Detecting Calls to Action in Text Using Deep Learning

Er, Esref

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science

Detecting Calls to Action in Text Using Deep Learning

Master Thesis

Esref Er

Supervisors:

Dr. ir. Erik Quaeghebeur
Dr. Oele Koornwinder

Committee Members:

Dr. ir. Erik Quaeghebeur
Dr. Oele Koornwinder
Prof. Dr. Antske Fokkens

Eindhoven, June 2022

Abstract

Letters, notes, websites, and other forms of official government communication can be difficult for many Dutch citizens to understand. This makes it difficult for them to function effectively as citizens and users of government services. Requests from the government to citizens frequently call for significant tasks that must be completed, and as a result, the requests must be communicated clearly and effectively. This thesis focuses on automatically detecting these requests – these calls to action – in text using machine learning methodologies. The Dutch Tax Administration's internal datasets were labeled and used to train various machine learning pipelines, including novel deep learning methods such as fine-tuning BERTje. By fine-tuning BERTje on an in-house labeled dataset, we achieved an accuracy of 96.4%, which suggest that detecting calls to action might be feasible in other domains as well.

Preface

First and foremost, I would like to express my sincere gratitude to my supervisors, Erik Quaeghebeur and Oele Koornwinder for their guidance and moral support during this process. I would also like to thank my colleagues at the Dutch Tax Administration for their great cooperation. In particular, Jasper den Hamer, Sandra van Wijk and Kees van de Reepe. I have often been able to spar with them effectively about my research. Finally, I would like to thank my parents and brother. Their wisdom and motivational words have helped me complete this thesis.

Abbreviations

AI	artificial intelligence
AUC	area under the curve
BERT	bidirectional encoder representations from transformer
CBOW	continuous bag-of-words
CNN	convolutional neural network
CTA	call(s) to action
DNN	deep neural network
DT	decision trees
EDA	easy data augmentation
FN	false negatives
FP	false positives
GRU	gated recurrent unit
HMM	hidden markov model
IDF	inverse document frequency
KNN	k-nearest neighbours
LDA	linear discriminant analysis
LR	logistic regression
LSTM	long short-term memory
NB	naive Bayes
NLP	natural language processing
NMF	non-negative matrix factorization
PCA	principal component analysis
PoS	part-of-speech
RF	random forest
RNN	recurrent neural network
ROC	receiver operating curve
SGD	stochastic gradient descent
SVM	support vector machine
SVM	support vector machine
TF-IDF	term frequency-inverse document frequency
TF	term frequency
TN	true negatives
TP	true positives
UDA	unsupervised data augmentation
t-SNE	t-distributed stochastic neighbor embedding

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter begins by explaining the project's topic and scope. Next, the formal objective and the research question are presented. Finally, we discuss the remainder of the paper's structure.

1.1 Topic and Context

Numerous Dutch citizens have difficulty understanding letters, notes, websites, and other types of official communication [24]. This impairs their ability to perform effectively as citizens and consumers of government programs. Clarity in government-to-citizen communication is critical not only for citizens but also for government entities. Texts that are easily understood can result in cheaper processing and fewer complaints overall. As a result, various government entities, including the Dutch Tax Administration, seek to improve communication between citizens and the government. However, improving communication is a difficult issue to address as individuals communicate with one another in a variety of ways.

In everyday conversation, people are attuned not only to the sentences they exchange but also to the speech acts that those utterances perform: apologies, warnings, invitations, promises, requests, and the like [26]. In linguistics, a speech act is something that is expressed by an individual, conveys information, and also performs an action [4]. For instance, the phrase "I would like the Shoyu ramen; could you please make it for me?" is considered a speech act as it expresses the speaker's desire to obtain ramen, and it presents a request that someone make the ramen for them. While such acts are ubiquitous in communicative life, they did not become a sustained subject of study until the mid-twentieth century, at least in the English-speaking world [49]. Since that time, speech act theory has gained traction in a variety of scholarly disciplines including linguistics, psychology, legal theory, and artificial intelligence (AI) [26].

Requests from the government to citizens frequently involve quite significant tasks that must be completed, and so the requests must be communicated effectively and plainly. A request is a directive speech act whose purpose is to persuade the hearer to do something in circumstances where it is unlikely that he or she will do so in the normal course of events [54]. The more straightforward the request, the more transparent it is and the less burdensome it is to understand for the recipient. One way to mitigate the effects of this imposition is to employ indirect rather than direct strategies. The directness scale can be classified into three strategies [56]:

I Direct (explicitly designated as requests)

I'd like to ask you to clean the bathroom.
Clean up the bathroom.
I really wish you'd clean up the bathroom.
You'll have to clean up the bathroom.
I'm asking you to clean up the bathroom.

II Conventionally indirect (referring to the preconditions necessary)

Could you clean up the bathroom, please?
How about cleaning up?

III Non-conventionally indirect (referring to an object in light of its context)

I have a boyfriend (in response to a persistent harasser).
You have left the bathroom in a right mess.

Each year, the Dutch Tax Administration sends thousands of letters and emails to citizens with multiple requests each. The government has the vision to make these calls to action (CTA) more direct and transparent so that citizens understand precisely what they are being asked to do. To improve the transparency and directness of the CTAs, they must first be classified. However, manually classifying these sentences is a lengthy process that requires a high level of linguistic expertise. This is the primary reason for our desire to find an approach that automatically identifies these CTAs.

1.2 Focus and Scope

Natural language processing (NLP) is a subfield within AI resulting from a century of research in computational linguistics, statistical modeling, and more recent advances in machine learning. NLP capabilities have improved dramatically in recent years as a result of advances in deep learning algorithms and the invention of transformer-based models such as bidirectional encoder representations from transformers (BERT) [20], ELMO [23], and XLNet [65]. Prior to the advancements in transformer models, state-of-the-art models of the time such as LSTM [30] models, had difficulty capturing the true meaning of words. These transformer models may theoretically be effective for detecting CTAs as they are a type of text classification that is heavily context dependent. Therefore, the focus of this study is on machine learning methodologies.

NLP techniques have been primarily developed in the English language. However, the focus of this study is detecting CTAs in Dutch-language letters and emails sent by the government to citizens. The majority of advances in NLP are biased toward English. As a result, additional steps must be taken to ensure the use of these state-of-the-art transformer models.

Despite extensive research into speech act recognition in a variety of fields, developing CTA recognition for Dutch-language letters and emails sent by the Dutch Tax Administration is challenging. A major challenge is that emails and letters usually have no labeled data for training statistical speech act recognizers [31]. Labeled data is available in other domains such as meeting and telephone conversations. However, this data is not labeled for directness and is presented in a different language (English). In addition, the domains are too dissimilar in terms of language use to that on which this research is conducted. Therefore, the only data used is that made available by the tax authorities. The project's scope is limited to detecting CTAs and not on making the CTAs clearer and more transparent.

1.3 Objective

Our objective is to develop a method for recognizing CTAs in Dutch-language letters and e-mails.

1.4 Formalization

Given a document $D = \{x_1, x_2, ..., x_n\}$ where x_i refers to a text segment, each text segment x_i is associated with a class label $y \in Y = \{1, 2\}$, such that $1 = CTA$ and $2 = other$. The objective is to find a model f, with a prediction function $f(x) = \hat{y}$, such that $\hat{y} \in Y$, and preferably $y = \hat{y}$.

Area under the curve (AUC), accuracy, running time, and F1-score are four classification metrics we use to assess the model's performance. The accuracy measure provides a percentage of how many samples were correctly classified out of the entire dataset, and the F1-score is an aggregate metric that takes into account the precision and recall of the model. The AUC metric shows how well the model performs under various classification thresholds.

1.5 Research Question

The research question for this study is as follows:

Is it possible to identify calls to action in Dutch letters and emails (written by the Dutch Tax Administration) using current state-of-the-art NLP approaches?

1.6 Outline

The remaining parts of this work are broken down into the following Chapters. An examination of the relevant literature is provided in Chapter 2. The materials and the various experiments are broken down in detail in Chapter 3. The findings of the experiments are presented and then

discussed in Chapter 4. In Chapter 5, we summarize and reflect on the research and answer the main research question.

Chapter 2

Literature Review

In the literature review, we first elaborate on the field of NLP and deep learning. Thereafter, we examine weakly supervised learning, which is a subfield of machine learning that deals with inexact, incomplete, and inaccurate data. Next, we discuss several frameworks for text classification based on transformer models and advanced data augmentation. Finally, we summarize our findings from the review.

2.1 Natural Language Processing

NLP is a subfield of AI, which is concerned with teaching computers to understand text and spoken words similarly to how humans accomplish this task. There are many ambiguities in language, making it challenging to create software that accurately assesses intended meaning. For natural language-driven applications to be useful, developers must teach them to recognize and understand sarcasm, idioms, metaphors, grammar, and the like. Several NLP tasks assist the computer to understand human text and voice input. The following are some of these tasks [38]:

- **Speech recognition:** The technique of accurately turning auditory data into text, often known as speech-to-text. Speech recognition is required for any application that responds to verbal requests or commands.
- **Part of speech tagging:** The practice of determining the part of speech of a word or passage of text based on its usage and context is also known as part of speech tagging.
- **Named entity recognition:** (NER) refers to the process of classifying words or phrases as entities. For instance, NER recognizes the word "*Eindhoven*" as a geographical location.
- **Text classification:** alternatively referred to as text tagging or text categorization, divides text into specified classes. Text classifiers can analyze text automatically and then categorize it using a set of predefined categories or tags. In general, text classification can be applied at one of four levels: document, paragraph, sentence, or subsentence.

Over the last few decades, text classification problems have been extensively studied and addressed in a variety of real-world applications [32, 36, 37]. The majority of text classification systems can be broken down into five stages: preprocessing, feature extraction, dimensionality reduction, classifier selection, and evaluation [38]. We depict these five stages as a pipeline in 2.1.

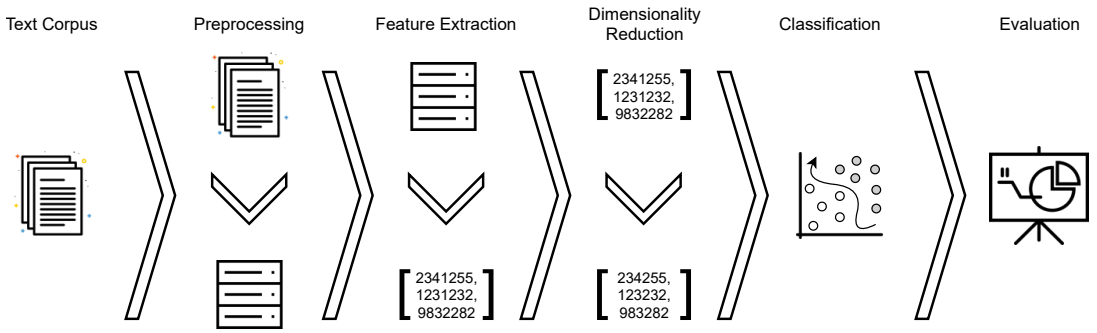

Figure 2.1: Traditional text classification pipeline

Preprocessing: Texts and documents, in general, are unstructured data sets. However, when mathematical modeling is used as part of a classifier, it is necessary to turn these unstructured text sequences into a structured feature space. To begin, the data must be preprocessed into a format that is more structured, so that the machine learning model can understand it. This can be accomplished through the use of techniques such as stop-word removal, lower casing, punctuation removal, stemming, and the like. Subsection 2.1.1 details each of these techniques.

Feature Extraction: Most machine learning algorithms generate output for test data by learning from a predefined set of features in the training data. However, the primary issue with language processing is that machine learning algorithms cannot work directly on raw text. As a result, techniques are required for converting text to a matrix (or vector) of features. Term

frequency (TF) [52], Term frequency - inverse document frequency (TF-IDF) [52], and Word2Vec 2.1.2 are frequently used feature extraction techniques. In Subsection 2.1.2, we discuss these popular techniques.

Dimensionality reduction: Due to the fact that text or document datasets frequently contain a large number of unique words, feature extraction steps can be slowed by excessive time and memory requirements. Dimensionality reduction is a frequently used approach for resolving these issues. Dimensionality reduction, or dimension reduction, transforms high-dimensional data into a low dimension while keeping important features of the original data. Section 2.1.3 discusses the most frequently used techniques for dimensionality reduction.

Classification: The most critical step in the pipeline is selecting the optimal classifier, and the most efficient model for a text classification application cannot be determined without a complete conceptual understanding of each algorithm. In Section 2.1.4, we discuss traditional text classification techniques. Deep learning classifiers are discussed later in Section 2.2.

Evaluation: Numerous techniques exist for evaluating supervised techniques. In Section 2.1.5, we discuss the various methods for evaluating classification algorithms.

2.1.1 Preprocessing

Most text datasets contain numerous superfluous words such as stop-words, misspellings, and slang. Noise and superfluous features can have a detrimental effect on the performance of many algorithms, particularly statistical and probabilistic learning algorithms. Fortunately, numerous methods exist for extracting or correcting these features from text. This subsection discusses these methodologies.

Lowercasing: Lowercasing all data is a straightforward and effective method of text preprocessing. It is applicable to the vast majority of text-mining and NLP problems and can help significantly with expected output consistency in cases in which the dataset is not very large.

While lowercasing is a common practice, there are times when it is critical to maintain capitalization, for instance in a system where abbreviations can be important: the abbreviation “US”, which stands for United States, is distinct from the term “us”. Lowercasing the two makes them identical, resulting in a loss of critical predictive features in the classifier.

Stopword Removal: “Stop-word” is a general term referring to a collection of frequently used words in any language, not merely English and Dutch. Stop-word removal is critical for many applications because it places the focus on important words rather than on words that are frequently used in a given language.

When we performed text classification and sentiment analysis in this study, stop-words had to be eliminated because they added no information to our model; however, when performing language translation, stop-words are useful because they must be translated alongside other words

Stemming: Stemming is the practice of reverting words with inflections (e.g. connect, connected, connection, connections, connects) to their base form (e.g. connect). It is possible that the word is not a true base word, but rather a canonical variant of the original word. There are various stemming algorithms. The snowball stemmer [45] on of the most prevalent algorithm, which is also recognized to be beneficial for the Dutch language.

Tokenization: Tokenization is the process of dividing a text string into small units called tokens. A token can be a single word, a segment of a word, or simply characters such as punctuation marks. Most text classification techniques require the use of a parser to tokenize the documents. Consider

the following sentence: "I enjoy Shoyu ramen!" If we use a word tokenizer to tokenize this sentence, we get the following output: "I," "enjoy," "Shoyu," "ramen," "!".

Lemmatization: Lemmatization is superficially comparable to stemming in that the objective is to delete inflections and transfer a word to its base form. The primary distinction between the two is that lemmatization does not accomplish this by simply cutting things off; it actually converts words to their base form, for instance "better" corresponds to "good." It appears as though lemmatization is superior to stemming, but this is frequently not the case. Depending on the algorithm used, it may be significantly slower than when a very basic stemmer is used, and the part of speech of the target word might need to be known in order to obtain a correct lemma.

Normalization: Text normalization is the process of turning a text into its standard (canonical) form. For instance, the words "2morrow" and "2mrw" can be transformed into the standard form "tomorrow." Another example is the reduction of nearly identical terms such as "data set" and "dataset" to the singular "data set."

Normalization is critical for noisy texts such as text messages, blog comments and social media comments that frequently contain misspellings, abbreviations, and the use of out-of- vocabulary words. Ranjan et al. [53] improved the accuracy of sentiment analysis on Twitter messages by applying only normalization.

Spelling correction: Texts and documents frequently contain spelling mistakes. Numerous methods and techniques are available for this purpose, including spelling correction via the Trie and Damerau-Levenshtein distance bigram [11] as well as hashing-based and context-sensitive spelling correction [22].

Noise removal: Noise removal is the process of removing digits, characters and fragments of text that might obstruct the text analysis. Noise removal is a critical step in the preprocessing of text, and it is highly domain specific.

Text augmentation: A way to address the issue of limited data is to perform various transformations on the available data in order to generate new data. The act of synthesizing new data from existing data is referred to as "data augmentation."

Data augmentation can be used to address both the requirement for and quantity of training data. Apart from these two applications, augmented data can also be used to address the problem of class imbalance in classification tasks. This is a fairly complex subject that will be discussed in greater detail in Section 2.3.2.

2.1.2 Feature Extraction

A word is naturally represented by a character sequence. However, using raw character sequences to represent words is inefficient and ineffective. First, words with variable lengths are difficult to process in machine learning. Character sequences are also very sparse as only a few arrangements are meaningful. For example, Dutch words are usually composed of 1–15 characters from the Dutch alphabet, but most possible character combinations, for example "qqqqqqq," have no meaning.

Another way to represent words is through one-hot representation, which assigns each word a unique index. However, this is not an appropriate method of representing words because it cannot capture the semantic relationship between words. In addition, one-hot representation is an inefficient, sparse, high-dimensional representation that is extremely inflexible when it comes to dealing with new words, which requires assigning new indexes and altering the representation's dimensions.

Machine learning algorithms require input to be in the form of floats with fixed lengths and dimensions. As a result, representing a word as a vector plays a crucial role. Text must be in

meaningful floats in order to effectively train machine learning models. This subsection discusses various techniques for extracting features and representing words as vectors

One hot encoding: A one-hot is a group of bits in which the only valid value combinations are those that contain a single high (true/1) bit and in which all other bits are low (false/0). Given a vocabulary $V = \{w_1, w_2, ..., w_{|V|}\}$, the main idea is to represent each word w as a unique and distinct one-hot such that all words w have length $|V|$.

Term frequency–inverse document frequency: TF-IDF [48] is a metric used in statistics to determine the relevance of a word to a document in a given corpus. Expanding the one-hot encoding representation is one way this metric works. Since, all words $w \in V$ hold the $|w| = |V|$ constraint, this idea can be extended to represent a given sentence $s = \{w_1, w_2, ..., w_n\}$ with n words as:

$$s = \sum_{i=1}^{n} w_i \tag{2.1}$$

Therefore, the sentence s is the sum of all one-hot representations of all the words in the sentence. Consequently, each element in s represents the TF of the corresponding word, that is the number of times the word appears in the sentence. The issue with this straightforward representation is that it does not take into account the relative importance of various words. For example, words such as "the," "so," and "what" often appear in different sentences and carry little meaning. As a result, the inverse document frequency (IDF) is applied to determine the significance of w_i as follows:

$$idf(w_i, D) = \log \frac{|D|}{df(w_i, D)} \tag{2.2}$$

where $df(w_i, D)$ represents how often the word appears in a given gorpus D. The TF-IDF representation can now be determined by taking the element wise product of s and $idf(w_i, D)$. More formally, the TF-IDF of s can be computed as follows

$$tfidf(s, w_i, D) = s \times idf(w_i, D) \tag{2.3}$$

Word2Vec: Mikolov et al. [42, 41] introduced word-to-vector (Word2Vec) representation as an enhanced word-embedding architecture in which linguistic terms such as "dog" and "cat" are measurably similar. Word2Vec is a shallow neural network with two layers that is trained to reconstruct the linguistic contexts of words. Each unique word in the corpus is allocated a corresponding vector in the vector space, which typically has several hundred dimensions. These word vectors are positioned in the vector space in such a way that words with common contexts in the corpus are close neighbors in the space.

When algebraic operations are performed on these word embeddings, it is possible to obtain a close approximation of word similarity. For instance, the vector of "sir" minus the vector of "man" plus the vector of "woman" yields a vector very close to that of "madam." An example of an algebraic operation on word embeddings is as follows:

$$[5, 3] - [2, 1] + [3, 2] = [5, 4] \tag{2.4}$$

where [5,3] = sir, [2,1] = man, [3,2] = woman, and [5,4] = madam. Word2Vec comes in two versions: skip-gram and continuous bag of words (CBOW). These models are similar in terms of algorithmic complexity. Figure 2.2 illustrates a simple skip-gram model that attempts to find words that may occur near each word, whereas the CBOW model attempts to find a word based on previous words.

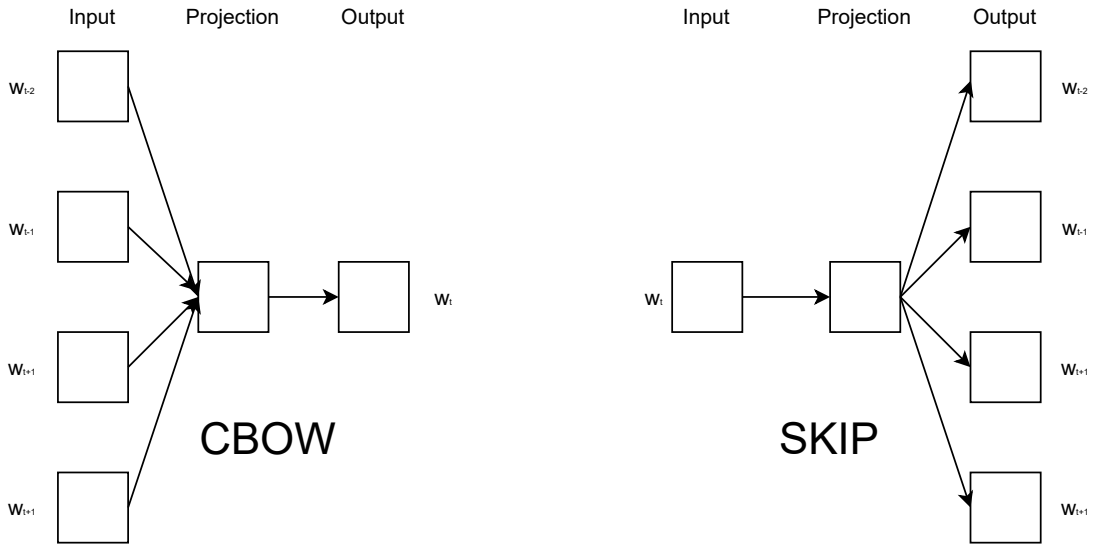

Figure 2.2: The Skip-gram and the continuous bag-of-words (CBOW) architecture.

FastText: One significant disadvantage of word-embedding techniques such as Word2Vec is their inability to handle out-of-corpus words. These embedding techniques consider a word to be the smallest entity and attempt to learn its associated embedding vector. Thus, if a word does not appear in the corpus, Word2Vec will fail to obtain its vectorized representation. FastText [34] considers each word to be made up of n-grams; that is, if the value of n is 3, the n-gram representation of the word "ramen" is "ra," "ram," "ame," "men," "en." For the word "ramen" can be inferred the entire vector as the sum of all the characters' n-gram vector representations.

For instance, suppose a word commonly appears in the testing dataset but does not appear in the training set, but the training set includes a vector representation of each of its n-grams. As a result, the vectorized representations of all the word's constituent n-grams can simply be averaged.

2.1.3 Dimensionality Reduction

There are frequently too many factors in text classification problems on the basis of which the final classification is made. These variables are referred to as features. The more features there are, the more difficult it is to visualize and work with the training set. Occasionally, the majority of these characteristics are correlated and thus redundant. The most frequently used techniques for dimensionality reduction in the text classification domain are principal component analysis (PCA) [50], nonnegative matrix factorization (NMF) [39], linear discriminant analysis (LDA) [5], random projection [47, 15], autoencoders [40], and t-distributed stochastic neighbor embedding (t-SNE) [58].

2.1.4 Classification

From the 1960s to the 2010s, text classification models based on shallow learning were dominant. Shallow learning models, for example support vector machine (SVM) [12], K-nearest neighbor (KNN) [44], and naive Bayes (NB) [51], are based on statistics. In this section, these shallow learning methods are explained. The more novel deep learning methodologies are discussed in greater detail in Section 2.2.

PGM-based methods: Probabilistic graphical models (PGMs) combine probability and graph theory. Bayesian networks [27], NB, and hidden Markov models [57] are three examples of PGMs

that are frequently utilized in practice due to their simplicity. However, these models' simplicity has limitations in some circumstances as they do not perform well on text that requires context. Gaussian NB is an example of a common PGM-based approach in which it is assumed that the values associated with each class follow a Gaussian or normal distribution.

KNN-based methods: All the techniques in this group are based on the KNN algorithm [13], which is a fundamental though often overlooked classification technique in machine learning. It is a subset of supervised learning and is widely used in pattern recognition, data mining, and text classification. It is extensively applicable in real-world circumstances since it is nonparametric, meaning that no underlying assumptions about the distribution of data are made. However, when solving large-scale problems, KNN can be limited by data storage constrains [61, 38].

SVM-based methods: Cortes and Vapnik proposed the SVM [12] to address the binary classification problem in pattern recognition. Joachims [33] applied the SVM method to text classification by representing each text as a vector. Originally, SVM was developed for binary classification tasks. However, many researchers use this technique to solve classification problems involving multiple classes.

To obtain more accurate results, SVMs can be stacked on top of one another; this is referred to in the literature as stacking support vector machines (SSVM). As illustrated in Figure 2.3, the SSVM model employs a hierarchical classifier with multiple layers.

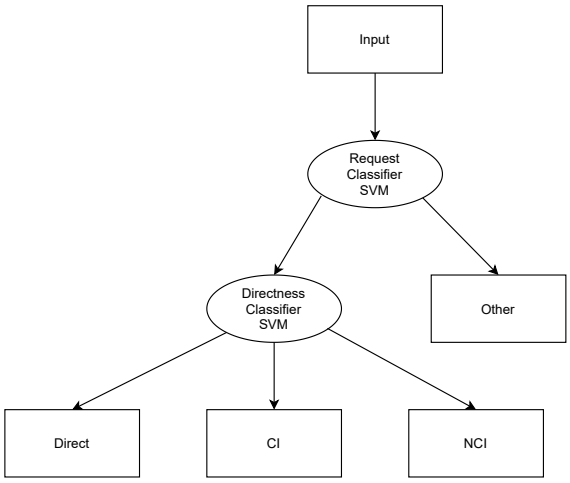

Figure 2.3: Stacking support vector machines

DT-based Methods. The decision trees (DT) based methodologies [43] constitute a recursive supervised tree structure learning method. The basic idea is to create a tree for categorized datapoints based on their attributes. However, the primary difficulty with a DT is determining which feature or attribute should be at the child level and which should be at the parent level of the tree. To address this primary difficulty, De Mántaras [17] introduced statistical modeling methodology for feature selection. The DT-based methods are quite fast, but they are particularly sensitive to slight changes in data and are prone to overfitting [25].

A random forest (RF) [28] is an approach that is comprised of many decision trees. The number of decision trees is a metric that you select based on your data and the surrounding context. Each decision tree is trained on a random subset of data. By ultimately integrating these distinct and hence 'random' decision trees, you avoid outliers from having a (bad) impact on the prediction's outcome and its repercussions.

Regression-based methods When the output variable is a continuous or real value, such as "salary" or "weights," a regression problem may exist. There are numerous available models, the simplest being linear regression. It attempts to match data with the optimal hyperplane that traverses the points.

Using logistic regression (LR), you can develop a predictive model to estimate the likelihood of a positive outcome for a categorical dependent variable. This is possible with one or several independent variables. Here, the categorical variable is often dichotomous, but a variable with more than two categories is also feasible. This means, for instance, that you forecast the likelihood of a positive outcome.

2.1.5 Evaluation

Measurable evaluation metrics are available to compare different methodologies. Due to the fact that the underlying mechanics of these vary, it is critical to understand what each of these metrics represent. Accuracy, precision, recall, AUC, and F-measure are a few examples of these metrics. True positives (TP), true negatives (TN) , false negatives (FN), and false positives (FP) are used to compute these metrics.

Accuracy: The ratio of correctly predicted observations to the total number of observations.

$$accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{2.5}$$

Precision: The ratio of correctly predicted true positives to all true positives.

$$precision = \frac{TP}{TP + FP} \tag{2.6}$$

Recall: The proportion of known positives that are predicted correctly

$$precision = \frac{FP}{FP + FN} \tag{2.7}$$

F1-score: An aggregated metric that takes the model's precision and recall into consideration.

$$F1 = \frac{2 \times (precision \times recall)}{precision + recall} \tag{2.8}$$

Receiver operating curve (ROC): An ROC is a graph that illustrates the performance of a classification model across all classification thresholds. This curve depicts the relationship between recall and precision. By lowering the classification threshold, more items are classified as positive, leading to an increase in both true positives and false positives.

Area under the ROC curve (AUC): AUC is an aggregated metric that measures the performance of all possible classification thresholds. More precisely, the entire two-dimensional area under the ROC curve is measured by the AUC. A classification model that has an AUC close to 1 demonstrates a high degree of separability. A model with an AUC close to 0 has the worst possible measure of separability.

2.2 Deep Learning

As discussed in the previous section, traditional models require good sample features to use them for classical machine learning algorithms. As a result, the method's effectiveness is determined largely based on feature extraction. Deep learning, unlike traditional machine learning models,

adds feature engineering into the model-fitting process. This is accomplished through the use of a set of nonlinear transformations that help directly map features into outputs. A typical pipeline for deep learning text classification is similar to the traditional one depicted in Figure 2.1. However, the deep learning pipeline can perform the steps of feature extraction, dimension reduction, and classification in a single step. This is visualized in Figure 2.4.

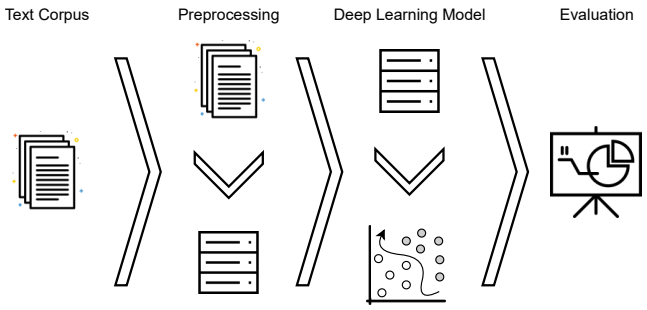

Figure 2.4: Standard, deep learning pipeline for text classification

These deep learning models have delivered state-of-the-art results in many domains, including a wide variety of NLP applications. Prior to deep learning, many conventional machine learning methods required handcrafted feature extractors built by experts with domain knowledge. Deep learning for text and document classification spans multiple architectures. The most important architectures are described in this subsection.

DNN-based methods: Deep neural networks (DNNs) are designed to learn through multiple layer connections, with each layer receiving connections only from the previous layer and providing connections only to the next layer. The input layer may be constructed using Word2Vec, FastText, TF-IDF or another method for obtaining features. Figure 2.5 illustrates the structure of a typical DNN. For multi-class classification, the output layer is equal to the number of classes; for binary classification, it is equal to one. Given a set of example pairs (x, y), the objective is to learn a function $f : x \rightarrow \hat{y}$, such that $\hat{y}$ is a probability score that ideally aligns with y. In a neural network setting, f represents the whole neural network, which contains the network parameters (weights and biases of the network). During the training process these network parameters are tuned using, for example, stochastic gradient descent (SGD).

RNN-based methods: The recurrent neural network (RNN) is an additional neural network design utilized for text classification and mining by researchers. A RNN provides greater weights to preceding datapoints in a sequence. Therefore, this network is an effective way for classifying sequence, string, and text data. RNNs are able to conduct more precise semantic analysis because they take into account the data from the nodes that came before them in an effective manner. RNNs for text classification mostly works by using gated recurrent unit (GRU) or long short-term memory (LSTM). Figure 2.6 depicts a standard RNN, which contains an input layer, hidden layers, and an output layer. Despite these advantages, RNNs are susceptible to the exploding gradient and vanishing gradient problems [6].

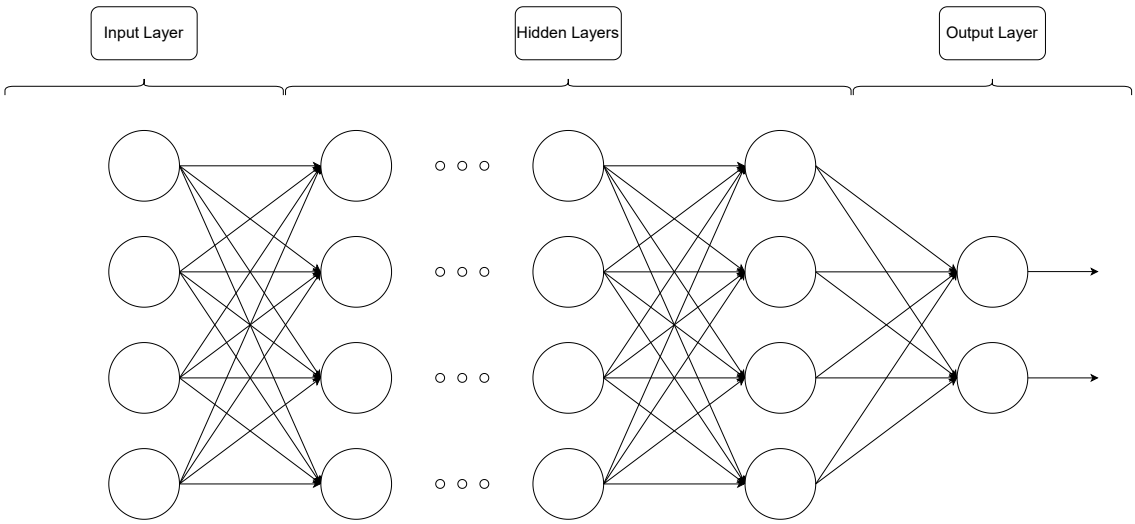

Figure 2.5: Standard, (fully connected) deep neural network (DNN)

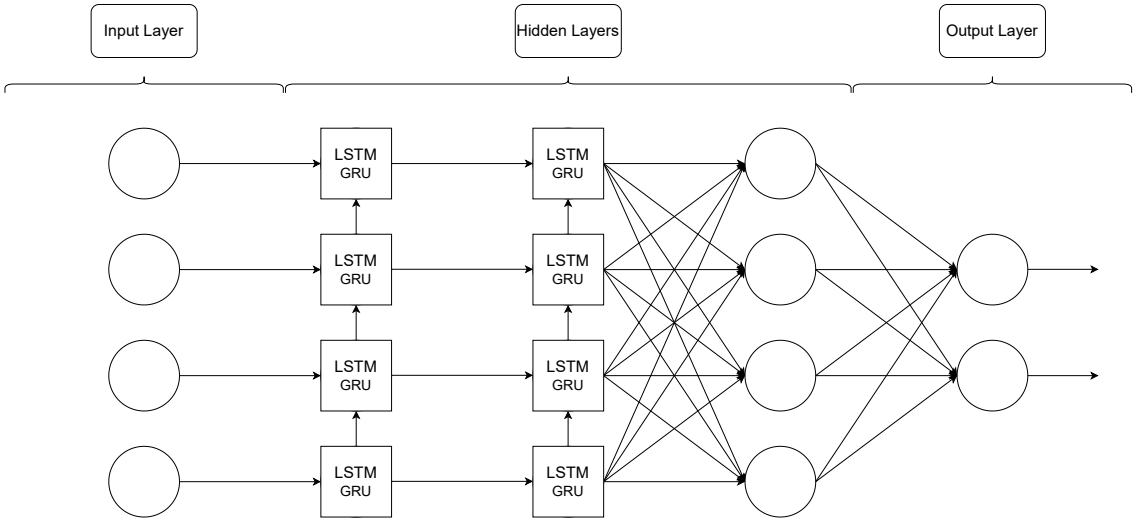

Figure 2.6: Standard, GRU/LSTM network

CNN-based methods: Convolutional neural networks (CNNs) [1] employ convolution filters and where initially proposed for image classification to extract characteristics from images. However, CNNs can also be used for a variety of NLP applications, including text classification. CNNs are distinguished by a sequence of convolutional layers followed by nonlinear and pooling layers. The convolutional layer generates the next feature map by performing a series of convolutional operations on the input feature maps using a learnable filter. These learnable filters enable the CNN to learn local patterns within the feature maps. Then, by extracting the most significant characteristics from a feature map, a pooling layer can be utilized to reduce the dimensionality of the feature map. Finally, a fully linked layer is frequently used to predict the category as a vector. Figure 2.7 depicts the architecture of a typical CNN.

BERT-based methods: In 2017, Vaswani et al. [62] invented a new sort of neural network design called a transformer. The transformer architecture was initially developed to solve the language translation problem. This model consists of an encoder-decoder structure combined

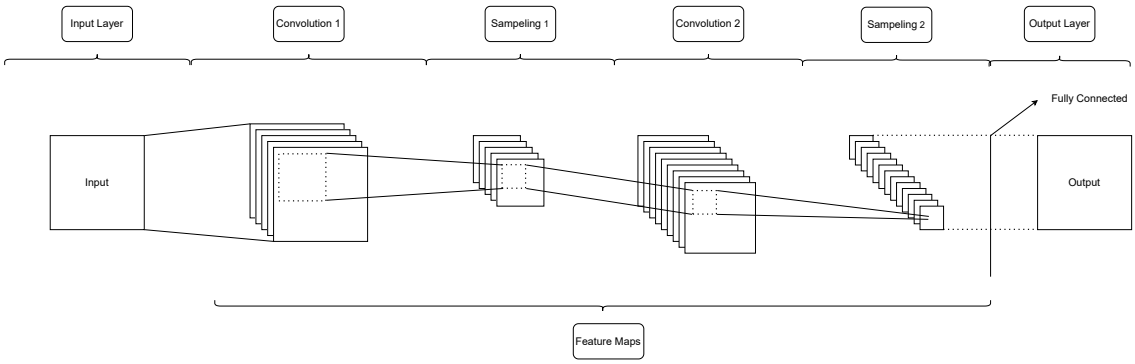

Figure 2.7: Standard, Convolutional Neural Network (CNN)

with a multihead self-attention mechanism, and it obtained state-of-the-art results, outperforming existing CNN- and RNN-based methods. A significant advantage of this model (over other neural networks) is that it considers a word's extended context in a more computationally efficient manner [62, 46].

The transformer's architecture is depicted in Figure 2.8. The encoder on the left side of the transformer's (highlighted in green) architecture is responsible for mapping an input sequence to a sequence of continuous representations, which is subsequently given to a decoder. On the right side of the architecture (highlighted in red), the decoder receives the encoder's output along with the decoder's output from the previous step to form an output sequence.

Consider a scenario in which a Dutch text is to be translated into English with the use of a transformer model. The encoder's task is to take the Dutch words and convert them to word embeddings. Then, the decoder takes these Dutch word embeddings from the encoder along with the previously generated English words from the decoder and uses them to predict the next English word. The decoder predicts the English words one at a time until the end of the phrase is predicted.

Abstractly, the encoder learns Dutch, while the decoder learns the relationship between Dutch and English. Both the encoder and decoder have an underlying understanding of language. Because of this, the transformer architecture can be disassembled to create new models. BERT is fundamentally a language transformer model with changeable encoder layers and self-attention heads. The architecture is nearly identical to the implementation of the original transformer in Vaswani et al [62].

Two models exist for the original English-language BERT: [20] $\text{BERT}_{\text{LARGE}}$ contains 24 encoders with 16 bidirectional self-attention heads, and $\text{BERT}_{\text{Base}}$ contains 12 encoders with 12 bidirectional self-attention heads. These models are depicted in figure 2.9. Both models are pretrained on unlabeled data gathered from the 800 million-word BooksCorpus[67] and the 2.500 million-word English Wikipedia [2].

BERT was pretrained on two tasks: next sentence prediction and language modeling. It acquires contextual embeddings for words during the training process. Following a computationally intensive pretraining phase, BERT can be fine-tuned using fewer resources on smaller datasets to optimize its performance on specific tasks [67].

As an alternative to fine-tuning, the document representation provided by the output layer of the entire network can be extracted, and then the classification task using classical machine learning classifiers can be performed. This allows comparing the performance of the BERT neural network model when used as a classifier versus a feature extractor.

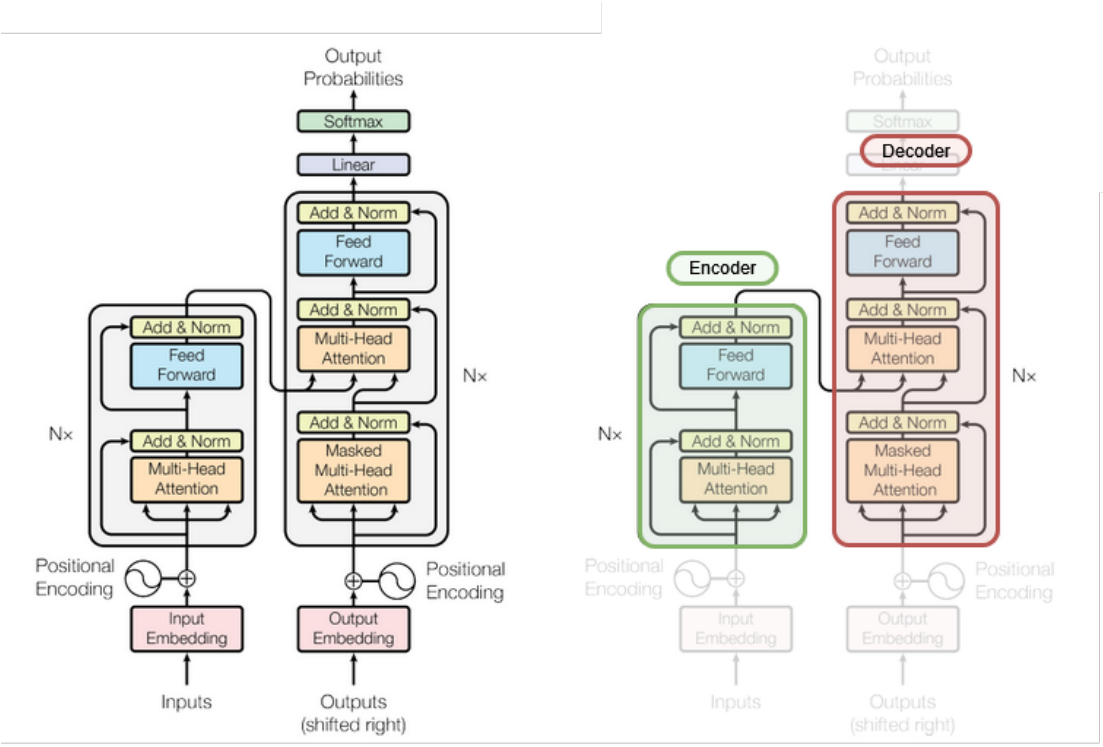

Figure 2.8: The original transformer architecture [62]

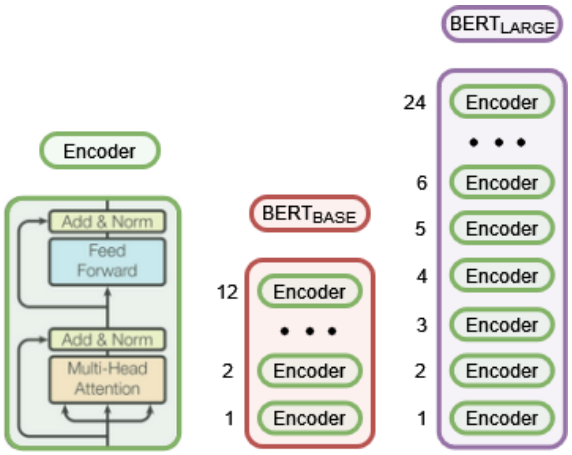

Figure 2.9: Architecture of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$

Because BERT is trained on a English corpus, its application to a Dutch corpus is dubious. To address this issue, BERT has been retrained on 104 different language corpora and named mBERT [20]. It was specifically trained on Wikipedia articles with a shared vocabulary across all languages. Small languages were oversampled and large languages were undersampled to combat the imbalanced content of Wikipedia. Dutch is one of the 104 different languages.

BERTje [18] is a BERT model trained exclusively on a dutch corpus of 2.4 billion words. It was trained using a variety of genres to ensure that it is more representative of general Dutch language use. The model was trained on Wikipedia text as well as fiction novels, Dutch news articles, and articles from a multigenre reference corpus. The resulting model outperformed mBERT on downstream NLP tasks such as NER, sentiment analysis, and part-of-speech (PoS) tagging.

2.3 Beyond Supervised Learning

A large number of training examples are used to construct predictive models using supervised learning techniques. Each training example has a label indicating the output of the ground truth. Despite the fact that current supervised methodologies have achieved considerable success, it should be noted that, in many tasks, strong supervision information is difficult to obtain. Therefore, it is desirable for machine-learning approaches to be able to operate with limited (weak) supervision. Zhou [66] distinguishes between three types of limited (weakly) supervised learning techniques, which includes inaccurate, incomplete and inexact supervision. Each of the three types is explained in detail below.

2.3.1 Inexact

Inexact supervision is based on training models that include some supervision information but not quite enough. A typical problem is when only coarse-grained label information is available, such as in the multiple-instance learning (MIL) classification problem [21]. This classification problem is defined as follows: consider a bag B which contains n instances $i_1, ..., i_n$, such that B is negative if all instances of B are negative and B is positive if at least one instance in the bag is positive. The problem is that the dataset is labeled at a bag level rather than at an instance level, making this an inexact, weakly supervised problem. MIL-classification problems are typically solved in one of two ways: aggregating the instance-level predictions or constructing a bag-level feature representation for classification.

2.3.2 Incomplete

Incomplete supervision refers to a situation in which there is an insufficient quantity of labeled data to train a functional learner, despite the availability of large amounts of unlabeled data. More formally, the objective is to learn a function $f : X \to Y$ from a dataset $D = \{(x_1, y_1), \ldots, (x_l, y_l), x_{l+1}, \ldots, x_m\}$, where there are l number of labeled training examples and $u = m-l$ number of unlabeled instances.

Incomplete supervision encompasses two primary techniques: active and semisupervised learning [55]. Active learning requires the presence of a human so that the machine learning algorithm can query the human to obtain labels for the unlabeled data. On the other hand, semisupervised learning requires no human intervention; the algorithm attempts to learn from both labeled and unlabeled data

Unsupervised data augmentation (UDA): In general, the more data available, the higher the performance will be. However, annotating a large amount of training data is far too time consuming. As a result, proper data augmentation is necessary to improve the model's performance. Data augmentation has shown great potential in reducing the need for more labeled data, although it has thus far been deployed predominantly in supervised situations with modest success. By augmenting labeled and unlabeled data, UDA [64] works as a semisupervised learning strategy that provides state-of-the-art outcomes on a number of visual and language tasks.

MixText: MixText [10] is a semisupervised text classification method that makes use of the newly developed data augmentation technique TMix. By interpolating text in hidden space, TMix generates a large number of augmented training samples. In addition, it utilizes current developments in data augmentation to produce labels with low entropy for unlabeled data. This enables the usage of unlabeled things as labeled data. MixText greatly surpasses state-of-the-art semi-supervised and fully supervised learning approaches, such as UDA and BERT, on multiple benchmarks for text classification. The improvement is most noticeable when supervision is severely restricted. For example, the MixText model, which was trained using only 20 labeled examples form the IMDb dataset, outperformed the BERT model, which was trained using all $25,000$ labeled examples [10].

2.3.3 Inaccurate

Inaccurate supervision refers to a situation in which the supervision data is not always accurate. A typical problem is when an algorithm relies on data labeled by multiple workers: the same input by two or more workers could potentially be labeled differently due to a difference in human judgment. In practice, the basic idea is to identify mislabeled samples and then attempt to rectify the situation by either relabeling or removing the sample.

2.3.4 Unsupervised

Unsupervised learning, commonly referred to as unsupervised machine learning, uses machine learning algorithms to evaluate and cluster unlabeled information. The goal of these algorithms is to find hidden patterns or data clusters without requiring human intervention. Their capacity to identify similarities and contrasts in data makes them a good choice for exploratory data analysis. There are three primary applications for unsupervised learning models: dimensionality reduction, clustering, and association. In this subsection, only unsupervised clustering algorithms are discussed.

Clustering: Clustering is a technique for data mining that organizes unlabeled data based on their similarities or differences. Several types of clustering algorithms exist including overlapping, exclusive, hierarchical, and probabilistic. Exclusive clustering is a method of grouping in which a datapoint can reside in only one cluster. This method is also known as hard clustering. K-means [3] clustering is a frequent instance of an exclusive clustering approach in which datapoints are assigned to k groups, where k is the number of clusters depending on the distance from each group's centroid. The datapoints closest to a specific centroid are grouped into the same category. A greater k-value indicates smaller groupings with greater granularity, while a lower k-value indicates larger groupings with less granularity.

The issue with K-means clustering is that the algorithm's user does not know beforehand how many clusters exist in the dataset. This emphasizes the need to execute K-means multiple times to determine which k clusters the dataset the best. Evaluation criteria such as the silhouette score are crucial for achieving this goal.

The silhouette coefficient quantifies how similar an object is to its own cluster (cohesion) relative to other clusters (separation). The silhouette varies from -1 to $+1$, with a high value indicating that an object is well suited to its own cluster but poorly matched to nearby clusters. If the majority of objects have high values, the clustering design is suitable. If a large number of points have low or negative values, the clustering configuration may contain too many or too few clusters. The abstract formulas below illustrate how the silhouette coefficient $s(p)$ is calculated for a given point. After computing the silhouette coefficient for each point, averaging them out results in the silhouette score.

$$a(p) = \text{The average distance of point } p \text{ from all other points in the same clusters} \tag{2.9}$$

$$b(p) = \text{The average distance of point } p \text{ from all the points in the closest cluster to its cluster} \tag{2.10}$$

$$s(p) = \frac{b(p) - a(p)}{max(b(p), a(p))} \tag{2.11}$$

2.4 Part-of-Speech Taggers

Until now, we have addressed machine learning models whose ultimate objective is to recognize CTA in text. However, there is also a language method for identifying CTA. For instance, a

rule-based system can be created to determine whether particular words are included. Sentences containing action verbs such as "send," "click," or "download" in e-mail traffic have a high likelihood of containing CTA. Depending on the corpus and the number of rules that have been established, creating an algorithm based on words and their parts of speech may provide acceptable results. However, this requires a precise PoS tagger and the linguistic competence to establish the rules. In this section, we restrict the discussion to PoS taggers that can be implemented in rule-based systems.

A PoS tagger identifies the function of words in a particular language by assigning them to one of several categories. In English, there are nine different parts of speech: conjunction, preposition, interjection, adjective, pronoun, adverb, verb, article, and noun. This section discusses popular Dutch PoS taggers.

Frog: Frog [7] is an integration of NLP modules developed for Dutch. It performs tokenization, PoS tagging, lemmatization, and morphological segmentation of word tokens. The PoS tagger employs the tag set of Corpus Gesproken Nederlands (CNG). The PoS tagger in Frog is based on a memory-based tagger-generator and tagger (MBT) [14] trained on a large Dutch corpus of 10,975,324 words in 933,891 sentences. This corpus constitutes a mix of several manually annotated corpora, but about 90% of the data comes from the transcribed CNG of about nine million tokens [59].

SpaCy: SpaCy [29] is an open-source software package for NLP that includes prebuilt pipelines with Dutch PoS taggers. The PoS tagger employs the tag set of CNG; it is trained on the Dutch UD Lassy small [60] dataset containing 6,641 sentences and uses FastText as a backbone. SpaCy provides multiple pretrained pipelines featuring PoS taggers; however, nl_core_news_lg is the most accurate pipeline as of now.

Other: A PoS tagger classifies a word in a sentence; therefore, the pipelines discussed in Section 2.3 can also be utilized to develop a custom PoS tagger. For example, the creators of BERTje also benchmarked BERTje as a PoS tagger and achieved state-of-the-art results on the UD Lassy small dataset [60].

Chapter 3

Materials and Methods

In this chapter, we first elaborate on the datasets used to train various models to detect CTAs. Thereafter, we examine the methodologies that can be used to detect CTAs. Finally, we present the hardware and software packages used to train the presented models.

3.1 Materials

This section provides a detailed description of the datasets used in the experiments described in Section 3.2.

3.1.1 GDC

GDC is a letter generation system used by the Dutch Tax Administration. The GDC dataset consists of all possible template letters that can be generated with the XML-based building blocks of the GDC system. After personalisation these letters can be sent to Tax Service customers in a massive process. Because of the lack of labeled data for this dataset at the start of the project, a small percentage of it was manually labeled during the course of the project. To ensure the most equitable selection of labeled data, an unsupervised clustering method was used. Following that, samples were collected from these clusters. Section 3.2.1 describes the rationale and methodology behind this clustering. This dataset for training and validating the models consists of 1233 sentences and we use an independent test set of 248 sentences, see table 3.2

Table 3.1: Number of sentences in the GDC dataset used for training, validating and testing.

Dataset	CTA	Non-CTA	Unlabeled
Train/validation	621	621	0
Test	124	124	0
Rest	0	4975	476856

3.1.2 GDB

GDB is a document management system of the Dutch Tax Administration which has been developed for the purpose of order registration, development of (re)productions, (web)publication, distribution and management of documents for a broad range of Tax Service processes. It contains standard documents for external and internal purposes, among which letters, forms and attachments, often containing detailed information about fiscal matters. Additionally, the dataset is labeled using the same methodology as for the GDC dataset. This dataset consists of 4284 sentences and we use an independent test set of 856 sentences, see table 3.2.

Table 3.2: Number of sentences in the GDB dataset used for training, validating and testing.

Dataset	CTA	non-CTA	Unlabeled
Train/validation	2142	2142	0
Test	428	428	0
Rest	0	9159	2237104

3.1.3 BD

The BD dataset contains the content of the Dutch Tax Administration's web pages. Due to the small size of the dataset, clusters were manually selected. Although the BD dataset is extremely small, it is of high quality. This is because it has been labeled by a panel of linguistic experts during a workshop given at Utrecht University. This small dataset consists of 54 sentences and we use an independent test set of 12 sentences, see table 3.3.

Table 3.3: Number of sentences in the BD dataset used for training, validating and testing.

Dataset	CTA	non-CTA	Unlabeled
Train/validation	27	27	0
Test	6	6	0
Rest	0	140	0

3.1.4 UD Dutch Lassy Small

This corpus contains sentences from the Lassy Small treebank's Wikipedia section [60]. A treebank is a parsed text corpus that contains annotations for semantic or syntactic sentence structure in linguistics. The Lassy Small treebank is a manually checked Dutch treebank annotated with phrasal nodes and dependency labels. The dataset contains 16 universal PoS tags and is already split into test, train, and validation sets. This dataset is used to train and evaluate a PoS tagger. As not all content in the Lassy Small treebank may be freely accessible, UD Dutch Lassy Small only contains the content from Wikipedia.

Table 3.4: Sentences from the The Lassy Small Treebank's Wikipedia section [60]

Dataset	Train	Validation	Test
UD_Lassy	6641	350	350

3.2 Methods

This section describes in detail how the data labeling technique and text classification methodologies are carried out.

3.2.1 Equitable Selection

Because only a tiny portion of the dataset is labeled, it is crucial that the labeled portion accurately represents the full dataset. If the dataset is randomly selected, there is a high likelihood that not enough of each letter type will be included. As a result, it was decided to cluster the datasets unsupervised so that samples from each cluster could be selected and labeled to ensure that the sub-dataset is as representative as possible. It is critical to note that speed and simplicity take precedence over accuracy (given the size of the datasets and limited time of the project).

Preprocessing: Each text document first goes through a preprocessing phase in which lowercasing, noise removal, and stop-word removal are central. The steps involved in preprocessing are listed below.

- Documents with fewer than 100 characters are excluded from the dataset.
- A replace command is used to remove extraneous white space.
- All words with fewer than two characters are omitted.
- Stop-words are removed from the document. This is accomplished by utilizing the Dutch stop-word list, which is included in the Python NLTK package
- The text is tokenized using the Python NLTK package.
- The tokens are stemmed using the snowball stemmer for the Dutch language.

Feature Extraction: To extract the features of a series of tokens, the TF-IDF metric is used. This metric was calculated using the scikit-learn implementation.

Unsupervised Classification: Due to the limited time available for the project, the size of the dataset, and the availability of computing resources, the dependable and quick K-means++ [3] algorithm was chosen to cluster the dataset. This is performed for all values of k from 2 to 20.

Evaluation: For all k from 2 to 20, the elbow method and the silhouette score are used to determine how many clusters the datasets can be divided into.

3.2.2 Equitable Labeling

After constructing a representative dataset, it is essential that the data be appropriately and accurately labeled. However, there is a significant problem in that the definition of a CTA is ambiguous and therefore subjective. The dataset was labeled by a single individual, so the likelihood of a substantial bias in the labeled dataset is almost certain. To quantify this bias, the BD dataset was labeled entirely by this individual and then again by a group of four experienced linguistic experts, and the results were compared.

3.2.3 Traditional Text Classification

To compare the various feature extractors and classifiers, we created a text classification pipeline in which only the feature extractors and classifiers were modified. This is to ensure that a fair comparison can be made later. This pipeline accepts short texts, typically a sentence, that are either labeled as a CTA or not. Table 3.5 contains the global configurations for these pipelines. To extract features from a series of tokens, we have three options: TF-IDF, Word2Vec, and BERTje.

Table 3.5: Global configurations for traditional text classification pipelines

Step	**Setup**
0: Corpus	GDC GDB BD
1: Preprocessing	Noise removal Tokenization Stemming Stop-word removal
2: Feature extraction	TF-IDF Word2Vec BERTje
3: Dimensionality reduction	N.A.
4: Classification	NB RF LR SVM
5: Evaluation	Accuracy F-Score AUC Running time

BERTje: We use BERtje as a feature extractor for a variety of reasons. Due to the environment's limited computing power, fine-tuning the model takes a long time. However, by using BERTje as a feature extractor, we only need to perform a forward pass through the network and read the output layer with 768 neurons, which requires relatively little computing power. Additionally, this procedure only needs to be performed once, after which the vectors can be saved. As a result, no backpropagation is used in this approach, and BERTje's parameters remain frozen.

In the case of BERTje as a feature extractor, little preprocessing is required. This is because BERTje employs byte-pair encoding (BPE) to reduce the size of its vocabulary, which means that words like "run" and "running" are eventually decoded as "run" + "ing." Additionally, BERTje is a cased model, which means that during training, this pretraining language model encountered capital letters, eliminating the need for lowercasing. Additionally, removing high-frequency words is not needed as BERTje employs attention mechanisms that ensure that the focus is only on important words.

Word2Vec: The CBOW algorithm was used to train the Word2Vec model on the GDC and GDB letters. We ran two variants of the Word2Vec model, one with lowercasing, stemming (snowball), and punctuation removal and one without. Additionally, a vector/feature count of 768 was chosen as this is the vector size used by BERT models. This helps level the playing field between the two models. Additionally, the minimum word count was set to 40; the downsampling parameter was set to 10^{-3}, and the window size was set to 10. The letters were converted to sentences with the help of spaCy's sentencizer and then used to train the model

TF-IDF: Additionally, we convert the documents to TF-IDF features using scikit-learn's TD-IDF vectorizers, and use these as features for the rest of the pipeline.

Classification. We used gaussian NB, RF, and LR as our traditional machine learning classifiers. We used scikit-learn's implementation of all of the above-mentioned classifiers. Utilizing grid searching, the hyper parameters were obtained, all these classifiers' hyperparameters are listed in appendix A.

Evaluation. A five-fold cross-validation was performed, and accuracy, F1-score, running time, and AUC were evaluated.

3.2.4 Deep Learning Text Classification

Although BERTje was used as a feature extractor in the previous method, the entire network can also be used as a classifier with the use of fine-tuning. The global configurations for these pipelines are listed in Table 3.6. Figure 3.1 illustrates the model overview when BERTje is fine-tuned as a text classifier. As depicted, a feed-forward layer is added on top of BERTje, followed by a sigmoid layer to predict the correct class label. By maximizing the log probability of the correct label using binary cross-entropy loss [16], we fine-tuned all BERTje's parameters and the parameters for the feed-forward layer simultaneously. The fine-tuning was performed utilizing the ADAM [35] optimizer. Grid-searching for a good set of hyperparameters was unrealistic due to the absence of computational resources. This is why the spaCy package was used to implement a linear warm-up strategy. All parameters for fine-tuning BERTje and the linear warm-up are listed in appendix A.5. To evaluate the pipeline, a five-fold cross-validation was performed, and accuracy, F1-score, running time, and AUC were evaluated.

3.2.5 Part-of-Speech Tagging

For the development of a rule-based method to identify CTA, it is crucial to have an effective PoS tagger. Frog and nl_core_news_lg are the two primary Dutch PoS taggers currently available in the

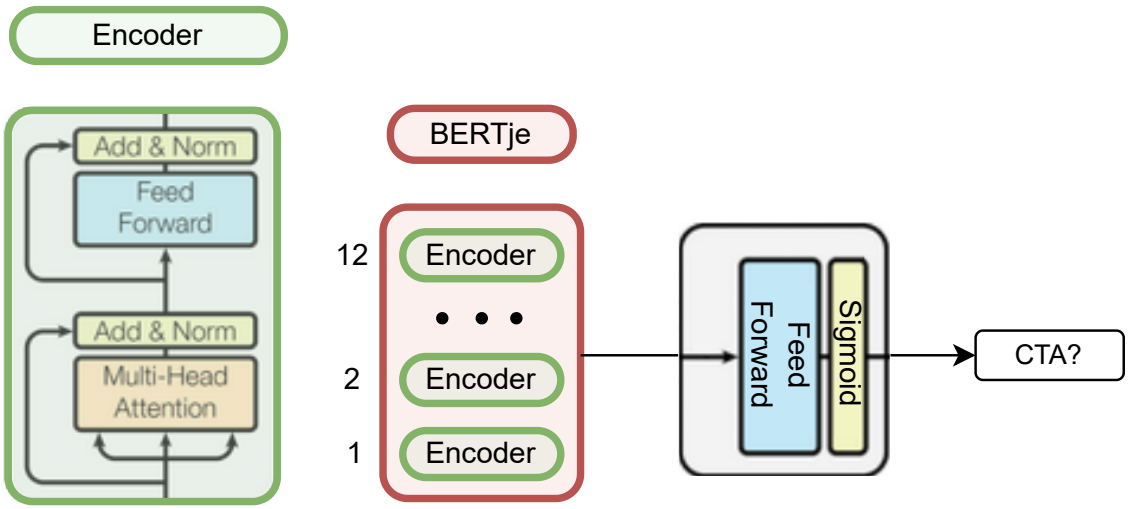

Figure 3.1: Fine-tuning BERTje for the detection of CTAs

Table 3.6: Global configurations for deep text classification pipeline

Step	**Setup**
0: Corpus	GDC CAV GDB BD
1: Preprocessing	Noise removal
2: Classification	BERTje (Unfrozen)
3: Evaluation	Accuracy F-Score AUC running time

department in which the research is being performed. However, both methods utilize non-state-of-the-art algorithms. BERTje has shown promising benchmarks for PoS tagging. As a result, we trained our own custom PoS tagger with BERTje as a backbone. Since PoS tagging is in essence a multiclass classification problem, we can use the same pipeline created in the previous section to train a PoS tagger. The main difference is that we replace the sigmoid function in Figure 3.1 with a softmax function. The UD Lassy Small dataset is used to train the PoS tagger, which is then evaluated against other relevant Dutch PoS taggers.

3.2.6 MIXText

Due to the lack of labeled data, a semisupervised method that utilizes both labeled and unlabeled data could outperform the previously mentioned fully supervised methods. In this experiment, MixText was modified to be compatible with the Dutch language. This was accomplished by replacing the MixText backbone BERT with BERTje. MixText uses back-translation to augment data. Due to the limitations of the development environment, we replaced this with easy data augmentation (EDA) [63] techniques. The unaltered code for MixText can be found on https://github.com/GT-SALT/MixText.

3.3 Tools

All code was executed on a shared virtual machine on a very restricted environment with the following specifications:

Table 3.7: Hardware configuration for the shared virtual machine.

Operating System	CPU	GPU	RAM
Linux	Intel(R) Xeon(TM) 12 cores @ 2.80 GHz	None	128GB

All experiments were carried out in Python 3.6. Within Python, the libraries given in table 3.8 were used.

Table 3.8: The Python packages used for replicating this study

Package	Version
spaCy	3.1
scikit-learn	1.0.2
pandas	1.3.5
numpy	1.21.6
nltk	3.7
fairseq	0.12.1
matplotlib	3.2.2
nlpaug	1.1.10
transformers	4.20.1
torch	1.11.0+cu113

Chapter 4

Evaluation

In this chapter, we first present the results of our text classification experiments. Next, we assess the model's performance using the AUC, accuracy, running time, and F1-score metrics, which are widely used in classification tasks. Last, we elaborate on the results within the context of the project's scope.

4.1 Equitable Selection

Figure 4.1 shows the silhouette coefficient for each k from 2 to 20 for the GDB and GDC datasets. The value of k for which this coefficient has the greatest value is deemed the optimal number of clusters for the unsupervised learning algorithm.

For the GDB dataset, there is a upward trend from $k = 2$ to $k = 18$, followed by a downward trend after $k = 18$. Hence, the optimal number of clusters based on this experiment is 18 with a silhouette coefficient of 0.0744.

For the GDC dataset, there is a distinct spike at $k = 17$. In addition, there is a upward trend to $k = 13$, with $k = 8$ being an outlier. Furthermore, there is a declining trend from $k = 13$ to $k = 17$ and from $k = 18$ to $k = 20$, thus indicating that $k = 17$ with a silhouette coefficient of 0.778 is a suitable number of clusters for the GDC dataset.

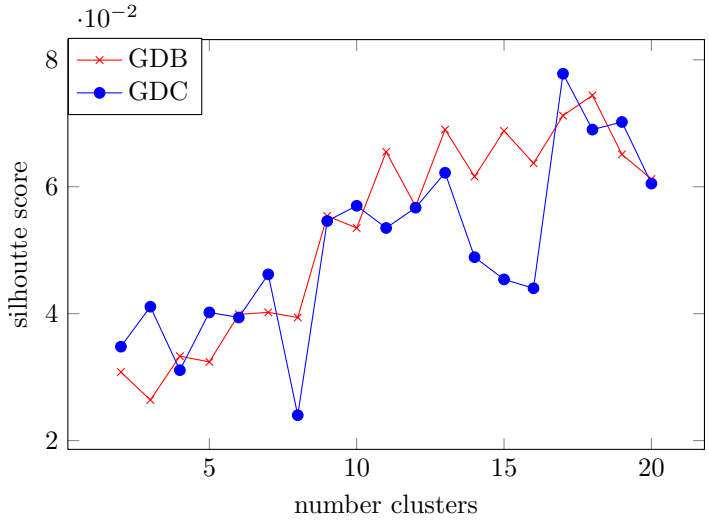

Figure 4.1: Silhouette coefficient of unsupervised clustering for each k from 2 to 20 for the GDB dataset.

We calculated the average vector per cluster and determined which words were most closely associated with a given word. Table 4.1 lists the leading terms for each cluster in the GDB dataset. This table reveals that the top terms for each cluster are expected and have little overlap. However, we observe that the top terms in Clusters 1 and 6 are notably different. Cluster 1 contains English terms as top terms, indicating that the dataset is not entirely composed of Dutch letters as was initially believed. In addition, Cluster 6 still contains words like "textblock," which, according to additional research, are the letters generated by the GDB system that contain XML.

Table 4.1: Leading top terms per cluster for the GDB dataset

Cluster #	Word 1	Word 2	Word 3
1	the	you	to
2	vdp	innovatiebox	dd
3	aftrek	vrag	woning
4	bezwar	brief	gaa
5	aangift	btw	hebt
6	textblock	tekstblok	test
7	douan	goeder	vergunn
8	beslag	belastingdienst	geacht
9	scherm	getoond	formulier

10	greken	rekeningnummer	wij
11	zeker	nederland	vooraf
12	korting	heffing	lon
13	toeslagpartner	toeslag	zorgtoeslag
14	convenant	partij	belastingdienst
15	accijn	accijnsgoeder	olie
16	werkgever	werknemer	verwijs
17	artikel	inkomstenbelast	bedoeld
18	bpm	motorrijtu	bestelauto

Table 4.2 displays the top terms for each cluster in the GDC dataset. This table demonstrates that the top terms are quite logical and have minimal-to-no overlap. In addition, there are no anomalies such as unparsed XML files or the use of other languages.

Table 4.2: Leading top terms per cluster for the GDC dataset

Cluster #	Word 1	Word 2	Word 3
1	clint	blijkt	gegeven
2	kwijtscheld	verlen	clint
3	onderteken	handteken	plat
4	informatie	vrag	belt
5	adres	docvariabl	omzetbelastingnummer
6	beslag	zak	hebt
7	bedrag	vastgesteld	beslagvrij
8	formulier	stur	verzoek
9	uitstel	betal	verschuldigd
10	soort	verzoek	belastingplicht
11	aanslagnummer	openstaand	aanslag
12	fiscal	recht	person
13	bezwar	brief	bezwaarschrift
14	belastingdienst	ondertekenar	belastingdeurwaarder
15	loonheff	aangift	besliss
16	aansprak	ingevolg	afgekocht
17	douan	bpm	brief

The left-hand sides of Figures 4.2 and 4.3 display the silhouette score for each sample per cluster, allowing a visual evaluation of cluster density and separation. Clusters with higher scores have wider silhouettes, whereas less cohesive clusters fall below the average score across all clusters, which is represented by a red, vertical dashed line. We utilized the PCA dimensionality reduction technique to reduce the vectors to two dimensions. This was plotted on a graph to illustrate the density and separation per cluster, where each color represents a distinct cluster, and the colors correspond to the colors used in the left-hand figure. The plot is visible on the right side of Figures 4.2 and 4.3.

Figure 4.2 depicts the graphs associated with the GDB dataset. Clusters 0, 1, 5, 11, 12, and 16 are remarkable in that the silhouette coefficients are quite high and thus have little overlap with the remaining clusters. Furthermore, Cluster 4 is not distinct and has substantial overlap with other clusters since the silhouette coefficient is negative for most datapoints in the cluster. Table 4.1 reveals that the top terms for Cluster 4 are quite generic and thus can be in any Dutch letter, which could cause the overlap.

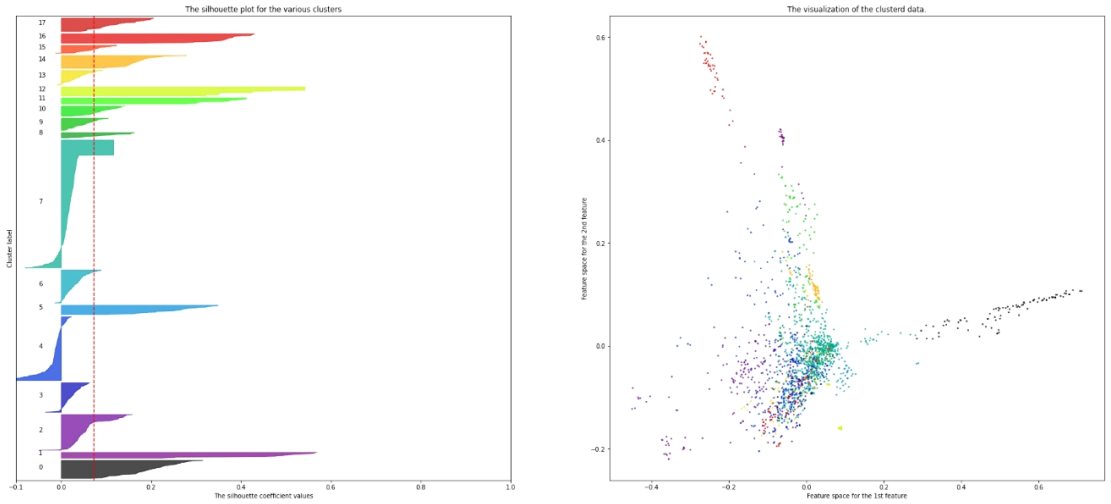

Figure 4.2: Visual representation of the silhouette score of each sentence in the GDB dataset

Figure 4.3 depicts the plots for the GDC dataset. Clusters 1, 2, 3, 4 all score high and thus have little overlap with the remaining clusters, and they are all quite distinct. However, this cannot be said for Clusters 5, 6, 7, and 8. Quite a few clusters overlap with the rest of the data, but in Table 4.2, it appears that the top three terms are indeed quite unique. Therefore, the precise reason that these clusters overlap cannot be determined at this time and requires further study. We do not discuss this further because there are also a large number of datapoints in these overlapping clusters with relatively high silhouette coefficients.

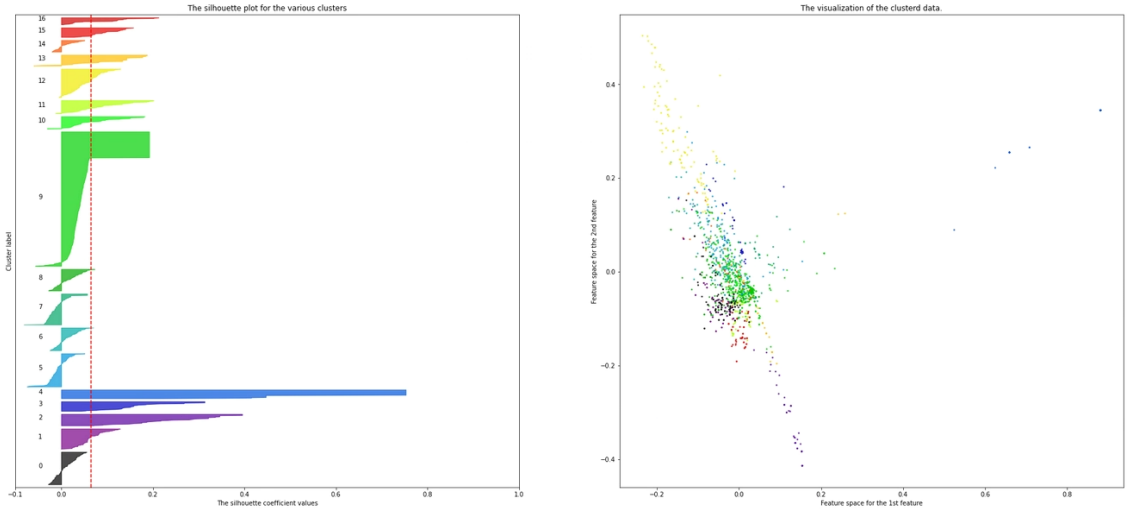

Figure 4.3: Visual representation of the silhouette score of each sentence in the GDC dataset

After unsupervised clustering of both datasets, 1% of each cluster was extracted to produce a subset that accurately and fairly represents the entire dataset.

4.2 Equitable Labeling

A group of four experienced linguists and one other individual labeled the BD dataset separately. During the labeling of the dataset by the linguistic group, there was an extraordinary amount

of debate regarding what constitutes a CTA and what does not. This suggests that the dataset of the linguistic group may contain a substantial amount of bias due to the fact that a CTA is a subjective speech act, thus its definition is quite vague. Therefore, we cannot use the dataset of linguistic expertise as the absolute truth but only as an indication. The linguistic experts identified 35 CTA in the BD dataset containing 208 sentences, while the individual identified 29 in the same set of sentences. The confusion matrix is shown in Table 4.3. It is remarkable that nearly all errors are false positives. The majority of false positives involved more complex and indirect CTA. It is essential to note that this experiment was conducted only after the GDC and GDB datasets were labeled.

Table 4.3: Confusion matrix of the equitable labeling experiment

		Linguists		
		Positive	Negative	Total
Individual	Positive	28	1	29
	Negative	7	172	179
	Total	35	173	208

4.3 Traditional Text Classification

In this section, we elaborate the findings of our experiments using NLP techniques along with classical machine learning classifiers for the prediction task.

TF-IDF: Table 4.4 displays the outcomes of experiments in which TF-IDF was utilized as a feature extractor in conjunction with conventional machine learning classifiers. What is remarkable is how poorly the NB classifier performed in comparison to the other methods. The standard deviation of the NB classifier is also quite high, indicating that the results are highly variable; however, the classifier converges faster than the others. The F1-score and accuracy of the SVM are significantly higher than those of LR, whereas the AUC is lower. This is due to the fact that the SVM classifier in this experimental setup has a high number of false negatives at the expense of good performance on the positive classes. In addition, it is evident that the RF model outperforms the other methodologies with the exception of the time metric.

Table 4.4: Results of the traditional machine learning pipeline with TF-IDF as feature extractor on the GDC, GDB and BD datasets.

Model	Accuracy	F1-score	AUC	Time (s)
TD-IDF + SVM	0.88 ±0.01	0.92 ±0.01	0.95 ±0.01	0.1 ±0.2
TD-IDF + LOR	0.90 ±0.01	**0.94 ± 0.01**	0.92 ±0.01	0.1 ±0.3
TD-IDF + NB	0.81 ±0.03	0.87 ±0.03	0.81 ±0.03	**0.01 ± 0.0**
TD-IDF + RF	**0.91 ± 0.01**	**0.94 ± 0.01**	**0.96 ± 0.01**	0.1 ±0.1

Word2Vec: In Table 4.5, the outcomes using Word2Vec as a feature extractor in conjunction with traditional machine learning models are compared. This again demonstrates that the NB classifier is faster than the other classifiers; however, the model performed significantly worse than the other classifiers on the remaining metrics. Furthermore, LR clearly outperformed the SVM in this instance. In contrast to accuracy and F1-score, the AUC score of the SVM is still relatively high. Thus, the phenomenon observed in Table 4.1, in which the AUC score of the SVM is higher than that of the logistic regression, is reflected to a small degree here. The RF and LR classifier have comparable scores, with the exception of the AUC score, where the RF classifier scored slightly higher. RF also appears to be the best algorithm based on the experiment, but this cannot be determined with certainty due to the 0.1 standard deviation in the AUC score.

Table 4.5: Results of the traditional machine learning pipeline with Word2Vec as feature extractor on the GDC, GDB and BD datasets.

Model	Accuracy	F1-score	AUC	Time (s)
Word2Vec + SVM	0.91 ±0.01	0.95 ±0.01	0.94 ±0.01	0.1 ±0.1
Word2Vec + LOR	**0.93 ± 0.01**	**0.96 ± 0.01**	0.96 ±0.01	0.1 ±0.1
Word2Vec + NB	0.67 ±0.04	0.71 ±0.03	0.79 ±0.04	**0.01 ± 0.00**
Word2Vec + RF	**0.93 ± 0.01**	**0.96 ± 0.01**	**0.97 ± 0.01**	0.1 ±0.1

Table 4.6: Results of the traditional machine learning pipeline with BERTje as feature extractor on the GDC, GDB and BD datasets.

Model	Accuracy	F1-score	AUC	Time (s)
BERTje (FE) + SVM	0.90 ±0.01	0.94 ±0.01	0.94 ±0.01	0.1 ±0.2
BERTje (FE) + LOR	0.90 ±0.01	0.94 ±0.01	0.94 ±0.01	0.1 ±0.1
BERTje (FE) + NB	0.72 ±0.01	0.82 ±0.01	0.69 ±0.01	**0.01 ± 0.0**
BERTje (FE) + RF	**0.94 ± 0.01**	**0.96 ± 0.01**	**0.97 ± 0.01**	0.1 ±0.1

BERTje (FE): Table 4.6 displays the outcomes after applying the BERTje transformer model as a feature extractor alongside some traditional machine learning classification techniques. The NB classifier was the worst performer, excluding running time, as was the case in previous experiments. Compared to the other feature extractors, the standard deviation was quite low this time around. The most notable aspect in this table is that the RF classifier outperformed all other feature extractors, except running time, by a significant margin.

Running times: Table 4.7 displays the amount of time required to convert the datasets to vectors for input to the traditional machine learning classifiers. In light of this, we conclude that the running time of the traditional machine learning classifiers is negligible compared to the time required to extract the features.

Table 4.7: Running times of the feature extractors on the GDC, GDB and BD datasets

Feature Extractor	Time (s)
BERTje (FE)	894
TDF-IDF	2
Word2Vec	1

4.4 Transformer Text Classification

In this section, the results of fine-tuning BERTje as a classifier are discussed and compared to the best performing pipelines from the previous section. The model stopped converging after 44 epochs with a final accuracy of 96% on the test set. This comparison is summarized Table 4.8. Since the model is trained on a CPU, lengthy execution time was to be expected. Despite this, it scored higher on all other evaluation metrics. The most notable difference between this model and others is the significant improvement in accuracy.

Table 4.8: Results of fine-tuning BERTje on the GDC, GDB and BD datasets.

Model	Accuracy	F1-score	AUC	Time (s)
TD-IDF + RF	0.91 ±0.01	0.94 ±0.01	0.96 ±0.01	**0.1 ± 0.1**
Word2Vec + RF	0.93 ±0.01	0.96 ±0.01	0.97 ±0.01	**0.1 ± 0.1**
BERTje (FE) + RF	0.94 ±0.01	0.96 ±0.01	0.97 ±0.01	**0.1 ± 0.1**
BERTje	**0.96 ± 0.01**	**0.97 ± 0.01**	**0.98 ± 0.01**	55010 ±6802

4.5 Part-of-Speech Tagging

In this section, we compare part-of-speech pipeline with BERTje as a language model with other relevant part-of-speech taggers. The model stopped converging after 22 epochs, and Figure 4.9 depicts the accuracy per epoch, with a final accuracy of 96.4. mBERT is a multimodal language that outperforms all other PoS taggers on the UD Lassy dataset, which is surprising given that all other PoS taggers employ underlying models optimized for the Dutch language. We also note that our model scored higher than did BERTje in the original paper. Since the train and validation sets of all scores are identical, we believe it is a matter of choosing the appropriate hyperparameters. The model is trained using spaCy, which employs the linear warum pretraining strategy, beginning with a very low learning rate and gradually increasing it. In addition, we utilized the same Adam optimizer.

Table 4.9: POS tagging on UD Lassy small

Model	Accuracy
nl_core_news_lg [29]	95
Frog [19]	91.7
mBERT [19]	**96.5**
RobBERT [19]	96.4
BERTje [18]	96.3
BERTje (our)	96.4

4.6 MixText

The MixText experiment was terminated because the algorithm took too long to execute on a single CPU as MixText uses unlabeled data and BERTje for semi-unsupervised learning. This means that the entire dataset is used multiple times for training, and only 1% of the dataset is labeled.

4.7 Discussions

The results of the experiments in this chapter are further discussed in this subsection.

4.7.1 Equitable Selection

Using K-means++, we clustered the dataset and analyzed the significance of the cluster. We found that the created cluster is as expected and logically interrupts the dataset, allowing us to sample from this cluster and thus create representative training, validation, and test sets that accurately represent the entire dataset. We also discovered through this process that the dataset contains some English terms/letters, so the option of using a multilingual model like mBERT might be useful since these letters are also within the target domain.

4.7.2 Equitable Labeling

Due to the fact that the GDC and GDB datasets were labeled by a person with limited linguistic expertise, this experiment revealed that there is most likely a substantial bias in these datasets. This can significantly affect the practical application of trained models that detect CTA.

4.7.3 Traditional Text Classifcation

In this experiment, BERTje (FE) and Word2Vec outperformed TF-IDF by a significant margin, and they produced results that are comparable. In contrast to BERTje, Word2Vec has the advantage of being trained on the entire GDC and GDB datasets. We think that BERTje's performance as a feature extractor could be enhanced if it were additionally pretrained on tax authority-specific language. Furthermore, we observed that the bottleneck in terms of performance is not due to the classifiers, but rather to the feature extraction.

4.7.4 Transformer Text Classification

Fine-tuning BERTje produced the best outcomes, excluding running time. The fine-tuning of BERTje took 15 hours on average. However, this does not mean that the model cannot be used since after training, only a forward pass on the network is required, which would yield roughly the same speed as BERTje (FE). A problem could occur, for example, if the entire dataset were labeled since the fine-tuning would take about 1,500 hours, assuming that the training time scales linearly with the size of the dataset.

4.7.5 Part-of-Speech tagger

We trained a BERTje-based PoS tagger, which is now accessible in the client's environment. Frog and nl_core_new_lg are the two additional PoS taggers currently available in the client's environment. Our benchmark on the UD Lassy small dataset demonstrates that our PoS tagger outperforms the other two PoS taggers, but Frog was trained on a much larger dataset. Doing this for our PoS tagger would improve its practical performance in domains other than the UD Lassy small dataset. Furthermore, we see that the multilingual model mBERT outperformed all other models, which is a reason to replace BERTje with mBERT.

Contrary to rule-based software, machine learning and deep learning algorithms it is hard to track back to the previous if and else statement. We refer to this lack of transparency in deep learning as the "black box" [9] problem. This gives reason to investigate a rule-based method using the trained PoS tagger from this experiment.

4.7.6 MixText

Due to the fact that MixText uses augmented labeled and unlabeled data with BERTje as a backbone, the execution time on a CPU was far too long and had to be terminated. After roughly one day, no epoch was concluded.

Chapter 5

Conclusions

In this chapter, we summarize and reflect on the research and results of the experiments. Then, we restate the research question and answer it. Next, we make recommendations for future work on the topic. Finally, we state what new knowledge has been contributed through this project.

In order to detect CTA in letters written by the Dutch Tax Authority, we examined traditional machine learning and deep learning models including Gaussian NB, LR, RF, and SVM in combination with Word2Vec, BERTje, and TF-IDF as feature extractors. In addition to using BERTje to extract features, we fine-tuned the model and obtained promising results. The best-scoring traditional machine learning pipeline consisted of BERTje as a feature extractor and RF as a classifier, receiving accuracy, F1, and AUC scores of 0.94, 0.96, and 0.97, respectively. The average execution time, including the time required to extract the features, was 894.1 seconds.

The best-scoring model was the deep learning classification pipeline with BERTje fine-tuned as its backbone. It scored 0.96, 0.97, and 0.98 on accuracy, F1-score, and AUC, respectively. However, this model's training phase is significantly longer than that of traditional machine learning pipelines.

The data used to train the models is a subset of the total data; we attempted to make the subset as representative as possible by unsupervising the cluster using the K-means++ algorithm. Then, samples were taken from this cluster to obtain representative training, validation, and test sets. Although the subset selection is representative, we cannot conclude that the labeled dataset is of sufficient quality for this research. We learned from the equitable labeling experiment that there is most likely a significant bias in the dataset because a CTA can be quite subjective, and the data was labeled by a single person with little linguistic knowledge.

In addition, we modified the semisupervised MixText methodology by converting the BERT backbone to BERTje and the data augmentation technique to EDA. Due to the lengthy duration of this experiment's execution, we are unable to report any results.

Our primary research question, as defined in Section 1.5, is as follows:

Is it possible to identify calls-to-action in Dutch letters/emails (written by the Dutch Tax Administration) using current state-of-the-art NLP approaches?

The best-performing pipeline was that in which we fine-tuned BERTje; it achieved an accuracy of 0.96, which is significantly higher than random guessing. However, we cannot answer the research question due to the high potential for bias in the labeled dataset. We determined this high potential for bias in the data through the experiment on equitable labeling.

5.1 Future Work

5.1.1 MixText

Due to the lack of computational resources, we were unfortunately unable to run the modified MixText methodology. We recommend doing this in the future since MixText performed well on similar tasks in which the labeled data was severely restricted. For example, the MixText model, which was trained using only 20 labeled examples form the IMDb dataset, outperformed the BERT model, which was trained using all $25,000$ labeled examples [10].

5.1.2 mBERT

We found that mBERT outperformed BERTje for PoS tagging on the UD Lassy small dataset, as shown in Table 4.9. In addition, we observed that Dutch is not the only language in the text corpus; therefore, we suggest loading mBERT into the environment and conducting the same relevant experiments with mBERT instead of BERTje.

5.1.3 Context-Based Approach

Whether or not something is a CTA can depend on the context of the previous or following sentences, so we recommend trying methodologies that take the previous and following sentences into account. Such as an RNN with 3 utterances in context [8].

5.1.4 Rule-Based System

Since the models may be used to detect CTAs in official government correspondence, ethics play a crucial role. Due to the black box problem, it can be challenging to determine which parameters machine learning models employ to classify CTAs. This transparency can be provided by a rule-based system, so we recommend investigating these systems with the PoS tagger that we trained in this study.

5.1.5 Relabeling

Due to the fact that the dataset was labeled by a person with limited linguistic expertise, we concluded from the equitable labeling experiment that the labeled dataset may contain a substantial amount of bias. Therefore, we recommend that a group of linguistic experts relabel the data and rerun the relevant experiments.

5.2 Contributions

To our knowledge, we are the first to investigate CTA detection in Dutch text using deep learning. In this regard, we investigated NLP techniques alongside traditional machine learning algorithms and state-of-the-art deep learning methods.

5.2.1 Text Classification Pipelines

We also set up text classification pipelines for the client to detect CTA. However, they can also be used for any other type of text (multi)classification task, including PoS tagging.

5.2.2 MixText

In addition, we have modified the MixText methodology so that it should work better with Dutch-language text; however, this has not been tested due to a lack of computational resources.

5.2.3 PoS Tagger

We've trained a BERTje-based PoS-tagger that is currently the most advanced PoS-tagger in the client's environment.

Appendix A

Model Parameters

A.1 Logistic Regression

A grid of various parameters was established, and the optimal parameters for each feature extraction method was chosen. The selected parameters for TF-IDF, Word2Vec, and BERTje in combination with logistic regression are displayed in table A.1, and the range of parameters used for grid searching is listed below.

Penalty: L1,L2 and none.

C: 0.1, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75 and 3.0

Table A.1: Parameters for the logistic regression model with TF-IDF, Word2Vec and BERTje as feature extractor.

Parameter	TF-IDF	Word2Vec	BERTje (FE)
Penalty	L2	L1	L2
C	3.0	1.25	2.25
Solver	lbfgs	lbfgs	lbfgs

A.2 Random Forest

The optimal parameters for each feature extraction method were selected from a grid of various parameters. The selected parameters for TF-IDF, Word2Vec, and BERTje in combination with random forest are displayed in table A.2, and the parameter range used for grid searching is detailed below.

Estimators: 100, 500, 1000, 1500, 2000, 2500 and 3000.

Max depth: 8, 16, 32 and none.

Table A.2: Parameters for the random forest model with TF-IDF, Word2Vec and BERTje as feature extractor.

Parameter	TF-IDF	Word2Vec	BERTje (FE)
Estimators	2500	2000	2500
Max depth	none	none	none

A.3 Gaussian naive Bayes

A grid of different parameters was made, and for each method of feature extraction, the best parameters were chosen. The selected parameters for TF-IDF, Word2Vec, and BERTje with Gaussian naive Bayes are shown in table A.3, and the parameter range used for grid searching is shown below.

Variable smoothing: 10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3} and 10^{-2}.

Table A.3: Parameters for the gaussian naive Bayes model with TF-IDF, Word2Vec and BERTje as feature extractor.

Parameter	TF-IDF	Word2Vec	BERTje (FE)
Variable smoothing	10^{-5}	10^{-7}	10^{-9}

A.4 Support Vector Machine

The best parameters for each feature extraction method were selected from a grid of different parameters.The range of parameters used for grid searching is mentioned below, and the chosen parameters for TF-IDF, Word2Vec, and BERTje in conjunction with the support vector machine model are shown in table A.4.

C: 0.1, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75 and 3.0

Table A.4: Parameters for the support vector machine model with TF-IDF, Word2Vec and BERTje as feature extractor.

Parameter	TF-IDF	Word2Vec	BERTje (FE)
C	2.0	0.25	0.5

A.5 BERTje

Table A.5 displays the model parameters for fine-tuning BERTje with spaCy.

Table A.5: Parameters for fine tuning BERTje with spaCy

Parameter	Value
optimzer	Adam.v1
beta1	0.9
beta2	0.999
l2	0.01
dropout	0.1
warmup_steps	250
initial_rate	0.00005

Bibliography

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017. 14

[2] Issa Annamoradnejad and Gohar Zoghi. Colbert: Using bert sentence embedding for humor detection. *arXiv preprint arXiv:2004.12765*, 2020. 15

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006. 18, 23

[4] John Langshaw Austin. *How to do things with words*. Oxford university press, 1975. 2

[5] Suresh Balakrishnama and Aravind Ganapathiraju. Linear discriminant analysis-a brief tutorial. *Institute for Signal and information Processing*, 18(1998):1–8, 1998. 10

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. 13

[7] Antal van den Bosch, Bertjan Busser, Sander Canisius, and Walter Daelemans. An efficient memory-based morphosyntactic tagger and parser for dutch. *LOT Occasional Series*, 7:191–206, 2007. 19

[8] Chandrakant Bothe, Cornelius Weber, Sven Magg, and Stefan Wermter. A context-based approach for dialogue act recognition using simple recurrent neural networks. *arXiv preprint arXiv:1805.06280*, 2018. 37

[9] Davide Castelvecchi. Can we open the black box of ai? *Nature News*, 538(7623):20, 2016. 34

[10] Jiaao Chen, Zichao Yang, and Diyi Yang. Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification. *arXiv preprint arXiv:2004.12239*, 2020. 17, 37

[11] Viny M Christanti, Dali S Naga, et al. Fast and accurate spelling correction using trie and damerau-levenshtein distance bigram. *Telkomnika*, 16(2):827–833, 2018. 8

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 10, 11

[13] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967. 11

[14] Walter Daelemans, Jakub Zavrel, Peter Berck, and Steven Gillis. Mbt: A memory-based part of speech tagger-generator. *arXiv preprint cmp-lg/9607012*, 1996. 19

[15] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of the johnson-lindenstrauss lemma. *International Computer Science Institute, Technical Report*, 22(1):1–5, 1999. 10

[16] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005. 24

[17] R López De Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine learning*, 6(1):81–92, 1991. 11

[18] Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. Bertje: A dutch bert model. *arXiv preprint arXiv:1912.09582*, 2019. 16, 33

[19] Pieter Delobelle, Thomas Winters, and Bettina Berendt. Robbert: a dutch roberta-based language model. *arXiv preprint arXiv:2001.06286*, 2020. 33

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 3, 15, 16

[21] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2):31–71, 1997. 17

[22] Juliusz Dziadek, Aron Henriksson, and Martin Duneld. Improving terminology mapping in clinical text with context-sensitive spelling correction. *Informatics for Health: Connected Citizen-Led Wellness and Population Health*, 235:241, 2017. 8

[23] Sergey Edunov, Alexei Baevski, and Michael Auli. Pre-trained language model representations for language generation. *arXiv preprint arXiv:1903.09722*, 2019. 3

[24] Johan Van Hoorde en Carel Jansen. Direct duidelijk, overheid moet klare taal spreken, May 2018. 2

[25] Christian Giovanelli, Xin Liu, Seppo Sierla, Valeriy Vyatkin, and Ryutaro Ichise. Towards an aggregator that exploits big data to bid on frequency containment reserve market. In *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*, pages 7514–7519. IEEE, 2017. 11

[26] Mitchell Green. Speech Acts. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021. 2

[27] David Heckerman. A tutorial on learning with bayesian networks. *Innovations in Bayesian networks*, pages 33–82, 2008. 10

[28] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995. 11

[29] Matthew Honnibal and Ines Montani. spaCy 3: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2021. 19, 33

[30] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015. 3

[31] Minwoo Jeong, Chin-Yew Lin, and Gary Geunbae Lee. Semi-supervised speech act recognition in emails and forums. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1250–1259, 2009. 3

[32] Mingyang Jiang, Yanchun Liang, Xiaoyue Feng, Xiaojing Fan, Zhili Pei, Yu Xue, and Renchu Guan. Text classification based on deep belief network and softmax regression. *Neural Computing and Applications*, 29(1):61–70, 2018. 6

[33] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998. 11

[34] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016. 10

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 24

[36] Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE, 2017. 6

[37] Kamran Kowsari, Mojtaba Heidarysafa, Donald E Brown, Kiana Jafari Meimandi, and Laura E Barnes. Rmdl: Random multimodel deep learning for classification. In *Proceedings of the 2nd International Conference on Information System and Data Mining*, pages 19–28, 2018. 6

[38] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019. 6, 11

[39] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999. 10

[40] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011. 10

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 9

[42] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 9

[43] Tom Mitchell. Machine learning. 1997. 11

[44] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009. 10

[45] Martin F Porter. Snowball: A language for stemming algorithms, 2001. 7

[46] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. 15

[47] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: replacing minimization with randomization in learning. In *Nips*, pages 1313–1320. Citeseer, 2008. 10

[48] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011. 9

[49] Adolf Reinach and John Crosby. The a priori foundations of the civil law [1913]. 2012. 2

[50] Markus Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303–304, 2008. 10

[51] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001. 10

[52] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988. 7

[53] Ranjan Satapathy, Claudia Guerreiro, Iti Chaturvedi, and Erik Cambria. Phonetic-based microtext normalization for twitter sentiment analysis. In *2017 IEEE international conference on data mining workshops (ICDMW)*, pages 407–413. IEEE, 2017. 8

[54] John R Searle and John Rogers Searle. *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press, 1969. 2

[55] Burr Settles. Active learning literature survey. 2009. 17

[56] Blum-Kulka Shoshana, Juliane House, and Gabriele Kasper. Cross-cultural pragmatics: Requests and apologies. *Grazer Linguistische Studien*, (Heft):349–357, 1989. 2

[57] Antal van den Bosch. Hidden markov models. *Encyclopedia of Machine Learning and Data Mining*, 2, 2017. 10

[58] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 10

[59] Frank Van Eynde, Jakub Zavrel, and Walter Daelemans. Part of speech tagging and lemmatisation for the spoken dutch corpus. In *LREC*. Citeseer, 2000. 19

[60] Gertjan Van Noord, Gosse Bouma, Frank Van Eynde, Daniel De Kok, Jelmer Van der Linde, Ineke Schuurman, Erik Tjong Kim Sang, and Vincent Vandeghinste. Large scale syntactic annotation of written dutch: Lassy. 2013. viii, 19, 22

[61] V Vapnik and A Ya Chervonenkis. A class of algorithms for pattern recognition learning. *Avtomat. i Telemekh*, 25(6):937–945, 1964. 11

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. vii, 14, 15, 16

[63] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019. 25

[64] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation for consistency training. *arXiv preprint arXiv:1904.12848*, 2019. 17

[65] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019. 3

[66] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National science review*, 5(1):44–53, 2018. 17

[67] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015. 15