Binary classification of speeches from the Parliament of Estonia

Business Data Analytics, Quantitative Methods and Visualization (BDMAO1023E)

Ralf Joosep Kask

Pages: 15

Characters: 35889

Hand-in Date: 2024-05-22

COPENHAGEN BUSINESS SCHOOL

HANDELSHØJSKOLEN

Contents

1 Introduction

Political debate needs to be transparent and public in order to democracy to thrive. People need to understand decisions being made. Thus, there needs to be supervision and this role is played by unbiased journalism. Tools such as machine learning allow to use data to generate models enabling tasks inefficient for humans. Therefore, utilizing machine learning allows to analyze vast amounts of data from parliaments, where political debate is held. In particular, such models can help to detect bias and/or talking points in political speeches, yielding more transparency, supervision and better democracy as a result.

This paper aims to predict political affiliation and find the important talking points and features of debate held between binary groupings, namely between coalition and opposition in Estonian parliament. The paper does so by exploring different models for finding best describing models in Estonian political context.

The paper is divided into 7 sections. Firstly, the data being used is explained. Then, data preparation is revealed. Thirdly, data modeling section explains briefly utilized models. Fourth section focuses on results, namely hyper parameter tuning and model comparison. Then, discussion (separated to visualizations and improvements sections) and relevance section follow, ending with a wrap-up.

2 Data understanding

The following paper analyzes political landscape of Estonia. Every time an meeting is held in Estonian's legislative organ, The Riigikogu, the verbatim records of the plenary sittings are generated by artificial intelligence software from the audio files and these texts are accessible on the official website of Riigikogu (Riigikogu, n.d.). Due to Riigikogu Rules of Procedure and Internal Rules Act, the working language of the Riigikogu is Estonian and thus the verbatim records are also in Estonian (Riigi Teataja, 2019). Data fed into the models are also in Estonian, results presented here include translations to English. Data taken under the analysis is from the period of 6th of March 2023 to 18th of April 2024, during which government and members of legislative body have not changed. Also, the start of the period is also the start of the fifteenth legislature of the Estonian Parliament after the elections held on 5th of March 2023 (Riigikogu, 2023). As, new government took the office, the stand-off between coalition and opposition was fierce and, thus, might give great ground upon which to analyze the division of rhetoric and ideas of coalition and opposition. Thus, the goal is to predict the political affiliation (coalition or opposition) based on the speech. In particular, the problem is binary classification and the target variable is belonging to the group of coalition or opposition.

Taking a quick look into our data, we can get some basic, yet insightful results right off the bat without any machine learning models. First of all, it is wise to look at how our data is distributed, i.e. for example, the class distribution. This is shown in the figure 1. As there are more instances with target label equal to False (or 0), to ensure that training data also contains instances with target label 1, it might be beneficial to stratify the data in train-test split step.

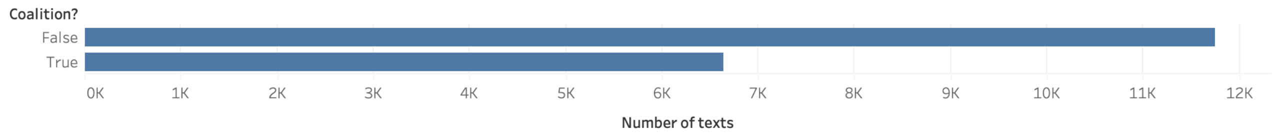

Figure 1: Word count frequency

Next, let us also consider texts based on which our models will have to make predictions. Most straight-forward unit of analysis for text would be single word and thus let us look the distribution of all the word counts. Figure 2 reflects that most words (about 60000 words) have occurrence of 1, which is expected from text data.

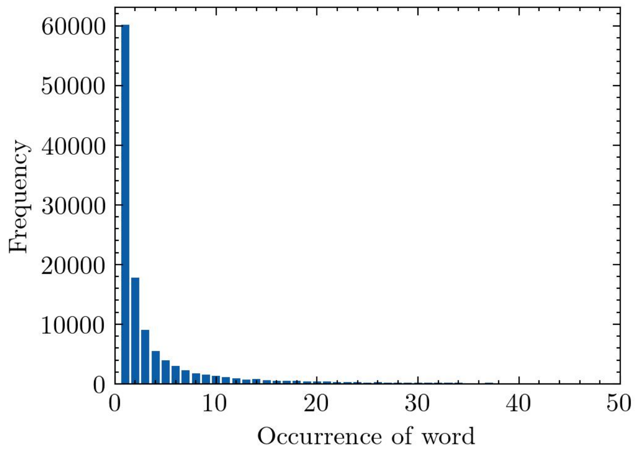

Figure 2: Word count frequency

Considering the words, which occur in our data a lot, the figure 3 shows top 20 most common words. The graph aligns with Zipf's law (Hosch, n.d.). As expected, these are also the most common verbs and nouns as in English. Since those words are common in both types of texts (coalition and opposition) and do not convey any information (political affiliation), it is expected these words are not very useful for make predictions. Thus, we can simply exclude them by use of stopwords or by use of hyperparameter `max_df` in our Vectorizer. Even if we let them into our feature pool, it is still expected these words are not very indicative of class belonging.

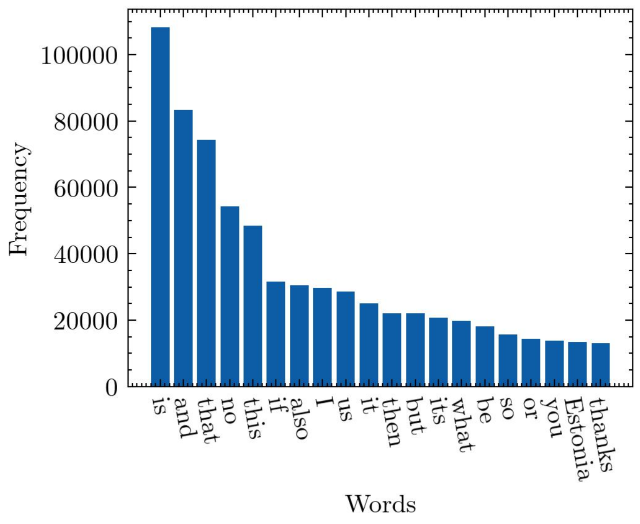

Figure 3: Most occurring words

3 Data preparation

3.1 Data preprocessing

The data extraction was done using simple Python script scraping every HTML page from the URLs fed into the script. Then, every HTML text was split into speeches accompanied with speaker names. Since, every speaker had some political affiliation (coalition or opposition), belonging label (target label) was assigned to each text. In particular, every coalition speech was given target label of 1 and opposition got target label of 0, and the table was formed. Extract of the table is shown on Table 1.

Speech	Target
Ja ravi peame inimestele ikkagi endiselt pakku...	0
Hea juhataja! Head kuulajad! Vähiteema on alat...	1

Table 1: Extract of instances from the table.

Now in tabular format of 36450 rows and 2 columns, the speeches were more thoroughly cleaned by only including those members and speakers for whom the political belonging was known. On top of that, there was a need to exclude speeches of the presidents of the parliament. The table after data cleaning consisted of 18394 rows and 2 columns (speech and target label).

Before continuing further with data preparation for modelling, there is a need to perform test and train data split using function `train_test_split()`. This is important so that after building (training) the model there is data to test the performance of the model. This evaluation part is crucial as we strive for models which have predictive power. We preserve standard 0.75/0.25 split ratio not altering the `test_size=0.25` parameter. In order to ensure training data contains both speech with target values of 0 and 1, we include `stratify=y` parameter. In order to ensure reproducibility, we include `randomstate=42` parameter.

3.2 The feature extraction

As speech is not continuous nor categorical, but freeform string data, there is a need to carry out the feature extraction. The feature extraction itself consists of tokenization, vocabulary building and encoding (Ali, 2022).

Tokenization refers to the process of converting a sequence of text into smaller parts, known as tokens (Awan, 2023). While one can tokenize by characters, most sensible is to split text into words by whitespace and punctuation. After that, all tokens are aggregated into one big vocabulary - this process is called vocabulary building. This vocabulary actually is eventual feature names. When it comes down to encoding, several approaches can be used such as bag-of-words and term frequency-inverse document frequency method. These are methods for extracting features from text, i.e. converting texts into numerical features. (Ali, 2022).

Bag-of-words is an approach where for each document, texts are converted into occurrences for every word in the previously built vocabulary (Müller & Guido, 2016, p. 328). Bag-of-words however do not count infrequent, yet possible important words, as important. This problem is solved using term frequency-inverse document frequency (tf-idf) method. Term frequency-inverse document frequency (tf-idf) method is a method, adding more weight to words appearing frequently in some documents, infrequently appearing words across the documents. The formula for that is

$$\mathrm{tfidf}(w, d) = \mathrm{tf} \cdot \log \frac{N+1}{N_w+1} + 1,$$

where term frequency tf accounts for word frequency of word w in the document d, N is the number of documents, N_w is the number of documents in the training set that the word w appears in (Müller & Guido, 2016, p. 336).

As previously mentioned, tokenization and vocabulary building are same for all three method using function `fit()` for each of the objects. Bag-of-words can implemented through `CountVectorizer()`, tf-idf can be found from the object `TfidfVectorizer()` by having boolean `use-idf=True` parameter. Calling function `transform()` converts texts into numerical features. As already mentioned, tf-idf is more sophisticated and truthful encoding method and thus this paper opts for tf-idf and `TfidfVectorizer()` for encoding.

4 Data modeling

Data modeling step consists of applying our prepared data to various models and testing them. Models taken under consideration include Logistic regression, Naive-Bayes, Support Vector Machine. While the model itself tries to minimize loss function, we care about the model's ability to generalize as evaluation metric. In particular, for binary classification best and universal such metric is accuracy, the number of correctly predicted instances over the total number of predictions on test set. This means that the model was able to generalize on data which was not part of training.

The process of modeling and testing was carried out through pipelining and hyper-parameter search. The following code block reveals how exactly was the search carried out. Pipelining (line 2) allows us to use both encoder and classifier together passing in parameters we are interested in tweaking (line 1). For hyper-parameter search, there are two general ways: grid search and randomized search. As grid search is the most widely used method for parameter optimization, this paper chose because with grid search one can specify the values to be tested and finds the best (Scikit-learn, n.d.-c). Grid search is implemented through `GridSearchCV()` and line 3 reflects brings together the search space and the pipeline, `n_jobs=-1` ensures that all processors are utilized, i.e. program is run in parallel, yielding faster outcome. Line 4 is there to eventually train the model with our data.

```
params = {'vect__max_df':[0.1,0.2,0.3,0.4,0.5,0.6,0.7],...}
pipeline = Pipeline([('vect', TfidfVectorizer()), ('model', Classifier())])
grid = GridSearchCV(pipeline, cv=3, n_jobs=-1, param_grid=params)
grid.fit(Xs_train, y_train)
```

4.1 Dummy Classifier

Dummy classification is a most straight-forward model of them all. Dummy classifier predicts the class ignoring the inputs. Implementation can be achieved through `DummyClassifier()`. This classifier is great and simple baseline to compare against other more complex classifiers. Regarding the hyper-parameter `strategy` the default hyperparameter `strategy="most_frequent"` predicts the class which is most frequently occurring in fed-in train dataset. With `strategy="stratified"` the class probability corresponds to the class distribution in training set and `strategy="uniform"` generates predictions uniformly at random. (Scikit-learn, n.d.-b)

4.2 KNeighbors Classifier

The k nearest neighbors is an algorithm, which uses proximity of k nearest neighbors to predict the grouping of instance (IBM, n.d.). The k nearest neighbors algorithm can implemented through `KNeighborsClassifier()`. Hyperparameter such as `n_neighbors` and `weights` can help to tweak the model in order obtain best decision boundaries.

4.3 Logistic Regression

Logistic regression tries to estimate probability of occurring event for predicting particular class. That is for binary classification, the probability of particular glass greater than 0.5 yields the prediction of this incident class. In particular, the formula for a class is

$$p(X) = \frac{1}{1 + e^{f(X)}},$$

where f is linear model. The opposite probability $1 - p(X)$ represents the probability that the instance is not of this class, for binary classification, it is the other class. (GeeksforGeeks, 2024a). In scikit-learn, logistic regression can be deployed with `LogisticRegression()`. Hyperparameter `C` represents the inverse of regularization strength, meaning smaller `C` yields higher regulation and vice versa.

4.4 Naive Bayes

Naive Bayes algorithm group is set of methods, which also, like Logistic Regression, estimate the probability of occurring event and based on that predict the class. The formula for predictive class is

$$P(y|x_1, \ldots, x_n) = \frac{P(y) \cdot P(x_1, \ldots, x_n|y)}{P(x_1, \ldots, x_n)} = \frac{P(y) \cdot \prod_{i=1}^{n} P(x_i|y)}{P(x_1, \ldots, x_n)}.$$

The first equality is the result of Bayes' theorem and the second equality comes under assumption that features are pairwisely conditionally independent, i.e. $P(x_i|y, x_j) = P(x_i|y) \forall i, j$. Now, since the vocabulary, the set of features x_i's, are fixed, $P(x_1, \ldots, x_n)$ is fixed and thus has no influence on the class prediction, $P(y)$ can be estimated as the relative frequency of class in the training set and $P(x_i|y)$ calculation differs based on which concrete Naive Bayes model is used. This paper uses Multinomial Naive Bayes model, implemented on sklearn as `MultinomialNB()`. (Scikit-learn, n.d.-a)

4.5 Support Vector Machine

Support Vector Machines at its heart try to find hyperplane decision boundary. It is implemented through `SVC()` in `sklearn.svm`. The hyperparameter `kernel` describes how exactly the boundary should be found. Specifying `kernel="linear"` we make sure classes are separated by lines. (GeeksforGeeks, 2024b)

4.6 Decision Tree

Decision tree uses the algorithm that searches over all possible tests and finds the one that is most informative about the target variable during each recursive step. This recursion is carried out until all the leafs (regions of instances) have been purified, i.e. only consist of single class. In order to prevent overfitting, for trees there is pre-pruning, especially setting maximum depth (hyperparameter `max_depth`) of the tree. (Müller & Guido, 2016, pp. 74–75)

5 Results

Moving on to the results, we firstly present results for each model respectively and then draw conclusions from the entire result pool.

5.1 Hyper parameter tuning

5.1.1 Dummy Classifier

The hyper parameter tuning results for Dummy Classifier are shown in table 2. As it makes predictions that ignore the input features (Scikit-learn, n.d.-b), only model parameters are those that tweak the performance, hence vectorizer's hypermeter tuning was omitted. As shown in table 2, GridSearchCV found most frequent prediction most accurate from the other two, which also makes sense since we in our training data we had more target values having value 0 than 1, like shown in figure 1. The uniform strategy scores 0.5 because each correct prediction is like a coin flip and the number of coin flips does not alter number of correctly prediction over total predictions. The stratified classifier reflects test score of $x^2 + y^2$, where x, y are the class proportions in training set as we also stratified our train-test split.

Model	Model parameters	mean_test_score
DummyClassifier	strategy="most_frequent"	0.638710
DummyClassifier	strategy="stratified"	0.543748
DummyClassifier	strategy="uniform"	0.504603

Table 2: Dummy Classifier

5.1.2 KNeighbors Classifier

The hyper parameter tuning results for KNeighbors Classifier are shown in table 3. In this case we only iterated over the number of neighbors `n_neighbors` and `max_df`. In particular, for vectorizer we iterated over `max_df` from 0.1 to 0.7, for the model we considered `n_neighbors` values from the list `[10, 100, 300, 1000, 2000, 3000]`. For model, we also allowed `weights` to be equal to `uniform` (all points weighted with equal significance) and `distance` (weighting points by distance, giving closer points more weight). We kept `ngram_range` only unigrammed as otherwise the number of features would have increased, resulting more dimension and more need for computational power and speed. As seen from table 3, best result is achieved with parameters `n_neighbors=100`. This means that taking into account features far away distort the prediction, just as only considering relatively close instances in the vector space.

Model	Vect (max_df)	Model (weights)	Model (n_neighbors)	mean_test_score
KNeighborsClassifier	0.2	distance	100	0.802101
KNeighborsClassifier	0.2	uniform	100	0.800651
KNeighborsClassifier	0.4	distance	100	0.800362

Table 3: Top 3 of KNeighbors Classifier

5.1.3 Logistic Regression

The hyper parameter tuning results for Logistic Regression are shown in table 4. Since we iterated over 70 combinations of hyperparameters only top 3 results are shown here. In particular, for vectorizer we iterated over `max_df` from 0.1 to 0.7 and `ngram_range` of (1,1) and (1,2), and for model itself we tweaked regularization coefficient `C` from 0.1 to 1000. As seen from table 4, `C=100` might look like overregulation, but higher `C`'s yield worse test score, going past the optimum.

Model	Vect (max_df)	Vect (ngram_range)	Model (C)	mean_test_score
LogisticRegression	0.1	(1,2)	100	0.885248
LogisticRegression	0.1	(1,2)	1000	0.884596
LogisticRegression	0.3	(1,2)	1000	0.884234

Table 4: Top 3 of Logistic Regression

5.1.4 Naive Bayes

The hyper parameter tuning results for multinomial Naive Bayes classifier are shown in table 5. The ranges taken under closer consideration are for vectorizer, `ngram_range` from (1.1), (1.2), `max_df` ranging from 0.1 to 0.7. and for model `alpha` from 0.0001 to 100. As seen from table 5, best regulation is achieved with `alpha=0.01`.

Model	Vect (max_df)	Vect (ngram_range)	Model (alpha)	mean_test_score
MultinomialNB	0.7	(1,2)	0.01	0.865314
MultinomialNB	0.6	(1,2)	0.01	0.865241
MultinomialNB	0.5	(1,2)	0.01	0.865241

Table 5: Top 3 of MultinomialNB

5.1.5 Support Vector Machine

Table 6 displays the top 3 hyper parameter combinations for `SVC` from GridSearchCV. The combinations were generated from cross multiplications of `C` ranging 0.01 to 1000, `max_df` ranging from 0.1 to 0.7 and values of (1,1) and (1,2) for `ngram_range`. As seen from table 6, best regulation is achieved with `C=1`.

Model	Vect (max_df)	Vect (ngram_range)	Model (C)	mean_test_score
SVC	0.1	(1,2)	1	0.888075
SVC	0.1	(1,2)	10	0.887423
SVC	0.2	(1,2)	10	0.885973

Table 6: Top 3 of Support Vector Machine

5.1.6 Decision Tree

Top cross validation results for Decision Tree are shown in table 7. The ranges gone through were 0.1 to 0.7 for `max_df`, values (1,1) and (1,2) for `ngram_range`, values from `[None, 'log2', 'sqrt']` for `max_features` and `max_depth` range from 10 to 10000 and None.

Model	Vect (max_df)	Vect (ngram_range)	Model (max_depth)	Model (max_features)	mean_test_score
DecisionTreeClassifier	0.7	(1,1)	10	None	0.742008
DecisionTreeClassifier	0.1	(1,2)	100	None	0.741066
DecisionTreeClassifier	0.7	(1,2)	10	None	0.739325

Table 7: Top 3 of Decision Tree Classifier

5.2 Model comparison

The best hyper parameters were taken under further consideration. The result for used Vectorizer and Model for received accuracies are shown in table 8. All models score both better on train and test data than our baseline, DummyClassifier with strategy of most frequent prediction. Our best model on our simple classification task is Logistic Regression with `C=100` with test score of 88.8%. This is followed by SVC with linear kernel and `C=1` scoring 87.0% and MultinomialNB with `alpha=0.01` scoring 86.4%. The worst non-baseline classifier is Decision Tree with `max_depth` equal to 10 and `max_features` equal to None scoring 73.8%. It is expected to have Decision Tree in the bottom in our case, because trees suffer badly in such high dimensional feature spaces as it has hard to classify documents based on single word splits, rather text carry value as a grouping. KNeighborsClassifier got test accuracy of 80.1%, placing itself in the bottom two. Our results match with the fact that multinomial Naive Bayes and Support Vector Machine are great simple models for text classification (Wang & Manning, 2012). The results also align with Logistic Regression outperforming both Decision Tree based models and KNeighbors Classifier for text classification (Shah et al., 2020).

Vect	Model	Train score	Test score
TfidfVectorizer()	DummyClassifier(strategy="most_frequent")	0.638709	0.638617
TfidfVectorizer(max_df=0.2)	KNeighborsClassifier(weights="distance",n_neighbors=100)	0.996955	0.801478
TfidfVectorizer(max_df=0.1,ngram_range=(1,2))	LogisticRegression(C=100)	0.997607	0.888019
TfidfVectorizer(max_df=0.7,ngram_range=(1,2))	MultinomialNB(alpha=0.01)	0.995433	0.864100
TfidfVectorizer(max_df=0.1,ngram_range=(1,2))	SVC(kernel="linear",C=1)	0.994563	0.870189
TfidfVectorizer(max_df=0.7,ngram_range=(1,1))	DecisionTreeClassifier(max_depth=10,max_features=None)	0.781805	0.737551

Table 8: Model comparison

As Logistic Regression scored the best, we take a closer look on models performance and its features. First and foremost, figure 4 highlights the relationship between model's accuracy and C value. In particular it is clear that the sweet spot (also indicated on the figure) is at 100, corresponding with our findings. Furthermore, going past `C=100` tends to overfit (as test score starts to slowly decrease) and `C<100` clearly shows underfitting near `C=1`.

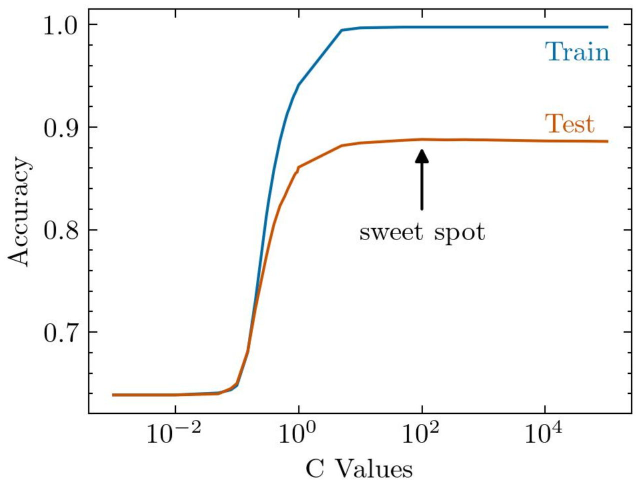

Figure 4: Logistic Regression accuracies based on C value

Figure 5 shows most significant features of our best model, Logistic Regression. In particular, it reflects top 20 lowest and highest coefficient values. The feature with lowest coefficient value is with name label "chairman dear" and the feature with highest coefficient value is with name label "for question". In general, we see that among lowest most significant features are those, which are addressed towards chairman of the parliament, i.e. held at the beginning of one's speech. When it comes to the highest most significant features, those are said in response to the members of the parliament, who mostly ask question from the coalition's decision. Still, we can see that among lowest coefficient feature list is name label "car tax", which saw fierce criticism and resistance from the opposition since government introduced and went through with The Motor Vehicle Tax Bill (Riigikogu, 2024). Also, lowest coefficient feature list includes label "kaja", which reflects that opposition mentions and addresses government leader, the prime minister of Estonia, Kaja Kallas a lot. Looking at highest coefficient feature list, the most interesting label is "russia", which also makes sense as coalition and government have used Russia's invasion of Ukraine (and increased threats of Russia towards Estonia and NATO) as important factor why certain decisions have been made (Simon, 2024).

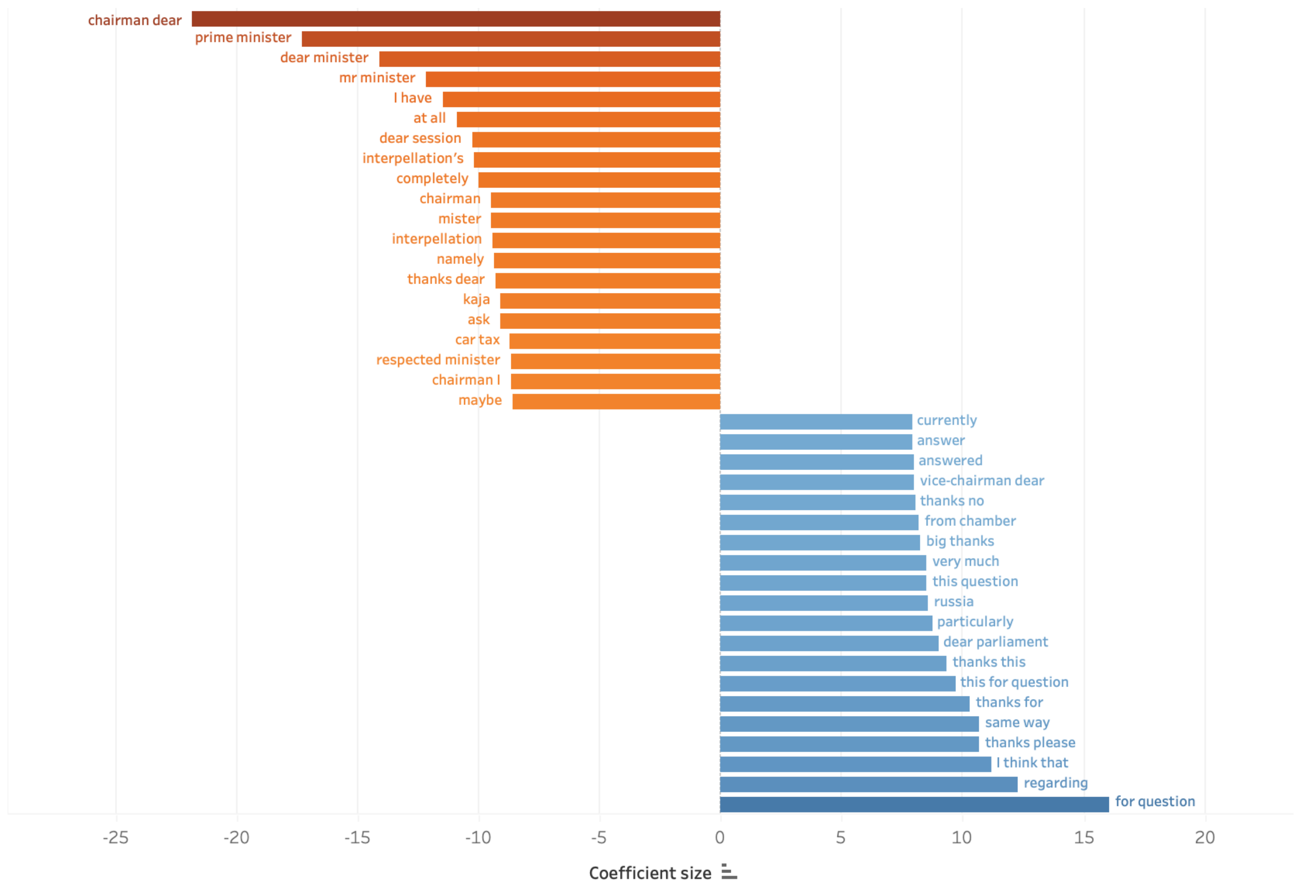

Figure 5: Most significant predictive features

6 Discussion

6.1 Visualization choices

Alberto Cairo has defined five principles of great visualizations. Particularly, great visualization is truthful, functional, beautiful, insightful and enlightening (Cairo, 2016). Also, ideas of graphical excellence had its input on visualization, namely creating figures giving greatest number of ideas forward in the shortest time with the least ink in the smallest space (Tufte, 2001, p. 51). These notions were kept in mind while author was in the process of creation of visualizations. The figures were created by Python or Tableau, where the former was used for simple visualizations and the latter allowing aggregations (in case of figure 1) and easier colour coding and label positioning (for figure 5).

Truthful condition was fulfilled by using entire, relevant data and decoding it into functional charts. As Cairo puts it, "the five qualities of great visualizations aren't independent from each other but are tightly interrelated" (Cairo, 2016). The property of functionality is mostly achieved through right choice of chart type. Figures 1, 2, 3 and 5 were decided to most functionally display meaning as bar charts, while figure 4 is line chart, as this type of chart is great way to compare change of one variable to another. To make charts beautiful, for figures 2, 3, 4 created in Python, SciencePlots theme was used to make charts more pleasing, while keeping only relevant information, preserving simplicity (Garrett, 2021). Figures 1, 5 were created in Tableau and figure 5 reflects concepts of keeping data ink minimal (by moving bar labels next to the bars themselves) and colouring bars based on coefficient size (with negatives having reddish undertone and positive receiving blue). Space consideration was made with 1 and 5 keeping them horizontal (and cutting unnecessary labels as already mentioned) and colour choice also carefully considered opting for default blue vs red/orange colour scheme as it is one of the most common colorblind-friendly colour palettes (Shaffer, 2016). Insightfulness is covered by e.g. highlighting sweet spot on figure 4. While other figures might not exactly contain insight by itself, they add much value to the text surrounded by figures and to the paper in general. Finally, the enlightening factor is more deeply covered in section 7 about the relevance of the paper, giving some background why those figures tell the story of research process and figures 3 and 5 together give a picture that most frequently occurring words are not the most significant ones for sentiment classification.

6.2 Improvements

Improvements can be cut extracted from data collection, preprocessing, model selection steps. Also, as binary classification might not be best way to identify sentiment and/or political opinion (especially as both coalition and opposition talk about same things from different sides), more classes (e.g. party classification) might reveal more insight on sentiment and improve the models.

The data collection process can be improved by including more texts and more carefully considering which texts to take under consideration. For example, excluding texts which consists of one sentence, which were included in this paper. Also, if we care about generalization, it might be wise to include political texts and speeches that are not held in the parliamentary setting as data used here is only a subset of available political texts.

Data preprocessing step can be refined by stopwords, stemming or lemmatization. On top of that, excluding formalities of the speeches such as addressing chairman at the beginning of speech in the parliament might improve accuracy and reveal more sentiment in texts.

Improvements for model selection step include use of more precise models and even wider and deeper search for best hyperparameters.

7 Relevance

Debates and discussions are part of democratic decision making. Debates also give a public an opportunity to follow the process and make up its own mind on the discussed matter. This in turn incentivizes people to participate in elections and cast a vote on desired people and ideas. Thus, understanding what is being discussed in the parliament is crucial part of democracy and better future in general. Turning an eye on verbatim records held in parliament allows to analyze and summarize discussed. When identifying talking points of political parties or groups, public can get real reflection on for which topics which groups stand for and how relevant is issue for them.

Considering the prediction, such models might also help politicians change political group based on their speeches, i.e. identify best matching political affiliation. These models are not only valuable for politicians but voters too, because models can help electorate to find most suitable groupings for vote casting, such as popular voting election compasses.

Another application of such models consisting of those records can be helpful receiving insights on idea affiliations of those who are not in particular political group or those who are supposed to be unbiased from politic debate such as legal entities according to the rule of law. Bias detection is important as we are living in the age of information. We generate lots of information and this is quickly accessible. Thus, bias detection and political affiliation detectors also play a significant part in media and journalism. That is why there are companies such as The Bipartisan Press, who fight for transparent journalism with their political bias AI. And just as when there is bias in the media and lack of disclosure of bias, media distrust is going up and polarization follows (The Bipartisan Press, n.d.).

Just as every tool can be used in a good way, it can be also exploited for bad causes. One such example can be micro targeting, based on talked, written and read content of a person (Naldal, 2018). All of these content forms can be used to extract and detect political affiliations and activate electorate. One such example would be infamous case of Cambridge Analytica, using user data from Facebook to target voters (Dizikes, 2023).

8 Conclusion

The paper considered different machine learning models and its use in binary political affiliation classification context. The trained models with the verbatim records-data of Estonian parliament gave also an opportunity to get insights on talking points, rhetoric and debate features of respective groupings.

This paper found the best performing model to be Logisitic Regression with predictive accuracy of 88.8%, followed closely by Multinomial Naive-Bayes and Support Vector Classifier with linear kernel. While all the models did better than the baseline, predicting most frequent, the worst non-baseline model was Decision Tree classifier with test accuracy of 73.8%. Also, while most significant features of our best model, Logistic Regression, were part of speaker's individual vocabulary and formalities of parliamentary speeches, the paper also found that feature names containing actual sentiment such as "russia" or "car tax" own high expressive value for this binary classification task.

References

Ali, M. (2022). Understanding Text Classification in Python. Retrieved May 6, 2024, from https://www.datacamp.com/tutorial/text-classification-python/

Awan, A. A. (2023). What is tokenization? Retrieved May 6, 2024, from https://www.datacamp.com/blog/what-is-tokenization

Cairo, A. (2016). *The truthful art: Data, charts, and maps for communication* (1st). New Riders Publishing.

Dizikes, P. (2023). Study: Microtargeting works, just not the way people think. Retrieved May 13, 2024, from https://news.mit.edu/2023/study-microtargeting-politics-tailored-ads-0621

Garrett, J. D. (2021). garrettj403/SciencePlots. https://doi.org/10.5281/zenodo.4106649

GeeksforGeeks. (2024a). Logistic regression in machine learning. Retrieved May 7, 2024, from https://www.geeksforgeeks.org/understanding-logistic-regression/

GeeksforGeeks. (2024b). Naive Bayes vs. SVM for Text Classification. Retrieved May 7, 2024, from https://www.geeksforgeeks.org/naive-bayes-vs-svm-for-text-classification/

Hosch, W. L. (n.d.). Zipf's law. Retrieved May 6, 2024, from https://www.britannica.com/topic/Zipfs-law

IBM. (n.d.). What is the k-nearest neighbors (knn) algorithm? Retrieved May 7, 2024, from https://www.ibm.com/topics/knn

Müller, A. C., & Guido, S. (2016). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, Inc.

Naldal, S. L. (2018). Investigating political speeches using machine learning. https://research-api.cbs.dk/ws/portalfiles/portal/59756232/542666_THESIS.pdf

Riigi Teataja. (2019). Riigikogu Rules of Procedure and Internal Rules Act. Retrieved May 6, 2024, from https://www.riigiteataja.ee/en/eli/504062020005/consolide

Riigikogu. (n.d.). *Verbatim records*. Retrieved March 19, 2024, from https://stenogrammid.riigikogu.ee/et?rangeFrom=06.03.2023&rangeTo=18.03.2024

Riigikogu. (2023). XV Riigikogu. Retrieved May 6, 2024, from https://www.riigikogu.ee/en/parliament-of-estonia/composition/

Riigikogu. (2024). The Motor Vehicle Tax Bill passed the first reading in the Riigikogu. Retrieved May 19, 2024, from https://www.riigikogu.ee/en/sitting-reviews/the-motor-vehicle-tax-bill-passed-the-first-reading-in-the-riigikogu/

Scikit-learn. (n.d.-a). Naive bayes. Retrieved May 7, 2024, from https://scikit-learn.org/stable/modules/naive_bayes.html

Scikit-learn. (n.d.-b). sklearn.dummy.DummyClassifier. Retrieved May 7, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier

Scikit-learn. (n.d.-c). Tuning the hyper-parameters of an estimator. Retrieved May 7, 2024, from https://scikit-learn.org/stable/modules/grid_search.html

Shaffer, J. (2016). 5 Tips on Designing Colorblind-Friendly Visualizations. Retrieved May 21, 2024, from https://www.tableau.com/blog/examining-data-viz-rules-dont-use-red-green-together

Shah, K., Patel, H., Sanghvi, D., & Shah, M. (2020). A comparative analysis of logistic regression, random forest and knn models for the text classification. *Augmented Human Research*, *5*. https://api.semanticscholar.org/CorpusID:219756231

Simon, S. (2024). Estonian Prime Minister on how Russia's invasion of Ukraine has impacted her country. Retrieved May 19, 2024, from https://www.npr.org/2024/03/16/1238981479/estonian-prime-minister-on-how-russias-invasion-of-ukraine-has-impacted-her-coun

The Bipartisan Press. (n.d.). About Us. Retrieved May 13, 2024, from https://www.thebipartisanpress.com/about-us/

Tufte, E. R. (2001). *The Visual Display of Quantitative Information* (Second). Graphics Press. https://www.edwardtufte.com/tufte/books_vdqi

Wang, S., & Manning, C. (2012, July). Baselines and bigrams: Simple, good sentiment and topic classification. In H. Li, C.-Y. Lin, M. Osborne, G. G. Lee, & J. C. Park (Eds.), *Proceedings of the 50th annual meeting of the association for computational linguistics (volume 2: Short papers)* (pp. 90–94). Association for Computational Linguistics. https://aclanthology.org/P12-2018