Exploratory Study on Protest-related Sentence Classification in News Media

Simge Ekiz (s4706757), Robin Wubs (s4135229)

13 January 2019

1 Abstract

We explore the feasibility of several machine learning algorithms on the classification of sentences from English language articles as past protest, planned protest or non-protest event. We work with a dataset consisting of 23452 English sentences provided by Koç University in Turkey. We experiment with several different feature selection methods to find the best performing feature set. When using f1-scores the algorithms perform very poorly. We discuss reasons why this happens and what could be done to improve performance.

2 Introduction

Protests are import events for a country. Protests show that the people are unhappy and are an important tool for effecting change. In the recent years the Arab Spring has shown us how protests can incite the people and spread through countries with far reaching consequences. But even at a small scale protests have a measurable impact. Last year the Dutch government changed it's policy on what to do with the animals in a small nature reserve called the Oostvaardersplassen in response to a protest. Such an important political tool should of course be studied, but understanding protests is not just important for researchers. The Arab Spring is an important example of how social media and online news has influenced the way protests spread. News organizations need to be able to follow these processes as they unfold and tracking protest events is one of the most important aspects.

One of the most promising tools for tracking protest events is automated news article classification. However, event extraction from news articles is a complicated task. Even though event-related entities are correctly detected, ambiguities occur while forming an event from them. By correctly classifying event occurrence on the sentence level we expect to reduce the chance of ambiguity. Moreover, we classify the sentences as past protest, planned protest or non-protest event. The added granularity of analyzing articles on the sentence level decreases ambiguity as many articles on planned protests may also mention past protests confusing the classes on an article level.

All work is done in Python 3.5, open sourced, and can be found in our GitHub repository[1].

3 Related Work

Text classification is not a novel idea and many articles have been written about it. From comparisons of feature selection (Forman, 2003) to entirely new classifiers (Joulin et al, 2016). Using news text mining to predict future events has also been researched (e.g. Randinsky and Horvitz 2013). One of the publications that is most closely related to our research is by Muthiah et al. (2015) They have developed a detection system for protest events from key phrases in news and social media texts. Their approach differs from ours in that they start with pre-selected words to build key-phrases to search for. These key phrases are constructed from a key word (e.g. protest, demonstration) linked to a future-oriented verb (e.g. plan, announce). Instead we start with fully formed phrases (sentences) which are labeled, and features are extracted from these sentences. This gives a large scope of association between phrases and classes not limited to certain key words.

4 Data Preparation

The dataset we used is created and provided by Koç University Artificial Intelligence Laboratory[2] and consists of 23452 English language sentences from 1642 separate articles. These articles are obtained from New Indian Express, an Indian newspaper. Of these sentences 8337 are annotated by two annotators by three tags which are (0) non-protest, (1) protest, (2) planned protest. The annotated data consists of 6876 non-protest, 1299 protest and 162 planned protest labels.

During the annotations, when a disagreement occur between the annotators, they discuss their thoughts and choose a label together as the final decision. In this study, we only use this agreed and finalized version of the annotations. Therefore, no inter-annotator agreement measurement is applied, since the annotators solve their disagreement together.

5 Feature Extraction

We extract two feature sets that we combine to form our final feature set to be given to a model after a feature selection method is also applied.

[1]Protest Events Detection source code: https://github.com/simgeekiz/ProtestEventDetection, Accessed on 13 January 2018

[2]Koc University AI Lab. homepage: http://ai.ku.edu.tr/, Accessed on 13 january 2018.

The first feature set is a matrix of TFIDF scores of the words. TFIDF stands for term frequency-inverse document frequency and assigns a value to each word based on the frequency in the text. Because the term frequency would give unreasonably high values to very common words there is a penalty applied based on how many documents the word appears in. For this set of features there are several pruning options. First, of all, the TfidfVectorizer function, in Scikit-learn[3] Python library, has a built in functionality to remove stop words from the feature set, this removes the most common words in the English language that have no real feature value. Another pruning option is restricting the usage of words based on their document frequencies by defining minimum and maximum thresholds. However, deciding on these thresholds and selecting a stop words set cannot be done without running tests. Thus, we have optimized selection of these parameters by training a Support Vector Classifier (SVC) and using GridSearchCV functionality in the same library. For selecting the document frequency thresholds we test the percentiles given in Table 1. Likewise, we test three different versions of stop words sets. First option is that not removing any word, the second one is using the built-in English stop words set, and the last is only pruning numbers from text. We see that the numbers set is the best performing. The reason may be the fact that the built-in set contains words like 'will' which can be useful to detect future protests while the numbers do not add any additional information.

The second feature set is generated using spaCy[4] NLP package and consists of a 'bag of tags' containing two types of tags, namely part-of-speech (PoS) tags and named entity recognition (NER) tags. As PoS tags, we use OntoNotes 5 version of the Penn Treebank tag set[5] which contains 55 different labels including labels for different tenses of verbs. We expect that this may contribute to separating future protests from present ones. As NER tags, we use OntoNotes 5 tagset which contains 18 labels including labels for nationalities, religious or political groups, various types of locations, organization names and dates. We expect that using such information as feature may help to detect protest event related text because a news about protests is likely to mention about political groups, organizations and give locations and dates to describe an event.

Maximum DF:	1, 0.90, 0.80, 0.70, **0.60**
Minimum DF:	0, 0.00009, **0.0001**, 0.0003, 0.0005, 0.0009, 0.001, 0.002
Stop Words Set:	None, Built-in Set, **Numbers**

Table 1: Optimization options for TfidfVectorizer. Best scoring combination is marked as bold.

Finally, after combining the two feature sets by concatenating the vectors, we use feature selection to select only the most descriptive features using a

[3]Scikit-learn Python library: https://scikit-learn.org, Accessed on 13 January 2018.
[4]spaCy NLP package: https://spacy.io/, Accessed on 13 January 2018.
[5]PoS tags in spaCy: https://spacy.io/api/annotationsection-pos-tagging, Accessed on 13 January 2018.

percentile cutoff. To determine which features are more effective and should be selected, we use the χ^2 measurement. What percentile of the data to be kept is tuned inside the hyper-parameter optimization process of each model separately as explained in Section 6. Thus, we use the resulting data to train our models.

6 Experimenting with Models

We train and evaluate several different models and compare them to find the best fitting one for this task. The models we experiment with are Support Vector Classifier (SVC), Decision Tree and Random Forest. In order to apply these algorithms we use their implementations in Scikit-learn Python library, as well.

For each model we apply hyper-parameter optimization and use GridSearchCV to find the parameters that give the best results. GridSearchCV trains a model for each of the combinations of parameters provided and uses a scoring function to measure the performances of the models. In this study, we explicitly use macro averaged f1-score to prevent one class dominating the others in the scoring. Because mean accuracy and the micro averaged f1-score are measuring the performance on the total number of labels, regardless of the type or importance of the classes. Therefore, when the accuracy or micro-average scoring is used in our case, we observe high scores because most of the instances are labelled as non-protest and the classifier predicts most of the labels as the non-protest. When the accuracy score for the label non-protest is high, the overall result becomes high as well, even though the other classes perform low. This kind of high score does not mean our classifier performs well because we are actually interested in getting high scores on protest and planned-protest classes.

For every model we select a number of parameters to optimize. In most cases the parameters we select are numerical attributes (e.g. maximum tree depth) and are varied over a large range. Where appropriate we also varied discrete attributes such as choosing a linear or polynomial kernel for support vector classification. Parameters for advanced custom criteria or memory management were not included in these tests (i.e. left to their default). For both the Decision Tree classifier and Random Forest classifier we choose the Gini impurity as the splitting criteria in all cases. The other criteria supported by these classifiers is information gain (entropy), which has similar results but at increased computational requirements (Raileanu and Stoffel, 2004). The full overview of the optimized parameters and the best combinations can be found in the appendix.

For the final evaluation, the results obtained from the GridSearchCV parameter optimization are used to cross validate the models on the whole dataset. For all of these models we calculated the precision, recall and f1 score by using the resulting predictions of the cross validation processes.

Finally, we make an additional exploration on the effects of the bag of tags features. For that reason, we run tests for several combinations of TFIDF with and without the spaCy bag of tags data for each model that we train.

7 Results

In the final evaluation results of our experiments, we observe that in any combination of feature sets, the Support Vector Classification algorithm outperforms the rest on all for precision, recall, and so, f1-score. While SVC scores around 0.55 and 0.56 f1-scores, the Decision Tree classifier performs around 0.52 and the Random Forest classifier around 0.44 and 0.47.

As can be seen in Table 3 and in Table 2, when we use the TFIDF features without including the bag of tags set, almost all of the performances are getting better. Thus, the idea of adding part-of-speech and named entity recognition tags to improve the results is not holding for this particular task. Although, there is an exception with the Decision Tree classifier which scores higher results when the bag of tags included, the precision of our best performing algorithm, SVC, drops as much as 0.10 points.

Metrics/Algorithms	Random Forest	Decision Tree	Support Vector Classif.
Precision	0.60	0.59	**0.76**
Recall	0.45	0.50	**0.51**
F1 score	0.47	0.52	**0.56**

Table 2: Performance scores of the models trained only on TFIDF features

Metrics/Algorithms	Random Forest	Decision Tree	Support Vector Classif.
Precision	0.52	0.64	**0.69**
Recall	0.42	0.49	**0.52**
F1 score	0.44	0.52	**0.55**

Table 3: Performance scores of the models trained on a combination of TFIDF and bag of tags features

8 Conclusions

In this study, we have explored the performances of several machine learning algorithms with different feature sets on the task of classifying protest-related sentences.

The results have shown us that the best performing algorithm for this task is Support Vector Classification when it is trained on TFIDF features, without including a feature set of part-of-speech and named entity recognition tags. However, it is likely that a bias towards SVC is created since the TfidfVectorizer is optimized only with an SVC classifier. Moreover, the same optimized feature set is used to train all the other algorithms. Thus, the feature set may be created in a way that the SVC uses it well, while the features work better for the others may get lost. It was necessary to separate the optimization processes of the models from the feature extraction due to the exponential increase in the need of computational power.

In a follow up study, we would like to run a single hyper-parameter optimization process to tune everything for each model we compare to prevent such possible biases. Likewise, we would like to understand the insights of the data more extensively by looking at the best performing parameters and discover the reasons behind them. By doing this, we may explore the parameter space of the models in a guided way and fit them better.

We'd also like to further analyze our results to provide better insight in performance. If planned and past protest classes are confused but not non-protest and protest classes we could look into separating the process into finding protest related sentences first and determining past or future separation at a later stage.

In a future study, we plan to experiment with several more algorithms including K-Nearest Neigbours, Multi Layer Perceptron, Neural Networks. Moreover, we plan to explore using word embeddings and sentence embedding as features.

References

[1] Forman, G. (2003) *An Extensive Empirical Study of Feature Selection Metrics for Text Classification.* Journal of Machine Learning Research, 3(Mar), 1289-1305

[2] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). *Bag of Tricks for Efficient Text Classification.* `arXiv:1607.01759 [cs.CL]`

[3] Muthiah, S., Huang, B., Arredondo, J., Mares, D., Getoor, L., Katz, G., & Ramakrishnan, N. (2015, January) *Planned Protest Modeling in News and Social Media* In AAAI (pp. 3920-3927)

[4] Radinsky, K., & Horvitz, E. (2013) *Mining the Web to Predict Future Events.* In Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM'13) 255–264.

[5] Raileanu, L. E., & Stoffel, K. (2004). *Theoretical comparison between the gini index and information gain criteria.* Annals of Mathematics and Artificial Intelligence, 41(1), 77-93.

A Appendix

Min samples split :	**50**, 90, 130, 170
Max depth :	None, 15, 20, 25, **30**
Min samples leaf :	**5**, 7, 9, 11, 13
Max features :	**None**, 'sqrt', 'log2'

Table 4: Optimization options for Decision Tree Classifier with using PoS tag NER tag and also TFIDF Vectors as features. Best parameter combinations are marked as bold.

N-Estimators :	**30**, 70, 100, 150
Max depth :	None, 65, 80, 95, **110**
Min samples leaf :	25, 30, **40**, 45, 50, 100
Max features :	**'sqrt'**, 'log2'
Bootstrap :	True, **False**

Table 5: Optimization options for Random Forest Classifier with using PoS tag NER tag and also TFIDF Vectors as features. Best parameter combinations are marked as bold.

Kernel :	**'linear'**, 'poly', 'rbf', 'sigmoid'
C :	0.025, 0.25, 0.5, 1, 2, **3**

Table 6: Optimization options for Support Vector Classifier with using PoS tag NER tag and also TFIDF Vectors as features. Best parameter combinations are marked as bold.

Min samples split :	**50**, 90, 130, 170
Max depth :	**None**, 15, 20, 25, 30
Min samples leaf :	**5**, 7, 9, 11, 13
Max features :	**None**, 'sqrt', 'log2'

Table 7: Optimization options for Decision Tree Classifier with using TFIDF Vectors as features. Best parameter combinations are marked as bold.

N-Estimators :	**30**, 70, 100, 150
Max depth :	**None**, 65, 80, 95, 110
Min samples leaf :	25, 30, 40, **45**, 50, 100
Max features :	**'sqrt'**, 'log2'
Bootstrap :	True, **False**

Table 8: Optimization options for Random Forest Classifier with using PoS tag NER tag and also TFIDF Vectors as features. Best parameter combinations are marked as bold.

Kernel :	**'linear'**, 'poly', 'rbf', 'sigmoid'
C :	0.025, 0.25, 0.5, 1, 2, **3**

Table 9: Optimization options for Support Vector Classifier with using PoS tag NER tag and also TFIDF Vectors as features. Best parameter combinations are marked as bold.