COMP90051 Project 1 - Who Tweeted That?

Report by Team 192: 'The Last Two'

Introduction

We were presented with a training dataset of 328,932 tweets authored by 9,297 twitter handles, tasked with designing a classifier able to predict the author given a tweet. Our report highlights experiments, useful observations & future directions on using this dataset to develop machine learning (ML) models.

Feature description

Tweets in our training set are 'new-line' separated. Each tweet starts with a numerical user handle followed by unstructured text. Training tweets were relatively short with an average length of 84.3 characters. The mean number of tweets per author was 35.38, which is likely insufficient for ML algorithms. Training data also had a high class imbalance, with a standard deviation of 28.14 tweets. Furthermore, other challenges included various colloquial writing styles, hashtags, spelling mistakes, and URLs. To minimize noise for ML algorithms, pre-processing was done and compared with raw tweets to see if predictive performance improved.

Pre-processing

To make the tweets more interpretable and reduce noise, the tweets were 'tokenized'. That is, they were partitioned into tokens (words) separated by whitespaces, along with these changes:

Lemmatization	Reduce a word to its root form. Example: *stronger -> strong*. To do this, we used spaCy, an NLP Python package [3].
Removing *@mentions*	The provided dataset had the user mentions omitted. These are unlikely to help determine the tweet's author.
Stop word removal	Frequently used words such as 'and' and 'someone' do not appear to provide useful information about the writer. SpaCy's stop list was used to filter them.
Lowercasing & removing punctuation	Self-explanatory (e.g. *GOOD!! -> good*), so that ML algorithms could better identify the actual word.
Removal of duplicate tweets	These were tweets starting with RT (we observed >22,000 duplicate rows). We later found that this had a negative impact on our model's accuracy, as there may be users who mostly retweet, which could assist the classifier to predict.

During our experiments, we found our ML models suffered in predicting the test set (via Kaggle), compared to training with raw tweets. To remedy this, additional changes (or 'refinements') were made to pre-processing to perform better than on raw text:

Keeping emoticons	These may help determine the writing style or sentiment of tweet.
Changing URL format	Links such as *https://youtube.be/xyz1* are converted to *youtube.be*. This may help if an author often posts Tweets to certain websites.
Smaller stop word list	SpaCy's stop list had words which were unlikely to be used informally. Conversely, fewer Tweeters write so 'formally' that it may actually help the classifier detect them.
Replacing numbers	Unlikely to indicate the author, these were turned into a text: *<num>*.

Encoding unstructured text

Most ML algorithms require text to be encoded, as they cannot handle them unstructured. Thus, after preprocessing we looked into well-known approaches to encode them:

Bag-of-words (BOW, frequency-based)

Tweets are encoded as vectors $x = (x_1, ..., x_d)$ for d distinct words in the dataset, where each x_k is the number of occurrences of a distinct word w_k. This is simple to encode, but strips word ordering and semantic relationships (e.g. relatively close words may imply contexts/topics). With >220,000 distinct words, this can be very time and memory consuming for many ML algorithms.

'Hashing trick'

Instead of mapping words to distinct features, the word is mapped by a hash function. Given the hash function is uniform and resilient to small changes in words, by Zipf's law it is very likely one of the words is rarely used, if a hash collision occurs between two words [1]. Bag-of-words can then be compressed, which we use for SVM.

TF-IDF (Relevance Based)

The bag-of-words model may only account for term frequency. In TF-IDF, it accounts to also giving less importance to words which appear often in a corpus (i.e. our collection of tweets), so that the TF-IDF score for more meaningful (non-stop) words are higher.

Word embeddings

Instead of encoding by word counts, words are represented as real vectors. An example of this is word2vec [4]. Operations such as cosine similarity can then be made with two words to measure semantic similarity. Each tweet can then consist of a list of integers, where each integer is mapped to a word embedding.

Model selection & approach

After preprocessing, different models were trained to analyse their predictive accuracy on test data. Primarily, we focused on utilising

simple learners suitable for text classification. Except for CNNs, the *scikit-learn* software was used to create ML models [7].

Attempted Models

(Multinomial) Naive Bayes: Baseline Model

Naive Bayes has been shown to work well for NLP problems. As such, we chose it as our baseline for future model comparisons. It is a probabilistic model which assumes independence between features. We found it to perform well using bag-of-word encodings. An explanation could be that 'dependencies' between words, which form specific topics, are similar to that of other topics and thus 'cancel' each other out [2]. Moreover, Naive Bayes was quick to train with linear time complexity $O(Nd)$, where N is the number of tweets to train and d is the number of features.

Support Vector Machines (SVM)

SVMs were chosen on the basis it was also found to work well with text classification problems. However, SVMs have time complexity $O(d^3)$ in primal form and $O(N^3)$ in dual form; this is infeasible with such a large training set. Instead, stochastic gradient descent (SGD) optimiser was used to train a linear SVM model, which is possible due to it being a convex problem. Training samples were then partially fed to the SGD algorithm. Despite being unable to use non-linear kernels with SGD, better results were achieved.

Convolution Neural Networks (CNN) and other models

To utilise more recent models in literature [4, 5], CNNs were formed with word embeddings learned in its hidden layers. We found this provide the best test score on Kaggle. Other models tried included the perceptron (validation accuracy 17.8%) and random forest classifiers but are not further discussed due to finding relatively low accuracy in our experiments.

Training approach

After selecting a model, to avoid overfitting on the test set, a validation set was created by splitting 3.5% of the training set (approximately 11,500 tweets). Models were tested on this set to see how well it generalise, with classification accuracy as the main metric, in line with Kaggle's used metric. For SVM, multiple tweet encodings were also used to observe their effect on accuracy.

Cross-validating methods such as k-folding would be infeasible with the training set size, even though it would give a better idea on the bias-variance errors of our models. Instead, we compromised to creating (varying random seeds) a random validation set each time we trained a model. This is also done to avoid overfitting on the validation set. Moreover, we found validation score to be often similar to the reported test score from Kaggle, with a difference of at most 1.5%. This could suggest for each group of learners trained (e.g. multiple SVMs by varying the hyperparameter α), the variance may not be overly high but the unexplained error may be much higher, indicating features might need further engineering.

Each of our models also had hyperparameters to be adjusted. Varying these notably affected their performance. As some of these parameters had no clear rule for their optimal values, these were empirically found and tested with validation sets. Due to the size and thus time taken for training, methods such as performing a large grid search over a model's hyperparameters were not considered. Finally, when the best parameters were found, the model was trained (without validation set) and submitted to Kaggle.

Discussion of results

Multinomial Naive Bayes - Baseline

Firstly trained on raw tweets as bag-of-words, a validation accuracy of 19.01% was reached on the initial hyperparameter $\alpha = 1$, which represented the Laplacian smoothing of multinomial distributions. Eventually $\alpha = 0.01$ was found to provide the best results for both training and validation sets. Due to memory constraints, only 20,000 most common words in the training corpus were taken. Note performance degradation when we first cleaned our tweets, which was later improved ('refined cleaned tweets') to be on par with training on raw tweets.

	Validation	*Training*
Original tweets	*22.03%*	*81.19%*
Initial cleaned tweets	*19.30%*	*78.94%*
Refined cleaned tweet	*22.05%*	*78.80%*

Figure 1: *Accuracies on training and validation, trained on 20,000 words with highest term frequencies and* $\alpha = 0.01$.

We note the Laplacian smoothing α is aimed to avoid zero probabilities $P(x_i|y)$ to account for unseen training instances, where y represents some author and $x = (x_1, ..., x_d)$ is a tweet in its bag-of-words form. Setting it too high presumed the feature (or word) x_k was likely to come from author y, which may not be true in our overall dataset (training + test data). This parameter can be seen as regularisation, where as $\alpha \to 0$ reduces model 'complexity', so as to prevent overfitting.

Naive Bayes was tested (on refined cleaned tweets) with a larger amount of words counted, showing improved performance (as high as 24% validation), whereas lowering it to 5,000 features reduced it to 17.67%. However, expanding the feature space demanded very high memory use. Tested on Kaggle on the refined tweets, a test score of 22.47% was achieved as our baseline.

Linear SVM

Using a SGD classifier with the hinge loss function L, the training samples were shuffled and partially fed to the algorithm. This shuffling & training was repeated 10 times (iterations). Different regularisation functions (L1 and elastic net) were tried but found the validation score worsened. Consequently, L2 regularisation was used throughout our trained SVM models.

$$E(w, b) = \sum_{i=1}^{N} L(y_i, f(x_i)) + \alpha||w||^2$$

Figure 2: *Error function to find optimal weights for linear SVM.*

The hyperparameter α was also adjusted, which is related to the error function E in figure 2. Increasing α meant the weights w representing the hyperplane $f(x) = w^T x + b$ was sparser (i.e. less model complexity); the default scikit-learn value $\alpha = 10^{-4}$ yielded much lower validation accuracy than the Naive Bayes baseline. Consequently, α was lowered to the near-optimal value 10^{-6}, for lesser sparse trained weights whilst avoiding overfitting.

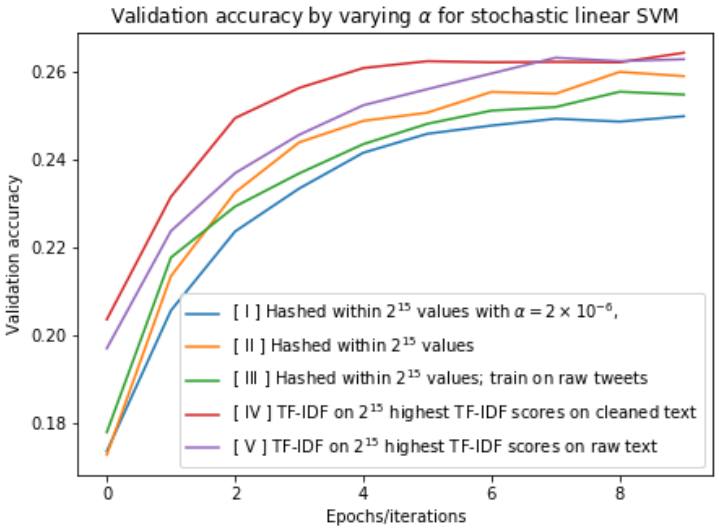

Figure 3: Validation accuracy with $\alpha = 10^{-6}$ (unless otherwise stated) with TF-IDF and hashed vector count representations.

To reduce the dimension of our feature space, hashing functions were used. From figure 3, hashing the words to 2^{15} features (notably less than $\approx 120,000$ distinct words in our cleaned set), yielded a validation accuracy 25.8% from model II; model III is also not far off, which was trained on raw tweets. This was better than our baseline from Naive Bayes.

For models IV and V, accounting for inverse document frequency (TF-IDF) without hashing to the top 2^{15} words with the highest TF-IDF scores further improved the validation score to 26.28% from model IV. Notably, models IV & V shared similar performance. This might be because by keeping words with highest TF-IDF scores, many stop-words are removed in the raw-text. Tested on Kaggle, a test score of 25.613% was achieved.

Convolutional Neural Networks

Using Keras [6], a simple CNN network was created with various inputs. Namely, tweets were into tokenized into words, bi-grams and tri-grams (e.g. 'Word' -> ['Wor', 'ord']). These distinct tokens were then given a vector embedding, which was learned in the CNN. A NVIDIA GTX 970 GPU was utilised to accelerate training.

Using a simple CNN architecture [5], it was composed of:

- A trainable embedding to map input, followed by a 1D convolution layer with 512 filters of height 2.
- Max pooling layer with window size 2 and stride 2.
- Finally, these were concatenated to a softmax function.

In essence, the filters could be interpreted as finding patterns in pairs of words; each kernel then has its size reduced in half to the most important characteristics by max pooling, which are fed into a softmax function to determine best classification via probabilities. For training, we found 2 epochs over the training set was enough. After that, we found the loss function used, the cross-entropy categorical loss function, over the validation set only increased.

Unit of embedding	*Validation score*
Word (on refined cleaned tweets)	15.677%
Bi-grams	24.046%
Tri-grams	27.339%

Figure 4: Validation accuracy on differing embeddings were made to represent words or n-grams.

We found splitting by words yielded poor results, likely due to the short character length of tweets. Thus, tweets were split into bi-grams and tri-grams which showed noticeable improvement.. Tested with Kaggle (tri-gram), a score of 26.38% was achieved.

Conclusion

In this project, aimed at predicting authors using Twitter posts, we initially looked at the dataset and preprocessed, which was then refined to remove noise. Afterward, suitable models were chosen and trained with results discussed. A more detailed discussion on their performances was made, with convolutional neural networks to perform the best in our experiments. In future work, different ways to represent tweets, as well as exploring NLP techniques, would be of interest to help further understand the problem.

References

1. Gimpert, B. (2013). *Hashing Language*. Some Ben?. Available at: blog.someben.com/2013/01/hashing-lang [Visited on 9/19/19].
2. Zhang, H. (2004). The optimality of naive Bayes. *AA*, *1*(2), 3.
3. Honnibal, M., & Montani, I. (2017). spaCy 2: Natural Language Understanding with Bloom Embeddings. *Convolutional Neural Networks and Incremental Parsing*.
4. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).
5. Shrestha, P., Sierra, S., Gonzalez, F., Montes, M., Rosso, P., & Solorio, T. (2017, April). Convolutional neural networks for authorship attribution of short texts. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* (pp. 669-674).
6. Francois Chollet and others. Keras [Computer Software]. Available at: https://keras.io.
7. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.