Deciphering Sweden's Political Language

Niclas Herzing
Fredrik Lindholm

December 2023

Introduction

This project was made as a part of the course DD1418 Language Engineering with Introduction to Machine Learning at KTH Royal Institute of Technology. This particular project aims to create machine learning models that can classify texts as being affiliated to one of the eight Swedish parliament parties.

Background

We chose to look at Swedish politics, and tried to use various methods for word vectorization and models to classify text as belonging to any of the eight political parties in the Swedish parliament: Centerpartiet, Kristdemokraterna, Liberalerna, Miljöpartiet De Gröna, Moderaterna, Socialdemokraterna, Sverigedemokraterna, and Vänsterpartiet. The idea was to train machine learning models to recognize patterns found in texts from each party. We decided on trying a few different methods which are described further below. This would also allow us to evaluate these methods against each other to see which one performs the best.

Implementation

Data

The data consists of all party programs and election manifests from 1897 to current date. These were extracted from the website 'Språkbanken'[1]. Five more texts were also added two for 'Socialdemokraterna'[2] and three for 'Centerpartiet'[3], both taken from the respective parties' websites. These texts were added after the implementation of our data processing which showed that both of those parties had far fewer data points. Even after adding five more texts the number of texts per party was still only around 15, which we adress before our data processing. All of the texts are cut into segments of 2500 characters and inserted into the dataset with the same labels as they had before being cut into segments. This gave far more data points for each party than what we had before. The data was still however very imbalanced with some parties having twice as many datapoints as others due to the different parties writing different lengths of texts. We balance the data by setting all parties to have the same amount of data points, the removal is based on which year the text is from. This was decided since the newer texts should be more closely aligned with the party's current views. The figure below shows the most common words in the party platforms for the last Swedish election 2022 which is one of the datapoints in our set. The text used to create the figure was passed through our data processing function, with lemmatization, described in the next section.

Data Processing

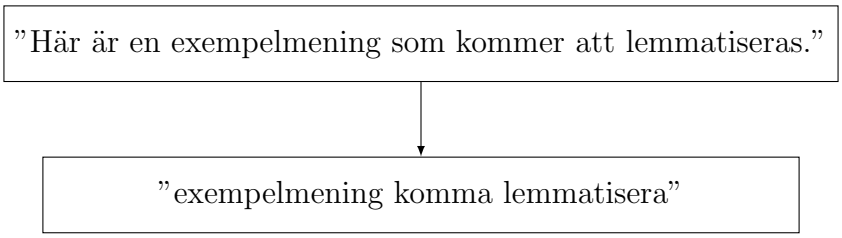
Before doing any data processing the set was split into train and test sets, 80% for training and 20% for testing. The input text from the documents both had to be 'cleaned' and lemmatized. With cleaning, we mean removing symbols such as commas and hyphens, free standing numbers were also removed. Stopwords were also removed which are the most common words in the Swedish language. All characters are also rewritten as lowercase and we decided to remove all party names from the texts. After that we create two datasets, one where the text is lemmatized

[1] https://spraakbanken.gu.se/resurser

[2] https://www.socialdemokraterna.se/var-politik/partiprogram-och-riktlinjer

[3] https://www.centerpartiet.se/vart-parti/partistamma-och-stammobeslut/2021

and one that remains the same. This is why we split our data into training and testing sets before the processing part since we want both the text with and without lemmas to be on the same datapoints. Lemmatization is the process of reducing words to their standard form. This means that there will be a higher frequency of the same word since they are all in standard form. Even though lemmatization changes a lot of the content of a text the context is not lost. The result of this process can be represented in the following figure:



As can be seen, common words such as ’Här’, ’är’, ’en’, ’som’, and ’att’ have been removed, and the remaining words have been reduced to their standard form. The decision to create two sets of texts, one with lemmas and one without is to see whether that will have an effect on the performance of the machine learning models.

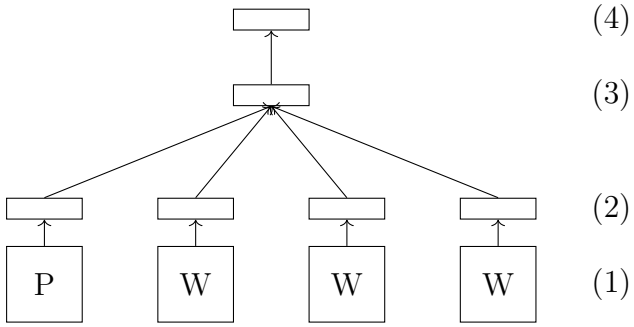
Word Vectorization

Once the data had been processed, cleaned and lemmatized, all words needed to be vectorized. This was done through two methods, since it wasn’t decided what lengths all the texts would be at this point. We employed the following two methods for vectorization, found in an article online[4]. The two methods for word-vectorizations are explained below.

1. PV-DM

PV-DM is short for Distributed Memory Model of Paragraph Vectors. It vectorizes words through the process in the below figure:

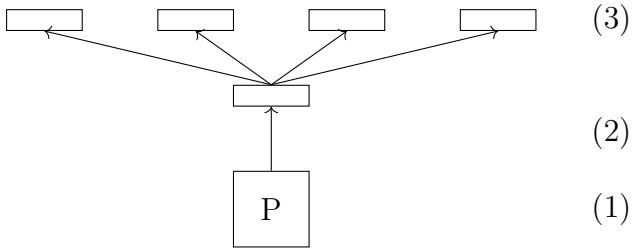
[4]https://medium.com/@danilo.najkov/detecting-political-bias-in-online-articles-using-nlp-and-classification-models-c1a40ec3989b



One website gives the following explanation for steps (1) to (2): "[...] every paragraph is mapped to a unique vector, represented by a column in matrix P and every word is also mapped to a unique vector, represented by a column in matrix W [...] The paragraph vector is shared across all contexts generated from the same paragraph but not across paragraphs". In step (2) to (3), the word vectors from the matrices W and the paragraph vector from matrix P are concatenated in order to finally classify / predict the next word in step (4)[5]. This method is well-suited for large amounts of text.

2. PV-DBOW

The next vectorization method is called PW-DBOW, and is short for Distributed Bag of Words version of Paragraph Vector. Similar to PV-DM, it also has a Paragraph Matrix, but it differs in its mechanics, as seen below:



Again, we have the paragraph matrix P, in which each paragraph in the text is represented by a column, where the paragraph vector is shared across all contexts generated from the same paragraph but not across paragraphs. Instead of predicting a focus word and from its context, this method predicts the context based on a focus word. This is very similar to the SkipGram method, and is well-suited for short amounts of text[6]. At (1) we have our paragraph matrix. At (2), we extract one row vector, which corresponded to a paragraph. In (3), based on some focus word, the model aims to predict its context words.

Models

This study applies three models: Naïve-Bayes, Random Forest, and a Neural Network Model. On all three models, both methods of word vectorization and text processing were tested. We wanted to see which of the methods perform best, and whether or not some of the models would perform worse than the others. The comparison would then include both sets of text processing, both types of vectorization, trained and tested on all three machine learning models presented below.

[5]https://sh-tsang.medium.com/review-distributed-representations-of-sentences-and-documents-doc2vec-86ef911d4515

[6]https://sh-tsang.medium.com/review-word2vec-efficient-estimation-of-word-representations-in-vector-space-f9dbe2145afa

Naïve-Bayes

As has been mentioned in this course, Naïve-Bayes is built on Bayes theorem, where the aim is to maximise the class y (in our case a political party) based on some word features $w_1, ...w_n$:

$$\underset{y}{\text{argmax}}\ P(y|w_1, ..., w_n)$$

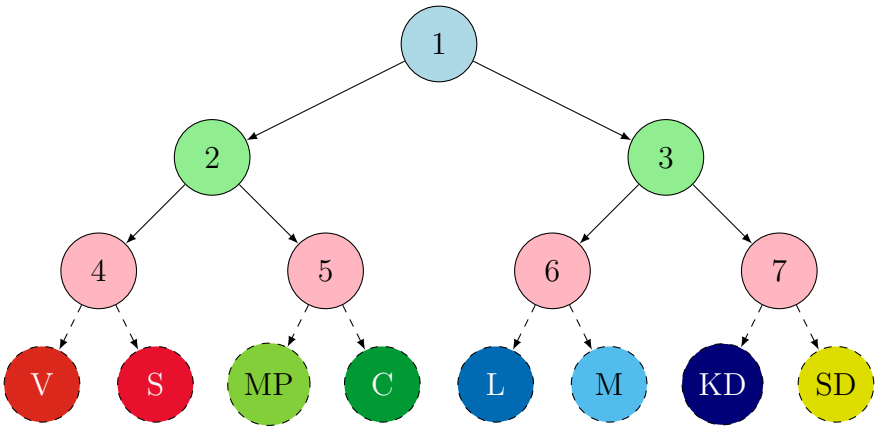
In this study, the number of features are $n = 300$. Naïve-Bayes then uses Bayes Theorem, and assumes independence between the features $w_1, ...w_n$, resulting in the following:

$$\underset{y}{\text{argmax}}\ P(y|w_1, ..., w_{300}) = \underset{y}{\text{argmax}}\ P(w_1|y)...P(w_{300}|y)P(y)$$

These factors are then estimated using Maximum-Likelihood[7]. The y that maximises the expression above is the political party that the model predicts has written the text. Implementing this in code, the library scikit-learn was used, by defining functions in which the training and test data were passed to built-in functions from that library. The model was initialised with GaussianNB(), to which the training data was passed. The test data was then predicted with a built-in method, and lastly, the model was evaluated on accuracy, and with a confusion matrix.

Random Forest

The Random Forest Model implements multiple decision trees (hence the name Random Forest), represented below:



One begins in (1), makes a decision to move to either (2) or (3), and so on, and after a certain amount of levels n, the decision tree has come to a leaf node corresponding to one of the eight classes. In practice, the algorithm can be explained as follows:

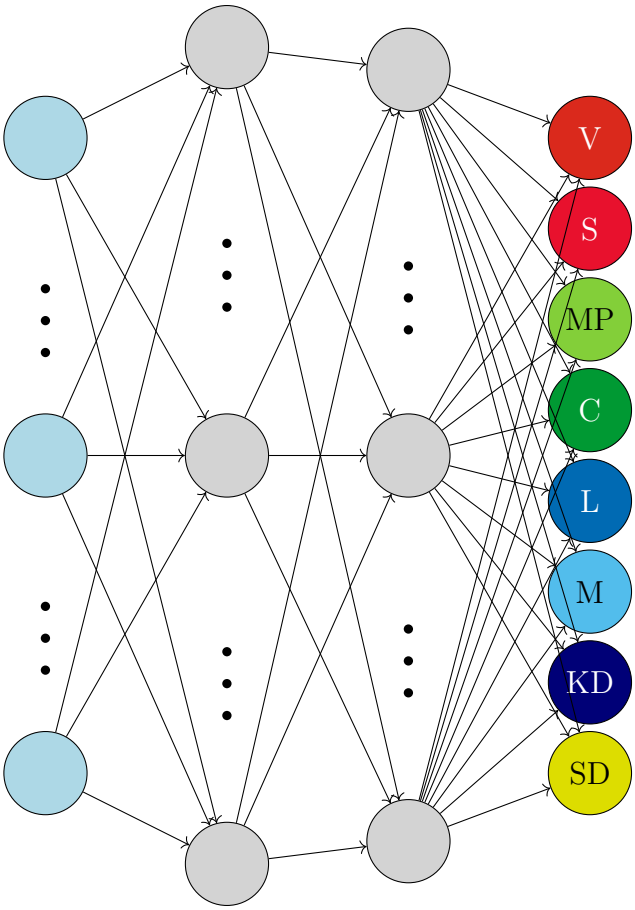
- First, we had m data points with features $x_1, ..., x_k$, and a class y.
- Next, *bootstrapping* was used, meaning from this data set new data sets were created, also with m data points but with a random sample of the original data points. This meant that in the new data set a data point could occur multiple times.
- On these new data sets it was then randomised which of the features $x_1, ..., x_k$ the random decision tree as depicted above would use in its 'decision-making'.
- Finally, the class that this algorithm deemed correct was made by a majority vote by the outcomes of these decision trees. This is called *aggregating*[8].

[7] https://www.csc.kth.se/ jboye/
[8] https://www.ibm.com/topics/random-forest

By using multiple random forest decision trees the algorithm prevents over-fitting (i.e., low bias but high variance) of the data set and can make more accurate predictions[9]. Implementing this in code scikit-learn was once again used, see the appendix for the code.

Neural Network Model

As has been taught in this course, the idea of neural networks is to connect multiple binary logistical regression models, where the output of one model becomes the input to the next model, in the next layer. This can be illustrated below:



Input layer 200 nodes | Hidden layer 128 nodes | Hidden layer 96 nodes | Output layer 8 nodes

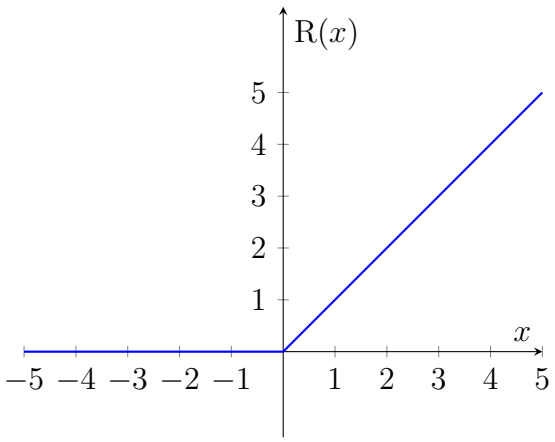
Usually, the sigmoid function is used as *activation function* between the layers:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

however, our model uses the RELU function instead:

$$R(z) = \max(0, z)$$

[9] https://en.wikipedia.org/wiki/Random_forest



and then the softmax function is used in the output layer:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

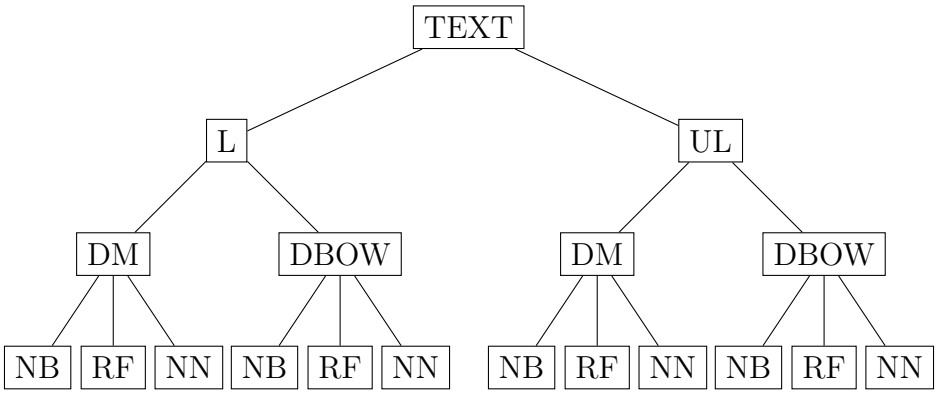
To implement this model, there were quite a few parameters that needed to be adjusted to give the best accuracy scores while still maintaining low loss levels for both training and validation data. We also made sure that the validation loss did not go up as the training loss went down, since this is a clear indicator of an overfitted model. Here are the parameters that needed to be 'optimized', and what they were eventually set to:

Epochs: The number of times the training algorithm works through the entire data set, and adjusts its parameters. To little epochs, and the model has low accuracy - too many epochs, and the model may become overfitted. We set the number of epochs to 5, based on testing several cases, balancing between accuracy and not overfitting the model. We used the comparison between training and validation loss as a metric of overfitting; if the training loss goes down while the validation loss goes up, then this is a clear indicator of an overfitted model[a].
Hidden layers: How many hidden layers to have between the input and output layers. We set the number of hidden layers to 2. This number was again set by balancing between accuracy and not overfitting the model.
Number of nodes in the hidden layers: We set the amount of nodes in the first hidden layer to 128, and the second hidden layer to 96. This number was again set by balancing between accuracy and not overfitting the model.
Optimisation Model: With optimisation models, we mean methods such as stochastic gradient descent which have been part of this course, and are used for minimising the loss function with respect to the parameters (data weights). However, after doing some research, our model uses an optimisation model called 'Adam', which is a mix of Momentum and RMSprop (Root Mean Square Propagation)[b]. The reason for choosing this method, was because it is simple to implement, computationally efficient, requires little memory, and is appropriate for situations with large data sets and parameters[c].
Activation Function: As has already been mentioned, we used the ReLU function as activiation function between the layers. This was also something that was decided upon through trial and error, to obtain the best accuracy.

[a] Lecture from DD1418
[b] https://towardsdatascience.com/optimisation-algorithm-adaptive-moment-estimation-adam-92144d75e232
[c] https://blog.marketmuse.com/glossary/adaptive-moment-estimation-adam-definition/

Structure of all models

The tree figure below shows all of the models that were trained and tested. The first step in the tree shows the data processing part, where the dataset was split into two, one with lemmas and one without. Each of those is then vectorized using both PV-DM, and PV-DBOW. This results in what we call four 'scenarios' for which we want to train and test the machine learning models. As can be seen in the figure each machine learning model is trained and tested on all 'scenarios', resulting in a total of 12 final models to evaluate.



Methods for Testing and Evaluation

We evaluated the models based on their respective confusion matrices and their accuracy. For the Neural Network Model, we also plotted the training and validation losses against the epochs to find a suitable number of epochs for training.

Results

In the table below, we present the test accuracy scores for each 12 cases.

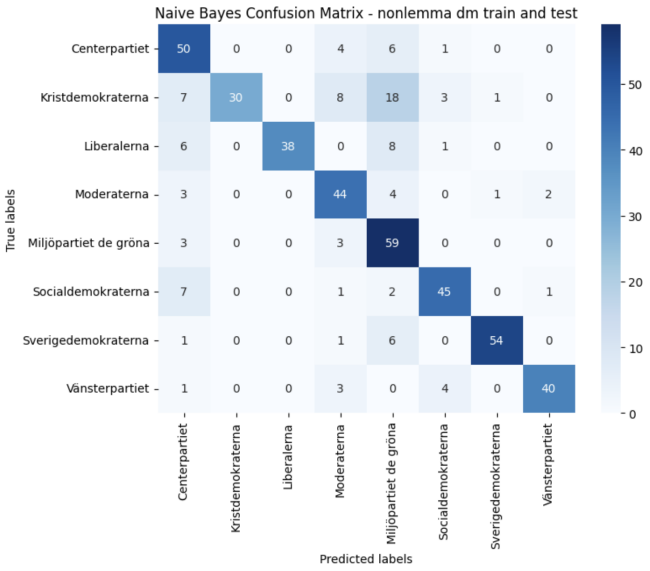
Lemmatized						Unlemmatized					
PV-DM			PV-DBOW			PV-DM			PV-DBOW		
NB	RF	NN	NB	RF	NN	NB	RF	NN	NB	RF	NN
77%	78%	86%	86%	86%	90%	77%	77%	81%	88%	88%	88%

The biggest factor in a higher accuracy score seems to be the vectorization based on the table. All of the models with PV-DBOW perform better when looking at their accuracy.

Next, instead of showing the result of all 12 cases, we will show the results of only the best and worst cases in terms of accuracy.

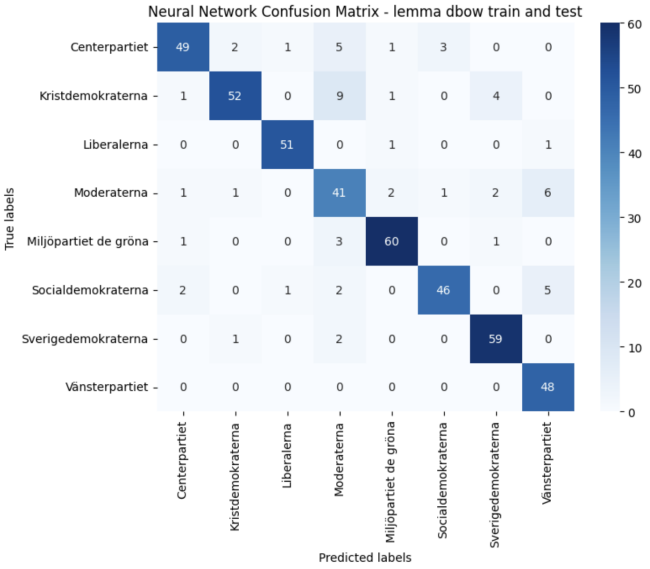
Unlemmatized, PV-DM, Naïve Bayes

As can be seen in the table above, this case gave an accuracy of 77%. This is the corresponding confusion matrix:



Lemmatized, PV-DBOW, Neural Network

As can be seen in the table above, this case gave an accuracy of 90%. This is the corresponding confusion matrix:



To view the rest of the confusion matrices, we invite you into our Google Collaboration, where all results can be viewed, as well as the code in its entirety. The link is found in the Appendix.

Discussion

The Result table shows that the biggest factor in the model performing well was the vectorization method used. All models using PV-DBOW had a higher accuracy than almost all of the models using PV-DM. This is probably a result from our decision to cut the texts into shorter

sections consisting of 2500 characters. Before we implemented the part where we cut the texts into shorter segments PV-DM performed better. The decision to cut the texts into segments was however deemed necessary in order to produce a larger number of data points for training and testing. The result does not show much of a difference between the regular text and the one which has been lemmatized. We expected there to be more of a difference between the lemmatized and unlemmatized; before the models were cut into shorter segments the models performed better on the longer lemmatized texts. Our data processing implementation was also much slower when having to lemmatize the text. Even though lemmatization could sometimes add small gains in the performance of each model the time it takes to lemmatize the text should also be taken into account. Note that this is because of our code implementaion of lemmatization and it could probably be optimized to run faster. As can be seen in the result table, the Neural Network model is consistently more accurate than the other two models, however the difference is not by much, so no conclusions can be drawn regarding which model would be better for other purposes. In this particular case, it is a few percentages better than the other models, but some other conditions, for example other word vectorization methods, could perhaps yield different results.

The data used in this project is very limited and could be extendend in order to produce models that have a better understanding of the parties' views as can be seen from other sorts of texts such as transcripts from parliament speeches or party opinions expressed in articles or social media. This would also add more data points for each party for the models to train on.

The models produced in this project can be used to classify texts as having affiliation to one of the eight Swedish parliamentary parties. The use of this is however only suitable for texts which are known to be of similar origin. Using the model to classify anything other than a political text will give a classification of party, but the result will not mean anything since the text was not of a political nature. Our project could be extended to give meaningful results for any given text. This would require adding a large number of others kinds of texts to the training and testing data, possibly including other kinds of texts containing political messages such as debate transcripts or tweets from politicians. A new model could then be created from that data set which is trained to classify whether or not a text contains anything political. Any given text could then be passed to that model to classify whether that text is of political nature and depending on that result if it is meaningful to pass it through to our existing models which classifies party affiliation.

Conclusion

The best performing model was Neural Network trained on lemmatized text which had been vectorized using PV-DBOW. And the largest contributing factor in achieving a high accuracy score was the choice of vectorization with PV-DBOW performing better. Lemmatizing the text did not contribute to the extent that we expected with the models using lemmatized text only performing slightly better. However as noted in the discussion above, our dataset was very limited and the texts are all of very similar nature with most of them being party platforms or party programmes. An extended dataset containing more nuanced text that still describes the different party views would make the models more suitable for classification of political texts from different platforms such as articles or social media posts.

References

[1] Språkbanken

[2] Socialdemokraterna

[3] Centerpartiet

[4] medium.com

[5] medium.com

[6] medium.com

[7] Lecture 7 DD1418

[8] IBM

[9] Wikipedia

[a] Lecture 8 from the course

[b] towardsdatascience.com

[c] marketmuse.com

Appendix

The path to the Google Collaboration is in this link.