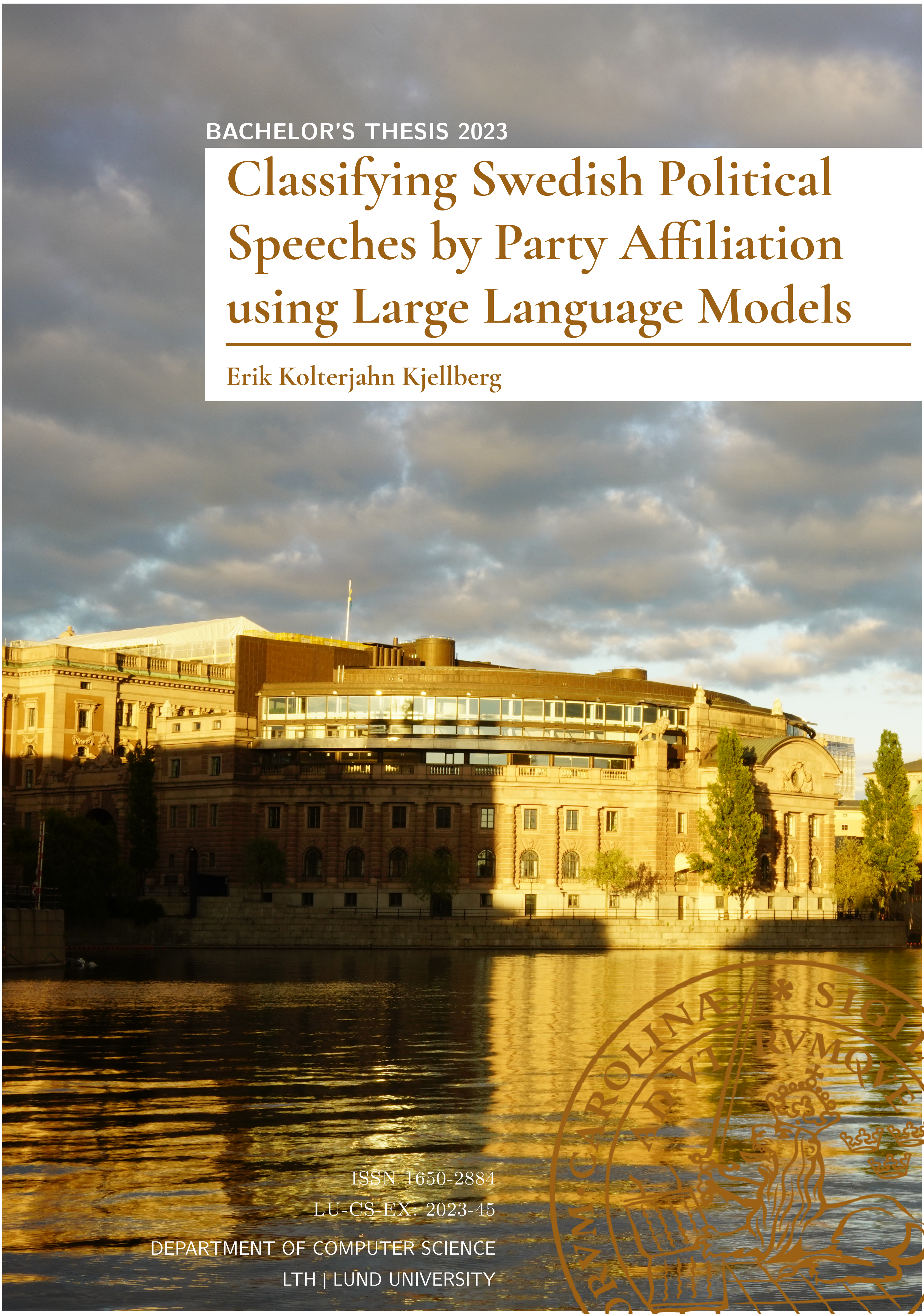


BACHELOR'S THESIS 2023

Classifying Swedish Political Speeches by Party Affiliation using Large Language Models

Erik Kolterjahn Kjellberg

ISSN 1650-2884

LU-CS-EX: 2023-45

DEPARTMENT OF COMPUTER SCIENCE

LTH | LUND UNIVERSITY

KANDIDATARBETE
Datavetenskap

LU-CS-EX: 2023-45

Classifying Swedish Political Speeches by Party Affiliation using Large Language Models

Klassificering av svenska politiska uttalanden per partitillhörighet med hjälp av språkmodeller

Erik Kolterjahn Kjellberg

Classifying Swedish Political Speeches by Party Affiliation using Large Language Models

Erik Kolterjahn Kjellberg
er3165ko-s@student.lu.se

September 21, 2023

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

Supervisor: Marcus Klang, marcus.klang@cs.lth.se

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

Abstract

Analyzing language within Politics and telling the differences between different narratives and opinions is a difficult task. With Large Language Models and large amounts of data, however, the last decade has created a possibility to be able to analyze the semantics of such texts computationally in order to get a well-founded picture of general patterns within different narratives. In this thesis, I use models based on BERT language models combined with recurrent neural networks to classify political speeches in the Swedish parliament by party affiliation. The most advanced model tested, KB-BERT-HAN, combines Swedish BERT word embeddings with a Hierarchical Attention Network (HAN). This network has the ability to put different amounts of attention on different parts of the speech, both in terms of words and sentences. Overall, KB-BERT-HAN performs vastly better than a baseline model based on tf-idf, achieving a macro F_1 score of 68.5% compared to 46.8% for the baseline model. Thanks to its hierarchical structure, it is also possible to visualize the model through the attention on the word and sentence levels. Because of its structure, it can, aside from merely making predictions, give useful information about how narratives belonging to different political affiliations relate to each other.

Keywords: Natural Language Processing, Text Classification, Swedish Politics, Long Short-Term Memory, Hierarchical Attention Network

Acknowledgements

I would like to give a big thank you to Marcus Klang, my supervisor, for the interesting meetings and great support and guidance he has provided throughout the project. During this period I have learned more than I thought would be possible about both the theory and practice of several concepts which I previously was only slightly acquainted with.

I am also thankful for the comments by my examiner Pierre Nugues, which lead to this slightly revised, final version of the report.

I would also like to thank my dear friend and coursemate Erik Danielsson, for following this project with a big interest, giving valuable feed-back and supporting me.

Last but not least, I would really like to take this opportunity to thank my parents Yrsa Kolterjahn and Björn Kjellberg, my sister Hanna Kolterjahn Kjellberg and my girlfriend Elsa Bäcklund, for always being there, supporting me and believing in me, and for pushing me on when I need it the most.

Contents

Chapter 1
Introduction

1.1 Background

This thesis concerns using computational methods to be able to tell different political narratives apart, within the Swedish political context. Political texts are difficult to analyze, partly because they contain a lot of information which may not contribute to the overall meaning of the speech. It is therefore of great value to have a model which is able to find general well-founded patterns in the language of politicians. A model which is dependent on deep structures to make its predictions would have the ability to not only predict correctly, but also tell us important things about the difference in the documents it analyzes. The idea for this project is to construct and evaluate models which can be used to give a picture of how different narratives relate to each other semantically. I do this by analyzing and comparing the model's internal representations of texts.

I chose the Swedish parliament as a source of data, as the Natural Language Processing research done and the language models constructed in Swedish is of a much smaller scale, as compared to English, since it is a smaller language. It is my own first language and I therefore thought it interesting to work with.

1.2 Introduction to the Problem

The problem in question is a text classification problem. The goal of the task is to assign the correct label to a whole document, or in other words, put documents into different classes. In the experiments conducted in this thesis, the documents are transcribed speeches uttered in the Swedish parliament and the classes the political affiliation (the party) of the speaker. An illustration of the input and output of the problem is seen in Figure 1.1.

In order to solve this task, a model needs to in one way or the other distinguish between the semantic content of speeches by different parties, and make the distinction between them

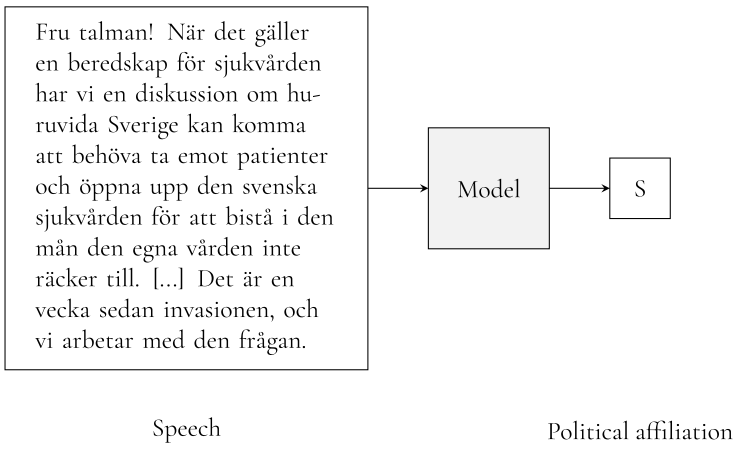

Fru talman! När det gäller en beredskap för sjukvården har vi en diskussion om huruvida Sverige kan komma att behöva ta emot patienter och öppna upp den svenska sjukvården för att bistå i den mån den egna vården inte räcker till. [...] Det är en vecka sedan invasionen, och vi arbetar med den frågan.
Model
S
Speech
Political affiliation

Figure 1.1: An input and output example. The goal of the model is to predict the correct political party.

large enough to be able to tell different parties apart. In order to get a deep understanding of these narratives, I test several models based on neural networks. The idea is that these should be able to identify deep patterns in the language of politicians which help deduce their party affiliation. Most importantly, a Hierarchical Attention Network (HAN)(Yang et al., 2016), being able to put attention on different words and sentences within a document, is trained on this task. Furthermore, the models' internal representations of words and sentences will be based on language models, and more specifically, BERT models. These models have shown state-of-the-art performance in many different NLP tasks (Devlin et al., 2019).

1.3 Purpose and Research Questions

The purpose of the experiments conducted in this thesis is to get an understanding for how well ML models in general, and Deep Learning models in particular, can be developed to be able to distinguish between different political narratives and opinions. A further goal is to learn from the models themselves what differs between narratives. It is of great interest to know on what basis a model makes its predictions, compared to just developing a model which makes good enough predictions, but does nothing more. Are the predictions based on certain words, and are these words meaning-bearing words or not? Is it what different speakers say or the way that they say it that matters? The idea is that a model which makes good predictions also has an understanding of differences and nuances in language which can be used for other tasks, such as getting a statistical overview over the differences in language use of politicians.

I will attempt to answer the following research questions in the report:

(RQ1): How can neural network models based on contextual word embeddings be developed to classify political documents by party affiliation?

(RQ2): How well do these models perform compared to a baseline model based on tf-idf?

(RQ3): Is it possible to explain the models, and if so, how?

1.4 Earlier Work

Prior attempts to classify political documents by party using Machine Learning methods have been made. Dahllöf (2012) predicted the gender, age and political affiliation (left or right) of individual politicians in the Swedish parliament. This study was conducted in the advent of the Deep Learning era and used Support Vector Machines (SVMs). The politician-level party affiliation was predicted using multiple document-level party affiliation predictions from that politician. Since the models predicted left-wing or right-wing as opposed to the party of the politician, it is difficult to compare to the results which will be seen in this report, but overall, the highest recall rates for the document-level predictions were 65% and 66%, for left and right, respectively.

Language models have also been used to try to solve this task. In Doan et al. (2022), the authors classified political documents in Norwegian, German and English by party. They compared baseline methods based on tf-idf to BERT language models. They used the multilingual model *bert-base-multilingual-cased* for all three languages, as well as *bert-base-cased* for English documents, the Norwegian *nb-bert-base* for Norwegian documents and the German *bert-base-german-cased* for German documents. They finetuned these models to train on the specific task. The highest accuracy they achieved on Norwegian speeches was 73%.

In a follow-up article (Doan et al., 2023), the same authors developed a new BERT model, called SP-BERT, which was trained on Norwegian, Icelandic, Danish and Swedish parliamentary speeches, and compared this to other BERT models. It was evaluated on two tasks: 1) classifying the speeches as left or right leaning, and 2) classifying the party affiliation of the speech. Notably, the KB-BERT model, which will be used in this report, and the SP-BERT model were both evaluated in task 2, using Swedish speeches. They got macro F_1-scores of 63% and 62%, respectively.

1.5 Limitations

The data used in this report consists solely of speeches in the Swedish parliament between 1993 and 2022. I have only included speeches from members of parties currently in the parliament. The models are based on BERT embeddings combined with (recurrent) neural networks, and the purpose of the models is only to predict the party affiliation of the person who has uttered the speech, and it is according to this they are evaluated. Apart from this, however, internal states of the most advanced model, KB-BERT-HAN, are also used to visualize attention on words and sentences, as well as how different speeches relate to each other.

1.6 Outline

The rest of the report will be structured as follows:

2. **Theoretical Background:** In this section, I give the necessary background needed in order to understand the models later described. This covers concepts within ML and NLP as well as the frameworks used.

3. **Methodology:** This section accounts for how I chose the datasets and constructed the models, as well as how I evaluated them.
4. **Results:** Here, I list the results of the evaluation of the models, and visualize some parts of the most advanced model's inner workings.
5. **Discussion:** I discuss the results and make a comparison to earlier work in this section. I also discuss the technical aspects of training the models. Finally, I answer the research questions.
6. **Conclusion:** This section concludes the report, and mentions future work.

1.7 Prior Knowledge Needed

This report assumes knowledge of basic university level mathematics such as linear algebra, multi-variable calculus and probability theory and statistics. It does not, however, assume knowledge in Machine Learning (ML) or Natural Language Processing (NLP), and I will attempt to define the concepts related to ML and NLP used in the experiments as clearly as possible.

Chapter 2

Theoretical Background

In the following, I will introduce the concepts used in the models. First, I will give some background to Natural Language Processing and the text classification task as well as word embeddings and BERT. Then, I will go through the Machine Learning theory needed for the models. Finally, I define Artificial Neural Networks, the LSTM and Attention concepts and the Hierarchical Attention Network, within the context of Natural Language Processing.

2.1 Natural Language Processing

Natural Language Processing (NLP) is the study of making computers able to process and understand human (natural) language, and it is a field inheriting from Artificial Intelligence and Linguistics (Khurana et al., 2022). The possible applications of NLP include parsing, machine translation, sentiment analysis, natural language generation and many more. Up until the 1980's, most successful NLP systems were based on handwritten symbolic logical rules, and designed by expert engineers, such as in Nilsson (1982). With increasing amounts of empirical data and computational power, the field shifted to using more methods from Machine Learning in the 1990's (Manning and Schütze, 1999). These new methods outperformed earlier methods in virtually all tasks. In the last 15 years, however, the field has been increasingly dominated by Deep Learning, just like the field of AI, seen for example in Bengio (2009). Deep neural networks gained attraction already in the 1990s, but were not successful because of the lack of enough amounts of data and proper design and learning methods. It was not until the 2010's that these methods have been successfully applied to real-world problems. For a more thorough presentation of the history of NLP, see Deng and Liu (2011, pp. 1-7) or Khurana et al. (2022).

2.1.1 Text Classification

Text classification is one of the tasks within NLP. It has a long history and has applications such as sentiment analysis (Nasukawa and Yi, 2003) and anti-spam filtering (Cohen, 1996). It is simple to formulate: given a text document, assign to it a class $y \in Y$, where Y is a set of possible classes. This class could be anything from sentiment (positive, negative or neutral) to a topic, depending on the application. There are many ways to tackle this task, and the methods I present in this report will all first compute a vector representation of the text and then feed this through a classifier.

2.1.2 Term Frequency - Inverse Document Frequency

A simple way to vectorize a document is to count the words in each document and use this as a basis for creating the vector. This can then be used for classifying documents. Term-document matrices were first described in the context of information retrieval by Salton (1971).

Term frequency - inverse document frequency (tf-idf) is one possible measure for the importance of a word in a document. The term frequency is defined for each term in each document as the amount of times it occurs in this document. The inverse document frequency, on the other hand, is the inverse of the amount of documents that the term occurs in overall (Sparck Jones, 1972). Only counting the term frequency would mean that the most common words are taken into account more when predicting, but a lot of these are words which are common across all documents, such as common non-lexical words. In order to tackle this issue, the inverse document frequency makes sure that a word in a document becomes more important for that document if it does not occur in many other documents. Given the term frequency and the inverse document frequency, the tf-idf score is calculated as follows:

$$\mathrm{tfidf}(t) = \mathrm{tf}(t) \cdot \mathrm{idf}(t),$$

where $\mathrm{tf}(t)$ is term frequency of the term t and $\mathrm{idf}(t)$ is the inverse document frequency of the term. Both $\mathrm{tf}(t)$ and $\mathrm{idf}(t)$ can be defined in multiple ways. In the implementation which was used in this report, the idf was smoothed, which meant that the inverse document frequency is defined as

$$\mathrm{idf}(t) = \log \frac{1+n}{1+\mathrm{df}(t)} + 1.$$

Here, n is the total amount of documents and $\mathrm{df}(t)$ is the amount of documents which contain the term t. The vector resulting from this are then normalized according to the Euclidean norm. Computing tf-idf results in one vector for each document. This vector may then be input to a classifier such as logistic regression, which will be described in section 2.2.1, in order to classify the document.

2.1.3 Word Embeddings

Word embeddings is a way of representing words through numerical vectors. It builds on the distributional hypothesis which says that words with similar meanings tend to occur in similar contexts (Harris, 1954), (Firth, 1957), (Joos, 1950). Following this, it is possible to compute word embeddings given the distribution of words in documents. Word embeddings may be static, which means that a given word always has the same embedding. Examples of widely used vector space models for static word embeddings are word2vec (Mikolov et al., 2013a), (Mikolov et al., 2013b), which trains a neural network to learn static representations of words, and GloVe (Pennington et al., 2014). Other word embeddings are contextual, where the words around it are taken into account, such as in BERT (Devlin et al., 2019). This has the advantage, compared to static embeddings, of capturing the property of natural language that a word does not have a single meaning. Instead it may have many different meanings depending on context. Different word meanings can take place both through polysemy (related but different senses of the same word) and homonymy (unrelated senses of the same word). For a more thorough description, see for example Saeed (2016, pp. 60-61).

2.1.4 BERT Models

BERT (Bidirectional Encoder Representations from Transformers) models are based on the Transformer architecture and are used to contextually embed words or sentences. Given a sentence (or sequence) input, they output numerical vectors which represent the meanings of the words within the specific context. Transformers use a mechanism called *self-attention* which enables the network to directly use information from arbitrarily large contexts and take this into context when computing representations of words. The Transformer model was first described in the 2017 paper (updated in 2023) Vaswani et al. (2023). BERT models are pretrained on two tasks: masked language modeling (MLM) and next sentence prediction (NSP). MLM is a task consisting of filling in the right word in a sentence where words have been left out (masked). When training on NSP on the other hand, for each training example consisting of two sentences the model predicts if one reasonably comes after the other in a document. The result of the training process is that the model learns the semantics of words in context. Pretrained BERT are general but they may then be finetuned for specific tasks (Devlin et al., 2019).

Tokenization within BERT Models

When processing a sequence of words, BERT model perform an internal *tokenization*. This is the process of dividing the sequence into *tokens*. BERT uses WordPiece embeddings, first described in Wu et al. (2016), to get tokens. Many tokens consist of a complete word but some words are also divided into multiple tokens. Subsequent tokens after the first token of a word then begin with `##`. There are also special tokens, such as the `[CLS]` token in the beginning of a sequence and the `[SEP]` token which is used as a separation token. `[PAD]` tokens may also be added if multiple sentences are encoded at once, and they need to be padded to the longest length. An example of tokenization using BERT can be seen below:

Original sentences: "På sjätte året Aniara drog med oförminskad fart mot Lyrans bild.", "Chefsastronomen höll ett föredrag för emigranterna om rymdens djup."

Tokenized:

```
['[CLS]', 'På', 'sjätte', 'året', 'An', '##iar', '##a', 'drog', 'med',
'oför', '##min', '##skad', 'fart', 'mot', 'Ly', '##rans', 'bild', '.',
'[SEP]'],
['[CLS]', 'Chef', '##sas', '##tr', '##onom', '##en', 'höll', 'ett',
'föredrag', 'för', 'emig', '##ranterna', 'om', 'rymden', '##s',
'djup', '.', '[SEP]', '[PAD]']]
```
Text from "Aniara: En revy om människan i tid och rum" by Harry Martinson (1956).

Sentence Transformers

Sentence-BERT is an optimization of BERT which computes embeddings for whole sentences at an extremely high speed compared to the original BERT models, while still being as accurate as BERT (Reimers and Gurevych, 2019). The Python framework for Sentence-BERT is called Sentence Transformers. It has the possibility of directly loading multilingual models, of which I used *paraphrase-multilingual-mpnet-base-v2*, trained on 50 languages, for this project.

KB-BERT

Kungliga biblioteket (KB) in Stockholm has developed a BERT model trained on Swedish language data, called KB-BERT. It is trained mainly on Swedish newspapers, but also on government reports, legal e-deposits and all Swedish Wikipedia articles. The model has a dictionary size of roughly 50 000 tokens (Malmsten et al., 2020).

2.2 Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence (AI), in which computers improve their performance for a given task, by observing examples and learning from them, in contrast to getting explicit rules (Kohavi and Provost, 1998). An example is a program trying to predict whether an image portrays a dog or a cat. By iterating through datasets consisting of images of dogs and cats, and comparing its own predictions to the true value (dog or cat), it learns implicitly what distinguishes these two, without getting explicit rules such as "a cat has pointy ears". ML consists of many different models, ranging from very simple ones such as logistic regression to more complex ones, such as deep neural networks.

2.2.1 Classification

Two of the main tasks within Machine Learning are classification and regression. In the regression task, the goal is to predict a continuous variable. In the classification task, on the other hand, the purpose of the model is to be able to place predefined labels on input data, or in other words, put them into the correct *classes*. It therefore differs from the regression task in that the variables are discrete. The classes do not necessarily need to be discrete numbers, even though this is the way they are represented internally in the model. In this report, the examples will be speeches and the classes are the parties belonging to each speech.

Logistic Regression

Logistic regression is one of the simplest classification algorithms. It was initially introduced in the context of Statistics by Berkson (1944), and it has many applications, including in Machine Learning. In the original statistical context, logistic regression uses a logistic curve to model a dependency between an explanatory variable x and an outcome y, by predicting $\hat{y}$ according to

$$\hat{y} = \text{logistic}(x) = \frac{e^x}{1 + e^x}.$$

In the Machine Learning context, inputs are more often in the form of vectors. If an example consists of an input vector $\mathbf{x} = [x_1 x_2 \dots x_N]$ and a true class y, having one of the possible values $y = 0$ or $y = 1$, logistic regression predicts the label according to

$$\hat{y} = \text{logistic}(\mathbf{w} \cdot \mathbf{x}) = \frac{e^{\mathbf{w} \cdot \mathbf{x}}}{1 + e^{\mathbf{w} \cdot \mathbf{x}}},$$

i.e. it finds a weight vector $\mathbf{w}$, takes the scalar product of this and $\mathbf{x}$ and uses this as input to a logistic function, which outputs a value between 0 and 1 (Walker and Duncan, 1967). This value represents the probability that the model "thinks" the correct label is 1. Therefore, if it is greater than 0.5, the final prediction is 1, and otherwise it is 0. Here, 1 and 0 could represent any discrete variable with two values, such as "is a cat" and "is not a cat".

Multinomial Logistic Regression

When the number of classes is greater than 2, the corresponding model is called multinomial logistic regression. The weights are then no longer stored in a single vector but instead in one vector for each class, $\mathbf{w}_c = [w_{c,1} w_{c,2} \dots w_{c,N}]$, forming a matrix W. In order for all probabilities to sum up to 1, a generalized version of the logistic function, called softmax, is used. It is defined as

$$\text{softmax}(\mathbf{x})_n = \frac{e^{x_n}}{\sum_{i=1}^{i=N} e^{x_i}}.$$

Multinomial logistic regression (Engel, 1988) uses this to find the probability of each class

$$p_c = P(y = c) = \text{softmax}(W \cdot \mathbf{x})_c = \frac{e^{\mathbf{w}_c \cdot \mathbf{x}}}{\sum_{c=1}^{C} e^{\mathbf{w}_c \cdot \mathbf{x}}}.$$

In order to get the final prediction, the class c with the highest probability p_c is chosen, i.e.

$$\hat{y} = \text{argmax}(\mathbf{p}) = \text{argmax}([p_1 p_2 \dots p_C]).$$

The logistic function is also called the sigmoid function, and in the following we will use the notation σ for both the logistic and the softmax function.

2.2.2 Loss Functions and Cross Entropy Loss

In order to measure how well a model has performed, a loss function is used. A good loss function returns 0 if the model predicts correctly at all times and a high number if the error is large. For multiclass classification, the *cross entropy loss* is suitable. It is also one of the simplest loss functions used in Deep Learning (Tian et al., 2022). It is used directly on the probability output of a model prior to applying softmax, $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2 ... \hat{y}_C]$, where C is the number of classes. For an example i with a prediction vector $\hat{\mathbf{y}}^{(i)}$ and true class $y^{(i)}$, the cross entropy loss between them is defined as

$$l_i = -\ln P(y = y^{(i)}) = -\ln \frac{e^{\hat{y}^{(i)}_{y_i}}}{\sum_{c=1}^{C} e^{\hat{y}^{(i)}_c}},$$

where $\hat{y}_c^{(i)}$ denotes the prediction value for the c:th class of the i:th example. The total cross entropy loss between a whole matrix of N prediction vectors $\hat{Y} = \begin{bmatrix}\hat{\mathbf{y}}^{(1)} & \hat{\mathbf{y}}^{(2)} & \dots & \hat{\mathbf{y}}^{(N)}\end{bmatrix}^T$ and the corresponding true labels $\mathbf{y} = \begin{bmatrix}y^{(1)} & y^{(2)} & \dots & y^{(N)}\end{bmatrix}^T$ is then the average of the cross entropy losses over all examples,

$$L(\hat{Y}, \mathbf{y}) = \frac{\sum_{i=1}^{N} l_i}{N} = -\frac{1}{N}\sum_{i=1}^{N} \ln \frac{e^{\hat{y}_{y_i}^{(i)}}}{\sum_{c=1}^{C} e^{\hat{y}_c^{(i)}}}.$$

2.2.3 Training a Model – Gradient Descent

Suppose we are able to calculate the performance of a model through a loss function, $L(\mathbf{w})$, where $\mathbf{w}$ is the vector consisting of the d weights (parameters) of the model. The objective is now to find weights which minimize the loss function L. A way to do this is through gradient descent. In gradient descent, we update the weights of the model according to the gradient of the loss function with respect to the weight vector, $\nabla_w L(\mathbf{w}) = \begin{bmatrix}\frac{\partial L}{\partial w_1} & \frac{\partial L}{\partial w_2} & \dots & \frac{\partial L}{\partial w_d}\end{bmatrix}$. In the simplest case, batch gradient descent, the gradient is computed for the entire dataset and the weights updated accordingly:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_w L(\mathbf{w}).$$

Here, η is the learning rate, which determines how large steps are taken when trying to find the minimum.

Stochastic gradient descent (SGD), on the other hand, updates the weights for each new example:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_w L(\mathbf{w}, x^{(i)}, y^{(i)}).$$

In practice, gradient descent is most often performed on minibatches of n items, so called minibatch gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_w L(\mathbf{w}, x^{(i:i+n)}, y^{(i:i+n)})\text{(Ruder, 2017).}$$

. This is the version that will be used in the experiments in this report. When performing gradient descent, we calculate the losses and change the weights repeatedly until the model has fitted well enough to the data.

Gradient descent can be viewed as traversing a space of the same dimensionality as the weight vector, where the height in each point in this space is corresponds to the value of the loss function at this point. What gradient descent then does is to find the steepest downward slope that eventually will lead to a minimum. For bigger batches, the algorithm is very slow. It is however more stable, compared to using smaller batches. This has the risk of being too quick and skipping over minima. On the other hand, it can easier leave local minima and may then be able to find a better minimum (Ruder, 2017).

There are many ways to optimize gradient descent. It is for example a good idea to be able to adjust the learning rate η in order to minimize the risk of skipping minima. One of the methods with overall best performance, and the one which will be used in the experiments in this report, is called Adam (Kingma and Ba, 2017).

2.2.4 Overfitting and Underfitting

The goal of a good Machine Learning model is to be able identify patterns in the data seen, which also hold for unseen data, in order to be able to predict correctly given this new data. In some cases, the model fits perfectly or close to perfectly to the training data, and finds patterns which cannot be generalized, and thus performs worse on the testing data. This is known as overfitting (Ying, 2019).

2.2.5 Evaluation – Precision, Recall and F_1

The precision and recall are two separate metrics which are computed for each class in the dataset a Machine Learning model is evaluated on. The precision for a class c is defined as the number of times c was predicted and true divided by the number of times c was predicted, and it therefore answers the question "How often is the model correct when it guesses c?". The recall for a class c, on the other hand, is the amount of times c was predicted and true divided by the amount of times c occurred, i.e. it answers the question "In how many of the cases when c was the correct class did the model predict correctly?" (Sammut and Webb, 2010, p. 781). The F_1 score for each class is then defined as the *harmonic* mean of the precision and recall (Sammut and Webb, 2010, p. 398), i.e.

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}},$$

where P is the precision and R is the recall. The macro F_1 is computed by taking an unweighted average of the F_1 scores of all classes (Santos et al., 2011).

2.3 Artificial Neural Networks and Applications in NLP

Artificial Neural Networks (ANNs) is a branch of Machine Learning which is inspired by biological neural networks. An ANN consists of layers which consist of nodes called units, with units in different layers being connected to each other. The value of a unit is determined by computing a weighted sum of the values of the units it is connected to, where one weight is associated with each edge to the unit. This sum is then the input to an activation function, the output of which is the resulting value of the unit (Rumelhart et al., 1986).

Let x_j be the value of unit j and $w_{i,j}$ the weight of the directed connection from unit i to unit j. Then

$$x_j = g_j(\sum_i w_{i,j} x_i),$$

where g_j denotes the activation function associated with unit j.

Some common activation functions are

1. the aforementioned sigmoid function $\sigma(x) = \frac{e^x}{1+e^x}$ returning values between 0 and 1,
2. the hyperbolic tangent, $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, returning values between -1 and 1, and

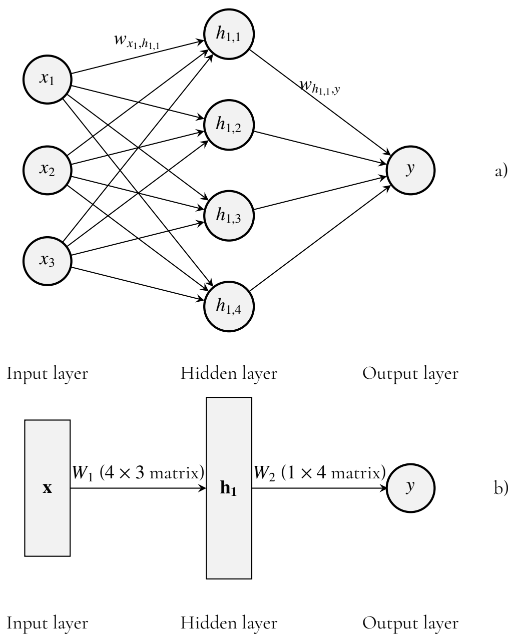


Figure 2.1: Two representations of an artificial neural network containing an input layer of size 3, a hidden layer of size 4 and an output layer of size 1. a) is the units representation and b) the matrix-vector representation. Activation functions have been omitted for simplicity.

3. the rectified linear unit, $\mathrm{ReLU}(x) = \max(0, x)$ (Fukushima, 1969).

The simplest ANN consists of just one layer, in which there is a direct mapping from input to output. A notable early example of such a network is the Perceptron, introduced in Rosenblatt (1958). Multilayer neural networks contain layers in between the input layer and the output layer, known as hidden layers. The most straight forward kind of multilayer neural networks is called feedforward neural networks (FNNs). In these networks, layers feed into each other in the forward direction, from input to output. The first such network containing multiple hidden networks, a deep network, was described in Ivakhnenko and Grigorevich (1967). When all the nodes in one layer is connected to all the nodes in the next layer, this will be referred to as a fully connected layer. Fully connected layers may be used together with an activation function such as softmax as the last layer in the network.

The units in a single layer of a network may also be represented as a vector. For example, with N units in a layer, we can represent this layer with $\mathbf{x} = [x_1 x_2 \ldots x_N]$. Computing the values of the units in a layer given the units in the previous layer then becomes a multiplication with a weight *matrix*. For example, if layer i contains nodes $x_{i,1} \ldots x_{i,N}$ and layer $i+1$ contains nodes $x_{i+1,1} \ldots x_{i+1,M}$, we can compute the values in layer $i+1$ through

$$\mathbf{x_{i+1}} = g(W\mathbf{x_i}),$$

where W is an $(M \times N)$ matrix and g is an activation function, applied element-wise. We may also add a bias b (which can be modeled with a unit of constant size 1),

$$\mathbf{x_{i+1}} = g(W\mathbf{x_i} + b)\text{(Rumelhart et al., 1986).}$$

An illustration depicting both the units representation as well as the matrix-vector representation of a simple ANN can be seen in Figure 2.1.

2.3.1 Training ANNs – Back-propagation

When modifying the weights of a neural network, it is not enough to just modify the weights according to the gradient descent algorithm, since there are weights in multiple layers and it is not sufficient to calculate one simple gradient where the loss is a function of all the weights. Instead, we use an algorithm known as the back-propagation algorithm, originally popularized by Rumelhart et al. (1986), to compute derivatives of the loss in an ANN. This operates on the mathematical principle of the chain rule, and uses this rule backwards in order to modify weights from the output to the input. The back-propagation algorithm consists of three parts which follow sequentially; the forward pass, the backward pass, and updating the weights:

1. **The forward pass:** First, we determine the values of the input units by taking input data from the dataset. We then calculate the values of all the following units by repeatedly stepping forward in the network. Finally the output is reached and we compute a loss value, depending on the difference between the prediction output and the true value.

2. **The backward pass:** Then, we compute the gradient of the loss with respect to the weights on the edges. First, we calculate the gradient with respect to the weights directly before the output. Then, we use the chain rule to repeatedly calculate derivatives with respect to the weights in the backwards direction until the input is reached.

3. **Updating the weights:** Finally, we update the weights of the network by following the negative direction of the gradient of the loss.

2.3.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a kind of ANN which are different compared to regular FNNs in that the information is not only fed in one direction. While an FNN passes information from the input to the output through one or more hidden layers, an RNN also passes information back onto itself. This enables it to not only take the current input but also its own output of earlier inputs into account when computing the output (Jordan, 1986), (Schmidt, 2019). Within NLP, RNNs may be used to process sequences of words. The network will then use the *hidden state* to store the representation of what it has seen so far, and use the current word as well as the last hidden state, depending on previous words, to compute a new hidden state. Figure 2.2 shows a simple RNN. In the case of sentences with words, each $\mathbf{x}^{(t)}$ would represent a word. For text classification, one can feed a sequence of length T through an RNN and then use the last hidden state $\mathbf{h}^{(T)}$ as a hidden representation of the whole text. This can then be input through a fully connected layer to give a class prediction. Training an RNN is known as *back-propagation through time*, and works similarly to back-propagation, except the hidden layers are also processed multiple times, backwards in the order of the sequence (Werbos, 1990).

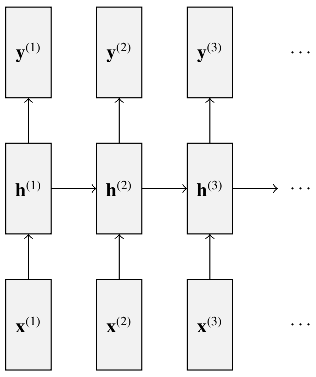


Figure 2.2: An example of an RNN structure, where each hidden state $\mathbf{h}^{(t)}$ feeds into the next hidden state $\mathbf{h}^{(t+1)}$.

2.3.3 Long Short-Term Memories

Multiple difficulties arise when training ordinary RNNs. One feature which they lack is the ability to easily track distant information (Bengio et al., 1994). As an example, in the following sentence,

"I went to the store to buy some milk, but it was already closed.",

the last phrase "it was already closed" refers back to "store" and has nothing to do with "milk". In an ordinary RNN, it is difficult to model this long term dependency, when each hidden word representation is merely a function of the hidden word representation before it. A better solution would be to remember "store", then process "buy some milk", and once the last phrase is reached, connect this to "store" again.

Another problem in training RNNs is what is known as the *vanishing gradients* problem (Kolen and Kremer, 2001). It arises from the fact that the hidden layer at time t contributes to the loss in the following time steps. Because of this, many multiplications between the hidden layers are made. Often this means that the gradients become very close to zero.

Long short-term memories (LSTMs) are a special kind of RNN and were introduced by Hochreiter and Schmidhuber (1997) to solve the issue of vanishing gradients, and they also solve the first mentioned problem. They perform well in handling dependencies between items far apart in a sequence, such as words in a sentence or sentences in a document, but may also be used for timeseries analysis in which events connected to each other occur with a big time gap in between them (Van Houdt et al., 2020). This is because they contain several gates which allow them to learn what to memorise and forget from a sequence.

An LSTM consists of an input gate, a forget gate, and an output gate, as well as a cell state $\mathbf{c}^{(t)}$ and a hidden state $\mathbf{h}^{(t)}$. The cell remembers information over arbitrary time intervals and can be seen as the "long-term memory", keeping track of important information from the sequence so far. The hidden state, on the other hand, acts as the "short-term memory", providing a hidden representation of the last item in the sequence, with earlier items taken into account. Both $\mathbf{h}^{(t)}$ and $\mathbf{c}^{(t)}$ are vectors of a length called the *hidden size* which may be different from the length of each item $\mathbf{x^{(t)}}$, $t \in [1, T]$ in the input sequence.

For each new item $\mathbf{x}^{(t)}$, the LSTM computes the following:

1. In the **forget gate**, the previous hidden state $\mathbf{h}^{(t-1)}$ and the current input $\mathbf{x}^{(t)}$ are con-

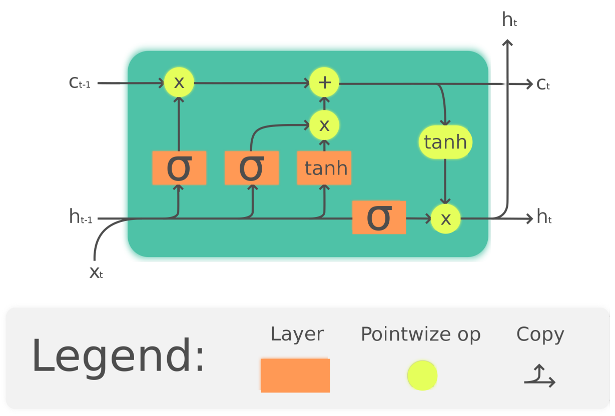


Figure 2.3: A simple illustration of an LSTM layer and its components. Source: `https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg`

catenated, multiplied with a weight matrix and then put through a sigmoid function, resulting in a vector of values between 0 and 1, often denoted $\mathbf{f}^{(t)}$. This represents what to remember and what to forget from the previous cell state $\mathbf{c}^{(t-1)}$.

2. The part of the LSTM devoted to remembering new information consists of the block input and the input gate.

   The **block input** is calculated similarly to $\mathbf{f}^{(t)}$, but goes through another activation function, often a tanh function, giving values between -1 and 1. This vector can be seen as candidate values to add to the cell.

   In the **input gate**, $\mathbf{x}^{(t)}$ and $\mathbf{h}^{(t)}$ are multiplied with yet another weight matrix, and put through a sigmoid function, just like in the forget gate. This vector, called $\mathbf{i}^{(t)}$, is pointwisely multiplied with the block input. It can therefore be seen as the vector that is deciding what to keep from the candidate values. The resulting vector is added onto the result from the forget gate, forming the new cell state $\mathbf{c}^{(t)}$.

3. The **output gate** works completely like the forget and input gates, and uses the sigmoid activation function as well as a separate set of weights. The resulting vector $\mathbf{o}^{(t)}$ determines what parts of the cell state should be included in the final output. The final step consists of letting the new cell state $\mathbf{c}^{(t)}$ go through a tanh activation function to normalize it, and multiply the result pointwisely with $\mathbf{o}^{(t)}$. The result is the new hidden state $\mathbf{h}^{(t)}$.

The procedure I have accounted for above is the one the Pytorch implementation uses. A very similar version which also includes so called peep-hole connections is described in more detail in Sak et al. (2014). All in all, the LSTM computes a new hidden state and a new cell state for every new item in the sequence. From here on, the whole sequence $h = [\mathbf{h}^{(1)}\mathbf{h}^{(2)} \ldots \mathbf{h}^{(T)}]$ will be referred to as the sequence output of the LSTM.

A conceptual illustration of the LSTM layer can be seen in Figure 2.3.

In order to prevent overfitting too early in the training process, dropout layers may also be introduced. The simplest dropout layer removes a portion of the units in a layer, which is chosen randomly but is of a constant size (Srivastava et al., 2014). An LSTM network may also consist of several layers. In this case, the output of the i:th layer is the input to the $i+1$:th layer. Dropout layers in between the LSTM layers may then be added to decrease overfitting.

2.3.4 Attention

Attention is a mechanism which lets a model focus more on certain parts of a sequence than other. This is done through attention weights. An attention layer takes as input a sequence and outputs a weighted average of each element in the sequence (Bahdanau et al., 2016). For example, an attention layer on the word level may take as input representations of words, and output a representation of the whole sentence consisting of a weighted average of the representations of the words. There are multiple different attention mechanisms, the simplest one being dot-product attention.

Dot-Product Attention

In the following, I will account for an attention mechanism used for text classification, described in Yang et al. (2016).

Given a representation of a sequence $h = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \dots & \mathbf{h}_T\end{bmatrix}$, where each item $\mathbf{h}_t$ is a vector representation of a word or a sentence, dot-product attention does the following: First, a hidden representation of the sequence is computed. Each element in this new sequence $\mathbf{u}_t$ is then compared to a context vector $\mathbf{c}$ by taking the dot-product between the two vectors. This is then the input to a softmax function, which produces a normalized distribution of weights among the items in the sequence, here called $\mathbf{a}$. A new representation of the sentence, $\mathbf{s}$ is then computed by multiplying each item in the original sequence with its corresponding weight. Mathematically it looks like the following:

$$\mathbf{u}_t = \tanh(W\mathbf{h}_t + \mathbf{b}), t \in [1, T], \tag{2.1}$$

$$a_t = \frac{e^{\mathbf{u}_t \cdot \mathbf{c}}}{\sum_{t=1}^{t=T} e^{\mathbf{u}_t \cdot \mathbf{c}}}, \tag{2.2}$$

$$\mathbf{s} = \sum_{t=1}^{T} a_t \cdot \mathbf{h}_t. \tag{2.3}$$

The context vector $\mathbf{c}$ is initially randomized and then learned during the training process (Yang et al., 2016).

Through this method, a new representation of a sequence with the dimensions the items in the sequence used to have, is created.

2.3.5 Hierarchical Attention Network

The hierarchical attention network (HAN) was introduced in 2016 by Yang et al. (2016) and uses attention on multiple levels. It can be used for different sorts of data and tasks but in the following I will explain how it is used for document classifications. First, word sequences are encoded using an RNN, such as an LSTM. The sequence of hidden representations resulting from this are then fed through a word-level attention layer, which outputs a sentence embedding, being a weighted average of the word embeddings. All the sentence embeddings are then fed through an RNN as well, whose sequence output is the input to another attention layer, but on the sentence level. This attention layer computes one document embedding as a weighted average of the sentence embeddings.

Mathematically, we can represent each document as $D = [S_1 S_2 \dots S_L]^T$, containing sentences $S_i = [\mathbf{w}_{i,1}\mathbf{w}_{i,2} \dots \mathbf{w}_{i,W_i}]$, where each $\mathbf{w}_{i,j}$ is a word embedding of the j:th word in the i:th sentence, $i \in [1, L]$, $j \in [1, W_i]$. In other words, each sentence is here represented as a matrix of word embeddings. The model first applies an RNN to each sentence. The RNN is bidirectional which means that it is first applied from the first to the last item and then from the last to the first item in the sequence. Denoting these two networks $\overrightarrow{\mathrm{RNN}}$ and $\overleftarrow{\mathrm{RNN}}$, we compute

$$\begin{aligned}\overrightarrow{h}_{i,j} &= \overrightarrow{\mathrm{RNN}}(\mathbf{w}_{i,j}), j \in [1, W_i]\\ \overleftarrow{h}_{i,j} &= \overleftarrow{\mathrm{RNN}}(\mathbf{w}_{i,j}), j \in [W_i, 1],\end{aligned}$$

and then concatenate these two into

$$\mathbf{h}_{i,j} = [\overrightarrow{h}_{i,j}\overleftarrow{h}_{i,j}],$$

which is the hidden representation of the word sequences. These sequences are then the input to a word attention network, in which a sentence vector $\mathbf{s}_i$ for each sentence is formed according to equations 2.1-2.3.

These sentence embeddings are also encoded using an RNN:

$$\begin{aligned}\overrightarrow{h}_i &= \overrightarrow{\mathrm{RNN}}(\mathbf{s}_i), i \in [1, S],\\ \overleftarrow{h}_i &= \overleftarrow{\mathrm{RNN}}(\mathbf{s}_i), i \in [S, 1],\\ \mathbf{h}_i &= [\overrightarrow{h}_i\overleftarrow{h}_i].\end{aligned}$$

These hidden representations are then the input to a sentence attention layer, which outputs an embedding for the entire document, $\mathbf{d}$. Finally the probability of the classes are achieved through applying a fully connected layer,

$$\hat{\mathbf{y}} = W_d\mathbf{d} + \mathbf{b}_d.$$

In order to get one single prediction, softmax followed by argmax are applied to this vector.

2.4 Frameworks Used

I used the following frameworks in the experiments. *SpaCy* is a framework which contains models for NLP, such as tokenization models for many languages, and is implemented in Python. *Beautiful Soup* is a Python package dedicated to parsing HTML documents. It may be used to extract information within certain tags. *Scikit-learn* is a Machine Learning library, which contains many different models as well as methods to evaluate them. It does not, however, focus on deep learning. *Pytorch*, on the other hand, is a library mainly devoted to deep learning models and includes implementations of many of the concepts brought up in this chapter, such as RNNs, as well as practical tools, like loss functions and convenient ways to create datasets and load data for training and testing. It also offers GPU support, which was needed in order to implement the most computationally heavy models in this report. Links to these projects as well as other packages used can be accessed in appendix A.

Chapter 3
Methodology

In this section I will account for how the datasets were chosen and divided into training, validation and test sets, how the data was pre-processed, and most importantly, how the models were constructed, trained and evaluated.

3.1 Pipeline

The whole pipeline, from collecting the data to getting a prediction, can be divided into the steps seen in Figure 3.1. The main component that differs is between different experiments is the classifier. However, the final model in this report, KB-BERT-HAN, features another BERT model than the other models, which is on the word level as opposed to the sentence level.

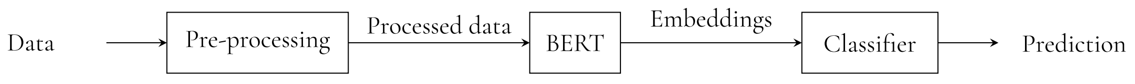


Figure 3.1: The complete pipeline for predicting a party, given a text.

3.2 Datasets

3.2.1 Raw Data

Data consisting of speeches in the Swedish parliament is available on Sveriges Riksdag (2023a). The data reaches from parliamentary years 1993/1994 to 2022/2023. Each speech is contained in one file, with several attributes, the most important ones used in the experiments being the speech text, the party and the speech ID. The parties currently in the Swedish parliament are Socialdemokraterna (S, Social Democrats), Moderaterna (M, Moderate Party), Sverigedemokraterna (SD, Sweden Democrats), Vänsterpartiet (V, Left Party), Miljöpartiet (MP, Green Party), Centerpartiet (C, Centre Party), Kristdemokraterna (KD, formerly KDS, Christian Democrats) and Liberalerna (L, formerly FP, Liberals).

Distribution of Parties in the Dataset

Out of the aforementioned parties, S, M, V, MP, C and KD have had seats in the parliament throughout the whole dataset, while MP entered the parliament in 1994 after leaving it in 1991 and SD did not enter it until in 2010 (Sveriges Riksdag, 2023c). This means that for a model training on all available speeches, speeches from the third biggest party in the parliament today, SD, will be nonexistent in more than half of the dataset. From 1993 and up until now, most governments have been run by S, the biggest exception being the government lead by Reinfeldt (M) between 2006 and 2014 (Sveriges Riksdag, 2023c). Table 3.1 shows how the parties are currently distributed in terms of seats in the parliament and in terms of amount of speeches in the dataset most widely used in the experiments.

Party	Seats 2022		Speeches 2014-2022	
	No.	Rel. freq.	No.	Rel. Freq.
S	107	30.7%	32842	33.4%
SD	68	19.5%	9609	9.8%
M	73	20.9%	18260	18.6%
V	24	6.9%	8058	8.2%
MP	18	5.2%	10543	10.7%
C	24	6.9%	6792	6.9%
KD	19	5.4%	6440	6.6%
L (FP)	16	4.6%	5752	5.9%
Total	349	100%	98296	100%

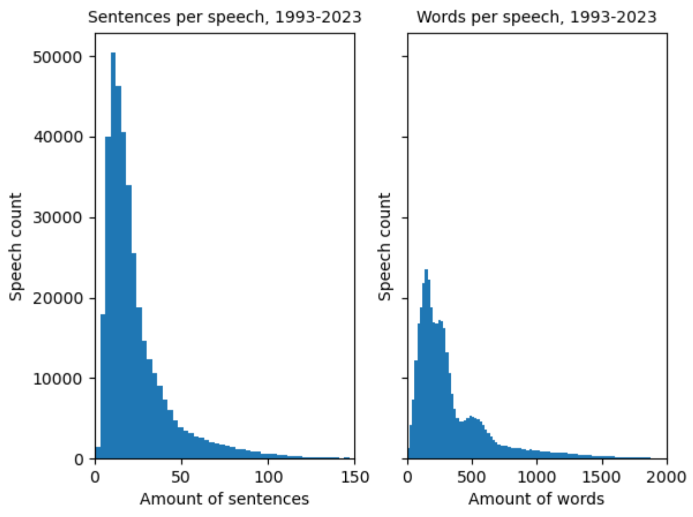
Table 3.1: Distribution of speeches between 2014/15 and 2021/22 as well as parliament seats after the 2022 election for the parties in the Swedish Parliament. Source: Sveriges Riksdag (2023b)

Sentence and Word Counts of Speeches

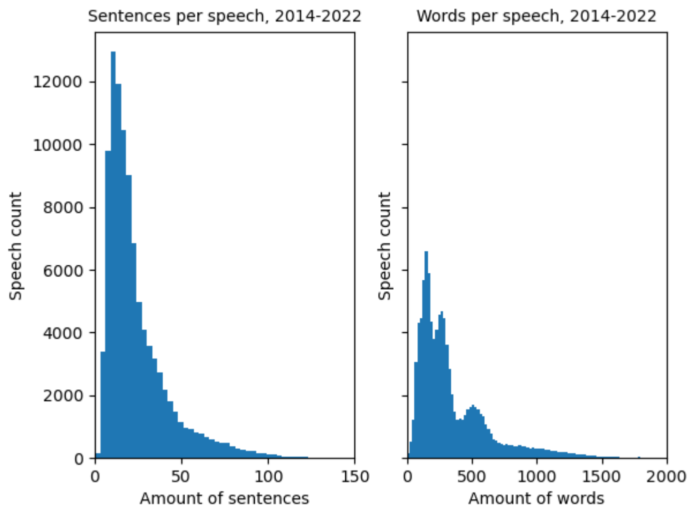
To get an overview of the speeches, I also computed the distribution of the amounts of sentences and words in the speeches. I did this for both all speeches from 1993 to 2023, as well as the speeches from 2014 to 2022, which constituted the most used training set. The resulting means, medians and standard deviations can be seen in Table 3.2. The complete distributions can be seen in Figure 3.2.

Dataset	Amount of sentences			Amount of words		
	Mean	Median	Std. dev.	Mean	Median	Std. dev.
Speeches 1993/94-2022/23	23.3	17	20.8	351	249	325
Speeches 2014/15-2021/22	23.6	18	19.1	355	262	297

Table 3.2: The mean, median and standard deviation for the amount of sentences and words for speeches within the complete dataset as well as the most used training set.



(a)



(b)

Figure 3.2: The distributions of sentences and words per speech in a) the complete dataset, b) the dataset most widely used in the experiments.

3.2.2 Training, Validation and Test Sets

I divided the data set into training sets, as well as a validation set and a test set. The main training, validation and test sets together constitute speeches from the last two complete terms (2014-2018 and 2018-2022). Throughout these terms, the governments Lövfen I, II and III consisting of S and MP, as well as the Andersson government consisting of S have been in place. The main training set consists of the first seven years; that is all speeches in the parliament between 2014/15 and 2020/21. All models were trained on this dataset. The validation and test sets together make up the last year of speeches in this term, that is, speeches from the parliamentary year 2021/22.

I used dataloaders which shuffled the datasets and gave examples to be classified, consisting of a randomly selected batch of speeches and the corresponding parties.

Once all models had been run on the eight year period, I investigated the most advanced model, KB-BERT-HAN, further. I trained it on all available speech data from the years 1993/94 to 2020/21 and tested it on the year parliamentary year 2021/22 once again. I also trained the baseline model on these 29 years of speeches.

3.2.3 Sentence VS Word Embeddings in Datasets

All neural models except KB-BERT-HAN use datasets which directly outputs sentence embeddings computed using Sentence Transformers. For KB-BERT-HAN, a specific dataset had to be created. This outputs the sentences themselves, tokenized by the KB-BERT tokenizer. The tokenized sentences were then processed by the KB-BERT model outside the dataset.

3.3 Pre-processing

Before being loaded into datasets for training and testing, I processed the speech data from Riksdagen, formatted as json files, using the `json` module in Python. From each relevant json file, I extracted the speech text and the party as well as the ID. In some speeches, the person uttering the speech is either the speaker of the parliament or a member of a party not currently in the parliament. I did not include these speeches in any datasets. I parsed the paragraphs of each speech using Beautiful Soup and divided the text into sentences using the spaCy tokenizer for Swedish using the model *sv-core-news-lg*. I also manually filtered out some unneccesary strings, such as "(Applåder)" and formatting text.

3.3.1 Filtering out Party Names

For KB-BERT-HAN I constructed yet another, more filtered, data set. In this dataset, I replaced words for parties such as "Vänsterpartiet" with a placeholder party "Partiet", and words for members of a party such as "vänsterpartister" with "partister". I state the rationale for this in section 5.2.3.

3.3.2 Oversampling

Since the different parties in the parliament are of different sizes, in terms of the amount of votes and members in the parliament, it is expected that the amount of speeches available will differ among parties as well. This is also the case, with S being represented roughly twice as often as any other party in the datasets available. This poses a problem because of the way Machine Learning works. Since a model always seeks to minimize loss, is has a tendency to predict a larger class more often because it is a quick way to decrease the overall loss and increase the overall accuracy. As a result, the classifier achieves a lower recall on other classes. One way to tackle this issue is to oversample, that is, to introduce more examples from smaller classes so that all classes get the same portion of the training set (R. C. Prati and Monard., 2009). This can be done automatically by using the package Imbalanced learn (Lemaître et al., 2017). In this report, I tested random oversampling, which simply duplicates examples with underrepresented classes, for all models.

3.4 Classification Models

All models take speeches as input and compute document embeddings which are used as input for classification by party. I constructed one model utilizing tf-idf scores and logistic regression as a baseline to compare the more advanced models to. The more advanced models are ANNs which use BERT word or sentence embeddings to compute the document embeddings. The models utilizing sentence embeddings use Sentence Transformers with the model *paraphrase-multilingual-mpnet-base-v2*. The most advanced model, KB-BERT-HAN, computes individual word embeddings and uses the *bert-base-swedish-cased* model and tokenizer from KB for this.

3.4.1 Tf-idf Baseline Model

The baseline model is a tf-idf model. First, we remove stop-words such as "a", "an" and "the" as they do not contribute to the meaning of the document using this method. The model then performs a tf-idf weighting of the words in all documents according to equation 2.1.2. The parameter `min_df`, which controls the amount of documents a word needs to be in for it to be counted, was set to 10. For a dataset containing many documents, the tf-idf vectors become very large. In order to lower the dimensions of the vectors, I use singular value decomposition (SVD). Using tf-idf together with truncated SVD is known as latent semantic analysis (LSA)(Dumais, 2004). In this case the vectors were reduced from the amount of unique words down to 512.

I then fed the matrix of dimensions $n \times 512$, where n is the amount of documents, through a multinomial logistic regression model, in order to get a party prediction from each document vector in the matrix. I used the cross entropy loss as a loss function for the logistic regression algorithm.

3.4.2 Average of Sentence Embeddings

The first ANN I developed for this task computes the embeddings of individual sentences in a speech using Sentence Transformers and the *paraphrase-multilingual-mpnet-base-v2* model. The model then calculates the global average of these embeddings in order to receive an embedding of the whole speech. It then feeds this embedding through a fully connected layer which outputs the predicted probability of a label, given the speech embedding it has received.

3.4.3 LSTM on Sequences of Sentences

A more sophisticated model used a bidirectional LSTM layer on a whole speech of sentences. The LSTM layer takes a sequence of sentence embeddings as input. The model uses the last hidden state of the LSTM as input for a fully connected layer with an output size corresponding to the amount of parties. I constructed this model in two variants: one which had a one-layer LSTM and one which had a two-layer LSTM with a dropout probability of 30% between the layers.

3.4.4 LSTM with Attention

I then added an attention mechanism onto the LSTM model. This model still used Sentence Transformers to compute sentence embeddings. These were encoded with an LSTM, and in contrast to the model before this model uses the hidden state of each sentence, not only the last hidden state. These hidden states were the input to an attention layer which computed a weighted average. The result of the attention layer is an embedding for the whole document which is used for classification.

3.4.5 KB-BERT-HAN

The most complex model I tested for this task was a HAN. I constructed the model by modifying a HAN model already implemented in Python using Pytorch, by Kim (2019). I modified the `HierarchicalAttentionNetwork` class to take an already tokenized batch of sentences as input. I combined the model with the tokenizer and language model by KB in order to form KB-BERT-HAN. Running this model in practice follows this procedure:

1. Optionally, we create embeddings for each document beforehand, each consisting of a tensor of word embeddings, using *bert-base-swedish-cased* from KB. We save these in a dictionary for accessing later in the training process.
2. Then, each forward pass through a batch of documents looks like the following:
    (a) All embeddings corresponding to the documents in the batch are accessed from the BERT dictionary or computed directly if the dictionary was not created.
    (b) The model creates a vector representation of each sentence in each document from the word embeddings belonging to that sentence using a word attention network.

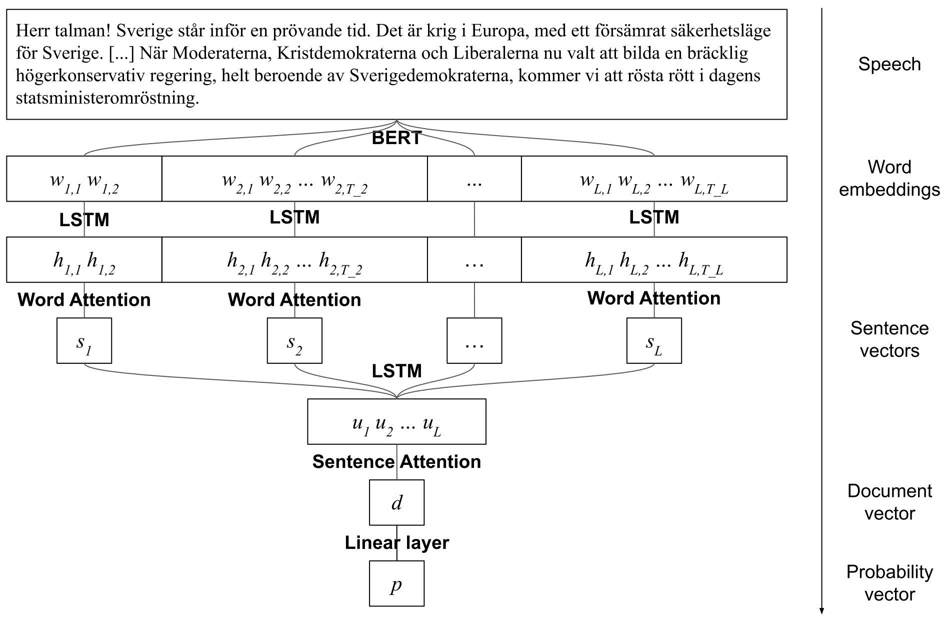

Herr talman! Sverige står inför en prövande tid. Det är krig i Europa, med ett försämrat säkerhetsläge för Sverige. [...] När Moderaterna, Kristdemokraterna och Liberalerna nu valt att bilda en bräcklig högerkonservativ regering, helt beroende av Sverigedemokraterna, kommer vi att rösta rött i dagens statsministeromröstning.
BERT
$w_{1,1}\ w_{1,2}$
$w_{2,1}\ w_{2,2} \dots w_{2,T\_2}$
...
$w_{L,1}\ w_{L,2} \dots w_{L,T\_L}$
LSTM
$h_{1,1}\ h_{1,2}$
$h_{2,1}\ h_{2,2} \dots h_{2,T\_2}$
$h_{L,1}\ h_{L,2} \dots h_{L,T\_L}$
Word Attention
s_1
s_2
s_L
LSTM
$u_1\ u_2 \dots u_L$
Sentence Attention
d
Linear layer
p
Speech
Word embeddings
Sentence vectors
Document vector
Probability vector

Figure 3.3: A simplified illustration of the complete KB-BERT-HAN model. Each word embedding $w_{i,j}$ is here a vector of the length 768.

(c) These sentence vectors are input to a sentence attention network, which outputs document vectors.

(d) Each document is the input to a fully connected layer, giving unnormalized class probabilities.

The word and sentence attention models work very similarly and use LSTMs with the same parameters: hidden size, attention size (the size the sequence gets in the attention step after it has been encoded, and therefore also the size of the context vector), number of layers and dropout probabilities.

An illustration of the complete model can be seen in Figure 3.3.

I created two versions of the model; one in which all of the word embeddings were computed beforehand and stored in the RAM and one in which they were computed for every new prediction. The first version has the perk of being faster once training the HAN, but also has the downside of being very storage intensive. The second version on the other hand, is able to process much more data, but does this slower. Due to memory and storage limitations, I opted for the second version.

3.5 Practical Difficulties for Data Handling

I implemented all models except the baseline model in Pytorch, which uses tensors as its data type. These are multidimensional matrices in which all elements are of the same datatype. Because of this, tensors cannot contain vectors of variable length, which becomes a problem in this case since every speech contains a variable amount of paragraphs which in turn contain variable amounts of sentences. In order to handle this, a sequence may be padded or packed. A padded sequence fill in sequences with padding elements, for example 0 or $-\infty$, when needed. to make all sequences the same length. A packed sequence on the other hand stores all sequences in a list and keeps track of the length of each, which makes it possible to store all sequences together in less space while being able to reconstruct them. Both padded and packed sequences are implemented in Pytorch. The LSTM layer in Pytorch is also able to directly take a batch in the form of a packed sequence of tensors as input and automatically handle each one separately. In the implementations, all models implemented in Pytorch process multiple speeches simultaneously as a packed sequence.

3.6 Training

I trained the models on batches of the training set. I optimized the gradient descent with Adam. For evaluating a model's performance during training, I used the cross-entropy loss.

I trained and evaluated the models according to the following scheme. Each epoch consists of the following steps:

1. Training - do the following for every batch in the training set:
   (a) Use the model to predict the classes of a whole batch, i.e. make a forward pass.
   (b) Calculate the training loss by comparing to the true classes.
   (c) Reset the gradient of the optimizer.
   (d) Make a backward pass to calculate gradients.
   (e) Use the optimizer and the calculated gradients to change the weights of the model.
2. Evaluating:
   (a) Predict all classes in the validation set.
   (b) Calculate the validation loss by comparing to the true classes.

3.6.1 Hyperparameters

I investigated hyperparameters such as batch size, the maximum learning rate and the amount of epochs. I found a batch size of 32 to be suitable for most models. I tested the maximum learning rate in logarithmic steps, and learning rates from $1 \cdot 10^{-3}$ to $1 \cdot 10^{-4}$ were tested. For KB-BERT-HAN, I tweaked the learning rate and amount of epochs further, since it was the most advanced mode, and therefore the one i focused the most on. As previously stated, I constructed the validation and test sets by extracting a full year of speeches, different from the speeches in the training set. From this set, I constructed the validation and test sets by randomly selecting half of these speeches into each set. This makes the evaluation unbiased in terms of seasonal differences in speeches. For the same reason, I did not choose the parliamentary year 2022/23 to neither train nor evaluate, as it is not yet a complete parliamentary year.

3.7 Implementation

3.7.1 Software

I implemented all the models in Python 3.9.16. For the baseline model, I used Scikit-learn (Pedregosa et al., 2011), since this offers easy to use tf-idf and SVD implementations. For the neural models, on the other hand, I used Pytorch 2.0.1 (Paszke et al., 2019).

3.7.2 Hardware

The largest models are very memory and CPU intense. I therefore used the computation cluster COSMOS at LUNARC (LUNARC, 2023) for computing. Each node in the cluster is equipped with two 24-core AMD7413 CPUs, 512GB RAM as well as an A100 GPU with 80GB VRAM, which was used to train the models. I used a single node of the cluster for each training session.

3.8 Evaluation

After training the models, I evaluated them on the test set. As evaluation metrics, I used the accuracy and macro F_1 score.

3.9 Visualizing KB-BERT-HAN

Once I had trained and evaluated KB-BERT-HAN, I visualized the version of it with the highest evaluation scores.

First, the final embeddings before the last layer determining the party can be reduced down to two dimensions and visualized. I did this using the UMAP algorithm (McInnes et al., 2020), which is a non-linear dimension reduction algorithm. This gives a clear view on how far apart different political narratives are according to the model. If clear clusters are formed, this means that there is a clear distinction between parties, but if the clusters are harder to tell apart, the different narratives are closer to each other according to the model. I used the parameter values `n_neighbors=1200` and `min_distance=0.1`, and the cosine metric for measuring distance between the vectors.

I also visualized the attention weights of the model, for both sentences and words. When visualizing, the colorfulness of a token is determined by multiplying the token weight and the sentence weight of the sentence in that token. This is normalized for each speech by dividing by the largest of these products within that speech.

Chapter 4

Results

The models performed according to the results below. Since KB-BERT-HAN had the best performance, I also investigated this model more thoroughly.

4.1 Results per Model

4.1.1 Summary

The accuracy and F_1 scores for the models evaluated can be seen in Table 4.1. The accuracies and F_1 scores shown are from the runs which resulted in the highest F_1 scores.

Model	Training set	Accuracy	Macro F_1
Tf-idf	2014/15-2020/21	55.2%	46.8%
Tf-idf	1993/94-2020/21	48.8%	38.6%
Embedding average	2014/15-2020/21	39.1%	35.1%
One-layer LSTM	2014/15-2020/21	61.6%	55.8%
Two-layer LSTM	2014/15-2020/21	63.4%	58.1%
LSTM with attention	2014/15-2020/21	61.9%	56.0%
KB-BERT-HAN	2014/15-2020/21	**72.9%**	**68.5%**
KB-BERT-HAN	1993/94-2020/21	71.8%	67.2%

Table 4.1: A summary of the highest macro F_1 and accuracy scores for the models evaluated.

4.1.2 Tf-idf

When trained on speeches from the parliamentary years 2014/15 to 2020/21 and evaluated on speeches from 2021/22, the tf-idf model achieves an accuracy of 55.2% and a macro F_1 score of 46.8%. When trained on speeches all the way from 1993 to 2021, the tf-idf model achieves an accuracy of 48.8% and a macro F_1 score of 38.6%. In these experiments, I used SVD on the entire matrix of document vectors and reduced the tf-idf vectors to 512 dimensions.

4.1.3 Average of Sentence Embeddings

The embeddings average model managed to receive an accuracy of 44.1% and a macro F_1 score of 28.2%, when trained for 20 epochs, with a batch size of 32 and a maximum learning rate of $3.33 \cdot 10^{-4}$. The precision and recall was very low for the smaller classes. Oversampling while keeping the number of epochs and the learning rate increased the accuracy and F_1 scores to 39.1% and 35.1%, respectively.

4.1.4 LSTM on Sentences

An LSTM model using a one-layer LSTM network with a hidden size of 128 achieved an accuracy of 61.6% and a macro F_1 score of 55.8%, with a batch size of 32 and a maximum learning rate of $3.33 \cdot 10^{-4}$. This model also had a 50% dropout after the LSTM layer.

When using a two-layer LSTM with a hidden size of 128 and a dropout probability of 50% between the layers, the accuracy and F_1 scores increased to 63.4% and 58.1%, respectively.

4.1.5 LSTM with Attention

Using attention, but only on the sentence to document level, produced similar results: a macro F_1 score of 56.0% and an accuracy of 61.9%. This model had a hidden size of 128, an attention size of 256, and 2 layers in the LSTM with a dropout probability of 30% between them. It was also trained on a batch size of 32 and with a maximum learning rate of $3.33 \cdot 10^{-4}$.

4.1.6 KB-BERT-HAN

KB-BERT-HAN achieves at most an accuracy of 72.9% and a macro F_1 score of 68.5% when trained on seven years of speeches. For this model, the hidden size was 128, the attention size 256 and there was a dropout probability of 30% between the two layers of each LSTM layer. I trained it on batches of 32 examples, for 20 epochs, with a maximum learning rate of 3.33 $\cdot 10^{-4}$. When evaluated on oversampled sets, the HAN produced very similar results. It also achieved practically the same scores when trained on 29 years of data instead, for 10 epochs. The results for different datasets and hyperparameters can be seen in Table 4.2.

A table illustrating the performance of the model per class can be seen in Table 4.3. Confusion matrices displaying how many examples there were of each predicted and true label can be seen in Figure 4.1.

Training set	Max lr	Epochs	Accuracy	Macro F_1
2014/15-2020/21	$3.33 \cdot 10^{-4}$	20	72.4%	67.4%
2014/15-2020/21	$1 \cdot 10^{-3}$	20	**72.9%**	**68.5%**
2014/15-2020/21, refiltered	$3.33 \cdot 10^{-4}$	20	60.1%	50.4%
1993/94-2020/21	$3.33 \cdot 10^{-4}$	10	71.8%	67.2%
1993/94-2020/21	$1 \cdot 10^{-3}$	10	72.0%	67.2%

Table 4.2: The macro F_1 and accuracy scores with different hyperparameters and datasets for KB-BERT-HAN.

Class	Precision	Recall	F_1-score	Support
S	0.8460	0.8328	0.8394	2052
M	0.6808	0.7508	0.7141	963
SD	0.7096	0.7229	0.7162	747
V	0.7013	0.7094	0.7053	523
MP	0.4957	0.6410	0.5591	454
C	0.7197	0.5723	0.6376	498
KD	0.6590	0.6005	0.6284	428
L	0.7716	0.6043	0.6778	369
Accuracy			0.7290	6034
Macro avg	0.6980	0.6792	0.6847	6034
Weighted avg	0.7356	0.7290	0.7299	6034

Table 4.3: The classification report after evaluating KB-BERT-HAN, trained on speech data from 2014/15 to 2020/21 and evaluated on speech data from 2021/22.

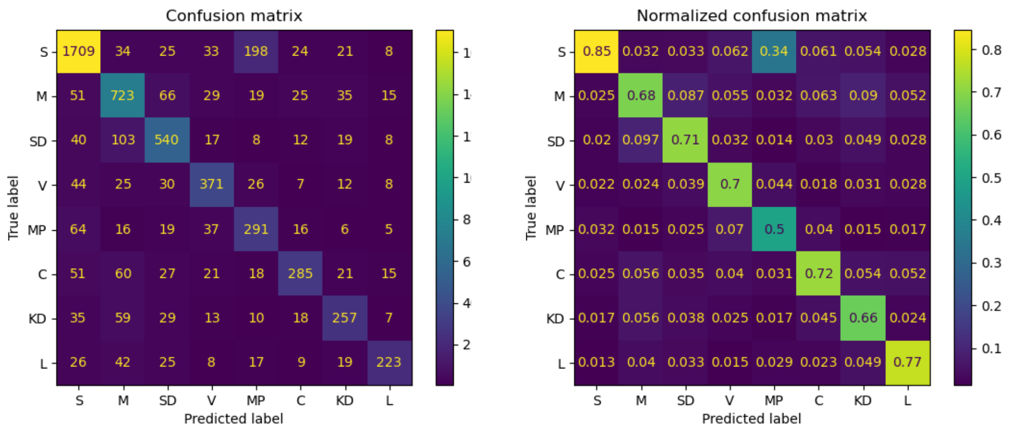


Figure 4.1: Unnormalized and normalized confusion matrices for KB-BERT-HAN, trained on speech data from 2014/15 to 2020/21. The normalized matrix is normalized according to the columns, which means that the values on the diagonal are the precision values for each class.

When I trained the model on the dataset where the party names and affiliations had been

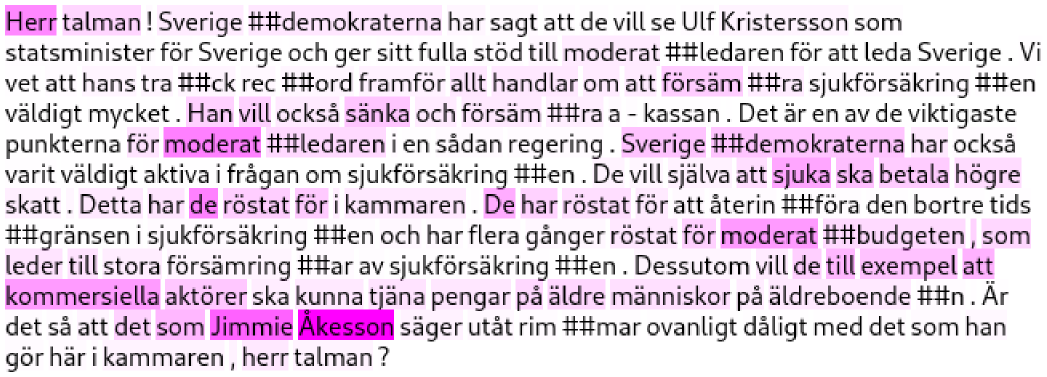

Herr talman ! Sverige ##demokraterna har sagt att de vill se Ulf Kristersson som statsminister för Sverige och ger sitt fulla stöd till moderat ##ledaren för att leda Sverige . Vi vet att hans tra ##ck rec ##ord framför allt handlar om att försäm ##ra sjukförsäkring ##en väldigt mycket . Han vill också sänka och försäm ##ra a - kassan . Det är en av de viktigaste punkterna för moderat ##ledaren i en sådan regering . Sverige ##demokraterna har också varit väldigt aktiva i frågan om sjukförsäkring ##en . De vill själva att sjuka ska betala högre skatt . Detta har de röstat för i kammaren . De har röstat för att återin ##föra den bortre tids ##gränsen i sjukförsäkring ##en och har flera gånger röstat för moderat ##budgeten , som leder till stora försämring ##ar av sjukförsäkring ##en . Dessutom vill de till exempel att kommersiella aktörer ska kunna tjäna pengar på äldre människor på äldreboende ##n . Är det så att det som Jimmie Åkesson säger utåt rim ##mar ovanligt dåligt med det som han gör här i kammaren , herr talman ?

Figure 4.2: A visualization of the word weights for each sentence in an example speech. Higher sentence and word weights, multiplied, correspond to a stronger color.

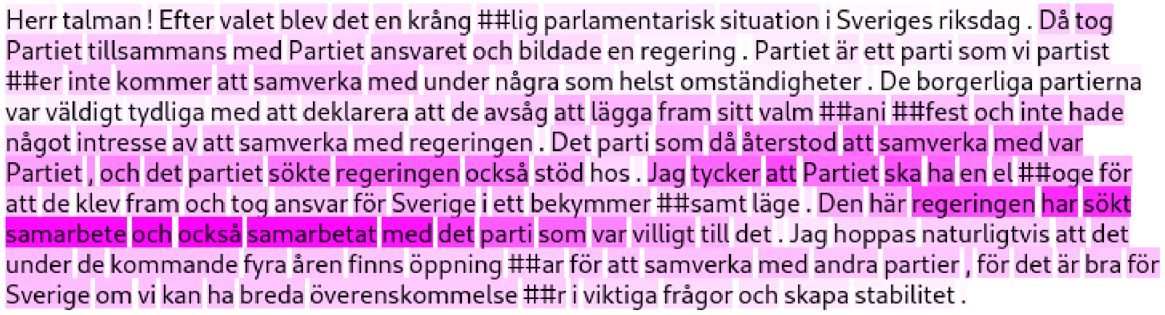

Herr talman ! Efter valet blev det en krång ##lig parlamentarisk situation i Sveriges riksdag . Då tog Partiet tillsammans med Partiet ansvaret och bildade en regering . Partiet är ett parti som vi partist ##er inte kommer att samverka med under några som helst omständigheter . De borgerliga partierna var väldigt tydliga med att deklarera att de avsåg att lägga fram sitt valm ##ani ##fest och inte hade något intresse av att samverka med regeringen . Det parti som då återstod att samverka med var Partiet , och det partiet sökte regeringen också stöd hos . Jag tycker att Partiet ska ha en el ##oge för att de klev fram och tog ansvar för Sverige i ett bekymmer ##samt läge . Den här regeringen har sökt samarbete och också samarbetat med det parti som var villigt till det . Jag hoppas naturligtvis att det under de kommande fyra åren finns öppning ##ar för att samverka med andra partier , för det är bra för Sverige om vi kan ha breda överenskommelse ##r i viktiga frågor och skapa stabilitet .

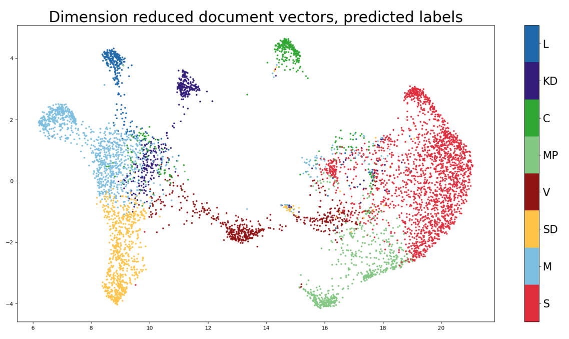
Figure 4.3: A visualization of the word weights for each sentence in an example speech where party names have been replaced with a generic word. Higher sentence and word weights, multiplied, correspond to a stronger color.

filtered out from the speeches, the performance of the model decreased. When trained with the same model parameters as the best performing KB-BERT-HAN prior to changing the dataset, it achieved a macro F_1-score of 50.4% and an accuracy of 60.1%. The precision and recall values were unbalanced over the classes, with S achieving a precision of 80.7% and a recall of 80.4% while MP achieved a precision of 34.0% and a recall of 39.2%.

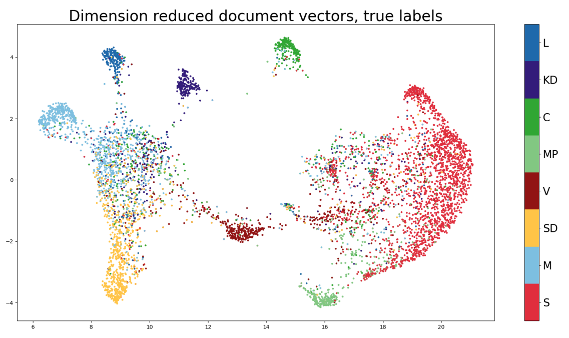
4.2 Results from Analyzing KB-BERT-HAN

An example of a visualization of attention weights for words and sentences can be seen in Figure 4.2. Another example, where party names have been filtered out, can be seen in Figure 4.3.

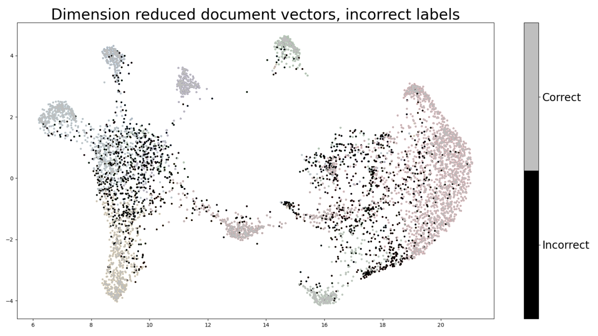
A UMAP visualization of the document vectors can be seen in Figure 4.4.



(a)



(b)



(c)

Figure 4.4: A visualization of the different document vectors for the best performing KB-BERT-HAN, reduced to two dimensions using UMAP. In a), the colors match the predicted label, and in b), they match the true label, and in c) the incorrect and correct predictions are shown.

Chapter 5

Discussion

In the following, I compare the models to each other according to how they performed, and discuss what these differences are based on. I also discuss results from analyzing the inner states of KB-BERT-HAN.

5.1 Differences between the Models

The results indicate that the LSTM models are superior to both the tf-idf model and the embedding average model in solving the task. The best tf-idf model predicts correctly more than 55% of the time, compared to the 12.5% a random guess would give. It therefore clearly manages to find some distinctions between different parties. It works on the assumption that words that occur in the same documents are similar and places all documents in a semantic space according to their tf-idf vectors. However, it lacks a way of relating different words within individual sentences to another, and therefore does not take the nearest context into account. In this sense it also does not get a picture of the meanings of individual sentences. Interestingly, training on the larger training set makes the model perform worse. This may be due to the fact that political discourse varies over time, and that the same party may have a completely different narrative further back in time.

Most of the models using static sentence embeddings and LSTMs (the embedding average model being the exception) achieve accuracy and F_1 scores notably higher than the baseline model. However, the fact that the embeddings are static for the whole sentences makes it harder to take information within the sentences into the political context. KB-BERT-HAN, on the other hand, is able to change its representation of sentences through encoding the words and putting different amounts of attention on different words, before relating sentences to each other, which has made it even more accurate. It outperforms the tf-idf model by 21.7 percentage units, macro F_1-wise.

5.2 Analysis of the Results from KB-BERT-HAN

KB-BERT-HAN has a high accuracy, and somewhat balanced F_1 scores for all classes, meaning that it is able to clearly distinguish any of the eight parties from the others a majority of the time.

5.2.1 Document Vectors

It can be seen in Figure 4.4 that KB-BERT-HAN has placed documents corresponding to S and MP, who have been in the same governments throughout most of the dataset from 2014 until 2022, quite close together in the space. The second and third biggest parties, M and SD, are also quite closely connected. The parties which were in the same government as M from 2006 until 2014, L, C and KD, are mostly separated from M, and C is especially far away. V, which has not been in a government throughout the period of the whole dataset, is partly connected to and partly separated from S and MP.

5.2.2 Misclassification Analysis

The class with the lowest precision is MP, and the single biggest misclassification by a great margin is misclassifying MP as S (34%, see Figure 4.1). This is not surprising since these two parties have been in the same government for the years from which the data was taken. Interestingly, S is very seldom misclassified as MP. This may be due to the fact that S has a much bigger representation in the dataset. These misclassifications are also consistent with the look of the document vector plots.

The second largest misclassification is misclassifying SD as M, followed by misclassifying M as KD and M as SD. This is also not surprising, as M, C and KD have been in the same governments, and SD is cooperating with the current M-led government.

5.2.3 Qualitative Evaluation and Refiltration of Data

From attention plots like the one in Figure 4.2 it becomes clear that the model puts a lot of attention on names of persons and parties, as well as political and societal concepts. The dataset I used initially contains a lot of sentences where the actual party speaking is mentioned, such as "Vänsterpartiet anser att den utredning som regeringens förslag bygger på inte har visat på något behov av att införa dessa nya brott." ("The Left party believes that the investigation which the proposal of the government is built on has not shown any need for introducing these new crimes."). In examples with sentences like this, it can be seen that the model puts clear emphasis (attention) on these sentences, and that within these sentences there is also a clear emphasis on the words indicating the party, such as "Vänsterpartiet", "vi moderater", etc. The speech shown in Figure 4.2 is also an example of this. This is an interesting find because it confirms that the model has been able to make a connection between these terms and the party speaking. It could however also be seen as a flaw, since the idea is to find *underlying* differences in the rethoric between different parties. It could be argued that mentioning the party itself in the speech makes it a bit too easy for the model to predict (even though other parties are also mentioned), without understanding the speeches deeply. As I did not discover the magnitude of this problem until late in this project, it was not possible to try the new dataset thoroughly on the model. However, the results from training a new KB-BERT-HAN with the same model parameters as the best one, on this new dataset, shows that it is a more difficult task if all these clues are removed. The unbalance in F_1-scores per class for this model suggests that the model would benefit from oversampling.

5.3 Comparison to Related Work

I have described prior experiments similar to this, in which texts are also classified by political affiliation, in section 1.4. The biggest difference between the models I have used and those described in that section, is the use of LSTMs and attention, and especially hierarchical attention in combination with BERT embeddings. On the other hand, I did not finetune the BERT models in this project. The results of the simpler models based on sentence embeddings perform worse than the best results in Doan et al. (2022) and Doan et al. (2023). However, KB-BERT-HAN gets comparable and in some cases better results.

5.4 Technical Difficulties and Necessary Resources

The models require memory, processing power and storage increasing with the complexity. I noted the times required to complete one epoch of training for each model, when trained on a node in COSMOS, using 8 tasks. The results can be seen in Table 5.1.

Model	Iteration time	Epoch time
Average of sentence embeddings	135 it/s	20 s/epoch
One-layer LSTM	104 it/s	26 s/epoch
Two-layer LSTM	73 it/s	37 s/epoch
LSTM with attention	60 it/s	45 s/epoch
KB-BERT-HAN (pre-computed embeddings)	6.0 it/s	7 min 30 s/epoch*
KB-BERT-HAN (embeddings computed each iteration)	1.2 it/s	37 min/epoch

Table 5.1: The amount of iterations per second as well as the time one epoch takes when training on the dataset featuring speeches from 2014 to 2021 for the different BERT based models. Each iteration is here a batch of 32 speeches.
*Estimated time, since it was not possible to run this model on such a big dataset.

KB-BERT-HAN is especially memory intensive, and on the COSMOS node these models were run on it was not possible to store all word embeddings in the available 80 GB of GPU memory. This is the reason why this model is much slower than the other models, and a KB-BERT-HAN which uses pre-computed word embeddings is 5 to 6 times quicker. With

GPU memory, this is something which could be done even with a large dataset like the one featuring 7 years of speech data.

5.5 Answers to the Research Questions

(RQ1) In this report, I have developed and evaluated neural models with varying degrees of complexity. The LSTM and attention mechanism prove practical when constructing a model to handle text sequences, which is reflected in the performance of the models utilizing these techniques.

The results show that neural models using LSTMs and attention mechanisms are able to distinguish quite clearly between narratives from different political parties, given that they have access to a sufficient amount of data, and that they are sufficiently complex. The results indicate that a model benefits from being able to acquire and be attentive to information not only on the sentence level but also on the word level, as KB-BERT-HAN vastly outperforms the other models.

(RQ2) All neural networks I evaluated except the embedding average model get better scores than the baseline model. This is especially notable with KB-BERT-HAN getting an F_1 score 21.7 percentage units above the baseline model.

(RQ3) It is possible to partly explain KB-BERT-HAN. Through the UMAP algorithm, applied on the document vectors computed by KB-BERT-HAN, it is possible to see how close different parties are according to the model. Since KB-BERT-HAN operates on the word level, it is also possible to get an insight into it and explain it. Through visualizing its attention weights on the word and the sentence level, we partly understand how the model reaches its conclusions.

Chapter 6
Conclusion

In this report, I have combined large language models, and in particular BERT models, with LSTM networks and Hierarchical Attention Networks in order to attempt to distinguish between different party affiliations. The results show that these models outperform baseline models based on tf-idf, and that KB-BERT-HAN, utilizing Swedish BERT word embeddings as well as hierarchical attention, is particularly accurate, outperforming tf-idf by a substantial margin. The reason for this is the fact that it is able to change its representation of language down to the word level. Due to its hierarchical structure it can relate both different words and sentences to each other as well as put more attention on some words or sentences than others. This model is also interpretable in that the attention weights can be visualized to see the parts of documents and sentences the model focuses more on. From the model's internal representation of documents we can also visualize how they relate to each other using a dimension reduction algorithm, and use these relations to tell something about differences and similarities between different political affiliations. Analyses like this can be used further in applications were one wants to get a picture of different political narratives.

6.1 Future Work

Due to time constraints and an already quite high complexity in KB-BERT-HAN, there were various ideas that arose, which I never implemented. On the technical side, the models could be optimized. Given enough storage or memory, KB-BERT-HAN could also utilize pre-computed embeddings, like the other models, which would allow for faster training.

It would also be interesting to analyze the differences in speech content through time, and perhaps accomodate for these differences when constructing models. Since narratives within the same party affiliation sometimes changes a lot over time, it might not be useful to train the models on close to 30 years of data. After all, both tf-idf and KB-BERT-HAN actually performed better when trained on just 7 years of data.

There are also improvements to KB-BERT-HAN itself which could be made to possibly achieve higher scores. I did not explore the possibility of finetuning in this report, but the KB-BERT-HAN model could utilize finetuning on the embedding layers within BERT. Furthermore, the model could be divided into more hierarchies. A lot of the speech data available is divided into paragraphs. Introducing this level of hierarchy, leading to a word-sentence-paragraph-document hierarchy, might lead to better predictions. This is due to the fact that it may be the case that paragraphs contributes roughly equally to the overall semantic content of the document, despite being of different lengths. It is also motivated to make such a division since the speeches are between 23 and 24 sentences long on average.

Most of all, it would be interesting to *explain* KB-BERT-HAN more thoroughly as well. In the visualizations seen in Figure 4.2 it becomes clear what the model puts most attention on. However, this does not explain which words and sentences contribute the most to predicting a given class, as opposed to all the other possible classes. This could however be done using packages like SHAP (Lundberg and Lee, 2017), and would make the model more practically usable in applications.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate. arXiv:1409.0473 (visited on 2023-09-18).

Bengio, Y. (2009). *Learning Deep Architectures for AI*. NOW Publishers, Delft.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Berkson, J. (1944). Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227):357–365.

Cohen, W. W. (1996). Learning rules that classify e-mail. *AAAI spring symposium on machine learning in information access*, 18:25.

Dahllöf, M. (2012). Automatic prediction of gender, political affiliation, and age in Swedish politicians from the wording of their speeches—A comparative study of classifiability. *Literary and Linguistic Computing*, 27(2):139–153.

Deng, L. and Liu, Y. (2011). *Deep Learning in Natural Language Processing*. Springer Verlag, Singapore.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805v2 (visited on 2023-08-03).

Doan, T. M., Kille, B., and Gulla, J. A. (2022). Using language models for classifying the party affiliation of political texts. In *Natural Language Processing and Information Systems*, pages 382–393, Cham. Springer International Publishing.

Doan, T. M., Kille, B., and Gulla, J. A. (2023). Sp-bert: A language model for political text in scandinavian languages. In Métais, E., Meziane, F., Sugumaran, V., Manning, W., and Reiff-Marganiec, S., editors, *Natural Language Processing and Information Systems*, pages 467–477, Cham. Springer Nature Switzerland.

Dumais, S. T. (2004). Latent semantic analysis. *Annual Review of Information Science and Technology (ARIST)*, 38:189–230.

Engel, J. (1988). Polytomous logistic regression. *Statistica Neerlandica*, 42(4):233–252.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930-1955. In *Studies in Linguistic Analysis*.

Fukushima, K. (1969). Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333.

Harris, Z. S. (1954). Distributional structure. *Word*, 10:146–162.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–1780.

Ivakhnenko, A. G. and Grigorevich, L. V. (1967). *Cybernetcis and forecasting techniques*. American Elsevier Pub.

Joos, M. (1950). Description of Language Design. *The Journal of the Acoustical Society of America*, 22(6):701–707.

Jordan, M. I. (1986). Serial order: A parallel distributed processing approach. *ICS Report 8604*.

Khurana, D., Koli, A., Khatter, K., and Singh, S. (2022). Natural language processing: State of the art, current trends and challenges. *Multimedia Tools Appl.*, 82(3):3713–3744.

Kim, J. (2019). Pytorch-hierarchical-attention-network. `https://github.com/JoungheeKim/Pytorch-Hierarchical-Attention-Network`. (visited on 2023-08-05).

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization. arXiv:1412.6980 (visited on 2023-09-20).

Kohavi, R. and Provost, F. (1998). Glossary of terms. *Machine Learning*, 30:271–274.

Kolen, J. F. and Kremer, S. C. (2001). *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243.

Lemaître, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

LUNARC (2023). Cosmos. `https://www.lunarc.lu.se/systems/cosmos/`, (visited on 2023-08-02).

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Malmsten, M., Börjeson, L., and Haffenden, C. (2020). Playing with words at the national library of sweden – making a swedish bert. arXiv:2007.01658 (visited on 2023-07-27).

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge.

McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426 (visited on 2023-08-03).

Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, Red Hook, NY, USA. Curran Associates Inc.

Nasukawa, T. and Yi, J. (2003). Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd International Conference on Knowledge Capture*, K-CAP '03, pages 70—-77, New York, NY, USA. Association for Computing Machinery.

Nilsson, N. J. (1982). *Principles of Artificial Intelligence*. Springer Berlin, Heidelberg.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. arXiv:1912.01703 (visited on 2023-07-26).

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

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

R. C. Prati, G. E. B. and Monard., M. C. (2009). Data mining with imbalanced class distributions: concepts and methods. *Indian International Conference Artificial Intelligence*, pages 359–376.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv:1908.10084 (visited on 2023-07-27).

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408.

Ruder, S. (2017). An overview of gradient descent optimization algorithms. arXiv:1609.04747v2 (visited on 2023-07-12).

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

Saeed, J. I. (2016). *Semantics*. Wiley Blackwell, 4th edition.

Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv:1402.1128 (visited on 2023-08-04).

Salton, G. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, Inc., USA.

Sammut, C. and Webb, G. I. (2010). *Encyclopedia of Machine Learning*. Springer.

Santos, A. M., Canuto, A. M. P., and Neto, A. F. (2011). A comparative analysis of classification methods to multi-label tasks in different application domains. *International Journal of Computer Information Systems and Industrial Management Applications*, 3:218–227.

Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. arXiv:1912.05911 (visited on 2023-07-12).

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Sveriges Riksdag (2023a). Anföranden. `https://data.riksdagen.se/Data/Anforanden/` (visited on 2023-05-30).

Sveriges Riksdag (2023b). Partierna i riksdagen efter valet 2022. `https://www.riksdagen.se/sv/teckensprak/sa-fungerar-riksdagen/partierna-i-riksdagen-efter-valet-2022/` (visited on 2023-07-31).

Sveriges Riksdag (2023c). Tidigare regeringsbildningar och statsministrar. `https://www.riksdagen.se/sv/sa-fungerar-riksdagen/demokrati/sa-bildas-regeringen/tidigare-regeringsbildningar-och-statsministrar/#regeringar-sedan-1876-10` (visited on 2023-07-31).

Tian, Y., Su, D., Lauria, S., and Liu, X. (2022). Recent advances on loss functions in deep learning for computer vision. *Neurocomputing*, 497:129–158.

Van Houdt, G., Mosquera, C., and Nápoles, G. (2020). A review on the long short-term memory model. *Artificial Intelligence Review*, pages 5929–5955.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need. arXiv:1706.03762 (visited on 2023-07-30).

Walker, S. H. and Duncan, D. B. (1967). Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54(1/2):167–179.

Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144. arXiv:1609.08144 (visited on 2023-09-19).

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168(2).

Appendices

Appendix A
Source Code, Libraries and Image Credit

The Python code written in order to construct the datasets as well as implement, train and test all models described in this report is available on GitHub: `https://github.com/ErikKjellberg/Speech-Classification`. Aside from this there is also a Jupyter notebook which can be used to visualize some examples with KB-BERT-HAN, without needing to train it.

Except the built-in Python libraries, the following libraries were also used:

- PyTorch: `https://pytorch.org/`
- Scikit-learn: `https://scikit-learn.org/stable/`
- NumPy: `https://numpy.org/`
- Beautiful Soup: `https://www.crummy.com/software/BeautifulSoup/`
- SpaCy: `https://spacy.io/`
- Imbalanced-learn: `https://imbalanced-learn.org/stable/`
- Transformers: `https://github.com/huggingface/transformers`
- SentenceTransformers: `https://www.sbert.net/`
- Matplotlib: `https://matplotlib.org/`

Front-page image credit: Ray Swi-hymn / CC-BY-SA-3.0. Url: `https://commons.wikimedia.org/wiki/File:20180625_Riksdagshuset_7968_(48413007596).jpg`