Predicting Political Orientation in Journal Articles Concerning Climate Change – A Performance Comparison Study between SVMs and MLPs

Robin Sommer (s2997592)
r.m.sommer@student.rug.nl
&
Ákos Stéger (s4096908)
a.c.steger@student.rug.nl
&
Nik van 't Slot (s2536420)
n.g.van.t.slot@student.rug.nl

1 Introduction

For decades, global warming and humanity's impact on it have been a monumental worry for the planet. Multiple large-scale initiatives have been creating with the goal of slowing down this process. In 1992, the United Nations created an international treaty with the goal of reducing climate change. As part of this treaty, the United Nations Framework Convention on Climate Change (UNCCF) was called to life. This convention holds bi-yearly meetings which are used to discuss any progress (or regress) with regards to stopping climate change and reducing greenhouse gas emissions. These meetings are so-called Conferences of the Parties (COP)[1].

Due to the potentially catastrophic consequences of climate change, media coverage is widespread. Not every outlet focuses on the same aspects of the issue and the COP meetings. Where The Washington Post cover the bureaucratic processes and discussions between countries during a specific COP meeting, The Australian highlights the tax implications and investment suggestions that follow from the very same COP meeting.

In this research, we attempt to automatically predict political orientation of articles based around the coverage of climate change and global warming. Should the classification task reach high scores on the evaluation metrics, the argument could be made that coverage on this divisive issue varies greatly between right-centred and left-centred newspapers. Furthermore, the automatic detection of political orientation can be implemented into systems that try to analyze differences and patterns of typical political mindsets. Further research on these theory could be warranted if results of this research prove to be promising.

Orientation	Counts
Right-centre	9148
Left-centre	24512
Total articles	33660

Table 1: Distribution of political orientation labels in the training data.

In order to get a broader picture of what model works best for this binary classification task, we propose a comparison between a Support Vector Machine (SVM) model and a Multi Layer Perceptron (MLP) model. Additionally, we compare both systems to a simple baseline model in order to test benefits of either implementation. Validation is performed by 5-fold cross-validation to tweak the optimal feature selection and parameter values.

2 Data

The data that is used for this research consists of newspaper articles about climate change. These articles have been categorized into time periods centred around specific COP meetings. The training data consists of a total of 33660 articles. The articles come from 9 different newspapers from 4 countries. Each newspaper is classified as either a right-centre or a left-centre political orientation. Distribution of the political orientation label is reported in Table 1. The classes are heavily imbalanced with 78,2% of the articles classified as Left-centred. A more complete data distribution for the training set is illustrated in Appendix A. The class imbalance seems to stem from a staggering amount of articles originating from Australian and American newspapers. All newspapers that hold the majority share of articles in these countries are classified as left-centred. We discuss how we dealt with this class imbalance in section 3.

[1] https://unfccc.int/process/bodies/supreme-bodies/conference-of-the-parties-cop

As mentioned before, the articles have been split into several files corresponding to COP meetings. Data distribution per COP version is illustrated in Appendix B. It is important to note that COP6 and COP6a have been combined in this graph, which is the cause for the relatively large number of articles. COP1 does not feature any right-centred articles. Every COP version features more left-centred articles than right-centre articles, which means the COP version itself might not be an informative feature for our classifiers. The political orientation of every individual article has not been annotated for this data set. Instead, it can easily be derived from the gathered orientations of the newspapers themselves. The newspapers have been annotated on political orientation by AllSides[2] and Media Bias Fact Check[3]. AllSides allows individuals to rate the bias rating to create a community feedback per newspaper instead of reporting inter-annotater agreement. Inter-annotator agreement of Media Bias Fact Check is not reported.

The files for the training data are structured to contain the following information: COP edition, starting date, ending date and articles. The COP edition specifies the number of the specific COP meeting the articles are centred around. The starting and ending dates are the starting and ending points of the publication times of the articles in the file. They are one week before and one week after the meeting respectively. The articles key is an array which contains all the individual articles. Every article in turn has the following features:

- Path to the original document
- Raw text from the original document
- The name of the newspaper it has been published in
- The headline of the article
- The processed body of text of the article
- Several classifications by Lexis Nexis[4]. Examples include a list of subjects discussed and a list of organizations discussed.

As a final note, the testing data will be available to the authors after this report has been submitted. As such, it is not yet known what the label distribution or size of the testing data is. However, it can be reported that the testing data will consist of the articles from the next COP meeting and is identically structured to every individual training data file.

[2] https://www.allsides.com/media-bias/media-bias-ratings

[3] https://mediabiasfactcheck.com/

[4] https://www.nexisuni.com/

3 Method

In the current study, we have built both an SVM and MLP model in order to make predictions with them on the data set that was previously described. In this section we will apply a step-wise approach. After describing our baseline system, we will continue with introducing our two models. In both cases, we will start with describing a basic model where the intrinsic parameters of respective systems are set left at a default value. Furthermore, our default models will purely rely on unprocessed textual input. Subsequently, we will present the flow in which we selected our final features that resulted in our best models.

On a related account, although our data set allowed for a wide range of possibilities in terms of feature selection, in the present study we resorted to using the article bodies to predict the binary labels in question.

For the sake of evaluation, we have employed the classical metrics of machine learning, implemented from the sub-library *sklearn.metrics*. These entail a class-wise accuracy, precision, recall and F1 scores. For all the above metrics, a macro value will also be provided. For all of the models, we devised a 80%-20% split between training and test data respectively. In the current study we have used cross-validation instead of devising a separate validation set. For time constraints we resorted to use a 5-fold cross-validation in this experiment, also implemented from the *sklearn* library.

3.1 Baseline

As part of the exploratory research, we implemented a simple model based on the Zero Rule prediction strategy to establish a baseline for our study. Said baseline was implemented from the *sklearn* library. This approach, also called the most-frequent sense baseline, assigns to every label the most frequent value in the data set. Its results can be reflected upon in the following informative table:

This table provides insight into a a set of results

	Acc	Pre	Recall	F1
Right-Centre	66%	66%	100%	80%
Left-Centre	66%	0%	0%	0%
Macro	66%	33%	50%	40%

Table 2: Class-wise predictions of baseline classifier

	Acc	Pre	Recall	F1
Right-Centre	88%	87%	95%	91%
Left-Centre	88%	89%	73%	80%
Macro	88%	88%	84%	86%

Table 3: Class-wise predictions of default SVM classifier

that are indeed low, but these may not strike us as unexpected after all. The values above also reflect the great degree of skewness in the data set, that was already described in the Data-section above.

3.2 SVM

Our SVM is implemented through the SVC function of the *Sklearn.svm* sub-library. As described above, first we assemble a basic model with the default values of this utility. Among other various parameters that this function allows for, we have modified three in total: the regularization parameter 'C', kernel coefficient 'gamma' and the type of the 'kernel'.

The type of the 'kernel' is a rather self-explanatory element of our model. It provides several different possibilities depending on the separability of our data points. The default model comes equipped with a radial *basis function kernel* (rbf) which is also considered a computationally efficient method highly suitable for handling non-linearity.

'C' is responsible for penalizing our system for misclassification, and a higher number implies a greater margin for error. In the default scenario this is at 1.0.

'gamma' is a parameter inherent to non-linear kernels and by default it is set to a scaling function.

As the first step towards fine-tuning our system, we initially built our system with the aforementioned values. Also, since we wished to see how big an influence pre-processing of the textual input has on the performance, we left the text unprocessed in our default model. However, in order to be able to input our data into the system, our text data did pass through a pipeline containing a 'Tf-Idf' vectorizer. Upon investigating Table 3, we can see the results of our default model that uses the aforementioned default parameters and uses unprocessed text data that were directly extracted from the JSON files.

These results already reflect a substantial increase compared to our baseline model's comparison. This is a reassuring feedback, since we want to end up with a model that explains our data set with higher performance than a chance finding. In the following, we will see how our features and parameters were changed in order to obtain our best model.

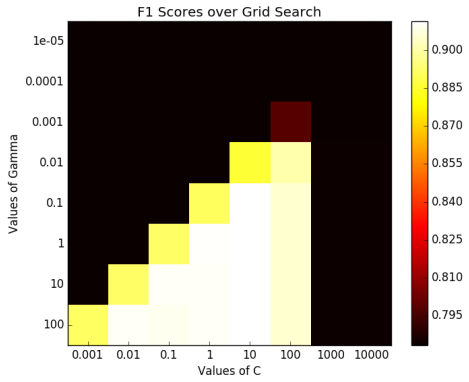


Figure 1: Heat map of results returned by GridSearchCV

The parameters we used for our best model were determined by running a GridSearch, using the 'GridSearchCV' function the corresponding *sklearn* sub-library. The search was conducted for the three aforementioned parameters, namely 'C', 'gamma' and 'kernel'.

Since we had to refrain from running a grid search on the whole data set for time constraint reasons, we took a representative subset of all data. When selecting the subset, we confirmed that the ratio of the two labels present in the subset is similar to that of the whole data set. Thus, we ensured that the findings of the heat map bear a proportional relationship to the whole data set. The grid search was run on 12415 files that represents 37% of all files and ran for 1231.9 minutes. The results are summarised in the heat map below:

As we can see in the heat map above, models with the highest F-scores used 'C' values between 0.01 and 10, and gamma values between 0.1 and 100. Apart from the best parameters, the features used as input also underwent modifica-

	Kernel	C	Gamma
sklearn.svm.SVC	RBF	100	0.01

Table 4: Best hyperparameters returned by GridSearchCV

tion. This, in our project entails including a simple pre-processing pipeline. This we have defined as a function that takes raw text as input and after lowercasing them removes stopwords, punctuation, digits and other non-alphanumeric signs. In the end, a list of clean tokens is returned: in the case of our current data set one token is equivalent to a paragraph. Just as above, the final tokenization in which the text is split into a list of standalone words will be finished by our vectorizer. For our best model, we retained the 'Tf-Idf' vectorizer for this purpose.

We explored several other ways to modify our features: at an early stage of the study we experimented with the usage of a stemmers and lemmatizers. We tried both 'PorterStemmer' and the 'Snowball' stemmer as well as the *WordNetLemmatizer* from the *NLTK* library. However, since we soon confirmed that setbacks showed both in terms of decreasing performance of our SVM model and a much higher computational cost, we decided to exclude these from our final models.

We also experimented with Ngrams, implemented through the built-in arguments of the 'Tf-Idf' vectorizer. Latter allows for specifying both a concrete Ngram number, or an interval thereof. Although we saw a rather moderate increase in the overall metrics, the most lucrative configuration (which was the usage of an interval between bigrams and trigrams) resulted in forbidding building times. Given the time constraints this paper was conceived under, we opted for dismissing Ngrams.

Note, that the text pre-processing protocol described above is the same we used in the case for our MLP as well.

3.3 MLP

For the default model we will construct a basic MLP which consists of a hidden layer with 200 neurons using no activation function, and an output layer which uses the softmax activation.
For constructing a feature vector we use the 'Tf-Idf' vectorizer as a default vectorizer. This provided the results shown in table 5 Here we see that the MLP does not perform at all. It does achieve an accuracy of 67%, however this accuracy does not say anything because in the table we can read the the MLP does not predict the Right-Center class at all. This has to do with MLP's susceptibiltiy for class imbalance. Even by a change in vectorization the MLP will still only predict the minorty class with great exception. To create a proper baseline we have tweaked the vectorization process as well as resolving the class imbalance in the data by undersampling the minority class, such that we have a 50/50 distribution. After this we obtained the following model:

- **epochs:** 15
- **batch-size:** 20
- **Vectorizer:** Word-embeddings. We construct the feature vector by taking the top 10 words of the article and applying word vectors of 300 dimensional GloVe embeddings. this will result in a feature vector of length $10 * 300 = 3000$.
- **Undersampling:** Since the results using the unfiltered data set provide an accuracy of about 67%, which is achieved by only predicting the majority class we will use undersampling to define a baseline which is able to predict both Left-Center and Right-Center articles.

The baseline for this model can be seen in 6.

label	precision	recall	f1-score	support
Left-Center	0.66	1.00	0.80	4467
Right-Center	0.00	0.00	0.00	2265
Accuracy: 0.67				6732

Table 5: Default parameters; results on the last 1/3 of the data set

label	precision	recall	f1-score	support
Left-Center	0.56	0.78	0.65	1839
Right-Center	0.62	0.37	0.47	1821
Accuracy: 0.58				3660

Table 6: Slightly changed parameters; results on the last 1/3 of the data set

3.3.1 Best Model

To improve on the base line we obtained by the default model we will tweak some hyperparemeters

as well as adding preprocessing of the document bodies, such that we will have a body which contains less irrelevant data.

Undersampling
Balancing vs no balancing: In the base line section we already saw that this makes a huge difference for the model. Without resolving the class imbalance of the classes the MLP practically only predicts the majority class. For the best model we use the same class imbalance resolving as the default model; undersampling the majority class such that both classes represent 50% of the data.

Preprocessing
We will examine a few preprocessing possibilities. The first is text normalization. This is a frequently used preprocess technique. This removes the punctuation and unnecessary characters from the text and also removes stopwords (the most common words in english) from the articles.
Next to this we have experimented with lemmatization and stemming the words in the article. In table 7 We can see the results. From this we can see that text normalization including lemmatization offers the best accuracy.

Method	**Accuracy**
Normalization	0.74
Normalization + Stemming	0.76
Normalization + Lemmatization	0.73

Table 7: Accuracies for including preprocessing; text normaliazion alone or combined with lemmatization or stemming

Number of epochs
In table 8 we can see the results of our experiments regarding the epochs we train our MLP. Here we can see we that we definitely need more than a few epochs, however for a large number of epochs the increase in accuracy on the development set becomes less and less, while the time increases linearly.

Epochs	5	10	15	20	30
Time (s)	27.9	60.5	87.7	166.62	220.09
Accuracy	58%	64%	66%	67%	68%

Table 8: Some table with numbers of epochs

Batch-size
In table 9 we can see the results of our experiments regarding the batch-size we use while training our MLP. We can observe that a large batch-size is not desirable, while a small batch-size does offer a better accuracy. Time-wise it is logical that the smaller batch-size takes more time. For our final best model we will choose a batch-size which is small, since the training time is less important than the eventual performance of the model.

Batch-size	30	20	10	5
Time (s)	166.62	267.72	389.06	743.68
Accuracy	67%	67%	69%	71%

Table 9: Some table with numbers for batch-size

Vectorizer
The vectorizer we used in the default model, which provided us with a base line was using the top 10 most frequent words from each article and using word embeddings to vectorize those 50 words and thus obtaining a feature vector for that article.
To improve on this we will create our own word embeddings from the training data. The word embeddings are created with the Word2Vec Python implementation Gensim [5]. The Word2Vec model is trained on the articles from COP1 until COP24. In order to do this, the article bodies first have to be preprocessed and tokenized. Preprocessing includes transforming all capital letters into lowercase letters and removing every stop word. Stop words are removed because we do not consider them to be as informative for training word embeddings as content words are.
The Embeddings are saved in a JSON file with 100-dimensional vectors. Only words that occur a minimum of 15 times in the articles are included in the final file. The articles contain a relatively large amount of longer, more complex sentences. Because of this, we opted to set the window parameter to 10. This means that every word is directly influenced by the 10 words before and after it.

Final model:
The final model consists of the following settings:

- **epochs:** 30
- **batch-size:** 5
- **Vectorizer:** Word-embeddings. Constructed from the training data of which the parame-

[5] https://radimrehurek.com/gensim/

ters are mentioned above. This results in a feature vector of length $10 * 100 = 1000$.

- **Undersampling:** Since the results using the unfiltered data set provide an accuracy of about 67%, which is achieved by only predicting the majority class we will use undersampling to define a baseline which is able to predict both Left-Center and Right-Center articles.

- **Preprocessing:** We added preprocessing to the model. The preprocessing that we have opted to use is basic text normalization and lemmatization as this proved to be most fruitful in our experiments.

4 Results

In this section we will proceed to present the results of the best models that we described in detail in the previous section.

The performance of the best SVM model can be observed below in the following table:

	Acc	**Pre**	**Recall**	**F1**
Left-Centre	89%	90%	94%	92%
Right-Centre	89%	87%	80%	83%
Weighted avg	89%	89%	89%	89%

Table 10: Class-wise predictions of best SVM classifier

The SVM classifier was executed with kernel type 'rbf', A C-value of 100 and a gamma-value of 0.1. We can see that the above values reflect a significant improvement compared to the baseline, however, if we compare it to the default model we might find as striking how little added effect the pre-processing protocol and GridSearch had. The model achieves a weighted macro F1-score of 89%. What could not be seen in Figure 1 was that the above model is also one of the most computationally effective ones at training times of 4302 and test times at 307 seconds.

To get a closer insight into the distribution of correctly labeled instances versus incorrectly labeled once, let us observe the following confusion matrix:

The confusion matrix of the binary classification task underlines the accuracy score that our system returned and the overwhelming majority of the instances are classified correctly.

	Left-centre	**Right-centre**
Left-centre	4186	281
Right-centre	464	1801

Table 11: Confusion Matrix of the best SVM model on a fold of 6732 articles

The performance of the eventual best MLP model was not as expected. We expected the MLP to challenge the SVM in performance, however this proved not to be the case as can be seen in table 12. The accuracy scores the final model produces dissappoint and the MLP model only marginally outperforms the default model and the base line.

label	**precision**	**recall**	**f1-score**
Left-Center	0.72	0.69	0.71
Right-Center	0.71	0.74	0.72
Accuracy: 0.72			
5 Fold Cross validation - Accuracy: 0.71			

Table 12: Class-wise predictions of best MLP Binary Classifier

Both the results shown in table 12 as well as the confusion matrix in table 4 show that it does well in predicting both the Left-Center and the Right-Center political orientation.

	Left-centre	**Right-centre**
Left-Center	1253	544
Right-Center	481	1358

Table 13: *Confusion Matrix of the MLP Classification on a fold of 3636 articles*

The MLP that is created is not necessarily the best model that can be created. Due to time constraints lots of parameters in the MLP model that can be tweaked as well as the construction of the MLP itself were not considered in this project.
It does however show that it is more difficult to construct an MLP that performs well on classifying large texts.

5 Discussion

This report raised several interesting questions both in terms of technicalities and contents.

First of all, we may declare that the study had a conclusive outcome with respect to our models outperforming the most common sense baseline we implemented in the beginning.

Second, we saw in the case of the SVM that the pre-processing protocol we devised and the grid search did not have a significant effect on the final scores of the best model after all. This raises the question as to what alternative techniques may be used for pre-processing and feature selection. In a future experiment the inclusion of further features can be considered, alongside with more effective randomized search techniques instead of the computationally costly grid search utilized in this study.

In this report, we attempted to create two machine learning models with the intention to predict the political inclination of newspaper articles that were released at the time of annual COP meetings. The bottom line of this experiment is twofold: first, we have proven that the above task can be carried out with a fairly great certainty. However, the fact that the language used by opposing political sides is in fact different to such a degree that it can be captured in the frames of an experiment like ours is thought-provoking to say the least. We believe it provides us a good opportunity to reflect on the need for a fresh, apolitical discourse regarding climate change. By doing so, we might establish a common ground among people regardless of their political views to be able to address this gravely serious issue together.

6 Individual Contributions

Below is a short listing of how we divided the workload.

6.1 Report

Our group split the report among us in the following way:

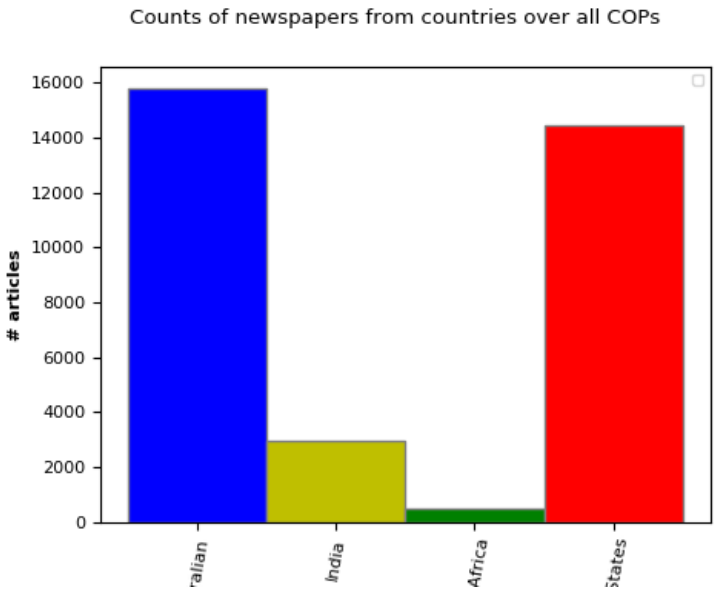
Ákos: Sections 3.0 & 3.1 & 3.2 & 5
Nik: Sections 1 & 2 & Small part of 3.3 & 4
Robin: Sections 3.3 & 4

6.2 Code

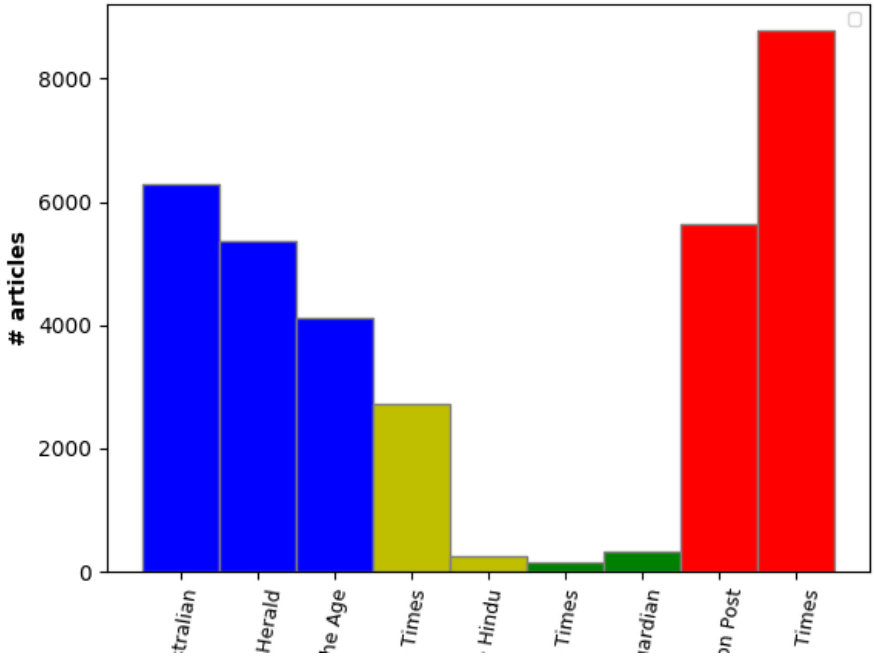
The GitHub repository for this project can be found under the link below, followed by the individual contributions in creating the code for this project: `https://github.com/Robin1711/finalprojectLFD`

Ákos: dummy.py, svm.py (with help from Nik)
Nik: word_embeddings.py, analyze_data.py, svm.py (final troubleshooting and debugging)
Robin: mlp.py, main.py, create_plots.py

A Figures of data distribution of total training data



Counts of newspapers over all COPs



B Figures of data distribution per COP