Automatic Detection and Classification of Suicide-related content in English texts

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Hubert Baginski, BSc
Matrikelnummer 01551118

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Allan Hanbury
Mitwirkung: Prof. Dr. David Garcia
Dr. Hannah Metzler
Assoc. Prof. Dr. Thomas Niederkrotenthaler, PhD

Wien, 10. März 2021

Hubert Baginski	Allan Hanbury

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Automatic Detection and Classification of Suicide-related Content in English Texts

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Hubert Baginski, BSc
Registration Number 01551118

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Allan Hanbury
Assistance: Prof. Dr. David Garcia
Dr. Hannah Metzler
Assoc. Prof. Dr. Thomas Niederkrotenthaler, PhD

Vienna, 10th March, 2021

Hubert Baginski	Allan Hanbury

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Hubert Baginski, BSc
Wagramer Straße 23/3/5.1, 1220 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10. März 2021

Hubert Baginski

Acknowledgements

I would first like to thank David and Allan for giving me the chance to be part of this extremely fascinating and challenging research project, which we eventually extended to this Master thesis.

Special thanks to Thomas who provided invaluable insights and freely shared his expertise, time, and research with me. Most of my thanks goes to Hannah. She guided the research and always took the time out of her busy schedule to listen to my ideas and offer advice whenever I needed it most. She also volunteered and edited this whole thing!

Finally, I want to thank my parents who still have no idea what I studied but supported and encouraged me anyways. Yes, I do stuff with computers . . .

Thank you.

Kurzfassung

Die Medienberichterstattung über Suizide hat wiederholt gezeigt, dass sie mit Suizidraten assoziiert ist. Die Auswirkungen der Suizidberichterstattung sind möglicherweise nicht nur auf schädliche Effekte beschränkt; Geschichten über die Bewältigung schwieriger Umstände haben schützende Effekte. Insbesondere ist die Aussetzung von Medienberichten über Todesfälle mit einem Anstieg der Suizide verbunden, was auf einen Werther-Effekt hindeutet. Die Untersuchung der Auswirkungen der Suizidberichterstattung erfordert die Klassifizierung verschiedener Merkmale von Medienartikeln, die schädliche oder negative Auswirkungen haben können, was sich als zeitintensiv und schwierig erweist. Die Verwendung von natürlicher Sprachverarbeitung für die Klassifizierung solcher Texte könnte diese umständliche Aufgabe erleichtern. Wir verwenden das bidirektionale Sprachmodell BERT und vergleichen seine Leistung mit TFIDF und Bag-of-Words.

Wir zeigen, dass Deep Learning und synthetische Datengenerierung die Entwicklung einer Anwendung ermöglichen, die in der Lage ist, englische Texte zu verarbeiten und spezifische Merkmale von suizidbezogenen Inhalten zu erkennen. Wir beschreiben ein effektives Klassifikationsmodell, das es dem Benutzer ermöglicht, das vorhergesagte Label eines bestimmten Variablencodes für den gegebenen englischen Eingabetext abzurufen.

Einfache binäre Klassifikationsaufgaben werden am besten durch ein fein abgestimmtes BERT-Modell gelöst und erreichen $85\% - 95\% F_1$, verglichen mit der menschlichen Leistung von $F_{1human} = 100\%$. Mittlere binäre Klassifizierungsaufgaben profitieren oft von einer synthetischen Balancierung mit Leistungen um $75\% - 80\% F_1$ ($F_{1human} \sim 80\%$). Schwierige binäre Klassifizierungs- und Mehrklassen-Klassifizierungsaufgaben profitieren immer von einer synthetischen Balancierung. Welche Balancierungsmethode am besten funktioniert, ist jedoch aufgabenspezifisch, und die Leistungen liegen jeweils zwischen $F_1 \sim 70\%$ ($F_{1human} \sim 80\%$) und $\sim 80\%$ ($F_{1human} \sim 95\%$).

Unsere Ergebnisse zeigen, dass bidirektionale Sprachmodelle unglaublich gut funktionieren. Verbesserungen scheinen jedoch hauptsächlich von größeren Modellen und mehr Daten zu kommen. Das synthetische Balancieren der Minderheitenklassen liefert mehr Trainingsdaten und verbessert die Fähigkeit des Modells, auf neue Texte zu generalisieren. Unsere Anwendung wird es den Forschern ermöglichen, die Auswirkungen verschiedener Merkmale von Texten über Suizid in großem Maßstab zu untersuchen und dabei helfen, die Richtlinien für die Berichterstattung zu verbessern und so effektiv zur Prävention von Selbstmorden beizutragen.

Abstract

Media reporting on suicide has repeatedly been shown to be associated with suicide rates. The impact of suicide reporting may not be restricted to harmful effects; rather, stories of coping and recovery in adverse circumstances may have protective effects. Specifically, exposure to media reports about deaths is associated with increases in suicides, suggesting a Werther effect. In contrast, exposure to content describing stories of hope and coping are associated with a decrease in suicides, which has been labeled as the Papageno effect. Investigating the impacts of suicide reporting requires classifying various characteristics of media-items that may have harmful or negative effects, which proves time-intensive and challenging. Using natural language processing for the classification of such texts could facilitate this tedious task. We use the bidirectional language model BERT and compare its performance against TFIDF and Bag-of-words.

We show that deep learning and synthetic data generation allow developing an application, which is capable of processing English texts and detecting specific characteristics of suicide-related content. We describe an effective classification model that enables the user to retrieve the predicted label of a specific variable code for the given English input text.

Simple binary classification tasks are best solved by a fine-tuned BERT model trained on the original data, achieving $85\% - 95\% F_1$, compared to human performance of $F_{1human} = 100\%$. Intermediate binary classification tasks often benefit from synthetically balancing the data, with performances around $75\% - 80\% F_1$ ($F_{1human} \sim 80\%$). Difficult binary classification and multi-class classification tasks always benefit from synthetically balancing the data. However, which balancing method works best is task specific, and performances range between $F_1 \sim 70\%$ ($F_{1human} \sim 80\%$) and $\sim 80\%$ ($F_{1human} \sim 95\%$), respectively.

Our results show that pre-trained bidirectional language models work incredibly well. Yet, improvements seem to mostly come from bigger models and more data. Synthetically balancing the minority classes provides more training data and improves the model's ability to generalize to new inputs. However, limiting the amount of synthetic data is crucial, since performance appears to tail off when the balance is tipped too far in favour of the synthetic data.

Our application will enable researchers to investigate the effect of different characteristics of texts about suicide at large scales and help improve reporting guidelines, thereby effectively contributing to the prevention of suicides.

Contents

CHAPTER 1 Introduction

1.1 Motivation

Suicide is a major public health problem worldwide: Every year close to 800'000 people take their own life, and a lot more attempt suicide[1]. A prior suicide attempt is the single most important risk factor for suicide in the general population. Furthermore, suicide is the third leading cause of death in 15-19-year-olds worldwide and the second leading cause of death in Austria and the USA among individuals between the ages of 10 and 34[2].

Media reporting on suicide has repeatedly been shown to be associated with suicide rates (Pirkis et al., 2006; Niederkrotenthaler et al., 2010, 2012; Sinyor et al., 2018). That media reports about suicide elicit imitative suicidal behavior is referred to as the "Werther effect" (Phillips, 1974). It is likely stronger when the media coverage is extensive, prominent, sensational, explicitly describes the method of suicide, and condones or repeats widely held myths about suicide (Niederkrotenthaler et al., 2020).

The increase of suicides after media reporting on suicide cases might be explained by an increased cognitive availability of suicide as an acceptable way to cope with difficulties (Niederkrotenthaler et al., 2014). Some studies suggest that reporting on suicide might increase suicidal thoughts. Such thoughts, particularly if they evolve around suicide plans, increase the risk of suicidal behavior (Nock et al., 2008).

However, the impact of suicide reporting is not restricted to harmful effects. Stories of hope, recovery, and coping with adverse circumstances or suicidal thoughts may have protective effects (Niederkrotenthaler et al., 2010). The association of stories of coping with a decrease in suicides has been labelled the Papageno effect. That different types of

[1]https://www.who.int/news-room/fact-sheets/detail/suicide
[2]https://www.nimh.nih.gov/health/statistics/suicide.shtml

media content have distinct effects suggests that how the media report about suicide is important. It further underlines the importance of the guidelines for responsible reporting developed by the World Health Organization (World Health Organization, 2014, 2017). Reports about suicides written according to recommended reporting guidelines show strong potential to help prevent suicide.

In addition to reporting of suicide cases or coping stories, there are many other characteristics of suicide-related news media content that could potentially influence suicidal behaviour. For example, if media reports contain information about resources such as the Suicide Prevention Lifeline[3], or stories about successfully dealing with suicidal ideation, they might have a prevention effect by encouraging struggling individuals to seek help.

Investigating the impacts of different types of reports about suicide requires detecting and classifying such characteristics in large quantities of news media content. Doing this manually is time-intensive and challenging. Developing natural language processing (NLP) tools for the automatic detection and classification of such content would strongly increase the efficiency of this tedious task and provide unprecedented availability of large labelled collections of media items. In a next step, scientists could then explore associations of exposure to these types of content at a large scale across geographic regions and time.

The different characteristics are associated with varying levels of methodological challenges. The simplest challenges are binary detection tasks that only depend on the occurrence of specific keywords in the text and just return whether the model has found the content in the text or not. We hypothesize that the characteristics on this first difficulty level are easy to detect, learn, and predict. The next level contains binary classification tasks of intermediate difficulty that most likely depend on a rather diverse set of keywords. These models must learn to detect multiple characteristics that indicate the same positive class. The third level contains classification tasks that require the detection of a narrative or the emotional connotation and meaning of texts. We hypothesize that this detection is likely more difficult, even though it still remains a binary classification task. Finally, the most difficult tasks require detecting several different characteristics, and determining to which extent they are covered in the text compared to other characteristics. For instance, determining the main focus of the media item is especially challenging since suicide-related reporting can discuss many different topics, and multiple topics might occur within a single text. Therefore, the model must learn to detect the primary focus of the text.

We propose an application that provides researchers with an effective method to discover a broad range of suicide-related characteristics that might have beneficial or harmful effects on suicidal behavior in English texts. The findings of this research may eventually be used to improve reporting guidelines for suicide-related news content. Furthermore, this application allows for the automatic unsupervised labeling of massive collections of documents. Machine learning will enable automatic checks of journalistic content in

[3]https://suicidepreventionlifeline.org/

accordance with reporting guidelines and highlight potentially beneficial or harmful text passages prior to publishing.

1.2 Problem Statement

This thesis studies numerous characteristics of media content that could potentially influence suicide rates. They were chosen based on scientific evidence regarding harmful characteristics, as well as best practices in media reporting about suicide. We can divide the methodological challenges associated with the detection of these different characteristics into four difficulty levels:

1. *Simple binary classification*:
   the classification most likely only depends on the presence of a few keywords.
2. *Intermediate binary classification*:
   the classification most likely depends on a rather diverse set of keywords.
3. *Difficult binary classification*:
   the classification requires the detection of a narrative, or the emotional connotation or meaning of a sentence.
4. *Main focus classification with multiple classes*:
   the classification depends on the extent to which a certain topic is discussed in comparison to other topics.

1. focuses on binary classification that relies on specific keywords e.g. if alternatives to suicidal behavior are mentioned, or if it provides information about specific strategies or resources for suicide prevention. All suicide characteristics in this group are binary classification tasks, with two labels: yes (*positive instances*, the content is present in the text) and no (*negative instances*, content is not present). We hypothesize that prediction for the majority of characteristics in this group will be trivial. Example tasks included in this thesis are:

- the text reports on suicidal ideation or behaviour of a celebrity (for a full list with concrete examples that indicate suicide-related characteristics see Appendix A)
- the text reports on a completed suicide

2. focuses on binary classification that most likely depends on a rather diverse set of keywords, because different types of content may qualify an observation as a positive instance. This often applies for meta-categorical items, which ask about whether one specific example for a type of content is mentioned, e.g., does the text mention one of multiple possible prevention actions or warning signs. The class depends on *what* is present, but multiple different contents qualify as a positive instance. Example tasks included in this thesis are:

- the text reports on preventing suicide at the individual level: different actions qualify as yes.
- the text reports on alternatives to suicidal behaviour: many possible alternatives exist.

3. requires the detection of a narrative or the emotional connotation and meaning of a text. The classification depends on *how* a topic is described in a text. Example tasks included in this thesis are:

- the texts reports on a healing story, i.e., the meaning or the emotional content such as coping, hope, or recovery.
- the text enhances a common public myth about suicide (explicitly mentions or implicitly hints at one of 9 defined myths). Enhancing means confirming/mentions without denying, if the myths is mentioned but debunked, this does not qualify.

4. depends on the extent to which a certain topic is discussed within a text in comparison to other topics. Classification tasks on this level, usually contain multiple categories and we seek to determine *how much* each category is discussed. Example tasks included in this thesis are:

- the text discusses whether the topic is described as a problem, solution, both or neither.
- the text describes the the suicide method, multiple may be present in the text.

We will train the machine learning models on a limited amount of training data, which were manually labeled by domain experts. To systematically investigate which characteristics have protective and harmful effects, and which are most important, huge quantities of text data must be labeled to allow analyzing statistically significant associations. Consequently, we will deploy models to label large collections of text data focusing on the most impactful characteristics of suicide reporting. This will enable researchers to draw consistent and reliable conclusions about associations of characteristics of suicide-related content that are as of now undetermined.

The main hypothesis of this thesis is the following:

> Deep neural networks in natural language processing that leverage transfer learning and exploit fine-tuning outperform traditional NLP models in detecting various kinds of suicide related content.

In other words, we argue that in the classification of suicide-related characteristics, transfer learning outperforms supervised learning, with two caveats:

(1) Transfer learning may be less helpful when a sufficient number of training samples are available.

(2) Transfer learning may be less useful if there are severe imbalances in the class frequency distributions. To address this issue, we develop and test different methodologies to balance the data in order to answer the following research question:

> Which impact do our two novel synthetic data methodologies, (1) back translation; (2) synonym replacement, have on BERT's performance, measured by the F_1-score?

1.3 Contributions & Thesis outline

We develop an application that processes English texts and classifies different suicide-related characteristics. Furthermore we propose and test two novel synthetic data generation techniques, back translation and synonym replacement, which are used to balance the class frequency distributions. The rest of this thesis is structured as follows:

In **Chapter 2**, we provide background information that is relevant to understand the contents of this thesis. We review the fundamentals of machine learning and natural language processing, and discuss basics of neural networks.

In **Chapter 3**, we build onto the basics of neural networks and introduce BERT. We analyze the concept of attention, specifically self-attention, and the two learning steps of BERT: (1) pre-training and (2) fine-tuning.

In **Chapter 4**, we present our experimental workflow that we developed over the course of this thesis. Next, we inspect the strengths and limitations of the dataset and dive into the synthetic data generation techniques. Then, we document the experimental setup for both the traditional NLP and BERT models to ensure reproducibility.

In **Chapter 5**, we document the results of the four levels of methodological challenges. First, we present the results of the simple binary classification tasks, followed by intermediate and difficult binary classification tasks. Last, we analyze the multi-class classification tasks. In all experiments we compare the performance of the models on the original data against the performance on the synthetically extended data.

In **Chapter 6**, we present the main contributions of this work and provide a brief discussion about the results, specifically the synthetic data generation techniques, and the potential future work this research enables.

CHAPTER 2

Preliminaries

This chapter provides background knowledge that enables the reader to follow the subsequent chapters. It reviews the fundamentals of machine learning (§2.1) that are essential to the techniques we apply throughout this thesis. It then introduces the reader to natural language processing (§2.2), specifically, by showing an overview of the elementary methods and most common tasks in NLP. It subsequently presents the concept of word vector representations (§2.2.4) which are universally used in state-of-the-art models across the most common natural language processing benchmarks. Finally, we examine a particular type of machine learning models, neural networks (§2.3), which have become ubiquitous in toady's machine learning landscape.

2.1 Machine Learning

In this section, we introduce the reader to machine learning, which studies computer algorithms that can learn from data by building mathematical models from it. We introduce many concepts in this section that will reappear throughout the thesis, either creating the foundation for more advanced concepts such as neural networks (§2.3) or supplying the theory applicable to all proposed models. Most of all, we will regularly return to the topics of generalization in machine learning and performance metrics as we will be trying to create models that generalize to other domains.

In machine learning, the data is usually represented as a vector $\mathbf{x} \in \mathbb{R}^d$ of d *features*, where each feature contains the value of a specific attribute of the data. Therefore, we can represent an entire dataset as a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ that contains n observations, one observation in each row.

Machine learning can take several forms depending on the data and the task. Two common sub-disciplines of machine learning are *i) supervised learning* and *ii) unsupervised learning*.

In *supervised learning*, for every observation $\mathbf{x_i}$, the output is typically a separate *label* y_i, which can be represented as a vector $\mathbf{y}$ for the entire dataset $\mathbf{X}$. The dataset for supervised learning can be represented as $\mathbf{X} \in \mathbb{R}^{n \times d},\ y \in d$. The goal is to correctly label unknown data. Two common categories of supervised machine learning tasks are *classification* and *regression*. In regression, the model tries to predict a continuous variable, e.g., the prediction of household income, temperature, or housing prices. In classification, the label y_i belongs to one element of a fixed set of classes $\mathbf{C}$ of size t, $y_i \in \mathbf{C},\ \mathbf{C} = \{c_1, \ldots, c_t\}$. Classes are often referred to as *categories* or *labels*, those terms are used interchangeably.

Classification can be further split into three subtasks: *binary classification*, *multiclass classification*, and *multilabel classification*. Binary classification exclusively deals with two classes $\mathbf{C} = \{c_1, c_2\}$, while multiclass classification deals with more than two classes $\mathbf{C} = \{c_1, \ldots, c_t\},\ |C| > 2$. Generally, in both binary and multiclass classification, every observation x_i has only one correct label y_i. However, in multilabel classification, every observation x_i may be associated with multiple labels $\mathbf{Y}_i = \{y_1, \ldots, y_t\},\ |\mathbf{Y}_i| \geq 1$.

In *unsupervised learning*, for every observation x_i, the output y_i is not in the dataset $\mathbf{X}$, this can be expressed as $\mathbf{X} \in \mathbb{R}^{n \times d},\ y \notin d$. There is no information on which and how many classes or other structures are present in the dataset. Therefore, unsupervised learning looks for previously undetected patterns in a dataset with minimal human supervision. The goal is to find new structures via clustering, association rules, or learning to find outliers and/or anomalies in the data. It is most often associated with data mining and clustering.

While supervised learning deals with the identification of functional dependencies, the objective of unsupervised learning is to capture statistical dependencies, i.e., the structure of the underlying data distributions. The goal of both methods is robust modeling, i.e., the obtained knowledge is generic and, as far as possible, independent of the particular training set provided (Hansen and Larsen, 1996).

The experiments in this thesis will focus solely on classification, with a strong focus on both binary and multiclass classification. We use supervised machine learning models to generate baseline performances which are compared against the performance of unsupervised deep learning models.

2.1.1 Performance metrics

In supervised machine learning, there are several methods of evaluating the performance of learning algorithms and their associated classifiers. Most quality measures for classification are calculated based on a *confusion matrix*, which represents whether observations in each class were correctly or incorrectly recognized by the classifier (Sokolova et al., 2006).

		Predicted	
		+	-
Actual	+	True Positive (TP)	False Negative (FN)
	-	False Positive (FP)	True Negative (TN)

Table 2.1: The confusion matrix for binary classification

The components of the confusion matrix are the fundamental population quantities for binary classification. Table 2.1 presents a confusion matrix for binary classification, with two classes $\mathbf{C} = \{+, -\}$.

The evaluation of a classifier's performance plays a critical role in the construction and selection of the classification model. Even though many performance metrics have been proposed in the machine learning community, there are no general guidelines available regarding which metric ought to be selected to evaluate the classifier's performance for any specific use case (Liu et al., 2014). The correct selection of performance metrics is one of the most crucial issues in evaluating the performance of classifiers. It depends heavily on the application and the consistency of the expected output.

Definition 2.1.1 (Accuracy)**.**
The *accuracy* (Acc) of a machine learning classification algorithm measures how often the algorithm classifies an observations correctly. Accuracy is the number of correctly predicted observations divided by all observations:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy has been the primary metric for assessing classifier performance and has been used across many applications and domains Rosenfield and Fitzpatrick-Lins (1986), Foody (2002), Schlesinger et al. (2007), and García et al. (2009).

However, evaluating a classifier solely on its accuracy does not always provide a reliable and robust estimate of the model's performance (Sturm, 2013). Some applications suffer from severe class imbalances where one class, the *majority class*, is overrepresented.

Definition 2.1.2 (Majority class)**.**
The class in an imbalanced classification predictive modeling problem that is the most prevalent in the class frequency distribution.

$$c_1 = c_{majority} \iff P(c_1|x_i) > 0.50,\ \mathbf{C} = \{c_1, c_2\},\ x_i \in \mathbf{X}$$

Many *information retrieval* (IR) tasks, e.g., patent classification (Fall et al., 2003) and document classification (Sebastiani, 2002), optimize for *precision* to maximize the confidence in predicting the relevant class. Severe class imbalances, i.e., a small number of relevant documents juxtaposed to the large number of all documents, might lead to a classifier with very high accuracy, which, however, would always predict the nonrelevant documents and never the relevant one. A model that is optimized for precision will predict the nonrelevant documents with very high confidence but miss many relevant ones and predict them as nonrelevant. Such a model, regardless of high accuracy, would be worthless for the detection of the relevant class.

Two important performance metrics that weaken the majority class bias are:

Definition 2.1.3 (Precision)**.**
Precision is the fraction of correctly predicted instances out of all retrieved instances.

$$Precision = \frac{TP}{TP + FP}$$

Definition 2.1.4 (Recall)**.**
Recall is the fraction of correctly predicted instances out of all instances of the category.

$$Recall = \frac{TP}{TP + FN}$$

The combination of precision and recall in addition to the classification accuracy is widely applied in information retrieval (Baeza-Yates and Ribeiro-Neto, 1999), spam filtering (Wang et al., 2008) and (Ketari et al., 2012), and medicine (Ching et al., 2018). Spam filtering maximizes the precision of classifying spam, this ensures that *only* spam emails are actually filtered, and maximizes the recall of not spam to ensure that the bulk of non-spam mails can pass through the filter unobstructed.

Practitioners commonly face the situation where a classifier performs well on one performance metric but poorly on others (Yeh, 2000). The general widely accepted consensus is that the practical requirements of a specific application determine the selection of appropriate performance metrics. Furthermore, many metrics are derived from a calculated confusion matrix, which implies that some of those performance metrics are closely related, which may cause redundancy (Liu et al., 2014).

To address this redundancy, the utilization of the F_1-Score (also F_1-Measure) was widely adopted across many fields (Baeza-Yates and Ribeiro-Neto, 1999) because it represents a balance between precision and recall.

Definition 2.1.5 (F_1-score)**.**
F_1-score (F_1) represent the harmonic mean between precision and recall and provides a good estimate of the overall quality of a model.

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall}$$

Another important metric that is used in both conventional machine learning algorithms and state-of-the-art neural networks is the *mean squared error*, which provides the basis for the optimization of the learning algorithm during training.

Definition 2.1.6 (Mean Squared Error (MSE))**.**
Mean Squared Error is a principal and frequently used metric, which measures the difference between the predicted value by a classifier and its true value.

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

where $\hat{y}_i$ is the model's prediction for the i-th observation.

The experiments in this thesis will focus on the maximization of F_1 or precision, depending on each specific task, while retaining high classification accuracy and minimizing the training error.

2.1.2 Generalization

The primary objective of machine learning is to develop *generalized* models that perform well on new and previously unseen inputs. To achieve this goal, the dataset $\mathbf{X}$ is commonly split into three distinct subsets: the *training set*, the *validation set*, and the *test set*.

- **training set**: $\mathbf{D} \subset \mathbf{X}$
  the part of the dataset that is used during training and for fitting the parameters of the classifier.
- **validation set**: $\mathbf{V} \subset \mathbf{X},\ V \not\subset D$
  often referred to as *development set*, the part of the dataset used to tune the hyperparameters of a classifier. It is used to evaluate the model developed on the training data.
- **test set**: $\mathbf{T} \subset \mathbf{X},\ \mathbf{T} \not\subset \mathbf{D} \vee \mathbf{V}$
  the part of the dataset the model has no access to during training, simulating new data. The performance on the test set is used as a proxy for the model's ability to generalize to new inputs.

The consensus amongst practitioners is that the three subsets are typically assumed to be *i.i.d.* This assumption expects that each subset ($\mathbf{D}, \mathbf{V}$ and $\mathbf{T}$) is *independent* from the others, and that they all are *identically distributed*, i.e., the observations are drawn from the same probability distribution. A violation of the *i.i.d.* property may introduce a sample selection bias and decrease the model's ability to generalize (Cho and White, 2011).

During training, we compute the *training error*, a measure of the model's performance on the training set, e.g., MSE (2.1.6), which we try to minimize. However, the actual measure we want to optimize is the *generalization error* or *test error*, the model's performance on the test set, which was not seen during training. This dichotomy between the optimization of the training error and the generalization error is known as the *bias-variance trade-off*. If the model is not able to achieve a low training error, it has high *bias*. This is usually caused by erroneous assumptions in the learning algorithm, e.g.: trying to find a linear separation when there is none, that causes it to overlook crucial relations in the data. Contrary to high bias, there is high *variance*, which measures a large gap between the training error and the test error. The model learns the training data closely, but it is extremely sensitive to small fluctuations and learns to model the random *noise* in the training data rather than its underlying distribution.

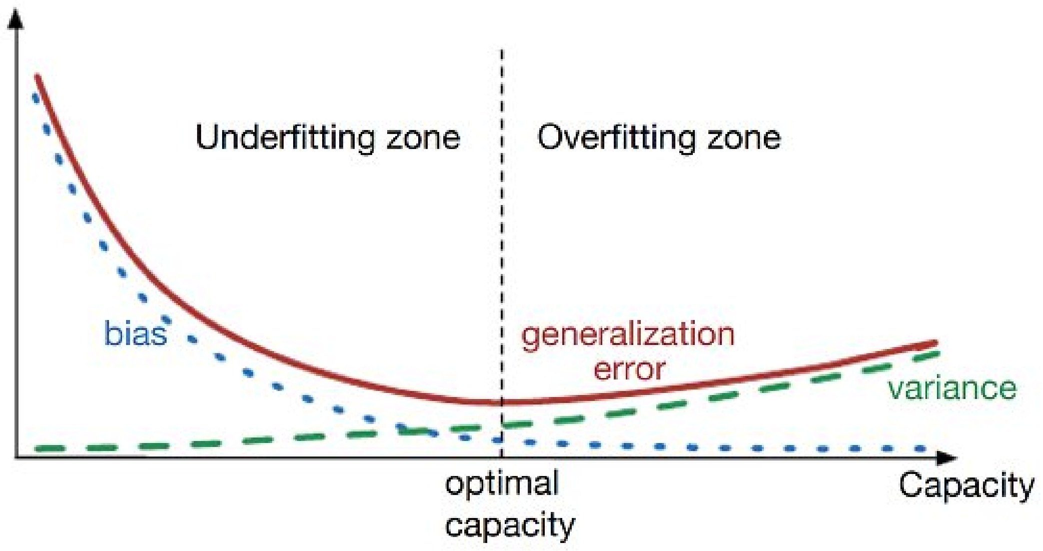


Figure 2.1: Visualization of the bias-variance trade-off (Goodfellow et al., 2016).

Figure 2.1 shows that with increasing model capacity (x-axis), the bias (dotted) tends to decrease and the variance (dashed) tends to increase, yielding an U-shaped curve for the generalization error (bold). If we vary the capacity along one axis, we can find the optimal capacity between the underfitting zone where the capacity is below this optimum and the overfitting zone when its above.

In machine learning, the *no free lunch theorem* (Wolpert and Macready, 1997) shows that there is no algorithm that is universally better than any other, given a noise-free dataset, i.e., no random noise - only trend, and a generalization error optimization problem. Wolpert (1996) proved that for all possible observations drawn from a uniform data generating probability distribution, the average performance (error rate) for all classification algorithms is the same.

Therefore, the goal in practice is to bias the algorithm towards distributions that we are more likely to encounter in the real world and to design algorithms that perform well on specific tasks. In particular, the difference between training error and generalisation error has been shown to grow with the capacity of the model, but shrink with the increasing number of training observations (Vapnik and Chervonenkis, 1971). If the model's capacity is too restricted, it will not be able to represent the full complexity of the distributions, while a high capacity model will provide many different solutions to the learning problem and is likely to focus on the nongeneric details of the particular training set (*overfitting*) (Hansen and Larsen, 1996).

If the dataset is too small and it is not possible to sensibly (*i.i.d.*) split the data into a training set and a test set, *cross-validation* is usually applied. Cross-validation is a method to resample data to assess the model's ability to generalize and to prevent overfitting. The most common variant is *k-fold cross-validation*, which splits the data into k equally sized subsets and repeats the training and evaluation k times, using $k-1$ splits for training and the remaining one for testing. Finally, the performance of all k runs is averaged and reported (Stone, 1978; Hastie et al., 2009).

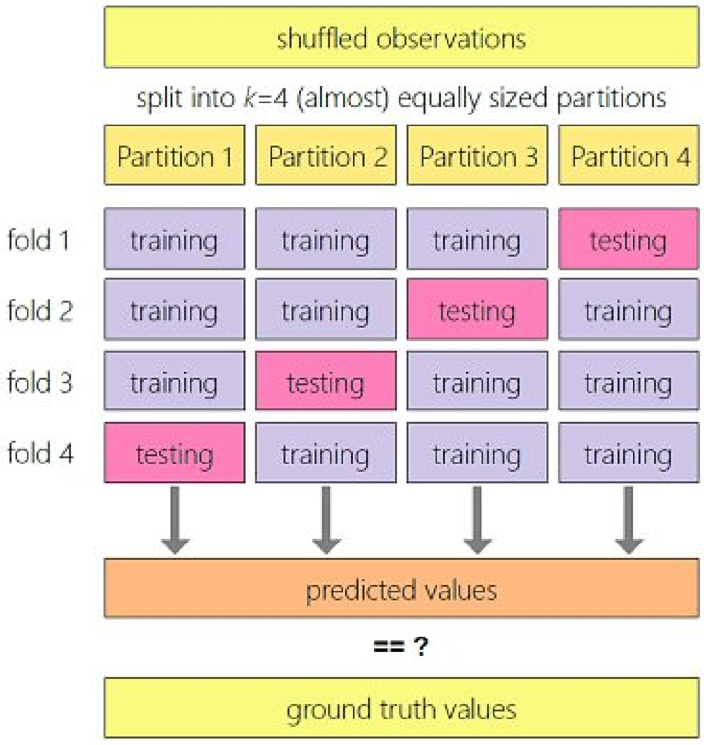


Figure 2.2: Visualization of k-fold cross-validation, where $k = 4$.

In our case, it was not feasible to extend the dataset by including additional observations of minority classes, since one employee spent an entire year manually labelling transcripts to generate the current dataset. Therefore, we applied k-fold cross validation for the training of all supervised models in this thesis.

2.2 Natural Language Processing

Natural Language Processing (NLP) aims to teach a computer to understand natural language by analyzing text based on both a set of theories and a set of technologies.

Definition 2.2.1 (Natural Language Processing)**.**
"Natural Language Processing is a theoretically motivated *range of computational techniques* for analyzing and representing *naturally occurring texts* at one or more *levels of linguistic analysis* for the purpose of achieving *human-like language processing for a range of tasks or applications*" (Liddy, 2001, p.2).

We can reduce the level of abstraction of this definition by dissecting some of its key components. The *range of computational techniques* offers room for multiple methods or techniques to accomplish a distinct type of language analysis. Next, the specification of *naturally occurring texts* limits the scope of the text to languages used by humans to communicate with one another, regardless of language, mode, or genre. The text can be *written* or *oral* as long as the prior limitations are met. The concept of *levels of linguistic analysis* captures the fact that humans produce or comprehend language through a combination of multiple types of language processing (see 2.2.2). NLP is a subfield of machine learning, but it strives for human-like performance. Therefore, *human-like language processing* indicates that NLP is also considered a discipline within Artificial Intelligence (AI). Last, *for a range of tasks or applications* is necessary because NLP in and of itself is not considered a goal. It is merely a means to accomplish a particular task. Thus, many NLP applications have been developed that pursue a specific goal while exploiting a range of NLP techniques (see §2.2.1) (Liddy, 2001). Summarizing, we can define the general goal of NLP as:

to develop a model that accomplishes human-like language processing.

Definition 2.2.2 (Levels of linguistics)**.**
The human language has a hierarchical structure and consists of many levels, ultimately reducing a word to its smallest indivisible unit, a single distinguishable sound (*phoneme)*.

Phonetics explores how linguistically relevant sounds of human languages are produced and how these sounds are perceived using both experimental and computational tools. It studies language at the level of sounds, how sounds can be distinguished and characterized by the manner they are produced. *Phonology* examines how sounds pattern in languages, how sounds are combined to make words, and how sounds are affected by their position in the word or phrase they occur (Halliday, 1964).

Morphology explores the structure of words and the principles that govern the formation of words through the combination of sounds into minimal distinctive units of meaning (*morphemes*). Composite words are made up of a number of units, e.g., *unhappiness* can be split into three separate morphemes: prefix *un*, root *happy*, and suffix *ness*, morphology deals with how languages add morphemes together (Aronoff, 1994).

Syntax investigates the grammatical structure of sentences and the common principles that determine how phrases and sentences are built up from words. It further explores the way that languages vary in their application of these common principles by looking at the variation across languages. Syntax conveys meaning in most languages because order and dependency contribute to meaning. The rules of syntax should explain how a grammatically correct and meaningful sentence is formed (Van Valin and LaPolla, 1997).

Semantics studies the meanings of words and sentences independently of any context. It seeks to explain how we achieve such a clear understanding of the language we use and analyzes the structure of meaning in language. Semantic processing determines the possible meanings of a sentence by focusing on the interactions among word-level meanings in the sentence (Karttunen, 1977; Liddy, 2001).

2.2.1 Natural language processing applications

Natural language processing provides both implementations and theory for a wide range of practical applications. Actually, any application that utilizes text is a candidate for NLP. A shortlist of the most frequent applications that utilize NLP includes the following:

- Information Retrieval (IR) - is concerned with the analysis, dissemination, organization, search, storage, and structure of information. An IR system is designed to make a given stored collection of documents available to the user. Historically, this information consisted of stored bibliographic items, e.g., online catalogs of books in a library or abstracts of scientific articles. However, in today's world, especially since the rise of Bing and Google, information is more likely to be full-length documents available in a widely distributed form, such as the Internet (Salton and Harman, 2003; Churcher, 2007).

- Language Modelling (LM) - is the use of various statistical and probabilistic techniques to determine the probability distributions of word sequences occurring in a sentence. This problem is usually reduced from learning word sequences to learning the conditional distribution of the next word given a fixed number of preceding words (Mnih and Hinton, 2007).

- Machine Translation (MT) - is one of the oldest of all NLP applications, its goal is to render in one language the meaning expressed by a word sequence in another language. It has developed significantly from its first purely statistical models (Brown et al., 1990) to today's state-of-the-art deep learning models (Singh et al., 2017).

- Question Answering - contrary to Information Retrieval, which provides a list of potentially relevant documents to the user's query, question answering provides the user with a textual output of the answer, or the passage in a document that provides the answer (Ravichandran and Hovy, 2002).
- Sentiment Analysis (SA) - studies the attitudes, emotions, and opinions of people towards an event, individual, or topic. SA focuses on identifying the sentiment expressed in a text and the subsequent analysis of the detected sentiment (Pang and Lee, 2008; Medhat et al., 2014).
- Text Classification (see §2.2.3) - is the process of categorizing a text into one of a set of predefined classes (Forman, 2003). This is the application addressed in the current thesis.

2.2.2 Natural language processing models

In this section, we present a selection of traditional NLP models that regard words as discrete symbols and enable a numerical representation of the textual inputs. There are many statistical techniques to generate corpus-wide statistics such as word counts and frequencies that can be passed as input to the machine learning models.

The most trivial text classification model is the *majority classifier*, which we introduce as the *naive classifier*. It does not look at the documents and does not need to transform them from their original textual representation. The model determines the majority class (see §2.1.2) from the class frequency distribution and predicts every new observation as belonging to the majority class.

Definition 2.2.3 (Naive classifier)**.**
The naive classifier always predicts the majority (most abundant) class in both binary- and multiclass classification.

In cases where the dataset is heavily skewed, e.g., $P(c_1|x_i) \geq 80\%$, this classification model can achieve very high accuracy ($:=$ 2.1.1). However, as discussed in §2.1.1, the F_1 score, which captures the overall fitness of a model across all classes, will most likely be very low. The subsequent models utilize the concept of *n-grams*, which represents a continuous sequence of n words.

Definition 2.2.4 (n-gram)**.**
An n-gram is a contiguous sequence of n items from a given sequence of text or speech. Based on the application, the items can be words, phonemes, morphemes, or letters.

A *unigram* is an n-gram of size one, i.e, it contains exactly one word. *Bigrams* and *trigrams* are n-grams of sizes two and three respectively. Brown et al. (1992) formalize this under the assumption that two histories are treated equivalently if they end up in the same $n-1$ words as: $k \geq n,\ P(w_k|w_1^{k-1}) = P(w_k|w_{k-n+1}^{k-1})$. They further estimate the parameters for a training text t_1^T through a training process. $C(w)$ is the number of times that the word w occurs in the text t_1^T. The n-gram model can be expressed as:

$$P(w_n \mid w_1^{n-1}) = \frac{C(w_1^{n-1}w_n)}{\sum_w C(w_1^{n-1}w)}$$

The utilization of n-grams might create reasonable word sequences which can provide additional insights to the data. However, we must choose the parameter n carefully and adapt it to the specific applications since the larger n becomes, the more computationally expensive the training becomes as the number of word sequences scales exponentially with n (Brown et al., 1992).

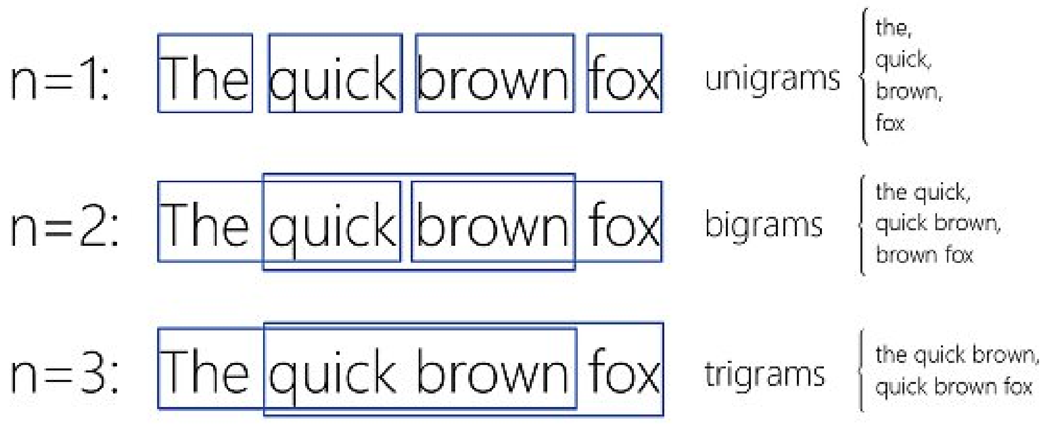


Figure 2.3: Exemplary split of a phrase into uni-, bi-, and trigrams.

A model that traditionally only used unigrams was the *bag-of-words* model, but it has since seen great successes with the inclusion of n-grams (Li et al., 2011, 2016, 2017).

Definition 2.2.5 (Bag-of-Words)**.**
The Bag-of-Words (BOW) is a method to quantize a text or a collection of documents to a multiset of words. BOW retains the multiplicity of words by counting the frequency of each unique word, but disregards grammar and syntax entirely.

The text $\mathbf{x} \in \mathbb{R}^{|V|}$ is represented as a BOW, where V is the vocabulary, and x_i the frequency of the i-th word in the vocabulary from the text. This frequency can be additionally weighted with the *term frequency-inverse document frequency* (tf-idf), which reflects the importance of a word in the entire collection of texts.

Definition 2.2.6 (term frequency–inverse document frequency)**.**
Term frequency–inverse document frequency (tf-idf) is a numerical statistic that reflects how important a *term* t (word) is to a *document* d (text) in a collection or corpus.

$$\textit{tf-idf}(t,d) = \textit{tf}(t,d) \times \textit{idf}(t), \;\; \textit{idf}(t) = \textit{log}(\frac{1+n}{1+\textit{df}(t)}) + 1$$

Our tf-idf formula slightly differs from the original formula because we add 1 in both the numerator and the denominator. This ensures that each word occurs at least once and prevents zero division (Aizawa, 2003; Robertson, 2004).

2.2.3 Text classification

In this thesis, all experiments are text classification tasks with different output variables. This section further explains the differences between the classification introduced in §2.1 and text classification.

First, we represent the dataset as the *document space* $\mathcal{X}$. It contains all observations d which are textual descriptions of a *document* $d \in \mathcal{X}$. The fixed set of classes $\mathbf{C} = \{c_1, \ldots, c_{|C|}\}$ is a unique feature and not part of the document space $\mathbf{C} \notin \mathcal{X}$. The terms *class* and *label* are interchangeable and will be used throughout the thesis. Second, many machine learning algorithms require a numerical representation of the documents rather than text. We achieve this by applying different NLP techniques and models to transform the document space into a high-dimensional feature space through, e.g., BOW or tf-idf (see 2.2.5, 2.2.6).

2.2.4 Word Vector Representations

Many NLP systems and techniques regard words as *atomic units*, thus, enabling frequency-based methods (BOW, tf-idf) to represent words as vectors. This often translates to a very sparse vector representation of the size of the vocabulary $|\mathbf{V}|$ with no notion of similarity between words, because they are represented as indices in the vocabulary. Those frequency based methods are called *one-hot* or *one-on* representations. Given a vocabulary of N words, this method assigns an integer index $i \in \{1, \ldots, N\}$ to each word. With this word-to-integer mapping, a word is subsequently represented as a N-dimensional sparse vector that is mostly composed of zeros, except for a single entry at the position corresponding to the word's index in the vocabulary that takes the value of the word's frequency in the text. We observe this behaviour because the number of words in the vocabulary greatly exceeds the words in any one document, enabling efficient calculation and computation (Mikolov et al., 2013).

We consider an example with two documents, d_1="the quick brown fox jumps over the lazy dog" and d_2="the lazy dog". First, the vocabulary V is created by iterating over all documents and adding each unique word to the vocabulary. The size of the vocabulary $V = \{\text{the, quick, brown, fox, jumps, over, lazy, dog}\}$ determines the length of the one-hot encoded vectors, i.e., $|\mathbf{V}| = 8$. Next, we iterate through every word in each document and map it to its corresponding one-hot encoded vector: $x_{the} = [2, 0, 0, 0, 0, 0, 0, 0]$, $x_{dog} = [0, 0, 0, 0, 0, 0, 0, 1]$, etc. Finally, the one-hot encoded word vectors of the same document are transposed, which yields a diagonal matrix and the determinant of that matrix is added to the final one-hot encoded matrix as shown below:

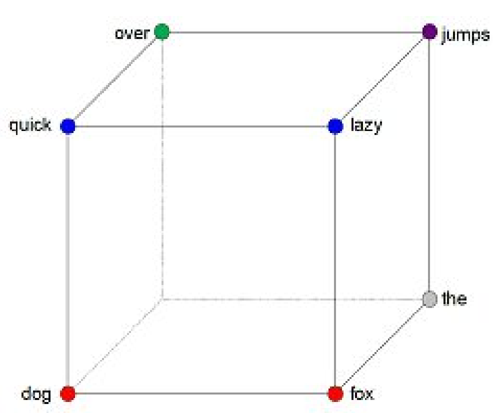
$$\begin{array}{cccccccccc} the & quick & brown & fox & jumps & over & lazy & dog & \\ \left(\begin{matrix} 2 \\ 1 \end{matrix}\right. & \begin{matrix} 1 \\ 0 \end{matrix} & \begin{matrix} 1 \\ 0 \end{matrix} & \begin{matrix} 1 \\ 0 \end{matrix} & \begin{matrix} 1 \\ 0 \end{matrix} & \begin{matrix} 1 \\ 0 \end{matrix} & \begin{matrix} 1 \\ 1 \end{matrix} & \left.\begin{matrix} 1 \\ 1 \end{matrix}\right) & \begin{matrix} \mathrm{d}_1 \\ \mathrm{d}_2 \end{matrix} \end{array}$$

This method has several benefits, namely, its simplicity, robustness, and ability to train simple models on large text corpora that outperform complex models trained on less data. Is is possible to train the popular n-gram model on virtually all available data (trillions of words) (Brants et al., 2007). However, the two major drawbacks of one-hot encoding are the loss of similarities between words (see Figure 2.4a), and the obvious feature size downsize, as the vector size scales proportionally to the size of the vocabulary.

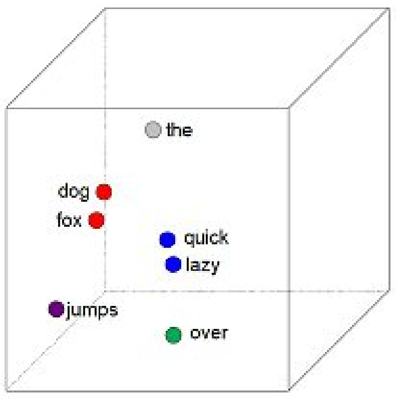
In recent years, the field of machine learning has seen great progress in the development of its techniques, it has become feasible to train more complex models on much larger datasets, and they consistently outperform the simple models. Furthermore, neural network-based language models significantly outperform n-gram models (Bengio et al., 2003; Schwenk, 2007). We will explain the basics of neural networks in §2.3 and focus on today's state-of-the-art models in §3.

Instead of this simplistic approach, NLP researchers turned to a very powerful idea of representing each word by means of its neighbours, under the assumption that words that occur in the same context tend to have similar meanings (Firth, 1957). Firth (1957) famously said: "You shall know a word by the company it keeps" and all *word embedding* methods rely on this assumption to some extent, even if their concrete implementations differ.

Definition 2.2.7 (word embedding)**.**
A *word embedding* is a dense, fixed-length, real-valued vector representation of a word. A word embedding model $\mathcal{W} \to \mathbb{R}^n$ is a parameterized function that maps words $w \in \mathcal{W}$ to n-dimensional word vectors. It serves as a dictionary of sorts for computer programs that would like to use word meaning (Bolukbasi et al., 2016).



(a) One-hot-encoded vector space.



(b) Semantically meaningful word embedding vector space.

Figure 2.4: Visualization of two approaches for encoding words in a vector space.

Figure 2.4 shows two different approaches of numerically representing texts. The left-hand side (a) shows the one-hot encoding that does not consider the similarities between words. The right-hand side (b) shows an idealistic word embedding representation that captures linguistically meaningful relationships between words.

In the last decade, word embeddings have become a core element of many NLP systems. The input of NLP systems is natural language, most commonly text, which is composed of smaller units that are not directly interpretable by computers. Therefore, word embeddings are deployed to transform the text to numerical representations that can be read, understood, and processed by computer programs. Word embeddings have become ubiquitous in deep learning models that achieve state-of-the-art on many natural language processing benchmarks due to representing the textual data numerically while retaining both the semantic and syntactic meanings of the words.

Word embeddings are used to represent the textual input as a semantically meaningful numerical representation in state-of-the-art deep learning models. We use them for the input representation of BERT (see §3.2.2).

Count-based models

The statistics of word occurrences in a text corpus provide the primary source of information that is available to all unsupervised machine learning models that learn word representations. However, the main challenge is how those statistics are used to generate meaning and how this meaning can be represented as word vectors. To generate a meaningful word vector representation, *count-based* models exploit co-occurrence matrices of words in a text. To build this matrix, the algorithm iterates over a massive dataset of texts to accumulate word co-occurrence counts, i.e., the frequency which shows how often two or more words occur together in the dataset.

In practice, there are two types of co-occurrences: *window-based* and *word-document*. In window-based co-occurrence the matrix $\mathbf{X}' \in \mathbb{N}^{N \times N}$ is limited by the specified window size N. $\mathbf{X'}_{ij}$ is the number of times word j appears in the specified window around the word i. This method allocates more weight to words in close proximity, i.e., within the specified window N, than to corpus wide statistics.

In *word-document* co-occurrences the global corpus statistics are considered and a well known model that exploits them is the Global Vectors for Word Representation (GloVe) model developed by Pennington et al. (2014). GloVe is an unsupervised learning algorithm that obtains vector representations for words. The training is performed on aggregated global word-word co-occurrence statistics from a large text corpus, and the resulting representations showcase interesting linear substructures of the word vector space. They represent the word-word co-occurrence counts as a matrix $\mathbf{X}$ for the global corpus statistics, where the entries $\mathbf{X}_{ij}$ tabulate the number of times that the word j occurs in the context of the word i. $\mathbf{X}_i = \sum_k \mathbf{X}_{ik}$ is the number of times any word appears in the context of word i. They determine the semantic similarity of two words, $i = ice$ and $j = steam$, by examining the ratio of their co-occurrence probabilities with various probe words k. The model returns a large ratio of co-occurrence probabilities if the words k are related to ice but not steam, e.g., $k = solid$. The model behaves similarly if the words k are related to steam but not ice. For words k that are either related to both ice and steam, or to neither, e.g., $water$ or $fashion$, the ratio should be close to 1. They formalize this as:

$$F(w_i, w_j, \hat{w}_k) = \frac{P_{ik}}{P_{jk}},$$

where $w \in \mathbb{R}^d$ are word vectors, $\hat{w} \in \mathbb{R}^d$ are separate context word vectors, and $P_{ij} = P(j|i) = \frac{\mathbf{X}_{ij}}{\mathbf{X}_i}$ the probability that word j appears in the context of word i.

Subsequently, they perform dimensionality reduction on the co-occurrence counts matrix to limit the solution space for F. Finally they factorize the matrix which returns a lower dimension matrix where each row is some vector representation for each word. The result is a semantically meaningful vector representation that has a small distance between related words (see 2.4 (b)).

Count-based word embedding models are used to generate the token embeddings of BERT (see §3.2.2), one of the models we tested in this thesis.

2.3 Neural Networks: Basics and Definitions

In recent years, neural networks (NN) have become ubiquitous in natural language processing. This section aims to provide an overview of the fundamental building blocks that constitute neural networks. We introduce them to provide the information that is necessary to follow the more advanced concepts that state-of-the-art neural networks employ.

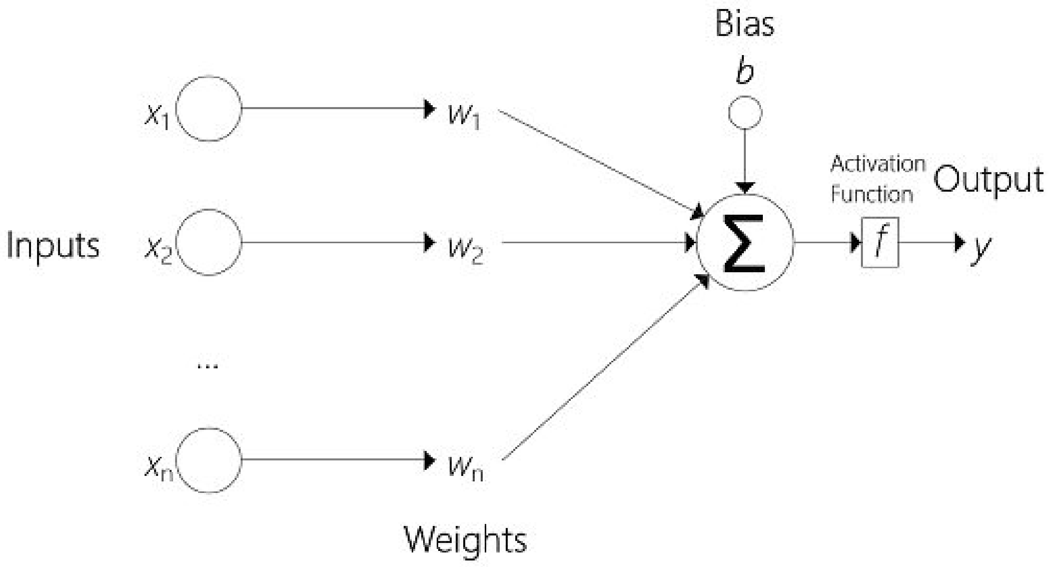


Figure 2.5: Visualization of a single neuron and its constituents.

Figure 2.5 shows a single neuron which consists of *inputs*, an *activation function*, and the *output*. The input can take the form of a n-dimensional vector $x \in \mathbb{R}^n$ and the output is computed as follows:

$$o = f(\mathbf{Wx} + \mathbf{b}),$$

where f defines an activation function, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix, $b \in \mathbb{R}^m$ the bias vector, and m the number of horizontally stacked neurons.

There are five commonly used activation functions:

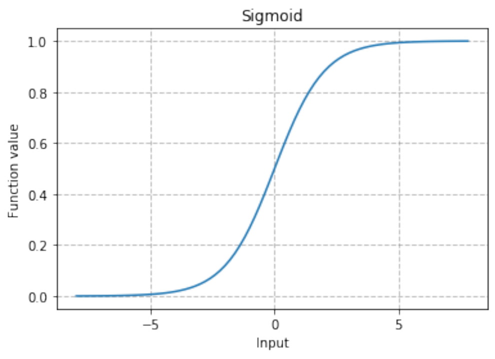
- sigmoid activation function or logistic function:
  it normalized the input to a value between 0.0 and 1.0:
  $$f(x) = \frac{1}{1 + e^{-x}}$$
- hyperbolic tangent activation function (Tanh):
  it is a sigmoidal function that normalizes the input to a value between −1.0 and 1.0:
  $$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
- softmax function, *softargmax*, or *normalized exponential function*:
  it is a generalization of the logistic function to multiple dimensions $f : \mathbb{R}^n \to \mathbb{R}^n$ and is defined as:
  $$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}},$$
  where $i \in [1, n]$, and $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$.

- rectified linear activation function (ReLu)
  it is a piecewise linear function that outputs the input directly if it is positive, otherwise it outputs 0.0

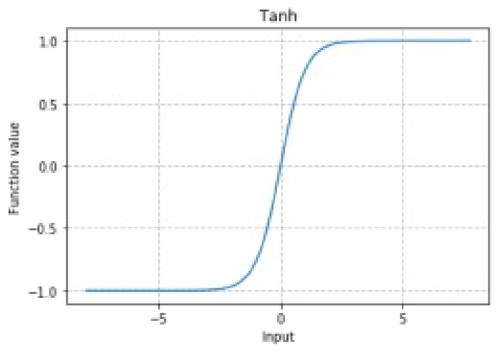
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

- Gaussian Error Linear Unit (GELU)
  it weights inputs by their value, rather than gating inputs by their sign as in ReLUs. $f(x) = xP(X \leq x) = x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian *cumulative distribution function* (cdf). We can approximate the GELU with:

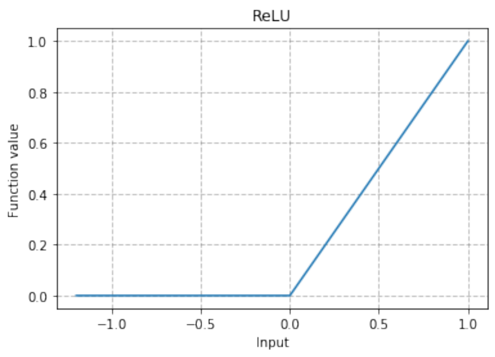
$$f(x) \approx 0.5x(1 + tanh[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)])$$



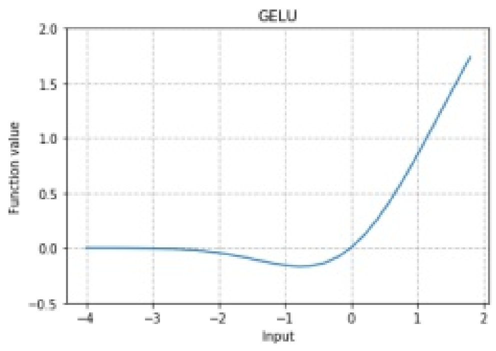
(a) logistic function (sigmoid)



(b) hyperbolic tangent (Tanh)



(c) rectified linear (ReLU)



(d) Gaussian Error Linear Unit (GELU)

Figure 2.6: Visualization of different types of activation functions.

The major concern of training deep neural networks is that after many iterations, i.e., updates of the weight vectors, the partial derivatives (steps of each update) of the error function, also called *gradients*, become *vanishingly* small. This effectively prevents updates of the weight vector, since the update steps are basically zero. On the other hand, if the values keep growing after every iteration, one can quickly run into the issue of *exploding gradients*, where the partial derivatives become exorbitant and the gradient updates become too large. These issues are addressed by applying an activation function before passing the output to the next layer, ReLU and GELU resolve the issue of vanishing gradients exceptionally well (see §2.3.1).

A neural network (NN) consists of many connected neurons, each producing a sequence of real-valued activations. The input neurons get activated through an activation function and return the output. The softmax and sigmoid functions are commonly used at the final or *output layer* of a neural network to obtain a categorical distribution. Non-output layers are referred to as *hidden layers*. A linear regression can be represented as a neural network without a hidden layer and a linear activation function. Generally, neural networks are named according to their number of hidden layers; a model with one hidden layer is called *one-layer feed-forward network*, which is also known as a *multilayer perceptron* (MLP):

$$\mathbf{o} = f_1(\mathbf{W_1 x} + \mathbf{b_1})$$

$$\mathbf{y} = \text{softmax}(\mathbf{W_2 o} + \mathbf{b_2}),$$

where f_1 is the activation function of the first hidden layer. Each layer has a unique parameterized weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$. The output $\mathbf{o}$ of the first layer is passed as input to the subsequent layer, which ultimately produces the output $\mathbf{y}$ of the entire neural network. This process is known as *forward propagation*.

In NLP, we work with text which is inherently sequential. Therefore, we must choose a neural network that can process a sequence of inputs. The most fundamental NN for sequential inputs is the *recurrent neural network* (RNN) (Elman, 1990). A RNN is a feed-forward neural network that has a dynamic number of hidden layers with the same parameters. This model does not focus on the *depth* of the network, but on its *width*. However, a major difference between a RNN and a MLP is that the RNN accepts a new input at every *layer* or time step. This is achieved by maintaining a hidden state $\mathbf{o}_t$ that represents the contents of the sequence at each time step t. Therefore, we can express the operations performed at every time step as:

$$\mathbf{o}_t = f_o(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{o_{t-1}} + \mathbf{b}_o)$$

$$\mathbf{y}_t = f_y(\mathbf{W}_y \mathbf{h}_o + \mathbf{b}_y),$$

where f_o and f_y are activation functions. The RNN applies $\mathbf{U}_o$ to modify the previous hidden state $\mathbf{o}_{t-1}$ and a transformation $\mathbf{W}_o$ to the current input $\mathbf{x}_t$, which produces the new hidden state $\mathbf{o}_t$. Additionally, at every time step t, the RNN produces an output $\mathbf{y}_t$. The RNN has trouble learning over a large number of time steps (Lipton et al., 2015).

As discussed in §2.2.7, word embeddings can be used in NLP tasks. Each word w_i in the vocabulary V that occurs in the input text is mapped to a vector $\mathbf{x}_i$, which is the *word embedding* of w_i. The word embeddings are stored in a word embeddings matrix $\mathbf{X} \in \mathbb{R}^{|V| \times d}$. Therefore, the original input sequence of words $w_1, \ldots, w_T$ is represented as a sequence of its corresponding word embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_T$, which is the - now numerical - input to a neural network.

Convolutional neural networks are a commonly used class of *deep* neural networks. We describe their application specifically to natural language processing tasks (LeCun et al., 1998; Collobert et al., 2011):

A convolutional layer slides *filters* $\mathbf{w} \in \mathbb{R}^{kd}$ of different window sizes k over the input word embeddings $[\mathbf{x}_1, \ldots, \mathbf{x}_T] \in \mathbb{R}^{T \times d}$, where d is the dimensionality of the word embeddings. Each filter generates a new *feature* $c_i \in \mathbb{R}$ for each window of k words $\mathbf{x}_{i:i+k-1} \in \mathbb{R}^{kd}$ according to the following formula:

$$c_i = f(\mathbf{w} \oplus \mathbf{x}_{_{i:i+k-1}+\mathbf{b}}),$$

where $\oplus$ refers to a concatenation between two vectors, f to an activation function, and $\mathbf{b} \in \mathbb{R}$ to a bias vector.

The filter $\mathbf{w}$ is applied to each possible window of words in the sentence $\{\mathbf{x}_{1:k}, \mathbf{x}_{2:k+1}, \ldots, \mathbf{x}_{T-k+1:T}\}$ to generate a *feature map*:

$$\mathbf{c} = [c_1, \ldots, c_{T-k+1}], \text{ with } \mathbf{c} \in \mathbb{R}^{T-k+1}$$

Each element $c_i \in \mathbf{c}$ in the feature map is the result of a calculation performed on a small segment of the input sequence $\{\mathbf{x}_{1:k}, \mathbf{x}_{2:k+1}, \ldots, \mathbf{x}_{T-k+1:T}\}$ and it shares parameters with its adjacent neighbours in the sequence by applying the same filter $\mathbf{w}$. Subsequently, a *max-pooling* strategy (Collobert et al., 2011) is applied over the feature map $\mathbf{c}$ and its maximum value $\hat{c} = \max\{\mathbf{c}\}$ is returned, with the premise that the most important feature is the one with the largest value. This process is repeated for each feature map of its corresponding filter. Finally, the maximum values of all feature maps produced by all filters are concatenated to a vector $\hat{\mathbf{c}} \in \mathbb{R}^C$, where C is the number of filters. This vector is forwarded to the next layer or a fully connected softmax output layer that returns the probability distributions over labels (Kim, 2014).

2.3.1 Error Backpropagation

The dynamic programming *backpropagation* algorithm (Bryson Jr et al., 1963; Rumelhart et al., 1986) follows the inverse direction of the feed-forward process to compute the gradients of the network's parameters. Dynamic programming enables an efficient method to re-use the parts of the feed-forward gradient computation that are identical in the backpropagation. These similarities become obvious once we apply the *chain rule of calculus* to compute the derivatives of the multiple layers. We formalize the chain rule, given two functions $y = g(x)$, and $z = f(g(x)) = f(y)$ we define the derivative $\frac{dz}{dx}$ of z with respect to x as the derivative of z with respect to y times the derivative of y with respect to x:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Given two vectors $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^n$, and a scalar z, we obtain the partial derivative of z with respect to x_i as follows:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

The gradient of z with respect to $\mathbf{x}$ containing all partial derivatives of z with respect to each x_i can be calculated through matrix-vector multiplication:

$$\nabla_{\mathbf{x}} z = (\frac{\partial \mathbf{y}}{\partial \mathbf{x}})^\top \nabla_{\mathbf{y}} z,$$

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m\times n}$ is the *Jacobian matrix* of g, it contains all partial derivatives. The backpropagation algorithm computes the Jacobian-gradient product for each operation in the neural network graph (Goodfellow et al., 2016). A deep feed-forward neural network with L layers, weight matrices $\mathbf{W}_l$ and bias vectors $\mathbf{b}_l$ where $l \in \{1, \ldots, L\}$ takes the input word embeddings $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_T] \in \mathbb{R}^{T\times d}$, where d is the dimensionality of the word embeddings, creates and output $\hat{y}$, and minimizes a *cost* or *loss function* $J = \mathcal{L}(\hat{y}, y)$:

$$\mathbf{o}_l = \mathbf{b}_l + \mathbf{W}_l\mathbf{o}_{l-1}$$

$$\mathbf{y}_l = f_l(\mathbf{o}_l),$$

where $\mathbf{o}_0 = \mathbf{x}$, and $\hat{y} = \mathbf{o}_L$. The feed-forward propagation advances from the first layer, computing the output $\mathbf{o}_l$ that passes through an activation function f_l to create a normalized output $\mathbf{y}_l$, which is provided to its subsequent layer. Figure 2.7a depicts a simplified forward pass with inputs $a = 2$ and $b = 1$, and output $e = c * d$. This process is repeated until the output $\hat{\mathbf{y}} = e$ and the loss J are calculated.

A common loss function that is often deployed in regression tasks is the MSE (see 2.1.6). However, deep neural networks in NLP utilize a more sophisticated loss function, the *cross-entropy loss*.

Definition 2.3.1 (Cross-entropy loss)**.**
Cross-entropy is a measure that builds upon entropy and calculates the difference between two probability distributions. It heavily penalizes incorrect predictions that are predicted with very high confidence and correct predictions with low confidence. It is calculated as follows (Zhang and Sabuncu, 2018):

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$$

The backpropagation algorithm executes these calculations in reverse order. First, it computes the gradient $\nabla_{\hat{y}} J$ of the loss function J with respect to the output $\hat{y}$. Next, it computes the gradients of the hidden state $\mathbf{y}^{(L)}$ and the output $\mathbf{o}^{(L)}$ for the last layer L, and reuses the already calculated gradient of the forward pass. This process is repeated until the algorithm arrives at the gradients of the first layer $l = 0$. The gradient $\nabla_{\hat{y}} J$ of the loss function with respect to the output is:

$$\nabla_{\hat{y}} J = \nabla_{\hat{y}} \mathcal{L}(\hat{y}, y),$$

and can be expressed as the gradient of the loss function with regard to the layer's output $\mathbf{o}_l$ (before the activation function is applied) with the chain rule:

$$\nabla_{\mathbf{o}^{(l)}} J = (\frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{o}^{(k)}})^\top \nabla_{\hat{y}} J = f'(\mathbf{o}^{(l)})$$

This formula shows why we choose easily differentiable activation functions f. The backpropagation algorithm efficiently reduces the computation of the gradient of the loss function with respect to the layer output to the first derivative of the layer output. We can now obtain the gradients of the model's parameters based on each parameter's unique contribution:

$$\nabla_{\mathbf{b}_l} J = \nabla_{\mathbf{o}_l} J$$

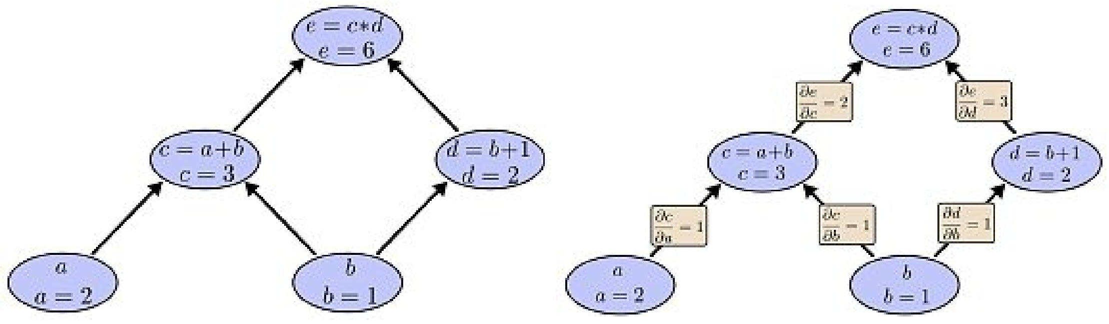
$$\nabla_{\mathbf{W}_l} J = \nabla_{\mathbf{o}_l} J \mathbf{y}_{l-1}^\top$$

The algorithm subsequently propagates the gradient to its underlying layer and repeats the calculations:

$$\nabla_{\mathbf{y}_{l-1}} = \mathbf{W}_l^\top \nabla_{\mathbf{W}_l} J$$

To compute the gradient of a parameter, the gradients of all computations that are part of the results of this parameter must be known. Figure 2.7b visualizes how to backpropagation algorithm re-uses the calculated values from the forward pass to calculate the partial derivatives. Subsequently, *gradient descent* is used to update the parameters of the corresponding layers by the value of the computed gradients.

We have already mentioned the issues of vanishing and exploding gradients, but this section has proven the utility of differentiable activation functions. In deep recurrent neural networks, the hidden state is multiplied often with the weight matrix. During backpropagation, this leads to repetitive multiplications of gradients with the same values. If the factors are small, they eventually converge towards zero, therefore the issue of *vanishing gradients* causes no updates of the parameters. On the other hand, if the factors are large, they eventually diverge towards infinity, hence the issue of *exploding gradients*. In both cases, the model is not able to learn, i.e., the model's parameters are not updated.



$e = c*d$
$e = 6$
$c = a+b$
$c = 3$
$d = b+1$
$d = 2$
a
$a = 2$
b
$b = 1$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$

(a) Visualization of the forward pass.

(b) Visualization of the backpropagation.

Figure 2.7: Illustration of the backpropagation algorithm[1].

2.3.2 Regularization

As discussed in §2.1.2, the primary objective of machine learning is to develop generalized models that not only perform well on the training data, but also on new inputs. There are many strategies that reduce the test error at the expense of a higher training error. In deep learning, these strategies are collectively known as *regularization*.

Regularization can be applied to any component of the deep learning model, either during the training process or the prediction procedure, which can account for the limitation of finite training data. Some regularization strategies put additional constraints on the model, such as including restrictions on the parameter values. If the constraints and penalties are chosen carefully, the new restrictions can lead to improved performance of the test set. In deep learning, most regularization strategies rely on regularizing estimators. The regularization of an estimator trades increased bias for reduced variance. An effective regularizer is one that balances the dichotomy between reducing the variance significantly while not overly increasing the bias (Goodfellow et al., 2016).

This section introduces a selection of the most common deep learning regularization methods available.

L^2 parameter regularization
L^2 regularization commonly known as *weight decay*, or ridge regression, is a regularization strategy that drives the parameters closer to the origin by adding a regularization term to the objective function. It forces the weights to become smaller, but they do not become zero:

$$\mathcal{L}_{\text{reg}}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\delta}{2}\|\mathbf{w}\|^2 + \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}),$$

where $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ is a vector of all weights, $\mathcal{L}$ the loss function, $\|w\|^2 := \sqrt{w_1^2 + \cdots + w_n^2} = \sqrt{\mathbf{w} \cdot \mathbf{w}}$ the L_2 norm (Euclidean norm), and $\delta \in (0, 1)$ the regularization parameter often $\delta \in [0.0001, 0.01]$. Weight decay is very common and almost

[1]Images created by Christopher Olah

always used in practice since it prevents certain inputs from dominating the output and encourages the model to use all inputs. By applying the product rule we can rewrite the normalized weight vector $\|w\|^2$ as $\mathbf{w}^\top\mathbf{w}$ and its gradient becomes $2\mathbf{w}$. The resulting gradient is:

$$\nabla L_{\text{reg}}(\mathbf{w}) = \delta\mathbf{w} + \nabla L(\mathbf{w}),$$

and the gradient update becomes:

$$\begin{aligned}\mathbf{w} &= \mathbf{w} - \alpha(\delta\mathbf{w} + \nabla L(\mathbf{w}))\\ \mathbf{w} &= \mathbf{w} - \alpha\delta\mathbf{w} - \alpha\nabla L(\mathbf{w})\\ \mathbf{w} &= (1-\alpha\delta)\mathbf{w} - \alpha\nabla L(\mathbf{w}),\end{aligned}$$

where α is the learning rate.

Dropout

Deep learning models often require many parameters with a high degree of expressiveness to capture the complex features of text classification. They are capable to overfit towards specific features; in some extreme cases only a few features are used. Generally, this issue can be addressed by increasing the size of the dataset, effectively providing additional highly expressive features for the network to learn. However, it is not always feasible to obtain new observations.

Dropout (Srivastava et al., 2014) is a regularization technique that improves NN by reducing overfitting through the co-adaptation of neurons by randomly dropping them out, i.e., the output of the neuron is 0 with a probability of p, and the input is forwarded with a probability of $1-p$ during training. Standard backpropagation learning relies on building extensive co-adaptations of neurons that work exceptionally well for the training data, but do not generalize well to new data. Random dropout breaks up these co-adaptations and ensures that the NN learns not to rely on certain neurons by temporarily discarding them.

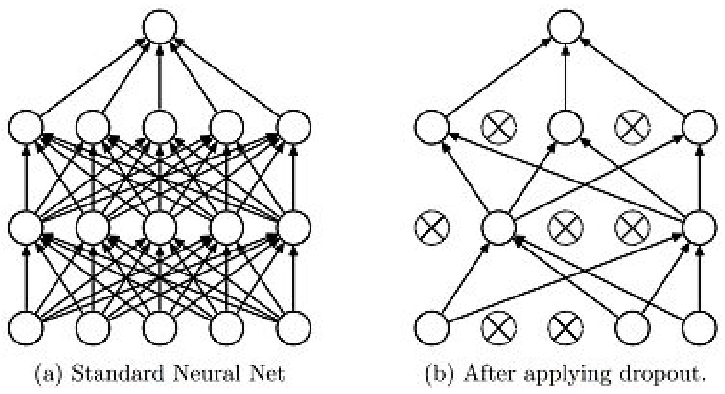


Figure 2.8: Visualization of neural dropout (Srivastava et al., 2014).

Figure 2.8 shows a standard neural network with two hidden layers (a) and the thinned neural network produced by applying dropout to the network (b). The crossed units have been dropped and are not connected to the network anymore.

Early stopping

When training deep neural networks with large enough capacity, we usually observe that the training error decreases consistently over time, but at some point the validation error increases again. This behaviour occurs consistently given enough time steps.

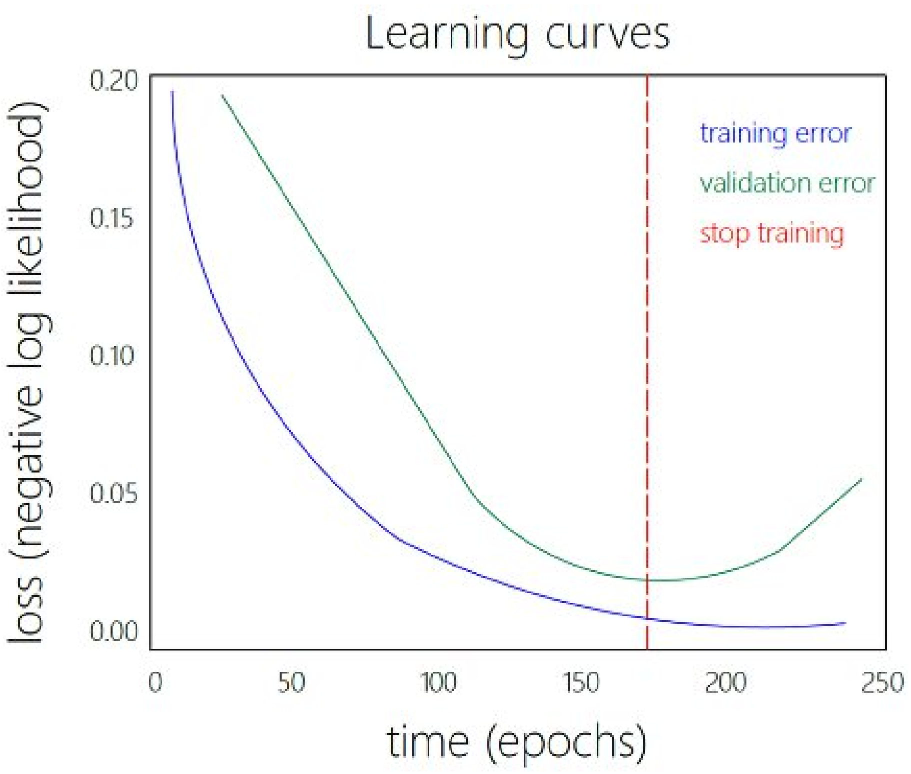


Figure 2.9: Visualization of early stopping, adapted from (Goodfellow et al., 2016).

We can exploit this behaviour by returning to the parameters (weights $\mathbf{w}$) of the time step with the lowest validation error. *Early stopping* aims to minimize overfitting by storing a copy of the parameters with the lowest validation error v, terminating the training algorithm if v has not been reduced for e epochs, and returning the stored parameters associated with the lowest validation error.

Algorithm 2.1: The early stopping meta-algorithm that determines the best amount of time for training.

```
   Result: Best parameters $\mathbf{w}^*$
1  Let $n$ be the number of steps between evaluations.
2  Let $p$ be the "patience", the number of epochs to observe a decrease in the
     validation error before terminating.
3  Let $\mathbf{w}$ be the initial parameters.
4  $\mathbf{w} \leftarrow \mathbf{w}_0$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $\mathbf{w}^* \leftarrow \mathbf{w}$; $i^* \leftarrow i$
5  while $j < p$ do
6      Update $\mathbf{w}$ by running the training algorithm for $n$ steps
7      $i \leftarrow i + n$
8      $v' \leftarrow \text{ValidationError}(\mathbf{w})$
9      if $v' < v$ then
10         $j \leftarrow 0$
11         $\mathbf{w}^* \leftarrow \mathbf{w}$
12         $i^* \leftarrow i$
13         $v \leftarrow v^*$
14     else
15         $j \leftarrow j + 1$
16     end
17 end
```

In practice, this reduces the total time required for training by terminating the process early, i.e., after the algorithm has not seen a reduction in validation error for e time steps. Additionally, we select a model with a good validation performance, which should be representative of the generalized test performance.

CHAPTER 3

BERT

BERT (Devlin et al., 2019), which stands for Bidirectional Encoder Representations from Transformers, is a deep contextual language representation model developed by Google AI. It is designed to pretrain deep bidirectional representations of word sequences from unlabeled text by jointly conditioning on both the left and right context in all of its layers. Consequently, the pretrained BERT model can be finetuned by including one additional output layer to create state-of-the-art models for a wide range of NLP tasks, without requiring extensive task-specific architecture modifications.

This chapter introduces BERT and its implementation which has obtained state-of-the-art results on many NLP tasks, e.g., text classification, question answering, and language inference. Specifically, we describe the model architecture, input representations, parameters, pretraining procedure, and different fine-tuning strategies for downstream tasks.

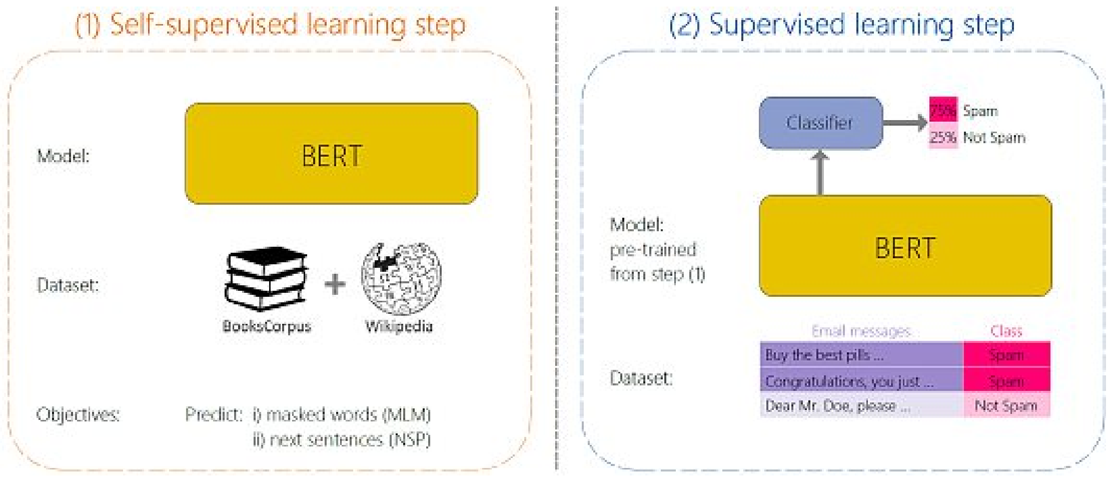


Figure 3.1: Learning steps of BERT: (1) pretraining and (2) fine-tuning.

Figure 3.1 depicts the two unique learning steps of BERT. (1) is the self-supervised training on large amounts of texts, e.g., book corpora or Wikipedia data. The model is trained on two NLP tasks, masked language modelling (MLM) and next sentence prediction (see §3.3). Once the pretraining process has concluded, the BERT model has language-processing abilities capable of empowering many models that can be build on top of it in a supervised fashion. This process is captured in the second learning step (2), where the pretrained model (1) is used as a foundation and modified to a specific NLP task and dataset. (2) shows the supervised training of a binary spam classification task with a labeled dataset.

3.1 Self-attention

In deep learning, the concept of *attention* can be described as mapping a query and a set of key-value pairs to an output, where keys, query, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is the correlation between the query and one corresponding key (Bahdanau et al., 2016). Figure 3.2 depicts an example text, split into words that serve as the model input (left), where the attention values are computed by one layer, and the output values between the query *it* and all keys, i.e., all remaining words (right).

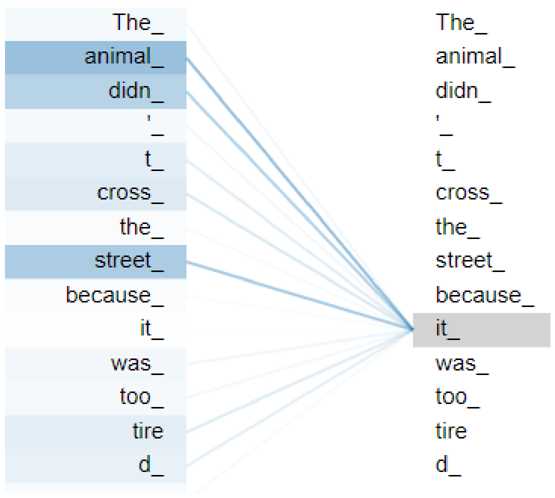


Figure 3.2: Visualization of attention values for word *it* in entire sentence[1].

Self-attention (Shaw et al., 2018) is a special form of attention that was first introduced with the Transformer model (Vaswani et al., 2017). It relates different positions of a single input sequence $\mathbf{x} = (x_1, \ldots, x_n)$ of n elements where $x_i \in \mathbb{R}^{d_x}$, to compute a contextual representation $z_i \in \mathbb{R}^{d_k}$ of the same length. Each output element z_i is computed as a

[1]Example created from the Tensor2Tensor Google Colab notebook.

weighted sum of a linearly transformed input elements:

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j \mathbf{W}^V),$$

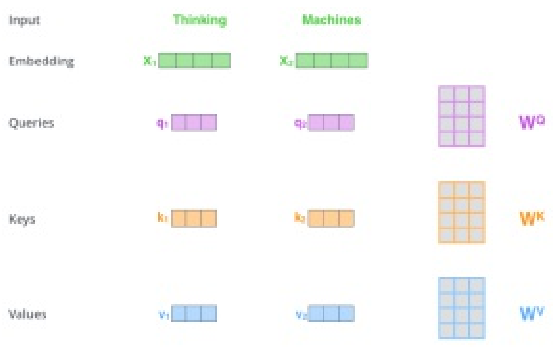
where each weight coefficient α_{ij} is computed by applying a softmax function:

$$\alpha_{ij} = \frac{\exp\ e_{ij}}{\sum_{k=1}^{n} \exp\ e_{ik}},$$

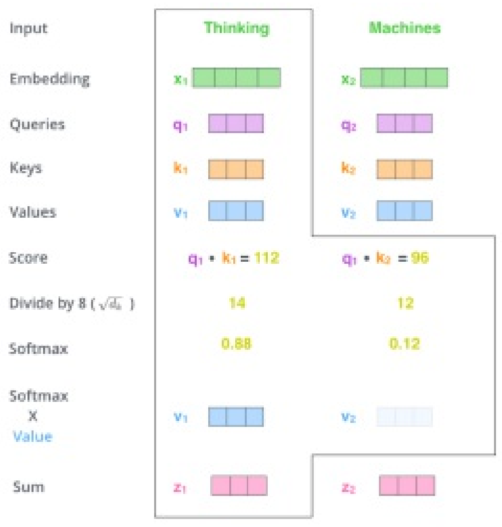
and e_{ij} is computed using a compatibility function that compares two input elements:

$$e_{ij} = \frac{(x_i \mathbf{W}^Q)(x_j \mathbf{W}^K)^\top}{\sqrt{d_k}},$$

where $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V \in \mathbb{R}^{d_x \times d_k}$ are parameterized matrices that are unique for each layer in the NN. Vaswani et al. (2017) claim that for large values of d_k, the dot products of two input elements $(x_i \mathbf{W}^Q)(x_j \mathbf{W}^K)^\top$ become extremely large in magnitude, which pushes the softmax functions into regions that run into the problem of vanishing gradients for e_{ij}. This problem is solved by dividing the dot product through the normalized output dimension $\sqrt{d_k}$ before applying the softmax function.



(a) Computation of query, key, and value vectors $x_i\mathbf{W}^Q$, $x_i\mathbf{W}^K$, and $x_i\mathbf{W}^V$ respectively.



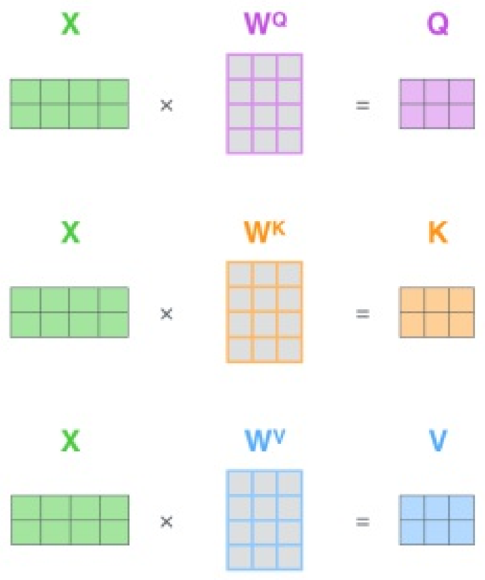
(b) Computation of the self-attention output vector z_i.

Figure 3.3: Illustration of the vector representation for input and output of self-attention (Alammar, 2018a).

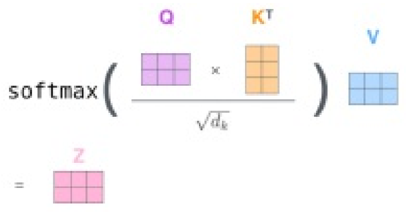
Figure 3.3 shows the process of calculating self-attention using vector representation. The first step is to create three vectors from each of the model's input vectors, i.e., the embeddings of each word. A *query vector*, *key vector*, and *value vector* are calculated and assigned for every word in the sequence. The next step is to determine the score e_{ij} by calculating the dot product of the query vector $x_i\mathbf{W}^Q$ and the key vector $x_i\mathbf{W}^K$ of the word that is currently scored. In this example, the first score for *Thinking* would be the dot product of q_1 and k_1, and the second score the dot product of q_1 and k_2. Then the score is normalized by the square root of the key vector dimension and the softmax function. The final step is to calculate the product between each value vector and its corresponding softmax score, summing the weighted value vector, and returning the self-attention score z_i for the word (z_1 for the first word *Attention*).

In practice, this process can be significantly sped up by applying the self-attention function to a set of input sequences simultaneously by representing the input embeddings as a matrix $\mathbf{X}$. We can represent N queries as a matrix $\mathbf{Q} \in \mathbb{R}^{N \times d_k}$. We adopt the same representation for the keys $\mathbf{K} \in \mathbb{R}^{N \times d_k}$ and values $\mathbf{V} \in \mathbb{R}^{N \times d_k}$. The output, i.e., the self-attention score matrix $\mathbf{Z}$, can be calculated through efficient matrix multiplication as follows:

$$\text{Self-attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{Z} = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V}$$



(a) Computation of query, key, and value matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ respectively.



(b) Computation of the self-attention output matrix $\mathbf{Z}$.

Figure 3.4: Illustration of the matrix representation for the input and output of self-attention (Alammar, 2018a).

3.1.1 Multi-head attention

Transformer-based models do not perform a single attention function with d_{model}-dimensional keys, queries, and values, but linearly project the keys $\mathbf{K}$, queries $\mathbf{Q}$, and values $\mathbf{V}$ h times with different learned linear projections $\mathbf{P}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{P}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{P}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, where $i \in [1, h]$ to d_k, d_q, and d_v dimensions respectively. For every projected version of the key, query, and value set, we perform the attention function in parallel, generating d_v-dimensional output values. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions, increasing the information gain. This method generates h different output matrices $\mathbf{Z}_i \in \mathbb{R}^{N \times d_{\text{model}}}$, called *attention heads*. Figure 3.5 depicts this process. Next, we concatenate the attention heads and project them into another representation subspace $\mathbf{W}^O \in d_{\text{model}} \times d_{\text{model}}$, which yields the final multi-head attention output matrix $\mathbf{Z} \in \mathbb{R}^{N \times d_{\text{model}}}$. This computation can be formally expressed as:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = (\mathbf{Z}_1 \oplus \cdots \oplus \mathbf{Z}_h)\mathbf{W}^O,$$

$$\text{where } \mathbf{Z}_i = \text{Attention}(\mathbf{Q}\mathbf{P}_i^Q, \mathbf{K}\mathbf{P}_i^K, \mathbf{V}\mathbf{P}_i^V),\ i \in [1, h],$$

where $\oplus$ is the concatenation operator, and $d_k = d_v = \frac{d_{\text{model}}}{h}$.

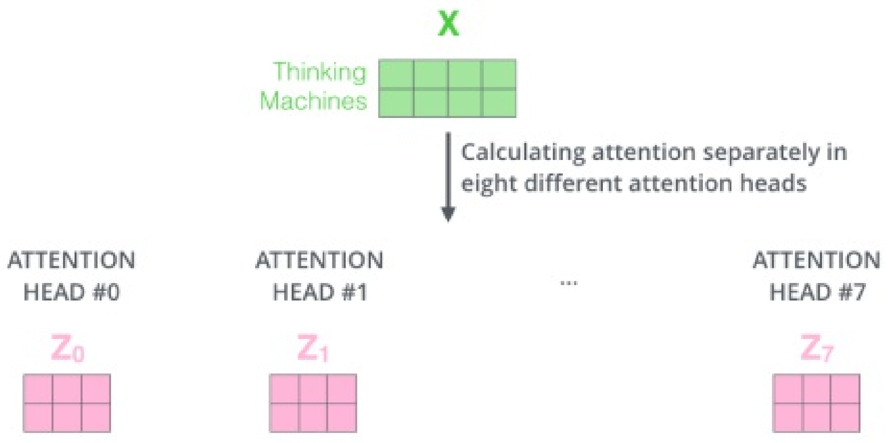


Figure 3.5: Computation of the attention heads $\mathbf{Z}_i$, $i \in [i, h]$ (Alammar, 2018a).

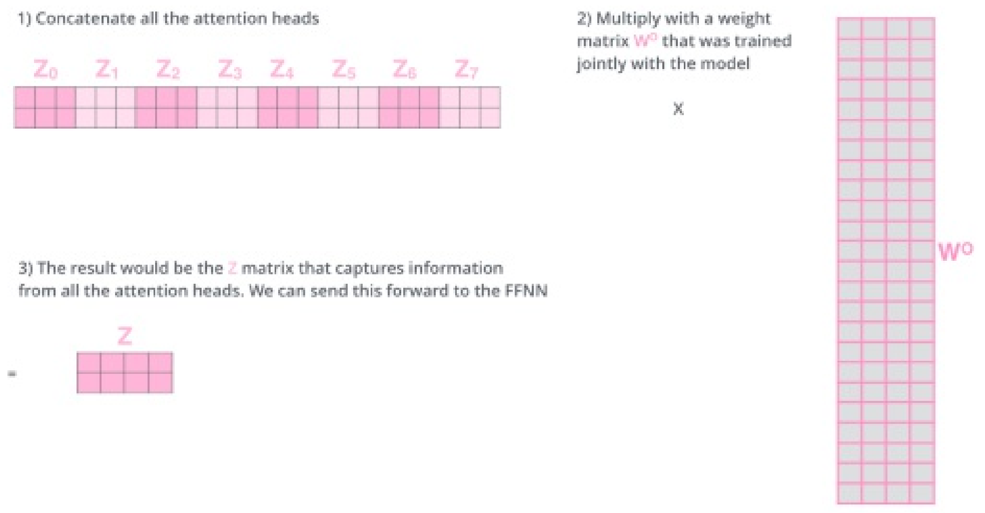

1) Concatenate all the attention heads
Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7
2) Multiply with a weight matrix W^O that was trained jointly with the model
x
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
Z
=

Figure 3.6: Computation of the final multi-head attention output matrix **Z** (Alammar, 2018a).

Figure 3.6 visualizes the parallel multi-head attention calculation with $h = 8$ different attention heads, the same value used in the original Transformers paper (Vaswani et al., 2017). BERT uses 12 or 16 attention heads dependent on the model architecture.

3.2 Model

In this section, we introduce the BERT model and its detailed implementation. We describe the model architecture including two different BERT versions, the BERT-specific input representation with the required preprocessing, and the model parameters.

There are two steps in the original framework: *(1) pretraining* and *(2) fine-tuning*. During pretraining, the model is trained on a large text corpora of unlabeled data over different pretraining tasks. For fine-tuning, BERT's first parameters are initialized from an already pretrained model, and all parameters are fine-tuned on a labeled dataset from a downstream task. These methods are explained in detail in §3.3 and §3.4 respectively.

3.2.1 Model architecture

BERT's model architecture is a multilayer bidirectional Transformer encoder inspired by the original Transformer implementation (Vaswani et al., 2017). The model consists of a stack of L identical layers, i.e., Transformer encoder blocks. Each layer L contains two types of sublayers. (1) *multi-head self-attention* mechanism that determines the relevance of all other words in the sequence while encoding an individual word and (2) position-wise fully connected *feed-forward network* , that consists of two linear transformations

$\left(\mathbf{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}, \mathbf{b}_1 \in \mathbb{R}^{d_{\text{ff}}}\right), \left(\mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}, \mathbf{b}_2 \in \mathbb{R}^{d_{\text{model}}}\right)$ that are applied to each position of the network. We can express this as:

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

The dimension of both input and output is fixed at d_{model}, and the dimension of the inner layer is $d_{\text{ff}} = 4d_{\text{model}}$. Furthermore, the feed-forward network uses the GELU activation function (see §2.3) rather the the commonly used ReLU function, because it was shown that GELU performs better within a Transformer encoder. Additionally, the encoder layer contains a residual connection (He et al., 2016) around both of its two sublayers, and is subsequently followed by a layer normalization (Ba et al., 2016) such that the output of each sublayer can be expressed as:

$$\text{LayerNorm}(\mathbf{x} + \text{Sublayer}(\mathbf{x})),$$

where Sublayer($\mathbf{x}$) represents the function that is implemented within the sublayer itself. To enable the residual connections, all sublayers of the model must produce outputs of the same dimensionality d_{model}. The left side of Figure 3.7 shows the 12 encoder layers and the right side depicts the detailed architecture of a single encoder.

As we have shown, the Transformer architecture allows us to stack L encoder layers to create a deeper model that is able to learn more distinct interconnected intra-layer features. The two most common BERT versions are:

- BERT-base: $L = 12$, $d_{\text{model}} = 768$, $h = 12$, $d_{\text{ff}} = 3072$ (110M total parameters)
- BERT-large: $L = 24$, $d_{\text{model}} = 1024$, $h = 16$, $d_{\text{ff}} = 4096$ (340M total parameters),

where L denotes the number of encoder layers, d_{model} the dimension of the input and output (which must be the same for all layers), h the number of attention heads in a self-attention sublayer, and d_{ff} the number of hidden units in a feed-forward sublayer. If the model has more parameters, it is able to learn more distinct features from the input sequence, however, this comes at the cost of additional computational resources and required time for both training and inference.

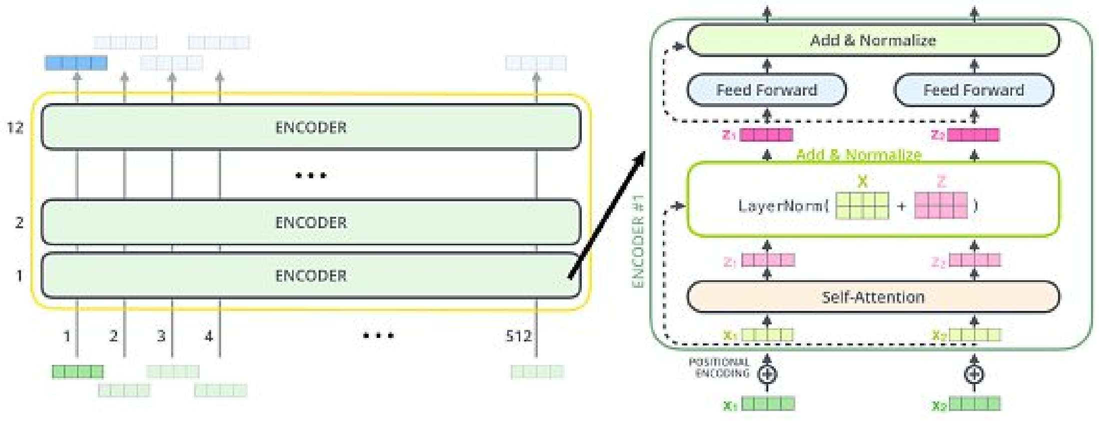


Figure 3.7: BERT's Transformer encoder architecture (Alammar, 2018b).

3.2.2 Input representation

In natural language processing, the input is usually a written text which must be first transformed into a numerical representation before passing it as input to a model (see §2.2.2).

BERT receives the input, a sequence of words that is limited to 512 tokens, and performs a transformation of the tokens to obtain a numerical representation which can be subsequently passed to the model. The input representation consists of three distinct embedding types: *token*, *segment*, and *positional* embeddings. Figure 3.8 shows the three components of the input embedding.

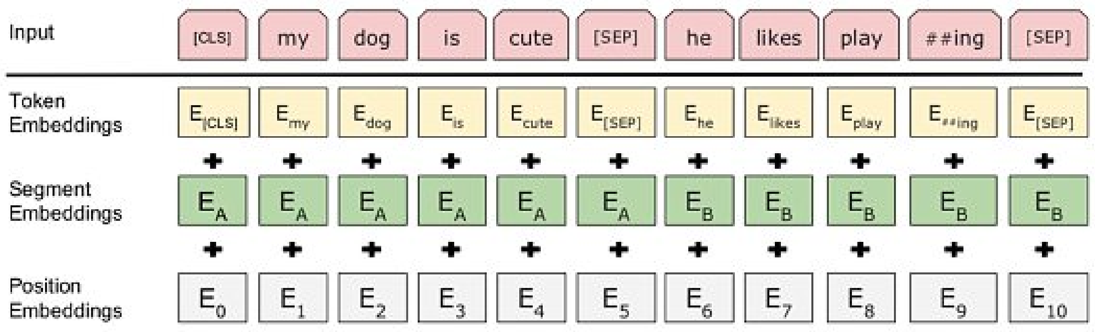


Figure 3.8: BERT input representation (Devlin et al., 2019). The sum of *token*, *segment*, and *positional* embeddings constitutes the input embeddings.

Token embeddings (TE)

BERT exploits WordPiece embeddings (Wu et al., 2016) to tokenize the words in an input sequence. WordPiece is a subword segmentation algorithm that generates a fixed-sized vocabulary containing individual characters, subwords, and words in a specific language.

The process of adding a token to the vocabulary is as follows: First, check if the word is in the vocabulary. If not, try to break it into the largest possible subwords contained in the vocabulary. Last, if the prior steps were unsuccessfully executed, decompose the word into its individual characters. Once the entire word has been processed into one or multiple WordPiece tokens, the ID's of those tokens are used to retrieve the corresponding embeddings in the learned token-embedding matrix (see Figure 3.9).

BERT's vocabulary contains 30'000 tokens consisting of the most common words and subwords in the English language and three BERT specific special tokens:

- `[CLS]`, is the first token of every sequence. The final hidden state that corresponds to this token is used as the aggregated sequence representation for classification tasks.
- `[SEP]`, is used to delimit a sequence that contains sentence pairs and is always used to mark the end of a sequence.
- `[MASK]`, is a token deployed during training to optimize the masked language modeling (MLM) objective function (see §3.11).

Segment embeddings (`SE`)

A sequence can contain sentence pairs, and the sentence embeddings are appended to every sentence indicating whether it belongs to sentence `A` or sentence `B` of the sentence pair (see Figure 3.9).

Positional embeddings (`PE`)

BERT uses *positional* embeddings to insert information about the relative or absolute position of the tokens within the input sequence. Similarly to the dimensionality requirements of the multiple encoder layers, the three embeddings, i.e., token, segment, and positional, must have the same dimension d_{model} to enable efficient summation. They can be computed as follows:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right),$$

where i represents the dimension, and pos the position. This allows us to depict each dimension of the positional embedding as a sinusoid, with a wavelength that follows a geometric progression from 2π to 20000π. According to Vaswani et al. (2017) this enables the model to easily learn the relative positions since for any fixed offset k, the corresponding positional embedding $\text{PE}(pos + k)$ can be represented as a linear function of $\text{PE}(pos)$.

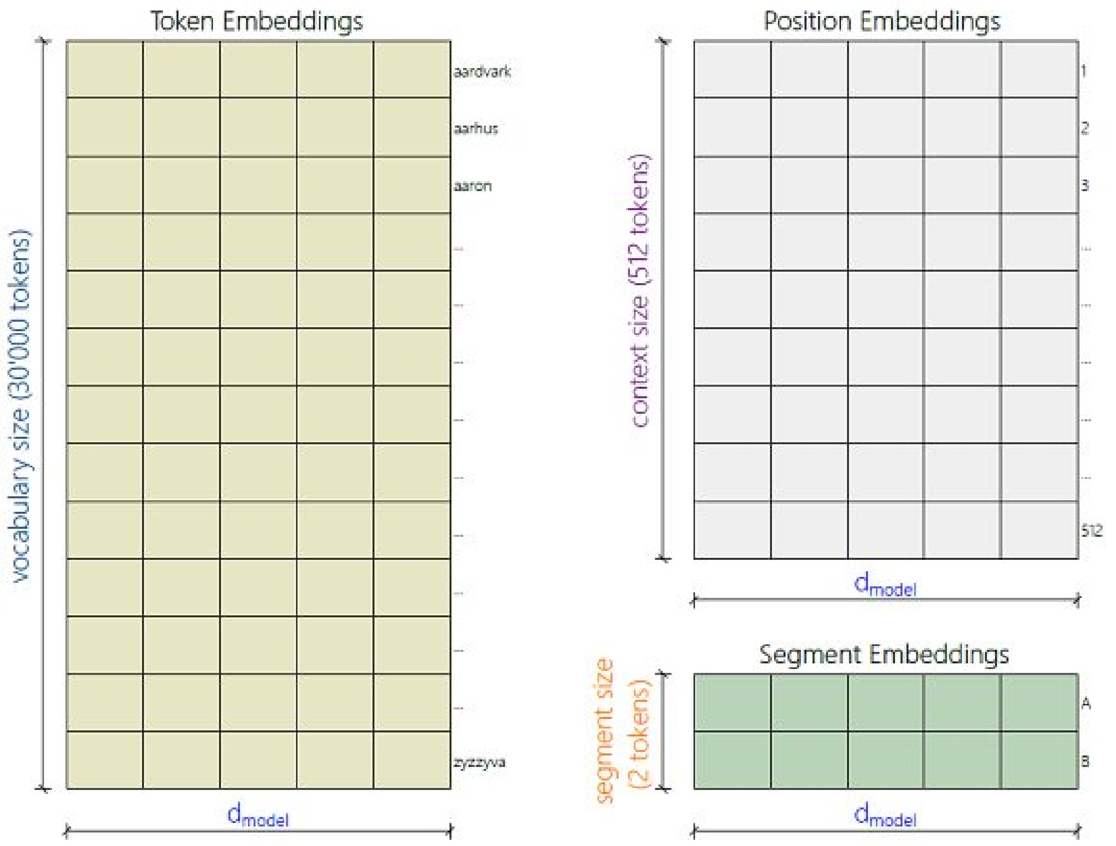


Figure 3.9: Visualization of BERT's parameterized input embeddings.

3.2.3 Model parameters

We have already mentioned that BERT comes in two versions (see §3.2.1), *base* and *large*. They differ in their depths, i.e., the number of Transformer blocks, number of attention heads, and the number of total model parameters. Generally, the large BERT model is able to learn more advanced features and interconnections between the layers, and consistently outperforms the base model. However, due to hardware restrictions, the experiments in this thesis will focus on the BERT-base model. We can summarize its parameters as follows:

- we can decompose the input embeddings into its three constituent parts: the *token*, *segment*, and *positional embeddings* and express them as matrices:
$$\mathbf{W}^{TE} \in \mathbb{R}^{d_{\text{doc}} \times d_{\text{model}}},\ \mathbf{W}^{SE} \in \mathbb{R}^{2 \times d_{\text{model}}},\ \mathbf{W}^{PE} \in \mathbb{R}^{d_{\text{context}} \times d_{\text{model}}}$$
- the self-attention (see §3.1) sublayer specific *key*, *query*, and *value weight matrices* are defined as:
$$\mathbf{W}^{K} \in \mathbb{R}^{d_{\text{model}} \times d_k},\ \mathbf{W}^{Q} \in \mathbb{R}^{d_{\text{model}} \times d_k},\ \mathbf{W}^{V} \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

- the h self-attention sublayer specific *multi-head linear projection triplets* (see §3.1.1), where h is the number of attention heads, are defined as:

$$\left(\mathbf{P}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k},\ \mathbf{P}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k},\ \mathbf{P}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}\right),\ \text{where } i \in [1, h]$$

- the self-attention sublayer specific *multi-head output projection* matrix expressed as:

$$\mathbf{W}^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$$

- the feed-forward sublayer specific network parameters (see §3.2.1), defined as:

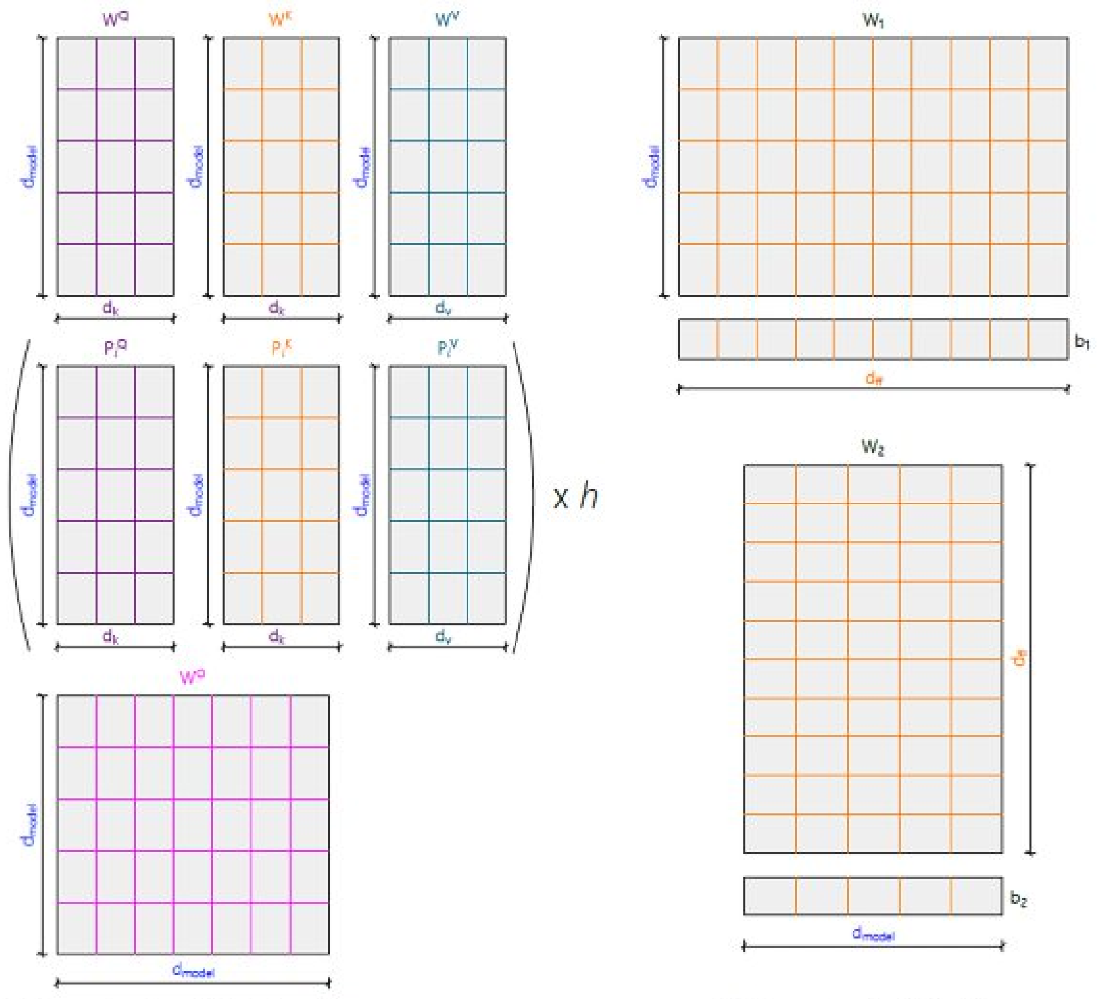
$$\left(\mathbf{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}, \mathbf{b}_1 \in \mathbb{R}^{d_{\text{ff}}}\right), \left(\mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}, \mathbf{b}_2 \in \mathbb{R}^{d_{\text{model}}}\right)$$

- the layer specific residual connection parameters, defined as:

$$\mathbf{W}^R \in \mathbb{R}^{d_{\text{model}}}$$

In this thesis, the following settings will be used for the dimension (of embeddings and matrices) and applied to subsequent BERT models:

$$d_i = \begin{cases} d_{\text{model}} = 768 \\ d_{\text{voc}} = 28'996 \\ d_{\text{context}} = 512 \\ d_{\text{ff}} = 4d_{\text{model}} \\ d_k = d_v = \frac{d_{\text{model}}}{h} \end{cases}$$



(a) Parameters of a self-attention sublayer.

(b) Parameters of a FFN sublayer.

Figure 3.10: Parameter visualization of an encoder layer.

3.3 Pre-training

Devlin et al. (2019) do not use the traditional left-to-right or right-to-left language models to train BERT. They use two unsupervised tasks simultaneously to pretrain BERT: *masked language modeling* (MLM) and *next sentence prediction* (NSP). The reported training loss is the sum of the mean MLM and NSP likelihoods.

3.3.1 Masked Language Modeling

In masked language modeling, the learning objective is to predict the next word given a sequence of previous words. We have mentioned that BERT is a bidirectional language model, which means that is is not solely trained on the left-to-right or right-to-left sequence, but on both simultaneously. However, bidirectional conditioning would allow each word to indirectly "see itself", and the model could trivially predict the target word in a multilayered context. BERT addresses this by randomly masking (replacing a token with the `[MASK]` token) a percentage of input tokens. This procedure is often referred to as a *Cloze* task in the literature (Taylor, 1953). Figure 3.11 shows that the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM.

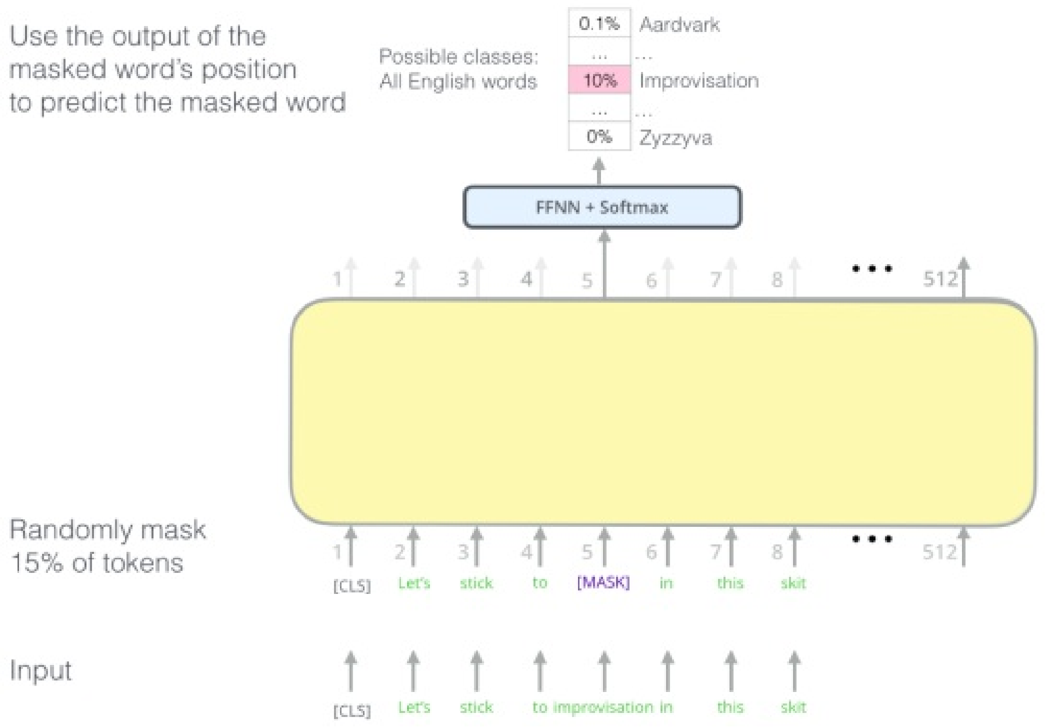

Use the output of the masked word's position to predict the masked word
Possible classes: All English words
0.1% Aardvark
...
10% Improvisation
...
0% Zyzzyva
FFNN + Softmax
1 2 3 4 5 6 7 8 ... 512
Randomly mask 15% of tokens
[CLS] Let's stick to [MASK] in this skit
Input
[CLS] Let's stick to improvisation in this skit

Figure 3.11: Visualization of the masked language modeling (MLM) training objective (Alammar, 2018b).

We can formalize the MLM training process as an input sequence $\mathbf{x} = [x_1, \ldots, x_N]$ of N tokens, randomly selecting a set of $k \in [1, N]$ positions to mask $\mathbf{m} = [m_1, \ldots, m_k]$. Next, the tokens in the selected positions $\mathbf{m}$ are replaced with the `[MASK]` token, yielding a masked input sequence $\mathbf{x}^{\text{masked}}$. BERT learns to predict the original identities of the k masked tokens by computing an output probability distribution $\hat{\mathbf{y}}^{(h)}$, $h \in [1, k]$ for each k. Given the h-th masked word x_{mh} from the sequence $\mathbf{x}$, the cross-entropy loss function is used by calculating the difference between the predicted probability distribution $\hat{\mathbf{y}}^{(h)}$, and the true distribution of the next word $\mathbf{y}^{(h)}$, which is the one-hot encoded vector for

x_{mh}. We can express this loss function as follows:

$$\begin{aligned} \mathcal{L}^{(h)}_{MLM}(\hat{\mathbf{y}}^{(h)}, \mathbf{y}^{(h)}) &= -\sum_{i=1}^{n} y_i^{(h)} \log(\hat{y}_i^{(h)}) \\ &= -\log(\hat{y}^{(h)}_{x_{mh}}) \end{aligned} \tag{3.1}$$

The overall loss of the masked sequence x^{masked} can be calculated by averaging the loss of all k masked tokens:

$$\begin{aligned} \mathcal{L}_{MLM}(\hat{\mathbf{y}}, \mathbf{y}) &= \frac{1}{k} \sum_{h=1}^{k} \mathcal{L}^{(h)}_{MLM}(\hat{\mathbf{y}}^{(h)}, \mathbf{y}^{(h)}) \\ &= \frac{1}{k} \sum_{h=1}^{k} -\log(\hat{y}^{(h)}_{x_{mh}}) \\ &= \frac{1}{k} \sum_{i \in \mathbf{m}} -\log(P(x_i | \mathbf{x}^{\text{masked}})), \end{aligned} \tag{3.2}$$

where P is the probability of masking a word in a sequence. BERT selects 15% of all WordPiece tokens in each training sequence randomly. If the i-th token is chosen, it is replaced with:

$$x_i = \begin{cases} P(x_i | \texttt{[MASK]}) = 0.80 \\ P(x_i | x_j) = 0.10, \text{ where } x_j \text{ is a random token} \\ P(x_i | x_i) = 0.10 \end{cases}$$

The allocation of x_i according to these probabilities prevents the mismatch between pretraining and fine-tuning. If we were to always replace x_i with `[MASK]`, the masked token would *never* be seen before fine-tuning.

3.3.2 Next Sentence Prediction

In *next sentence prediction* (NSP), the model receives pairs of sentences - sentence `A` and `B` - as input and learns to predict if sentence `B` is the subsequent sentence of `A` in the original corpus. This problem can be reduced to a binary classification problem, where the training objective learns the relationship between pairs of sentences. This relationship is especially important for many downstream NLP tasks such as *question answering* (QA) and *natural language inference* (NLI).

The model can be pretrained for the binarized NSP task, which can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences `A` and `B` for each pretraining example, 50% of the time `B` is the actual next sentence that follows `A` (labeled as `isNext`), and in the other 50% of the cases `B` is a random sentence from the corpus (labeled as `notNext`). The final hidden vector that corresponds to the `[CLS]` token is propagated to a softmax function which returns the output probabilities for the two classes (see Figure 3.12).

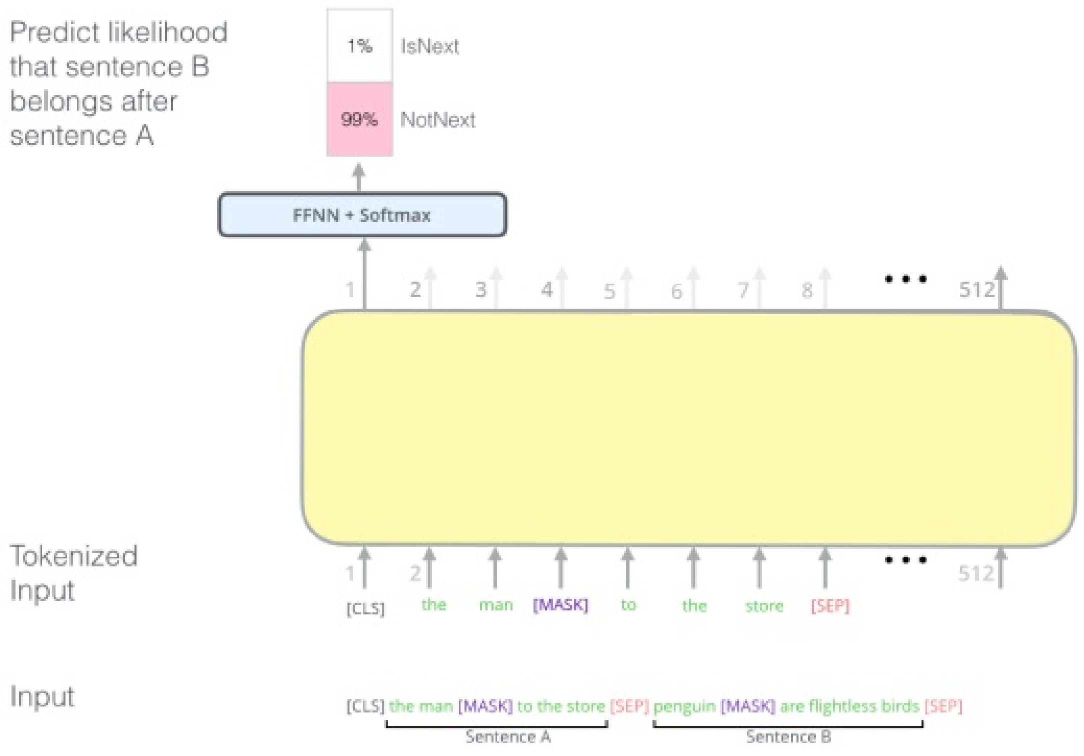

Predict likelihood that sentence B belongs after sentence A
1% IsNext
99% NotNext
FFNN + Softmax
1 2 3 4 5 6 7 8 ... 512
Tokenized Input
[CLS] the man [MASK] to the store [SEP]
Input
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]
Sentence A
Sentence B

Figure 3.12: Visualization of the next sentence prediction (NSP) training objective (Alammar, 2018b).

3.4 Fine-tuning

In practise, the pretraining of a BERT model requires training from scratch, i.e., raw text data, massive datasets, and exhaustive GPU resources, that take days, even weeks, to converge. This pretraining methodology is often referred to as *transductive* transfer (Blitzer et al., 2007) which has become unfeasible for most researchers. Current state-of-the-art NLP research focuses on *inductive* transfer, i.e. the fine-tuning of pretrained word embeddings, e.g. GloVe (Pennington et al., 2014), which utilizes a simple transfer technique that solely targets the models first layer (Howard and Ruder, 2018). The latter approach allows us to exploit transfer learning and consequently outperform the random initialization of the model parameters. Another major benefit is the significant reduction of training time and required resources since we only need to train the first layer of the network.

Google trained the *transductive* BERT model using many GPU's and millions of documents. Subsequently, they published and shared the BERT model (and its learned weights) such that researchers can use, modify, and fine-tune it for specific tasks. This enables the *fine-tuning* approach that introduces minimal task-specific parameters, and is trained on downstream tasks by modifying the pretrained parameters.

The dual pretraining objectives of BERT enable the input to be any single sequence or sequence pair, without requiring major task-specific model architecture modifications. For each task, only the model's first layer must be modified according to the task-specific input and the output according to the expected output. The subsequent training process updates the model parameters according to the selected task by running the end-to-end training process for a specified amount of time e. Figure 3.13 illustrates this fine-tuning approach on different common NLP benchmarks.

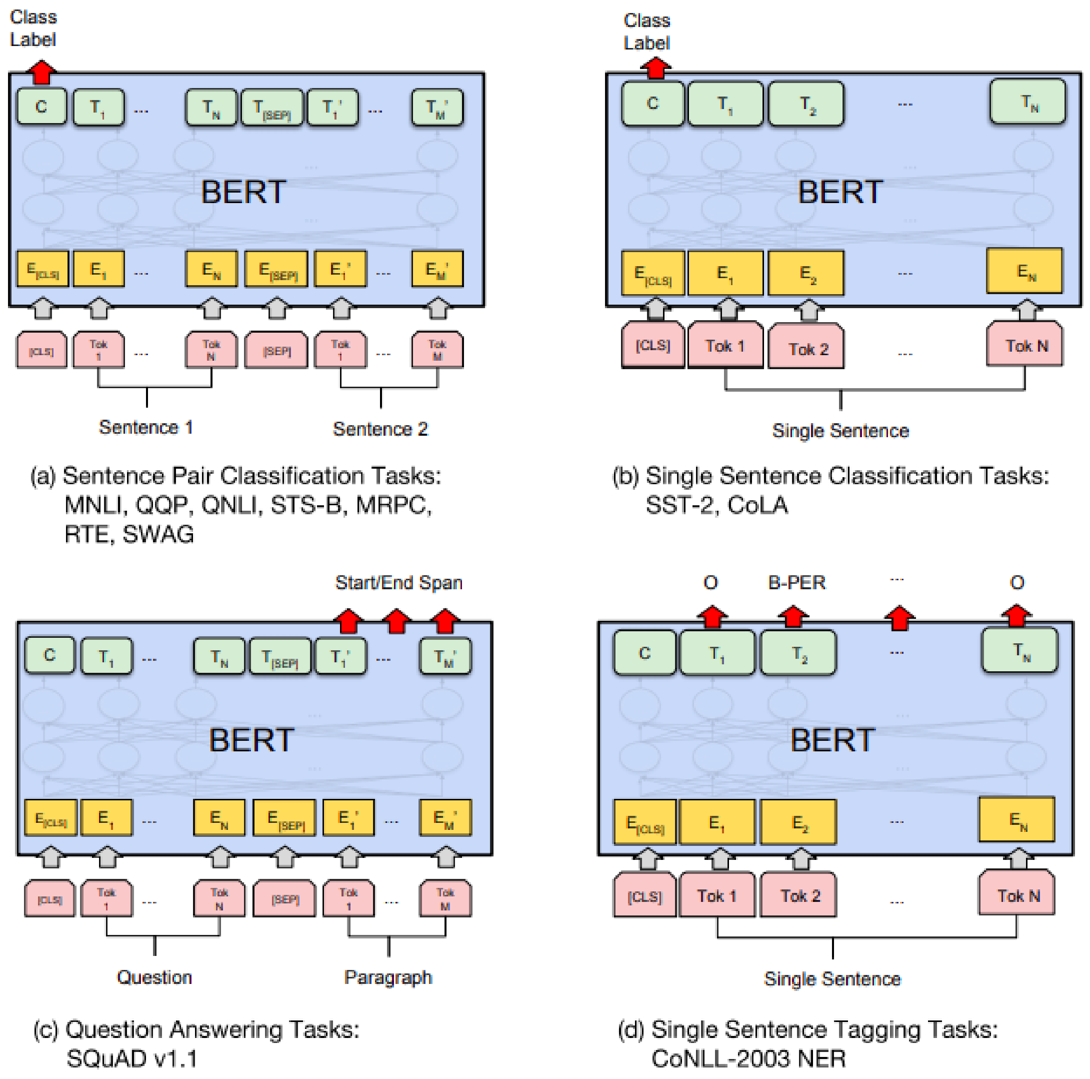

Class Label
BERT
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
BERT
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
BERT
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
BERT
Single Sentence
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure 3.13: Illustrations of the fine-tuning process of BERT on different NLP tasks Devlin et al. (2019)

At the input, sentence A and sentence B from pre-training are equivalent to:

1. sentence pairs in *paraphrasing*
2. hypothesis-premise pairs in *entailment*
3. question-passage pairs in *question answering*
4. a degenerate text-∅ pair in *text classification* or *sequence tagging*

At the output, the token representations are propagated to the output layer for token level tasks (sequence tagging or question answering), and the `[CLS]` representation is propagated to the output layer for text classification.

CHAPTER 4

Experiments

In this section, we introduce the workflow for the classification of suicide-related content in English texts. Specifically, in §4.1 we present the dataset and explain its unique characteristics. In §4.2, we introduce the concept of synthetic data generation as a means to balance the class frequency distributions. In §4.3, we introduce three traditional NLP approaches that are used to represent baseline performances, which are subsequently compared to the performance of deep neural networks that exploit transfer learning. Finally, in §4.4 we investigate the different fine-tuning methods for the pretrained language model BERT on multiple text classification tasks. We have developed a workflow (see Figure 4.1) that shows the steps that are executed to generate a machine learning model for a variable code.

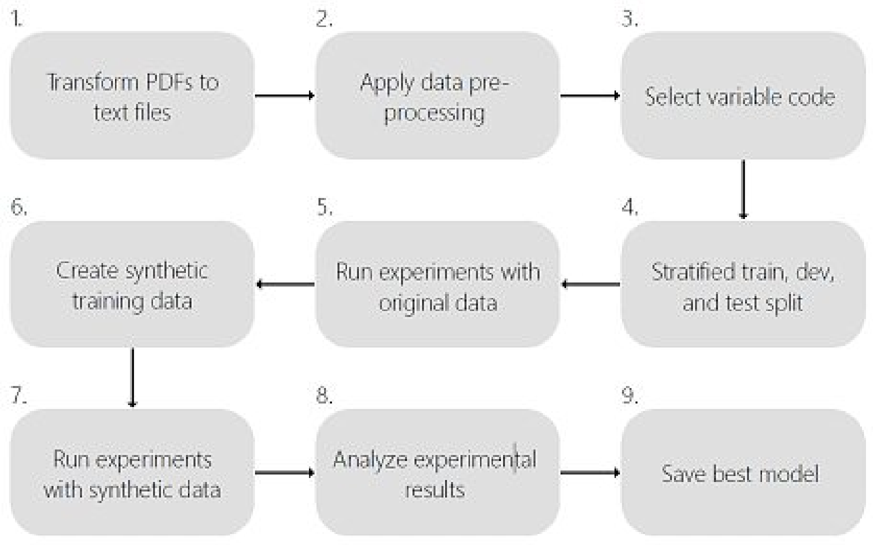

1.
Transform PDFs to text files
2.
Apply data pre-processing
3.
Select variable code
4.
Stratified train, dev, and test split
5.
Run experiments with original data
6.
Create synthetic training data
7.
Run experiments with synthetic data
8.
Analyze experimental results
9.
Save best model

Figure 4.1: Experimental workflow.

4.1 Dataset

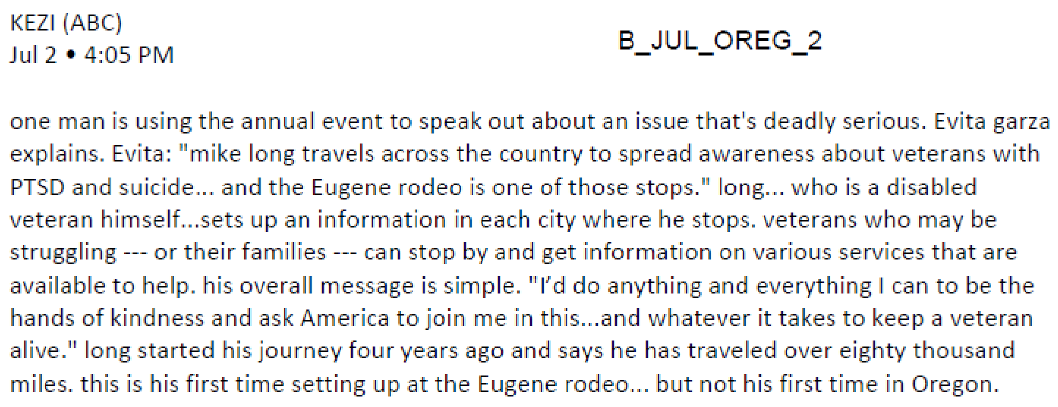
We conducted this thesis as part of the Oregon Media Project, which evaluates the impact of media content released during a suicide prevention campaign. The campaign promotes the adoption of coping strategies and help-seeking in individuals at risk of suicidal behaviour, following best practices as outlined in published recommendations for the reporting on suicide[1] [2]. The human content analysis of media items follows an evaluation system developed by the team of Thomas Niederkrotenthaler. This coding system contains over 150 variable codes (unique indicators describing aspects of suicide reporting, e.g. whether the item mentions alternatives to suicidal behaviour, **AU01_01**). Each variable code represents one characteristic of suicide reporting. Thus, a unique classification task is associated with each variable code.

All data used and all conducted analyses conform to the ethical guidelines of the Declaration of Helsinki and local guidelines by the ethics board of the Medical University of Vienna. We only analyze data that is publicly available, such as media articles or TV and radio broadcasts. All results are presented in an aggregated and anonymized format.

We focused on a dataset that includes transcripts of television and radio broadcasts about suicide published during and after a prevention campaign in Oregon, USA. While the campaign lasted for one month, the available dataset includes media items from eleven additional months. This dataset includes many different variable codes and most of them are binary, such as "AB01 - Does the item report a recent suicide or not?". We determined a set of the most interesting variable codes and developed a unique model for each code.

Figure 4.2 depicts an example of a transcript from the original input PDF files. It contains metadata about the radio/TV channel, the time of broadcast, a unique identifier (ID), and the transcribed broadcast. We converted the PDF files into an open file format (text file) and created a preprocessing pipeline that separates the metadata from the text. This transformation of the data allows to represent one transcribed broadcast as a row with the ID and its associated text.

[1]https://www.who.int/mental_health/suicide-prevention/resource_booklet_2017/en/
[2]https://reportingonsuicide.org/

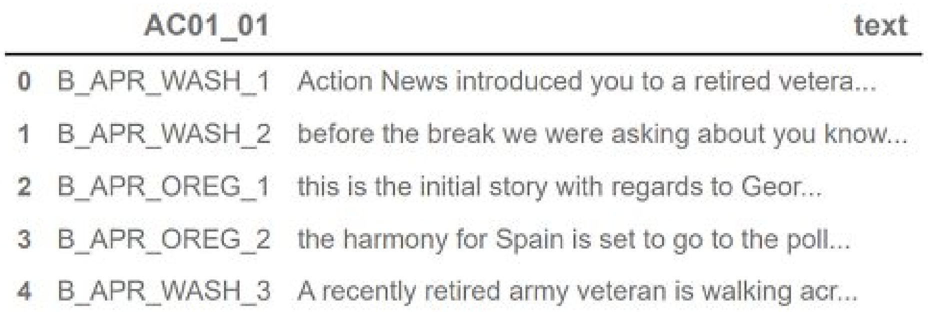

KEZI (ABC)
Jul 2 • 4:05 PM

B_JUL_OREG_2

one man is using the annual event to speak out about an issue that's deadly serious. Evita garza explains. Evita: "mike long travels across the country to spread awareness about veterans with PTSD and suicide... and the Eugene rodeo is one of those stops." long... who is a disabled veteran himself...sets up an information in each city where he stops. veterans who may be struggling --- or their families --- can stop by and get information on various services that are available to help. his overall message is simple. "I'd do anything and everything I can to be the hands of kindness and ask America to join me in this...and whatever it takes to keep a veteran alive." long started his journey four years ago and says he has traveled over eighty thousand miles. this is his first time setting up at the Eugene rodeo... but not his first time in Oregon.

Figure 4.2: Example of a transcribed broadcast.

Figure 4.3 shows the transformed data. Specifically, the variable code **AC01__01** represents the unique ID for every broadcast transcript and the **text** does not contain any metadata. We had access to 3028 transcripts from radio and TV broadcasts during and following the suicide prevention campaign, which were released over a time span of 12 months (April 2019-March 2020).



	AC01_01	text
0	B_APR_WASH_1	Action News introduced you to a retired vetera...
1	B_APR_WASH_2	before the break we were asking about you know...
2	B_APR_OREG_1	this is the initial story with regards to Geor...
3	B_APR_OREG_2	the harmony for Spain is set to go to the poll...
4	B_APR_WASH_3	A recently retired army veteran is walking acr...

Figure 4.3: Head of the transformed input data.

Next, we determined the length of the transcripts. This plays a crucial role since many state-of-the-art language models are limited to a sequence length of 512 tokens. Consequently, longer texts will be cut off after the 512'th token, resulting in information loss. We compensated this loss by training the traditional NLP models (see §4.3) on the full sequences and compare their respective performances. Figure 4.4 depicts the sequence length histogram. We included three vertical lines: (1) the black line represents the mean sequence length of all transcripts (316); (2) the red line depicts the 95'th percentile (944) and (3) the green line the 99'th percentile (1'411). We conclude that there are some broadcasts that are significantly longer than the average broadcast, however, the majority of broadcasts are rather short, i.e., less than 500 words.

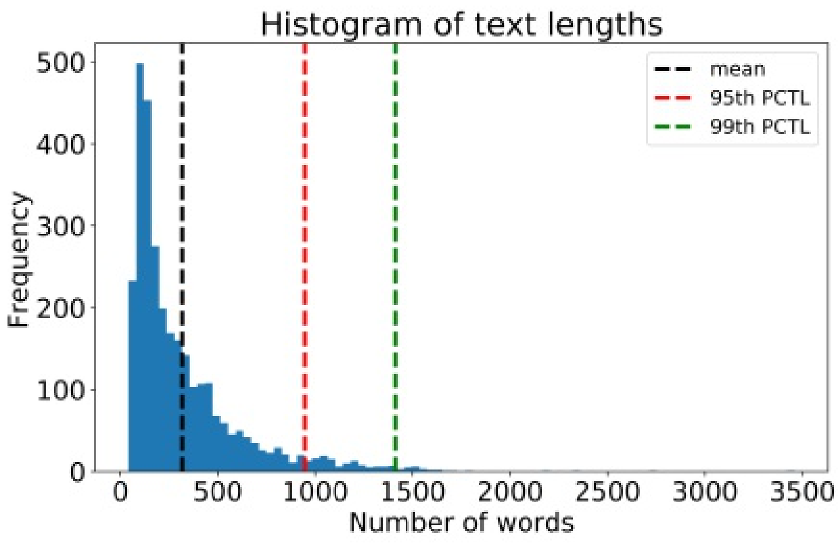
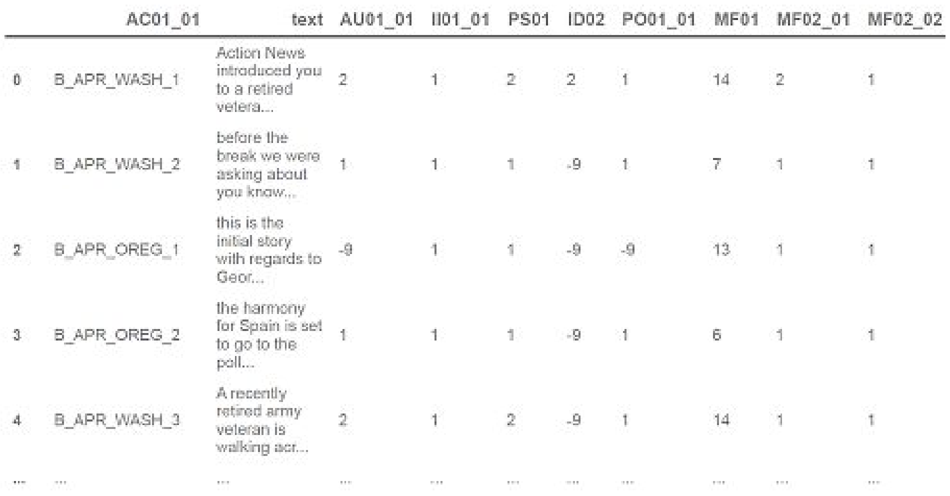


Figure 4.4: Distribution of text length (number of words) in the transcripts.

Subsequently, we joined the transformed dataset with the manually labelled variable codes from an Excel file. This representation allows us to quickly and easily select a variable code we want to investigate and pass it into the classification workflow. We note that there are around 150 variable codes, each representing a unique classification task. Thus, we developed a semi-automated pipeline that requires minimal user input and returns either a newly trained model, or a prediction for the selected variable code. Figure 4.5 shows the joined dataset and a selection of variable codes.



	AC01_01	text	AU01_01	II01_01	PS01	ID02	PO01_01	MF01	MF02_01	MF02_02
0	B_APR_WASH_1	Action News introduced you to a retired vetera...	2	1	2	2	1	14	2	1
1	B_APR_WASH_2	before the break we were asking about you know...	1	1	1	-9	1	7	1	1
2	B_APR_OREG_1	this is the initial story with regards to Geor...	-9	1	1	-9	-9	13	1	1
3	B_APR_OREG_2	the harmony for Spain is set to go to the poll...	1	1	1	-9	1	6	1	1
4	B_APR_WASH_3	A recently retired army veteran is walking acr...	2	1	2	-9	1	14	1	1
...	...	...	...	...	...	...	...	...	...	...

Figure 4.5: Head of the joined dataset.

After inspecting the class distributions of some variable codes, we found that many suffer from severe class imbalances. This poses an issue especially for deep language models which maximize the classification accuracy during training rather than the F_1-score. If the data is too heavily skewed towards one class, the model will become a majority classifier and never predict the minority class. We can visually confirm this by looking at Figure 4.6 where the imbalance becomes clearly apparent. This skewed distribution is consistent across most of the variable codes, with most binary classification tasks containing a majority of *no* or *irrelevant* classes. However, finding the infrequent positive instances is crucial for our research, because these are the characteristics of interest for suicide prevention research.

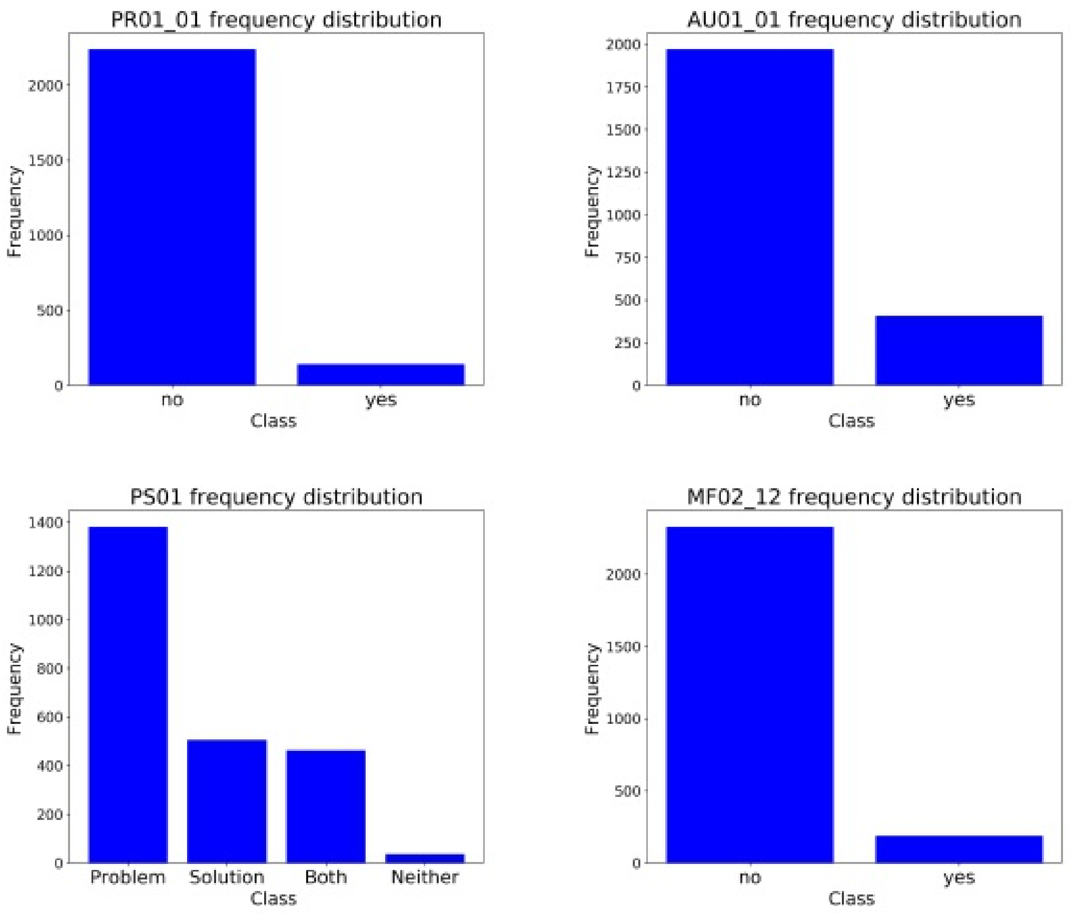


Figure 4.6: Frequency distributions of a subset of variable codes. The different variables will be described in detail further below.

The importance of class balance becomes even more apparent when inspecting Figure 4.7. Our methodology requires us to split the data into three sets which must contain at least one observation from each class (the training, validation and test set). Some classes only contain a few dozen positive observations which leaves the model with very little information to learn from. We address this issue by presenting our data balancing strategies in §4.2.

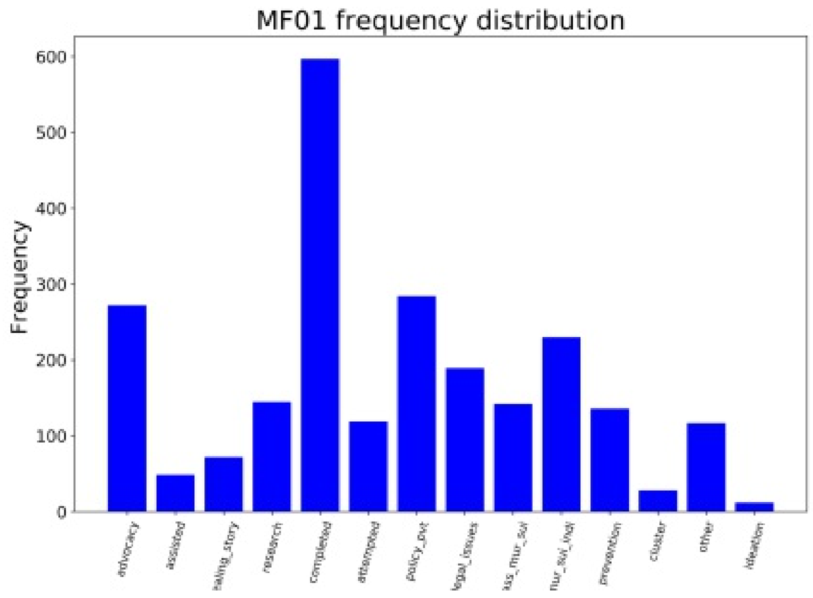


Figure 4.7: MF01 - main focus - frequency distribution. The main topic of the text. Specific labels will be further described below.

In order to compute a global score for each of our chosen performance metrics, i.e., precision, recall, F_1-score, and accuracy, in our multi-class classification tasks, we implemented *macro-averaging*. Macro-averaging returns the arithmetic mean of the per-class metrics:

$$x_M = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

where N is the number of classes in the classification problem. This ensures that each class is given an equal weight, irrespective of class imbalances. In our experiments we always report the macro-averaged results.

Finally, we ensure that the class frequency distribution is consistent across the three independent data subsets. We apply sklearn's `train_test_split()`[3] to create stratified train, validation, and test sets with 64%, 16%, and 20 % of the data respectively.

A part of the observations (between 20 and 50 per variable code) was annotated manually by two human coders, which provides estimates of inter-coder agreement which we

[3]sklearn.model_selection.train_test_split()

compare against the accuracy of the models. We calculate the F_1 score from their data and use it as a proxy for the human F_1-score performance on the test set. We use these as benchmarks for the models' performance for the available variable codes.

4.2 Synthetic data generation

We observed severe class imbalances in some of the variable codes we wanted to investigate. This leads to good overall performance in respect to classification accuracy on both the training and validation sets - the higher the imbalance, the higher the classification accuracy - but poor performance in classifying the minority class, which is usually the relevant class. The model is not incentivized to predict the minority class since it achieves better accuracy by always predicting the majority class.

We tested different methods to address this issue. The most common strategy to fix class imbalances is to *oversample* from the minority classes until the class frequency distribution is equal, i.e., an observation from the minority class can occur more than once in the training data. In text classification, oversampling proves difficult in practise, because the same document of the minority class is sampled with replacement, which increases the bias towards the training data, and consequently, increases the variance in the test data. Additionally, the model does not learn new features, but learns to rely on specific features that occur often in duplicate documents (Drummond and Holte, 2003). A common alternative, is to *undersample* from the majority class. In undersampling, a random sample of the size of the minority class is drawn to balance the classes.

However, this reduces the number of documents that are used to train the model, and in cases where only a few observations are present for the minority class, the model learns on an extremely small sample and performs poorly.

We investigated three alternative strategies, beginning with Synthetic Minority Oversampling TEchnique (SMOTE) (Chawla et al., 2002) which is a very popular oversampling method proposed to improve random oversampling.

Additionally, we developed and tested two novel synthetic data generation techniques, (1) *back translation*, and (2) *synonym replacement*, that both focus on the transformation of the input text to generate additional samples, effectively balancing the dataset. The sections below describe all three strategies in more detail.

4.2.1 SMOTE

With Synthetic Minority Oversampling TEchnique (SMOTE), the new instances are not just copies of existing minority cases; instead, the algorithm takes samples of the feature space for each target class and its nearest neighbours, and generates new examples that combine features of the target case with features of its neighbors. This approach increases the features available to each class and makes the samples more general.

A hyperplane is created over the features of the minority class and the iteration starts by first selecting an instance at random. Next, the k nearest neighbours, measured in Euclidean distance, are obtained for that instance. At last, n of these k instances are chosen to interpolate new synthetic instances. Due to the interpolation, the new synthetic instances lie on the hyperplane as depicted in Figure 4.8.

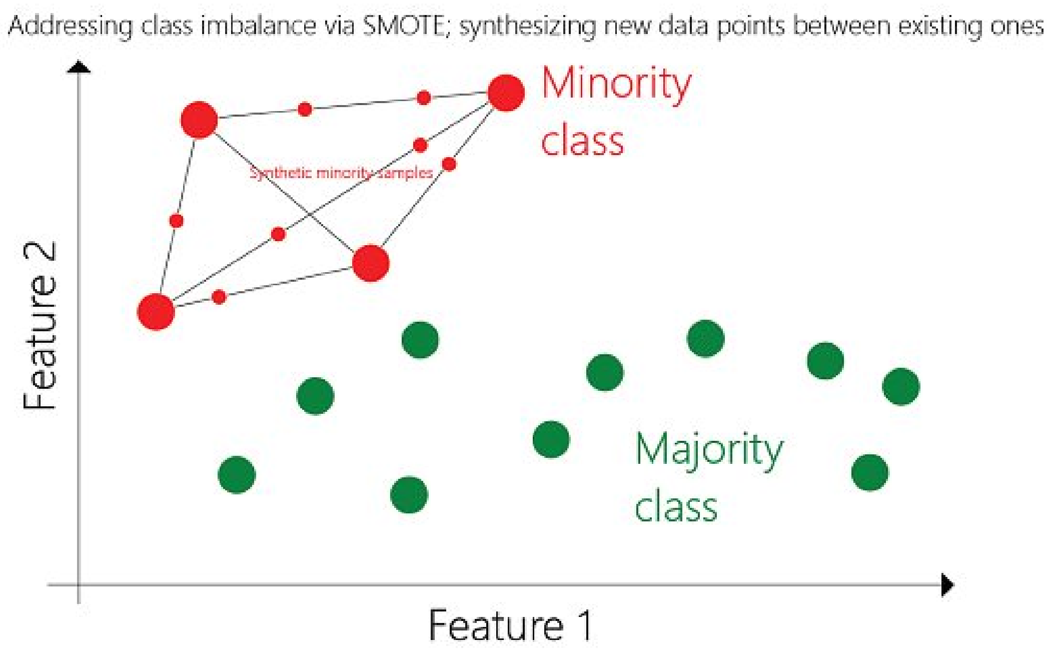

Addressing class imbalance via SMOTE; synthesizing new data points between existing ones
Minority class
Synthetic minority samples
Feature 2
Majority class
Feature 1

Figure 4.8: Synthesizing new data points by balancing via SMOTE

However, experiments using low-dimensional data showed that simple undersampling tends to outperform SMOTE in most situations (Van Hulse et al., 2007). In text classification, each word represents a feature, therefore we analyze the impact of SMOTE on high dimensional data. In practice, in the high-dimensional setting only k-NN classifiers based on the Euclidean distance seem to benefit substantially from the use of SMOTE, provided that variable selection is performed before using SMOTE; the benefit is larger if more neighbors are used. SMOTE for k-NN without variable selection should not be used, because it strongly biases the classification towards the minority class (Blagus and Lusa, 2013). In our experiments, we use SMOTE for k-NN with the Euclidean distance and test 5, 7, and 10 nearest neighbours in the hyperparamter search.

4.2.2 Back translation

In back translation, we translate the training text data into some language and then translate it back to the original language. This method preserves the context of the text but helps generating additional data by using different words in the translation process.

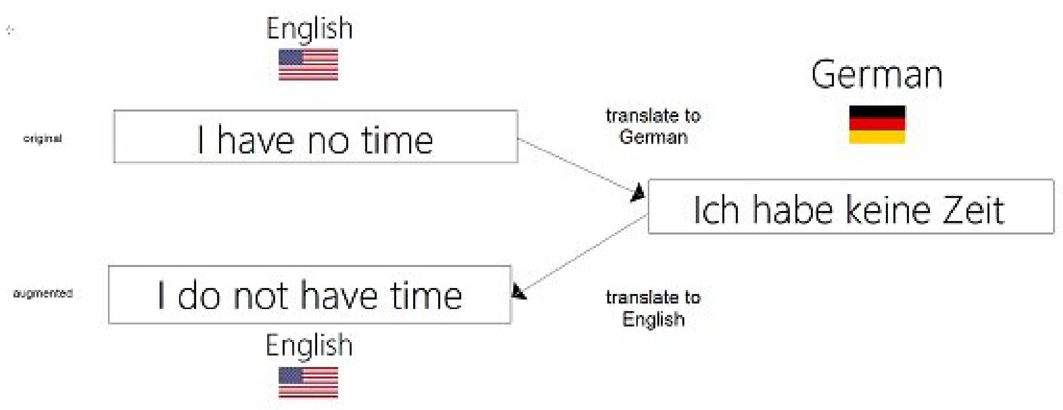


Figure 4.9: Visualization of the back translation process.

Intuitively, language models built using synthetic data should not perform well. A text translated by a machine can contain errors, so a model trained on such data may learn and replicate these mistakes. Yet, Sennrich et al. (2016) demonstrated that using back-translated data (in combination with human-translated data) during training can have a positive impact on the performance of the model.

Poncelas et al. (2018) analyzed hybrid neural language models built by incorporating back-translated data into human-translated data, and showed that while translation performance tends to improve when larger amounts of synthetic data are added, performance appears to tail off when the balance is tipped too far in favour of the synthetic data. Furthermore, their experiments only translated between two languages, specifically English and German, and only chose the number of back-translated samples as a hyper-parameter. In line with these findings, we hypothesize that the performance of the model will degrade if the synthetic data is overly dominant in the training set, i.e. the benefit of using high-quality authentic data may be outweighed by the synthetic back-translated data.

We had to create many additional samples because of the severe class imbalances. Therefore, we decided to randomly select a language, translate the entire training data labelled with the minority class into the randomly chosen language, and retranslate it into English. This process is repeated with another unique language until the classes are perfectly balanced. Next, we tested how varying the percentage of translated data affected model performance, using percentages $\in [0.1, 1]$, step$= 0.1$.

4.2.3 Synonyms

The main objective behind the back translation approach is to introduce additional words in the training corpus while preserving the semantic context and simultaneously balancing the dataset. For the same purpose, we also used *synonyms*, replacing words with a synonym with a certain probability P_s while sampling with replacement from the minority class. The most common NLP tool that works with synonyms is WordNet.

WordNet (Miller, 1995) is a large lexical database for English words where nouns, verbs, adjectives, and adverbs are grouped into sets of cognitive synonyms (synsets) that express a distinct concept. We utilize the WordNet database to replace P_s words within a minority class sample:

Original: The quick brown fox jumps over the lazy dog .
Augmented text: The speedy brown fox jumps complete the lazy dog .

An important caveat to WordNet is that it is a thesaurus, i.e., a dictionary that focuses on semantic relationships between words, and is not intended to be used as a lexical resource for the synonym replacement we deploy. WordNet is built of synsets, not of words, and a word goes into a synset if it is a synonym of a word. All words in the same synset are treated synonymously independent of the part of the speech (noun, verb, adjective, adverb) the original word belongs to.

An alternative to the WordNet synonym lookup is the Paraphrase Database (PPDB) (Pavlick et al., 2015), which is an automatically extracted database containing millions of paraphrases in 16 different languages. The goal of PPBD is to improve language processing by making systems more robust to language variability and unseen words. We exploit this variability to generate more expressive augmented training data. For any given input phrase to PPDB, there are often dozens or hundreds of possible paraphrases, this means, even if we replicate one observation multiple times with the same probability P_s to replace a word, it is very unlikely that an augmented observation will have a duplicate in the final, balanced dataset.

Original: The quick brown fox jumps over the lazy dog .
Augmented text: The quick brown fox climbs over the lazy dog .

In our experiments we use $P_s = [0.1, 1]$, step $= 0.1$, as the hyperparameter and test the impact of different values for the replacement probability of each word in the training sequence. In §5 we present the results obtained with PPDB because it consistently outperformed WordNet.

4.3 Traditional NLP

We used three traditional NLP models and used their performance as the baseline to compare with our deep language models. We used the *naive classifier* (see 2.2.3) as a trivial model, and *Bag-of-Words* (see §2.2.5) and *tf-idf* (see §2.2.6) to create a simple numerical representation of the textual input. We tested different hyperparameters such as the inclusion or removal of stopwords and punctuation and the impact of unigrams, bigrams, and trigrams respectively. Additionally, we tested the performance if we reduce the text to its n top features ordered by term frequency, where $n \in \{10'000, 25'000, 50'000, None\}$.

Finally, both BOW and tf-idf, require a machine learning model to be trained on the transformed input representation. For this purpose, we used a Support Vector Machine because it is very efficient regarding computation, inference and performance. For a detailed explanation on SVM's we refer the reader to Suykens and Vandewalle (1999). We tested different hyperparameters, namely, the applied penalty, either L_1 or L_2, the regularization parameter $C \in (0, 1]$ that determines the strength of the regularization, and the class weight $\mathrm{cw} \in \{\text{balanced}, \text{none}\}$, which determines if the weights of the classes are automatically adjusted inversely proportional to class frequencies.

4.4 BERT

We used the pretrained uncased BERT-base model [4] due to the practical applicability of pretrained models for researchers. All models were fine-tuned on domain-specific text classification tasks in association with the corresponding variable code and closely compared with the performance of the traditional NLP models. Additionally, the models were trained on the synthetically balanced data.

We build upon the classification fine-tuning method used by Liu et al. (2019),by adding one dense layer to reduce the dimensions of the model's last layer to the number of labels in the classification task, and fine-tuning the model on each classification task (variable code), training all parameters simultaneously. Furthermore, we had to consider two factors: (1) that the maximum sequence length of BERT is 512 tokens and the input text must be processed accordingly, and (2) the overfitting problem. The longer a model is trained, the better it optimizes the loss on the training data but its ability to generalize deteriorates. Therefore, we aimed for an optimizer with an appropriate learning rate that generalizes well to the test data.

When fine-tuning BERT for a specific task, there are many ways to utilize BERT, e.g., the different layers of BERT capture different levels of semantic and syntactic information and intuitively, the lower layer may contain more general information. Therefore, we fine-tuned the layers with different decaying learning rates and followed the approach of Howard and Ruder (2018) by splitting the parameters θ into $\{\theta^1, \ldots, \theta^L\}$, where θ^l contains the parameters of the l-th layer of BERT. The parameters were updated as follows:

$$\theta_t^l = \theta_{t-1}^l - \eta^l \cdot \nabla_{\theta^l} J(\theta),$$

where η^l represents the learning rate of the l-th layer. We started with the base learning rate η^L and use $\eta^{k-1} = \xi \cdot \eta^k$, where $\xi \leq 1$ is a decay factor, and k the current iteration of the training process. When $\xi < 1$, the lower layer has a lower learning rate than the higher layer. When $\xi = 1$, all layers have the same learning rate, which is equivalent to stochastic gradient descent (SGD).

To determine generally good base learning rates, we ran a hyperparameter search and visually analyzed the associated plot (see Figure 4.10) to create a shortlist of candidate

[4] https://huggingface.co/bert-base-uncased

learning rates. Figure 4.10 shows the loss associated with different learning rates. We tried to find the maximal learning rate that is associated with a still-falling loss (prior to the loss diverging). The plot shows that learning rates up to 10e-5 are still associated with a falling loss. Lower learning rates do not impact the loss since the update steps become too small. Contrarily, once the learning rate reaches ≥ 0.01 the loss explodes. This is caused by too large updates of the weights matrix.

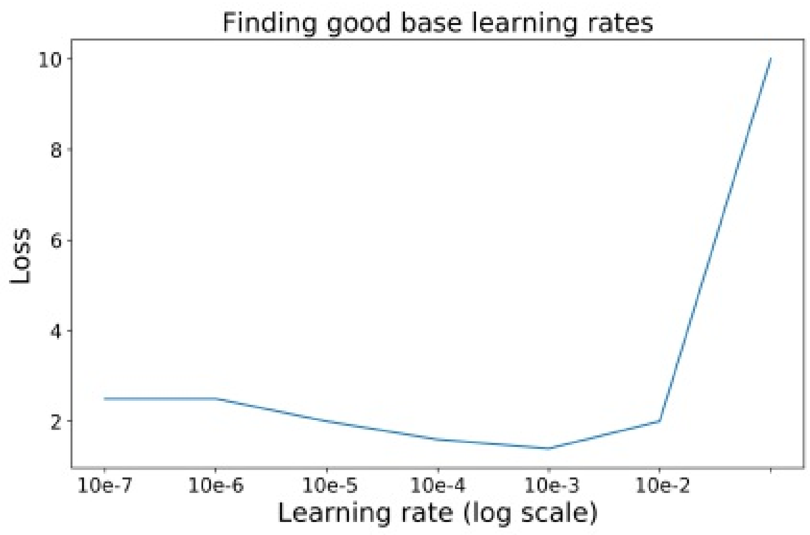


Figure 4.10: Finding a good base learning rate for the BERT model.

CHAPTER 5

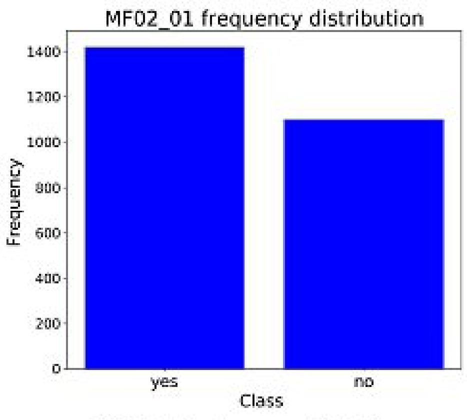
Results

In this section, we present the results for a selection of variable codes that are associated with the four different levels of methodological challenges. Each variable code represents a unique classification task implemented according to the experimental workflow depicted in Figure 4.1. This section first describes the simplest classification tasks and then illustrates other tasks with increasing levels of difficulty.

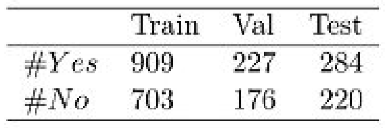
5.1 Simple binary classification

5.1.1 MF02_01 - completed suicide

This variable code determines if the text focuses on completed suicides. In order to qualify as a positive instance, one sentence in an average length text should mention a completed suicide, or the framing of the text suggests that the context is related to a completed suicide.

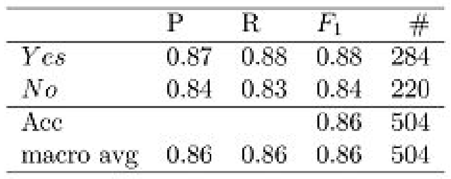


MF02_01 class frequency distributions.



	Train	Val	Test
$\#Yes$	909	227	284
$\#No$	703	176	220

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Yes	0.87	0.88	0.88	284
No	0.84	0.83	0.84	220
Acc			0.86	504
macro avg	0.86	0.86	0.86	504

Test performance metrics of the best model (unbalanced BERT).

Figure 5.1: Experimental results for MF02_01.

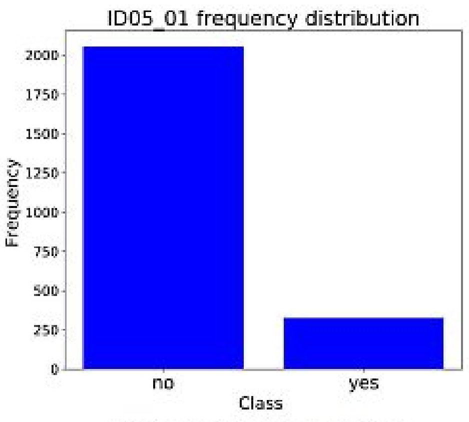
System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							1.0	1.0
Naive	0.282	0.5	0.360	0.563	0.281	0.5	0.360	0.563
BoW	0.814	0.813	0.813	0.816	0.782	0.783	0.783	0.786
tf-idf	0.835	0.826	0.829	0.834	0.796	0.783	0.786	0.794
BERT	0.835	**0.832**	**0.834**	**0.841**	**0.857**	**0.857**	**0.857**	**0.859**
SMOTE								
tf-idf	0.841	0.826	0.830	0.836	0.782	0.772	0.775	0.782
Backtranslation								
tf-idf	0.826	0.807	0.811	0.819	0.792	0.771	0.776	0.786
BERT	0.830	0.816	0.820	0.826	0.805	0.794	0.797	0.804
Synonyms								
tf-idf	0.834	0.810	0.816	0.824	0.798	0.769	0.774	0.786
BERT	**0.843**	0.829	0.833	0.839	0.816	0.797	0.801	0.810

Table 5.1: MF02_01 - completed suicide - results, the best performance for each column is marked in **bold**. The best model, underlined and marked in **bold**, is chosen by selecting the model with the highest validation F_1.

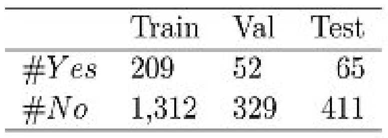
The majority class of this variable code is the positive class, i.e., the one that we are interested in. Therefore the synthetic data generation techniques are applied for the irrelevant class, in contrast to all following classification tasks. Although this seems counter-intuitive, we included it in the analysis to follow the complete workflow as for all other classification tasks. We observe good performance with the original data and the traditional NLP approaches. Only the fine-tuned BERT model slightly outperforms them on the validation set. However, the major difference is that the BERT models generalize better to the test set. This behaviour is consistent across all balancing strategies for this variable code. Finally, we observe that balancing the irrelevant classes only outperforms the best model, *original data - BERT*, on one performance metric, the validation precision. In all other performance metrics, both in the validation and test set, the best model outperforms the remaining models.

5.1.2 ID05_01 - suicidal behaviour or ideation of a celebrity

This variable code reports on suicidal behaviour or suicidal ideation of a celebrity and applies to anyone who was famous prior to their suicidal ideation or suicidal behaviour, but not because of it, i.e., Robin Williams and Avicii, but not Amanda Todd. Furthermore, if the text contains indicators of celebrity status in the text, e.g., someone who runs a TV show, the text is coded as a positive class.

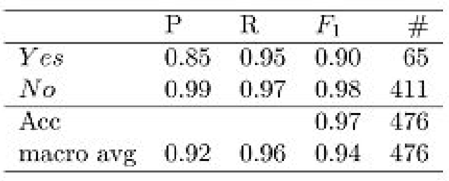


ID05_01 class frequency distributions.



	Train	Val	Test
$\#Yes$	209	52	65
$\#No$	1,312	329	411

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Yes	0.85	0.95	0.90	65
No	0.99	0.97	0.98	411
Acc			0.97	476
macro avg	0.92	0.96	0.94	476

Test performance metrics of the best model (unbalanced BERT).

Figure 5.2: Experimental results for ID05_01.

System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							1.0	1.0
Naive	0.432	0.5	0.463	0.864	0.432	0.5	0.463	0.863
BoW	0.890	0.871	0.880	0.945	0.968	0.905	0.933	0.971
tf-idf	0.932	0.896	0.913	0.961	0.974	0.891	0.927	0.968
BERT	0.938	**0.98**	**0.958**	**0.979**	0.921	**0.964**	0.941	0.971
SMOTE								
tf-idf	0.857	0.784	0.815	0.921	0.927	0.840	0.876	0.947
Backtranslation								
tf-idf	0.943	0.861	0.896	0.971	0.876	0.916	0.896	0.964
BERT	0.944	0.907	0.925	0.966	0.962	0.919	0.939	0.973
Synonyms								
tf-idf	0.951	0.777	0.836	0.937	0.967	0.777	0.839	0.939
BERT	**0.962**	0.901	0.928	0.969	**0.980**	0.930	**0.953**	**0.979**

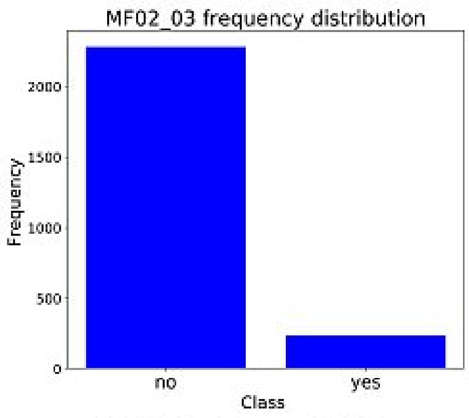
Table 5.2: ID05_01 - suicidal ideation or behaviour of celebrity - results.

The class distribution of this variable code is heavily skewed towards the irrelevant class (see left hand side of Figure 5.2). The model focuses on celebrities, which can be detected easily via a few keywords (celebrity names). We confirm this by inspecting the results of BoW and tf-idf for the unbalanced data. They perform extremely well achieving 0.88 and 0.913 validation F_1. We see that the unbalanced BERT model achieves the best validation $F_1 = 0.958$ and $F_1 = 0.941$; $Acc = 0.971$ on the test set. However, we observe that the synthetic data generation techniques we deploy sacrifice some validation performance for better test performance. Table 5.2 shows that the synonym balancing achieves even better test performance due to the additional random keywords introduced to the training data. This suggests that carefully balancing the minority class improves the model's capacity to generalize to new data.

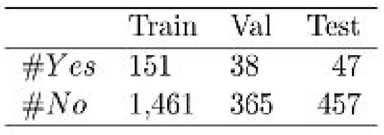
5.2 Intermediate binary classification

5.2.1 MF02_03 - suicidal ideation

This variable code determines if the focus area of the text lies on suicidal ideation that is not accompanied by suicide or a suicide attempt. Aborted suicide attempts also qualify as suicidal ideation. In order to qualify as a positive instance one sentence in an average length text should mention suicidal ideation or the framing of the text suggests that the context is related to suicidal ideation.

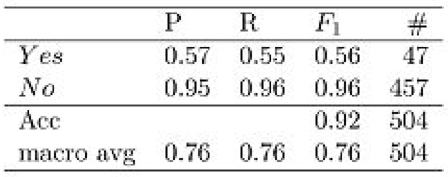


MF02_03 class frequency distributions.



	Train	Val	Test
$\#Yes$	151	38	47
$\#No$	1,461	365	457

Sample distribution for train, validation, and test sets.

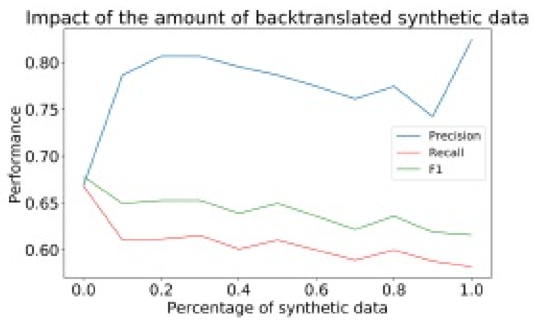


	P	R	F_1	#
Yes	0.57	0.55	0.56	47
No	0.95	0.96	0.96	457
Acc			0.92	504
macro avg	0.76	0.76	0.76	504

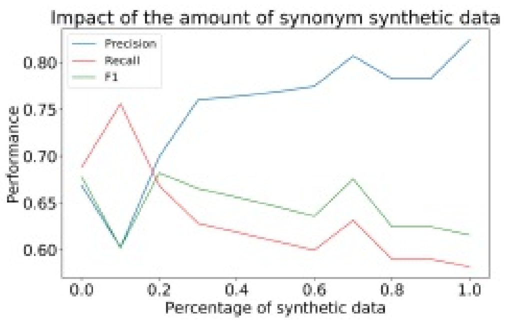
Test performance metrics of the best model (unbalanced BERT).

Figure 5.3: Experimental results for MF02_03.

To determine the effectiveness of the synthetic data generation techniques we fixed a set of hyperparamters for the tf-idf model and trained the same model with different inputs. Specifically, we balanced the training set, with both back translation and synonym replacement, such that the classes were perfectly balanced. We refer to this as *100% synthetic data*, which yields 1,461 *yes* instances, compared to the original 151. Subsequently, we trained the models and documented the performance on the test set with different amounts of synthetic data. The synthetic data percentage p_{syn} was chosen from $[0, 1]$ in 0.1 steps, where $p_{syn} = 0$ is the original data with 151 positive instances and $p_{syn} = 1$ the perfectly balanced data with 1,461 positive instances. The synthetic data was randomly sampled based on p_{syn} with a fixed seed to ensure the same observations from $p_{syn} = 0.1$ are present in the sample of $p_{syn} = 0.2$, and $p_{syn} = 0.3$, and so on.



(a) Backtranslated synthetic data.



(b) Synonym synthetic data.

Figure 5.4: tf-idf test performance for different amounts of synthetic data.

In Figure 5.4a we can see that the more synthetic data we incorporate, the worse the recall and F_1 score become. This behaviour is consistent for both synthetic data generation techniques, but we see an outlier at $p_{syn} = 0.7$ for the synonym approach (see Figure 5.4b). However, the precision increases significantly, and we see that between $p_{syn} = 0.1$ and $p_{syn} = 0.3$ we can efficiently trade recall for precision. This trade-off loses some of the overall fitness of the model due to the decreasing F_1 score, but we can trade a loss of 5% recall for a 15% gain in precision, only sacrificing 3% F_1 score compared to the best, i.e. $p_{syn} = 0$ model. The results show that including too much synthetic data biases the model towards the training samples and it learns to rely on specific features that indicate the positive class in the training set. However, those features are not always present, therefore, the test performance decreases continuously with the incorporation of additional synthetic data. Consequently, we modify our synthetic data balancing strategy accordingly and introduce a cutoff at $p_{syn} = 0.3$ for all subsequent results.

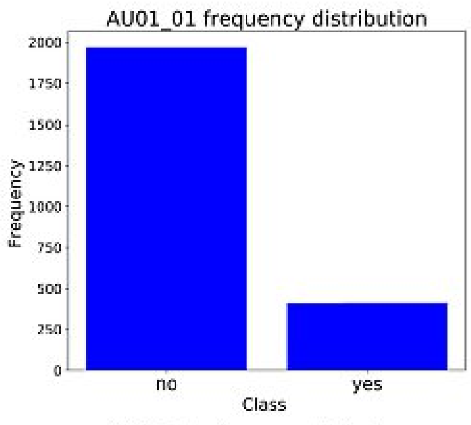
System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							0.75	0.875
Naive	0.452	0.5	0.475	0.906	0.453	0.5	0.476	0.907
BoW	0.696	0.673	0.683	0.898	0.701	0.678	0.689	0.901
tf-idf	0.689	0.716	0.701	0.891	0.669	0.688	0.678	0.885
BERT	0.766	**0.791**	**0.778**	0.921	0.760	**0.755**	**0.757**	**0.919**
				SMOTE				
tf-idf	**0.913**	0.617	0.668	0.926	0.778	0.570	0.598	0.913
				Backtranslation				
tf-idf	0.784	0.612	0.651	0.916	0.738	0.607	0.640	0.911
BERT	0.769	0.609	0.646	0.915	0.776	0.623	0.663	0.916
				Synonyms				
tf-idf	**0.913**	0.617	0.668	0.926	**0.824**	0.582	0.616	0.917
BERT	0.819	0.701	0.743	**0.928**	0.744	0.646	0.679	0.913

Table 5.3: MF02_03 focus on suicidal ideation - results.

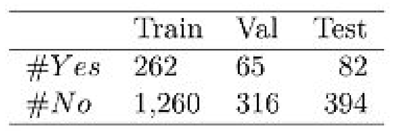
This variable code suffers from class imbalance, however, the unbalanced BERT model is able to learn the most important features better than any other model we tested. We were able to train this BERT model with a lower learning rate (1e-5) for a longer time (8 to 12 epochs) without overfitting too strongly, even though the hyperparameter search for the learning rate recommends a higher learning rate (see Figure 4.10). The synthetically balanced models were able to minimize the training error much more quickly (~ 5 epochs) and tended to overfit too much after that. From this, we conclude that the more epochs BERT can be trained without overfitting, the better its ability to generalize becomes, even if there are only relatively few samples. The data balancing strategies we developed performed poorly for this variable code, implying that the semantic structure and the word choice play a crucial role in determining the positive instances.

5.2.2 AU01_01 - alternatives to suicidal behaviour

This variable code reports on alternatives to suicidal behaviour, where *any* alternative counts towards the positive class. This might include a specific action taken by an individual instead of suicidal behaviour, e.g., a suggestion or advice to seek help; "going for a walk to calm down"; "how to make new friends".

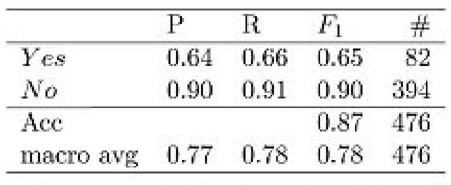


AU01_01 class frequency distributions.



	Train	Val	Test
$\#Yes$	262	65	82
$\#No$	1,260	316	394

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Yes	0.64	0.66	0.65	82
No	0.90	0.91	0.90	394
Acc			0.87	476
macro avg	0.77	0.78	0.78	476

Test performance metrics of the best model (unbalanced tf-idf).

Figure 5.5: Experimental results for AU01_01.

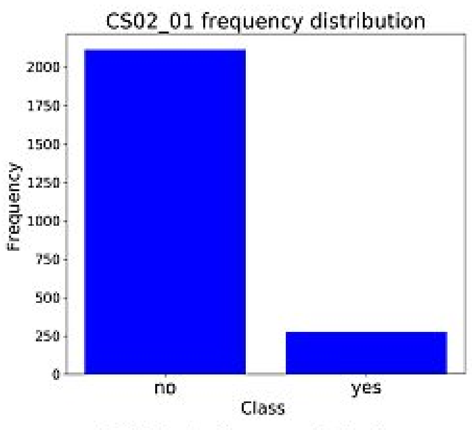
System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							0.7	0.85
Naive	0.415	0.5	0.453	0.83	0.414	0.5	0.453	0.828
BoW	0.790	0.758	0.772	0.877	0.793	0.757	0.773	0.878
tf-idf	0.805	**0.820**	**0.812**	**0.890**	0.771	**0.781**	**0.776**	0.870
BERT	0.767	0.763	0.765	0.866	0.777	0.717	0.741	0.868
				SMOTE				
tf-idf	0.820	0.772	0.792	0.890	0.746	0.724	0.734	0.855
				Backtranslation				
tf-idf	0.774	0.721	0.743	0.866	0.796	0.748	0.768	0.878
BERT	0.745	0.702	0.719	0.853	0.698	0.662	0.676	0.832
				Synonyms				
tf-idf	**0.811**	0.648	0.684	0.864	**0.825**	0.710	0.748	**0.880**
BERT	0.735	0.746	0.740	0.848	0.728	0.732	0.730	0.845

Table 5.4: AU01_01 - alternatives to suicidal behaviour - results.

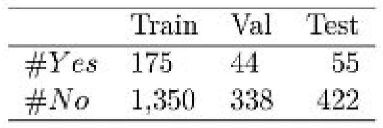
This variable code focuses on alternatives to suicidal behaviour. We observe that the count-based methods outperform the BERT models, both in the unbalanced and balanced setting. This implies that even though we only train on 262 positive instances, the information, i.e., key words and phrases, are the strongest indicator for the positive class. The context and semantic structure of the text seem to be less important. The best model is the *unbalanced tf-idf* model, which outperforms all remaining systems by 5% F_1 on both the validation an the test set. We can confirm this hypothesis by analyzing the tf-idf performances for the back translation and the synonyms. Specifically, we can see that the back translation is slightly better than the synonyms, but both are significantly lower than the original tf-idf model. This behaviour is caused by the dilution of the idf values for the most important words that indicate the positive class. We can see that the back translation performs slightly better, which implies that the back translations of the key words were successful, translating back to the original word. In contrast, the synonym approach might have replaced those very key words, yielding an inferior model since crucial information was replaced.

5.2.3 CS02_01 - suggesting monocausality

This variable code reports if exactly one possible motive, cause, or trigger of suicidal behaviour is reported. An example for a positive instance is: "the 19-year old boy, who suffered from depression, listened to voices in his head, and committed suicide."

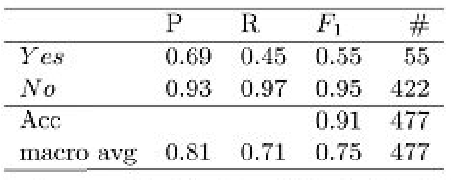


CS02_01 class frequency distributions.



	Train	Val	Test
$\#Yes$	175	44	55
$\#No$	1,350	338	422

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Yes	0.69	0.45	0.55	55
No	0.93	0.97	0.95	422
Acc			0.91	477
macro avg	0.81	0.71	0.75	477

Test performance metrics of the best model (backtranslated tf-idf).

Figure 5.6: Experimental results for CS02_01.

System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							0.72	0.95
Naive	0.442	0.5	0.469	0.885	0.442	0.5	0.469	0.885
BoW	0.685	0.717	0.699	0.866	0.664	0.697	0.678	0.855
tf-idf	0.823	**0.737**	**0.771**	0.919	0.730	0.677	0.699	0.891
BERT	0.782	0.637	0.676	0.901	0.703	0.603	0.629	0.885
				SMOTE				
tf-idf	**0.953**	0.602	0.645	0.908	0.748	0.550	0.562	0.889
				Backtranslation				
<u>tf-idf</u>	<u>0.878</u>	0.710	<u>0.763</u>	**0.924**	0.813	**0.714**	**0.751**	**0.914**
BERT	0.84	0.58	0.60	0.90	0.695	0.524	0.518	0.885
				Synonyms				
tf-idf	0.909	0.612	0.657	0.908	**0.819**	0.595	0.630	0.899
BERT	0.903	0.601	0.641	0.906	0.757	0.567	0.588	0.891

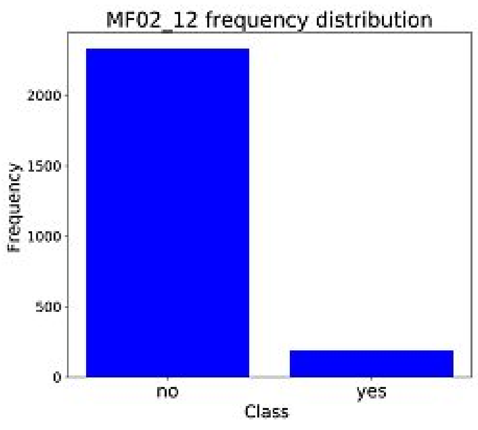
Table 5.5: CS02_01 - monocausality - results, the performance metrics that were used to select the best model are underlined.

Detecting monocausality poses a difficult challenge for BERT models, regardless of the class frequency distributions. In our experiments the tf-idf models always outperform their corresponding BERT models, which implies that syntax and semantic structure are only secondary to the presence of multiple keywords. Even though the unbalanced tf-idf model achieves the highest validation F_1, we chose the backtranslated tf-idf model as the best model for this variable code. We decided to choose the model with the higher precision and slightly lower F_1 with the idea that the balanced model is able to generalize better as we have already experienced in §5.1.2. However, we can see that the translation of the keywords, as applied in the synonym approach, decreases the models performance by negatively impacting the associated idf values. The back translation approach correctly backtranslates the keywords and is able to introduce additional highly informative keywords, further enhancing the models ability to generalize.

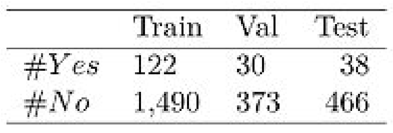
5.3 Difficult binary classification

5.3.1 MF02_12 - healing story

This variable code determines if the focus area of the text lies on a healing story, i.e., the meaning or emotional content evolves around hope, recovery from and coping with a suicidal crises or suicidal thoughts. In order to qualify as a positive instance one sentence in an average length text should mention a healing story or the framing of the text suggests that the context is related to a healing story.

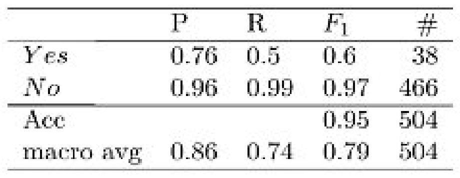


MF02_12 class frequency distributions.



	Train	Val	Test
$\#Yes$	122	30	38
$\#No$	1,490	373	466

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Yes	0.76	0.5	0.6	38
No	0.96	0.99	0.97	466
Acc			0.95	504
macro avg	0.86	0.74	0.79	504

Test performance metrics of the best model (SMOTE tf-idf).

Figure 5.7: Experimental results for MF02_12.

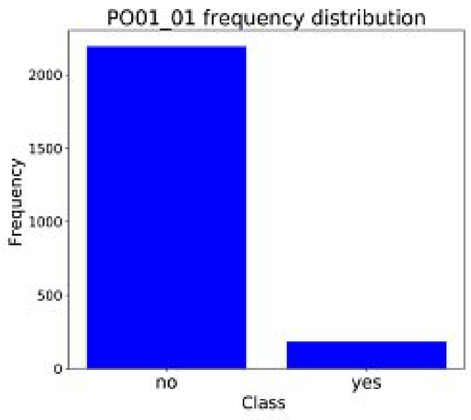
System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							1.0	1.0
Naive	0.463	0.5	0.481	0.923	0.462	0.5	0.480	0.925
BoW	0.812	0.724	0.759	**0.943**	0.759	0.734	0.746	0.932
tf-idf	0.757	**0.749**	0.753	0.933	0.780	**0.773**	0.777	0.939
BERT	0.797	0.723	0.753	0.940	0.793	0.750	0.770	0.940
SMOTE								
tf-idf	0.842	0.698	**0.763**	**0.943**	0.860	0.744	**0.788**	**0.950**
Backtranslation								
tf-idf	0.860	0.614	0.663	0.938	0.858	0.705	0.757	0.946
BERT	0.667	0.559	0.579	0.921	0.810	0.702	0.742	0.940
Synonyms								
tf-idf	**0.885**	0.582	0.622	0.935	**0.931**	0.644	0.705	0.944
BERT	0.769	0.595	0.632	0.931	0.851	0.692	0.744	0.944

Table 5.6: MF02_12 - healing story - results.

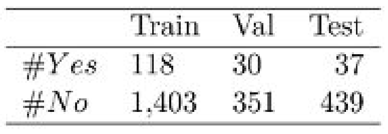
The performance of all three unbalanced systems is very similar for this variable code, where BoW performs the best on the validation set, and tf-idf on the test set. Neither backtranslation nor synonym replacement are able improve upon the baseline performances through balancing the data set. However, SMOTE is able to generate synthetic samples from the tf-idf values, that effectively improves both the validation and the test performance. This implies that even though there are only 122 training samples the information contained inside of them is enough to generate a "good" model. SMOTE creates additional observations with similar tf-idf features, but not the same, therefore expanding the feature space and improving upon the original performance.

5.3.2 PO01_01 - examples of a positive outcome of suicidal crisis

This variable code reports if the text contains example(s) of some positive outcome related to a suicidal crisis. This can be an example of a person experiencing a suicide attempt or suicidal ideations, and mastering his or her crisis or leading to positive behaviour. The ending is positive.

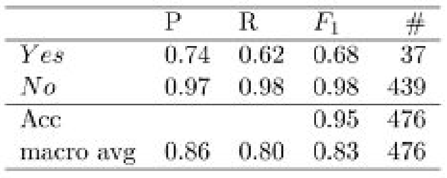


PO01_01 class frequency distributions.



	Train	Val	Test
$\#Yes$	118	30	37
$\#No$	1.403	351	439

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Yes	0.74	0.62	0.68	37
No	0.97	0.98	0.98	439
Acc			0.95	476
macro avg	0.86	0.80	0.83	476

Test performance metrics of the best model (unbalanced BERT).

Figure 5.8: Experimental results for PO01_01.

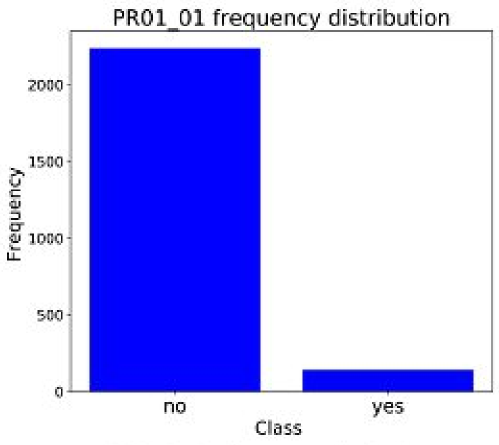
System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							1.0	1.0
Naive	0.461	0.5	0.480	0.921	0.461	0.5	0.480	0.922
BoW	0.796	0.722	0.752	0.937	0.786	0.743	0.762	0.937
tf-idf	0.809	0.771	0.788	0.942	0.785	0.806	0.795	0.939
BERT	0.902	0.762	**0.815**	**0.955**	0.855	0.802	**0.826**	**0.954**
				SMOTE				
tf-idf	0.836	**0.773**	0.801	0.948	0.793	**0.807**	0.8	0.941
				Backtranslation				
tf-idf	0.915	0.632	0.689	0.940	**0.922**	0.714	0.778	0.952
BERT	0.828	0.693	0.739	0.94	0.796	0.744	0.767	0.939
				Synonyms				
tf-idf	**0.969**	0.617	0.673	0.940	0.902	0.66	0.720	0.943
BERT	0.904	0.697	0.759	0.948	0.860	0.75	0.793	0.95

Table 5.7: PO01_01 - positive outcome of suicidal crisis - results.

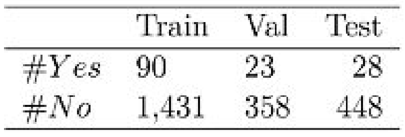
We observed that the BERT models always outperform the corresponding NLP models. Therefore we conclude that in order to determine if a suicidal crisis had a positive outcome, it is not enough to consider the words and their frequencies but one must also look at the context and semantic structure of the texts. Furthermore, we saw that the only balancing strategy to improve upon the baseline performances was SMOTE. This suggests that neither back translation nor synonyms could provide contextually relevant translations or word replacements respectively. To further improve upon the best model's performance, *unbalanced BERT*, one must add additional human labelled data for the positive class, since we were not able to generate new highly informative features.

5.3.3 PR01_01 - enhancing myths about suicide

This variable code reports whether an item enhances (a) public myth(s) on suicidal behaviour or not. This also includes implicit enhancements. Some examples that are classified as a positive instance are: those who talk about suicide are less likely to attempt suicide; there are no preceding warning signs; there is nothing you can do about suicidality.

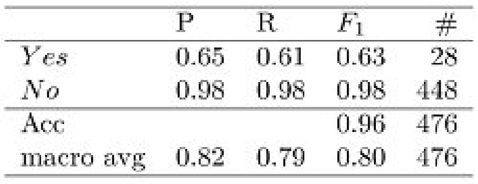


PR01_01 class frequency distributions.



	Train	Val	Test
$\#Yes$	90	23	28
$\#No$	1,431	358	448

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Yes	0.65	0.61	0.63	28
No	0.98	0.98	0.98	448
Acc			0.96	476
macro avg	0.82	0.79	0.80	476

Test performance metrics of the best model (SMOTE tf-idf).

Figure 5.9: Experimental results for PR01_01.

System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							1.0	1.0
Naive	0.471	0.5	0.485	0.942	0.471	0.5	0.485	0.941
BoW	0.733	0.713	0.723	0.942	0.679	0.725	0.698	0.924
tf-idf	0.878	0.746	**0.796**	**0.963**	0.836	0.709	0.755	0.956
BERT	0.723	0.543	0.562	0.942	**0.974**	0.554	0.583	0.948
SMOTE								
tf-idf	0.797	**0.784**	0.791	0.955	0.815	**0.794**	**0.804**	**0.958**
Backtranslation								
tf-idf	0.881	0.679	0.739	0.958	0.805	0.69	0.732	0.952
BERT	0.83	0.655	0.706	0.953	0.761	0.653	0.690	0.945
Synonyms								
tf-idf	**0.976**	0.591	0.642	0.953	0.849	0.552	0.58	0.945
BERT	0.812	0.632	0.681	0.950	0.758	0.67	0.703	0.945

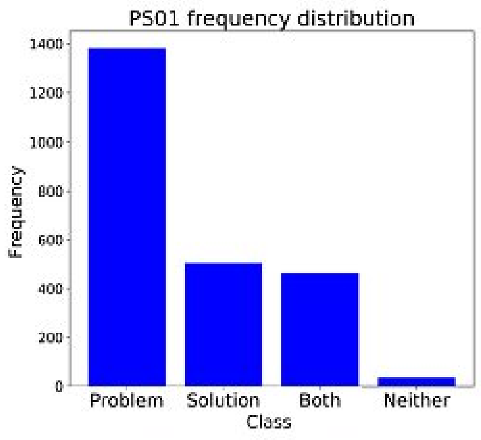
Table 5.8: PR01_01 - public myth enhanced - results.

This is the first variable code where we have less than 100 positive classes in the training sample and it represents the heaviest imbalance in the binary classification tasks. Therefore, the unbalanced BERT model performs poorly even though we train it with a very low learning rate and for many epochs. The samples that are present in the training data are not enough for BERT to create expressive features for the positive class, and it only predicts a few positive instances from the test set. Contrary to this, we see that the traditional NLP approaches handle this imbalance quite well, with the tf-idf model performing best, i.e., highest validation F_1, in the unbalanced dataset. However, only the SMOTE model is able to compete with the performance of the original tf-idf model, sacrificing precision for recall. We can see that balancing with both synthetic data generation strategies, always improves BERT's performance. We follow our established logic, i.e., that the balanced model generalizes better even though the validation performance is slightly worse, and select the *SMOTE tf-idf* model as our best model. This model does in fact achieve the best test performance suggesting that generating observations that are very similar but not duplicates during training impacts the performance positively.

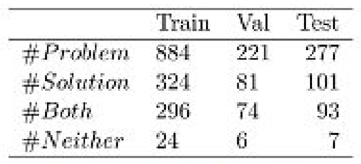
5.4 Multi-class classification

5.4.1 PS01 - problem or solution focus

This variable codes if the text describes suicidal behaviour and/or suicidal ideation as a *problem* without offering solutions, or if the main focus lies more on a *solution* rather than the problem. However, there might be some cases where the text contains both problems and solutions where the main focus is unclear, those observations are classified as *both*. Similarly, if the text does not show features of either a problem or solution focus, the observations are classified as *neither*.

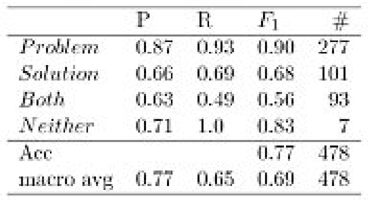


PS01 class frequency distributions.



	Train	Val	Test
#*Problem*	884	221	277
#*Solution*	324	81	101
#*Both*	296	74	93
#*Neither*	24	6	7

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Problem	0.87	0.93	0.90	277
Solution	0.66	0.69	0.68	101
Both	0.63	0.49	0.56	93
Neither	0.71	1.0	0.83	7
Acc			0.77	478
macro avg	0.77	0.65	0.69	478

Test performance metrics of the best model (synonyms BERT).

Figure 5.10: Experimental results for PS01.

System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							1.0	1.0
Naive	0.145	0.25	0.183	0.579	0.15	0.25	0.183	0.579
BoW	0.668	0.700	0.680	0.780	0.690	0.628	0.654	0.745
tf-idf	0.661	**0.744**	**0.692**	0.757	0.685	0.613	0.636	0.738
BERT	0.662	0.664	0.663	0.785	0.704	0.648	0.671	0.759
SMOTE								
tf-idf	**0.687**	0.523	0.564	0.764	0.499	0.464	0.471	0.741
Backtranslation								
tf-idf	0.684	0.677	0.680	**0.801**	0.702	0.612	0.645	**0.770**
BERT	0.658	0.666	0.661	0.775	0.750	0.642	0.681	0.753
Synonyms								
tf-idf	0.657	0.558	0.583	0.772	0.755	0.528	0.558	0.757
<u>BERT</u>	<u>0.681</u>	0.697	<u>0.688</u>	0.780	**0.768**	**0.652**	**0.693**	**0.770**

Table 5.9: PS01 - problem or solution - results.

This variable code represents a four class classification task, where the majority class is *Problem*. Two classes, *Solution* and *Both*, have similar frequencies with roughly one third of the majority class frequency. The remaining class *Neither* contains all observations that do not fit into one of the other classes and we only have 37 observations in total. The initial experiments with the unbalanced data show similar performance between

BoW and tf-idf on the validation set, where tf-idf performs best. BERT performs slightly worse on the validation set, but generalizes better achieving the best unbalanced test set performance. Balancing the training data with SMOTE only slightly improves the validation precision, but worsens all other performance metrics. This behaviour is consistent for all balanced tf-idf experiments, implying that the balancing of the minority classes generates synthetic data that is very similar to the original and dilutes the idf values of key features. Contrarily, BERT's performance benefits from the synthetic data, where the synonyms approach outperforms the back translations. Even though the synonyms BERT validation performance is slightly worse than the unbalanced tf-idf, we follow our logic, i.e., that the model trained on a balanced dataset generalizes better towards unseen texts, and choose *synonyms BERT* as the best model for this variable code. These experiments show, that the incorporation of carefully generated synthetic data into the training set can improve the performance of the model. This consistently leads to improved precision in trade for some recall, and the model's capacity increases, improving it's ability to generalize.

5.4.2 MF01 - main focus

This variable code reports on the *main focus* of the item, meaning the main theme of the text. If a proportion of a text is dedicated to the specific focus, or the framing of the text implies the specific focus, it is designated as the main focus.

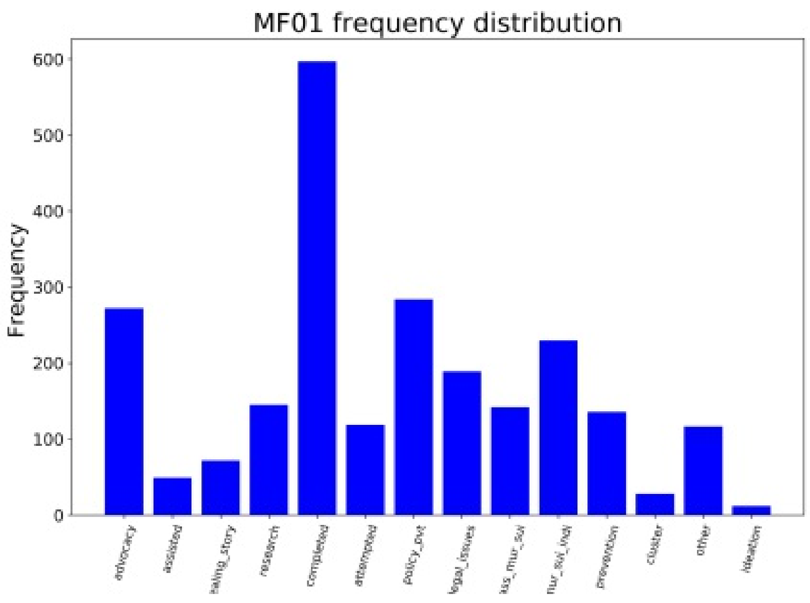
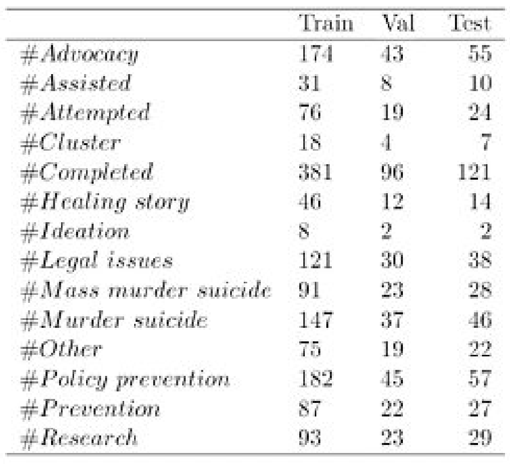


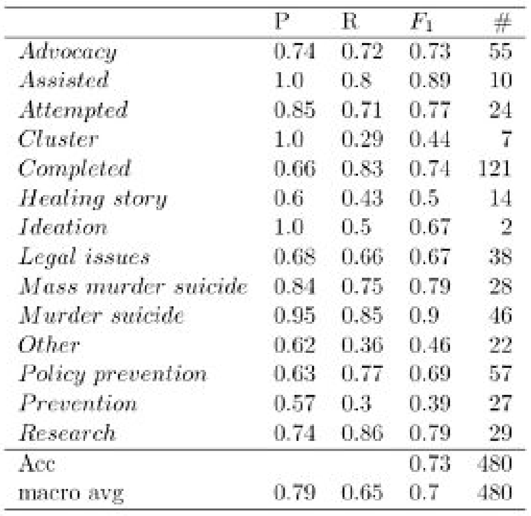
Figure 5.11: MF01 class frequency distributions.

This variable code is the multi-class classification problem with the most classes and the most severe class imbalances. Figure 5.11 visualizes the imbalance of the entire dataset while the left hand side of Figure 5.12 shows how the observations are distributed across the training, validation and test set respectively.



	Train	Val	Test
#Advocacy	174	43	55
#Assisted	31	8	10
#Attempted	76	19	24
#Cluster	18	4	7
#Completed	381	96	121
#Healing story	46	12	14
#Ideation	8	2	2
#Legal issues	121	30	38
#Mass murder suicide	91	23	28
#Murder suicide	147	37	46
#Other	75	19	22
#Policy prevention	182	45	57
#Prevention	87	22	27
#Research	93	23	29

Sample distribution for train, validation, and test sets.



	P	R	F_1	#
Advocacy	0.74	0.72	0.73	55
Assisted	1.0	0.8	0.89	10
Attempted	0.85	0.71	0.77	24
Cluster	1.0	0.29	0.44	7
Completed	0.66	0.83	0.74	121
Healing story	0.6	0.43	0.5	14
Ideation	1.0	0.5	0.67	2
Legal issues	0.68	0.66	0.67	38
Mass murder suicide	0.84	0.75	0.79	28
Murder suicide	0.95	0.85	0.9	46
Other	0.62	0.36	0.46	22
Policy prevention	0.63	0.77	0.69	57
Prevention	0.57	0.3	0.39	27
Research	0.74	0.86	0.79	29
Acc			0.73	480
macro avg	0.79	0.65	0.7	480

Test performance metrics of the best model (SMOTE tf-idf).

Figure 5.12: Experimental results for MF01.

System	Val				Test			
	P	R	F1	Acc	P	R	F1	Acc
Human							0.819	0.85
Naive	0.018	0.071	0.028	0.248	0.018	0.071	0.029	0.251
BoW	0.543	0.527	0.525	0.650	0.71	0.582	0.622	0.674
tf-idf	0.564	0.590	0.572	0.661	0.737	**0.69**	0.702	0.704
BERT	0.58	**0.609**	**0.592**	0.715	0.696	0.608	0.626	0.716
SMOTE								
tf-idf	0.599	0.597	**0.592**	0.702	0.794	0.654	0.703	0.729
Backtranslation								
tf-idf	0.586	0.576	0.576	0.697	0.834	0.677	**0.725**	**0.745**
BERT	0.584	0.582	0.578	0.702	0.764	0.661	0.694	0.722
Synonyms								
tf-idf	**0.616**	0.486	0.516	0.666	**0.845**	0.513	0.586	0.66
BERT	0.595	0.59	0.586	**0.718**	0.766	0.651	0.691	0.727

Table 5.10: MF01 - main focus - results.

The initial experiments with the unbalanced data show that the fine-tuned BERT model performs better on the validation set than the tf-idf model. However, the difference in

F_1 is only 2%. The major difference is the performance on the test set. We trained the BERT model with a low learning rate (1e-5) for many epochs (15), which optimized the training and validation error exceptionally well, but it performs worse on the test set than the tf-idf model. The poor performance on the test set implies that more of the training specific features were present in the validation set than in the test set. The tf-idf model does not seem to suffer from this issue, most likely because it additionally weights the term frequencies by the idf values.

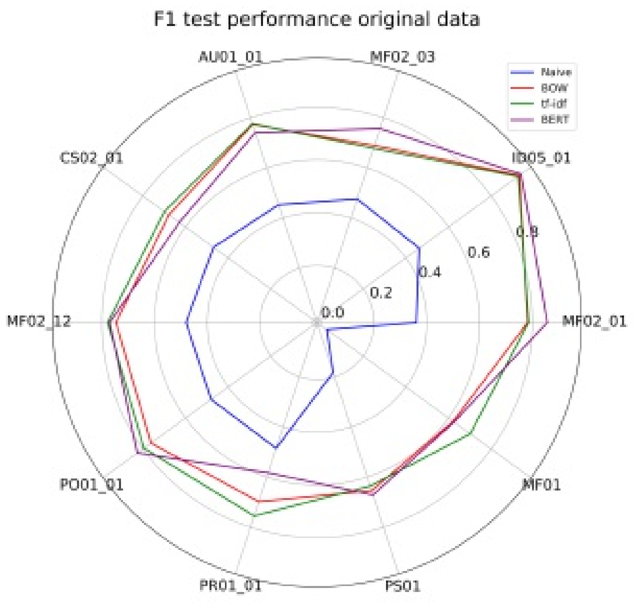
The BERT models that are trained on the balanced data, via both back translation and synonyms, improve the model's ability to generalize, which is consistent to our observations from the binary classification experiments. This effectively improves the test performance by 5% F_1 and sacrifices only $\leq 2\%$ F_1 on the validation set. However, no BERT model is able to surpass the baseline performance of the unbalanced tf-idf model.

The balanced tf-idf models produce more promising results, except for the synonym balancing strategy. According to our workflow, we select the model with the best performance (highest F_1) on the validation set. Therefore, we choose the *SMOTE tf-idf* model. It achieves the highest F_1, and seconds highest precision and recall across all experiments on the validation set. However, the backtranslated tf-idf model can clearly compete with the best model. It performs slightly worse on the validation set $\leq 2\%$, but it's test performance is slightly better $\sim 2\%$.

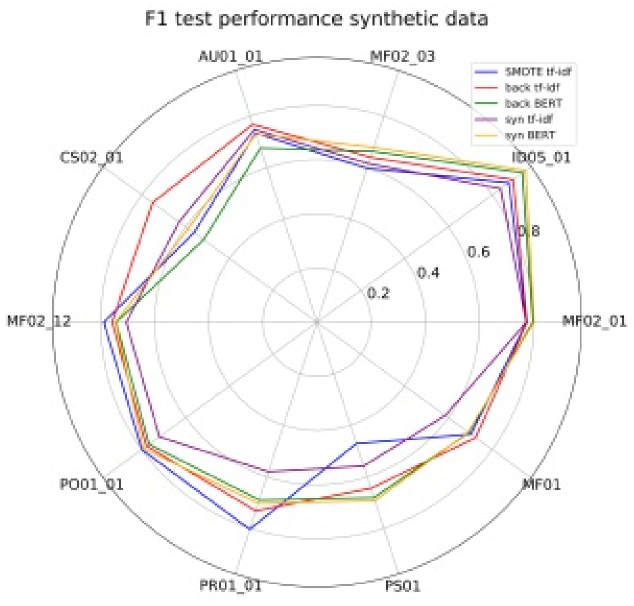
Our best models achieve a performance of $\sim 70\%$ F_1 and $\sim 73\%$ Acc on the test set, which is very close to human-like performance of $F_1 = 0.819$ and $Acc = 0.85$. We conclude that carefully balancing the training data with the correct synthetic data generation technique enhances the model's capacity. This must be combined with a training strategy that minimizes the overfitting and maximizes the model's ability to generalize.

5.5 Summary

This section visually presents the results of this chapter. Figure 5.13 visualizes the experimental results by reporting the macro-averaged F_1 score on the test set for the original data (see Figure 5.13a) and the synthetically extended data (see Figure 5.13b).



(a) Experiments on original data.



(b) Experiments with additional synthetic data.

Figure 5.13: Macro-averaged F_1-scores on the test set.

CHAPTER 6

Conclusion

In this thesis, we have explored the application of bidirectional language models that exploit transfer learning through extensive pretraining on massive document collections. This section summarizes our findings and discusses the effectiveness of synthetic data generation techniques.

We encountered many binary variable codes with severe class imbalances, however, the tf-idf model worked surprising well most of the time. BERT suffers from a massive fine-tuning issue, where selecting the appropriate learning rate and number of epochs becomes crucial. Even though the creators of BERT and the literature recommend learning rates $\in \{2e-5, 3e-5, 5e-5\}$ and epochs $\in \{3, 4, 5\}$, we discovered that selecting a lower learning rate e.g., $1e-5$, enables the model to go through more iterations (epochs=≥ 8) before encountering the overfitting issue. With every additional iteration the model is able to learn more distinct features with the limited amount of training data. The experiments with higher learning rates and lower epochs, e.g., $lr = 5e-5.epochs = 3$, produced models that only slightly outperformed the naive classifier, essentially always predicting the majority class, and performed worse than the low learning rate - high epoch models. This issue can be addressed by including additional data.

We implemented and tested two novel synthetic data generation techniques, back translation and synonym replacement, and analyzed their impact on the performance (measured in F_1). Both methods consistently improve the precision while sacrificing some recall. In our experiments, this trade-off was beneficial if a maximum of 30% of the synthetic data was added to the training set. Adding more synthetic data tips the balance too far in favour of the synthetic data and the performance appears to tail off. We conclude that both methods dilute the original samples too heavily, forcing the model to learn random noise instead of the original features. Furthermore, we discovered that including up to 30% of the synthetic data consistently improves the model's ability to generalize towards new inputs, showing significant improvement from the validation performance to the performance on the test set. One caveat must be noted: some variable codes benefited from neither back translation nor synonym replacement, but saw massive improvements from balancing the minority classes with SMOTE. These variable codes performed best if the balancing strategy generated observations with very similar, but not identical, features to the original training data. Therefore, we conclude that balancing the dataset with synthetic data generation techniques is sensible, but must be specifically tailored to each classification task and extensively tested. In all difficult binary and multi-class classification problems, synthetically balancing the training data *always* improved the performance on the test set.

Our experiments showed that pre-trained bidirectional language models work incredibly well for the detection and classification of suicide-related content. However, the models are extremely expensive in both training and inference, and improvements seem to mostly come from using even more expensive models and more data.

Future Work

This thesis provides the methodology needed to label large amounts of data according to numerous characteristics that are relevant for media effects research on suicide. In addition to the models presented in this work, the developed code can be easily adapted to train similar models for detecting many other potentially relevant features. Future studies will be able to use these models to label large datasets, to then investigate which characteristics of suicide reporting have harmful or beneficial effects on suicide cases and help seeking behavior. Investigating these associations at large scales and across larger time spans than previous studies will provide more robust estimates of the importance of different characteristics.

APPENDIX A

Examples for positive instances

- **ID05_01**: the text reports on suicidal ideation or behaviour of a celebrity
  - The death of Robin Williams still hurts.
  - Avicii's music will keep him alive.
- **MF02_01**: the text reports on a completed suicide
  - The farmer shot himself because of debt.
  - He committed suicide this Friday.
- **II01_01** the text reports on preventing suicide at the individual level: different actions qualify as yes.
  - They removed the ropes from prisoners in order for them to not hang themselves.
  - If you feel someone might think about suicide, ask him / her directly about it.
- **AU01_01** the text reports on alternatives to suicidal behaviour: many possible alternatives exist.
  - Going for a walk to calm down
  - How to make new friends?
- **CS02_01** - the item suggests that suicidal behaviour is monocausal
  - The 19-year old boy, who suffered from depression, listened to voices in his head, and committed suicide.
  - She wanted to break up with him, that was his death sentence.

- **PO01__01** - the item includes example(s) of some positive outcome related to a suicidal crisis.
  - I finally don't think about killing myself anymore.
  - I just called the suicide prevention lifeline and I feel much better!
- **PR01__01** - the item enhances (a) public myth(s) on suicidal behaviour. It also includes implicit enhancement.
  - Someone who has history of making "cries for help" will not die by suicide.
  - Talking about suicide encourages suicide.
  - Only the mentally ill exhibit suicidal behavior.
- **PS01** - the text discusses whether the topic is described as a problem, solution, both or neither.
  - Problem: "every 40 secs someone commits suicide, and that 2160 lives a day"
  - Problem: "More than 96 in-patients died by suicide in Ontario hospitals since 2007"
  - Solution: I have had those terrible thoughts non-stop, for most of my whole life. But one day I took the phone and called the Lifeline.. I was on the phone and it was the first time for me to realise that I was born this way and...
- **MF01** - the text describes the the suicide method, multiple may be present in the text.
  - Completed: The man hung himself.
  - Attempted: He tried to kill himself but emergency services were able to save his life.

List of Figures

List of Tables

List of Algorithms

Bibliography

A. Aizawa. An information-theoretic perspective of tf–idf measures q. *Information Processing and Management*, page 21, 2003.

J. Alammar. The Illustrated Transformer, 2018a. URL `http://jalammar.github.io/illustrated-transformer/`.

J. Alammar. The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), 2018b. URL `https://jalammar.github.io/illustrated-bert/`.

M. Aronoff. *Morphology by itself: Stems and inflectional classes*. Number 22. MIT press, 1994.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016. URL `http://arxiv.org/abs/1607.06450`. arXiv: 1607.06450.

R. Baeza-Yates and B. Ribeiro-Neto. *Modern information retrieval*, volume 463. ACM press New York, 1999.

D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016. URL `http://arxiv.org/abs/1409.0473`. arXiv: 1409.0473.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A Neural Probabilistic Language Model. page 19, 2003.

R. Blagus and L. Lusa. SMOTE for high-dimensional class-imbalanced data. *BMC Bioinformatics*, 14(1):106, Dec. 2013. ISSN 1471-2105. doi: 10.1186/1471-2105-14-106. URL `https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-14-106`.

J. Blitzer, M. Dredze, and F. Pereira. Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 440–447, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P07-1056`.

T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, 29:4349–4357, 2016.

T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean. Large Language Models in Machine Translation. page 10, 2007.

P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A Statistical Approach to Machine Translation. *Computational Linguistics*, 16(2):79–85, 1990. URL `https://www.aclweb.org/anthology/J90-2002`.

P. F. Brown, V. J. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer. Class-Based *n*-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–480, 1992. URL `https://www.aclweb.org/anthology/J92-4003`.

A. E. Bryson Jr, W. F. Denham, and S. E. Dreyfus. Optimal programming problems with inequality constraints. *AIAA journal*, 1(11):2544–2550, 1963.

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

T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, M. Zietz, M. M. Hoffman, W. Xie, G. L. Rosen, B. J. Lengerich, J. Israeli, J. Lanchantin, S. Woloszynek, A. E. Carpenter, A. Shrikumar, J. Xu, E. M. Cofer, C. A. Lavender, S. C. Turaga, A. M. Alexandari, Z. Lu, D. J. Harris, D. DeCaprio, Y. Qi, A. Kundaje, Y. Peng, L. K. Wiley, M. H. S. Segler, S. M. Boca, S. J. Swamidass, A. Huang, A. Gitter, and C. S. Greene. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141): 20170387, Apr. 2018. ISSN 1742-5689, 1742-5662. doi: 10.1098/rsif.2017.0387. URL `https://royalsocietypublishing.org/doi/10.1098/rsif.2017.0387`.

J. S. Cho and H. White. Generalized runs tests for the IID hypothesis. *Journal of Econometrics*, 162(2):326–344, June 2011. ISSN 03044076. doi: 10.1016/j.jeconom.2011.02.001. URL `https://linkinghub.elsevier.com/retrieve/pii/S0304407611000285`.

G. E. Churcher. INFORMATION RETRIEVAL. page 11, 2007.

R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural Language Processing (Almost) from Scratch. *NATURAL LANGUAGE PROCESSING*, page 45, 2011.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL `http://arxiv.org/abs/1810.04805`. arXiv: 1810.04805.

C. Drummond and R. C. Holte. C4.5, Class Imbalance, and Cost Sensitivity: Why Under-sampling beats Over-sampling. pages 1–8, 2003.

J. L. Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 1551-6709. doi: https://doi.org/10.1207/s15516709cog1402_1. URL `https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1`. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1.

C. J. Fall, A. Törcsvári, K. Benzineb, and G. Karetka. Automated categorization in the international patent classification. *ACM SIGIR Forum*, 37(1):10–25, Apr. 2003. ISSN 0163-5840. doi: 10.1145/945546.945547. URL `https://dl.acm.org/doi/10.1145/945546.945547`.

J. R. Firth. Applications of General Linguistics. *International Journal of Applied Linguistics*, 17(3):402–413, 1957. Publisher: Wiley Online Library.

G. M. Foody. Status of land cover classification accuracy assessment. *Remote Sensing of Environment*, 80(1):185–201, Apr. 2002. ISSN 00344257. doi: 10.1016/S0034-4257(01)00295-4. URL `https://linkinghub.elsevier.com/retrieve/pii/S0034425701002954`.

G. Forman. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research*, 3:1289–1305, Mar. 2003.

S. García, A. Fernández, J. Luengo, and F. Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Computing*, 13(10):959–977, Aug. 2009. ISSN 1432-7643, 1433-7479. doi: 10.1007/s00500-008-0392-y. URL `http://link.springer.com/10.1007/s00500-008-0392-y`.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

M. A. K. Halliday. The linguistic sciences and language teaching. 1964. Publisher: ERIC.

L. K. Hansen and J. Larsen. Unsupervised learning and generalization. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 25–30 vol.1, June 1996. doi: 10.1109/ICNN.1996.548861.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning – Data Mining, Inference, and Prediction*. 2009.

K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. *arXiv:1603.05027 [cs]*, July 2016. URL `http://arxiv.org/abs/1603.05027`. arXiv: 1603.05027.

J. Howard and S. Ruder. Universal Language Model Fine-tuning for Text Classification. *arXiv:1801.06146 [cs, stat]*, May 2018. URL `http://arxiv.org/abs/1801.06146`. arXiv: 1801.06146.

L. Karttunen. Syntax and semantics of questions. *Linguistics and philosophy*, 1(1):3–44, 1977. Publisher: Springer.

L. M. Ketari, M. Chandra, and M. A. Khanum. A Study of Image Spam Filtering Techniques. In *2012 Fourth International Conference on Computational Intelligence and Communication Networks*, pages 245–250, Nov. 2012. doi: 10.1109/CICN.2012.34.

Y. Kim. Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882 [cs]*, Sept. 2014. URL `http://arxiv.org/abs/1408.5882`. arXiv: 1408.5882.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. Publisher: Ieee.

B. Li, Z. Zhao, T. Liu, P. Wang, and X. Du. Weighted Neural Bag-of-n-grams Model: New Baselines for Text Classification. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1591–1600, Osaka, Japan, Dec. 2016. The COLING 2016 Organizing Committee. URL `https://www.aclweb.org/anthology/C16-1150`.

B. Li, T. Liu, Z. Zhao, P. Wang, and X. Du. Neural bag-of-ngrams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017. Issue: 1.

T. Li, T. Mei, I.-S. Kweon, and X.-S. Hua. Contextual Bag-of-Words for Visual Categorization. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(4): 381–392, Apr. 2011. ISSN 1051-8215, 1558-2205. doi: 10.1109/TCSVT.2010.2041828. URL `http://ieeexplore.ieee.org/document/5401099/`.

E. D. Liddy. *Natural Language Processing*. Natural Language Processing. In Encyclopedia of Library and Information Science. Marcel Decker, Inc., NY, 2 edition, 2001.

Z. C. Lipton, J. Berkowitz, and C. Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv:1506.00019 [cs]*, Oct. 2015. URL `http://arxiv.org/abs/1506.00019`. arXiv: 1506.00019.

Y. Liu, Y. Zhou, S. Wen, and C. Tang. A Strategy on Selecting Performance Metrics for Classifier Evaluation. *International Journal of Mobile Computing and Multimedia Communications*, 6:20–35, Oct. 2014. doi: 10.4018/IJMCMC.2014100102.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, July 2019. URL `http://arxiv.org/abs/1907.11692`. arXiv: 1907.11692.

W. Medhat, A. Hassan, and H. Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, Dec. 2014. ISSN 2090-4479. doi: 10.1016/j.asej.2014.04.011. URL `http://www.sciencedirect.com/science/article/pii/S2090447914000550`.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, Sept. 2013. URL `http://arxiv.org/abs/1301.3781`. arXiv: 1301.3781.

G. A. Miller. WordNet: A Lexical Database for English. *Communications of the Acm*, 38:39–41, 1995.

A. Mnih and G. Hinton. Three New Graphical Models for Statistical Language Modelling. *Proceedings of the 24th international conference on Machine learning*, page 8, 2007.

T. Niederkrotenthaler, M. Voracek, A. Herberth, B. Till, M. Strauss, E. Etzersdorfer, B. Eisenwort, and G. Sonneck. Role of media reports in completed and prevented suicide: Werther v. Papageno effects. *The British Journal of Psychiatry*, 197(3): 234–243, Sept. 2010. ISSN 0007-1250, 1472-1465. doi: 10.1192/bjp.bp.109.074633.

T. Niederkrotenthaler, K.-w. Fu, P. S. F. Yip, D. Y. T. Fong, S. Stack, Q. Cheng, and J. Pirkis. Changes in suicide rates following media reports on celebrity suicide: a meta-analysis. *J Epidemiol Community Health*, 66(11):1037–1042, Nov. 2012. ISSN 0143-005X, 1470-2738. doi: 10.1136/jech-2011-200707. URL `https://jech.bmj.com/content/66/11/1037`.

T. Niederkrotenthaler, D. J. Reidenberg, B. Till, and M. S. Gould. Increasing Help-Seeking and Referrals for Individuals at Risk for Suicide by Decreasing Stigma. *American Journal of Preventive Medicine*, 47(3):S235–S243, Sept. 2014. ISSN 07493797. doi: 10.1016/j.amepre.2014.06.010. URL `https://linkinghub.elsevier.com/retrieve/pii/S0749379714002785`.

T. Niederkrotenthaler, M. Braun, J. Pirkis, B. Till, S. Stack, M. Sinyor, U. S. Tran, M. Voracek, Q. Cheng, F. Arendt, S. Scherr, P. S. F. Yip, and M. J. Spittal. Association between suicide reporting in the media and suicide: systematic review and meta-analysis. *BMJ*, page m575, Mar. 2020. ISSN 1756-1833. doi: 10.1136/bmj.m575. URL `https://www.bmj.com/lookup/doi/10.1136/bmj.m575`.

M. K. Nock, G. Borges, E. J. Bromet, J. Alonso, M. Angermeyer, A. Beautrais, R. Bruffaerts, W. T. Chiu, G. de Girolamo, S. Gluzman, R. de Graaf, O. Gureje, J. M. Haro, Y. Huang, E. Karam, R. C. Kessler, J. P. Lepine, D. Levinson, M. E. Medina-Mora, Y. Ono, J. Posada-Villa, and D. Williams. Cross-national prevalence and risk factors for suicidal ideation, plans and attempts. *British Journal of Psychiatry*, 192(2):98–105, Feb. 2008. ISSN 0007-1250, 1472-1465. doi: 10.1192/bjp.bp.107.040113. URL `https://www.cambridge.org/core/product/identifier/S0007125000233886/type/journal_article`.

B. Pang and L. Lee. Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135, Jan. 2008. ISSN 1554-0669. doi: 10.1561/1500000011. URL https://doi.org/10.1561/1500000011.

E. Pavlick, P. Rastogi, J. Ganitkevitch, B. Van Durme, and C. Callison-Burch. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 425–430, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-2070. URL https://www.aclweb.org/anthology/P15-2070.

J. Pennington, R. Socher, and C. D. Manning. GloVe: Global Vectors for Word Representation. page 12, 2014.

D. P. Phillips. The influence of suggestion on suicide: Substantive and theoretical implications of the Werther effect. *American sociological review*, pages 340–354, 1974. Publisher: JSTOR.

J. E. Pirkis, P. M. Burgess, C. Francis, R. W. Blood, and D. J. Jolley. The relationship between media reporting of suicide and actual suicide in Australia. *Social Science & Medicine*, 62(11):2874–2886, June 2006. ISSN 02779536. doi: 10.1016/j.socscimed.2005.11.033. URL https://linkinghub.elsevier.com/retrieve/pii/S0277953605006301.

A. Poncelas, D. Shterionov, A. Way, G. M. d. B. Wenniger, and P. Passban. Investigating Backtranslation in Neural Machine Translation. *arXiv:1804.06189 [cs]*, Apr. 2018. URL http://arxiv.org/abs/1804.06189. arXiv: 1804.06189.

D. Ravichandran and E. Hovy. Learning surface text patterns for a Question Answering System. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 41–47, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073092. URL https://www.aclweb.org/anthology/P02-1006.

S. Robertson. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60:2004, 2004.

G. H. Rosenfield and K. Fitzpatrick-Lins. A coefficient of agreement as a measure of thematic classification accuracy. *Photogrammetric engineering and remote sensing*, 52 (2):223–227, 1986.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. Publisher: Nature Publishing Group.

G. Salton and D. Harman. Information retrieval. In *Encyclopedia of Computer Science*, pages 858–863. John Wiley and Sons Ltd., GBR, Jan. 2003. ISBN 978-0-470-86412-8.

Y. Schlesinger, R. Straussman, I. Keshet, S. Farkash, M. Hecht, J. Zimmerman, E. Eden, Z. Yakhini, E. ben shushan, B. Reubinoff, Y. Bergman, I. Simon, and H. Cedar. Polycomb-mediated methylation on Lys27 of histone H3 pre-marks genes for de novo methylation in cancer. *Nature genetics*, 39:232–6, Mar. 2007. doi: 10.1038/ng1950.

H. Schwenk. Continuous space language models. *Computer Speech & Language*, 21(3): 492–518, July 2007. ISSN 08852308. doi: 10.1016/j.csl.2006.09.003. URL `https://linkinghub.elsevier.com/retrieve/pii/S0885230806000325`.

F. Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1–47, Mar. 2002. ISSN 0360-0300, 1557-7341. doi: 10.1145/505282.505283. URL `http://arxiv.org/abs/cs/0110053`. arXiv: cs/0110053.

R. Sennrich, B. Haddow, and A. Birch. Edinburgh Neural Machine Translation Systems for WMT 16. *arXiv:1606.02891 [cs]*, June 2016. URL `http://arxiv.org/abs/1606.02891`. arXiv: 1606.02891.

P. Shaw, J. Uszkoreit, and A. Vaswani. Self-Attention with Relative Position Representations. *arXiv:1803.02155 [cs]*, Apr. 2018. URL `http://arxiv.org/abs/1803.02155`. arXiv: 1803.02155.

S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain. Machine translation using deep learning: An overview. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 162–167, July 2017. doi: 10.1109/COMPâĎąIX.2017.8003957.

M. Sinyor, A. Schaffer, Y. Nishikawa, D. A. Redelmeier, T. Niederkrotenthaler, J. Sareen, A. J. Levitt, A. Kiss, and J. Pirkis. The association between suicide deaths and putatively harmful and protective factors in media reports. *Canadian Medical Association Journal*, 190(30):E900–E907, July 2018. ISSN 0820-3946, 1488-2329. doi: 10.1503/cmaj.170698. URL `http://www.cmaj.ca/lookup/doi/10.1503/cmaj.170698`.

M. Sokolova, N. Japkowicz, and S. Szpakowicz. Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, A. Sattar, and B.-h. Kang, editors, *AI 2006: Advances in Artificial Intelligence*, volume 4304, pages 1015–1021. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-49787-5 978-3-540-49788-2. doi: 10.1007/11941439_114. URL `http://link.springer.com/10.1007/11941439_114`. Series Title: Lecture Notes in Computer Science.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. page 30, 2014.

M. Stone. Cross-validation: A review. *Statistics: A Journal of Theoretical and Applied Statistics*, 9(1):127–139, 1978. Publisher: Taylor & Francis.

B. L. Sturm. Classification accuracy is not enough. *Journal of Intelligent Information Systems*, 41(3):371–406, Dec. 2013. ISSN 1573-7675. doi: 10.1007/s10844-013-0250-y. URL `https://link.springer.com/article/10.1007/s10844-013-0250-y`. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 3 Publisher: Springer US.

J. A. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999. Publisher: Springer.

W. L. Taylor. "Cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953. Publisher: SAGE Publications Sage CA: Los Angeles, CA.

J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 935–942, Corvalis, Oregon, 2007. ACM Press. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273614. URL `http://portal.acm.org/citation.cfm?doid=1273496.1273614`.

R. D. Van Valin and R. J. LaPolla. *Syntax: Structure, meaning, and function*. Cambridge University Press, 1997.

V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, 16(2):264–279, 1971.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin. Attention is All you Need. page 11, 2017.

A. Wang, W. Fan, J. Wu, Y. Shi, and D. Li. Spam filtering system study based on 2v-SVM. In *2008 7th World Congress on Intelligent Control and Automation*, pages 4213–4216, June 2008. doi: 10.1109/WCICA.2008.4594517.

D. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8, Mar. 1996. doi: 10.1162/neco.1996.8.7.1341.

D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997. ISSN 1089778X. doi: 10.1109/4235.585893. URL `http://ieeexplore.ieee.org/document/585893/`.

World Health Organization. *Preventing suicide: A global imperative*. World Health Organization, 2014.

World Health Organization. Preventing suicide: A resource for media professionals, 2017.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv:1609.08144 [cs]*, Oct. 2016. URL `http://arxiv.org/abs/1609.08144`. arXiv: 1609.08144.

A. Yeh. More accurate tests for the statistical significance of result differences. *arXiv:cs/0008005*, Aug. 2000. URL `http://arxiv.org/abs/cs/0008005`. arXiv: cs/0008005.

Z. Zhang and M. Sabuncu. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. page 11, 2018.