn refer to whether the classification was correct in identifying the sample's true class. Accuracy is the number of correctly labeled samples as a proportion of all samples.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{All samples}}$$

Precision is the fraction of correctly labeled samples out of all those predicted as positives. It is also called positive predicted value.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall is the fraction of correctly labeled elements out of all relevant elements in the population.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$F_1$-score is another measure of a classifier's accuracy. It is calculated as the harmonic mean of precision and recall.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Depending on the task, it can be more desirable to have high values in a certain selection of these than others, although preferably all of them should be high. A high precision value

only means the classifier does a good job at predicting samples that are relevant, i.e., it does not produce many false positives. However, a high precision says little about how many relevant samples that are left out and should have been identified as positives, i.e., the fraction of false negatives. Precision is important when the cost of a false positive is high. A high recall value means there are few relevant samples that are not correctly identified. A high recall value is desirable when the cost of a false negative is high [4].

In a multi-class classification setting, one needs to categorize a sample into one out of N different classes. The metrics above can still be used if the task is treated as a set of binary problems ("one-vs-all"). Similar to the binary case, precision, recall, and $F_1$-score can be calculated for each class respectively, but to get a single number to describe the classifier's overall performance these different scores must be combined, and there are different ways of doing so. The most straightforward way is to calculate the arithmetic mean of the per class scores, which gives equal weight to all classes and is called the *macro-averaged* score. Another alternative is to calculate the *micro-averaged* score, which gives equal weight to all samples, rather than all classes. This is common in the case of binary classification problems with imbalances in the datasets and is done by calculating over all samples. In a multi-class setting, when each sample is assigned to exactly one class, the micro-averaged value for precision, recall, and $F_1$ will always be equal to the accuracy and are therefore less insightful. This is due to the fact that each incorrectly labeled sample will be a false positive with respect to the predicted class, but at the same time be a false negative with respect to the true class. Given that the micro-averaged values of prediction and recall in this setting are given by

$$\text{Precision}_{micro} = \frac{\sum_c \text{TP}_c}{\sum_c \text{TP}_c + \sum_c \text{FP}_c} \quad \text{Recall}_{micro} = \frac{\sum_c \text{TP}_c}{\sum_c \text{TP}_c + \sum_c \text{FN}_c}$$

where $c$ is the class label, the fact that $\sum_c \text{FP}_c = \sum_c \text{FN}_c$, and that the denominator sums up to the total number of samples, will result in precision and recall (and thereby $F_1$-score) all being equal to accuracy. In this project we will report *macro* averages for precision, recall and F1-score when evaluating the models.

## 2.6 Imbalanced data

The problem of class imbalance is persistent and ubiquitous in machine learning when it comes to creating predictive classification models. The phenomenon appears when there are many more examples belonging to some classes than others, and can greatly impair the potential of classifier models [31]. Not only do large imbalances in the data distribution make general performance metrics like overall accuracy less useful. For example; In a dataset with two classes A and B, where 95% of samples belong to the prior, and only 5% belong to the latter, one can reach an accuracy of 95% by simply always predicting class A, which makes studying other metrics like precision and recall more important. Large class imbalance also tends to produce an inductive bias towards the majority class or classes in many models [32].

Imbalanced data can significantly compromise the performance of most standard learning algorithms, which assume or expect balanced class distributions or equal misclassification costs. While any unequal distribution between classes can be considered imbalanced, common understanding in the community is that the term "imbalanced data" generally refers to datasets exhibiting severe or extreme imbalance, and between-class imbalances in the orders

of 100:1, 1000:1 or even as large as 10000:1 are not uncommon [33].

When imbalances in data are a direct result of the nature of the data space, they are referred to as *intrinsic*. If they are caused by other factors, such as sampling time and storage, they are considered *extrinsic* [33]. Factors in the method of sampling can produce an imbalanced dataset from an otherwise balanced data space. Imbalances can also be *relative* or *absolute*, the latter being imbalance due to rare instances. While a class can be heavily outnumbered in a datasets, it is not necessarily imbalanced due to a lack of samples but instead because the majority class is very large.

The degree of imbalance is not the only thing that hinders learning. Studies show that the effect of class imbalance can be increased given high levels of complexity in the data [34]. Data complexity is a broad term that involves issues such as overlapping, lack of representative data, small disjuncts and more, and it has been identified as a determining factor of classification deterioration [33]. Small disjuncts, i.e. when the minority class is being decomposed into many sub-clusters with very few examples have been connected to degradation in classifier performance [35]. Overlapping between classes is defined as those regions of the data space in which the representation of the classes is similar, and can be among the most harmful issues for many classifiers [36, 37]. Class noise (or label noise) is another factor that has been shown to impair classification performance [38]. It refers to a situation where there are incorrectly classified samples, which make the representations for each class ambiguous and leads to increased data complexity [39].

### 2.6.1 Dealing with class imbalance

There are several common ways to handle class imbalance in machine learning. They include sampling techniques (such as oversampling of the minority class or undersampling of the majority class), using cost-sensitive learning where misclassifications of certain classes are penalized more than others, and generating synthetic data through data augmentation [32].

Over- and undersampling are basic techniques that each have strengths and weaknesses. Undersampling of the majority class has the benefit of not requiring the addition of 'fake' data to the set, but inevitably leads to information loss from the removed samples. Wasting data in this sense is therefore mainly a viable option when there is an abundance of available data. Oversampling of the minority class leads to no such loss in information, but can tend to reinforce patterns that do not generalize well and thereby cause the model to overfit on the training samples. Oversampling has been shown to both increase and decrease classification performance, depending on the situation [34]. It also does not provide any new information to the model, which can be achieved through data augmentation.

Data augmentation refers to methods used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data [40]. It is common in computer vision and has been particularly effective for object recognition [4]. Images can be manipulated in different ways to create these new samples. For example, a picture of a cat can be rotated, mirrored, or have its colors altered, and technically be the same picture to a human eye, but provide new information to a machine learning algorithm. Using new information rather than oversampling with identical samples, improves the diversity in the training data and helps the model generalize better on unseen data [40]. In contexts dealing with numerical data as model inputs, a common method to augment a dataset is to use *SMOTE* (Synthetic Minority Oversampling Technique), which has the benefit

of adding samples that are not exact replicas of the minority instances in the datasets, which forces the decision region of the minority class to become more general [41]. The SMOTE algorithm generates new samples by looking at random minority class samples and selecting nearby related samples (using $K$-nearest neighbor) and interpolating a line in feature space, then generating a new sample somewhere on that line. While this is theoretically possible to do when working with language and text data, it has to be done on the embeddings of words or sentences. Given that word embeddings are generally of large dimensions, especially when using transformer-based models, this complicates matters as the SMOTE technique is less effective in high dimensional feature spaces, as $K$-nearest neighbor classifiers perform worse in higher dimensions [42].

Data augmentation methods for NLP have been more explored recently, and analyzed in surveys by Li et al. [40], Feng et al. [43] and Liu et al. [44]. The interest has grown as NLP has seen more work appear in low-resource domains, with a plethora of new tasks and due to the popularity of large-scale neural networks that require large amounts of data [43]. A number of methods have been suggested which all in some sense manipulate language and text to form new samples, and can be broadly split into three categories: *Paraphrasing*, *Noising* and *Sampling* [40]. Methods vary regarding the extent they change the original texts, from replacing single words with synonyms, to rephrasing an entire sentence while still conveying the original meaning of the text. Machine translation is a natural means of paraphrasing and has become a popular approach given the availability of online translation API:s. Back-translation is a method in which a document is first translated into other languages, and then translated back to its original language, often obtaining a sample that is different but (hopefully) contains the same semantic information. In the multilingual application context, unidirectional translation can be used to generate a new sample in another language. While machine translation as a means to augment data has the advantages of being easy to use, having strong applicability and ensuring correct grammar and unchanged semantics, it is limited in its poor controllability and diversity because of the fixed machine translation models [40].

## 2.7 Related Work

In this section, we present related work in the field of transfer learning and fine-tuning of pre-trained transformer-based language models for text classification.

In *How to Fine-Tune BERT for Text Classification*, Sun et al. [45] investigate different fine-tuning methods for BERT on text classification tasks. While they note that the potential of BERT has yet to be fully explored and that there is little research to enhance BERT to improve prediction accuracy on target tasks further, they also provide a general solution for BERT fine-tuning. The proposed solution includes three steps: (1) Further train BERT through *masked language modeling* and *next sentence prediction* (in the same way it is trained during pre-training) on within-task training data or in-domain data; (2) optional fine-tuning of BERT with multi-task learning if several related tasks are available; (3) fine-tune BERT for the target task. The authors also explore the fine-tuning methods for BERT, and discuss the importance of choosing optimizer and learning rates appropriately to avoid overfitting and "catastrophic forgetting", where the pre-trained knowledge is erased during the learning of new knowledge. The authors show that further pre-training within the domain is an efficient way to enhance performance on the downstream task, but that generally, in-domain pre-training can bring

better performance than within-task pre-training.

In *Multiclass Imbalanced Classification of Quranic Verses Using Deep Learning Approach*, Aqsa and Ahmad [46] use a deep learning approach for classification of Quranic verses. Working with an imbalanced dataset with samples mapping to six different classes, they show that traditional oversampling and augmentation methods such as SMOTE can be used to tackle problems caused by data imbalance.

In *Transfer Learning Robustness in Multi-Class Categorization by Fine-Tuning Pre-Trained Contextualized Language Models*, Liu and Wangperawong [47] compare the effectiveness and robustness of multi-class categorization of Amazon product data using transfer learning on pre-trained contextualized language models, by fine-tuning BERT on a dataset as the number of classes grows from 1 to 20. They show an approximately linear decrease in performance metrics with the number of class labels, with an estimated performance degradation rate of approximately 1% per additional class, and achieve macro-averaged accuracy and $F_1$-score of 80% with 20 different classes.

In *Large Scale Legal Text Classification Using Transformer Models*, Shaheen et al. [48] tackle the problem of large multi-label text classification (a case of text classification where each sample or document can be assigned more than one label) in the legal domain. They describe how the setting of *long-tail* frequency distribution (where some labels or classes are used frequently, while others are used very rarely) at large scale is largely unexplored in the context of text classification. They compare DistilBERT to RoBERTa and show that the former, and smaller, model performs surprisingly well and on par with the latter.

Yu et al. discuss in their paper *Improving BERT-Based Text Classification With Auxiliary Sentence and Domain Knowledge* [49] a method of "constructing auxiliary sentence to turn the classification task into a binary sentence-pair one, aiming to address the limited training data problem and task-awareness problem", and analyze the use of post-training (further pre-training model in the task domain) to improve classification performance with mixed results on different tasks. For multi-class classification, the use of suitable auxiliary sentences seems to improve model performance, and while they do not see added benefits of post-training, they do believe it is due to the data and that the method of utilizing domain-related corpus still has potential.

Prabhu et al., in their paper *Multi-class Text Classification using BERT-based Active Learning* [50], explore methods of using active learning to improve multi-class classification for text. In a domain where labeled data is sparse, allowing models to select sets of data from an unlabeled set can be an effective way of increasing the available data for training, and reducing labeling costs by 85%. Active learning for improving text-classification is also explored in *On Using Active Learning and Self-training when Mining Performance Discussions on Stack Overflow* by Borg et al. [51] and in the master's thesis *Evaluation of Active Learning Strategies for Multi-Label Text Classification* by Zethraeus and Horstmann [52], although in multi-label contexts. Both studies similarly show that active learning strategies can help decrease the time needed for annotating data.

While multilingual transfer learning can be able to benefit both high- and low resource languages, Singh et el. show in their paper *BERT is Not an Interlingua and the Bias of Tokenization* [53] that the multilingual version of BERT partitions representations for each language rather than using a common, shared, interlingual space. They claim to show that the effect is magnified in the deeper layers of the model, which would indicate that the model does not abstract semantic content while disregarding languages. Through hierarchical clustering

based on similarity scores from *Cananical Correlation Analysis* a tree-like structure is revealed where the partitioning of representations in different languages mirrors that of the linguistics and evolutionary relationships between languages.

# Chapter 3
# Data

In this section we provide an overview of the data available for this project. We describe the exploratory data analysis and the assumptions, limitations, and other concerns on which the experimental phase is based.

## 3.1 Dataset Overview

The dataset studied in this project is in the form of texts in a variety of languages describing malfunctions in trucks, originating from workshop work orders. The work orders are produced by individuals working in workshops, and while the data used in this project comes from real work orders, the distribution of languages and classes shown do not necessarily represent the distribution with which the orders reach the company. The data has been extracted from a database containing work orders from multiple years and for each non-English sample, an automatic third-party translation service has attempted to generate an English translation of the text (not always in a successful manner, further explained in Section 3.2), together with the predicted language of the original text. The data is labeled, with each text sample corresponding to a *main group* and a *class*. The main group refers to what overarching segment the faults belong to, such as engine or chassis, and the class is an integer mapping to a particular sub-part that has been identified as the root cause for the vehicle malfunction in a workshop, such as oil pump or yoke. The latter category, i.e., the root cause class, is what we refer to as the target classes for the classifiers in this project. In table 3.1 an example containing modified dummy data is presented to show the structure in which the dataset was received. It is common to evaluate models on data that they have not been exposed to during training, to see how well a model can generalize (perform on unseen data), and avoid *overfitting* to the training data. The dataset in this project contains roughly 450,000 samples split into training-, validation- and test sets, as shown in Figure 3.1 This split was done before receiving the datasets to achieve similar distributions of classes and languages, and not altered throughout the project by request of the company.

**Table 3.1:** Example of data (with modified numbers).

| Main group | Original text | Predicted language | Translated text | Class |
|---|---|---|---|---|
| 8 | Noise from rear axle | en | Noise from rear axle | 1227 |
| 15 | Luftverlust am Fahrersitz | de | Air loss at driver's seat | 223 |

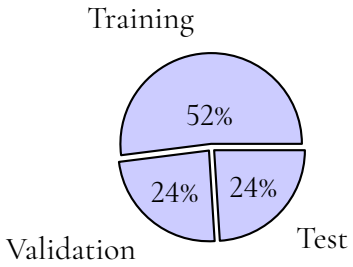



**Figure 3.1:** Data split into training, validation and test sets.

Some limitations and inherent issues with the data include the fact that the texts have been generated by individuals in different countries and situations, leading to a large variety in the quality of the samples and thereby the inherent semantic information that is available. Some examples contain only a single word, while others are long and detailed descriptions. A breakdown of the data is found in section 3.2. On request by the company, we will not present the actual names of the classes (i.e. the part that caused the malfunction of the truck), but instead the assigned integer values.

## 3.2 Exploratory Data Analysis

Through exploratory analysis by manual inspection, and using the Python libraries *Pandas* and *Matplotlib*, the data was studied to understand its characteristics. The findings are presented below.

### Language distribution

There are 38 languages identified in the dataset by the language detection service, and a subset of these are presented in Figure 3.2, together with examples of the languages present. The languages have been anonymized per request of the company. When the translation service has not been able to identify a language, a predicted language of "Unknown" (abbreviated "un" in Figure 3.2) has instead been assigned. The distribution of the most frequent languages can be seen in figure 3.2, showing the large imbalance between the languages. The ten most frequent languages make up 93.3 percent of the total samples, while the ten least frequently predicted languages make up 0.01 percent.

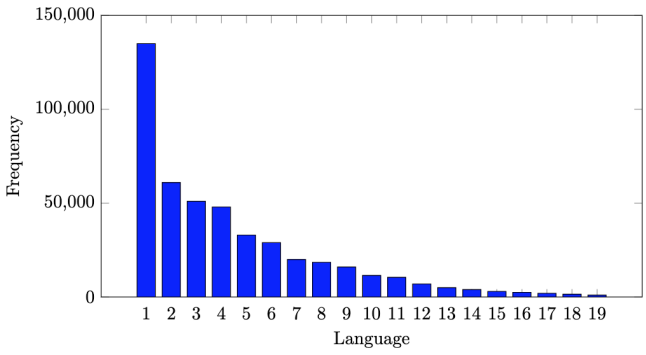
Through manual inspection, it is possible to identify flaws in the language detection and to find translations of cases where a language has been identified, but where the translation does not seem to be accurate. Examples of these are shown in Table 3.2 For example, there are cases where text in Swedish has been assigned a predicted language of Vietnamese. Another example is the presence of the language "Esperanto", which has been identified 14 times in the

whole dataset, but when inspected closer seem to be incorrect classifications of texts in languages like Danish and Latvian. Similarly, the language “Cebuano” has also been identified 14 times, mainly for a specific phrase in Spanish. When manually inspecting the samples for every low-frequency language, it was noted that only the 19 most frequent languages could with certainty be said to be represented in the dataset. For the 19 least frequent languages (consisting of between 1 - 44 samples per language), the textual description was clearly written in another language or lacked interpretable text to the extent that the language predictions were considered inaccurate. As it could not with certainty be said that these languages actually existed in the data, it was chosen to throughout this thesis not report precision metrics for them, although the samples were kept in the data.

**Table 3.2:** Example of samples with inaccurate language predictions. The predicted languages are Vietnamese, Haitian and Indonesian.

| Main group | Original text | Predicted language | Class |
|---|---|---|---|
| 14 | 110-5002.06 SKARVKOPPLING 6 MM PLAST | vi | 4 |
| 4 | TMI 04 15 02 19 | ht | 38 |
| 9 | tpm 48180 | id | 309 |

For the nearly 50,000 samples that have been assigned a predicted language of "Unknown", a translation has not been generated and the translated text is simply a not-a-number (‘NaN’) value. In 70% of the cases where a language has not been identified, the reason is that the original text is simply an empty string, containing a single whitespace character (’ ’). The cause for this could be mistakes in the manual data entry process, or due to empty descriptions being provided in the workshop work orders. These samples contain no information of value for either language domain (original or translated), and were therefore disregarded throughout all experiments. However, the samples with a predicted language of "Unknown" that contained texts were kept in the datasets.




| Identified languages | ISO Code |
|---|---|
| German | de |
| English | en |
| Swedish | sv |
| Finnish | fi |
| Norwegian | no |
| Danish | da |
| French | fr |
| Dutch | nl |
| Portuguese | pt |
| Italian | it |
| Czech | cs |
| Hungarian | hu |
| Spanish | es |
| Lithuanian | lt |

**Figure 3.2:** Language distribution (left) and examples of identified languages, not ordered (right).

## Class distribution

There are over 1,350 unique classes in the available data, all indicating a subpart that has been identified as being the root cause of a problem in a workshop work order. Examples of the classes are 'water valve', 'yoke', and 'oil pump'. The distribution between the classes is heavily imbalanced. This imbalance is mainly believed to be absolute and caused by intrinsic factors (as discussed in section 2.6), given that some errors and faults are more likely to appear earlier in the life cycle of a truck than others. The class distribution is not equal between the languages, meaning there is a difference in the number of unique classes that are represented in each individual language, as can be seen in figure 3.4. However, the class distribution between the training, validation, and test sets were similar, as shown in figure 3.3. In this thesis we occasionally refer to the number of instances of a given class in the datasets as *support*, which simply is the number of samples belonging to that class. The ten most frequent classes (i.e. those with the highest support) make up 14.8 percent of total samples, while the one hundred least frequent classes on the other hand make up less than 0.3 percent. As discussed in section 2.6, class imbalances make classification tasks more difficult and lead to complications when trying to create machine learning models, due to the tendency of models to overfit to the overrepresented classes.

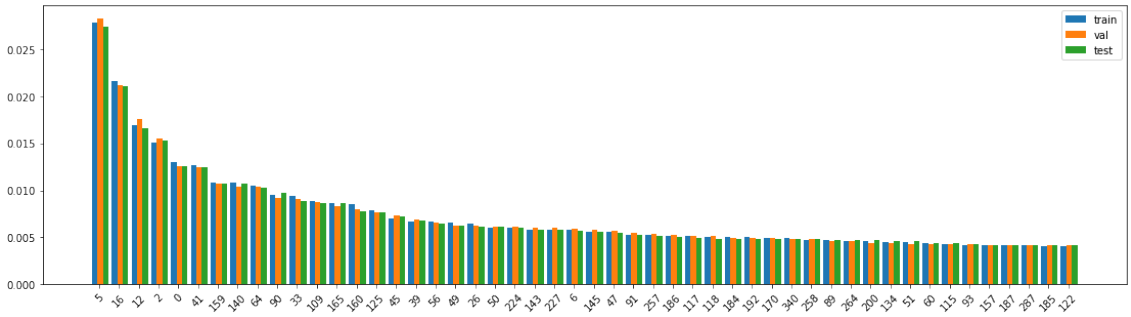



**Figure 3.3:** Distribution of the 50 most frequent classes in the training, validation and test sets as a percentage of the total amount of samples in each set.

As shown in figure 3.3, the most frequent class makes up just over 2.5% of the total amount of data, whereafter the amount of samples for each class decreases drastically. As the number of classes is too high to represent in a single figure it is hard to visualize the size comparisons for all classes.

The Venn–diagram in figure 3.5 shows the overlap of represented classes between the three most frequent languages. In the experiments in the project we do not limit the number of classes to those shared by all or certain languages, but note the fact that classes are not represented equally in the languages.

## Duplicates

Over 33 thousand duplicate samples were identified in the datasets. Not only are there samples containing the exact same text descriptions and class, but there are also samples with identical text descriptions mapping to different classes, and in some cases even to different main groups. This is believed to be caused by a number of factors. Workshops might be

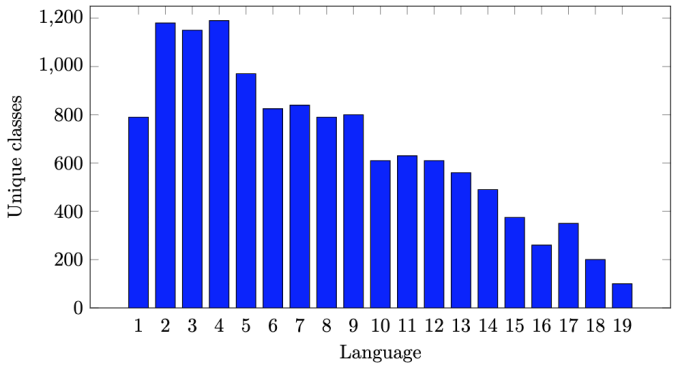



**Figure 3.4:** Number of unique classes represented in major languages ordered by language size.

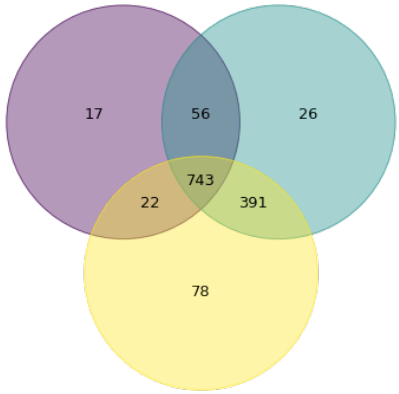



**Figure 3.5:** Overlap of represented classes for the three most frequent languages.

registering several orders simultaneously, and thereby copy-pasting similar texts for different orders to save time in the process. If one describes a problem using information from the dashboard display of a vehicle, some repetition is also expected. While duplicates are not unexpected, the finding that identical texts map to different classes was more so, and believed to contribute to the already existing issues of data complexity through class noise. It is possibly due to the fact that error codes can map to different faults, and that symptoms caused by different faults can be similar. As an example, "engine malfunction" can cover a wide range of faults. Another surprising finding was that within the duplicated samples, the variation in the number of identical samples was large. Some samples were only duplicated once (2 samples in total), whereas other identical samples were encountered over 100 times in the data.

An attempt was also made to investigate if duplicated samples were more common in certain languages. Figure 3.6 presents the total number of samples with the exact same input text (i.e. not considering class) for the different languages in the data. This figure shows that the distribution of duplicates for the different languages is similar to the overall language distribution.

## Sample length

The length of the descriptions varies greatly throughout the data. The shortest samples contain as little as one word, whereas the longest one consists of over 350 words. Figure 3.7 shows the distribution of word counts for the original text. The longer text inputs often consist of conversations between people in the workshop, or long descriptions of conducted troubleshooting or repairs. By manual inspection, it was noted that a large share of the short samples were void of valuable information. Examples of such texts are “Trouble code”, “NOT PAINTED” and “see below”.

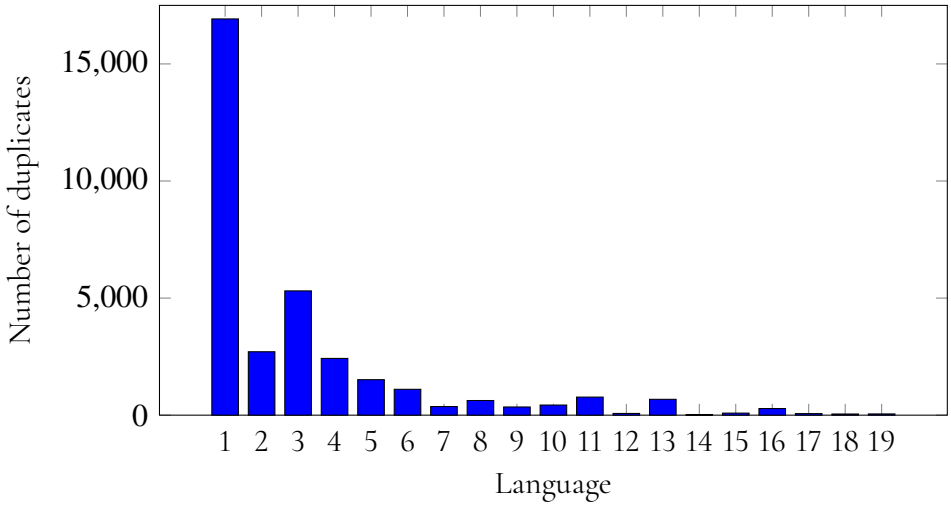



**Figure 3.6:** Number of duplicate samples represented in major languages ordered by language size.

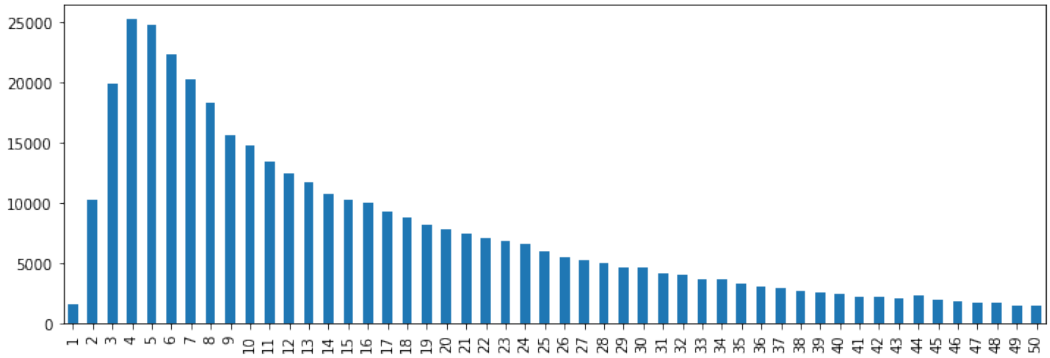



**Figure 3.7:** Distribution of sample length, showing samples containing up to 50 words.

## Special characters & phrases

The data was found to contain many special and non-alphanumeric characters. In total, the original (non translated) data consisted of 191 unique characters, of which 129 of them were not the standard a-z, A-Z, or 0-9. Naturally, many of the non-alphanumeric characters were punctuation characters such as ".", "!", "?" etc. However, since the data is written in different languages, there were many language-specific letters and symbols present in the data. Besides the common punctuation characters, the most frequent non-alphanumeric characters were vowels with different diacritical marks, such as á, å, æ, ę, ǒ, ü etc, in different combinations.

When looking at the translated data, the situation was somewhat different. The translated data consisted of 186 characters in total, meaning that only five unique characters were lost in the translation step. This is probably due to the translation service being unable to translate every single word in a given sentence and thus leaving some words with special characters untranslated. However, after translating the data the frequency of which special characters were present was much lower, as the translation step was shown to remove roughly

93% of the unique instances of special characters in the data.

Manually inspecting the data also showed that it contained a lot of phrases that are relatively common in the data, and does not provide any information regarding the actual problem with the truck. Examples of these phrases are “customer complaint”, “driver complaint”, “attend to”, “vehicle presenting” and other similar phrases. Inspecting to what extent these phrases are present in the data showed that around 7% of the samples in the training dataset contained at least one of these phrases. These phrases were only investigated in the context of translated data, as the phrases in their original language would be too many, making the work cumbersome to perform for all languages.

# Chapter 4
# Methodology

In this chapter we give a brief introduction to the experimental setup of the thesis project. We explain the motivation for the experiments and how they were conducted, as well as how they are evaluated.

## 4.1 Experiment Planning

The purpose of this thesis, as mentioned in the introduction, is to investigate the usefulness of pre-trained transformer-based language models in the context of the company's troubleshooting management. To ensure that the findings are useful and relevant, the experiments conducted were set up with the goal of yielding interpretable models, and results that are possible to evaluate in a manner that can bring insights as to how they could be used for large scale text classification at the company. The process of *CRISP DM* (Cross-Industry Standard Process for Data Mining) [54], seen in Figure 4.1, was followed to create a workflow. This is a framework developed with the purpose of standardizing the process of data mining, making it more efficient, more reliable and more manageable.

Primarily, the goal was to gain a business understanding (see *A* in Figure 4.1) and to understand the actual problem at hand. By determining the business objectives, including the background and success criteria, goals were developed for this thesis and its purpose was clarified in the context of the overarching project at the company, of optimizing the troubleshooting process for trucks. The primary goals identified were to understand the underlying data and how it affected the classification models currently in use, how performance on underrepresented classes and languages could be improved given the imbalances in the data, and if lexical resources and other unrelated sources of text data could be leveraged to improve the models' predictive performance.

Time was also spent on gaining thorough data understanding (see *B* in Figure 4.1) by looking at the data through exploratory analysis, and creating benchmarks and baselines. The findings from these steps were presented and used to form the hypotheses which led to

the initial experiments (see *C* and *D* in Figure 4.1). The results from the initial experiments were evaluated and analyzed to shed new light on the problem context and gain further understanding of the situation before designing the next steps in the experimentation phase (see *E* in Figure 4.1). This led to an iterative process with a focus on utilizing new insights to guide the project which was useful in setting more concrete goals in a an otherwise exploratory project, although it led to revisions in the project plan as the project went along. Note that the deployment step in CRISP-DM, i.e., *F* in Figure 4.1, is beyond the scope of this thesis project. Still, we provide a discussion on deployment aspects for the company context in Chapter 6.

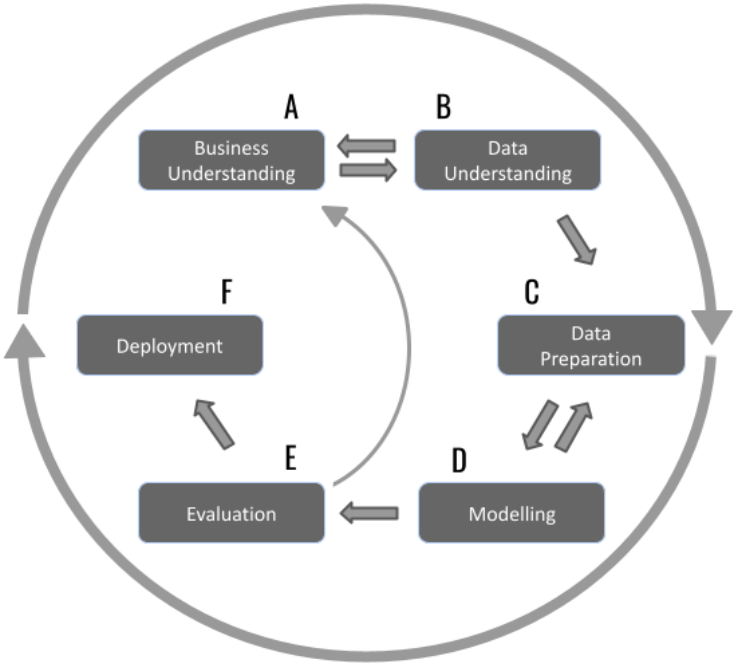



**Figure 4.1:** CRISP DM Process Methodology. Picture adapted from Wirth and Hipp [54].

## 4.2 Overview of the Experimental Design

In the project, we evaluate the models through experiments [55]. Figure 4.2 shows an overview of the experimental design. There are a number of controlled variables that were established through early modeling and kept unchanged throughout the project. The independent variables are those altered in the experiments. The dependent variables are those through which the results of the experiments are evaluated.

### 4.2.1 Controlled Variables

Though all experiments described in the following sections differ with regard to model choice, data pre-processing and datasets used, they all share some of the training configurations and

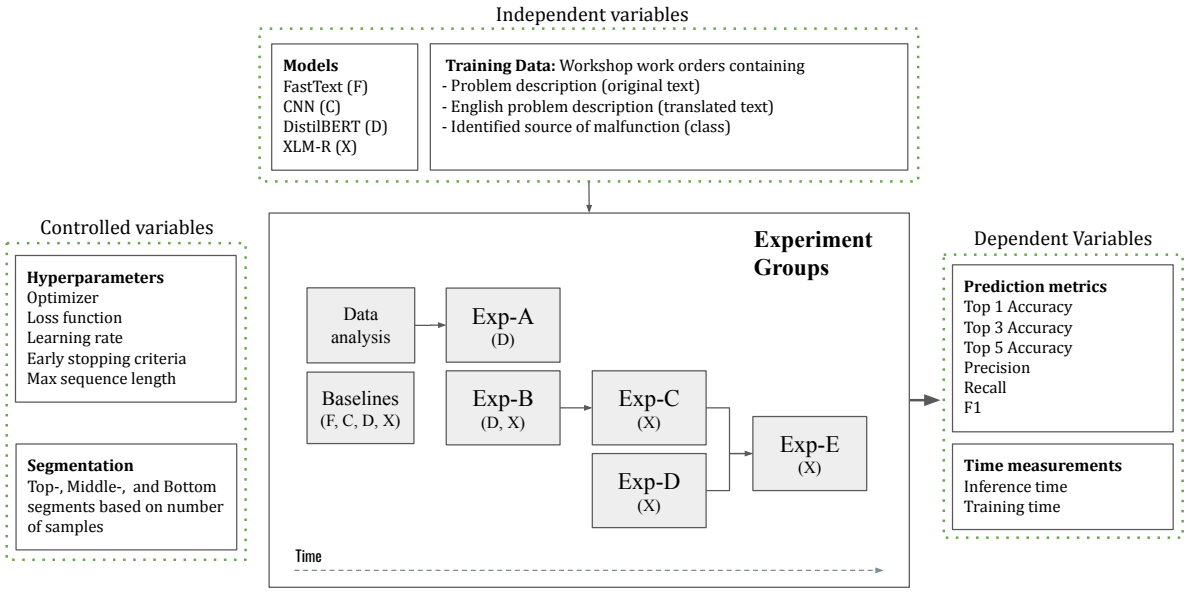



**Figure 4.2:** Experimental Design

hyperparameters. The configurations and hyperparameters throughout the experiments were not changed in order to isolate the variables to what the experiment was designed to investigate. The configurations and hyperparameters are further described in the following sections.

## Optimizer & Learning rate

The chosen optimizer used for all of the experiments was the Adam optimizer, due to it being commonly used to fine-tune BERT-based models [45, 24]. This optimizer uses the stochastic gradient descent method and is computationally efficient, has limited memory requirements, and is suitable for problems that are large in terms of data and parameters [56]. Adam is an adaptive learning rate method, which means that it computes individual learning rates for different parameters, by calculating moving averages of the gradient and the squared gradient. The parameters $\beta_1$ and $\beta_2$ control the decay rates of these moving averages [56]. The chosen learning rate and beta-values are based on early experimentation and the research of Mosbach et al. [24], and are presented in table 4.1.

**Table 4.1:** Optimization parameters.

| Model | Learning Rate | $\beta_1$ | $\beta_2$ |
|---|---|---|---|
| CNN | 5e-4 | 0.9 | 0.999 |
| DistilBERT | 5e-5 | 0.9 | 0.999 |
| XLM-R | 3e-5 | 0.9 | 0.980 |

## Loss function

For all experiments, the categorical cross-entropy loss function was used. This loss function is mainly used in multi-class classification problems, such as the problem at hand in this project. It is computed by the formula:

$$Loss = -\sum_{i=1}^{n} y_i \cdot log\ \hat{y}_i \tag{4.1}$$

where $n$ is the output size (i.e. the number of classes), $\hat{y}_i$ is the model's predicted logit for class $i$, and $y_i$ is the target value. The purpose of the categorical cross-entropy loss function is to take the output probabilities and measure the distance to the true values. Categorical cross-entropy is used in cases where the labels are one-hot encoded (such as in the case of this project), meaning they are in the form of a vector of size $n$, with a one at the index of the class the sample belongs to, and zeros elsewhere.

## Number of epochs & early stopping

When training models for the experiments described in the following sections, early stopping was used in order to avoid overfitting the models to the training data. Early stopping is a form of regularization that dictates how many epochs the model should train for before stopping. If too few epochs are run, there would still be room left for improvement. If too many epochs are run, the model would overfit to the training data and thus lose performance when predicting unseen data, such as the data in the test set. The early stopping was configured to monitor the accuracy on the validation dataset when training the model. This means that after every epoch of training, the model predicts the class of every sample in the validation dataset, and computes the overall validation accuracy. The early stopping terminates the training whenever the validation accuracy stops increasing, meaning that as soon as the model gets an accuracy on the validation dataset that is worse than what it had after the previous epoch, the training is stopped. The motivation for using validation accuracy as a stopping criterion rather than validation loss, which is another common method, is that early modeling showed the models stopping too early when using validation loss as a stopping criterion. Letting the models train for longer, until reaching decreasing validation accuracy, led to increased predictive performance, especially in the bottom segment.

Figure 4.3 is an example of what the training- and validation accuracy could look like during training. In this case, the validation accuracy starts decreasing after six epochs, which is why the training has stopped at this point. If the training were to continue, one could expect the training accuracy to continuously increase, while the validation accuracy will decrease more and more due to the model being overfit to the training data.

## Segmentation of classes and languages

To be able to study the performance with respect to under- and overrepresented classes, the metrics are also presented for grouped segments of classes with varying support. As mentioned before, the support for a class refers to its frequency in the training data, meaning the number of samples belonging to the respective class.

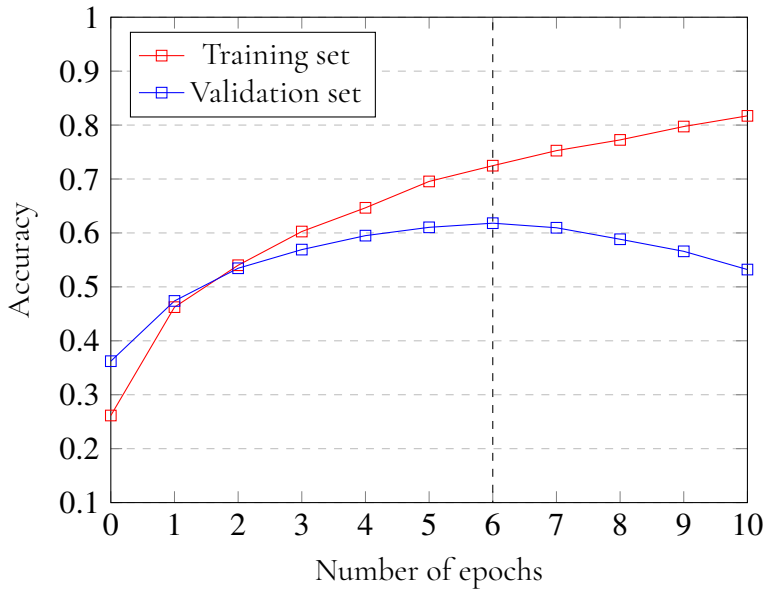



**Figure 4.3:** Example of accuracy on training- and validation set for different epochs. Dashed line shows point after which the model starts overfitting to the training data.

Two possible approaches for segmenting based on support were identified. The first was to select a certain percentage of the total number of classes with the highest and lowest support for each group, and evaluate the performance within these groups respectively. Having the same number of classes in both groups, however, resulted in a large imbalance of samples, where the top group contains a great number of samples, and the bottom a significantly smaller part. To demonstrate; if using a threshold of 25% of the most and the least well-represented classes (same amount of *classes* in each segment), the resulting top segment contains 80 percent of all samples, and the bottom segment contains less than 3 percent. The upper part of Figure 4.4 shows an example where the Top Segment and Bottom Segment contain 366 classes each, but 80% of the data resides in the Top Segment.

The second approach attempted to address this by splitting the groups based on sample size, resulting in different numbers of classes represented in the top and bottom group, but equal size with respect to the share of total samples. The motivation for this is that while it is desirable to perform well on rare classes, achieving good results on a tiny subset of the data might not bring useful insights, compared to evaluating on a larger set. In the case of fault classification for the company, it is also not as important to achieve great results on a very minor fraction of the samples, but instead more valuable to understand the performance gain on a larger portion of the data which might still be categorized as underrepresented. The segmentation of classes was therefore done as described in the second approach mentioned above. While this leads to equally sized top and bottom segments, it does as mentioned before result in the bottom segment having many more unique classes represented. The lower part of Figure 4.4 shows an example where the Top Segment and Bottom Segment contain 25% of the data each, corresponding to 27 classes in the Top Segment and 1,076 classes in the Bottom Segment.

Within each segment, the method of presenting an aggregate performance metric is to

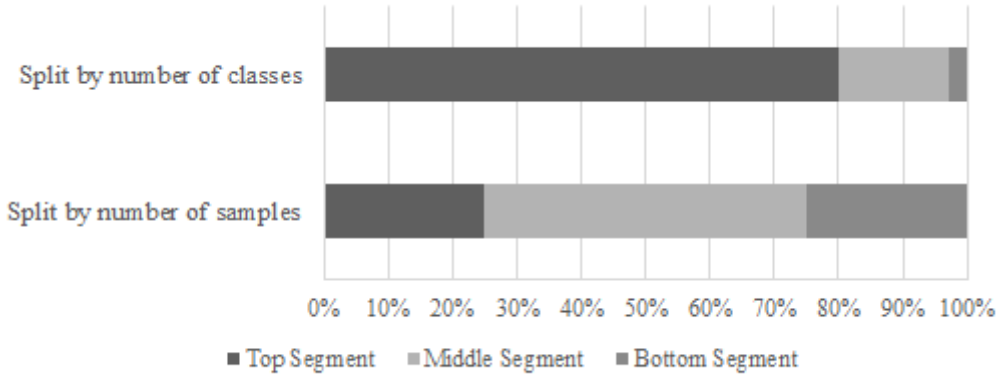



**Figure 4.4:** Visualization of the two identified methods for class segmentation based on support. Graph shows percentage of test samples in each segment for each method.

take the macro average of all classes belonging to each segment. The motivation for this is to be able to present interpretable and comparable results between models. A consequence of this is that the $F_1$-score presented for each group does not always fall between the values of precision and recall, which is normally the case. This is explained by the fact that the $F_1$-score is more heavily affected by the lower value of its components, and if precision and recall alternate in being the larger value between different classes, their averages can both end up being larger than the average $F_1$-score.

Evaluation with regards to language is done by calculating the language-wise accuracy and $F_1$-score.

### Environment

To run the experiments, the company offered access to two nodes in their high-performance computing cluster. The software and hardware that were used to produce the acquired results are presented in Table 4.2. The experimentation was performed using the programming language *Python* and the deep learning framework *TensorFlow*.

**Table 4.2:** Hardware and software setup.

| Operating system | Python version | TensorFlow version | GPU |
|---|---|---|---|
| Ubuntu 16.04.4 LTS | 3.7.11 | 2.3.4 | Nvidia Tesla P100-SXM2 |
| Ubuntu 16.04.6 LTS | 3.7.11 | 2.3.4 | Nvidia Tesla K80 |

## 4.2.2 Independent Variables

The independent variables are those altered during the experiments, the effect of which is studied and evaluated to understand the impact with regards to the dependent variables.

### Transformer models

Through the experimentation phase, pre-trained DistilBERT and XLM-R models were used. The models and their corresponding tokenizers were accessed through the *HuggingFace* Transformers library which provides open-source access to pre-trained transformer-based language models [57]. The Transformers library contains many different versions of these models, and the ones that were chosen when running the experiments were *TFDistilBERTForSequenceClassification* and *TFXLMRobertaForSequenceClassification*. These models are designed to be used with TensorFlow, and they consist of the respective base model with an added classification head on top, making them suitable for classification tasks. As for the models, the tokenizers used in the experiments were also accessed through the Transformers library. These tokenizers were also pre-trained (meaning that the tokens are not generated from the data related to this project). Each tokenizer is associated with a particular model and adapted to the data on which that model was pre-trained. Throughout all experiments, the maximum sequence length was chosen to 256. Sequence length refers to the number of tokens in each sample, and if the sample is over 256 tokens long, it is truncated to match the maximum sequence length.

### Training data

As will be explained later on in Section 4.4, the experiments are focused around studying the effect of training data through various methods of modifying and altering the training data.

## 4.2.3 Dependent Variables

The dependent variables in this project are the metrics through which we evaluate the predictive performance of the models, meaning the model's ability to correctly classify the samples in the datasets, and the models' performance with regards to training and inference time.

### Prediction metrics

To achieve interpretable and meaningful results a number of metrics are used. These are *top k categorical accuracy* (k=1,3,5), precision, recall, and $F_1$-score. Top $k$ categorical accuracy is defined as the number of times the true label is among the $k$ classes predicted with the highest probability, which in the case of $k = 1$ is simply traditional accuracy. Since a possible future use-case is using the model to predict the most likely causes for a vehicle malfunction based on a human-generated description, it is not necessarily crucial that the true class is the number one prediction. As it is possible to generate a list of likely causes, understanding how the model performs with regards to top 3 and top 5 categorical accuracy can be of importance. Precision and recall can give good insights into how mistakes are made if they are done within segments or in-between segments.

### Inference and training time

To measure the inference time of the transformer-based models studied in this project, the *TensorFlow Benchmark* class from the HuggingFace library *Transformers* was used. Inference time refers to the time it takes for the model to make predictions for one batch of data.

Inference time was studied for several configurations of different batch sizes and sequence lengths, presented in table 4.3. The environment specifications for testing is presented in table 4.4.

**Table 4.3:** Configurations for testing inference time.

| Hyperparameter | Values |
|---|---|
| Batch size | 1, 4 |
| Sequence Length | 8, 32, 128, 256 |

**Table 4.4:** Environment for test of inference time.

| | |
|---|---|
| **Transformers version:** | 4.10.3 |
| **Framework:** | TensorFlow |
| **Framework version:** | 2.5.0 |
| **Python version:** | 3.6.4 |
| **OS:** | Windows |
| **OS Version:** | 10.0.19042 |
| **CPU:** | Intel64 Family 6 Model 165 Stepping 2 |

## 4.3 Establishing Baselines

By establishing baselines, the results obtained from experiments can be compared and analyzed with more nuance. As a first baseline for this thesis, a non-deep learning model was first trained to study whether the use of more complex models was justified for this application, and thereafter a baseline for deep learning models was set to create comparability with later experiments. It was also done to create comparability to the previous research done at the company.

### 4.3.1 FastText Model

FastText was used to understand the lexical aspect of the problem and to be able to contrast the performance of deep learning models with more lightweight and efficient methods. Lexicality in this context refers to the vocabulary, words, or morphemes in the language, as a hypothesis from the data analysis was that much semantic information in the samples is captured in key words and technical terms. While deep learning models can capture context due to their size and complexity, certain key words might be more important in determining the class of a sample. If a FastText-model, having a size and training time being only a fraction of that of transformer-based models can generate high performance, it motivates studying trade-offs between performance, training-time and costs of deployment. The goal with this baseline is therefore to be able to evaluate the possible gains from using deep learning methods in the light of increased time and costs for both training and deployment in future applications.

A classifier was trained on the monolingual (translated) training data, using FastText word embeddings generated on the corpus, with a multinomial logistic regression classifier head. The motivation for using FastText to generate word embeddings is the efficient implementation it gives and its documented performance [58].

### 4.3.2 Deep Learning Models

As a baseline for the deep learning models, convolutional neural networks were used. Two separate models were trained, one on the translated (monolingual) texts and one on the original (multilingual) texts. This was done using custom SentencePiece tokenizers with a vocabulary size of 40,000 tokens, trained on all three datasets (training, validation, and testing), in the language domain of the respective model. The motivation for using convolutional neural networks was that it goes in line with previous research done at the company, and the model architecture was based on that of previous models. Moreover, convolutional neural networks have recently been successfully applied for text classification in various applications [59]. While recurrent neural networks work well for the NLP tasks where comprehension of long range semantics is required (e.g., Question-Answering), convolutional neural networks work well where detecting local and position-invariant patterns is required, such as key phrases [60, 59].

DistilBERT was used to set baselines for the performance on the translated data, to be able to evaluate the possible gains from using multilingual models, and the reduced need for automatic translation of large quantities of text. The motivation for using DistilBERT in favor of the original BERT was its documented performance compared to the original BERT, while being more lightweight and having significantly fewer parameters, thereby reducing training time [28, 48].

XLM-R was used when working with the original, multilingual data, as it is a BERT-based model showing state-of-the-art performance on multilingual tasks [30].

It is worth noting the difference in the use of the phrase "training a model" with regards to the convolutional neural networks and the transformer-based models. The CNNs are trained from scratch, while DistilBERT and XLM-R are *fine-tuned* (see Section 2.3.4), although we refer to both of these processes as "training" in this project.

## 4.4 Modelling and Experimentation

This section introduces the experiments conducted as part of this thesis project. The experiments are organized into *effect of cleaning data* (Exp-A1 – Exp-A5), *multilingualism* (Exp-B1 – Exp-B2), *language-wise performance* (Exp-C), *oversampling* (Exp-D1-D4) and *data augmentation through translation* (Exp-E1 – Exp-E2).

### 4.4.1 Experimental Group Exp-A: Effect of Cleaning Data

When studying the data as described in Chapter 3, many interesting characteristics were identified that were believed to possibly impact the model's learning. Based on these findings,

a number of experiments were designed to study the effect of different inherent traits of the data on model performance (as related to research question RQ1) by cleaning the data before training.

These experiments were performed using a monolingual DistilBERT model. In order to make the datasets applicable for a monolingual model that takes in the text samples translations as input, a general pre-processing step removing all samples without a translation was performed. A key hypothesis for this experiment was that based on the findings from the data analysis, cleaning the data could improve training and lead to an understanding of how the different characteristics in the data affected the models and increased interpretability. The experiments were performed by using different steps of pre-processing for the training data, but the test set was left untouched for all pre-processing configurations. The following parts describe the different pre-processing configurations in more detail. With these different pre-processing configurations a monolingual DistilBERT model was trained on each of the four pre-processing steps individually (Exp-C1 – Exp-C4), as well all of them at the same time (Exp-C5). For validation purposes, and to study the variability in the results, models were trained four times for each configuration.

## Exp-A1: Removing duplicate samples

As mentioned in Section 3.2, the data contained two different forms of duplicates that needed to be handled differently. Firstly, there were many duplicates that were identical with regard to both input text, main group, and class. For this pre-processing step, all but one of these identical samples were removed, keeping one in order to actually keep the information given in these samples. However, there were also duplicates contributing to *class noise*, meaning duplicate samples where the input texts were identical, but the samples were mapped to more than one class. It is easy to see how this could prove confusing to the model, and in an attempt to reduce the class noise, all of these duplicates were removed. These two steps removed around 11,9% of the data from the training data.

## Exp-A2: Removing non-alphanumeric characters

This pre-processing step consisted of making all text inputs in the training data alphanumeric. That meant removing all characters from the data that were something other than standard Latin alphabet letters or digits. As with the case for standard phrases, the non-alphanumeric characters were removed from the data, but the entire sample was not removed if the data contained any non-alphanumeric characters.

## Exp-A3: Removing long and short input texts

As described in chapter 3, the data contained many samples that were either very short (just a few words), or very long (over 350 words). Given the lack of information in the shortest samples, and the typically messy nature of the longest samples, the next pre-processing configuration consisted of removing the shortest and the longest samples from the training data. Based on manual inspection, the lower threshold was chosen to four words, removing samples with a length of three words and less. The upper threshold was in a similar fashion chosen to 67, removing samples with a length over 67 words. While it might seem arbitrary to draw

a sharp line at 67, the upper threshold was chosen in a range where the quality of samples was deemed to diminish and set to remove as close to 10% of the data as possible, and around equally many under the short threshold as over the long threshold. Note that the samples having a text shorter than 4 words or longer than 67 words were completely removed, and the long texts were thus not shortened to fit the interval.

### Exp-A4: Removing standard phrases

Another pre-processing configuration consisted of removing the identified standard phrases from the data. The standard phrases were as mentioned in section 3.2 phrases that were common in the data, but did not carry any information regarding the actual problem with the vehicle. The identified phrases that were removed from the data were the following:

**Table 4.5:** Standard Phrases Removed in Exp-A4.

| | | |
|---|---|---|
| "customer complaint" | "customer complained" | "customer complains" |
| "customer complaints" | "customer request" | "customer requested" |
| "customer reports" | "attend to" | "vehicle presenting" |
| "vehicle presented" | "driver complains" | "customer is complaining" |
| "vehicle presents" | "driver complaint" | "driver complained" |
| "driver complaints" | "driver request" | "driver requested" |

The standard phrases were removed from the text inputs in the training and validation set, but the whole sample was not removed if it contained one of the identified phrases.

### Exp-A5: Full Pre-Processing

As a final step in this experiment group, all four pre-processing methods were tested at the same time.

## 4.4.2 Experimental Group Exp-B: Multilingualism in the Dataset

When implementing multilingual models that use the original text input instead of the translated version, it was of interest to compare the predictive performance of models from both the monolingual and the multilingual domain, and study the effect of the translation step and whether it can be omitted (as related to research question RQ1). As there was a large number of samples where a translation had not been generated from the translation service, the multilingual data contained samples that had no corresponding counterpart available in the monolingual domain. Two ways of performing the comparisons were therefore identified to achieve fair comparability, where the models are trained and evaluated on comparable data.

### Exp-B1: Removing samples without available translation

First, it was of interest to perform a comparison where the multilingual dataset is reduced by removing the samples not available in the translated dataset. To accomplish this, all samples that were not able to be used when training a DistilBERT model were also removed when training an XLM-R model. Those were samples for which the translation service had not been able to identify a language, leaving the sample without a translated text. Removing these samples removed around 3.7% of the samples in the datasets when training the XLM-R model.

### Exp-B2: Company use-case comparison

Secondly, a comparison between the two models based on a more true use-case scenario was performed, as data in a real production case would not likely be discarded. Instead it would likely be replaced or at least translated using information regarding the origin of the workshop order from which the sample originated. It was therefore of interest to also compare the predictive performance of models in both the mono- and multilingual domain where these samples were included.

To perform this experiment a DistilBERT model was trained, and in all cases where a sample was missing a translated text, the original text was used instead. This means the monolingual model will see certain samples in other languages than English during training, but no data is discarded.

### Exp-B3: Training DistilBERT on multilingual data and XLM-R on monolingual data

An experiment was also conducted where the two models were trained and evaluated on switched data, i.e. where the DistilBERT model was trained on original data and the XLM-R model was trained on translated data. It was of interest to compare an XLM-R model trained on translated data to the DistilBERT baseline to see if a model being pre-trained in only English performs better if the subsequent classification task is only in the English domain, or if being pre-trained on additional languages helps the model perform better. The DistilBERT model was trained on original data in order to compare the results to the XLM-R baseline, and to see how a model being pre-trained using only English data would perform when the subsequent classification task was in a multilingual domain. This can give an understanding regarding whether it is the model size and complexity, or the model's pre-trained language understanding which is of benefit when using these pre-trained models.

## 4.4.3 Experimental Group Exp-C: Language-Wise Performance

As the languages are not evenly represented in the data, combined with the fact that the models used during the project are exposed to different amounts of data for separate languages during pre-training, a need was identified to understand how the model's performance for separate languages is affected by the amount of data it sees in that language during fine-tuning (as related to research questions RQ1 and RQ2). To test this an experiment was constructed

in which over-represented languages with high accuracy were artificially reduced to low levels, and models were trained with gradually increasing amounts of data in these languages present. By studying the relative performance, both overall but also for each of the mentioned languages separately, with the goal of gaining insights into how a model can learn in an individual language. Note that the two languages in this experiment are anonymized and referred to as X and Y. Starting with Language X from Figure 3.2, new subsets of data were created from the original training set, where all samples in Language X were removed and gradually replaced in different quantities (20, 40, 60, and 80 percent) through random sampling, after which XLM-R was trained on each of the sets respectively. The experiment was after this repeated where a dataset where all samples from Language Y were removed, and replaced in the same absolute quantities (rather than relative) as in the case of Language X (resulting in partitions of 8,16, 24, 32 and 40 percent of the Language data). The reason for not replacing all data in Language Y was to get an understanding of how the absolute number of samples for each language affected performance, and as the experiment in Language X showed a large early increase followed by diminishing returns, as shown in the next chapter.

An important aspect to keep in mind is that not all classes are represented equally in the different languages, and that there are variations in the amounts of unique classes present in each language.

### 4.4.4 Experimental Group Exp-D: Oversampling

To study the effect of using sampling techniques in an attempt to improve model accuracy for infrequent classes (as related to research question RQ2), four experiments were designed in which underrepresented classes were oversampled to even out the class distribution. Based on the previously mentioned theory of oversampling, the hypothesis was that by duplicating samples belonging to one of the classes in the "low-support" category, one could expect the model's predictive performance on these classes to improve. Even though the model will already have seen all the samples during every epoch of training and no new information is added by performing this oversampling, the dataset will be more balanced and the skewness in class distribution will be smaller.

A certain threshold was selected for each experiment, which was used as the minimum number of samples per class, shown in Table 4.6. All classes containing fewer samples than the selected threshold were upsampled in the training set, using the method described in Algorithm 1. An XLM-R model was thereafter trained on the different datasets.

**Table 4.6:** Thresholds for experimental group Exp-D.

| Experiment | Threshold |
|---|---|
| Exp-D1 | 10 |
| Exp-D2 | 20 |
| Exp-D3 | 30 |
| Exp-D4 | 50 |

The motivation for never adding more than 20 additional samples for any given class during oversampling (see Algorithm 1) is to avoid the least frequent classes being duplicated

**Algorithm 1** Oversampling

$C \leftarrow$ All classes in training set $\{c_1, ..., c_i, ..., c_N\}$
$X \leftarrow$ Threshold
**for** all classes in $C$ **do**
    $s_i \leftarrow$ support of $c_i$
    **if** $s_i < X$ **then**
        $n \leftarrow min(X - s_i, 20)$
        Random sample $n$ from $C_i$ with replacement and add to training set
    **end if**
**end for**

to a point where it fails to generalize. It only affects Exp-D3 and Exp-D4 as those are the cases where the difference between a class' support and the threshold can exceed 20.

### 4.4.5 Experimental Group Exp-E: Augmentation Through Unidirectional Translation

After seeing the results from Exp-C (presented in section 5.5), showing the effect of increased amounts of original data, and the results of Exp-D (presented in section 5.6), showing that oversampling can improve the predictive performance on underrepresented classes, it was of interest to study whether using the available translations could be used to augment the datasets. To clarify, the term "augmentation" is used when utilizing the translated texts as new versions of the samples' original texts. As discussed in section 2.6.1 unidirectional translation can be used as a way to augment data in NLP. Two experiments were set up, comparing the effect of using translated data to that of original data.

#### Exp-E1: Augmenting data for infrequent languages

The following experiment was designed to study the effect which an increase in data of a certain language has, depending on how well represented the language is and on how much data is used to augment the training set. The goal was to quantify the relationship between an increase in data and a possible performance gain as well as to investigate whether there was a point of diminishing returns where adding more data does not bring additional benefit. Additionally, the experiment aimed to study whether "synthetic" data could be an alternative to producing more original labeled data. The synthetic data in this case refers to all the translated samples that are available but not utilized when working in the multilingual domain. Since English was the only language in which translated samples were available, the experiment was designed to study the effect of augmenting data only for this language. In order to investigate the possible increase in performance based on how well represented the language originally is, two different configurations were run. To test the effect of augmenting data for low represented languages, two starting points of 10% and 20% of the original English data respectively were augmented with synthetic data up to a point where the total amount of data matched the original amount of English. For the second configuration the same approach was done, but this time using starting levels at 80% and 100% of the English data and augmenting up to a point where to total amount of data matched 300% of the original amount of English.

### Exp-E2: Augmenting data for infrequent classes

This experiment was again designed to study the effect of a larger training dataset, but this time when augmenting the dataset with respect to infrequent classes instead of languages. The reasoning for using this setup was again to get an understanding for if any information was lost in the translation step, or if this technique would yield similar results to those seen in Exp-D (section 4.4.4). Sampling was done following the process described in Algorithm 1, using a threshold of 50, while substituting the original text of the samples selected to augment the dataset with its English translation. The datasets used for validation and testing were kept in their original form. The experiment was performed using a multilingual XLM-R model.

# Chapter 5
# Results

In this chapter we present the results from the conducted experiments. All precision, recall and $F_1$ scores correspond to macro-averages as described in section 2.5. Numbers highlighted in bold are the highest value within each metric and segment (top, middle and bottom as described in section 4.2.1) for a set of tables presented together. To give a brief reminder, the segmentation based on class support is done so that the top and bottom segments include the samples belonging to the most over- and underrepresented classes respectively, so that they both contain 25% of the data.

## 5.1 FastText Baseline

The results from the FastText baseline are presented in table 5.1. The findings from this shallow model show that performance varies a lot between segments, and that the overall accuracy is just over 20%. With the larger precision values for the top segment and low recall values for the bottom segment it is clear that the model has a bias towards the over-represented classes in the data. In the process of training and evaluating the FastText model it was not possible to get metrics for top 3 and top 5 categorical accuracy the same way as when training deep transformer-based models, which is why these metrics are not presented in this section.

**Table 5.1:** FastText Benchmark

| Segment | **Total** | **Top** | **Mid** | **Bottom** |
|---|---|---|---|---|
| Accuracy | 0,203 | 0,253 | 0,235 | 0,089 |
| Precison | 0,205 | 0,475 | 0,378 | 0,158 |
| Recall | 0,090 | 0,250 | 0,212 | 0,057 |
| F1 | 0,109 | 0,287 | 0,248 | 0,073 |

## 5.2 Deep Learning Baselines

The results from models using convolutional neural networks are shown in 5.2 and 5.3. They show that the models reach accuracy levels of over 50%, and top 3- and top 5 categorical accuracy levels of over 67% and 72% respectively. The results from the two models show only minor differences, considering the differences in languages present in the datasets to which they have been exposed, and the fact that they are trained on tokenizers with equally sized vocabularies (i.e, number of tokens).

**Table 5.2:** Convolutional neural net on monolingual data (translated texts).

| Top 3 Cat. Acc. | | 0,678 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,732 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,513 | 0,706 | 0,548 | 0,238 |
| Precison | 0,379 | 0,633 | 0,541 | 0,335 |
| Recall | 0,238 | 0,692 | 0,491 | 0,169 |
| F1 | 0,265 | 0,643 | 0,495 | 0,203 |

**Table 5.3:** Convolutional neural net on multilingual data (original texts).

| Top 3 Cat. Acc. | | 0,675 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,724 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,523 | 0,724 | 0,563 | 0,228 |
| Precison | 0,349 | 0,641 | 0,532 | 0,300 |
| Recall | 0,233 | 0,708 | 0,505 | 0,158 |
| F1 | 0,254 | 0,657 | 0,498 | 0,188 |

Tables 5.4 and 5.5 present the baselines for the DistilBERT model trained on translated texts and the XLM-R model trained on original texts. Noteworthy is that samples with unknown predicted language, and thus not having a translation, were not applicable when training and evaluating the DistilBERT model. These samples are however included when training the XLM-R model, resulting in the XLM-R model being trained and evaluated on 3.7% more data. The results show that the XLM-R model marginally outperforms the DistilBERT model for the top- and middle segment, but that the DistilBERT model achieves much better results on the bottom segment, as well as higher top 3 and top 5 categorical accuracy. Table 5.6 shows that the two models perform equally well on most of the languages, but that the accuracy between different languages differs significantly. The results show that both of the pre-trained models outperform the convolutional networks by a margin of around 10 percentage points in most metrics and segments. The exception is that of the performance of XLM-R in the bottom segment, which actually shows an $F_1$-score lower than that of both convolutional neural networks, and an overall $F_1$-score on par with the two.

**Table 5.4:** DistilBERT Baseline

| Top 3 Cat. Acc. | | **0,779** | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | **0,826** | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | **0,619** | 0,815 | 0.643 | **0.370** |
| Precison | **0,440** | **0,746** | **0,600** | **0,397** |
| Recall | **0,365** | 0,794 | 0,601 | **0,290** |
| F1 | **0,364** | 0,765 | 0,585 | **0,305** |

**Table 5.5:** XLM-R Baseline

| Top 3 Cat. Acc. | | 0,754 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,792 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,613 | **0,820** | **0,666** | 0,298 |
| Precison | 0,284 | 0,745 | 0,579 | 0,204 |
| Recall | 0,264 | **0,802** | **0,630** | 0,166 |
| F1 | 0,254 | **0,770** | **0,593** | 0,162 |

**Table 5.6:** Accuracy per language for the DistilBERT model and the XLM-R model (%)

| Language | DistilBert | XLM-R |
|---|---|---|
| 1 | 71,3 | **72,4** |
| 2 | 61,3 | **61,4** |
| 3 | **61,0** | 58,5 |
| 4 | N/A | 44,3 |
| 5 | 48,3 | **48,5** |
| 6 | 52,4 | **53,2** |
| 7 | **62,8** | 60,5 |
| 8 | 55,3 | **54,0** |
| 9 | 61,0 | **61,4** |
| 10 | **57,4** | 57,1 |
| 11 | **38,0** | 36,0 |
| 12 | 62,4 | **64,8** |
| 13 | **57,8** | 55,5 |
| 14 | 61,7 | **62,0** |
| 15 | **55,1** | 50,5 |
| 16 | 46,2 | **54,6** |
| 17 | **51,4** | 46,3 |
| 18 | **54,5** | 49,7 |
| 19 | **40,8** | **40,8** |

**Table 5.7:** Confustion matrix for the 10 largest classes (DistilBERT)

| | 5 | 16 | 12 | 2 | 0 | 41 | 159 | 140 | 64 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|
| **5** | 2154 | 88 | 0 | 2 | 1 | 2 | 0 | 1 | 0 | 1 |
| **16** | 113 | 1534 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **12** | 0 | 0 | 1374 | 53 | 0 | 3 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 41 | 1181 | 0 | 2 | 0 | 0 | 0 | 0 |
| **0** | 1 | 0 | 1 | 1 | 1126 | 1 | 0 | 0 | 0 | 0 |
| **41** | 2 | 0 | 0 | 2 | 0 | 1081 | 0 | 0 | 2 | 0 |
| **159** | 1 | 1 | 0 | 1 | 0 | 0 | 735 | 128 | 0 | 1 |
| **140** | 4 | 0 | 1 | 1 | 0 | 2 | 161 | 762 | 0 | 1 |
| **64** | 10 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 890 | 0 |
| **90** | 2 | 1 | 0 | 1 | 0 | 0 | 1 | 2 | 0 | 645 |

**Table 5.8:** Confustion matrix for the 10 largest classes (XLM-R)

| | 5 | 16 | 12 | 2 | 0 | 41 | 159 | 140 | 64 | 90 |
|---|---|---|---|---|---|---|---|---|---|---|
| **5** | 2305 | 67 | 0 | 0 | 1 | 2 | 0 | 2 | 2 | 0 |
| **16** | 159 | 1595 | 0 | 2 | 0 | 0 | 0 | 0 | 1 | 1 |
| **12** | 0 | 1 | 1487 | 42 | 2 | 2 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 94 | 1216 | 0 | 1 | 0 | 0 | 0 | 0 |
| **0** | 0 | 0 | 1 | 1 | 1134 | 0 | 0 | 0 | 0 | 0 |
| **41** | 2 | 0 | 0 | 2 | 0 | 1083 | 0 | 0 | 1 | 0 |
| **159** | 1 | 0 | 0 | 1 | 0 | 0 | 794 | 117 | 1 | 0 |
| **140** | 7 | 1 | 1 | 0 | 0 | 2 | 184 | 790 | 1 | 1 |
| **64** | 12 | 2 | 0 | 0 | 0 | 1 | 0 | 1 | 906 | 0 |
| **90** | 3 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 660 |

Table 5.7 and 5.8 show the confusion matrices for the DistilBERT model and the XLM-R model for the ten most frequent classes. In the figures, the rows correspond to the true class and the columns to the predicted class. The high values on the diagonal of the matrices imply that both models predict the top 10 classes well, but there are some inconsistencies, most notably for classes 5, 16, 140 and 159.

## 5.3 Exp-A: Effect of Cleaning Data

The results from experiment group A (4.4.1) which studies the effect of different pre-processing configurations for the DistilBERT model are presented in tables 5.9 to 5.14. The results are averages from four runs for each configuration. The results are very similar, indicating that

there is no significant benefit from either configuration when comparing to the baseline model. The best results overall are achieved when not doing any pre-processing more than removing NaN-values. Removing standard phrases (Exp-A4, 4.4.1) is the only configuration that performed better in any of the segments, by a margin that is essentially insignificant.

**Table 5.9:** Baseline: Only Remove NaN

| | Average | | | Max |
|---|---|---|---|---|
| Top 3 Cat. Acc. | 0,777 | | | 0,778 |
| Top 5 Cat. Acc. | 0,825 | | | 0,826 |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy (Avg) | **0,618** | 0,808 | 0,645 | **0,371** |
| Precison (Avg) | **0,426** | **0,750** | 0,598 | **0,380** |
| Recall (Avg) | **0,354** | 0,790 | 0,601 | **0,287** |
| F1 (Avg) | **0,359** | **0,767** | 0,584 | **0,297** |

**Table 5.10:** Exp-C5: Fully Processed

| | Average | | | Max |
|---|---|---|---|---|
| Top 3 Cat. Acc. | 0,764 | | | 0,766 |
| Top 5 Cat. Acc. | 0,813 | | | 0,816 |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy (Avg) | 0,603 | 0,796 | 0,632 | 0,349 |
| Precison (Avg) | 0,408 | 0,740 | 0,589 | 0,358 |
| Recall (Avg) | 0,335 | 0,780 | 0,586 | 0,266 |
| F1 (Avg) | 0,340 | 0,757 | 0,572 | 0,276 |

**Table 5.11:** Exp-C1: Make Alphanumeric

| | Average | | | Max |
|---|---|---|---|---|
| Top 3 Cat. Acc. | 0,775 | | | 0,777 |
| Top 5 Cat. Acc. | 0,824 | | | 0,826 |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy (Avg) | 0,612 | 0,810 | 0,642 | 0,351 |
| Precison (Avg) | 0,418 | 0,739 | 0,592 | 0,371 |
| Recall (Avg) | 0,337 | 0,791 | 0,594 | 0,267 |
| F1 (Avg) | 0,343 | 0,760 | 0,576 | 0,279 |

**Table 5.12:** Exp-C2: Remove Duplicates

| | Average | | | Max |
|---|---|---|---|---|
| Top 3 Cat. Acc. | 0,773 | | | 0,775 |
| Top 5 Cat. Acc. | 0,821 | | | 0,823 |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy (Avg) | 0,612 | 0,806 | 0,643 | 0,351 |
| Precison (Avg) | 0,417 | 0,741 | 0,592 | 0,370 |
| Recall (Avg) | 0,339 | 0,787 | 0,597 | 0,268 |
| F1 (Avg) | 0,345 | 0,760 | 0,579 | 0,280 |

**Table 5.13:** Exp-C3: Long and Short Texts

| | Average | | | Max |
|---|---|---|---|---|
| Top 3 Cat. Acc. | 0,771 | | | 0,772 |
| Top 5 Cat. Acc. | 0,820 | | | 0,822 |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy (Avg) | 0,610 | 0,811 | 0,639 | 0,347 |
| Precison (Avg) | 0,410 | 0,729 | 0,591 | 0,361 |
| Recall (Avg) | 0,333 | 0,793 | 0,592 | 0,262 |
| F1 (Avg) | 0,339 | 0,755 | 0,575 | 0,274 |

**Table 5.14:** Exp-C4: Remove Standard Phrases

| | Average | | | Max |
|---|---|---|---|---|
| Top 3 Cat. Acc. | **0,779** | | | **0,780** |
| Top 5 Cat. Acc. | **0,827** | | | **0,829** |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy (Avg) | 0,617 | **0,813** | **0,647** | 0,357 |
| Precison (Avg) | 0,412 | 0,748 | **0,603** | 0,359 |
| Recall (Avg) | 0,341 | **0,795** | **0,602** | 0,269 |
| F1 (Avg) | 0,343 | **0,767** | **0,585** | 0,277 |

To give nuance to the results and understand how much the results varied between training runs, the box-plots in Figure 5.1 and Figure 5.2 show the variability in accuracy and F1-score for the different segments between the different runs. Each graph represents a class segment and each box corresponds to a pre-processing configuration. The box-plots show the mean (mid line), mean ± standard error (box edges), and maximum/minimum value (whiskers). As can be seen in the box-plots, the overall results ((i) in Figure 5.1) vary next to nothing between the training runs. While there is a little more variation within the segments, the differences are at most a single percentage point. A key finding is that the bottom segment was consistently negatively impacted by the modifications to the dataset.

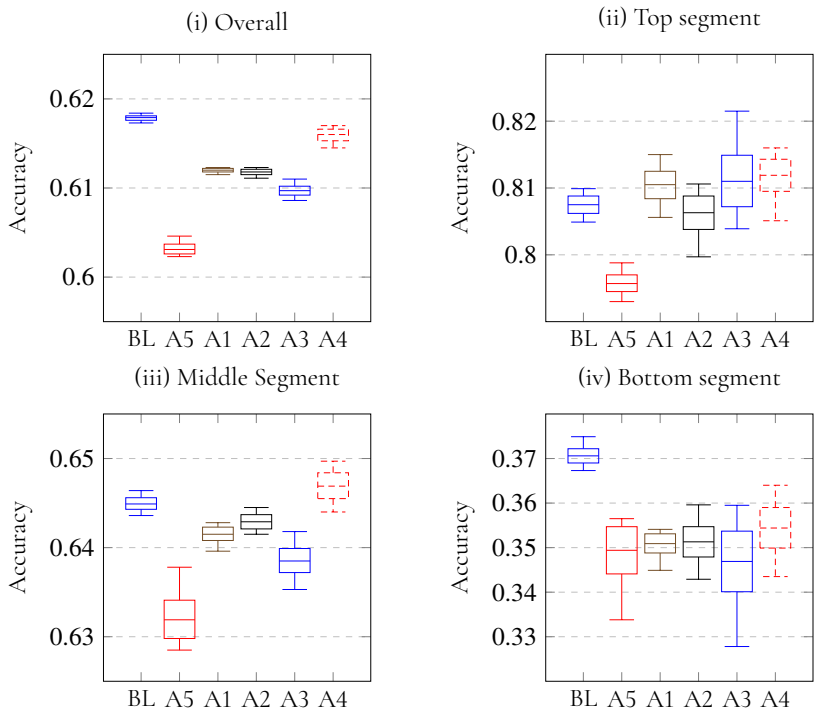



**Figure 5.1:** Variability in accuracy, overall and within each segment (i-iv), for baseline DistilBERT (BL) and each pre-processing strategy (A1-A5). Note the difference in scale on the y-axes.

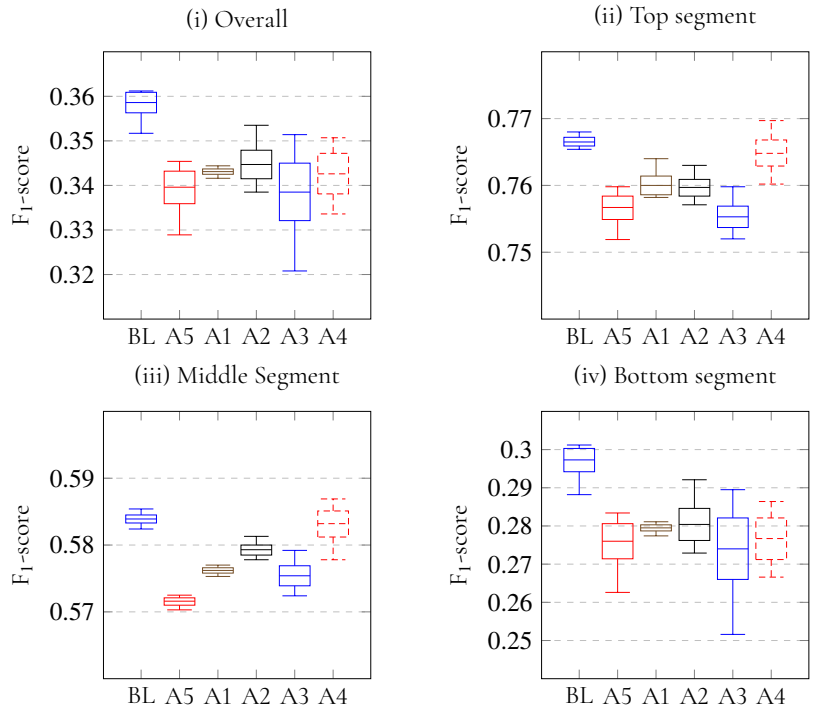



**Figure 5.2:** Variability in $F_1$-score, overall and within each segment (i-iv), for baseline DistilBERT (BL) and each pre-processing strategy (A1-A5). Note the difference in scale on the y-axes.

## 5.4 Exp-B: Multilingualism in the Dataset

The results from experiment group B (4.4.2) are presented in tables 5.16 to 5.17. This experiment was aimed at comparing the monolingual DistilBERT model and the multilingual XLM-R model in a context where the two models were trained on comparable data, and in a context based on the more realistic use-case scenario where samples without translation were not discarded. The results show that in both cases the XLM-R model performs better for the top- and middle segments of the classes but performs significantly worse for the bottom segment, while the overall accuracy for both models is very similar. When removing samples without available translations (Exp-B1), the distribution of the removed samples was the following: 34% belonged to the top segment, 44% belonged to the middle segment and 22% belonged to the bottom segment.

**Table 5.15:** DistilBERT: Using original text for samples without translations

| Top 3 Cat. Acc. | | **0,775** | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | **0,822** | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | **0,614** | 0,812 | 0,634 | **0,374** |
| Precison | **0,421** | 0,734 | **0,591** | **0,377** |
| Recall | **0,354** | 0,793 | 0,591 | **0,290** |
| F1 | **0,362** | 0,759 | 0,581 | **0,303** |

**Table 5.16:** XLM-R Baseline (same as reported in section 5.2)

| Top 3 Cat. Acc. | | 0,754 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,792 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,613 | **0,820** | **0,666** | 0,298 |
| Precison | 0,284 | **0,745** | 0,579 | 0,204 |
| Recall | 0,264 | **0,802** | **0,630** | 0,166 |
| F1 | 0,254 | **0,770** | **0,593** | 0,162 |

**Table 5.17:** DistilBERT Baseline (same as reported in section 5.2)

| Top 3 Cat. Acc. | | **0,779** | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | **0,826** | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,619 | 0,815 | 0,643 | **0,370** |
| Precison | **0,440** | 0,746 | **0,600** | **0,397** |
| Recall | **0,356** | 0,794 | 0,601 | **0,290** |
| F1 | **0,364** | 0,765 | 0,585 | **0,305** |

**Table 5.18:** XLM-R: Removing all samples that do not have a translation

| Top 3 Cat. Acc. | | 0,762 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,799 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | **0,620** | **0,825** | **0,678** | 0,297 |
| Precison | 0,282 | **0,747** | 0,579 | 0,202 |
| Recall | 0,265 | **0,807** | **0,639** | 0,164 |
| F1 | 0,255 | **0,773** | **0,597** | 0,163 |

The results from letting the DistilBERT model be trained on original data, and letting the XLM-R model be trained on translated data are presented in tables 5.19 and 5.20. The results show that the XLM-R model trained on translated data achieves roughly the same overall accuracy as the XLM-R baseline, but that the accuracy for the top- and middle segment is worse, and the accuracy for the bottom segment is somewhat better. The reverse relationship goes for the DistilBERT model. Comparing the results in Figure 5.19 to the DistilBERT baseline shows that the DistilBERT model trained on original data achieves similar overall accuracy, but has better accuracy on the top segment, and lower on the bottom segment, although in this case the accuracy for the middle segment is exactly the same. To see that the monolingual DistilBERT model performed well when trained and evaluated on original data was an interesting finding, further discussed in Section 6.1.3.

**Table 5.19:** DistilBERT: Trained and evaluated on original data

| Top 3 Cat. Acc. | | 0,769 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,817 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0.611 | 0,823 | 0.643 | 0.330 |
| Precison | 0.404 | 0.730 | 0.596 | 0.353 |
| Recall | 0.320 | 0,805 | 0,591 | 0.247 |
| F1 | 0.328 | 0,762 | 0,577 | 0.260 |

**Table 5.20:** XLM-R: Trained and evaluated on translated data

| Top 3 Cat. Acc. | | 0,760 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,805 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,610 | 0,806 | 0,654 | 0,322 |
| Precison | 0,294 | 0,752 | 0,569 | 0,218 |
| Recall | 0,280 | 0,788 | 0,615 | 0,189 |
| F1 | 0,267 | 0,769 | 0,582 | 0,182 |

## 5.5 Exp-C: Language-Wise Performance

The results from gradually removing amounts of data in a single language are presented for Language X and Language Y in Figure 5.3 and Figure 5.4. There is initially a sharp increase in performance when looking at accuracy, with a declining increase as more data is added. In the Language Y case, there is a higher level of initial performance, a larger initial increase, and a higher top accuracy reached than in the Language X case. Notice by looking at the performance of the other languages, that the average accuracy for non-Language X and non-Language Y in the respective cases is marginally affected. This points to the fact that performance in one language is dependent on the amount of data it receives in that language, and that there seems to be a point of diminishing returns after which the performance gain from new data in a language is reduced.

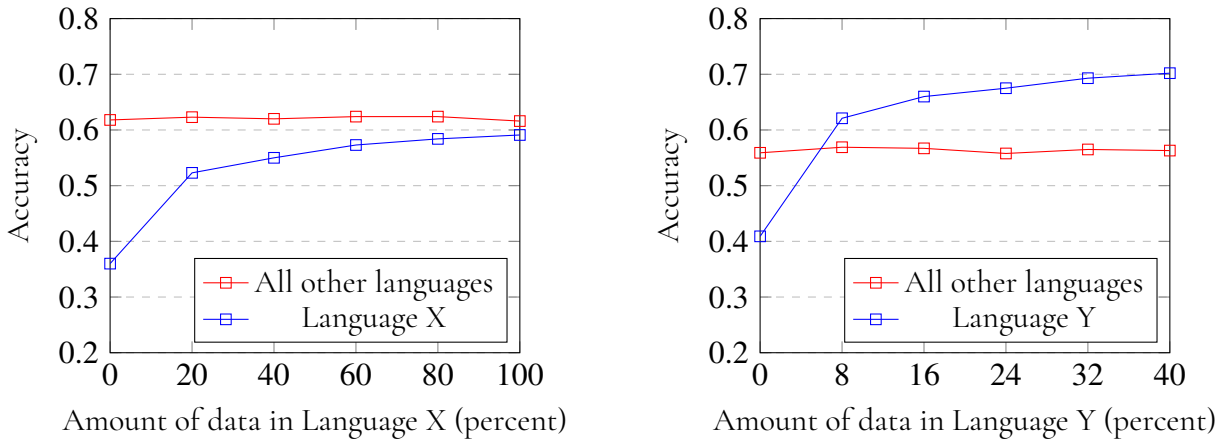



**Figure 5.3:** Accuracy per language plotted against what percentage of the samples in a given language is kept in the training set.

In order to further investigate what factors yield a high accuracy for a given language, the language-wise accuracy was plotted against various characteristics. The results are presented in Figure 5.5 and Figure 5.6, and are from the XLM-R baseline. The individual language understanding varied based on how many samples belonged to a certain language, as well as if the samples belonged to frequent or infrequent classes. Figure 5.5 presents scatter plots for accuracy per language, based on what percentage of the language's samples belong to a class with high support, and what percentage of the language's samples belong to classes with low support. Figure 5.6 also presents accuracy per language, but plotted against the total

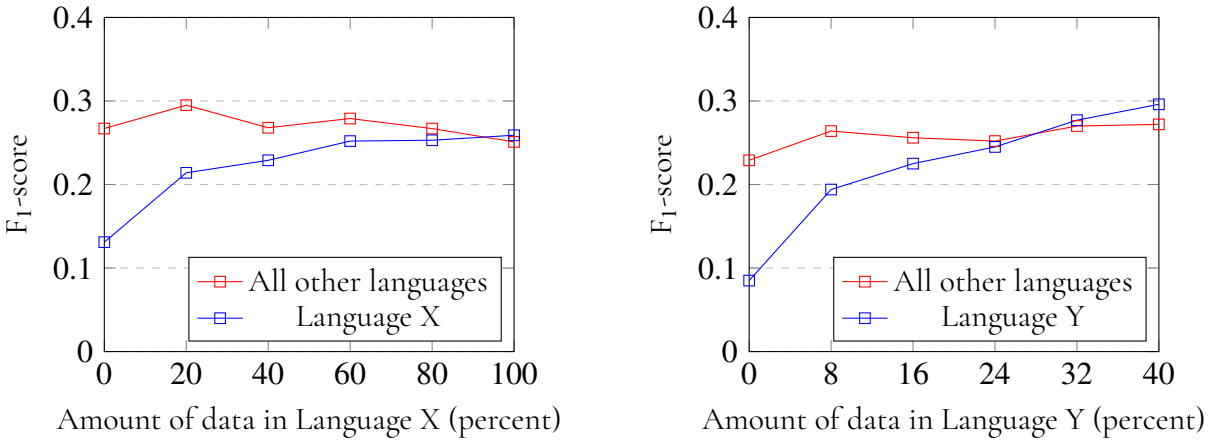



**Figure 5.4:** $F_1$-score per language plotted against what percentage of the samples in a given language is kept in the training set.

number of samples for each language, as well as the total number of unique classes for each language. While there is a large variability, figure 5.5 shows a trend of increased accuracy for languages that have most of their samples belonging to high support classes and vice versa. Figure 5.6 shows that a language being well represented in the data is more likely to have a high accuracy. The figure also shows a slight trend towards languages with more unique classes being predicted better. In all these cases it is noted that there is high variability and low $R^2$-values, due to which no conclusions can be drawn with certainty.

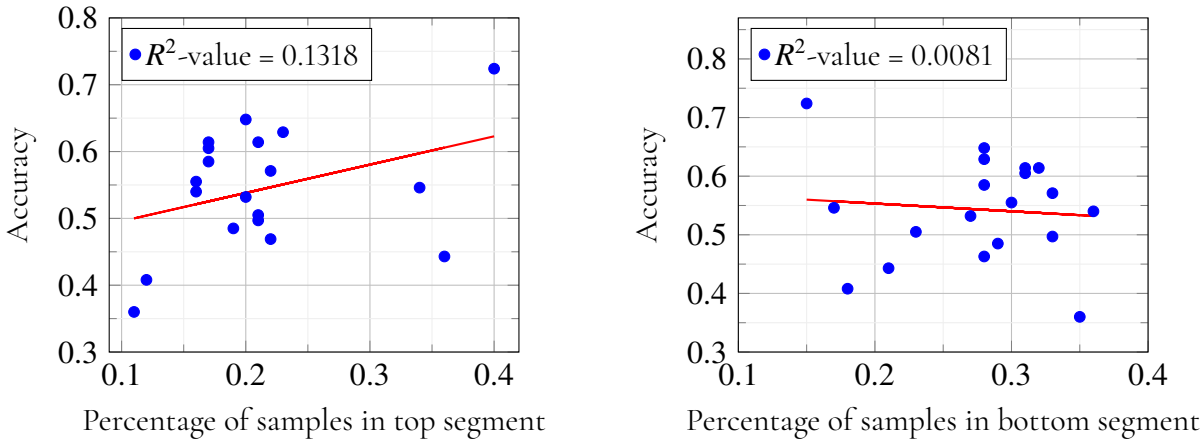



**Figure 5.5:** Accuracy per language plotted against what percentage of the samples are in top or bottom segments

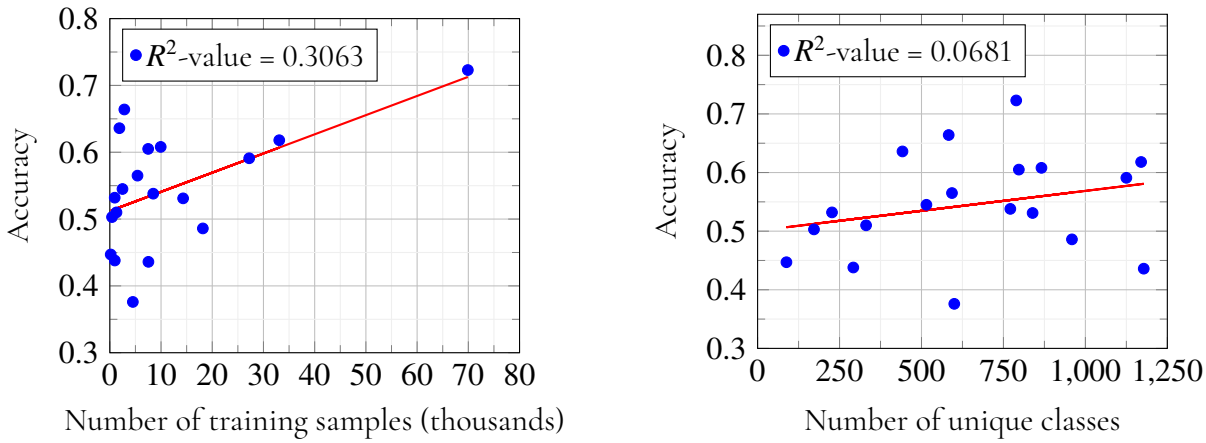



**Figure 5.6:** Accuracy per language plotted against number of samples (left), and number of unique classes (right)

## 5.6 Exp-D: Effect of Oversampling

The results from experiments D1–D4 (4.4.4) show that simple oversampling techniques can lift the performance of the bottom segment and increase the models' ability to correctly label under-represented classes. Tables 5.21 – 5.24 show the results of the different oversampling approaches for each segment. The largest improvements are shown when oversampling with the highest threshold of 50, and there is a clear trend of improved performance in the bottom segment as the sampling threshold is increased. Compared to the baseline XLM-R in section 5.2, experiment Exp-D4 shows an increase in overall accuracy of nearly 2 percentage points, an increase in precision and recall of 15 and 8 percentage points respectively, and an increase in overall $F_1$-score of 10 percentage points. Top 3- and 5 categorical accuracy are increased by 3 and 4 percentage points each.

**Table 5.21:** Oversampling with a threshold of 10

| Top 3 Cat. Acc. | | 0,773 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,816 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,625 | **0,824** | 0,666 | 0,344 |
| Precison | 0,344 | 0,744 | 0,596 | 0,275 |
| Recall | 0,304 | 0,805 | 0,626 | 0,216 |
| F1 | 0,300 | 0,771 | 0,600 | 0,218 |

**Table 5.22:** Oversampling with a threshold of 20

| Top 3 Cat. Acc. | | 0,774 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,819 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,626 | 0,819 | 0,663 | 0,356 |
| Precison | 0,347 | 0,746 | 0,595 | 0,279 |
| Recall | 0,314 | 0,802 | 0,627 | 0,229 |
| F1 | 0,309 | 0,769 | 0,602 | 0,230 |

**Table 5.23:** Oversampling with a threshold of 30

| Top 3 Cat. Acc. | | 0,775 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,819 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,627 | 0,816 | 0,660 | **0,369** |
| Precison | 0,390 | 0,756 | 0,596 | 0,336 |
| Recall | 0,336 | 0,797 | 0,624 | 0,260 |
| F1 | 0,337 | 0,774 | 0,602 | 0,267 |

**Table 5.24:** Oversampling with a threshold of 50

| Top 3 Cat. Acc. | | **0,784** | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | **0,830** | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | **0,632** | **0,824** | **0,668** | 0,366 |
| Precison | **0,420** | **0,757** | **0,598** | **0,371** |
| Recall | **0,350** | **0,806** | **0,632** | **0,275** |
| F1 | **0,353** | **0,778** | **0,606** | **0,285** |

To visualize how the model performs varied based on support for each class, figures 5.7, 5.8 and 5.9 show the $F_1$-score for each class plotted against its frequency in the training data. They show the results for the baseline XLM-R model, and the oversampled XLM-R model with a threshold of 30 and 50 respectively. Each point in the scatter plots represents an individual class, and the dashed red lines show the separation between the bottom, middle and top segments. There is a clear cut-off in figure 5.7 below a support of roughly 40 samples, where essentially no classes are correctly predicted. After oversampling this region is more densely populated by classes with higher $F_1$-values. Comparing a sampling threshold of 30 and 50 respectively, note that there are more classes with non-zero $F_1$-values in the latter, shown in figure 5.9. In the case of the baseline XLM-R model, there are 661 classes that are never correctly identified in the test set. The same number is 443 classes when oversampling with a threshold of 30, and 375 classes when oversampling with a threshold of 50.


**Figure 5.7:** $F_1$-score per class based on class support for baseline XLM-R model.

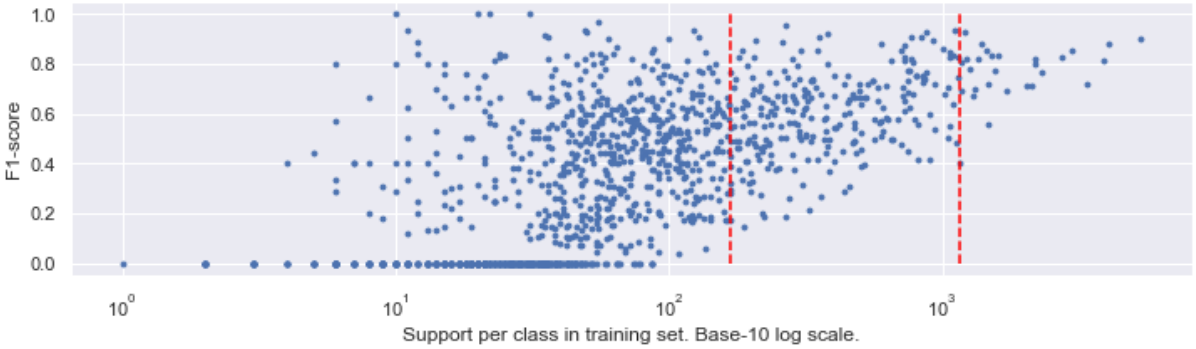



**Figure 5.8:** $F_1$-score per class based on class support for XLM-R model on oversampled data with a sampling threshold of 30.

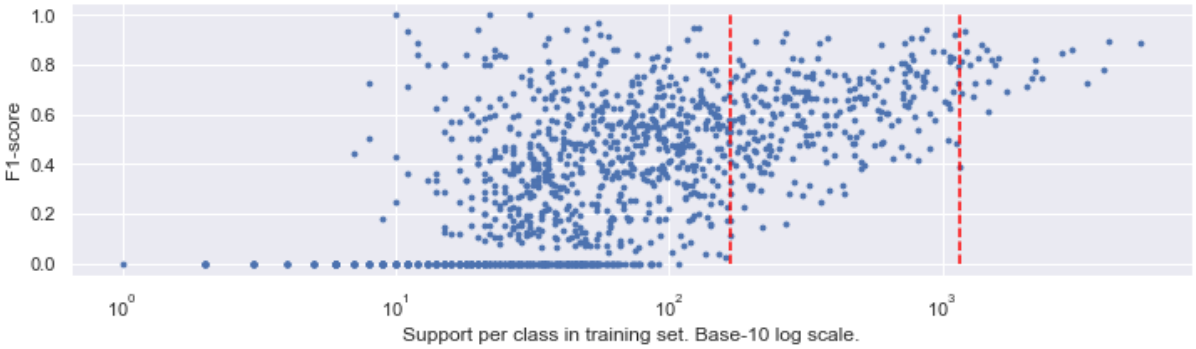



**Figure 5.9:** $F_1$-score per class based on class support for XLM-R model on oversampled data with a sampling threshold of 50.

## 5.7 Exp-E: Augmentation Through Unidirectional Translation

The results from experiment Exp-E1 (4.4.5) are shown in Figure 5.10 and Figure 5.11. The leftmost figure shows the results when starting with 10% and 20% original English data, and the rightmost figure when starting with 80% and 100%. Figure 5.10 shows no clear trends, as the accuracy and $F_1$ sometimes increase when more translated samples are added, and sometimes decrease. This could be due to the inherent variation between runs, as further discussed in section 6.3.3. Due to this, it is hard to draw any conclusions from these results, but what can be said is that adding translated samples seems to have a greater effect (positive or negative) if the original amount of English is small. Comparing the two graphs, the accuracy for the starting levels of 10% and 20% varies with just over 2 percentage points. For the starting levels of 80% and 100% (being augmented with more than three times the data) the accuracy varies with less than 1 percentage point.

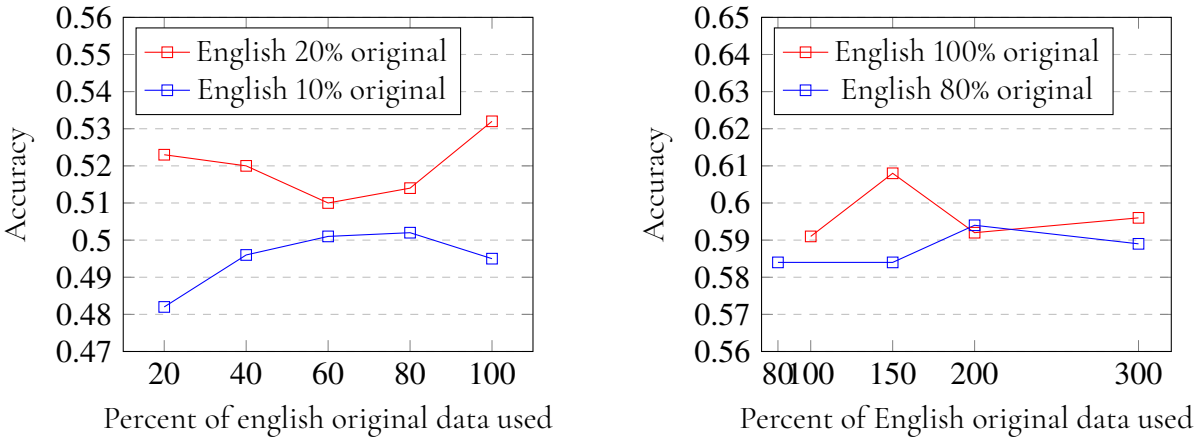



**Figure 5.10:** Accuracy on English data when augmenting with synthetic data. Starting levels at 10, 20, 80 and 100%.

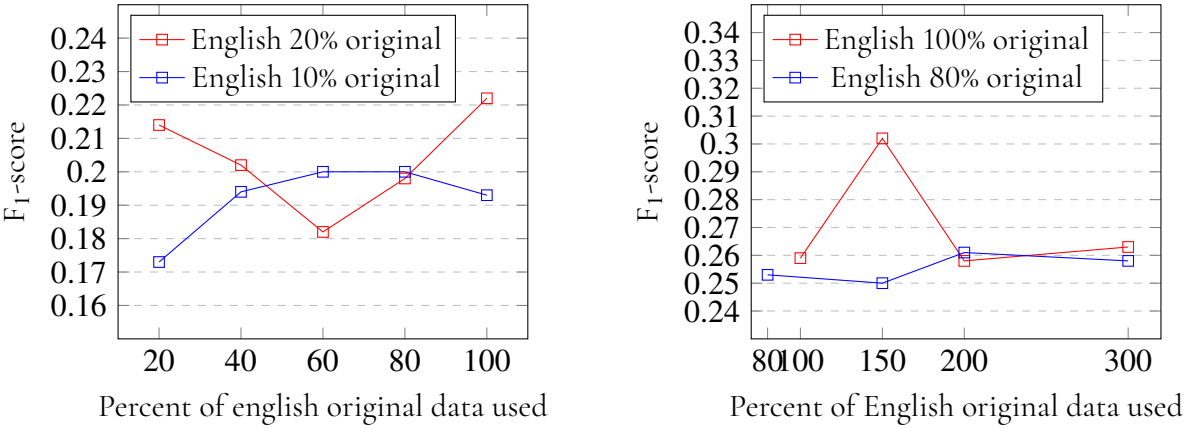



**Figure 5.11:** $F_1$-score on English data when augmenting with synthetic data. Starting levels at 10, 20, 80 and 100%.

**Table 5.25:** Augmenting with a sampling threshold of 50

| Top 3 Cat. Acc. | | 0,782 | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | 0,828 | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | 0,627 | 0,818 | 0,665 | 0,357 |
| Precison | 0,383 | **0,759** | 0,594 | 0,325 |
| Recall | 0,332 | 0,801 | 0,628 | 0,251 |
| F1 | 0,331 | **0,778** | 0,600 | 0,258 |

**Table 5.26:** Oversampling with a threshold of 50 (same as Table 5.24, for comparison)

| Top 3 Cat. Acc. | | **0,784** | | |
|---|---|---|---|---|
| Top 5 Cat. Acc. | | **0,830** | | |
| Segment | **Total** | **Top** | **Mid** | **Bottom** |
| Accuracy | **0,632** | **0,824** | **0,668** | 0,366 |
| Precison | **0,420** | **0,757** | **0,598** | **0,371** |
| Recall | **0,350** | **0,806** | **0,632** | **0,275** |
| F1 | **0,353** | **0,778** | **0,606** | **0,285** |

The results for augmenting data with translated samples for underrepresented classes are shown in Figure 5.25. Table 5.26 from section 5.6 that shows the same experiment but when oversampling with original English samples is also shown to allow for an easier comparison. Table 5.25 shows that the accuracy for the top- and middle segment is very similar if compared to the XLM-R baseline, but that augmenting with translated samples for underrepresented classes in fact does increase accuracy for the bottom segment by almost 6%. Comparing Table 5.25 to Figure 5.26 shows that oversampling using original English samples yields better results for almost all metrics calculated.

## 5.8 Inference and Training Time

The inference time of the models with respect to the different configurations when predicting on a CPU is shown in Figure 5.12 and Figure 5.13. The inference time for the smaller DistilBERT model is as can be seen significantly shorter than that of XLM-R.

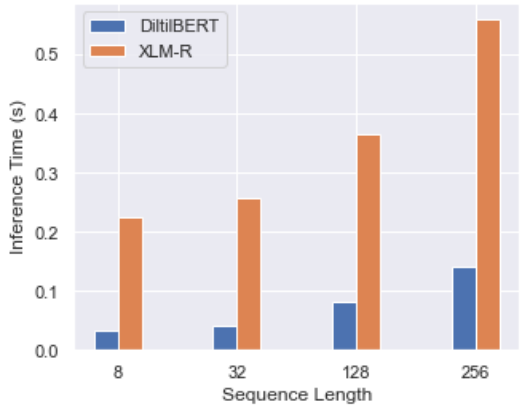



**Figure 5.12:** Inference time for different sequence lengths with a batch size of 1

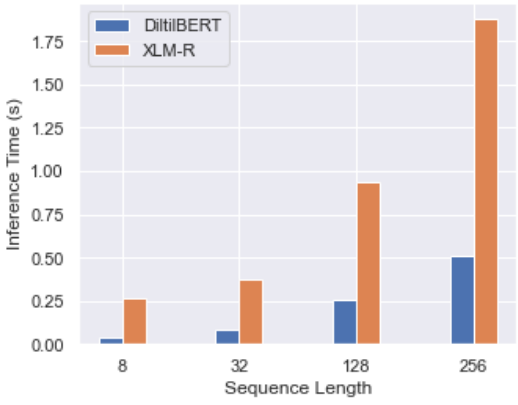



**Figure 5.13:** Inference time for different sequence lengths with a batch size of 4

When predicting using a GPU the inference time drops by a factor of 10 for both models, as shown in Figure 5.14 and Figure 5.15.

The training time for each model varied both between the models due to differences in parameter count, learning rate, and batch size, but also in-between runs for each model due to the nature of random initialization and the varied amount of training data. The average

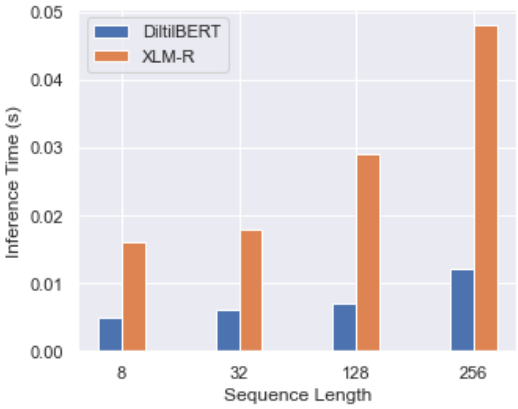



**Figure 5.14:** Inference time for different sequence lengths with a batch size of 1 on a Nvidia T4 GPU

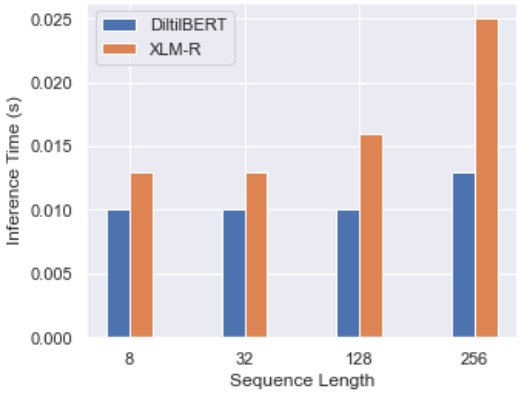



**Figure 5.15:** Inference time for different sequence lengths with a batch size of 1 on a Nvidia Tesla P100 GPU

training times are presented for the models in table 5.27, for models trained on a Nvidia Tesla P100-SXM2 GPU. Not only is the training time per epoch longer for the larger XLM-R, but the average number of epochs per training run is almost the double that of DistilBERT.

**Table 5.27:** Training time for DistilBERT and XLM-R.

| Model | Average no. of Epochs | Average Training Time | Average Time per Epoch |
|---|---|---|---|
| DistilBERT | 6.60 | 5h 45min | 52min |
| XLM-R | 12.75 | 26h 36min | 2h 5min |

# Chapter 6
# Discussion

In this chapter we will discuss the experiments conducted throughout this theses, the results yielded from these experiments, and what conclusions can be drawn. Furthermore, this chapter will discuss the use-case and implications for the company, and limitations for the way this thesis was performed. Lastly, we will present the conclusions and future work – possible areas to explore that we think could complement this thesis and lead to interesting research.

## 6.1 Experiments

In this section we discuss and interpret the results from the experimental phase, to bring nuance and contrast them to related research in the field.

### 6.1.1 Baselines

The results produced by a shallow model with FastText word embeddings showed to be drastically lower than those from deep learning models. While it is rather unsurprising given a problem of this scale, it shows that deep learning is a viable and promising area to explore for this context. The performance of the CNNs are significantly better than the FastText model, and are surprisingly equal between the monolingual and multilingual domain. This was unexpected due to the belief that training on the multilingual data would require a larger vocabulary for the tokenizer. It could however be the case that the size is already well beyond what is needed in the monolingual case and could be reduced without a loss in performance.

Given a classification problem with over 1,300 classes, the baselines for the deep learning models are impressive. The top 1-, 3-, and 5 accuracy levels of over 61%, 75% and 79% for DistilBERT and XLM-R are quite striking, considering the identified performance degradation

rate of nearly 1% per added class of Liu et al. [47]. To be noted is that the authors just mentioned achieved both accuracy and $F_1$-scores of 80%, whereas the baselines in this report show a significant difference between the two (accuracy and $F_1$). The large difference between top 1 accuracy and top 3 accuracy of 14 and 16 percentage points for XLM-R and DistilBERT respectively (see Figure 5.5) indicate that there might be potential to fine-tune the model even further as there are many cases where the model comes just short of selecting the right label. The results (found in table 5.4 and 5.5 show that the XLM-R model performs better on the top- and middle segments of classes, but significantly worse on the bottom segment, resulting in the XLM-R having a lower overall accuracy than the DistilBERT model. Why XLM-R has a tendency to perform worse on the bottom segment is unclear, but throughout the project there seems to be a connection between increased performance when training on the translated data, rather than using the original data.

The finding that DistilBERT can deliver results on par with those of larger models such as RoBERTa (and XLM-R) goes in line with the previous work of Shaheen et al. [48], but is not consistent with the findings presented in the original paper of Liu et al. regarding RoBERTa [29], on which XLM-R is based. As mentioned in section 5.2, one should have in mind that during these baseline experiments, the two models have been trained and evaluated on different amounts of data, since the samples without translations were inapplicable when training the monolingual DistilBERT model. On one hand, this can be thought to help the XLM-R model perform better, as it has in fact been trained on more data than the DistilBERT model. However, the "additional" data is solely samples that have unknown predicted language, meaning that these samples in general are of bad quality, with very limited amounts of interpretable text.

At first sight, it might seem strange that the XLM-R model has significantly lower overall precision, recall and $F_1$-score whilst having an accuracy that is almost the same as the DistilBERT model. This can be explained by the fact that overall accuracy is calculated for all samples, whilst (macro) precision, recall and $F_1$-score is calculated as an average over all classes. The bottom segment of classes consists of 25% of the data, and is hence affecting the overall accuracy the same amount as the top segment. However, due to the way the segmentation of classes was done, the bottom segments consists of over 1,000 of the 1,355 classes, meaning that a model's performance on the bottom segments has a very substantial effect on the macro-weighted overall precision, recall and $F_1$-score.

Regarding the confusion matrices in Table 5.7 and Table 5.8 it is noteworthy that the two classes (out of the top 10 classes) that got confused to the highest extent was class 5 and 16, and class 140 and 159. For both cases, the classes getting confused map to similar components of a truck, which are probably located close to each other and have similar functionality. Due to this, it is not hard to see how the symptom descriptions can be similar for these components, which is probably why the models confuse them to a higher extent than other classes.

### 6.1.2 Exp-A: Effect of Cleaning Data

When first designing the experimental setup to investigate whether different methods of cleaning the data could affect model performance, the hypothesis was that the results would

differ quite significantly between the different experiments. However, looking at the results it is evident that the different cleaning methods did not affect model performance significantly at all, and that performing none of the cleaning methods actually produced the best results for many of the evaluation metrics studied.

The only processing step that actually can be said to improve the model's predictive performance regarding some evaluation metrics was the removal of standard phrases. Why this is the case is not immediately clear, but there are some possible drawbacks with the other methods. For example, the removal of duplicates and removal of long and short samples, entire samples were removed. Even though the information removed was thought to be confusing to the model and thus lowering its performance, these cleaning techniques reduced the amount of training data.

Regarding the removal of all non-alphanumeric characters from the data, the processing step should in theory reduce the number of unique tokens in the data, and was thought to make predictions easier since all instances of characters not from the English language were removed. However, since this processing step removes certain letters from words whilst keeping the rest, it is possible that the changes in structure confuse the model, or that important context is lost. However, removing standard phrases does not really have a clear drawback. Since only the standard phrase is removed from a sample, no valuable information should be lost. Also, since entire phrases are removed, no words are left in an altered and unstructured way. It is also the cleaning method that is the most understandable with regards to how the data is actually altered.

## Different ways of cleaning data

The different ways of cleaning the data could naturally have been performed in many different ways, and some parameters could have been chosen with more concern. Regarding the removal of long and short samples, the thresholds of 4 and 67 could have been chosen differently to remove more or less data, possibly based on a more thorough inspection of the data to determine at what lengths the samples were deemed less informative. It could also have been done in a more sophisticated way, perhaps through finding a way to measure the informational content of a sample.

When removing all non-alphanumeric characters from the samples the characters themselves were removed, but other characters in the affected word were left untouched. An approach that might have proven to be more effective is to remove the different diacritics from the texts, instead of removing the entire character. That would mean that every instance of for example the characters é, ê, ç, ĕ and ë would simply be replaced by an e. However, the results from the experiment go somewhat in line with other research showing that BERT-based models are not improved by the removal of non-alphanumeric characters [61].

The identification of what standard phrases were present in the data was done solely based on manual inspection of the data. This approach is rather primitive, and could have been performed in a more sophisticated way. One interesting approach could have been to calculate what word-orders of a certain length that was most frequent in the data. Starting at n = 2, taking out all word-orders of length two and sorting them based on frequency would probably have led to more standard phrases being identified. It would then also be possible

to increase n, thus being able to identify longer phrases that are common.

When removing duplicate samples, the hypothesis was that many of the duplicates would confuse the model, especially those with identical input texts but with mappings to different classes. However, the results again show that this processing step actually makes the model's predictions worse. Regarding the samples with identical input text and class, it was hard to tell if they originated from the same workshop order but for some reason had been added to the dataset multiple times, or if they actually referred to different workshop orders. In hindsight, it would have been interesting to divide this experiment into two parts, one where all identical duplicates were removed, and one where only duplicates mapping to different classes were removed. This would allow for more nuanced insights into how these two types of duplicates affect the model. It is also probable that the original distribution will naturally include duplicates, which might be a reason to not eliminate duplicated samples in the future.

### Variation between runs

When looking at figure 5.1, it is evident that the accuracy varied between the four runs for the different segments. One insight from this figure is that the accuracy for all data (overall) does not vary between runs much at all, but that the accuracy for the different segments varies somewhat, especially for the top- and bottom segments. The reason for this is thought to be that the more samples the accuracy is calculated for, the more stable it should be between runs. With this reasoning it is not surprising to see that the variability between runs is the largest when calculated for the top- and bottom segments, as these two segments contain 25% of the data each. The middle segment consists of 50% of the data and overall accuracy is calculated for 100% of the data, which should according to this reasoning make the variability between runs smaller, as is the case.

## 6.1.3 Exp-B: Multilingualism in the dataset

When comparing the DistilBERT model and the XLM-R model in a context where the models were trained on comparable data, shown in Table 5.16 to Table 5.17 no large differences were shown in the results. The overall trend of the XLM-R model performing better on the top- and middle segments but worse on the bottom segment is yet again apparent in both cases. Due to the results being so similar, and the overall accuracy between the two models differing only 0.1 percentage point for both cases it is hard to get indications whether this classification problem is best tackled using a monolingual model on translated data or a multilingual model on original data. It is however interesting to see that both models perform better in almost every metric when trained and evaluated without the samples that do not have a translation. For the DistilBERT model the overall accuracy rises from 61,4% to 61,9%, and for the XLM-R model it rises from 61,3% to 62,0%. This result could imply that the samples without translations (all belonging to the unknown-category) in fact are of such bad quality that their presence in the data worsens the models' predictive performance. It should be noted, however, that the variability of the results between training runs means the differences can not only be attributed to the difference in data.

Table 5.20 shows that an XLM-R model also performs well when being trained on the

translated, monolingual data. This is rather unsurprising, as the XLM-R model is pre-trained on large amounts of English data. Compared to the XLM-R baseline, the results show that the model trained on translated data gets slightly better results for the bottom segment, but slightly worse for the top- and middle segments. It is also noteworthy that the XLM-R model trained on translated data performs almost as well as the DistilBERT baseline. This goes in line with the previous work by Rust et. al [62], showing only very small performance decreases for languages that are well represented in the multilingual model's vocabulary when compared to their monolingual counterpart. It is however important to recall that the DistilBERT model is of smaller size and has fewer parameters than XLM-R, and it is possible that these results would differ more should the two models be of more comparable sizes.

Perhaps the most surprising finding throughout this thesis was the realization that the smaller and monolingual DistilBERT model performs equally well on the original data as it did on the translated data, as shown in figure 5.19. Being pre-trained only in English, the expected results was that its performance on multilingual data would be far below that of the performance on monolingual, English texts, as the vocabulary is several times larger and contains words that intuitively would seem impossible to break down into sub-word tokens which carry any meaning given the pre-trained word embeddings in DistilBERT. Why this is the case is hard to tell.

One possible explanation could be that the specific task, in this case text classification, is a task that in a way does not require substantial amounts of general language understanding. It could be the case that classifying short text samples only requires the models to learn mappings between specific tokens and classes, and thus not really needing a thorough understanding for the language.

If the task would have been for instance multilingual question answering, where a model is expected to generate an answer sentence based on a question input in a given language, it might be required to actually understand the language and its grammar better. In such a setting, it is possible that a model that has been trained in a monolingual context would underperform immensely. However, further research is needed to understand why the DistilBERT model performs nearly as well as the XLM-R model on this specific multilingual dataset.

### 6.1.4 Exp-C: Language-Wise Performance

Just like in the dataset explored in this project, the distribution of languages on which XLM-R is pre-trained is uneven, meaning the model is exposed to some languages to a much larger extent than others. A consequence of this is that certain languages (those being over-represented during pre-training and among the most frequent languages during fine-tuning) are languages which the model was believed to perform well on, even if it is artificially reduced to an under-represented language. The question to study in Exp-C, however, was first if there was a relative increase in accuracy for a language, in relation to other languages and overall performance when augmenting with data in only the respective language.

The results in figure 5.3 show that an increased amount of data in Language X and Language Y results in increased accuracy for the respective language. It also shows that the accuracy levels for the non-Language X and non-Language Y languages stay somewhat constant, indicating that an increase of data in a certain language mainly seems to affect the perfor-

mance on that particular language. There is a reduction in the incremental increase in accuracy as more data is added, and the biggest increase in accuracy is gained when going from 0% data to 20% of the total amount of Language X data. This implies that there is a point of diminishing returns, after which adding more data in a certain language has a lesser effect.

We identified two possible explanations for the isolated increase in accuracy for a language from increasing the amount of data. First, it can be due to the model getting an increased understanding for that particular language, and the domain-specific technical terms in this language. Secondly, the observed effect can originate in an increase in the number of samples for the classes that are represented in the language. After investigating this possibility further, it turns out that there are only five classes that are unique to Language X and six classes unique to the Language Y language in the data. Thus, all other classes in Language X and Language Y are also to be found in other languages. The claim that the accuracy for Language X and Language Y increase due to an increasing understanding for the languages in isolation, rather than an increased amount of data for the classes represented in the languages cannot be said to be confirmed with certainty. However, the fact that the number of unique classes for both languages is very low, that the class distribution is equal between the languages (see Fig 3.3), and that the accuracy of other languages remains somewhat constant imply that this is the case. This also goes in line with the research of Singh et al. [53], who show that models like Multilingual BERT do not embed different languages into a shared space, indicating that the representations learned in one language do not necessarily transfer to other languages.

As seen in figure 5.5, there is a trend showing that an increased percentage of classes in the top segment is associated with higher accuracy for a language, and that an increased percentage of samples in the bottom segment is associated with a decreased accuracy. This is rather unsurprising, as it is clear from previous results that all models perform better on the top segment classes and worse on the low segments. What is interesting in figure 5.5 are mainly the "outliers", languages that do not follow the overall trend very well. Looking at the leftmost plot, there is one language with a very high percentage (36%) of its samples in the top segment of classes, but which is still only predicted with 44% accuracy. This is the unknown-category, and this result once again supports the assumption that the samples in this category are of lower quality than for the other languages, and thus harder for the model to predict.

Figure 5.6 also provides some interesting insights. Rather unsurprisingly, the figure shows that the accuracy for a language tends to increase when more data in that language is added. The rightmost plot however shows something interesting. A hypothesis before performing these experiments was that the more unique classes that were represented in a language, the harder the classification task would be, and hence the accuracy for that particular language would be lower. Figure 5.6 shows that this is not the case for the experiments done in this thesis. Although a somewhat weak trend, the relationship between the number of unique classes and accuracy seems to be the opposite. This could be explained by the fact that a language with many unique classes is also likely to be well represented in the data, as illustrated in figure 3.4. This figure shows that a language with a high number of unique classes is also likely to be well represented in the data, with Language Y being the only major exception.

This could be an explanation for why there is a trend showing that languages with many classes tend to be predicted better. For all scatter plots in figure 5.5 and figure 5.6 it should be noted that the data shows large variability, which is why it is hard to draw any certain conclusions from the figures, and the trend lines are simply for illustrative purposes.

### 6.1.5 Exp-D: Oversampling

While it is clear that oversampling led to improvements among underrepresented classes, it is important to study whether it comes on the expense of performance for the middle- or top segments. It can be argued that such a trade-off is not beneficial in a real use-case where it is crucial to have high predictive performance on the most frequent classes, and where the desired result is to increase the performance of the underrepresented classes without negatively affecting the performance in the other segments. However, there is no significant decrease in performance of the middle segment as shown in Figure 5.24. In fact, the overall performance of the oversampled XLM-R model (with a threshold of 50) is better than that of the baseline model. This indicates that balancing the dataset through duplication of samples aids the model in training to not overfit to the most frequent classes, and is a viable method to mitigate the class imbalance issue.

In experiment Exp-D, it was decided to only focus on oversampling of infrequent classes. The reason for not undersampling the most frequent classes was to not reduce the amount of data for the top segment, since it is a key goal to maintain high performance on frequent classes. Undersampling can be a more viable alternative when the cost of misclassification for an underrepresented class is much higher than that of overrepresented classes. Examples of such situations are in fraud detection or medical diagnostics of rare diseases, where it is more critical to correctly identify the low-frequency classes, even if it comes at a cost of reduced precision for the high-frequency classes. This was understood as not being the case in this context, and was therefore not prioritized.

Note the increase in top 3- and top 5 categorical accuracy, when comparing the results from oversampling (78.4% and 83.0%) to the baseline XLM-R model (75,4% and 79.2%). It was hypothesized that while oversampling minority classes might not result in perfect predictions for those classes, it should increase the model's predicted probability for them. It was therefore expected to see an increase in the mentioned metrics due to the correct class more often being in the top $k$ predictions. Comparing table 5.24 to the baselines in tables 5.4 and 5.5 it can be seen that the top 3- and 5 categorical accuracy for XLM-R improves by a few percentage points due to the oversampling. While these findings support the hypothesis, it is noteworthy that it is only marginally better than the results achieved with DistilBERT.

### 6.1.6 Exp-E: Augmentation Through Unidirectional Translation

The results in section 5.7 show only minor changes in accuracy when the English data is augmented with synthetic, translated samples. Although some improvements were made, the results imply that using translated samples perhaps is not the best way to increase performance for underrepresented languages in this context. The results are also somewhat hard to interpret since the accuracy sometimes increases as more data is added, and sometimes

decreases. This is thought to be due to two factors. First, since the differences in results are so small, it is possible that the inherent randomness between runs, as further discussed in Section 6.3.3, affects the results more than the actual adding of translated data. Secondly, it is possible that the data quality differs for the translated samples added for the different configurations. Since these samples are randomly sampled from the training data, it is possible that some configurations contained more ambiguous data, or more data in underrepresented classes. When using translated samples to augment the data for infrequent classes in Exp-E2, the results show a greater effect, as the accuracy for the bottom segment is 6 percentage points higher than the XLM-R baseline. However, comparing to the the case where the data was oversampled by original English samples shows that oversampling increases predictive performance even more.

Looking at the results from Exp-E2, the results show that the oversampling with original English samples yields better performance for almost all of the calculated metrics compared to augmentation with synthetic samples. One explanation to this result is that the translated samples used for augmentation are already present in the training data but in their original language. Therefore the translated samples are essentially duplicates, just in a different language. Perhaps the model's connections between words in different languages are so strong that adding the translated samples does not provide any new information. This is however not supported by the finding from Section 6.1.4 that the model seems to improve in different languages in isolation. Another explanation could be the quality of the translation. By translating the samples, it is possible that information is lost. As shown by the work of Vanmassenhove et. al [63], machine translation often fails to render the same lexical diversity and richness as in the original texts. Perhaps this causes the problem descriptions to lose specificity, thus rendering a translation that does not describe the problem as accurately as the original description. This again, however, goes against the findings that models generally perform well on the translated data.

## 6.2 Text Data and Future Use Case

In this section we discuss the importance of data quality in a future use-case, and other critical aspects that will be of importance when deciding on the use of transformer-based language models.

### Text data

As described in chapter 3, the inherent flaws that were identified in the data point to the fact that improved selection and filtering can be applied in advance, to not end up with datasets containing empty points with no information and to reduce the number of obscure and confusing samples of low quality. Data quality is an important factor in the creation of automated systems and can be a decisive factor of the final performance of machine learning models.

The ISO-25000 series of standards, *Systems and Software Quality Requirements and Evaluation*, has the goal of creating a framework for the evaluation of software product quality [64]. Specifically, The *Data Quality Model* defined in ISO-25012 establishes the main data quality

characteristics that must be taken into account when assessing the properties of a data product. It outlines how requirements on data can be formulated through 15 characteristics. Some of the characteristics are referred to as *inherent*. Inherent data quality refers to the "degree to which quality characteristics of data have the intrinsic potential to satisfy stated and implied needs when data is used under specified conditions" [65].

Given the results from the data analysis, we believe that there is a limit to model performance given the actual datasets, and that increased quality could allow for larger improvements in predictive performance than those achieved in this thesis. We also believe it can lead to higher potential quality of a final product. The areas where we see data quality could be improved are the following areas, described as per the ISO-25012 Data Quality Model.

- **Accuracy**: The degree to which data has attributes that correctly represent the true value of the intended attribute of a concept in a specific context of use.

- **Completeness**: The degree to which subject data associated with an entity has values for all expected attributes and related entity instances in a specific context of use.

- **Consistency**: The degree to which data has attributes that are free from contradiction and are coherent with other data in a specific context of use. It can be either or both among data regarding one entity and across similar data for comparable entities.

- **Credibility**: The degree to which data has attributes that are regarded as true and believable by users in a specific context of use. It includes the concept of authenticity such as the truthfulness of origins.

The data used in this thesis project contains many instances which do not go in line with these characteristics – such as empty or incoherent text descriptions not related to the context (for example long conversations), with lengths varying from a single word to several hundred words and identical samples mapping not only to different classes but also to different main groups. Increased "accuracy" in this context would mean that a sample actually contains an informative text description, instead of only a sequence of numbers or seemingly non-informative notes.

This is not only important to think about in the context of this trial case, but even more important when looking at the intended future use-case, to maximize the value of an application in practice. Using better filtering and curated datasets can probably yield more high-quality training data. If so, curating a dataset might achieve higher performance in a training context, but it is also important to consider the fact that the flaws in the dataset might be more representative of the true distribution of the data which the model will see in a real use-case. To increase the potential performance and benefit of a system like this in practice it therefore seems important to implement measures which reduce the risk of people entering text descriptions of low quality, not in line with the characteristics mentioned above.

Practices such as setting a minimum limit of words or nudging a user through the use of hints, prompts and descriptive questions in the application could lead to more consistency and relevance, especially if the text is generated by a person with less technical expertise and vocabulary. There are automatic frameworks for data validations that can aid in this process, such as *Great Expectations* [66].

## Future use-case for the company

The data used in this thesis is generated after troubleshooting of the truck has been done. However, it does also include the symptoms that made the customer aware of something being malfunctioning in the truck, so it is a combination of descriptions of symptoms and post-diagnostic results. In a future use-case, the data will most likely only be descriptions of symptoms, as one of the applications of these predictive models is to give mechanics an idea of what the faulting part might be before the truck is even analyzed. Since the text descriptions in the future use-case most likely not will be written by technicians at the company, but by people with less technical know-how, the jargon and language might differ. These factors would most likely lead to a change in the distribution, causing the models explored in this thesis to underperform. They do however, show promising results that indicate that the technology could perform well in other domains, and in a real-life use-case.

The potential benefits of using pre-trained language models in troubleshooting show themselves in several promising ways. For the workshops connected to the company, they could provide improved workflows with efficiency gains related to more specific and accurate estimations of the time requirements for the troubleshooting process in each individual vehicle case. Having an early prediction as to what could be wrong, gives an understanding for how long a troubleshooting might take. It can then be decided to schedule troubleshooting for a shorter period, say two hours rather than eight hours, just to be safe. It could also allow for more effective inventory planning and guarantee spare parts availability to a higher extent, thereby reducing potential delays in service. Both of these scenarios would directly translate into gained customer value and help guarantee uptime of customers' vehicles, in turn providing long-term positive effects for the company. More generally, this research area could allow for a new troubleshooting process, transforming diagnostics overall to focusing on symptoms, rather than being heavily focused on error codes and manual inspection.

The results shown in Figure 5.12 to Figure 5.15 highlight the implications of using large scale models in production. In a real use-case, latency would be of great concern and this in turn would decide what machines are needed for deployment. It is not easy to determine what magnitude of latency could be deemed as "acceptable" for these models in production, but if a repair needs to be scheduled when having direct contact with the customer, it is of importance that the models can give an output without delay to the service administrator. As described in *Usability Engineering*, a response time of 0.1 seconds is about the limit for having the user feel that the system is reacting instantaneously, which naturally is desirable for an application of this nature. Given that the total latency for an application using these models in a real use-case will be the sum of both the inference time of the model and the latency of the connection to the server on which the model is hosted, it would be desirable to have the model inference time under 50 milliseconds. When looking at using a batch size of 1, meaning that the model predicts one sample individually (which is likely to be the case in a real use-case), Figure 5.12 shows that running the models on a CPU is probably not a viable option. Although it is unlikely that models in a live use-case would need to have a maximum sequence length of 256 or even 128, Figure 5.12 shows that the XLM-R model is unable to be run on a CPU with satisfying inference times. Therefore the model would probably need to be hosted on a GPU. In Figure 5.14 and Figure 5.15 it is shown that the more powerful GPU Tesla P100 is able to perform predictions fast for both the DistilBERT model and the XLM-R

model, but that the less powerful Tesla T4 also is sufficient to provide inference times under 50ms for both models. Due to this, it is plausible to believe that this GPU (or another model of roughly the same capacity) would be chosen in a live use-case.

There are naturally costs associated with hosting these models in a real use-case. If hosted in a cloud environment such as Google Cloud or Amazon Web Services, costs would be associated with accessing a GPU based on what time the company wants to maintain this access. To get more concrete, yearly access to one Tesla T4 GPU would cost the company around $1300 yearly on Google Cloud. If for some reason the more powerful P100 GPU is deemed more appropriate for hosting the models, this number would increase to $6300*. Naturally, a GPU on one cloud environment would not be sufficient given the company´s global presence, and many environments hosted on servers in different geographic locations would probably be necessary. However, one GPU could be used to process the data from one geographic region, given that the intensity of model invokes does not exceed the inference time.

The training time of the models will affect how they can be used in production, as they will need to be re-trained at a certain interval, and perhaps updated with regards to changes in the manufacturing of vehicles. If a change or problem in the production line leads to an increased risk of a certain fault, this will propagate and change the distribution of faults over time. Having models which can adapt to these changes will require updates and training of models with new data. With XLM-R taking almost five times longer to train when compared to DistilBERT, it is easy to see how this might affect the pace of development.

The differences between the models also affect their respective usefulness when studying the future use-case in production. Not only are time requirements for training of importance, but during deployment more lightweight models are of course also preferable. When trying to analyze the benefits of using multilingual models in the context of reduced need for translation, there are two main areas involved. One is the financial aspect that relates to the cost of using automated translation services. The second aspect is integrating the translation step in the design of the application. As multilingual models are only trained on a certain set of languages, there are potentially markets where the company is present whose language is not included in the pre-training of some multilingual models, resulting in a continued need for translation services. The financial aspect related to costs of translations is considered to be the most important factor in this case. As the discussion regarding potential costs is based on estimates, the number of invocations needed are not necessarily reflecting what would be a true use-case scenario. To truly understand the benefit of multilingual models that reduce the need for translation services, a thorough analysis of the expected use is of importance to get an accurate estimate of the invocation intensity. Regarding latency, it is also hard to draw a sharp line regarding what is acceptable or not without knowing the true requirements. These are all factors that need to be considered when choosing whether to use a mono- or multilingual model in the future use-case.

*Numbers gathered from www.cloud.google.com at the time of writing this master's thesis.

## 6.3 Limitations

This section reports a critical discussion of our work including the most important threats to the validity of our conclusions.

### 6.3.1 Model Architecture Choice

When it comes to constructing machine learning models, deciding the architecture can sometimes seem more of an art than a science. In this thesis pre-trained language models have been acquired from the public library provided by *HuggingFace*, which limits the flexibility and possibility to customize models. In the future, adapting a model to the specific use-case could be of benefit. In the case of using pre-trained language models like the ones used in this thesis project, fine-tuning for a downstream task can be done in several ways, and there is a large room for customization. The model has to be adapted to the task at hand, which can lead to a great variety in model architectures. In the case of implementing and fine-tuning a pre-trained language model for a downstream classification task, the following are two possible approaches:

- Adding custom classification layer(s) on top of the base model, with the base model being trainable.
- Adding custom classification layer(s) on top of the base model, with the base model being non-trainable (frozen).

There are different motivations for using these different options. During pre-training, the models learn language understanding in its core layers. One risk connected to allowing for the base layers to be trained is that it can "disrupt" or interfere with the pre-training, essentially leading to the model "forgetting" its original understanding. For this reason, one can instead choose to freeze the base layers, meaning they receive no updates during training, hence staying intact. Instead an additional neural network can be added to the model, which takes as input the numerical word embeddings produced by the base layers and learns the classification task at hand. By using small learning rates, the risk of the model forgetting previous knowledge can be mitigated and allow for fine-tuning of the whole model. The reason for doing this is that it can adapt the full model to learn representations in a way more adapted to the target task at hand. The two above mentioned methods can also be combined, through the process of *gradual unfreezing*, mentioned in Section 2.7, where the parameters of different layers are gradually made trainable.

In this thesis, the chosen method was to allow for the training of the parameters in the original pre-trained base layers, as early experiments with only training custom classification heads proved to be less effective and often resulted in low-quality results. Using the pre-trained models available in the *HuggingFace*-library designed for sequence classification provided models which are readily available to use for text classification tasks, which also made implementation easy.

#### Comparability between models

Given that all models in these experiments are different to some extent, they are not always easily comparable. In the case of differences in the amounts of original and translated data,

the mitigation in this report has been to train both mono- and multilingual models on sets where missing translations are substituted by their original text, and also where the samples missing a translated text are dropped, as discussed in section 6.1.3. There are also significant differences in the sizes of the DistilBERT and XLM-R models, which impact the capacity for the models to adapt to the problem. However, from the experiments conducted in this thesis, this increased capacity does not necessarily lead to significant increases in predictive performance. With the expectation that XLM-R would be a generally more high-performing model with a larger capacity for learning and generalization, and given that the automatic translation step was expected to cloud or distort the data to some extent, it was quite unexpected to see DistilBERT outperforming XLM-R on the translated data. What could then be the reason for this? As DistilBERT is only pre-trained using English data, it has a language domain much smaller than that of XLM-R. The same is true for the vocabulary of tokens for the two models. It could then be speculated that moving from the multilingual domain to the monolingual domain simplifies the problem, given that the automatic translation service is of high quality and can thereby accurately translate the data. While XLM-R has a significantly larger capacity, it is also trained on over a hundred languages, which to some extent have to "share" this capacity. This should, however, not be a problem according to the creators of XLM-R, who claim to achieve state-of-the-art results comparable to those of RoBERTa in the monolingual case, and that the multilingual capacities do not sacrifice per-language performance [30].

### 6.3.2 Interpretability of results

The scale of the problem at hand makes fully understanding the results complicated. The large number of classes, multiple languages, and the size of the models make it hard to fully evaluate experiments in such a black-box situation as in this thesis project, as there is correlation between several important aspects. For example, the effect on both class-wise and language-wise performance from increased amounts of data will be highly correlated, and it is hard to study one or the other in isolation. As a focus in this thesis was to understand what would affect performance for classes and languages, not being able to isolate individual aspects of the problem makes it hard to understand what the success factors are in high-performing models. It also becomes difficult to answer questions such as why a monolingual model can perform just as well on multilingual data as a model with larger capacity, trained on multiple languages.

It is not easy to understand fully how the language understanding of models such as DistilBERT and XLM-R works. As mentioned by Danilevsky et al. [67], the advances in the quality of state-of-the-art models in NLP have come at the expense of interpretability. In the paper they identify five major explainability techniques, one being *feature importance*, where the main idea is to derive explanation by investigating the importance of different features used to output the final prediction. Such features can be lexical features in the text or latent features like the word embeddings and attention scores between words learned by the deep models used in this thesis project. While it is possible to extract attention scores, and study embeddings for words and sentences to compare their similarity with respect to different metrics and thereby draw conclusions as to why a model performs well on certain languages or classes, this kind of analysis is complicated and labor-intensive. Many levels of abstraction

make the details in the results less clear and interpretable. This makes it hard to understand what kind of characteristics would constitute a "high" or "low" quality sample, although we have tried to do so in section 6.2 by looking at the lexical features of samples. Danilevsky et al. refer to *local* explanations as those providing information or justification for a model's prediction on a specific input. In contrast, a *global* explanation reveals how the model's predictive process functions in general, not connected to a specific input [67].

In a context with over 1,300 classes, finding local explanations and justifications become very difficult. The choice in this thesis project to segment classes based on support leads to a simple breakdown which is easy to look at, but it is not possible to understand the effect of changes in performance metrics on a class-by-class level. In classification tasks of smaller scale, the use of confusion matrices and other tools can provide a detailed understanding of classifier behavior. It can give insights regarding between what classes, or types of classes, prediction errors are made. This can bring clarity to the question of what classes are prone to be misclassified in favor of another, and allow for further research as to why that is. As the full-scale confusion matrices in this project are of dimensions above $1{,}300 \times 1{,}300$ it was decided to only present reduced versions for the most frequent classes. However, a possible approach to help visualize could have been *confusion matrix ordering* [68].

### 6.3.3 Error Analysis

A potential source of error in this project is that working with translated data relies on an external machine translation service. This leads to little insights as to how the texts are translated, and what it might entail. While manual inspection can be used to verify translations to an extent, it is not certain that the translation service can perform well in a technical domain. As shown in the data analysis, there are several mistakes identified with regard to the language of a certain sample. It is likely that the translation service is better at performing translations in languages which it has seen more of earlier, which might be a problem if such a translation would be used to generate synthetic samples in an underrepresented language.

Another source of error in this thesis is the variation in results identified between different runs. Although two models trained on the same data with identical hyperparameters generally yielded very similar results, there were instances where the results between runs differed with around 3% for some prediction metrics. As described by Dodge et. al. in *Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping* [23], distinct random seeds can affect the model's performance, leading to substantially different results. According to Dodge et. al, the two factors influenced by the inherent randomness are the weight initialization of the model and the data training order, the latter being in what order the model processes the samples during training. It was with this in mind it was chosen to run the experiments for Exp-A (Effect of Cleaning Data) four times per configuration and average the results. Unfortunately, as reported in section 5.8, the training of XLM-R models was more time-consuming than that of DistilBERT models. Due to this it was deemed unfeasible to perform multiple runs for the experiments involving XLM-R models as it would have exceeded our time budget, even though it would have added validity to the results presented in this thesis.

There are other sources of errors in the work presented in this thesis that also arise from

inherent randomness. When creating datasets for all experiment configurations not run on the standard training set, the sampling from the original training set was done randomly. For instance, creating the datasets used in Exp-C: Language Wise Performance in section 5.5, the sampling from the total amount of Language Y and Language X data was done independently for each configuration. This means that the samples used when training on 40% of the Language X data could very well be completely different from when training on 20% of the data. The same goes for Exp-D: Effect of oversampling and Exp-E: Augmentation through unidirectional translation. This somewhat complicates the comparison of results between experiments, as some experimental configurations might have had more samples in the bottom segment of classes, or more samples of low quality than others. This was done in an attempt to not introduce external influence on the sampling, and to keep the process completely random, but is something that might affect the results presented in this thesis.

# Chapter 7
# Conclusion

In this thesis project, we have explored the use of pre-trained, transformer-based language models in the context of large-scale multilingual text classification for troubleshooting management. We have seen that models such as DistilBERT and XLM-R reach high levels of predictive performance considering the large number of classes, but that larger models do not necessarily yield better accuracy. While there seems to be an upper limit of accuracy believed to be caused by inherent traits in the data, there might be much potential left to explore in the use of models such as XLM-R, especially in the areas of domain-specific pre-training and improved fine-tuning (*RQ1*).

Increasing amounts of data in a given language lead to higher levels of accuracy for that language, but little of this knowledge seems to be transferred to other languages. We see a sharp early increase in accuracy for the first few thousand added samples in a language, and continuing but decreasing improvements after this point. As the experiments in this project show little potential in using translated texts to augment performance in underrepresented languages, it seems crucial to obtain more data in each respective original language. Simple techniques to balance and augment the training data, such as oversampling and automatic translation, have been shown to improve model performance, especially for underrepresented classes (*RQ2*).

Increased data quality, business understanding and usability requirements will be critical aspects for the company to consider regarding further implementing multilingual models for text classification, as the longer training- and inference time for the multilingual model can have a large impact in a real use-case scenario. While the training time for the large XLM-R model is long, it is worth noting that fine-tuning the DistilBERT-model to reach high levels of performance is actually faster than training CNN-models and similar non-transformer based models from scratch. This shows the promise of using pre-trained transformer-based models as long as they are not too large. Therefore, leveraging transfer learning through pre-trained language models is seen as beneficial in future research (*RQ3*).

Finally, the monolingual and multilingual domains do not seem to be very different in terms of complexity, as monolingual models perform remarkably well even on the multilin-

gual data, and smaller convolutional neural networks with custom tokenizers perform well in both domains without an increase in size of either vocabulary or model.

## 7.1 Future work

With a problem of this size, there are many ways to go about trying to solve it. It is a complicated context and there are several interesting ideas that have revealed themselves during the writing of this thesis. In this section, we present the identified future research areas which we think would complement the work done in this thesis.

The concept of ensemble models is one that could be of value in the company's context of large-scale multi-class classification. As there is a gradual hierarchy of abstraction when describing the sub-parts of a truck, there is a straightforward and logical way to see how classes form clusters in which more specialized models could possibly be trained on subgroups of the data, perhaps based on main group. A generalized model could be used to predict main group and several "specialist" models could be used to in turn predict the target part within that main group. A trial experiment on this particular subject was in fact conducted during the work of this thesis, where a generalized model was trained to predict main group instead of class. However, the data was found to contain large inconsistencies with regards to main group, where many individual classes were found in multiple main groups. Hence, the data would probably need to be more thoroughly structured for this approach to yield good results. Also, the ensemble model approach would naturally require a large number of models to be trained, and hence also increase both training time and inference time. Therefore one would need to consider the trade-off between time efficiency and accuracy if investigating if ensemble models can be used in a real-life use-case.

The time dimension is not taken into account throughout this thesis, but could carry valuable information regarding what errors are more probable than others. As some malfunctions are more likely to occur at different stages of a trucks lifetime, the distribution of classes would probably differ depending on if the truck is very new or has been in use for several years. Hence, information such as production year and mileage could be used by the models to get a better prior understanding of what classes are more probable than others. As this information was not included in the data used for this thesis it was not possible to test whether this approach could yield increased performance, but it would be interesting to see if such information could be leveraged by the models.

An alternative method to increase the number of training samples is to use unlabelled data. Such data from a similar domain is available to the company, and in the context of this problem it would be possible to leverage the large amount of unlabelled data on which an already fine-tuned model makes predictions. The predictions for which the model is most confident are then considered as true labels, and those samples are thereafter added to the training data on which another model is trained. This could be done iteratively, thus becoming a form of *Knowledge Distillation* as described in section 2.4.2, where the original model can be considered the teacher model that guides the learning of the second model with its predictions. This way it would be possible to add samples to the training data that are completely

new to the model. One could also refine this idea, possibly only adding samples that are predicted to a class in the bottom segment, and whose texts are dissimilar from the samples already in the training data based on some similarity score.

A topic briefly touched upon in section 6.3.1 is further pre-training of transformer-based models in a target domain. As the models are pre-trained on general language, but the target task in this setting has a distinct and technical vocabulary, it could potentially give the models a more domain-specific language understanding. This would require training the model in the same way the original pre-training was conducted, which for the models used in this thesis is Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). It would be possible to perform both *within-task pre-training*, in which the models are further pre-trained on the data from the target task, and *in-domain pre-training*, in which the models are trained on data from the same domain as the target task. Such data could for instance be technical documentation and company-specific lexical resources, such as catalogues for spare parts or troubleshooting manuals. As investigated in the paper *How to Fine-Tune BERT for Text Classification?*[45], within-task pre-training and in-domain pre-training can significantly boost the performance of transformer-based models. Since the workshop orders used to train the models in this thesis often contain technical terms and names of specific sub-parts, it would be interesting to see if further pre-training could improve the models' predictive performance.

# References


[1] Huggingface. `https://huggingface.co/`. Accessed: 2022-01-28.

[2] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[3] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[5] Nhan Cach Dang, María N. Moreno-García, and Fernando De la Prieta. Sentiment analysis based on deep learning: A comparative study. *Electronics*, 9(3), 2020.

[6] James F. Allen. *Natural Language Processing*, page 1218–1222. John Wiley and Sons Ltd., GBR, 2003.

[7] Ayse Pinar Saygin, Ilyas Cicekli, and Varol Akman. Turing test: 50 years later. *Minds and machines*, 10(4):463–518, 2000.

[8] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[9] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[10] F. Chollet. *Deep Learning with Python*. Manning Publications, 2018.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[12] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[14] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10):4291–4308, 2020.

[15] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. Tensor2tensor for neural machine translation. 03 2018.

[16] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In Věra Kůrková, Yannis Manolopoulos, Barbara Hammer, Lazaros Iliadis, and Ilias Maglogiannis, editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 270–279, Cham, 2018. Springer International Publishing.

[17] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[18] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[19] Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[20] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28:3079–3087, 2015.

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

[22] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[23] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*, 2020.

[24] Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*, 2020.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[26] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[28] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[30] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

[31] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Special issue on learning from imbalanced data sets. *ACM SIGKDD explorations newsletter*, 6(1):1–6, 2004.

[32] Qi Dong, Shaogang Gong, and Xiatian Zhu. Imbalanced deep learning by minority class incremental rectification. *IEEE transactions on pattern analysis and machine intelligence*, 41(6):1367–1381, 2018.

[33] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[34] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, pages 429–449, 2002.

[35] Krystyna Napierała, Jerzy Stefanowski, and Szymon Wilk. Learning from imbalanced data in presence of noisy and borderline examples. In Marcin Szczuka, Marzena Kryszkiewicz, Sheela Ramanna, Richard Jensen, and Qinghua Hu, editors, *Rough Sets and Current Trends in Computing*, pages 158–167, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[36] Alberto Fernández, Salvador Garcia, Francisco Herrera, and Nitesh V Chawla. Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, 61:863–905, 2018.

[37] Vicente García, José Salvador Sánchez, and Ramón Alberto Mollineda. An empirical study of the behavior of classifiers on imbalanced and overlapped data sets. In *CIARP*, 2007.

[38] Khaled Al-Sabbagh, Regina Hebig, and Miroslaw Staron. The effect of class noise on continuous test case selection: A controlled experiment on industrial data. 11 2020.

[39] Dragan Gamberger, Nada Lavrac, and Ciril Groselj. Experiments with noise filtering in a medical domain. 03 2001.

[40] Bohan Li, Yutai Hou, and Wanxiang Che. Data augmentation approaches in natural language processing: A survey. *arXiv preprint arXiv:2110.01852*, 2021.

[41] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[42] V. García, R. A. Mollineda, and J. S. Sánchez. On the k-nn performance in a challenging scenario of imbalance and overlapping. *Pattern Anal. Appl.*, 11(3–4):269–280, aug 2008.

[43] Steven Y Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.

[44] Pei Liu, Xuemin Wang, Chao Xiang, and Weiye Meng. A survey of text data augmentation. In *2020 International Conference on Computer Communication and Network Security (CCNS)*, pages 191–195, 2020.

[45] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[46] Aqsa Noor and Ahmad Ali. Multiclass imbalanced classification of quranic verses using deep learning approach. In *2021 4th International Conference on Computing Information Sciences (ICCIS)*, pages 1–6, 2021.

[47] Xinyi Liu and Artit Wangperawong. Transfer learning robustness in multi-class categorization by fine-tuning pre-trained contextualized language models. *arXiv preprint arXiv:1909.03564*, 2019.

[48] Zein Shaheen, Gerhard Wohlgenannt, and Erwin Filtz. Large scale legal text classification using transformer models. *arXiv preprint arXiv:2010.12871*, 2020.

[49] Shanshan Yu, Jindian Su, and Da Luo. Improving bert-based text classification with auxiliary sentence and domain knowledge. *IEEE Access*, 7:176600–176612, 2019.

[50] Sumanth Prabhu, Moosa Mohamed, and Hemant Misra. Multi-class text classification using bert-based active learning. *arXiv preprint arXiv:2104.14289*, 2021.

[51] Markus Borg, Iben Lennerstad, Rasmus Ros, and Elizabeth Bjarnason. On using active learning and self-training when mining performance discussions on stack overflow. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 308–313, 2017.

[52] Henric Zethraeus and Philip Horstmann. Evaluation of active learning strategies for multi-label text classification. Master's thesis, Lund University, Faculty of Engineering, 2021.

[53] Jasdeep Singh, Bryan McCann, Richard Socher, and Caiming Xiong. BERT is not an interlingua and the bias of tokenization. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 47–55, Hong Kong, China, November 2019. Association for Computational Linguistics.

[54] Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1, pages 29–39. Springer-Verlag London, UK, 2000.

[55] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, Berlin, Heidelberg, 2012.

[56] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[57] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

[58] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[59] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: A comprehensive review. *ACM Computing Surveys (CSUR)*, 54(3):1–40, 2021.

[60] Wei Wang and Jianxun Gang. Application of convolutional neural network in natural language processing. In *2018 International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pages 64–70, 2018.

[61] Deepak Kumar, Nalin Kumar, and Subhankar Mishra. NLP@NISER: Classification of COVID19 tweets containing symptoms. In *Proceedings of the Sixth Social Media Mining for Health (#SMM4H) Workshop and Shared Task*, pages 102–104, Mexico City, Mexico, June 2021. Association for Computational Linguistics.

[62] Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How good is your tokenizer? on the monolingual performance of multilingual language models. *arXiv preprint arXiv:2012.15613*, 2020.

[63] Eva Vanmassenhove, Dimitar Shterionov, and Andy Way. Lost in translation: Loss and decay of linguistic richness in machine translation. *arXiv preprint arXiv:1906.12068*, 2019.

[64] The iso/iec 25000 series of standards. `https://iso25000.com/index.php/en/iso-25000-standards`. Accessed: 2022-01-27.

[65] Iso/iec 25012. https://iso25000.com/index.php/en/iso-25000-standards/iso-25012. Accessed: 2022-01-27.

[66] Great expectations. https://greatexpectations.io/. Accessed: 2022-01-28.

[67] Marina Danilevsky, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen. A survey of the state of explainable ai for natural language processing. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 447–459, 2020.

[68] Martin Thoma. Analysis and optimization of convolutional neural network architectures. Masters's thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, June 2017.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-02-11

**EXAMENSARBETE** Multilingual Large Scale Text Classification for Automotive Troubleshooting Management
**STUDENT** Jacob Curman, Alv Romell
**HANDLEDARE** Markus Borg (LTH), Olof Steinert (Scania)
**EXAMINATOR** Pierre Nugues (LTH)

# Transformer-modeller för storskalig textklassificering

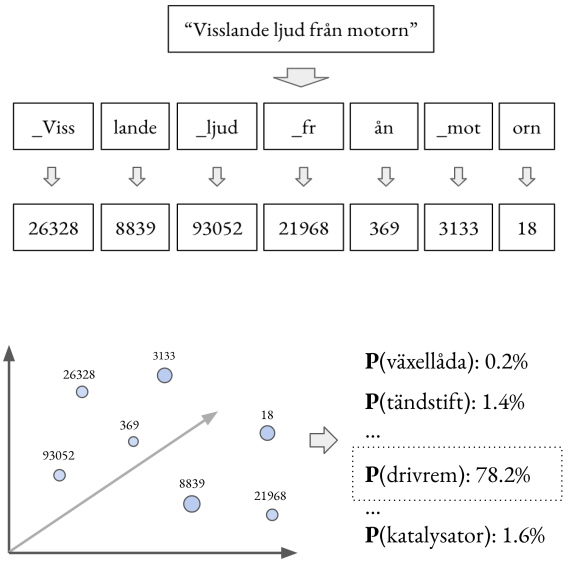
POPULÄRVETENSKAPLIG SAMMANFATTNING **Jacob Curman, Alv Romell**

Maskininlärningsmodeller baserade på så kallad transformer-arkitektur har visat sig mycket användbara när det kommer till att lära datorer förstå språk. Detta arbete handlar om att använda dessa modeller för att klassificera felbeskrivningar av problem i lastbilar.

Naturlig språkbehandling, eller *natural language processing*, handlar om att tolka och förstå det mänskliga språket. Inom detta område finns förutsättningar för att lära datorer och maskiner tolka och förstå språkdata och text, ofta i form av matematiska koncept likt vektorer i flerdimensionella rum. Detta innebär att *modeller* får lära sig representationer av ord genom att studera stora mängder text.

Storskaliga språkmodeller som för-tränats av aktörer likt Google och Facebook har visat sig vara av stor nytta i många maskininlärningsapplikationer gällande språkförståelse. Dessa modeller har tränats på data i en generell domän för att uppnå en allmän språkförståelse, men kan även finjusteras genom ytterligare träning för att bättre anpassas till specifika uppgifter och användningsområden. Modeller med *transformer*-akitektur utnyttjar ett koncept kallat *attention* (uppmärksamhet) för att förstå kontext och samband mellan ord i en mening. Dessa finns i olika varianter, där vissa blivit tränade till språkförståelse på ett specifikt språk (enkelspråkiga) och andra tränats till att uppnå förståelse på ett stort antal språk (flerspråkiga).

I detta examensarbete har två modeller med transformer-arkitektur genom vidareträning finjusterats till uppgiften att klassificera felbeskrivningar av problem i lastbilar. Datan som använts kommer från en svensk lastbilstillverkare, där stora mängder textdata produceras vid service- och garantiärenden. Företaget vill undersöka möjligheten att använda språkmodeller för att förutsäga vilken komponent i lastbilen som orsakat felet redan innan lastbilen nått en verkstad, då detta skulle kunna möjliggöra en mer ef-



**EXAMENSARBETE** Multilingual Large Scale Text Classification for Automotive Troubleshooting Management

**STUDENT** Jacob Curman, Alv Romell
**HANDLEDARE** Markus Borg (LTH), Olof Steinert (Scania)
**EXAMINATOR** Pierre Nugues (LTH)

fektiv schemaläggning av reparationer och lagerhållning av reservdelar. På grund av företagets globala närvaro är flera olika språk representerade i datan, där även en engelsk översättning finns för varje textbeskrivning. Som följd har en enkelspråkig och en flerspråkig modell använts.

Med dessa modeller har en rad experiment genomförts, med särskilt fokus på tolkningsbarheten kring hur dessa modeller uppnår språkförståelse, och olika möjligheter att öka modellernas prediktionsförmåga gällande textbeskrivningar som beskriver ovanliga fel, och på ovanligt förkommande språk. Rapporten undersöker även vilka faktorer som kommer vara viktiga i ett potentiellt framtida användningsfall av dessa modeller i en industri-kontext.

Resultaten visar att storskaliga transformerbaserade modeller når en hög prediktionsförmåga i den undersökta klassificeringsuppgiften, trots att det finns många olika unika fel-klasser, men att större modeller inte nödvändigtvis ger bättre resultat. En ökad mängd data på ett visst språk visas resultera i en ökad prediktionsförmåga för textbeskrivningar på just detta språk, medan prestandan för andra språk är relativt oförändrad. Dessutom är olika tillvägagångssätt för att göra datan mer balanserad med avsende på klasser och språk effektiva för att öka modellernas predktionsförmåga, framför allt för fel-klasser som är ovanliga.