Twitter Sentiment Analysis

Oliver Price - University of Warwick - CS918, Natural Language Processing

I. Introduction

This report outlines the approach and results from developing a sentiment classifier for a data set of tweets from the social media app *Twitter*. I start by discussing the data preprocessing steps and the logic behind choosing them in Section 2. In Section 3 I discuss the features and classifiers used in the classification task and present the macro-averaged F1-scores of them on the development set. Section 4 demonstrates the hyperparameter tuning conducted on the development set and the resultant scores. Section 5 is used to present the the results of applying the classifier to the test sets, and Section 6 discusses error analysis from these results.

II. Data Pre-processing

Given the nature of *Twitter* documents (tweets) - i.e. social media text - it is particularly necessary to conduct diligent pre-processing in order to bring the text as close to comprehensive text as possible. In this section I take the reader through each pre-processing step and the logic behind it. This is mostly done using regular expressions, and the code for these steps can be found in the attached preprocess.py file.

A. Removing URLs

Websites are usually not inherently positive or negative in nature, so the words in website URLs are unlikely to indicate any sentiment.

B. Remove mentions

In *Twitter*, mentions are when a user mentions another user in a tweet. The only time this might indicate whether a tweet is positive or negative is in the case of real-time tweet analysis. For example, there might be a positive or negative event surrounding a specific user who is frequently mentioned. In this task it is unlikely to be useful.

C. Remove hashtag symbols

The contents of a hashtag could provide some useful information in the classifier, so we remove just the hashtag symbol itself but keep the text. However, hashtags often use concatenated words, so this could prove to not be useful. I found no impact on the results from excluding this pre-processing step, so I kept it. This could be extended by applying a MaxMatching algorithm to the hashtags to split concatenated words, much like that used when doing text processing on Chinese.

D. Replace emojis

For this step I identified the most common emojis indicative of positive or negative tweets, and replace the positive ones with 'HAPPYEMOJI' and the negative ones with 'SADEMOJI'. This proves particularly relevant within the use of lexicons as features. The emojis are:

- Happy: :) (: :-) (-: :D :-D ;D ;-D ;) ;-) (; (-;
- Sad: :( ): :-( )-: ;(;-( );)-;

E. Removing punctuation

Most punctuation cannot indicate sentiment. It can be argued that some does, such as exclamation marks. However, I argue that exclamation marks are likely to only be useful in classifying the degree of the sentiment of a tweet (for example, sentiment on a scale of 0 - 5), because a tweet could be classified as 'negative', and if a string of concatenated exclamation marks is present, it might be 'very negative'. However, in this case a string of concatenated exclamation marks could just as easily indicate positivity as negativity. This is also why all words were lower-cased. (Completely capitalised words can be considered in the exact same way as multiple exclamation marks.)

F. Elongated words

It is important to identify and clean elongated words to reduce the number of words in the total corpus vocabulary, and hence improve the classifier. I identify all cases of 3 or more occurrences of a letter in a word and reduce it to 2. For example, "coooool" goes to "cool" and "screammmm" goes to "screamm". This means that elongated words in tweets can only cause at most 2 repetitions of the same word.

G. Stop words

Stop words can be useful in some applications of classifiers, so it is important to investigate whether or not they might be useful before removing them. I applied a simple logistic regression and SVM model on unigrams to quickly identify whether stop words have any clear impact. Figures 1 and 2 show that the macro-averaged F1-score on each

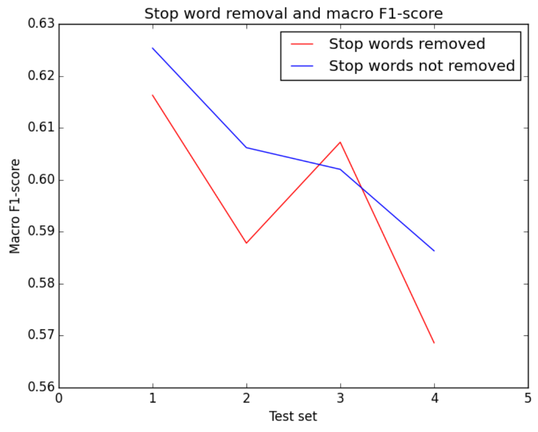


Fig. 1. The effect of removing stop words on support logistic regression with unigrams.

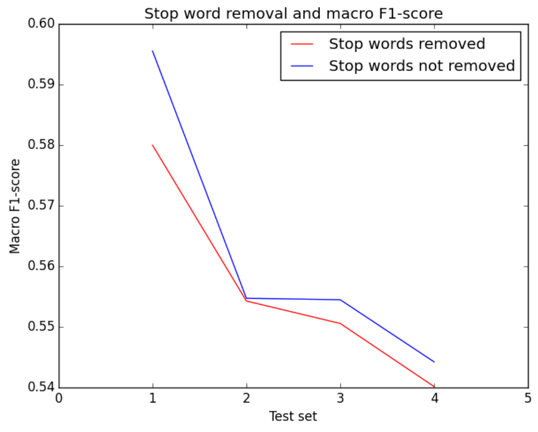
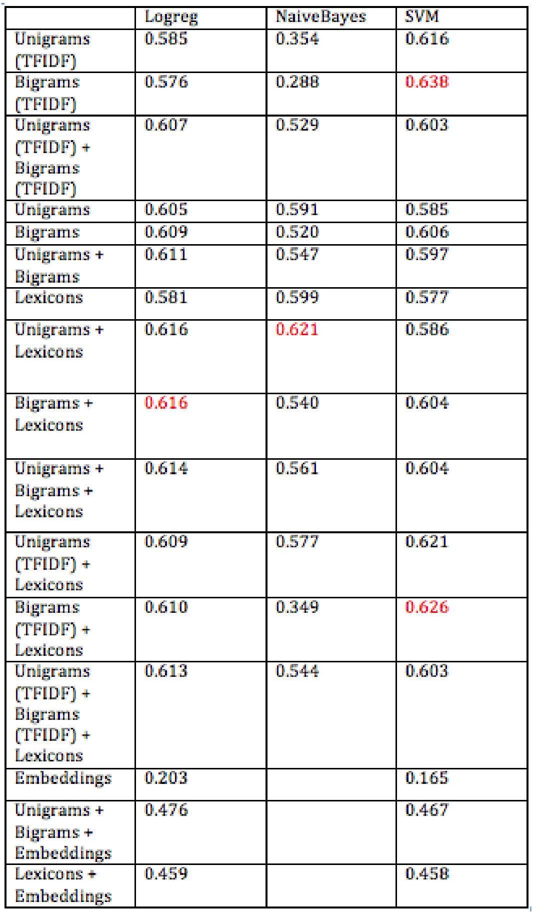


Fig. 2. The effect of removing stop words on support vector machines with unigrams.

test set is usually higher with stop words present. The difference is marginal, but it makes sense to keep them.

III. Features and Classifiers

Throughout this project I explored combinations of various different features and various different classifiers. The features explored were unigrams, bigrams, lexicons and word embeddings, and the classifiers were a baseline majority classifier, logistic regression, support vector machines and naive Bayes. In this section I choose 3 classifiers (as requested in the problem description), including the best-performing one, and discuss the results of each in detail. Figure 3 shows the macro-averaged F1-score (herein referred to as F1-score for brevity) on the development set for various combinations of features and classifiers prior to hyperparameter tuning. This was done in order to identify the inherently strong classifiers to use for tuning.



	Logreg	NaiveBayes	SVM
Unigrams (TFIDF)	0.585	0.354	0.616
Bigrams (TFIDF)	0.576	0.288	0.638
Unigrams (TFIDF) + Bigrams (TFIDF)	0.607	0.529	0.603
Unigrams	0.605	0.591	0.585
Bigrams	0.609	0.520	0.606
Unigrams + Bigrams	0.611	0.547	0.597
Lexicons	0.581	0.599	0.577
Unigrams + Lexicons	0.616	0.621	0.586
Bigrams + Lexicons	0.616	0.540	0.604
Unigrams + Bigrams + Lexicons	0.614	0.561	0.604
Unigrams (TFIDF) + Lexicons	0.609	0.577	0.621
Bigrams (TFIDF) + Lexicons	0.610	0.349	0.626
Unigrams (TFIDF) + Bigrams (TFIDF) + Lexicons	0.613	0.544	0.603
Embeddings	0.203		0.165
Unigrams + Bigrams + Embeddings	0.476		0.467
Lexicons + Embeddings	0.459		0.458

Fig. 3. Macro-averaged F1-scores from trying various combinations of features on each of logistic regression (logreg), naive Bayes, and SVMs. The classifiers highlighted in red are identified as the strongest, and were used in hyperparameter tuning.

As a starting point, I note that the majority classifier is successful with an accuracy of 45.95%, so it is imperative that any classifier that receives extended attention achieves a better result than this.

A. *Unigrams and Bigrams*

This is one of the most logical set of features to start with, since all aspects of any text are inherently embedded within unigrams, otherwise the existence of words would be meaningless. Within sentiment analysis, it is important to make considerations for negations, otherwise a classifier might frequently identify a positive word as indicative of a positive text, when in fact a negation is present and it is a negative text.

Involving unigrams and bigrams, the best F1-score comes from using TF-IDF-weighted bigrams on SVMs and is 0.638. Logistic regression combining bigrams with lexicons (see below for more on lexicons) and unigrams with lexicons achieves 0.616, and unigrams combined with lexicons also performs well using naive Bayes, achieving an F1-score of 0.621. (These can be seen in Figure 3.)

B. Embeddings

Word embeddings are vector representations of words, and there exist a number of open-source word embeddings that have been trained on large corpuses. In this project, I make use of pre-trained GloVe (Global Vectors) word embeddings[1]. It is generally considered that the Google pre-trained word2vec embeddings have more depth, but the word embeddings found in the footnote link have been pre-trained on Twitter data, and are likely to be more useful in this study.

Interestingly, the use of embeddings as features performed very poorly on their own, and caused otherwise good classifiers to perform poorly. One of the better results involving embeddings - but still poor - is combining unigrams, bigrams and embeddings. Logistic regression gets an F1-score of 0.476 and SVMs get 0.467. (These can be seen in Figure 3.)

C. Lexicons

It is natural to assume that positive and negative lexicons are likely to be one of the most useful features in sentiment analysis - positive and negative words provide direct indications of whether or not there is positive or negative sentiment. It is also natural to assume that when combined with bigrams, this can be a very powerful classifier since negations can also be somewhat factored into account.

In this study, I make use of *Scattertext* [1] to identify the words most indicative of positive and negative sentiment in the training set, and use these words to predict sentiment on test sets. *Scattertext* makes use of scaled F-scores. Scaled F-scores work by first taking the precision of a word with respect to a category (e.g. the precision of *beautiful* with respect to 'positive' is likely to be high), and the frequency of that word in the category, before calculating the F-score as

$$F(\text{prec}, \text{freq}) = (1 + \beta^2)\frac{\text{prec} \times \text{freq}}{\beta^2 \text{prec} + \text{freq}}.$$

The normal cumulative distribution function is used to scale and standardise the precisions and frequencies such that $\text{prec}, \text{freq} \in [0, 1]$. The scaled F-score is then calculated by taking the harmonic mean of these new precisions and frequencies. *Scattertext* has a useful visualisation tool to see the lexicons, which can be seen in Figure 4.

Indeed, combining the trained lexicons with a variety of n-grams leads to the majority of the best F1-scores for every classifier, and given the relatively small data set size this is a very good result. Instead of using unigram and bigram counts, the best results are achieved using TF-IDF weightings. SVMs achieves the best lexicon-based F1-scores of 0.626 by combining TF-IDF bigrams with lexicons, and then naive Bayes with 0.621 by combining unigrams with lexicons. Logistic regression follows closely behind with 0.616 on both unigrams combined with lexicons and bigrams combined with lexicons.

[1]https://nlp.stanford.edu/projects/glove/

D. Ensembles

Given that 3 different classifiers are used in this study, it is logical to consider the use of an ensemble of these classifiers to boost the F1-score. However, for each set of features an observation of confusion matrices across the different classifiers shows that there is no single classifier that performs oppositely to other classifiers. (Two classifiers would be opposite if one classifier got true positive results more on class A, and the other got true positive results more on class B.) Thus, it is unlikely that an ensemble would boost the performance.

IV. Hyperparameter Tuning

Hyperparameter tuning was conducted on the development set to improve on the results shown in Figure 3. This was done using a simple grid search across various hyperparameters for the highlighted classifiers, and in this section I show the new development set F1-scores.

The logistic regression on bigram and lexicon features is improved from 0.616 to 0.624 by considering a regularisation parameter of 5. This improvement occurs for both the sklearn liblinear solver and the newton-cg solver, but liblinear has about half the run-time.

The naive Bayes classifier on unigram and lexicon features improves from an F1-score of 0.621 to 0.630 by learning class prior probabilities and using Lidstone smoothing with $\alpha = 0.4$.

The F1-score for the SVM classifier on TF-IDF-weighted bigrams is improved marginally from 0.638 to 0.639 by using a regularisation parameter of 2 and squared hinge loss, but this is an irrelevant improvement.

So in summary, the three best performing algorithms and the ones tested on the test sets and presented in the code are:

- Logistic regression using bigrams and positive and negative lexicons, with a regularisation parameter of 5.
- Naive Bayes using unigrams and positive and negative lexicons, with learning of class prior

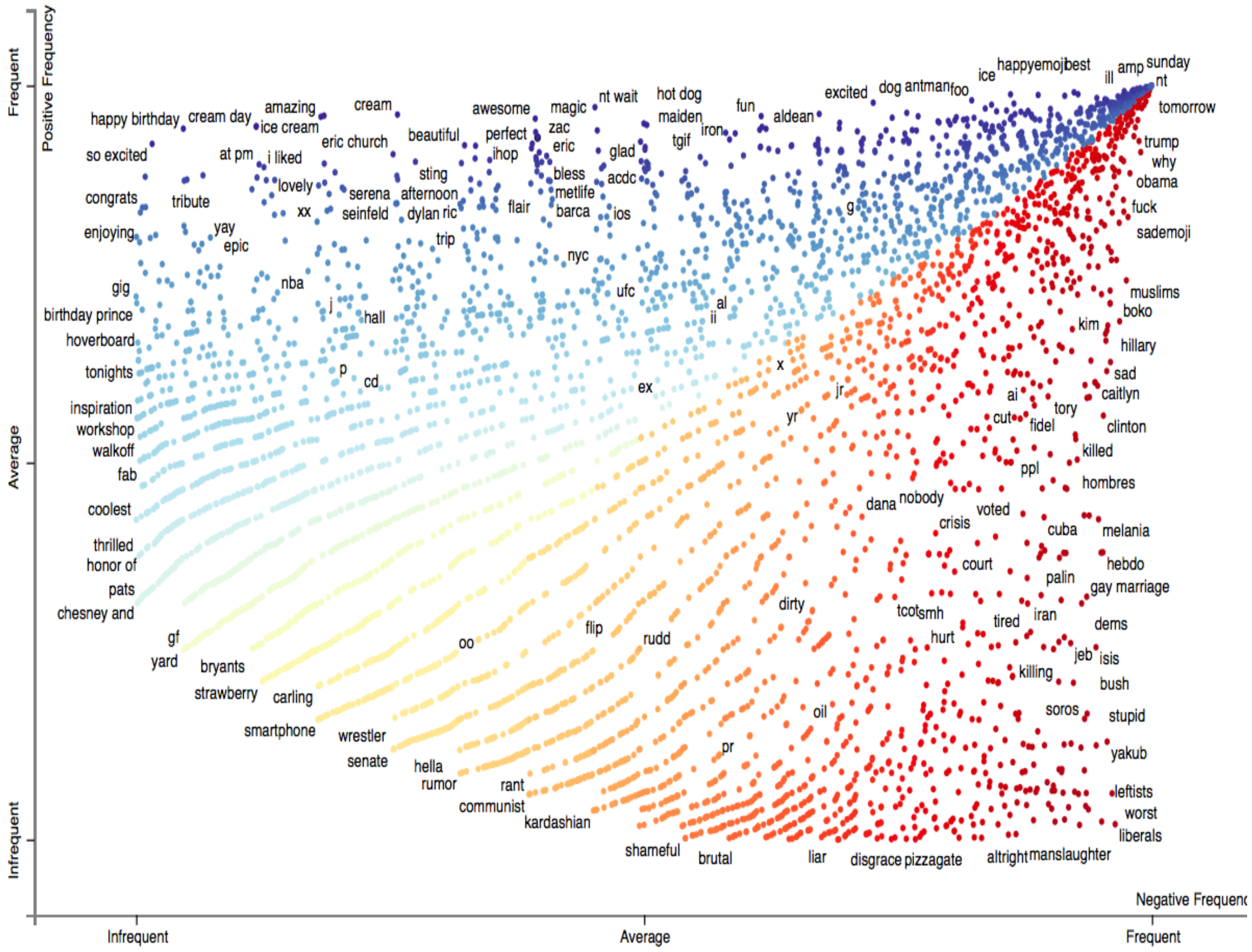


Fig. 4. A visualisation showing the words most likely to indicate positive or negative tweets. The closer to the top-left, the most 'positive' a word is, and the closer to the bottom right, the more 'negative' a word is. Interestingly, the words "liberals" and "leftists" indicate the most negativity, which indicates that there might be a glut of tweets from Donald Trump fans in the data set...

probabilities and a Lidstone smoothing parameter of 0.4.

- SVMs using TF-IDF bigrams. A regularisation parameter of 2 may or may not be used.

From here I refer to these as logistic regression, naive Bayes and SVM, respectively.

V. Test Set Results

In this section I present the macro-averaged F1-scores of performing the three classifiers above on the three test sets. They are:

1) **Test set 1**
   - Logistic regression: 0.593
   - Naive Bayes: 0.548
   - SVM: 0.595
2) **Test set 2**
   - Logistic regression: 0.593
   - Naive Bayes: 0.538
   - SVM: 0.600
3) **Test set 3**
   - Logistic regression: 0.546
   - Naive Bayes: 0.543
   - SVM: 0.560

From this it can be seen that SVMs with TF-IDF bigrams is the best performing classifier on this data set.

VI. Error Analysis

I focus the error analysis on the SVM classifier.

A. *Positive classified as negative*

Two aspects of this misclassification appear a number of times. The first is tweets that end on a positive note but start on a negative one. An example of this are the following positive tweets that were labelled as negative:

- *Beverly Hills, that's where I want to be! **But** not on Jan 8th because I want to be in Lap. D!*

- *If the Lakers still had Jordan Farmar,Trevor Ariza,Shannon Brown I'd be watching them ..I dont like the Lakers* ***but*** *they were entertaining*

Both of the above tweets are inherently positive, but the positivity is despite something negative. In future work this could be remedied by designing a feature that analyses the text before and after any occurrence of the word 'but', allowing an algorithm to learn that a tweet of the form "<NEGATIVE> but <POSITIVE>" is likely to be positive.

The second is tweets that are spoken with a positive attitude, but are saying something that could be construed as negative. This might be considered as 'banter'. Consider the following example:

- *I am the Remus Lupin of werebears. Just watch out at the next full moon. I may once again attempt to* ***steal*** *your honey...*

This tweet is a positive and seems like a light-hearted joke of some sort, but the presence of negative words such as 'steal' might be contributing to the negative classification. This sort of tweet appears frequently in these misclassifications.

B. Negative classified as positive

Amongst this set of misclassifications, there are no obvious trends in the tweets. However, it can be seen that positive lexicons can frequently appear in negative tweets (words such as *Friday* and *tomorrow*), whereas negative lexicons don't appear in positive tweets as much, because negative lexicons are generally more meaningful. Hence, it is likely that these misclassifications occur due to the appearence of positive lexicons in negative tweets that weren't necessarily intended as positive.

In order to remedy this in future work, it is necessary to look more deeply into positive and negative lexicons as features, perhaps developing a model that provides weights to certain lexicons that can be fed into the classifier.

REFERENCES

[1] Jason S. Kessler. Scattertext: a browser-based tool for visualizing how corpora differ. 2017.