Group 5 NLP Project
Stance Detection in Article Headlines

Andrea Alfieri - 5128315
A.Alfieri-1@student.tudelft.nl
TU Delft
Delft, Netherlands

Avinash Saravanan - 4993381
A.Saravanan@student.tudelft.nl
TU Delft
Delft, Netherlands

Diego Albo Martínez - 5043204
d.albomartinez@student.tudelft.nl
TU Delft
Delft, Netherlands

Tomasz Motyka - 5146844
t.t.motyka@student.tudelft.nl
TU Delft
Delft, Netherlands

ABSTRACT

As fake news proliferates, different methods have been tested. Although human fact checkers are reliable, they lack scalability of methods that can be automated. Stance Detection has been found to be particularly useful in detecting disinformation [6]. In this paper, we attempt to reproduce the results of [2] and expand upon the techniques used to prepare the data and the classification techniques used to determine stance. Using similar feature extraction techniques like Bag of Words, we found that we were able to either match or exceed the performance of the methods described in [2] and were able to determine which hyperparameters to use with the classifiers we introduced to improve training performance.

1 INTRODUCTION

Stance detection is a useful NLP method in many areas such as news, reviews, tweets, etc. It is a position towards a certain topic typically split into categories representing support or opposition of that topic. Stance detection has been gaining more attention as social media has become ubiquitous. In addition to this, with the 2016 U.S. Presidential election, stance detection has become more relevant as a way to help recognize bias and attempts to influence the political opinions of voters [10]. As the amount of information individuals consume through different media outlets increases, it becomes difficult for them to determine whether the media that they are consuming is mainly biased towards one stance, or if it is more balanced.

Because of the wide range of possible applications for Stance detection, we wanted to explore what methods are currently being used, and if variations in how the experiment is carried out can improve upon those methods. To do this, we decided to reproduce *Emergent: a novel data-set for stance classification* by Ferreira and Vlachos [2]. The goal of our project will be of reproducing and validating the results the author obtained while thinking about possible improvements on the topic and evaluating what the actual effects of those possible improvements may be. We focus on not only testing the same feature extraction and classification methods as [10], but also trying to see what additional methods can be used to potentially improve upon the results found in the original paper.

2 BACKGROUND

To determine what to consider for potential improvements, we looked to previous works in the field. Many other works examined other data sets specifically relating to social networks such as tweets from Twitter [5, 9]. Another area that has been examined using MAP (Mean Average Precision) is rumor classification in microblogs which share many similarities to tweets [3]. Although many works focus on tweets, there are similarities in the datasets we are using due to the brief nature of both types of data. In regards to the use of machine learning techniques, there are papers that examined the use of certain classifiers within NLP. Specifically, "Thumbs up? Sentiment Classification using Machine Learning Techniques" examines machine learning techniques for sentiment classification. [7].

3 APPROACH

In this section we introduce the features we selected to use in this classification problem. We explain the tools that we used. We explain the hypothesis testing and statistical methods used to that end in Section 4

Data Cleaning

We've employed the following methods for pre-processing the data:

- Removing stopwords.
- Substitution of upper case characters with lower case.
- Removal of the punctuation.
- Lemmatization
- Stemming
- Removal of non-alphanumeric characters

Additionally, in order to ensure better reproducibility of our work, we initialized all `random_state` parameters to 0 where applicable That includes train-test set split, classifier initialization and cross validation randomization.

Tools

We used the following tools and languages to carry out our experiments.

- Python v3.8.2
- scikit-learn v0.22
- StanfordNLP v0.20
- NLTK (Natural Language Toolkit) v3.4.5
- NumPy v1.18.1
- spaCy v2.2.3
- spaCy Vectors for PoS tagging: `en-core-web-md` v2.2.5

Bag of Words

The Bag of Words feature measures the headlines in the most straightforward way. After normalizing the dataset —lower case and lemmatization of all the words in the headline— we gather all the terms in all the headlines of the dataset, which constitute the *vocabulary* for our Bag of Words model. After obtaining the vocabulary, each headline is then represented by which and how many words of the vocabulary it contains.

For gathering this information, we make use of `sklearn`'s built-in `CountVectorizer` class, which allows us also to count n-grams apart from single unigrams appearing in the vocabulary. By doing this, the number of resultant features are far more than words are in the vocabulary, therefore the most frequent and least frequent n-grams should be pruned from this BoW vector representation.

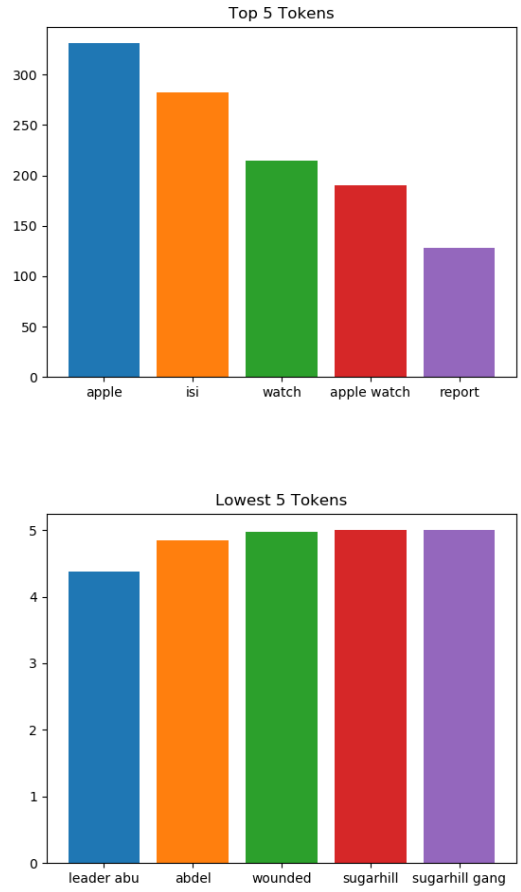
As part of preprocessing, we removed all non-alphanumeric characters and applied lower case to the headlines, then we lemmatized them using the `WordNetLemmatizer`. Furthermore, the stopwords[1] in the headlines were removed.

The `CountVectorizer` class allows the user to prune the vector implicitly when calling the constructor, hence we decided to keep the 1,000 most frequent unigrams and bigrams in the headline corpus.

TF-IDF

Aside from the standard implementation of the Bag of Words feature, given that it is headline stance what we are trying to model, some extra Bags of Words made up of refuting (e.g. however, not, despite) which typically negates an idea or hedging words (e.g. reportedly, possibly, might) which adds vagueness and passiveness to a document were added to the feature set.[2] This inclusion attempts to increase the ability to discriminate *against* and *observing* headlines respectively.

Figure 1: (a) Top 5 tokens in our BoW representation across all the headlines and claims (b) Least frequent tokens across the whole corpus. The stopwords present in the top tokens are removed to reduce the noise they introduce.



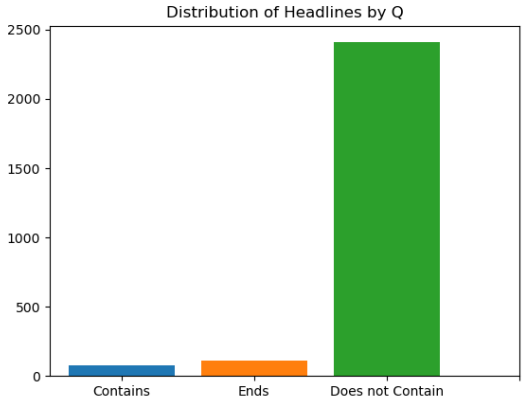
Based on the same idea but addressing some key drawbacks from the BoW model, TF-IDF —short for *Term Frequency-Inverse Document Frequency*— provides a more balanced and smoothed vector representation of the n-grams in a corpus. It does this by compensating for the rare or infrequent words and sharing some density from the most frequent words with them. Our goal with this feature is to exchange it with the BoW feature, and examine whether we obtain a better performance due to this smoothed representation of the words, instead of the more sparse one of regular BoW.

In the implementation, we use `sklearn`'s `TfidfTransformer` class in order to apply the transformation to the previously computer BoW vectors.

[1] Throughout this paper, whenever we refer stopwords, we refer to those defined in the `nltk.corpus` package as English stopwords

[2] This feature is referred to as Ref_Hedg_BoW

Figure 2: Distribution of the Q_{ends} and the $Q_{contains}$ features in the dataset. In the graph, the Contains column represents the subset of those headlines that contain a question mark but not in the end. The true number of headlines with a question mark would then be the sum of both columns



Q-Feature

When interpreting news headlines, one might often think of headlines supporting a certain claim as being more direct and to the point, while observing or refuting headlines might include terms reflecting certain disbelief or neutrality. One basic way of extracting this information is by examining whether the headlines contain interrogation signs. This is what we call *Q-Feature*, and its composition is two-fold. On the one hand, we extract whether the headline ends with a question mark —what we call Q_{ends} feature—, and on the other hand, the less restrictive condition of whether the headline contains a question mark, no matter the place, which we denote by $Q_{contains}$. The distribution of the headlines according to this feature can be found in Figure 2.

This unfolding of the Q-Feature into two different features is motivated by the fact that many headlines are not comprised of just a question, but ask a question to the reader and then proceed to develop on the content of the article further before the end of the headline. In this way, we can also capture that subtle difference and see how well we can differentiate the three classes thanks to it.

Word2Vec

In order to achieve high dimension similarity metrics between headline and claim, we choose to include also the word embedding or Word2Vec representation of the headline and compute its similarity with its corresponding claim. We do this with the hope that against headlines it might be harsher and therefore share fewer similarities with the objective claim. Since it is a high-dimensional representation we are going for, we use FastText vectors[3], which have the property that they not only identify unigrams, but also break words into multiple sub-grams and are able to match them[4]. These vectors have also the good property that they were trained on Common Crawl[5], which provides a huge and diverse corpus, and is very well suited to the colloquial level of expressions in the headlines we examine.

To be able to work with these vectors, we make use of the Spacy[6] library, which lets us create and import custom vectors to use as our model.

As the metric used to measure the similarity between word embeddings, we use the Spacy *similarity* measure, which in the end computes the same result as calculating the cosine distance between the averaged word vectors. Another method would be reducing the vectors by multiplying the components rather than averaging, however, given the number of vectors and dimensions, the product vanishes and never returns a usable result.

As preprocessing steps for this feature, the headlines are first converted to lower case and then stripped of non-alphanumeric symbols and stopwords. After that, we use the WordNetLemmatizer to lemmatize every token in the headlines and then apply the similarity measure described in the previous paragraph.

Length-Diff

Another intuition is that headlines that are observant might be more lengthy than the other two types of headlines. For this reason, we add the Legth-Diff feature, which measures the difference in length between the headline and the claim.

$$LD = len(headline) - len(claim)$$

We calculate the normal difference instead of the absolute value in order to allow a bigger variance of the values which might facilitate discrimination between the classes.

Root Dist

The Root Distance (RootDist) feature measures the distance from the specific word to the root of the sentence in the dependency tree of the sentence. In our case, we added 2 features using this approach. First/second denotes the minimum distance of any refuting/hedging word to the root of the sentence. Also, if no refuting/hedging word is present in the sentence we set the value of the feature to 0, to reduce its influence on the model (setting some arbitrary high

[3]The vectors used are number 3 from https://fasttext.cc/docs/en/english-vectors.html, which contain 2M vectors with 300 dimensions
[4]https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c
[5]https://commoncrawl.org/
[6]https://spacy.io/

value could make difference to the model even though the value has no meaning). The lists of refuting/hedging words were created by hand and the dependency structure for each sentence was created using Stanford CoreNLP[7] package. We chose this package because of its ease of use and because it's trained on a Wall Street Journal dataset, which might be beneficial since the article headlines may come from the newspapers. However, the downside is that the corpus probably does not contain a colloquial language and much of Internet gossip portals are written using colloquial language. The intuition that lies behind this feature is that the closer to the root the refuting/hedging word is, the more likely it affects the root.

Claim-Article Alignment

While the article headline often provides adequate features to classify its stance, we also need to take into account its entailment relation with the claim. Therefore, for each pair (claim, article headline), we compute a score denoting alignment. The procedure is as follows:

(1) For each word pairing between the claim and the headline we compute a weight by the following scheme:
   - if stems of the words are the same then assign *maxWeight*.
   - if the words are paraphrases according to PPDB[8], assign their maximum paraphrase weight
   - otherwise assign *minWeight*
(2) Now, after assigning weights, we obtained the bipartite graph, where disjoint sets are the words from the claim and the words from the headline. To compute the alignment score one needs to solve the maximum weight matching in a weighted bipartite graph.
(3) To solve such problems, we use the Kuhn-Munkres algorithm, which achieves $O(n^3)$ runtime complexity.
(4) Obtained maxed weight is then normalized by dividing the value by the number of words in the shorter of the claim or headline.

SVO

Subject-Verb-Object triplets (SVO) try to further expand on the bag of words model by incorporating certain syntactical knowledge into the feature extraction. Rather than relying on simple words counts or position within the sentence, SVO tries to find the main subject of the sentence, the verb or action, and also the object.

We extract these triplets (sometimes more than one per sentence), from the headline and its corresponding claim, and through the PPDB paraphrase database [8], we try to find whether the matching pairs of subjects, verbs and objects are in any way related. Using the fully-fledged version of PPDB, we attempt to not only detect exact one-to-one unigram relationships, but also phrasal and syntactical relationships between the terms.

For finding the triplets, we make use of Spacy's POS annotation functionality[9], as well as their medium-sized English vectors. The routine is as follows: first, we split the headline based on different punctuation signs, which renders multiple sentences in some cases. Then, we parse the sentence with Spacy and annotate all the chunks or Subject, Verb and Predicate. From those we extract based on the links between them —easily accessible through the POS methods— the core of those chunks and the word they're dependent on.

4 EXPERIMENT

Statistical Tests

Intending to evaluate the discriminative power of our features, we conducted several statistical tests to try and quantify how well we should expect our features to help the model perform better at telling apart the three classes.

For the numerical features —Word2Vec, Length-Diff—, we first ran a Shapiro-Wilk test of normality, which confirmed that none of our features would have been appropriate alongside a Student's t-test for significance. Thus, we performed the significance test using the non-parametric Mann-Whitney U test. Meanwhile, for the binary features like both Q-Features, we conducted the test by applying a T-test from descriptive statistics.[10]

Table 1: Results of Significance Tests of how well certain features impact class when taking into account each pair of classes. Column labels: For (for), Observing (Obs), Against (Agst). With an asterisk, the best feature to discriminate between each pair of classes

	p-value		
Feature	For-Agst	Obs-Agst	For-Obs
Q_{ends}	0.0114	2.24×10^{-5}	1.34×10^{-20}
$Q_{contains}$	5.68×10^{-7}	2.87×10^{-3}	4.40×10^{-24}
Word2Vec	1.87×10^{-6}	5.28×10^{-3}	3.2×10^{-3}
Length-Diff	9.96×10^{-6}	0.2198	1.35×10^{-13}
RootDist-hedging	4.99×10^{-36}	2.90×10^{-8}	$1.39 \times 10^{-102}*$
RootDist-refuting	$5.20 \times 10^{-186}*$	$2.42 \times 10^{-131}*$	2.54×10^{-5}
Claim-Article Align.	0.02	0.146	0.105

The obtained results can be visualized in Table 1. In all these tests, our null or initial hypothesis H_0 was that all the three classes presented the same distribution and that the designated feature would not be discriminative. The results

[7] https://stanfordnlp.github.io/stanfordnlp/
[8] http://paraphrase.org/#/download
[9] https://spacy.io/usage/linguistic-features
[10] https://docs.scipy.org/doc/scipy-1.4.0/reference/generated/scipy.stats.ttest_ind_from_stats.html

show, however, that except for the Length-Diff measure relating to the Observing and Against classes, all the measures we use are discriminative in terms of class distributions.

Other results that can be highlighted are the fact that Word2Vec similarity seems to be the least useful measure in terms of the average difference in distributions, as well as the good performance of the newly incorporated $Q_{contains}$ and Length-Diff features.

Something to take into account is that the most difficult classes to discriminate are the observing and against classes. We will discuss this fact in the following sections and evaluate the results according to how well these two categories are distinguished by our method.

Dataset Bias

Even though the original authors of the dataset as well as the paper we are reproducing [2] affirm that"none of the stance labels dominate the distribution", there is a perceivable imbalance in the labels in the dataset, with *for* accounting for 47% of the samples, while *observing* and *against* make up 37% and 16% of the samples respectively. As can be seen, the *Against* class is hugely under-represented in the data. A visual representation of the imbalance is given in Figure 3.

This could either be due to the true prior distribution of article stances, which would imply that writers are more prone to write headlines supporting or neutral with respect to the content, or that there was a mistake in the data gathering process. If the second was the case, we would have to try to balance the classes by dropping some examples from the other two classes to make the labels uniformly distributed.

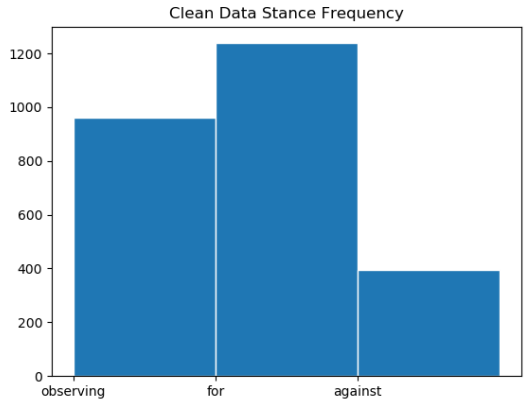
Another thing to note is that according to [2], each headline is constructed by a journalist to summarize an article that supports a particular claim. In addition to this, the journalist also decides the stance of the article once enough evidence has been collected. This means that there can be expert bias involved due to each journalist having a different opinion on stances.

Model Selection

Even though the author of the reproduced paper used a Logistic Regression model, we would like to cover a wider scope of models. For that reason, we aim to provide the results obtained by using four different classifiers.

(1) **Naïve Bayes.** Despite some if not many of our features being correlated, past studies show that Naïve Bayes can be one very good predictor even with dependent features [1]. Plus, the independence assumption makes NB faster to train than other models, and allows us to include a bigger feature space.
(2) **Logistic Regression.** Partly because it was the model that the author used —alongside L1 regularization to try to perform feature selection altogether—, and partly because it is shown to perform better than Naïve Bayes in most of the cases. It is also useful to look at the resulting coefficients of the L1-regularized model, since we can also get some insight into which features are more and less useful for the classifier for discriminating the classes and act accordingly.
(3) **Support Vector Machines.** Even though Logistic Regression has some nice properties like being maximum-likelihood based, and reporting good results across the board in other work, the scale worse than SVMs to high dimensional spaces. SVMs allow us to include more features in our model knowing that it will converge generally faster than Logistic Regression while also having nice non-overfitting properties through the regularization parameter than we will have to optimize.
(4) **Random Forest.** Because Random Forest is often used in applied settings by many companies for NLP, we were interested in seeing how it behaved with the emergent dataset. Unlike the previously mentioned classifiers, Random Forest uses a collection of decision trees that are generated by considering a subset of features at each step to create a variety of trees. This gives it the advantage of greater robustness to outliers.

Figure 3: Distribution of labels in the dataset. As can be seen, the *against* stance is more scarcely represented in the samples



Measures

Given the imbalance of classes present in our dataset and explained beforehand in Section 4, accuracy becomes a suboptimal criterion on which to judge the performance of our classifier, since it can underestimate the importance of misclassified examples of the least common class. For this reason,

throughout the evaluation phase we will employ measures that account for this imbalance in the labels, those are Precision, Recall and F1-Score, which is a combination of both.

When averaging these measures in the multiclass —more than 2 classes— case, there exist different criteria as to how to apply the averaging. In this paper, unless stated otherwise, we use the *weighted* measures, which take into account the number of samples from each class when calculating the overall score. The choice of this weighted average can render *a priori* wrong looking results like the F1 score not being in the interval between precision and recall, as can be seen in Table 3, however this is expected due to this particular method we use, which we consider the most suited for our type of data.

Tests

Concerning our evaluation methodology, we first randomly split the dataset in two different sets: A Training set on which we will perform Cross-Validation comprising 80% of the examples, and a separate Test Set made up from the remaining 20%, on which we will test the generalization performance of all the classifiers. As stated previously, the dataset has heavy label imbalance, for that reason, whenever we refer to Cross-Validation in the paper, we specifically mean **Stratified Cross Validation**, in which each fold has the same distribution of labels percentage-wise as the full dataset, thus minimizing the effect of the skewed label distribution.

We could further separate our process into two different cases depending on the nature of the classifier we are using:

(1) For classifiers that **do not require hyperparameter tuning**, like Logistic Regression and Naïve Bayes, we train them using 10-fold cross validation on the training set in order to get average training scores, and then test them on the test data.
(2) For those classifiers that **require hyperparameter tuning**, like Random Forest and Support Vector Machines, we first perform a Grid Search 5-fold cross validation on the training set to determine which hyperparameters to use. After that, we run another cross validation with the chosen parameters to estimate training performance, and finally test the best performing estimator on the separate test set.

We favor the use of k-fold cross-validation rather than having a separate validation set since, as suggested by literature [4], with not so big datasets the model might need all the data possible in order to fit the function more appropriately.

We choose 5-fold cross validation for Grid Search in order to balance performance with speed, as when searching in a big grid of possible parameters, a high number of folds greatly harms the converging speed of the search. For the rest of cross validations, we choose $k = 10$ since we have a considerable amount of samples and hence we are able to provide an estimate with less variance while also being able to make each fold a fair representation of the overall distribution of the labels.

Listing 1: Chosen hyperparameters for each classifier.

```
"Logistic Regression": {
  "penalty" : "L1",
  "C" : 1.0,
  "solver" : "liblinear"
  },
"SVM" : {
  "kernel" : "poly",
  "C" : 0.1,
  "degree" : 2.0
  },
"Random Forest" : {
  "criterion" : "gini",
  "max_depth" : 50.0
  "n_estimators" : 100.0
  }
}
```

5 RESULTS

After all the considerations explained in the previous sections, we ran the experiments and recorded the results. As stated above, we first tested how the classifiers performed on the 10-Fold cross validation on the training set. The results of these tests can be seen in Table 2. As can be seen, on average, it is Random Forest the classifier that performs better on the training set, also worth mentioning is the surprisingly good performance of Naïve Bayes even when having very correlated features like in our case.

Table 2: Performance achieved by the different classifiers on the 10-fold cross validation of the training set in terms of mean measures and their standard deviation throughout the 10 folds

	Metrics			
C	F1	Precision	Recall	ROC-AUC
LR	75.42 ± 3.01%	76.08 ± 2.92%	75.77 ± 2.98%	87.58 ± 2.34%
SVM	67.55 ± 2.25%	73.08 ± 1.81%	69.60 ± 2.00%	75.85 ± 3.41%
NB	67.53 ± 4.395%	73.25 ± 4.01%	69.79 ± 3.73%	77.68 ± 5.07%
RF	77.24 ± 2.90%	78.34 ± 2.72%	77.64 ± 2.80%	89.64 ± 2.07%

After optimizing the classifiers and getting a first estimate of what their performance would look like, we run the final testing on the same held-out test set. The results achieved can be seen in Table 3. In this case, Logistic Regression performs the best from among our classifiers. Random Forest falls behind in this generalization step. The relatively higher

Table 3: Performance achieved by the different classifiers on the Independent Test Set

	Metrics			
Classifier	F1	Precision	Recall	ROC-AUC
Logistic Regression	78.95%	79.77%	79.38%	81.93%
SVM	69.03%	73.99%	70.91%	73.36%
Naive Bayes	69.27%	74.15%	71.10%	73.55%
Random Forest	77.98%	79.20%	78.42%	80.84%

complexity of Random Forest might be the reason why it deals worse with generalization than its counterparts. Moreover, SVM also performs below expected in training and testing, which might be due to the relatively low amount of features compared to the number of examples.

We then compare the results obtained by our best generalizing classifier and the those presented by Ferreira and Vlachos, who also used a Logistic Classifier for the task. In this case, since we want to get a deeper intuition of which cases our classifier handles better or worse, we compare both approaches using per-class precision and recall, which provides interesting insights, as presented in Table 4. We include the accuracy of the model because the authors used it to describe their model in the original paper.

Table 4: Comparison of our best performing classifier with the results of Ferreira and Vlachos [2]. The first column represents the accuracy of the model. The following columns represent the precision/recall for a specific headline stance

Classifier	Accuracy	For	Against	Observing
Ours	79%	78%/91%	86%/63%	79%/70%
F&V	73%	71%/89%	82%/70%	74%/54%

As can be seen, our classifier performs better across the board when compared to the one proposed in [2]. The only point of decrease in performance is the recall for the *Against* class, although it improves in precision. These differences can be due to different feature engineering decisions or tools, such as the inclusion of the Length-Diff and $Q_{contains}$ features, or the use of Spacy for SVO extraction. In the end, we can say that possibly the low recall of the *Against* class has to do with the improved recall of the *Observing* class. Since the later is more than two times as common as the former, the classifier tends to prefer misclassifying *Against* examples as *Observing* or even *for* rather than the true label.

Ablation Test

Given the previous results, we went further in order to determine how different features affect the performance of the classifier and performed ablation study. We chose Logistic Regression classifier to do that, since it was the one that performed best on our test set. With this aim, we trained our model using all the features except for one, and varied this held-out feature to try and identify which feature plays a bigger role in the performance of our model.

The results are presented in Table 5, with the left column indicating which of the features was left out in the testing, and the other two columns reflecting the results on the training set (CV) and the independent test set. All the numbers given are the weighted F1-score since it condenses precision and recall, as well as compensates for the class imbalance of the labels.

Table 5: Ablation study. The three most relevant features are highlighted in bold in the first column. TF-IDF is not included since it performs worse than BoW in most cases

	Change in performance (F1-score)	
Feature	CV	test
-BoW	−5.014%	−6.238%
-Ref_Hedg_BoW	−0.011%	±0.0%
-Q	−2.728%	−1.769%
-W2V	−0.017%	−0.61%
-RootDist	−0.979%	−0.904%
-SVO	−0.0499%	−0.585%
-Claim-Article Align.	+0.044%	−0.413%
-Length-Diff	+0.138%	−0.203%

Forward selection

Moreover, in order to find the 5 most important features for our Logistic Regression model, we decided to run the forward selection algorithm, which is "a type of stepwise regression which begins with an empty model and adds in variables one by one"[11]. The resulting top 5 features, ordered the same way as they were added by the algorithm, are:

(1) Bag of Words
(2) Root-Dist
(3) Q-Features
(4) Claim-Article Alignment
(5) Length-Diff

These results confirm that BoW is the most relevant feature, however BoW is comprised of 1,000 different unigrams and bigrams, which also helps remark the importance of Root-Dist which manages to be the most discriminative after that while using only two numbers to quantify the classes.

Feature Relevance & Confusion Matrices

We examined the confusion matrix of our best performing classifier. As can be seen in Table 6 the classifier had the

[11] https://www.cs.princeton.edu/courses/archive/fall08/cos436/Duda/FS/stepwise.htm

biggest problems with misclassifying a lot of *Observing* samples as *For* and vice-versa. It is quite surprising, since based on the results of our statistical tests in Table 1, we could expect this 2 classes to be discriminated the best. This may be due to some hidden correlation between the features, that undermines the discriminative power for these 2 classes.

Table 6: Confusion matrix for our best classifier.

		Predicted			
		For	Observing	Against	Total
True	For	895	321	22	1238
	Observing	399	509	54	962
	Against	95	62	238	395
	Total	1389	892	314	2595

To verify the results of the statistical tests from Table 1, we looked at the confusion matrix of the prediction for our model. The table suggests that both *RootDist* features are significantly discriminative. The Logistic Regression model trained using just these 2 features gave the following confusion matrix:

Table 7: Confusion matrix for Logistic Classifier trained on both *RootDist* features.

		Predicted			
		For	Observing	Against	Total
True	For	1162	64	12	1238
	Observing	524	405	33	962
	Against	105	44	246	395
	Total	1791	513	291	2595

The *RootDist-hedging* feature is particularly good in discriminating between *For* and *Observing* classes. Removing it from the feature set and training just on *RootDist-refuting* feature should yield a confusion matrix, in which results for *For-Against* and *Observing-Against* pairs are more or less the same but there is a significant drop in accuracy in terms of pair *For-Observing*.

Table 8: Confusion matrix for Logistic Classifier trained solely on *RootDist-refuting* feature.

		Predicted			
		For	Observing	Against	Total
True	For	1226	0	12	1238
	Observing	929	0	33	962
	Against	149	0	246	395
	Total	2304	0	291	2595

As can be seen in Table 8, this is indeed what happened. Now, almost all the *Observing* samples are classified as *For*, since the *For* class is the predominant one. On the other hand, if we would keep the *RootDist-hedging* feature, we should see the drop in accuracy in terms of *Observing-Against* pair. Although *RootDist-hedging* feature has significantly less discriminative power than the *RootDist-refuting* feature in terms of *For-Against* pair, it is still large enough that we should not see the drop in the accuracy. The Table 9 confirms the expectations. The vast majority of *Against* headlines is now classified as *For*.

Table 9: Confusion matrix for Logistic Classifier trained solely on *RootDist-hedging* feature.

		Predicted			
		For	Observing	Against	Total
True	For	1172	66	0	1238
	Observing	540	422	0	962
	Against	286	109	0	395
	Total	1998	597	0	2595

6 CONCLUSION

In this paper, we tried to reproduce the results presented by Ferreira & Vlachos, while also adding some extra features and a more thorough evaluation with different classifiers than in the original paper. The results obtained were better than what the former paper suggested, which confirms that the results of the original paper not only are reproducible but improvable. We also confirmed that simple features like BoW and the presence of question marks are greatly discriminative in the process of distinguishing between stances. Nevertheless, syntactical features such as *RootDist* also play a big part in the good performance of the classifier. On the contrary, more refined features like word2vec seem to not have such a big influence in this particular setting.

Aside from the main objective of providing a successful reproduction, we also discussed the possible reasons for the lower performance of the method in less common classes, as well as conducted statistical analysis in order to support the results of the experiment on firmer pillars.

Overall, we also tried to guarantee the best reproducibility of our work, hence the detailed explanation of the features and the different criteria and implementation decisions that followed. We hope these explanations could help other people successfully reproduce this paper.

The source code developed for this project can be found in the following repository: https://github.com/andreaalf97/NLPproject.

We would also like to refer the readers to the original work The source code developed by [2], whose source code can be found in the following repository: https://github.com/willferreira/mscproject.

REFERENCES

[1] Pedro Domingos and Michael Pazzani. 1997. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning* 29, 2-3 (1997), 103–130.

[2] William Ferreira and Andreas Vlachos. 2016. Emergent: a novel data-set for stance classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, 1163–1168. https://doi.org/10.18653/v1/N16-1138

[3] Kazi Saidul Hasan and Vincent Ng. 2013. Stance Classification of Ideological Debates: Data, Models, Features, and Constraints. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*. Asian Federation of Natural Language Processing, Nagoya, Japan, 1348–1356. https://www.aclweb.org/anthology/I13-1191

[4] Max Kuhn and Kjell Johnson. 2013. *Applied predictive modeling*. Vol. 26. Springer.

[5] Michal Lukasik, Trevor Cohn, and Kalina Bontcheva. 2015. Classifying Tweet Level Judgements of Rumours in Social Media. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, 2590–2595. https://doi.org/10.18653/v1/D15-1311

[6] Razan Masood and Ahmet Aker. 2018. The Fake News Challenge: Stance Detection Using Traditional Machine Learning Approaches. (09 2018).

[7] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*. Association for Computational Linguistics, 79–86. https://doi.org/10.3115/1118693.1118704

[8] Ellie Pavlick, Pushpendre Rastogi, Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2015. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 425–430.

[9] Vahed Qazvinian, Emily Rosengren, Dragomir R. Radev, and Qiaozhu Mei. 2011. Rumor has it: Identifying Misinformation in Microblogs. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Edinburgh, Scotland, UK., 1589–1599. https://www.aclweb.org/anthology/D11-1147

[10] William Yang Wang. 2017. "Liar, Liar Pants on Fire": A New Benchmark Dataset for Fake News Detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, Vancouver, Canada, 422–426. https://doi.org/10.18653/v1/P17-2067