Political Orientation Prediction from Newspaper Headlines

Link to Github Repository

K. Tilman
s3229807
k.tilman@student.rug.nl

S. de Vries
s3186520
s.de.vries.44@student.rug.nl

1 Introduction

Imagine being able to predict the political orientation of a newspaper, based on the headline alone. This is a very interesting task since we know that news influences the public opinion.

In this paper data was analyzed from over 35,000 articles that have been published in the time period around the first 25 Conference of the Parties meetings. Using the headline of these articles various classification models were defined and trained to predict the political orientation of said newspapers. The political orientations discussed in the research are Left-Center and Right-Center. The orientations are based on analysis by AllSides [AllSides] and Media Bias Fact Check [MediaBiasFactCheck].

The results of this project can for example be used in discovering trends in political orientation based on headlines of news articles. In theory, the problem can be extended to deal with sentences in a more generic sense, such that it can also be used for the classification of tweets.

In this paper, we will discuss several models for classifying headlines and how we handled preprocessing and class imbalance. In section 2 we will first introduce our models, after which we will evaluate the results on the Conferences of the parties meetings in section 3. Finally, we will conclude our research by selecting the best model.

2 Methodolody

Using the data provided, four classification tasks were designed. These consisted of a classic baseline model, Naive Bayes (NB) [Webb2010]. This model was optimized into a Support Vector Machine (SVM) [Cortes and Vapnik1995]. As a next step an optimized Long short-term memory (LSTM) [Hochreiter and Schmidhuber1997] model with pretrained static embeddings using the GloVe Model [Pennington et al.2014], specifically with the Wikipedia 2014 and Gigaword 5 dataset, was composed. Lastly Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al.2019], a fine-tuned pretrained language model, was created. In addition, to be able to compare our solutions with existing extensive research at hand, we used FastText supervised learning [Joulin et al.2016].

The decision was made to look at the headline of each article as it requires little computation, it does not require extensive pre-processing in comparison to for example a body where text is separated in paragraphs. Moreover, this makes it more suitable for real time applications such as one that retrieves published article headlines from online resources and classifies these with our classifier.

2.1 Data preparation

Real-world data is generally noisy, incomplete and inconsistent. This implies that raw data tends to be corrupt, has missing values or attributes, outliers or conflicting values. Often the distribution of classes is also unequal, which is confirmed in figure 1.

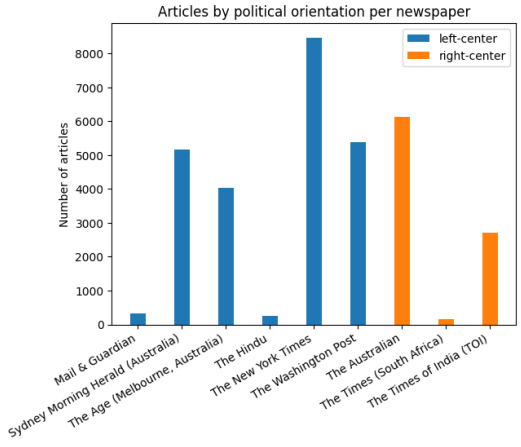


Figure 1: Political orientation per newspaper

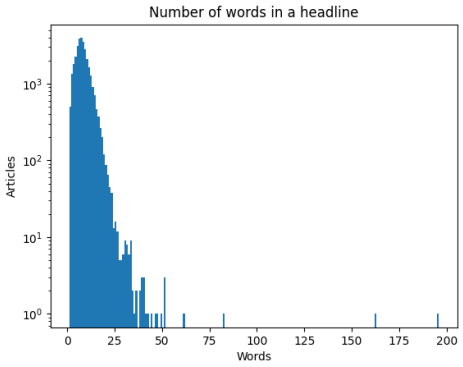


Figure 2: Histogram number of Words

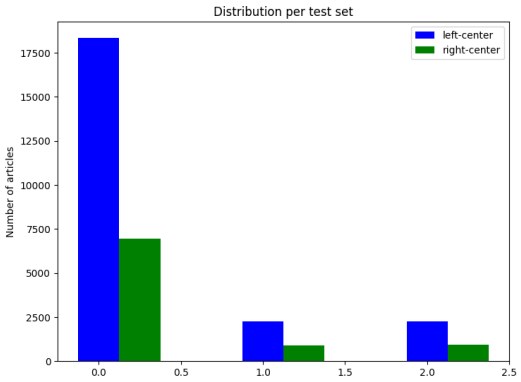


Figure 3: Distribution of classes in train/dev/test set

It can be observed that there are only three newspapers which are right-center oriented, while six newspapers can be described as left-center.

Moreover, we notice from the histogram of the length of an article (depicted in figure 2) that there are some outliers with extraordinary length. Despite these outliers not being representative with respect to length for the rest of the data they still contain information and are therefore kept in the data set.

Imbalanced data is is a widely researched area and several proposed solutions exist. One of the solutions we will use is the imbalanced learn [Lemaître et al.2017] toolbox which deals with the imbalanced data for us. Our strategy is to (randomly) under sample the majority class so that we are left with an even amount of samples for each class. An alternative and more advanced approach could be Synthetic Minority Oversampling Technique (SMOTE) [Chawla et al.2002] which we leave as future research.

After looking at the occurrences of headlines in the data set, in figure 4a, our first step is to remove non alpha-numeric characters and meaningless headlines like *'No Headline in Original'*. The cleaned data set, shown in figure 4b, is a lot more evenly distributed despite still containing several duplicates.

Next, we split all articles into a fixed split of train, dev and test. For this purpose we use the *train_test_split* helper function of the scikit-learn library [Pedregosa et al.2011] where we fix the random state to a seed in order to give consistent (random) results. We first split into train and test by taking randomly 80% of the articles to train on. The remaining 20% is split equally into a dev and test set. For each model the same split of train and dev is used in order to train and evaluate the model. The distribution of the sets can be seen in figure 3, which contains the dataset without undersampling. After finishing the tuning process, each model is tested on the test set to achieve a final comparison.

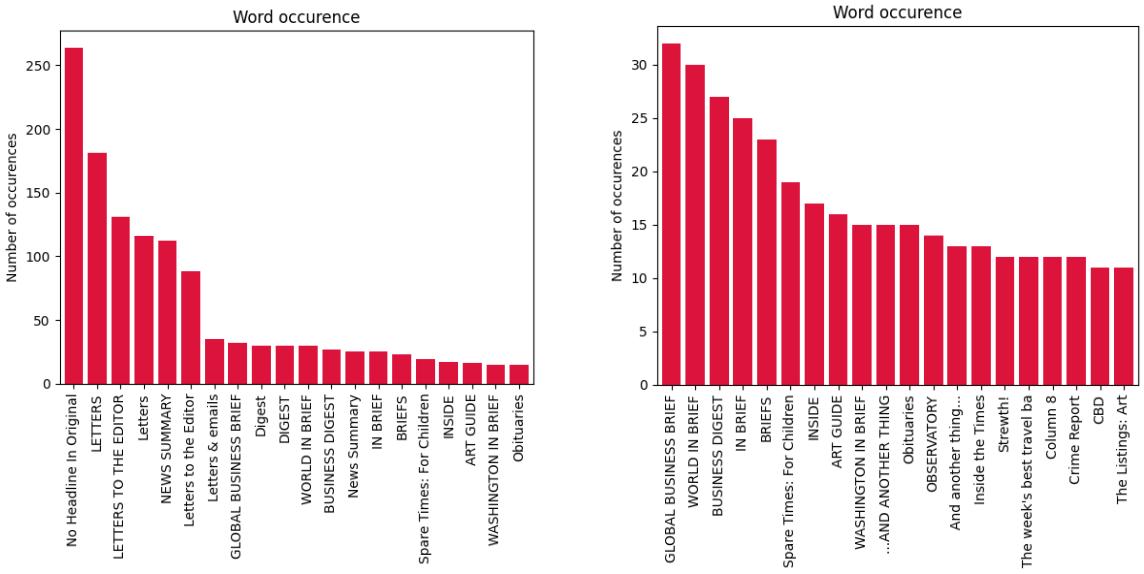
2.2 Naive Bayes

As a first step, a classic baseline model was designed using bag-of-words. The model chosen here was Naive Bayes which is a classification method based on Bayes rule. The naive variant of the classifier assumes that the value of a particular feature is independent of the value of any other given feature. Due to its naive and very simple design, Naive Bayes is often a good fit for a baseline.

For the implementation *MultinomialNB* was used as a classifier. As a vectorizer, experiments were performed with both a Term Frequency Inverse Document Frequency (TF-IDF) [Havrlant and Kreinovich2017] vectorizer *TfidfVectorizer* and a count vectorizer [Kulkarni and Shivananda2021] *CountVectorizer*. In order to find the optimal value for the smoothing parameter, grid search was performed. Grid search performs an exhaustive search on the specified parameter values and extends really well to multiple parameters. Additionally, a benefit of using grid search is that it comes with cross validation out of the box.

2.3 Support Vector Machine

In order to optimize our baseline, a support vector machine is used. Support vector machines are supervised learning models that analyze data for classification and regression analysis. For this task, the module *svm* was used. More specifically,



(a) Initial distribution

(b) Cleaned distribution

Figure 4: 20 most occurring headlines

the module *LinearSVC* was used, which implements linear support vector classification. It is also possible to use the module *SVC*, however the linear classification was chosen since it is more suitable for larger files.

In addition to the linear support vector classification, the pipeline consists of two vectorizers. Namely the count vectorizer and term frequency vectorizer. The two vectorizers are combined using the module *FeatureUnion* which concatenates the results of both objects.

Lastly, grid search is performed on a number of parameters. The parameters which are used to perform experiments are the maximum document frequency, minimum document frequency, the lower and upper boundary of the range of n-values for different n-grams to be extracted and lastly the C value of the linear classifier.

2.4 LSTM

The next model which was designed is a long short-term memory (LSTM) layer. Which is an artifical recurrent neural network architecture.

Before the actual algorithm itself the text is transformed to a vector. For this we use pretrained Glove embeddings consisting of 6 Billion tokens (constructed from the Wikipedia 2014 and Gigaword 5 dataset) which converts our headlines to 300 dimensional vectors. In order to get a LSTM model that performs reasonably well in terms of accuracy, experiments were performed with different sizes of the LSTM layer. The parameters were tuned ”manually” as grid search would be quite intensive to use in this case.

2.5 BERT

As a next step we try to push performance by using a pretrained language model (LM). More specifically, Bidirectional Encoder Representations from Transformers (BERT) is used. BERT is designed to pre-train deep directional representations from unlabeled text by jointly conditioning on both left and right context in all layers.

In order to retrieve the pretrained model, the auto class *TFAutoModelForSequenceClassification* was used. We only have to specify the model identifier, ”bert-base-uncased”, in order to use the LM. This model was chosen since it works well for English datasets and our data also contains uppercase characters, which makes it a suitable use case.

In order to prepare the data, all the labels are binarized and preprocessing is done using the autotokenizer.

Regarding the fine tuning of our model, as a loss function we are using *SparseCategoricalAccuracy* since it performed best. As an optimizer *Adam* was used. Experiments were performed for different batch sizes. In addition to this, a learning rate scheduler was used, more specifically *PolynomialDecay*.

	Acc.	Macro avg. f1-score	Left-center Precision	Recall	F1-score	Right-center Precision	Recall	F1-score
NB (Imbalanced)	73.9	58.0	75.0	95.0	83.8	63.4	21.6	32.2
NB (Balanced)	63.6	62.8	89.9	55.1	68.3	43.2	84.6	57.2
SVM (Imbalanced)	80.9	73.7	84.3	90.9	87.5	68.2	53.5	59.9
SVM (Balanced)	75.2	72.5	91.6	72.9	81.2	52.3	81.6	63.7
LSTM (Imbalanced)	77.5	71.7	83.0	85.9	84.5	61.8	56.5	59.0
LSTM (Balanced)	75.9	70.9	83.6	82.3	83.0	57.8	60.1	58.9
BERT (Imbalanced)	86.6	82.5	92.4	89.7	91.0	70.9	77.2	73.9
BERT (Balanced)	86.7	83.7	88.4	93.1	90.7	81.9	72.1	76.7
FastText (Imbalanced)	82.3	77.8	86.6	88.7	87.6	70.1	66.0	68.0
FastText (Balanced)	76.8	74.8	91.9	74.0	82.0	56.5	83.8	67.5

Table 1: Train set balancing performance on dev set

3 Results

The two most important metrics discussed in this section are the accuracy and macro average f1-score. In practice, the f1-score might be even of more importance as it takes the false negatives and false positives into account.

In addition, for each class separately (Left-Center and Right-Center), we evaluate the precision, recall and f1-score. For each individual model the results will be discussed, more specifically metrics for both the balanced and imbalanced datasets will be compared in table 1. Where the best performing model on the dev set is highlighted. Finally, the best version of each model is used on the test set in order to determine a final performance.

3.1 Naive Bayes

Firstly we will discuss the results of the Naive Bayes classifier. The final model, after performing grid search to find the optimal parameter, was tested on both the imbalanced balanced data set. The imbalanced data set showed an accuracy of 73.9%, whereas the balanced data set showed an accuracy of 63.6%. This is a decrease in accuracy, however the f1-score of running the model on the balanced dataset showed a notable increase from 32.2% to 57.2% for the "Right-center" class. This increase makes sense since the under sampling makes sure that the "Right-center" class is evenly represented.

3.2 Support Vector Machine

Compared to the NB classifier, the SVM model has a higher accuracy, 80.9% compared to a 73.9% accuracy for NB. Even though there is also a decrease in accuracy for the balanced dataset, it is still higher than the accuracy for the NB model. This improvement in accuracy makes sense since grid search is applied to more parameters. Additionally, the support vector machine contains a union of vectorizers which potentially contains more information leading to improvement performance. Again, a notable improvement can be observed for the recall of the "Right-center" class. The f1-score for the "Left-center" class slightly decreases and for the "Right-center" class slightly increases, however this change is not as great as we observed for the Naive Bayes model.

3.3 LSTM

The LSTM model using the imbalanced dataset does not show an improvement compared to the SVM model. The accuracy is 77.5% compared to 80.9% for the SVM model. However there is a small improvement in accuracy when using the balanced dataset, a small increase from 75.2% to 75.9% is observed. Then for the other statistics, there are no outstanding differences between the imbalanced and balanced dataset.

From the experiments with the number of LSTM units we observed that our initial LSTM of 3 units does not learn anything as the accuracy does not increase in between epochs. Increasing the number of units to half of the de-

fault batch size and equal to the default batch size we see that the accuracy and F1 score are higher for the larger number of LSTM units (respectively, $75,9\% \rightarrow 77,5\%$ and $67,4\% \rightarrow 70,4\%$)

We trained the model for 50 epochs and stop early if the loss on the validation data did not improve for 3 iterations. In practice this means that the algorithm stops training after roughly 10 epochs.

3.4 BERT

BERT was trained for different batch sizes as well as using a learning rate scheduler to optimize performance. The optimal batch size was found to be 16, in a range from 2 to 32. The model was trained on 50 epochs and was set to be stopped early when the validation accuracy did not improve in 3 consecutive iterations. In practice this meant that the algorithm stopped training between 4 and 7 epochs.

As can be observed from the table, BERT performs quite well compared to the other models. With an accuracy of 86.6% for the imbalanced dataset and 86.7% for the balanced dataset. Within the results, there are no notable differences between the imbalanced and balanced dataset.

3.5 FastText

As a benchmark for our programs we used the out-of-the-box implementation of Fasttext to see how well the algorithm performs against a well-known (text) classifier. First of all, there is a decrease for both the accuracy and the f1-score when the data is pre-processed. One possible explanation for this is that FastText comes with some (better) built-in methods that deal with imbalanced data. This is very likely as mentioned before imbalanced data is a common use case.

One possible drawback and strength at the same time of FastText is the way it takes input, namely from a file. This means that before being able to use it the text has to be transformed where '__label__' has to be prepended to the actual label itself. On the other hand it is not needed to further parse the corpus itself as this is handled by FastText. As an example, one can use n-grams with a program argument.

3.6 Discussion

As a last step we will do a final comparison of all the models on the imbalanced data. The reason why we do this on the imbalanced set is because the overall accuracy seemed to perform better on the imbalanced set. For this purpose we take a look at the final model performance on the test set that is unseen so far. The individual results can be found in table 2. The best performing model is BERT which has both the highest accuracy and f1-score, followed by Fasttext. The remaining models in decreasing order of accuracy and f1-score are SVM, LSTM and Naive Bayes. These models still achieved better performance than majority class prediction which would equal roughly 66%.

Model	Accuracy	F1-Score
Naive Bayes	73.0%	56.8%
SVM	81.9 %	75.1 %
LSTM	76.2%	69.6%
BERT	87.5%	84.3%
FastText	83.0%	78.9%

Table 2: Final model performance on test set

In addition to looking at performance, we also looked at the training times for the various models, which can be seen in table 3. It should be noted that for all models the training time with only the optimal parameters is used, instead of the time it takes to perform the entire grid search.

Comparing this to the results on performance we see that models that take longer to train tend to achieve better performance. Nevertheless, FastText honours its name and only takes 1 second and achieves $4,5\%$ less accuracy and 5.4% less f1-score when compared to BERT, which took 2800 times longer to train.

Model	Time in seconds
Naive Bayes	3
SVM	360
LSTM	150
BERT	2800
FastText	1

Table 3: Approximate training times

A general observation is that perhaps only the headline is not enough information to label the text as it is for example not ruled out that they are not ambiguous. An arbitrary example of such an ambiguous headline is *Hospitals are sued by 7 foot doctors*. From which it can not be inferred without deep contextual understanding, which is hard

to model in computers, if the doctors are 7 foot in length or they are seven doctors specialised in feet.

4 Conclusion

We have now taken a look at four different classification models, as well as a text classifier. As discussed in the results there was a continuous increase in accuracy for the consecutive models, NB, SVM, LSTM and BERT. However, this increase in accuracy also came with an increase in computation time, where BERT was the most time-intensive model. Even though this increase in performance is good, we conclude that is not great enough when compared to the FastText classifier. In other words, our tuned models do not outperform a default sophisticated text classifier when taking the training time and performance into account.

Some difficulties could arise when using this research in future projects is that the style of headlines could change over time, making the trained models less useful. Meaning that a model trained on articles from past COP editions might not be representative for future articles.

5 Task contribution

Additionally, in this section we list the individual contribution to this project in Table 4.

Task	Sjouke	Klaas
Data analysis	100%	0%
BaseModel	30%	70%
Naive Bayes	80%	20%
SVM	20%	80%
LSTM	100%	0%
BERT	20%	80%
FastText	100%	0%
Report	30%	70%
Overall contribution	50%	50%

Table 4: Task contribution

References

[AllSides] AllSides. Media Bias Ratings. https://www.allsides.com/media-bias/media-bias-ratingsl.

[Chawla et al.2002] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June.

[Cortes and Vapnik1995] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

[Devlin et al.2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

[Havrlant and Kreinovich2017] Lukáš Havrlant and Vladik Kreinovich. 2017. A simple probabilistic explanation of term frequency-inverse document frequency (tf-idf) heuristic (and variations motivated by this explanation). *International Journal of General Systems*, 46(1):27–36.

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

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

[Kulkarni and Shivananda2021] Akshay Kulkarni and Adarsha Shivananda. 2021. Converting text to features. In *Natural language processing recipes*, pages 63–106. Springer.

[Lemaître et al.2017] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *J. Mach. Learn. Res.*, 18(1):559–563, January.

[MediaBiasFactCheck] MediaBiasFactCheck. Mdia Bias Fact Check. https://mediabiasfactcheck.com/.

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

[Pennington et al.2014] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

[Webb2010] Geoffrey I. Webb, 2010. *Naïve Bayes*, pages 713–714. Springer US, Boston, MA.