CEUR-WS.org/Vol-3180/paper-217.pdf

Irony & Stereotype Spreader Detection using Random Forests

Profiling Irony and Stereotype Spreaders on Twitter (IROSTEREO) PAN2022

Tiago Filipe Nunes Ribeiro, Yana Nikolaeva Nikolova and Kaja Seraphina Elisa Hano

Department of Nordic Studies and Linguistics (NorS) - University of Copenhagen, Nørregade 10, 1165 København

Abstract

We present a model for classifying irony and stereotype spreaders on Twitter based on the dataset provided for the PAN2022 task IROSTEREO for this purpose. We take a feature engineering approach focusing on lexical and stylistic features and improve on the character n-gram baseline F1-score by 9% on cross-validation. Of the classification algorithms considered, we find that the Random Forest classifier performs the best, achieving a final F1-score of 96.04% with a 70/30 split on the train set and a final accuracy of 95.56% on the test data provided by PAN.

Keywords

irony detection, classification, random forest, Twitter, PAN, IROSTEREO

1. Introduction

This project aims to solve the task presented at PAN2022[1], Profiling Irony and Stereotype Spreaders on Twitter (IROSTEREO) 2022[1], which consists of classifying authors as irony and stereotype spreaders given a set of English tweets. Because irony is employed "to mean the opposite to what is literally stated", per the task authors, it can be used to scorn or stereotype vulnerable groups in ways that avoid conventional moderation techniques [2]. Identifying irony and stereotype spreaders could help improve the identification of hate speech and cyberbullying for moderation purposes [3, 4] and contribute to the problem of disambiguation in natural language processing [5, 6]. Our code is available on GitHub[2]. The results were submitted via the TIRA platform [7] and the overview paper detailing all the systems used for this task can be found at [8].

2. Related Work

2.1. Irony and sarcasm

The most common definition of verbal irony centers on the incongruity of the literal and intended meaning of an utterance [5, 9] for the purpose of expressing i.a. humor, sophistication,

CLEF 2022 - Conference and Labs of the Evaluation Forum, September 5–8, 2022, Bologna, Italy

tiri@di.ku.dk (T. F. N. Ribeiro); xjv550@alumni.ku.dk (Y. N. Nikolova); ltk953@alumni.ku.dk (K. S. E. Hano)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

[1]https://pan.webis.de/clef22/pan22-web/author-profiling.html

[2]https://github.com/tfnribeiro/zenodo-PAN22-aut-prof-irony-stereotype

group affiliation or retractability [9].

The task authors, as well as some previous research [6], identify sarcasm as a more aggressive and bitter subset of irony, even though the task does not require distinguishing between the two in classification. However, we can assume that irony in conjunction with stereotypes will often have a ridiculing and aggressive attitude towards the stereotyped groups and thus qualify as sarcasm. In this paper, we consider methods and previous work related to both irony and sarcasm detection without distinction.

2.2. Irony and Sarcasm Detection

Despite the complexity of irony and sarcasm as linguistic phenomena, a large variety of methods have been developed to solve classification and disambiguation tasks related to them.

In their survey of sarcasm detection methods, [6] distinguish two main types of methods. Rule-based methods identify indicators of sarcasm, for example hashtag sentiment [10], particular phrases [11] or situation phrases containing words with an incongruous sentiment [6]. Using the latter approach, [12] achieve a high F1-score of 0.90, but their method requires computationally intensive parsing and a more precise distinction between subtypes of irony and sarcasm.

More popular are feature-based approaches using supervised learning algorithms [6]. The literature attests to a wide variety of features. Bag-of-Words n-gram representations are most common for representing lexical information and some degree of context in irony detection [13, 14], although recently probabilistic language models have also been used to this effect [15]. This lexical base is usually supplemented with a variety of stylistic features relating to punctuation, emoji and hashtag use, as well as general stylistic features like type-token ratios and average word lengths [6, 15, 16]. Features related to POS-tags, affective features, and polarity contrast are also common [6, 17, 18]. Word embeddings are the most widespread technique for encoding semantic similarity, particularly in deep-learning approaches, which have been gaining traction in the field [19, 20].

2.3. Challenges

Identifying irony in textual media, and especially in microblog formats like tweets, is a difficult task because the incongruity it exploits is often contextual and paralinguistic in nature [6]. For example, *What lovely weather* may be an ironic utterance in a downpour, but it is not immediately identifiable as such in isolation [21]. In recent years, researchers have tried to incorporate topical and conversational context, in particular when dealing with data from social media or forums, into irony detection to improve classification [6]. This could be an incorporation of previous replies as a feature [18, 22] or identifying topics that are most likely to elicit sarcasm [23].

Another issue is that many markers of irony in spoken language, like pitch, nasalization and other spectral markers [9, 24], are not available in a textual modality. Punctuation, capitalization and emoji usage have been used as textual correlates of such markers [15, 25]. Contextual information, as described above, can also be used to compensate for “missing" irony cues in a text.

2.4. Twitter as Data Source

Twitter is one of the most popular data sources in the field of irony and sarcasm detection [6, 23, 25]. The size of the social media platform, combined with the short format of the posts and its easy-to-use API makes it an obvious choice for this type of research.

The most common and efficient way of annotating tweets for classification tasks is by scraping based on *#sarcasm* and *#irony* hashtags [5, 14] and assuming un-tagged tweets are non-ironic/sarcastic. This method has obvious downsides and tends to produce non-representative datasets that are easier to categorize than manually annotated tweet sets [6]. We assume the data used in the present study are manually annotated, so we might expect lower performance than we see in the literature.

3. Data

The data provided by PAN consists of 420 XML files, where each file is named for the user ID and contains 200 tweets. The tweets are anonymized, so hashtag, user and URL information is replaced by generic tags. Another file contains the author IDs and the ground truth labels – either I (ironic) or NI (non-ironic). The dataset is balanced.

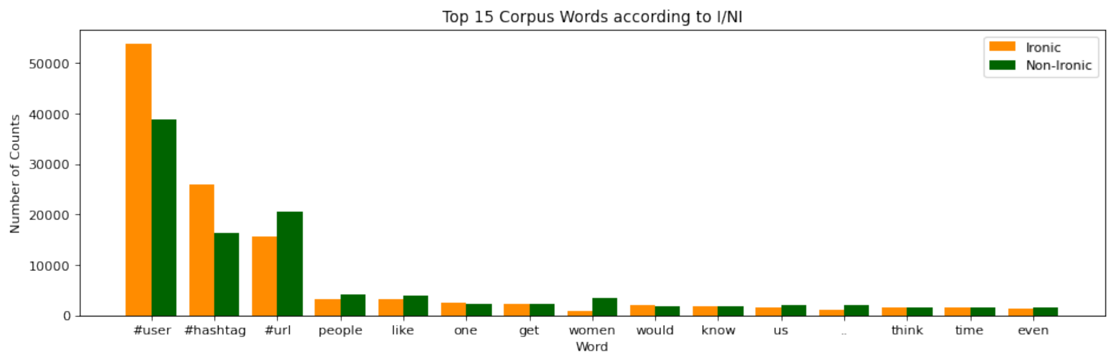

Figure 1: Bar plots for the top 20 most used words in the corpus according to the two classes.

3.1. Challenges of Data

In manual irony annotation of tweets, [15] found that among ironic tweets realized through polarity contrast (71% of all ironic tweets), half were only identifiable as such by hashtag use. This means that the absence of hashtags in the data will likely increase the difficulty of the task (see also [10] on hashtag-based classification).

Another limitation of the dataset is the fact that we have no way of accessing conversational context for reply tweets, which has been shown to be significant for identifying irony [18, 22]. Single tweets and reply tweets are treated the same in the dataset, even though we might expect that irony would function differently for them.

3.2. Data Pre-Processing

Our pre-processing methods are dependent on which features are being generated. However, for most features, we remove the generic hashtag, user and URL tags from the tweets before processing them. These are then tokenized by the *TweetTokenizer* provided by NLTK[3]. This is optimized to capture smileys and long words which are not standard English.

4. Features

We took a feature engineering approach to the task, as such approaches have yielded good results in the past [6, 25] and have the benefit of being more interpretable than end-to-end embedding approaches. We explain how the features were generated and our hypotheses about how they might contribute to distinguishing ironic and non-ironic users.

We divide this section into two sub-sections, the first detailing the features used in the final model as in Fig.2, and the second describing the features we experimented with but decided against using as part of the final feature vector.

4.1. Model Features

4.1.1. Stylistic Features

Our first approach was centered around creating features that represent the profile's style. To do this, we perform simple counts on each author's output, which are then divided by the total number of tweets by the author (200 in this task). We generated the features displayed in Table 1.

The intuition behind these features is to portray some of the main trends between the user-profiles. [25] argue that due to character limit restraints ironic tweets might achieve their goal by creative use of fewer words. We also believe that ironic users might use shorter replies and abbreviations to express irony. Structural features have also been used before in similar tasks [25]. All hashtags and Twitter handles have been replaced by generic markers in our data, but previous studies have found that hashtag use is connected to sarcasm [15], so we believe these counts would still be relevant for this task [6].

Avg. Vocab Size:	Number of unique tokens per tweet
Avg. Token Number:	Number of tokens on average per tweet
Vocab/Token Ratio:	Number of Vocab divided by tokens
Avg. Tweet Length:	Number of characters on average per tweet
Avg. HASHTAG Counts	Number of HASHTAGs on average per tweet
Avg. USERTAG Counts	Number of USERTAGs on average per tweet
Avg. URL Counts	Number of URLs on average per tweet
Avg. Emoji Counts	Number of Emojis on average per tweet
Avg. Capital/Lower Case Ratio	Ratio between upper and lower case per tweet

Table 1
Count features used in the final classifier.

[3]https://www.nltk.org/api/nltk.tokenize.casual.html

4.1.2. LiX Score

LiX is a readability score developed in Sweden by [26] that uses a combination of a word and sentence factors to measure text complexity. LiX is calculated by the following formula:

$$\text{LiX} = (\%N_{LongWords}) + (N_{words}/N_{sentence})$$

We have set the threshold for long words at $t = 7$, following [27], and we count the number of sentences by using a regular expression to find punctuation followed by a space or a new line:

```
[ ]+[.;?!]|[.;?!][ ]+|[.;?!]\n
```

While there might be a risk of underestimating the number of sentences, we obtain the following LiX statistics on our dataset: $\mu = 33.3; \sigma = 6.3; max = 61.6; min = 16.7$. The mean corresponds to a 7th-grade lexical complexity, which seems adequate for a social media with a microblog format.

Our intention with this metric, similar to the stylistic features, is to model that ironic profiles might use less complex vocabulary to reach a wider audience with an easier-to-understand tone.

4.1.3. TF-IDF Unigrams

Words frequencies have been used to model sarcasm [6], and it makes intuitive sense that the use of certain words might be indicative of irony. To model this problem, we implemented the Term-Frequency-Inverted-Document-Frequency (TF-IDF) statistic.

TF-IDF is based on the intuition that tokens which are rare in a corpus might contain more information if they appear in higher counts in a specific document. For this task, we implemented a parameter to allow switching between an author's whole output or a single tweet as a document. For the final feature vector, we used single tweets as documents and our entire training data as a corpus, as we found this improved performance relative to using author documents.

There are different approaches to TF-IDF when it comes to normalizing term frequency (TF). In our implementation, we use the log_2 scale, which means that eventually more counts contribute less and less.

IDF is calculated by first creating an encoding vector, where each position corresponds to a term in the corpus. We count in how many documents of the corpus each term occurs. We then can calculate the *idf* vector:

$$idf = log_2\left(\frac{N}{n_t}\right) \tag{1}$$

where N is the number of documents and n_t is the number of documents where t occurs. Note that we don't smooth out the division, as we only consider terms that are present in the corpus. The term *tf* is calculated by counting the number of terms which are present in the idf vector of size m for each tweet, and this is saved into a matrix of size $documents \times m$. We then want to ensure that terms are scaled to a logarithmic scale, and to do this we perform the following operation on the matrix:

$$\begin{cases} 0, tf = 0 \\ 1 + log_2(tf), tf > 0 \end{cases} \tag{2}$$

We then multiply each of the document vectors by the idf to obtain the final *tf-idf* vector for a specific author. In the case of using tweets as a document, we obtain the average tweet *tf-idf* vector for each author. We also use this methodology for profanity and emojis. For the profanity TF-IDF, we lowercase all the tweets and stem the words to match the same root, while for unigram TF-IDF we simply lowercase and remove any hashtags in the set. As the resulting vector is the size of corpus vocab, ≈ 77162, we decided to reduce the dimensionality using PCA, to capture the main variance in the TF-IDF vectors in the first 20 components.

4.1.4. TF-IDF Profanity

Some of the earliest works on irony detection [28] used the presence of profanity as a feature to identify irony in news headlines, and research has since shown that online communities use profanity in different ways and could be used to distinguish them [29]. Therefore we hypothesized that profanity use might also be different for ironic and non-ironic users, for example in the frequency and types of words used.

We implemented this feature based on a profanity list on GitHub[4] with 2864 words. We created TF-IDF vectors for the data and reduced dimensionality via PCA to 14 principal components. The data was pre-processed with tokenizing and stemming before creating the count vectors, as our profanity list only contained lemmas.

This feature obviously overlaps with TF-IDF unigrams, which also counts profanity along with all other words, but we hoped that isolating this lexical group might help us find patterns that would otherwise be lost in the unigram TF-IDF features.

4.1.5. TF-IDF Emojis

Face with tears of joy, the most popular emoji in our dataset with 1497 counts, was considered the "word of the year" in 2015 by the Oxford dictionary. It's no secret that emojis are used and are now commonplace in social media, and the research community has investigated these to evaluate their usefulness for classifiers or the meanings they encode [30, 31, 32]. [32] argue that some emojis might be redundant as they are just repeating the meaning of the message, while others might be replacing a word or a POS. [30, 31] both show that different communities might develop a particular emoji language to express different ideas through the usage of these pictograms.

As such, we believe that irony and stereotypes might also be expressed by the usage of certain emojis. At first, we encoded this information by having counts of the emoji usage, but in general emoji, tokens are more sparse than words so this didn't yield very good results. As emojis behave similarly to words when they are most meaningful, we decided to encode them using the same TF-IDF methods we used for unigrams. We then perform PCA to capture the variance across users and use this in our final classifier.

[4]https://github.com/zacanger/profane-words

4.1.6. POS Tag Counts

We use POS tag counts for a rough representation of syntactic information and complexity. To obtain POS tags we decided to use the *Universal Part-of-Speech Tagset*, which is described in NLTK's documentation[5]. It consists of 12 different POS tags which are counted across each author, after tokenization and removal of tweeter tags, averaged to obtain the average POS counts for each author. Some parts of speech tags like ADVs and ADJs have been shown to be lexical markers of irony [25]. We decided to drop the tags NOUN, X, and PUNC, as they are already encoded in other features. Nouns tend to correlate strongly with token counts, punctuation is encoded in its own feature as described in 4.1.8 and X, representing words that the model couldn't assign a POS to, are usually few and not relevant for classification.

4.1.7. Sentiment Analysis

Irony can be used to attribute a sentiment value towards a specific target [25], and so we would imagine that this would also be a feature that helps distinguish an ironic user from a non-ironic one. Furthermore, this feature could uncover sentiment polarity, as features based on incongruous sentiment have been used as an irony marker in previous work [17].

For this feature we used the framework *VADER-Sentiment-Analysis*[6], which is a lexicon and rule based sentiment analysis tool which returns 4 features: *positive*, *negative*, *neutral* and *compound* values. The latter is describe as "normalized, weighted composite score" by the authors, as it is a normalized metric between -1 (most negative) to 1 (most positive) across all the words in the lexicon.

Initially, we attempted to model sentiment by creating a metric which would count the number of sentences that have high polarity in sentiment (high values in both positive and negative sentiment). However, this method seem to filter out too much information and the results weren't too promising.

For this reason, we decided to instead calculate sentiment for each tweet and proceed to combine the scores using statistical methods. We attempted both the average vector across all tweets **(M)** as well as the standard deviation **(SD)** for each metric. From our experimentation, the latter seemed to perform better, and we speculate it is due to the fact it captures information about how a user usually tends to deviate from his average expression.

4.1.8. Punctuation

We also included several punctuation features. These are features often used for irony detection as in [11]. We counted each type of common multiple punctuation mark such as **!!!**, **???** and **...** as well as **“"** and single punctuation marks. They were expected to be used differently in ironic versus non-ironic text. We filtered these patterns by using the python regular expression library re [7]. In the end, we averaged each punctuation per tweet for each user.

[5]https://www.nltk.org/book/ch05.html#tab-universal-tagset
[6]https://github.com/cjhutto/vaderSentiment
[7]RegEx

4.2. Unused features

4.2.1. Word Embedding

We also experimented with word embeddings as our model did not have any features that would encode the context or semantic meaning of words. We assume that the semantics of certain words might be important to discern irony, namely when synonyms/antonyms are used to express a certain opinion. We experimented with the word embeddings provided by *Gensim*[8]. These are GloVe embeddings trained on a Twitter corpus and we tested with different embedding sizes ($n \in 25, 50, 100$).

In the model, we average the embeddings for each word in an author's tweet to represent it by its average embedding. We collect all of these representations for each of the author's tweets and then average them to collect the final embedding vector for the author.

We then concatenate this vector with the other features to be used by the classifier. The results we obtained using these embeddings resulted in generally worse performance when performing cross-validation. We also tested these features in isolation, and we could see that they did achieve about 80% accuracy, but when combined with the other features they didn't help the classifiers.

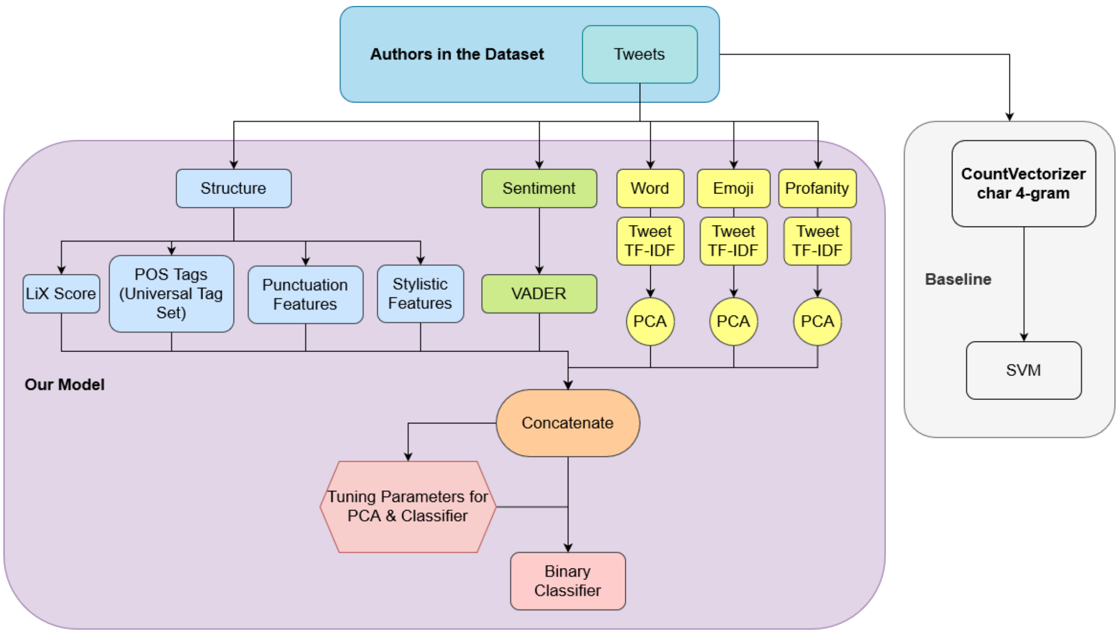

Figure 2: Final Model Used and Baseline Model representation

[8]https://github.com/RaRe-Technologies/gensim-data

4.2.2. Misspelling

We thought that misspelling might be a good indicator of irony, as it can be used intentionally for humoristic effect. By using PySpellchecker [9] we found the misspelled words and divided them by the total number of words. However, it the results greatly overlapped for both classes and it was not a very practical feature for the classification. Because the misspellings did not add significant information, we did not use this feature for our model training. We suspect that misspellings are in general very common on social media.

4.2.3. Syntactic Complexity

Another type of feature we considered was syntactic complexity, which we hypothesized could be higher for non-ironic users.

Syntactic complexity can be estimated in a number of ways including NP density, tree height and the number of high-level constituents per sentence [33]. All of these approaches require parsing, which we did with `spacy`'s Dependency Parser[10]. We tried implementing tree height, but the process proved very slow and expensive, and we dropped the feature so we could keep our model lightweight. We incorporate some syntactic information with POS counts, so it is not totally absent from our model.

5. Methods

5.1. Baseline

In the task webpage [25], the suggested baseline indicated was a char- or word-gram model, classified by an SVM. We followed this recommendation to create our own baseline, by using the *CountVectorizer* class and the `SVC` as implemented by `scikit-learn`. We experimented with different n using cross-validation and found that a 4-gram character model worked best. This will be used in our results as the baseline for this task.

5.2. Experiments

For all our classifiers and different features we followed the methodology of creating a $70 - 30$ split of the training data, where 70% of the data was used for training/cross-validation and the last 30% was used to test the classifiers.

To select different features, we would start by using them in isolation with the different classifiers and see the results on the cross-validation. If the accuracy was $> 70\%$ then the features would be considered to be concatenated to the final feature vector. The features that weren't included often meant that the models would perform the same or worse if they were added.

We also considered different dimensional reduction methods, namely `PCA` and `SparsePCA` for the TF-IDF vectors due to their large dimensions. It is also desirable that the final vector is somewhat interpretable, so we only applied these to the TF-IDF vectors, by far the largest

[9]PySpellchecker
[10]https://spacy.io/usage/linguistic-features#dependency-parse

dimensional features in our model. `PCA` was cheaper and the results were comparable to `SparsePCA` so this was used as our reduction method.

We found the best performing `PCA` dimensions for each of feature by performing a grid-search with cross-validation on the training data and the values found were emoji-PCA$_n = 4$, profanity-PCA$_n = 14$, word-PCA$_n = 20$. These results make intuitive sense, as the dimensions increase in terms of complexity of the features and their importance for the final classification.

In terms of classifiers, we used 4 different methods: **random forest** (`RandomForestClassifier`), **support vector machines (SVM)** (`svm.SVC(gamma="auto")`), **logistic regression** (`LogisticRegression`) and **nearest neighbour (NN)** (`KNeighborsClassifier`) for our task. All classifiers with the exception of random forests are sensitive to highly different scaled values, so we normalize all features using the `StandardScaler`, which subtracts the mean and divides by standard deviation, from `SKLearn`. When doing cross-validation, we always reported the values for each of these classifiers and we could see that each feature tended to impact them in different ways.

The *RandomForestClassifier* is the best performing across all metrics and features. For this reason, we attempted to do parameter tuning for this particular model. We did this by using grid-search but found no parameters which performed better than those provided by `scikit-learn`, so the parameters were left unchanged.

We present results for both the cross-validation and test splits. For the test, we chose the models which performed the best in the cross-validation. The final model we use is displayed in fig. 2. We also report the performance of our best classifier (Random Forests) on the task test set.

6. Results

We find that all classifiers except NN perform better than the baseline in both cross validation and on the test data we separated from the training data.

Tab. 2 shows the average test and train scores based on 7-fold cross validation using 70% of the data provided by PAN.

Our results show that all models, with the exception of 1-NN and 5-NN, outperform the baseline (Tab. 2). The best baseline model (4-Char) performs with an accuracy of 87.24% on the training set and on the test set with an accuracy of **84.35%** and an F1-score of **84.36%**. We see that the random forest classifier obtains the best results. It performs with 100% accuracy on the train data and **91.84%** accuracy on the test data, as well as an F1-Score of **91.83%** during cross validation using standard deviation on the sentiment (SD) across the user's tweets. The scores using the mean sentiment (M) per the user's tweets performs a little lower on the test set but still over 91%. The other models do not perform as well. Logistic regression and SVM outperform the baseline models with over 89% in both accuracy and F1-Score. NN performs the worst of all classifiers with the best, 3-NN, only achieving an accuracy and F1-score of 87.76% and 87.79%, respectively, on the test split. It is the only NN that outperforms the baseline in both accuracy and F1-score. Because of their comparatively worse performance, we left them out of further evaluation. We continued our work on the best baseline and the three best classifiers.

Our models also outperform the best baseline when we trained on 70% of the provided training

Classifier	Train Accuracy (%)	Test Accuracy (%)	F1-Score (%)
(SD) Random Forest	**100**	**91.84**	**91.83**
(SD) Log. Reg.	97.51	89.79	89.83
(SD) SVM	97.22	89.12	89.12
(SD) 1-NN	**100**	81.97	82.01
(SD) 3-NN	92.23	87.76	87.79
(SD) 5-NN	88.83	82.65	85.71
(M) Random Forest	**100**	91.50	91.49
(M) SVM	97.00	88.78	88.78
(M) Log. Reg.	97.78	90.14	90.16
Baseline (2-Char)	85.09	82.65	82.39
Baseline (3-Char)	87.24	83.67	83.60
Baseline (4-Char)	87.24	84.35	84.36
Baseline (5-Char)	88.89	82.65	82.70

Table 2
Cross-validation on 70% of the train data, n splits = 7. In bold, the best performing metrics. (M) denotes that the mean sentiment was used to combine the Tweet Sentiment, while (SD) denotes standard deviation was used.

data and tested on the remaining 30% (Tab. 3). The baseline model performs with an accuracy of **84.92%** and an F1-score of **84.96%**. All classifiers, except NN, outperform the baseline with our features both using sentiment M and SD. The best performing classifier is again random forest using SD for the sentiment with an accuracy and F1-score of **96.03%** and **96.04%** respectively. The same classifier using mean sentiment is close behind. Both logistic regression and SVM achieve accuracies and F1-scores above 90%. Random forest is the best performing classifier in all cases.

The results for the **PAN task's test dataset was an accuracy of 95.56%** , using the Random Forest and averaging the sentiments in tweets with VADER [(M) Random Forest]. These results are similar to the performance of the same classifier on our 70/30 split of the training data (acc: 95.24%, see tab. 3).

It seems that the performance on the 30% test split is a good indicator for performance on the official test set.

6.1. Feature Importance

We also discovered that not all features have the same importance, and that there are differences between classifiers. To investigate this we used `permutation_importance` from `sklearn.inspection` [11] to calculate the importance of each feature. This method calculates the influence each feature has by comparing the original accuracy score with the score obtained if different features are shuffled. The figures resulting from this analysis can be found in the Appendix.

For random forest `avg_user_hashtag_count` and `auth_vocabsize` seem to be the most

[11]permutation importance

Classifier	Accuracy (%)	F1-Score (%)
(SD) Random Forest	**96.03**	**96.04**
(SD) Log. Reg.	90.47	90.49
(SD) SVM	92.06	92.08
(M) Random Forest	95.24	95.25
(M) Log. Reg.	90.47	90.50
(M) SVM	92.06	92.08
Baseline (4-Char)	84.92	84.96

Table 3
Test results on the remaining 30% of the data. The baseline model is a 4-gram char BOW model followed by a SVM for classification. In bold, the best performing metrics.

important, while the other features have noticeably less influence, see Fig. 4. Both logistic regression and SVM have the highest importance for `word_pca11`, `auth_vocabsize` and `qu` (average number of question marks), Figs. 6 and 7. The feature importance for 1-NN is most equally distributed among features. The most important features are qu (single question marks), `LixScore`, `word_pca11` and `word_pca20` (see Fig. 5) but the other features are not far behind. We see that different features are more important than others for different classifiers but there is some overlap between classifiers. Random forest concentrates only on a few features while the other classifiers take more features into account.

7. Discussion

7.1. Model Performance

Our results show that the classifiers SVM, logistic regression and especially random forest perform very well with our features. Of the NN-classifiers, 3-NN performs best, but still lags a couple of percentage points behind the better-performing models. Our highest F1-score with the random forest is 96.04%, an F1-score that according to [6]'s survey of the field is only topped by [34], who obtain a score of 97.5% with a distributional semantics approach (on individual tweets).

Such high results are rather surprising considering the relative simplicity of the features used in our model. Especially the lack of semantic similarity or incongruity features (although perhaps sentiment could be interpreted as such), which are otherwise widely used [5, 17, 18, 19], makes this a remarkable result.

However, considering that a simple char-gram model paired with SVM is able to obtain an F1-score of 84.36%, we suspect that this dataset is simply easier to classify than previous datasets in similar tasks. This might have to do with how this dataset was annotated and how the Twitter users were sampled. Unfortunately, the task authors do not provide us with this information, so it's unclear what makes this dataset unusual. Nevertheless, we do know that some ways of collecting Twitter data, for example, based on #irony hashtags, can create datasets that are easier to classify [6], so data collection and sampling methods likely play a role here as well.

Unlike most previous research that focuses on tweet-level irony detection [6], we are classifying irony on the user level. This is significant, as previous research [3] shows that the majority of hate speech is produced by a small minority of people. It might therefore be more effective to identify hateful/sarcastic users than individual utterances. It might also be the case that identifying irony on the user level (both in terms of annotation and prediction) is more robust than doing so for individual tweets and that this is partly responsible for the high F1-score.

7.2. Feature Performance

While developing our different features we found that even the stylistic count features on their own performed similarly (about 80% accuracy) to the Baseline model, despite being quite primitive. This means that simply by looking at the general stylistic characteristics of an author a machine learning model is able to separate the two classes. This indicates that the difference between ironic and non-ironic tweeting styles really is very distinct on the user level. Every other feature yielded small improvements, with TF-IDF being the highest contributor.

Based on feature permutation (see Appendix), we found that vocabulary size, hashtag counts and question marks were among the most significant features, as well as some TF-IDF PCA vectors, i.e. stylistic and lexical features. This is consistent with previous studies [18, 35] where lexical features (especially unigrams) are used as a baseline and core of more complex models. Studies that find strong effects of punctuation and similar stylistic features include [36] and [15]. Per Tab. 2, using the sentiment standard deviation rather than the mean also generally improved results by a small amount. The standard deviation might be better than the mean at capturing incongruities in polarity, which are often indicative of irony. By comparing the difference in the standardized mean we see that some features are higher for the irony spreader and their counterparts. Non-ironic users (NI) especially seem to have a higher `LixScore` and `auth_vocabsize` while irony spreaders have a high number of qu, Fig. 3. Longer words and tweets are also associated with non-ironic users. In general, the pattern points to non-ironic users using more complex and diverse language than ironic users. On the other hadn, ironic users use more question marks, hashtags and user tags – this points to ironic users possibly being more interactive in their Twitter usage.

We were also curious about the significant PCA features, which are a bit harder to interpret. While looking into the values of different words in the TF-IDF, we noticed that sometimes words specific to a single user, as for instance a promotion code, will have very high values in the TF-IDF vector, as they fit the criteria of being used by only a single user multiple times. For instance, in `word_pca_10` when trained on 70% of the data, we can softmax the weights on the component and sort them according to their scores, to see what contributes to this component. The following word features score highest: “women", “:", “and", “men", “calm", “gay", “guard", “code", “USER-PROMO-CODE". As we can see, some words here might be related with irony and stereotypes, like “gay" and “women", but the “USER-PROMO-CODE" is not related at all to the task and might add noise. To ensure a better quality of these features, we could have opted for stemming and removing stop words or words that only appear in a single user (to capture cases like promotional posts), while accepting that we might lose some information.

Due to the nature of the task as an author classification task, we cannot say if a particular tweet is ironic. Instead, we are only able to classify by all their tweets, but not exactly point out where irony or stereotypes are present. However given the annotations for the dataset, one

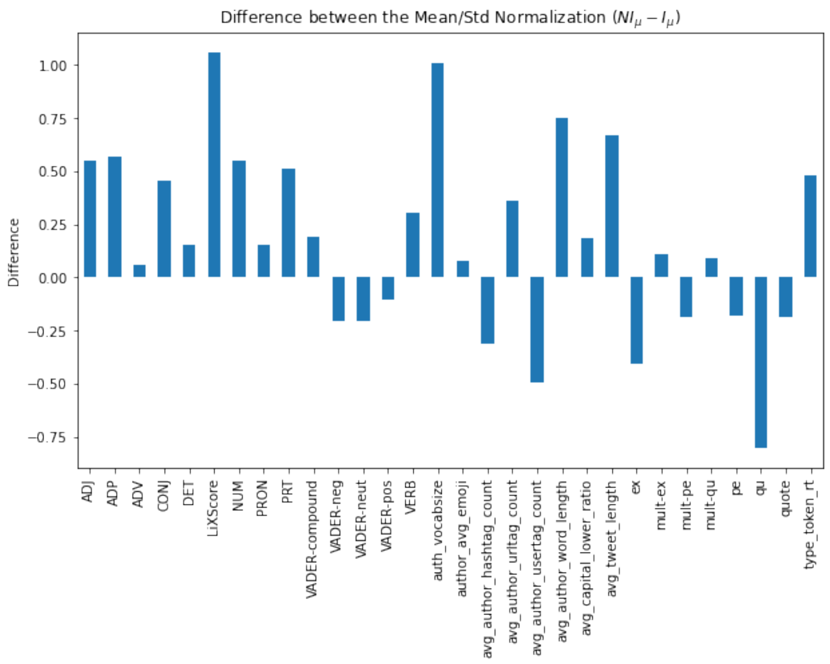

Figure 3: Differences between the Mean/Std normalization mean values of the NI and I classes.

would have to identify specific cases of irony/stereotypes manually to allow for this type of classification. It is also not clear how many ironic tweets make a user an irony-spreader or not. Therefore is unclear how the system would perform on smaller or larger datasets, when the annotation is not as clear or generated in other ways.

7.3. Limitations

Our model has several limitations and areas for improvement. A general drawback is the time it takes to generate the features for the training and test data, particularly for the different TF-IDF features. This means the model would not scale well in terms of time and space usage. To avoid this scalability problem the number of terms for TF-IDF could partially be reduced by clipping the vocabulary or stemming the words in each tweet and accepting some information loss in exchange for a faster performing system. Removing the profanity and emoji TF-IDF features is another strategy. When optimizing for the number of PCA dimensions, we did find that these features added information to the model and slightly improved performance, but the difference is very small (about 1% accuracy was gained by optimizing), indicating that there probably is a large degree of redundancy.

Currently, our model does not consider context. Our features at most capture a unigram language model by the usage of TF-IDF, but even then it's distorted by PCA. While we attempted embeddings, these were not beneficial to the task. This is a bit surprising considering the success others have had with word embedding-based features in conventional ML [19]. Word embeddings in isolation might not capture enough information and would need to be processed in some way before being used in the model. They would also likely be more useful with a deep-learning approach, which is their more conventional application [6, 20].

That being said, we believe our model would require features that encode semantic and pragmatic information, incongruities or even ontologies to interpret if some words are being used out of the expected context.

Another direction for future work relates to the fact that the different classifiers appear to rely on different features (see Appendix). Considering this, an ensemble method could be used to combine different classifiers to optimally use more of the features. Different models could also be optimized to for different feature subsets to find the best combination.

8. Conclusion

We tackled the challenge of detecting irony and stereotype spreaders by focusing on creating stylistic and lexical features. We compared the performance of our features with the suggested baseline for the task, a 4-char gram model followed by an SVM, with our own features. Our features combined with a random forest classifier outperform the baseline by $\approx 9\%$ on cross validation and $\approx 11\%$ on a 30% split of the training data. We also obtained 95.56% accuracy on the task's test set, by using a random forest classifier with our features. We see room for improvement in identifying passages for irony and sarcasm usage and developing features that focus on encoding context and semantic relationship between words.

References

[1] J. Bevendorff, B. Chulvi, E. Fersini, A. Heini, M. Kestemont, K. Kredens, M. Mayerl, R. Ortega-Bueno, P. Pezik, M. Potthast, F. Rangel, P. Rosso, E. Stamatatos, B. Stein, M. Wiegmann, M. Wolska, E. Zangerle, Overview of PAN 2022: Authorship Verification, Profiling Irony and Stereotype Spreaders, and Style Change Detection, in: A. Barron-Cedeno, G. D. S. Martino, F. S. Mirko Degli Esposti, C. Macdonald, G. Pasi, A. Hanbury, M. Potthast, G. Faggioli, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Thirteenth International Conference of the CLEF Association (CLEF 2022), volume 13390 of *Lecture Notes in Computer Science*, Springer, 2022.

[2] V. S. Greene, "deplorable" satire: Alt-right memes, white genocide tweets, and redpilling normies, Studies in American Humor 5 (2019) 31–69.

[3] C. Van Hee, G. Jacobs, C. Emmery, B. Desmet, E. Lefever, B. Verhoeven, G. De Pauw, W. Daelemans, V. Hoste, Automatic detection of cyberbullying in social media text, PloS one 13 (2018) e0203794.

[4] Z. Waseem, D. Hovy, Hateful symbols or hateful people? predictive features for hate speech detection on twitter, in: Proceedings of the NAACL student research workshop, 2016, pp. 88–93.

[5] A. Reyes, P. Rosso, D. Buscaldi, From humor recognition to irony detection: The figurative language of social media, Data & Knowledge Engineering 74 (2012) 1–12.

[6] A. Joshi, P. Bhattacharyya, M. J. Carman, Automatic sarcasm detection: A survey, ACM Comput. Surv. 50 (2017). URL: https://doi.org/10.1145/3124420. doi:10.1145/3124420.

[7] M. Potthast, T. Gollub, M. Wiegmann, B. Stein, TIRA Integrated Research Architecture, in: N. Ferro, C. Peters (Eds.), Information Retrieval Evaluation in a Changing World, The Information Retrieval Series, Springer, Berlin Heidelberg New York, 2019. doi:10.1007/978-3-030-22948-1_5.
[8] O.-B. Reynier, C. Berta, R. Francisco, R. Paolo, F. Elisabetta, Profiling Irony and Stereotype Spreaders on Twitter (IROSTEREO) at PAN 2022, in: CLEF 2022 Labs and Workshops, Notebook Papers, CEUR-WS.org, 2022.
[9] S. Attardo, Irony markers and functions: Towards a goal-oriented theory of irony and its processing, 2000.
[10] D. G. Maynard, M. A. Greenwood, Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis, in: Lrec 2014 proceedings, ELRA, 2014.
[11] L. A. de Freitas, A. A. Vanin, D. N. Hogetop, M. N. Bochernitsan, R. Vieira, Pathways for irony detection in tweets, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, 2014, pp. 628–633.
[12] S. K. Bharti, K. S. Babu, S. K. Jena, Parsing-based sarcasm sentiment recognition in twitter data, in: 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, 2015, pp. 1373–1380.
[13] A. Reyes, P. Rosso, Making objective decisions from subjective data: Detecting irony in customer reviews, Decision support systems 53 (2012) 754–760.
[14] C. Liebrecht, F. Kunneman, A. van Den Bosch, The perfect solution for detecting sarcasm in tweets# not (2013).
[15] C. Van Hee, E. Lefever, V. Hoste, Exploring the fine-grained analysis and automatic detection of irony on twitter, Language Resources and Evaluation 52 (2018) 707–731.
[16] D. Davidov, O. Tsur, A. Rappoport, Semi-supervised recognition of sarcasm in twitter and amazon, in: Proceedings of the fourteenth conference on computational natural language learning, 2010, pp. 107–116.
[17] E. Riloff, A. Qadir, P. Surve, L. De Silva, N. Gilbert, R. Huang, Sarcasm as contrast between a positive sentiment and negative situation, in: Proceedings of the 2013 conference on empirical methods in natural language processing, 2013, pp. 704–714.
[18] A. Joshi, V. Sharma, P. Bhattacharyya, Harnessing context incongruity for sarcasm detection, 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), 2015, pp. 757–762.
[19] A. Joshi, V. Tripathi, K. Patel, P. Bhattacharyya, M. Carman, Are word embedding-based features useful for sarcasm detection?, arXiv preprint arXiv:1610.00883 (2016).
[20] S. Zhang, X. Zhang, J. Chan, P. Rosso, Irony detection via sentiment-based transfer learning, Information Processing & Management 56 (2019) 1633–1644.
[21] S. Østergaard, The definition and processing of irony 40 (2014).
[22] B. C. Wallace, E. Charniak, et al., Sparse, contextually informed models for irony detection: Exploiting user communities, entities and sentiment, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2015, pp. 1035–1044.
[23] Z. Wang, Z. Wu, R. Wang, Y. Ren, Twitter sarcasm detection exploiting a context-based model, in: international conference on web information systems engineering, Springer,

2015, pp. 77–91.

[24] J. Tepperman, D. Traum, S. Narayanan, "yeah right": Sarcasm recognition for spoken dialogue systems, in: Ninth international conference on spoken language processing, 2006.

[25] D. I. H. Farías, V. Patti, P. Rosso, Irony detection in twitter: The role of affective content, ACM Trans. Internet Technol. 16 (2016). URL: https://doi.org/10.1145/2930663. doi:10.1145/2930663.

[26] C. H. Björnsson, Lix och läsbarhetsprövade skolböcker, Pedagogiskt utvecklingsarbete vid Stockholms skolor ; 2-12., Pedagogiskt centrum i Stockholm, 1968. Publication Title: Lix och läsbarhetsprövade skolböcker.

[27] J. Anderson, Lix and rix: Variations on a little-known readability index, Journal of Reading 26 (1983) 490–496.

[28] C. Burfoot, T. Baldwin, Automatic satire detection: Are you having a laugh?, in: Proceedings of the ACL-IJCNLP 2009 conference short papers, 2009, pp. 161–164.

[29] W. Wang, L. Chen, K. Thirunarayan, A. P. Sheth, Cursing in english on twitter, in: Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing, 2014, pp. 415–425.

[30] M. Kejriwal, Q. Wang, H. Li, L. Wang, An empirical study of emoji usage on twitter in linguistic and national contexts 24 (????) 100149. URL: https://www.sciencedirect.com/science/article/pii/S2468696421000318. doi:https://doi.org/10.1016/j.osnem.2021.100149.

[31] L. Hagen, M. Falling, O. Lisnichenko, A. A. Elmadany, P. Mehta, M. Abdul-Mageed, J. Costakis, T. E. Keller, Emoji use in twitter white nationalism communication, in: Conference Companion Publication of the 2019 on Computer Supported Cooperative Work and Social Computing, CSCW '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 201–205. URL: https://doi.org/10.1145/3311957.3359495. doi:10.1145/3311957.3359495.

[32] G. Donato, P. Paggio, Investigating redundancy in emoji use: Study on a Twitter based corpus, in: Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, Association for Computational Linguistics, Copenhagen, Denmark, 2017, pp. 118–126. URL: https://aclanthology.org/W17-5216. doi:10.18653/v1/W17-5216.

[33] S. K. Barnwal, U. S. Tiwary, Using psycholinguistic features for the classification of comprehenders from summary speech transcripts, in: International Conference on Intelligent Human Computer Interaction, Springer, 2017, pp. 122–136.

[34] D. Ghosh, W. Guo, S. Muresan, Sarcastic or not: Word embeddings to predict the literal or sarcastic meaning of words, in: proceedings of the 2015 conference on empirical methods in natural language processing, 2015, pp. 1003–1012.

[35] R. González-Ibánez, S. Muresan, N. Wacholder, Identifying sarcasm in twitter: a closer look, in: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 2011, pp. 581–586.

[36] M. Bouazizi, T. Ohtsuki, Sarcasm detection in twitter:" all your products are incredibly amazing!!!"-are they really?, in: 2015 IEEE Global Communications Conference (GLOBECOM), IEEE, 2015, pp. 1–6.

Appendix

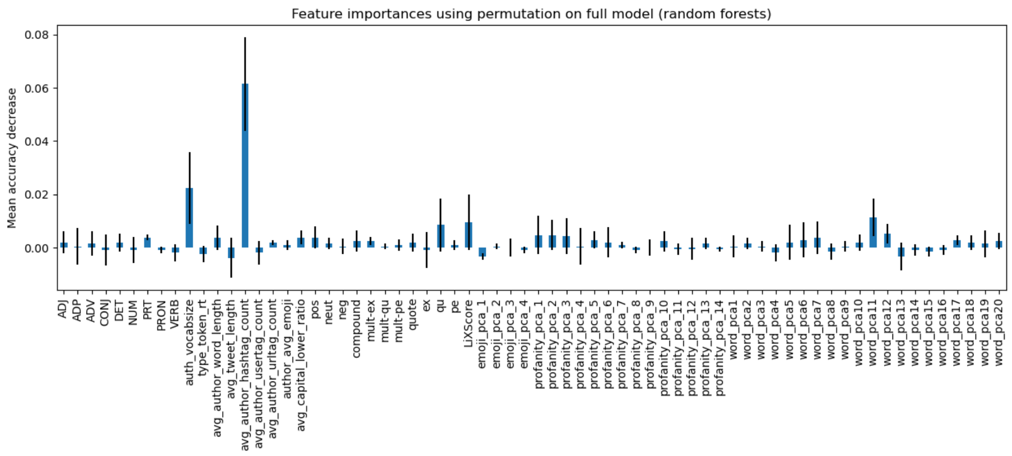

Figure 4: Feature permutation for random forest regression

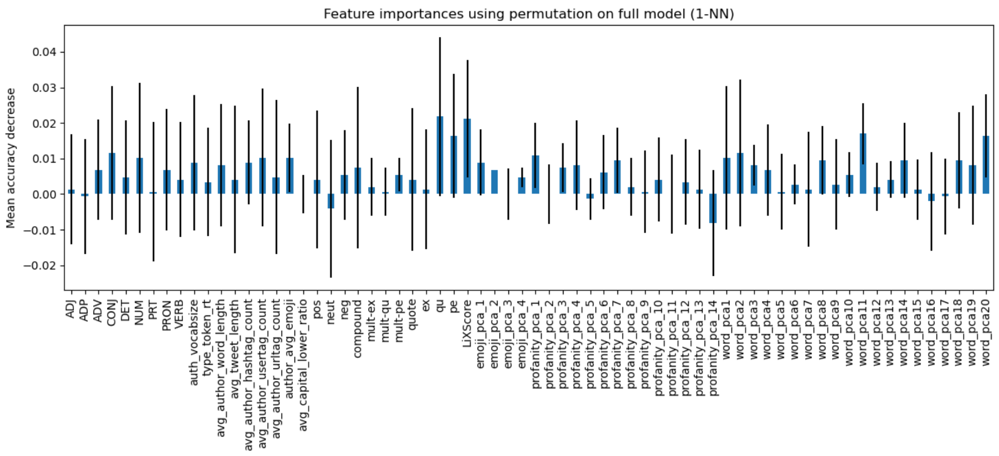

Figure 5: Feature permutation for 1-NN

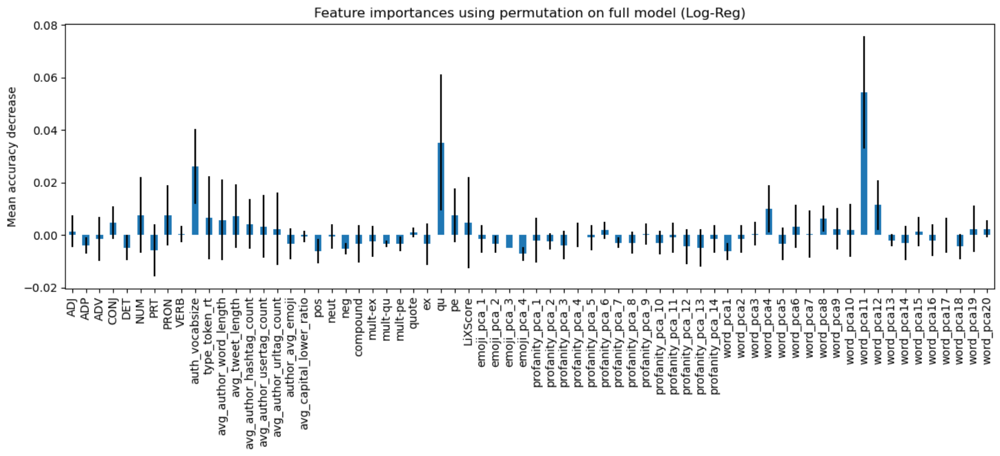

Figure 6: Feature permutation for Logistic Regression

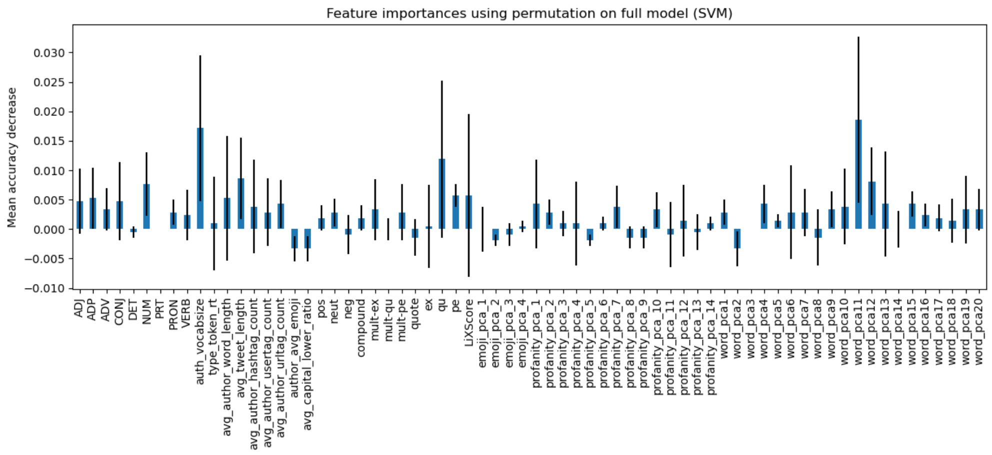

Figure 7: Feature permutatoin for SVM