COMP3222 Machine Learning Technologies

Coursework:

MediaEval 2015 'Verifying Multimedia Use' Task

Edward Gorman

eg6g17@soton.ac.uk

Introduction and Data analysis

Problem Characterization

The MediaEval 2015 "verifying multimedia use" problem involves classifying posts from social media websites (e.g. blogs, Twitter, Facebook) as ‘real’ or ‘fake’ posts by designing and building a classification algorithm. This problem is aimed to help journalists identify legitimate sources of information to fulfil the goals of journalism that impose a strict code of faithfulness to reality and objectivity.

After a high impact event it is vital to obtain factual information about the situation to help first responders or inform the public. There is likely to be an abundance of unverified sources making claims about the event and it is the journalist’s responsibility to analyse and decide which multimedia represents real information. This task is therefore to help journalists identify legitimate sources of information.

The definition of a fake post is the following:

- Reposting of real multimedia, such as real photos from the past, re-posted as being associated to a current event
- Digitally manipulated multimedia
- Synthetic multimedia, such as artworks or snapshots presented as real imagery

The problem requires a binary classifier algorithm for the classes real and fake, which are the ground truth labels provided in the dataset; this could be extended to a multi-class algorithm to include the humour class, however humour can be treated as a fake label. The algorithm will be used as a backend component and the task has not specified a visual representation or output for the results.

The area that is critical for the machine learning algorithm to improve is the classification of posts so that the journalists are exposed to a greater number of real posts. It should be developed in Python using libraries including sklearn, nltk and tensorflow, and there is no pre-existing data pipeline it must adapt to. The performance of the model will be assessed in terms of its f1-score.

Data Characterization

Origin: multimedia posts from social media sites(e.g. blogs, Twitter, Facebook), including metadata about the post (multimedia content has been removed as this is not being assessed). Each entry in the dataset is labelled with the ground truth, identifying the post as ‘real’, ‘fake’ or ‘humor’.

Format: text file, tab separated with 7 different fields, encoded in UTF-8:

- tweetId — unique value identifying tweet — (578854927457349632)
- tweetText — message associated with tweet — (kereeen RT @Shyman33: Eclipse…)
- userId — user who posted the tweet — (70824972)
- imageId(s) — unique id(s) of image(s) — (eclipse_01)
- username — name of the user — (peay_s)
- timestamp — time when the tweet was sent — (Fri Mar 20 09:45:43 +0000 2015)
- label — classifies as real/fake/humour — (fake)

Volume: there is approximately 3.5MB of data from 11 different news events. The training set contains 14483 entries of which 34.6% are real, 47.2% are fake and 18.3% are humour. In the testing set there are 3781 entries of which 32.2% are real, 57.8% are fake and 0% are humour.

Quality: the data for the most part is suitable requiring only moderate data processing, including removing duplicate rows and fixing some timestamp formatting issues. The tweetText text field is restricted to characters in the UTF-8 set (since the overall file is UTF-8) but may contain English/ foreign language or a mix of both as well as spelling errors. There is also punctuation to be removed, although emojis may be of use. Most entries also contain hashtags which are useful for identifying the subject of the tweet, as well as one or more URLs at the end.

Bias: the data was sourced from social media sites meaning that the users who made these posts are likely to be in the age range of 18-49 years old. There are also likely to be reposts since users will copy and paste information from others regardless of whether they think the story is true or false.

Restraints: we cannot extend analysis beyond the dataset itself but since the posts are public we can analyse them without author consent and use generic data sources including lists of stop words or common names in our models.

Algorithm design

For my model I decided to create an ensemble voting classifier using three trained models that I found to be adept at classifying posts. I used pandas dataframes to store/manipulate the data and sklearn's classification algorithms to implement the machine learning algorithms.

Data Pre-Processing

The data was loaded using pandas 'read_csv' method, which is a standard for reading any delimited file which in our case was tab delimited. One row in the training dataset was dropped because it was a duplicate entry, but the test set contained no duplicates.

The tweetText field required clean up before it could be used for some feature extraction methods; this included: converting text to lowercase, removing URLs, removing punctuation and tokenizing the remaining text. I considered removing non-english languages, but I achieved a high f1-score while still leaving them in so decided it would take too long to process and was unnecessary.

The timestamp field contained some formatting issues for certain entries, namely they sometimes contained a space after a colon (e.g. ': '), which was fixed using the pandas 'apply' method and a simple regular expression to remove such spaces in the timestamp entries.

I chose to treat the humour label as a fake label because after experimentation with my TFIDF analysis the end F1 score performed approximately 0.11 worse when humour entries were removed from the dataset.

Feature Extraction

For each data field I have detailed what features I have considered/visualised/used in the models.

tweetId:

> Could be used in the social media API to acquire additional data about the post but this was not allowed due to the problem constraints.

tweetText:

> The length of tweetText (no text cleaning applied) was one indicator used in the final model. The idea with this feature was that a real or fake post may differ in length because the event

they are reacting to may cause them to be briefer or more long-winded. For the real posts they had an average length of 93 characters whereas the fake posts were on average slightly shorter at 86 characters.

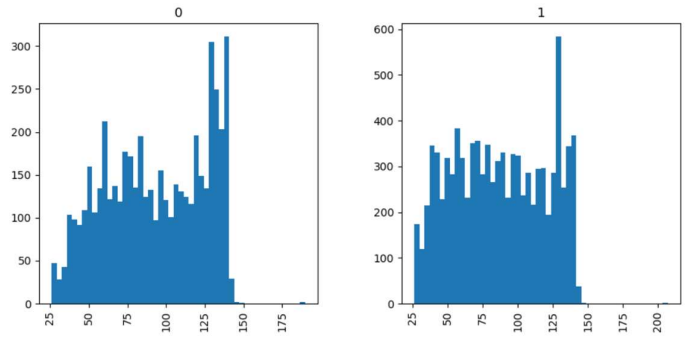


Figure 1: Histogram for length of text (0=real, 1=fake)

Statistical features of tweetText were considered after reviewing [3] including the number of retweets, hashtags, urls and mentions but this was not explored further. I have mentioned in my conclusion this is an area which could have predictive power similar to text length.

Bag of Words is another method for producing a vector representation of the words used in a post, which may be a useful feature because the words people use in a real vs fake event may be different. This was attempted using the cleaned version of tweetText and counting the term frequency for the top 300 words. When visualised it produced no discernible clusters and when combined with a model did not achieve useful performance.

Term-Frequency Inverse Document Frequency was another feature used in the final model. This measures the frequency of a word in a document relative to the rest of the corpus of documents, identifying key words for a given class. For this I extracted the top 4000 features from the training dataset using sklearn's TfidfVectorizer, which was then used to extract the TFIDF scores for the training and testing sets.

Parts of speech tagging was also attempted but had little success. I measured the frequency distribution for the tagged version of the tokenized tweetText including nouns, pronouns, verbs, adverbs and adjectives. When combined with a model did not achieve any useful performance and there was little statistical significance between the classes.

Sentiment of the text was another feature extracted from text which indicates whether the text was positive or negative based upon the words used. Using textblob you can extract the polarity of the text (-1 being negative and +1 being positive) as well as the subjectivity (0 being objective and 1 being subjective). After testing there is little statistical difference for these two measures of sentiment, making it a weak feature representation.

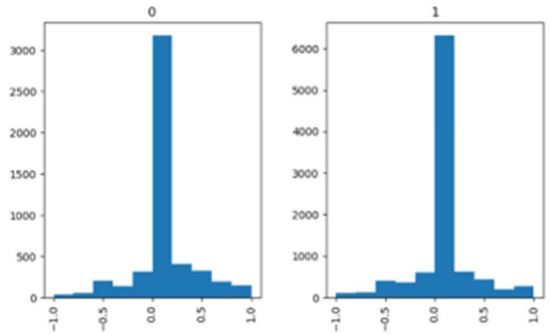


Figure 2: Histogram for text polarity (0=real, 1=fake)

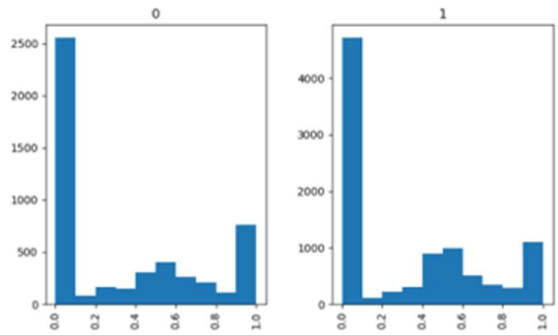


Figure 3: Histogram for text subjectivity (0=real, 1=fake)

userId:

Could be used in the social media API to acquire additional data about the user but this is not allowed due to the problem constraints.

imageId(s):

Could be used in the social media API to acquire additional data about the image but this is not allowed due to the problem constraints. Most data entries only contain 1 image with the max being 5, but there isn't a large enough statistical difference to differentiate the classes.

username:

This wasn't investigated but two potential features could be searching the username to see if it contains a real name or known news organisation which may suggest it is a more reliable source and more likely to be true.

timestamp:

This was a particularly useful feature for differentiating real and fake posts. Intuitively it makes sense that a fake post is unlikely to be sent when people are normally sleeping or working vs a real post which could happen at any time. After parsing the timestamp field into a datetime object I extracted the hour of day the post was sent, as well as the day of the week. This proved a useful feature as is described in the model selection phase.

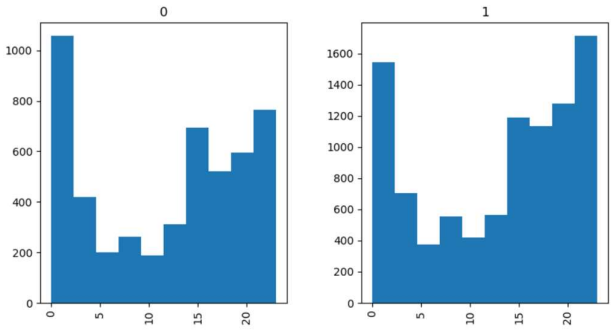


Figure 2: Histogram for time of day (0=real, 1=fake)

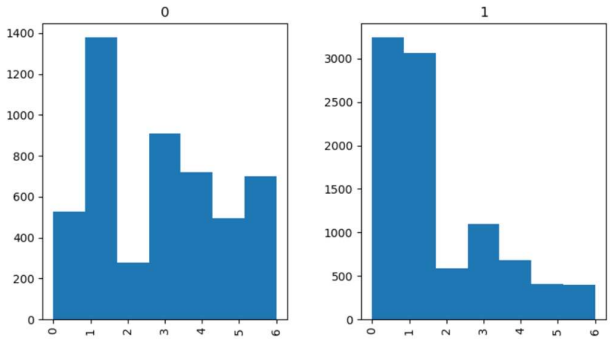


Figure 3: Histogram for day of week (0=real, 1=fake)

Model Selection

For each feature representation of the data I investigated several models and refined the hyperparameters of the most successful to achieve the highest F1 scores. I attempted combining all the features mentioned into one model but often found that it decreased the overall performance of the model. Consequently, I settled for an ensemble of the most successful methods afterwards.

I created a testing harness for the following classification algorithms to find the best algorithm for the given feature representation: Naïve Bayes, Support Vector Machines, Decision Trees, Random Forest, K Nearest Neighbours, AdaBoost and Neural Networks. This list of algorithms was obtained after reading [1] which discusses several of these in the context of text classifiers.

Length of tweetText -> AdaBoost

The first feature representation I experimented with was length of the tweetText field, finding that the AdaBoost classifier performed the best given the default set of parameters. I chose AdaBoost because its method of combining many weak classifiers for a given feature set and weighting erroneous predictions more heavily worked reliably to produce the best F1-score out of the algorithms listed above.

timeOfDay + dayOfWeek -> AdaBoost

The next features were the time of day and day of week which I decided to combine together since they were likely to be correlated. I also found that the AdaBoost classifier produced the best F1-score by default, and for similar reasons as above iterated upon it.

TFIDF of tweetText -> SVM

The final feature explored in depth was TFIDF which produced a vector representation of tweetText of the most important terms. I chose to apply an SVM algorithm due to my research into [2] which demonstrated how an SVM outperformed a KNN approach by achieving a micro-averaged F1-score of 0.97 (whereas the KNN achieved 0.94). They explain that an SVM is “robust in the presence of many features” which makes it particularly advantageous for TFIDF analysis.

AdaBoost + AdaBoost + AdaBoost -> Voting classifier

I combined these models into my own implementation of a voting classifier to combine their collective classification power, opting for a hard-voting method since the models it would be composed of produced discrete class predictions rather than probabilities. I applied a weighting on these models to favour the timestamp features and TFIDF values.

Dimensionality Reduction

Both AdaBoost algorithms required no dimensionality reduction since they were only 1 and 2-dimensional vector spaces respectively. I did not experiment with reduction for the SVM trained on the TFIDF data since I could control the vector space using the 'max_features' attribute of TfidfVectorizer. Hence, no dimensionality reduction was used in the final model.

Evaluation

Performance Metric

For evaluating the combination of feature extraction techniques and machine learning algorithms I have chosen the performance metric of the F1-score: a weighted average of a model's precision and recall, which takes into account what the model predicted and what the actual result was. This is explained further below:

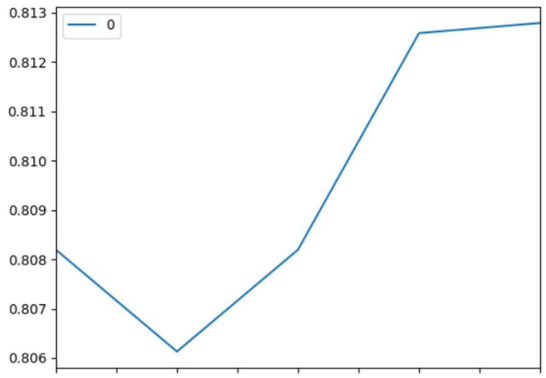
- Predicting a value as Fake is a Positive, while predicted a value as Real is a Negative
- The actual value being the same as the prediction is True, and False if it is not.
- Therefore,
    - True Positive = correctly predicted Fake values
    - True Negative = correctly predicted Real values
    - False Positive = falsely predicted Fake values
    - False Negative = falsely predicted Real values
- Precision is the ratio of correctly predicted positive observations to the total predicted positive observations (i.e. how often the Fake predictions were correct)
    - Precision = TP / (TP + FP)
- Recall is the ratio of correctly predicted positive observations to the total number of positive values there were (i.e. how often the Fake predictions were labelled)
    - Recall = TP / (TP + FN)
- The F1-score is a measure of both of these and is regarded as a better metric than accuracy in particular if there is an uneven distribution of entries per class.
    - F1-score = 2 * Precision * Recall / (Precision + Recall)

This demonstrates that F1-score is a better performance metric than accuracy and was used in the iteration process at finding the most successful models. This can be calculated using sklearn.metrics module which contains an 'f1_score' method, where the 'pos_label' is used to indicate the positive class (fake labels stored as 1 so 'pos_label=1').

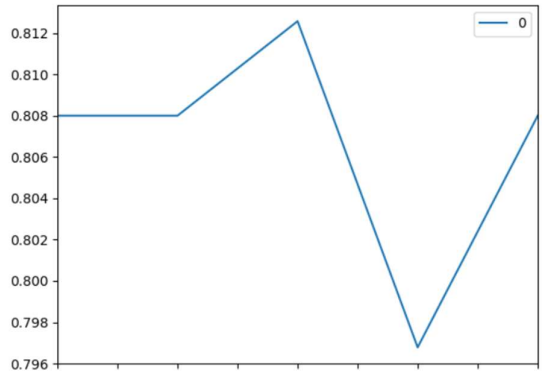
Model Iteration

After identifying the most successful models I refined these by experimenting with different hyperparameters:

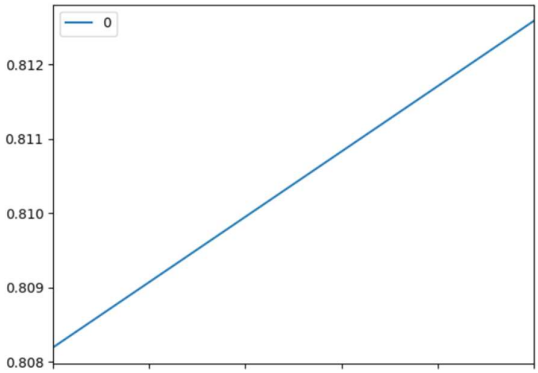
Length of tweetText -> AdaBoost



F1 score for n_estimators: 1, 10, 100, 500, 1000



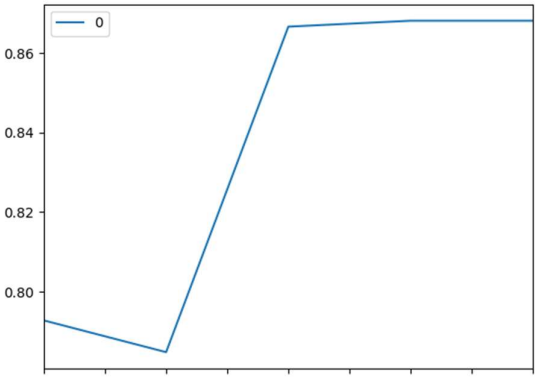
F1 score for learning_rate: 0.2, 0.5, 1.0, 1.5, 2.0



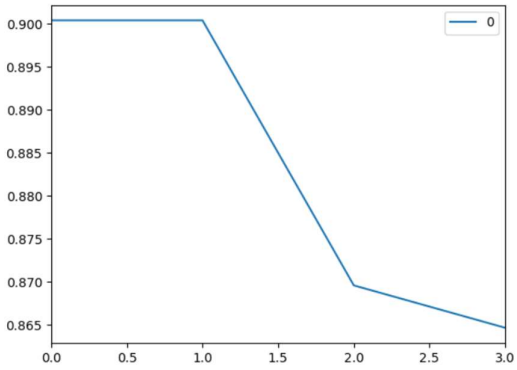
F1 score for algorithm: SAMME, SAMME.R

As can be seen the optimal values for this model were a n_estimator of 500, a learning rate of 1.0, and the SAMME.R algorithm however these values did little to significantly impact the F1 score. The maximum score achieved for this model was 0.81.

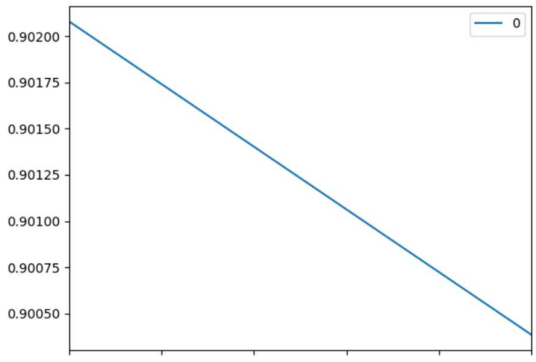
timeOfDay + dayOfWeek -> AdaBoost



F1 score for n_estimators: 1, 10, 100, 500, 1000



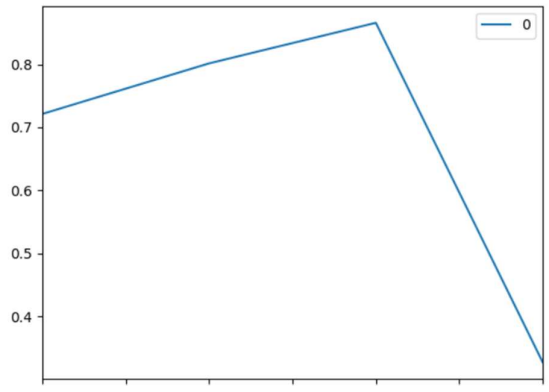
F1 score for learning_rate: 0.05, 0.1, 0.3, 0.5



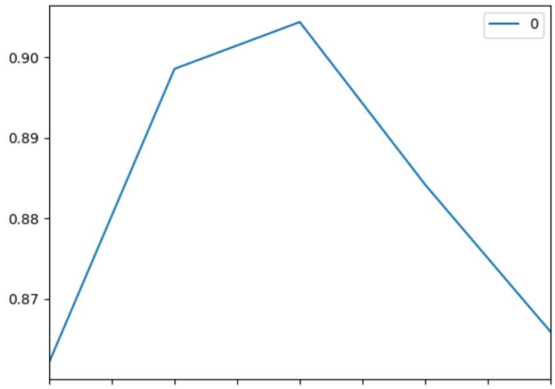
F1 score for algorithm: SAMME, SAMME.R

As can be seen the optimal values for this model were a n_estimator of 100, learning rate of 0.1 and the SAMME algorithm. I was surprised at how such low of a value for the learning rate impacted the overall F1 score, achieving a final maximum score of 0.90.

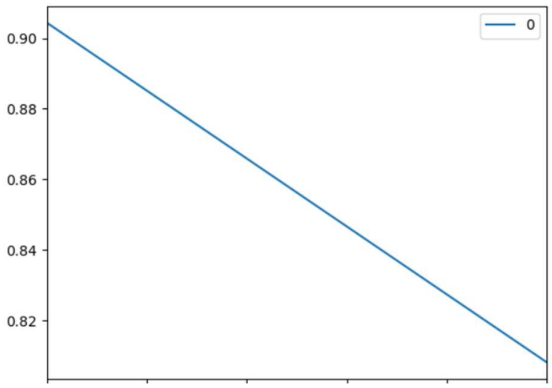
TFIDF -> SVM



F1 score for kernel: linear, poly, rbf, sigmoid



F1 score for C: 0.4, 0.5, 0.6, 0.7, 1.0



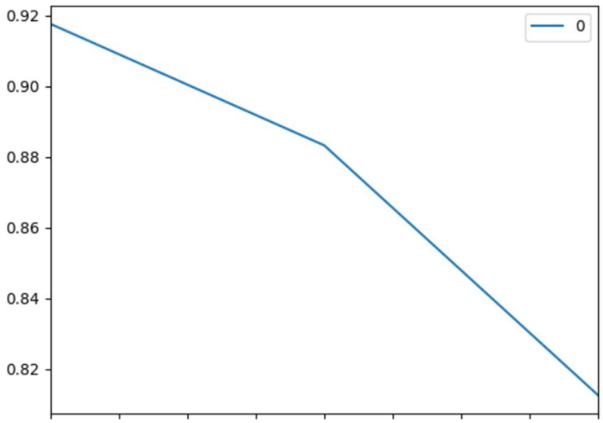
F1 score for gamma: scale, auto

As can be seen the optimal values for this model were an rbf kernel, c value of 0.6 and a gamma value of scale (any parameters left out were most optimal left as their default value). This managed to achieve a maximum F1 score of 0.90.

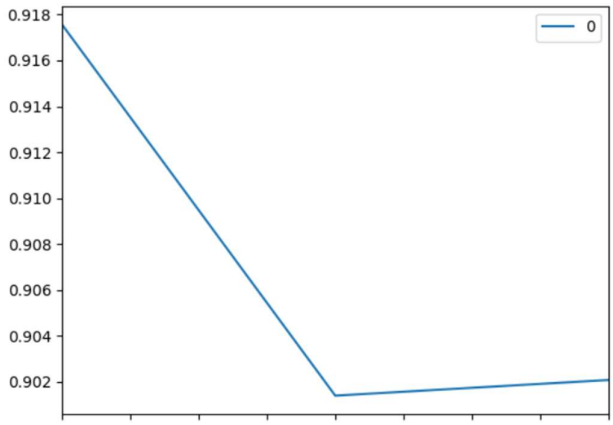
Comparing this to [2] they managed to achieve an F1 score of 0.97 for their TFIDF-SVM model, which means there may still be room for improvement in my model however it may also be likely the data is insufficient to achieve such a score, potentially due to the quality of language present in the data.

AdaBoost + AdaBoost + AdaBoost -> Voting classifier

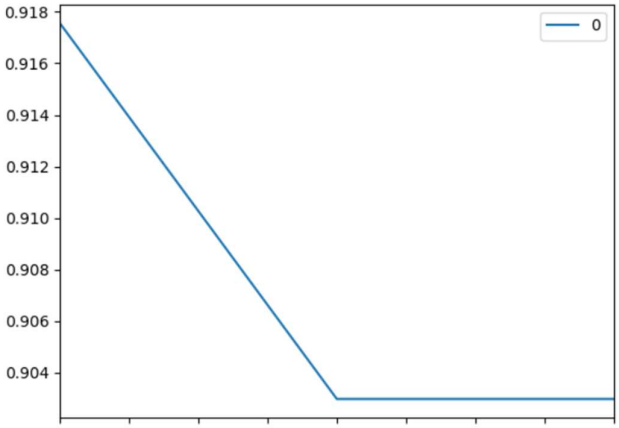
As discussed earlier I combined these models into an ensemble voting classifier where each model would have a weighted vote of whether they thought a given post was real or fake. To find the most optimal weighting for each model I tested the voting classifier with various weightings:



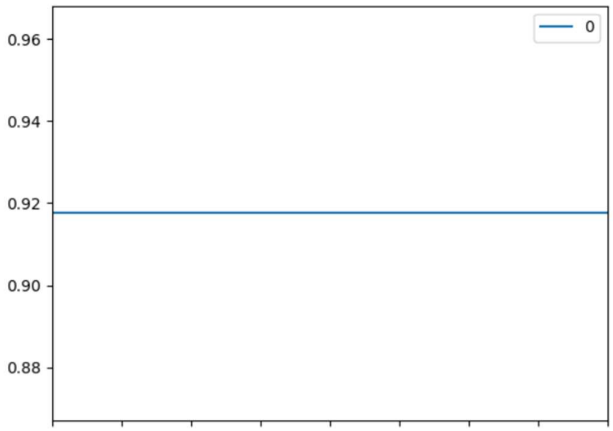
F1 score for weighting favouring text_length



F1 score for weighting favouring timestamp_feats



F1 score for weighting favouring tfidf



F1 score for weighting favouring text_length and timestamp_feats

As can be seen for the results there is no improvement by weighting certain combinations of models in the voting classifier. Indeed, even weighting text_length and timestamp_feats higher which were the most successful models saw no improvement better than an F1 score of ~0.92. Therefore, the optimal weightings for the voting classifier were equal weightings.

Conclusion

After experimentation I found the most optimal classification model to consist of a voting classifier built up of three algorithms trained on subsets of the feature vector space. These algorithms included AdaBoost twice for text length, again for time of day and day of week, and a SVM for the TFIDF analysis. I experimented with these algorithms to find the ideal hyperparameters to achieve the highest individual F1 scores, which were 0.81, 0.9 and 0.9 respectively. They were combined in a hard-voting classifier with equal weightings to produce the final F1 score of 0.92 (accuracy was 88%, precision was 0.88 and recall was 0.88).

From this process I learned that an ensemble of separate (working on different feature extractions) high performing models combine very well to produce an equally high or better voting classifier, which I would not have previously expected. I also found that the number of feature extraction techniques for text classification to be vast and highly dependent on the dataset and would identify this as an area for future improvement. A future version may expand its feature set to include the text analysis methods that I found less successful and experiment more to determine if they are useful indicators for classification.

I neglected to explore may deep learning algorithms for this task since they are not easily to create and I was unsure if the datasets contained the data required to apply deep learning methods, and I determined there were enough manual features I can identify that would make a standard machine learning algorithm suitable. If given more time I believe this would be a useful area to research, in particular [4] who explores applying text analysis techniques to a convolutional neural network. They found that “how well [their] model performs in comparisons depends on many factors, such as dataset size, whether the texts are curated and choice of alphabet”, demonstrating this may have been an unsuitable approach based off the characters of the given dataset.

References

[1] Aggarwal, C. and Zhai, C. (n.d.). A SURVEY OF TEXT CLASSIFICATION ALGORITHMS. [online] Available at: http://charuaggarwal.net/text-class.pdf [Accessed 2 Jan. 2020].

[2] Batal, I. and Hauskrecht, M. (n.d.). Boosting KNN Text Classification Accuracy by using Supervised Term Weighting Schemes. [online] Available at: http://people.cs.pitt.edu/~iyad/papers/CIKM_2009.pdf [Accessed 2 Jan. 2020].

[3] Boididou, C., Papadopoulos, S., Kompatsiaris, Y., Schifferes, S. and Newman, N. (2014). Challenges of computational verification in social multimedia. Proceedings of the 23rd International Conference on World Wide Web - WWW '14 Companion. [online] Available at: http://wwwconference.org/proceedings/www2014/companion/p743.pdf [Accessed 2 Jan. 2020].

[4] Zhang, X., Zhao, J. and Lecun, Y. (n.d.). Character-level Convolutional Networks for Text Classification *. [online] Available at: https://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf [Accessed 3 Jan. 2020].