COMP3222 Machine Learning Technologies Coursework Report

Harry Nelson
hjn2g19

January 2022

1 Introduction and Data Analysis

When events happen, such as natural disasters or terrorist attacks, people often flock to social media such as Twitter to find out about what happened in a fast and digestible way, without reading through and verifying sources. People often take advantage of this and attempt to misuse the format of tweets being short and easily found/spread and attempt to deceive by posting ”fake” tweets. This can be done by manipulating, synthesising or re-posting multimedia and sharing it as if it was related to a real event. We are tasked with designing a machine learning algorithm to detect these ”fake” tweets using the provided datasets (specifically, to sort a test dataset of tweets into ”real” and ”fake” classes).

The dataset contains ∼15,000 tweets focused around different events. It holds the tweet ID, the tweet text, the image ID, the username string and user ID, a timestamp, and a ground truth label for the tweet which is either ”real”, ”fake” or ”humor”, in a tab-separated-values format. In terms of bias, the split between real/fake/humorous tweets is about 1/3 each, and the events covered are mostly about Hurricane Sandy, with around ∼12,000 of them being focused on that, with the remaining sets of tweets covering events such as the Boston Bombings or a chemical plant explosion. The tweets are also in a wide range of languages, with most (∼80%) being in English, with a further ∼10% being in Spanish, and the rest a mixture of other languages.

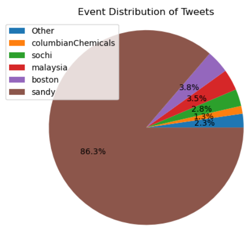


Figure 1: Event Distribution

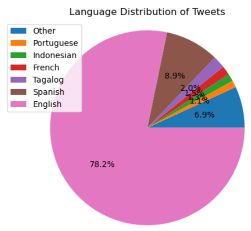


Figure 2: Language Distribution

The text has some issues, as there are escaped HTML characters left in tweets. In some cases the URL at the end of the tweet has been incorrectly parsed, with spaces and backslashes left in as well, such as ”http \/\/t.co\/5JBtF54cmg”. Finally, as there are un-escaped/un-paired quotation marks some readers will read multiple lines of the dataset as one entry without proper configuring. The text quality is also subject to spelling mistakes, slang, shortened words etc. as is common with users posting to the internet, and many of the tweets are ”retweets”, identified either by ”RT” being present or some text being surrounded by quotation marks.

For computation speed, as this classifier doesn’t need to work in real time we are quite unrestricted for computation time. It would be ideal to keep it low in order to make evaluation easier given the time constraints of the coursework, but there is no set limit. Personally I would like to aim for a given run of the classifier to take less than 5 minutes based on previous experience with the python libraries I plan to use (e.g. sklearn). It is reported that previously f1-scores of over 0.9 were obtained, so while the highest accuracy possible would be ideal I think aiming for 0.85 or higher is a good goal.

For legal and ethical characteristics of the data, Twitter prevents some kinds of profiling and sharing larger than 50,000 posts, and as the tweets are public we can use without author consent and without further approval as long as we don’t work with personal identifying information within the posts. As this is a university coursework that has already been ethically approved and we are working purely with the tweet content in order to determine how truthful it is I will not need to seek further approval to work with this dataset to create this classifier.

2 Algorithm Design

2.1 Pre-processing

The data is read in from the dataset into a pandas DataFrame, ignoring quotation marks and using tab as a delimiter. All humor labels are changed to fake labels, as I was unable to find a good way to take advantage of the alternate labels.

The text is made lower case, then the t.co URLs at the end of every tweet are removed using the regex expression 'https* [\s]*(\\\/\\/—\/\/)t.co(\\\/—\/)[a-zA-Z0-9]*'. Incorrectly parsed html characters are parsed, such as &amp; to &, and punctuation is removed by removing characters in the `string.punctuation` set. The tweet is split at spaces and the words are un-contracted using the `contractions` library, then any words that are in either the spanish or the english stopwords set from the `nltk` library were removed. These processed tweets are then saved into a new column in the dataframe.

Choices for pre-processing were based on knowledge from the lectures, visually inspecting the dataset to try and spot any issues, and the methods being mentioned in sklearn documentation or the literature detailed in the evaluation section. Different combinations of pre-processing techniques were then tested with the various algorithms tried to get the methods resulting in the best score.

Only stopwords from English and Spanish were removed, as I decided that translating all the tweets to English would likely lower the quality as well as severely increase the runtime of the pre-processing due to ratelimiting with the Google Translate api, and 90% of the tweets were either English or Spanish so removing the stopwords from those resulted in quite a high impact to accuracy without altering the data beforehand or an increase in runtime. Upon manual inspection several of the other languages identified by the `langdetect` library were inaccuracies also, and as I didn't want to go through and manually label each language/remove stopwords for many languages that may not even be present I ended up only removing them for English and Spanish. I also didn't correct spelling, somewhat due to the large amount of non-english tweets and my lack of confidence in correcting spellings in other languages, but also because some things like emphasis or slang could be a feature worth keeping to compare separate to the word it is originally.

2.2 Features

The tweets are put through the sklearn `TfidfVectorizer()` with the non-default parameters:

- use_idf: True
- norm: l2
- ngram_range: (1, 2)
- max_df: 0.75

These parameters were selected based on tuning detailed in the evaluation section, with the starting points based on the default values. `TfidfVectorizer()` was chosen as it was one of the feature extraction methods provided by sklearn [1] that was listed as commonly used for bag-of-words feature approaches, as it improves the accuracy over the `CountVectorizer()` as it weights features in the document based on their frequency multiplied by their inverse-document frequency, meaning features that appear in 100% of cases are less weighted compared to features that appear more sparingly. This was then selected over features like manual feature vectors based on present visual features, part-of-speech tagging or `CountVectorizer()` on its own based on performance comparisons detailed in the evaluation section.

2.3 Algorithm

The algorithm used was the sklearn `KNeighborsClassifier()`, with the non-default parameters:

- weights: uniform
- n_neighbors: 9
- n_jobs: -1

These parameters were also selected based on the tuning detailed in the evaluation section, with the starting points based on the default values. This algorithm was initially selected based on it being a commonly used algorithm for text classification problems [2] [3], and was chosen out of the classifiers detailed in the evaluation section due to the highest consistent performance.

3 Evaluation

I initially implemented some pre-processing from when I was analysing the dataset, such as stop word removal for the two most common languages, English and Spanish, substitution emojis and urls and removing punctuation. The processed tweets were then passed into a `TfidfVectorizer()` from the sklearn module to create a feature vector. Initial algorithm selection was based mostly on indications of commonly used algorithms, such as use in a paper by Pratma et al. in 2015 [3], selecting Multinomial Naive Bayes, Support Vector Machine and K-Nearest Neighbours algorithms. I also modified a script from this sklearn tutorial page [4] to quickly compare many of their classifiers with different combinations of pre-processing and features. A `MultinomialNB()` classifier achieved an f1 score of 0.79 with the default parameters for the `TfidfVectorizer()` and the classifier, so I decided that for this comparison I would use that as a baseline for performance when selecting other algorithms to try. Filtering the micro-f1 scores above 0.8 from this test, these were the best performing algorithms with no tuning of the 14 tried:

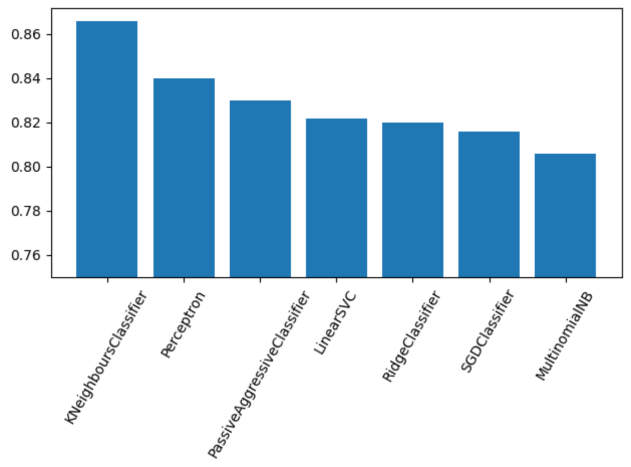


Figure 3: Accuracy of Classifiers

I decided to experiment with the top few classifiers from this test to make a voting ensemble to boost the f1 score more. I ran each with different combinations of hyperparameters to see what parameters resulted in the highest f1 score. I ended up with a K-Nearest Neighbours classifier scoring 0.87, a Perceptron Classifier scoring 0.87, a Multinomial Naive Bayes classifier scoring 0.83, a Passive Aggressive Classifier scoring 0.83, and a LinearSVC classifier scoring 0.83, with the following hyperparameters being tuned:

- TfidfVectorizer parameters tried:
  - use_idf: True and False
  - norm: l1 and l2
  - ngram_range: (1, 1), (1, 2), (1, 3)
  - max_df: 0.25, 0.5, 0.75, 1
- K-Nearest Neighbours parameters tried:
  - weights: uniform and distance
  - n_neighbours: 2, 4, 6, 8, 10, 12
- Multinomial Naive Bayes parameters tried:
  - alpha: 0.01, 0.001, 0.0001, 0.0001
- Perceptron parameters tried:
  - penalty: l1, l2 and elasticnet
  - alpha: 0.01, 0.001, 0.0001, 0.0001

  - max_iter: 50, 100, 200

- Passive Aggressive Classifier parameters tried:
  - C: 0.1, 0.5, 1, 2
  - max_iter: 50, 100, 200
  - fit_intercept: True and False

- LinearSVC parameters tried:
  - C: 0.1, 0.5, 1, 2
  - penalty: l1 and l2
  - fit_intercept: True and False
  - max_iter: 50, 100, 200

These were pooled into a custom voting ensemble that would pre-process the test data being passed to each classifier according to what had performed the best previously. Sadly, the micro-f1 score of this voting classifier ended up being 0.84, with quite a low precision for real posts (0.74) and a high precision for fake posts (0.91). This potentially means that all of the classifiers were slightly more biased towards voting for fake posts which makes sense given the bias in the dataset (1/3 real to 2/3 fake). I decided to instead choose the highest performing algorithms (KNN and Perceptron) and make more adjustments to those to try and increase their performance, as well as re-evaluate the feature sets.

I went back to try and find some new features to try. A paper by Sriram et al. in 2010 discussing alternate feature extraction methods to Bag-Of-Words features proposes an approach creating a small feature vector with several boolean identifiers to help sort tweets into multiple classes, with the identifiers including elements such as whether the tweet mentions another user, or whether the tweet contains an opinion [5]. I decided to analyse the dataset further to try and find any patterns that might prove useful for a similar approach.

I tried making features of the parts of speech of the tweets, adding the part of speech to the end of each word in the string to tag them and then turning that into a feature vector using the `TfidfVectorizer()`, however that lowered the f1 score of the Perceptron, KNN and MNB algorithms, so I discarded it. I could not spot any particular features common in one class of tweet and not the other. I tried making a feature vector of text features such as length of words, number of hashtags, number of mentions, whether it was a retweet or not, sentiment of the tweet, number of capitals etc. and feeding that into algorithms. With purely those features as a feature vector KNN got an f1 score of 0.57, which was surprisingly high given they were relatively weak features. I then tried adding these features to the `TfidfVectorizer()` processed tweets which lowered the f1 score to around 0.65 with KNN, then reducing the number of features added to things like hashtags, mentions, and sentiment, which also got around 0.65 with KNN. I tried both of these configurations with the Perceptron too but the extra features caused every tweet in the test set to be classified as ”real”, leading to an f1 score of less than 0.2.

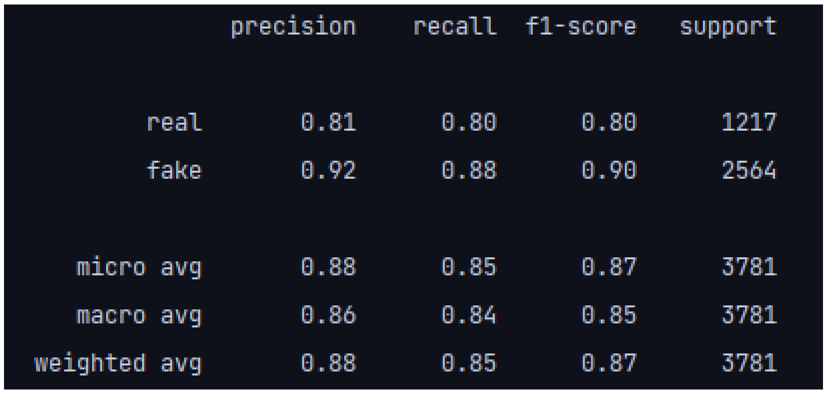
I tried a few more pre-processing techniques like using the `emoji` package to turn the emotes into a text representation of the specific emoji rather than just an identifier saying ”emoji”, as well as lemmatising the words, but both ended up reducing the f1-score by 0.01 rather than helping. I also tried running the KNN algorithm with no pre-processing as I realised I hadn't tried that but it also got a sub-0.8 f1 score. As my hyperparameter tuning was somewhat brief when I was performing it on many algorithms, I decided to do another run on the KNN algorithm with more parameter options.

- TfidfVectorizer parameters tried:
  - use_idf: True and False
  - norm: l1 and l2
  - ngram_range: (1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)
  - max_df: 0.7, 0.75, 0.8, 1
- K-Nearest Neighbours parameters tried:
  - n_neighbours: 6, 7, 8, 9, 10
  - p: 1, 2

This didn't lead to much change though, as the only parameter that changed was 9 neighbours becoming the best rather than 8 and the difference in f1 score less than 0.01, and it still rounded to 0.87. I took a look at other papers and websites using KNN algorithms and they reported similar accuracies on similar corpora, around 0.7 to 0.8 [6] [7].

Code used to evaluate different algorithms and the implementation of the voting classifier has been left in `evaluation_testing.py` and `ensemble_voter.py`.

4 Conclusion



```
              precision    recall  f1-score   support


        real       0.81      0.80      0.80      1217
        fake       0.92      0.88      0.90      2564


   micro avg       0.88      0.85      0.87      3781
   macro avg       0.86      0.84      0.85      3781
weighted avg       0.88      0.85      0.87      3781
```

Figure 4: Final Metric Classification Report

The final algorithm design ended with a micro-f1 score of 0.87, accurately identifying 92% of fake tweets in the test dataset and 81% of real tweets. The parameters and algorithm used were the best performing out of several different algorithms which were tested, filtered, tuned, tested and filtered again, with more tuning done on the KNN algorithm after it was selected as the best performing too.

Upon reflection I believe I could have done more research to find a better use for the humor label, as due to poor time management I was unable to find anything concrete towards the start and so pressed on with just the binary set of labels. I could also potentially have tried some more complicated feature sets, or libraries that weren't sklearn, however I went with sklearn due to familiarity from the module's labs. I think I did reasonably well at finding relevant literature, however there were several papers that looked largely cited I was unable to get access to due to the paywall or the server providing it being down, and there are undoubtedly many more papers I didn't find that contained some relevant information. I also think I wasn't able to analyse the dataset as well as I was hoping I would, as there were not many characteristics I was able to identify per class that could have helped me get a better score. One paper referenced found quite a lot of success with only 8 binary features per tweet for sorting into multiple categories, so I think if there were similar features identified for this assignment it might give better performance.

My implementation of a voting ensemble could probably have been improved by segmenting the dataset more and selecting algorithms based on their ability per-class then weighting them, rather than just having them all be equally weighted, as most algorithms I tested guessed fake more often than real. I could also have researched more effective ways to deal with the bias of the dataset being more fake than real, which could potentially improve its ability to detect real tweets. I was surprised at the accuracy of the Multinomial Naive Bayes implementation, achieving an f1 score of around 0.79 on the test and the validation sets without any pre-processing at all, only with the `TfidfVectorizer()`, however I was unable to increase it much further with any pre-processing or parameter tuning which resulted in me selecting the KNN algorithm.

References

[1] scikit-learn developers. Feature extraction, 2021. https://scikit-learn.org/stable/modules/feature_extraction.html#common-vectorizer-usage (accessed: 6.1.2022).

[2] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura E. Barnes, and Donald E. Brown. Text classification algorithms: A survey. *CoRR*, abs/1904.08067, 2019.

[3] Bayu Yudha Pratama and Riyanarto Sarno. Personality classification based on twitter text using naive bayes, knn and svm. In *2015 International Conference on Data and Software Engineering (ICoDSE)*, pages 170–174, 2015.

[4] scikit-learn developers. Classification of text documents using sparse features, 2021. https://scikit-learn.org/stable/auto_examples/text/plot_document_classification_20newsgroups.html (accessed: 6.1.2022).

[5] Bharath Sriram, Dave Fuhry, Engin Demir, Hakan Ferhatosmanoglu, and Murat Demirbas. Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, page 841–842, New York, NY, USA, 2010. Association for Computing Machinery.

[6] Harshiv Patel. Text classification using k nearest neighbors (knn), 2021. https://iq.opengenus.org/text-classification-using-k-nearest-neighbors/ (accessed: 13.1.2022).

[7] Khushbu Khamar. Short text classification using knn based on distance function. In *International Journal of Advanced Research in Computer and Communication Engineering*, 2013.