ECS763P: Fake News Detection Report

Eleanor Prashamshini (220772291)
Queen Mary University of London
e.prashamshini@se22.qmul.ac.uk

Introduction

This report describes the process of building, analysing and improving a natural language processor that classifies news statements as either 'FAKE' or 'REAL'.

NLP Pipeline

1. Input and Basic Pre-processing

The fake_news.tsv resource consists of 13 attributes from which we extract labels and statements. The label has six values, mapped to either 'FAKE' or 'REAL'. Each statement is converted to a bag-of-words (bow) stripped of any surrounding punctuations.

2. Basic Feature Extraction

Every bag-of-words is transformed to a feature vector with the word as key and its count as value. Additionally, we build on the global dictionary, adding new tokens discovered.

3. Cross-validation on Training Data

The dataset is divided into 80% development data and 20% test data. We are using a Linear Support Vector Machine Classifier (from sklearn.svm) to make predictions on the vector. To validate its performance in the training phase, we use k-fold cross validation. This method involves iterating over the development dataset, holding out a portion (size of dataset / k) of data and using the rest to train the classifier. Following this, the classifier is tested with the held-out data. This is repeated k times, as the window of held-out data moves across the entire dataset. Table 1 is the average performance report in terms of precision (p), recall (r), f1-score (f1), support (s) and accuracy (a) for each class across k folds of development data.

	p %	*r %*	*f1 %*	*s*
FAKE	49.52	49.17	49.31	356.2
REAL	61.06	61.40	61.21	463.0
macro avg	55.29	55.28	55.26	819.2
weighted avg	56.12	56.08	56.07	819.2
accuracy %	56.08			

Table 1: Average cross-validation report

4. Error Analysis

Currently, the classifier has an accuracy of 56.08%. However, this metric is insufficient to interpret what the classifier is doing right or wrong. Confusion matrices gives better insight about individual class performances (Table 2).

		Predicted	
		FAKE	REAL
True	FAKE	161	178
	REAL	194	287

Table 2: Confusion matrix heatmap of 1st fold

We observe that the number of False FAKE and False REAL predictions are imbalanced, f1(REAL) > f1(FAKE). The 'REAL' news is easily identified (yellow) in comparison to the 'FAKE' news (purple), and this in-turn balances the average performance metrics.

To better analyse the misclassified data, we print it to files and here are some deductions of possible reasons for confusion –

- Same words appear in both uppercase and lowercase at the start and in-between the text, resulting in two separate features. Ex: "Says U.S. … Obama says …"
- There are several punctuations in the files that could have contributed, had it been included in the feature space. Highest are '.' - 499 times and ',' - 218 times.
- Few contractions are missing the apostrophe while others are not. Ex: 'hasnt', 'thats'.
- A word appears in many forms and could be simplified to a lemma form to reduce dimensions in the feature space. Ex: 'seek', 'seeks' and 'seeking'.
- Unigrams do not provide enough context. Ex: 'attorney general', if we consider only 'general', it has a different meaning.
- There are a few conjoined words. Ex: "SaysMichael … prisonand …"
- Proper nouns occur in different variations but refer to the same person/place/thing. Ex: 'Obama', 'Barak Obama', "Obama's"
- Sentences vary in lengths, so a short sentence, say 5 words, might be represented with smaller magnitude as compared to a longer sentence, say 20 words, possibly skewing the feature vectors.

5. Optimising Pre-processing and Feature Extraction

Improving the feature vectors can boost the LSVM's performance. In this section, we explore different approaches to do this, and compare the effects using weighted average metrics and tables, where blue highlights the baseline and green the selected improvement.

(a) Pre-Processing: Table 3 records the

results of applying lowercasing (lc), including punctuations (pc), stemming (lm), removing stop-words (swr), and their combinations on the tokens.

	p %	*r %*	*f1 %*	*a %*
bow	56.12	56.08	56.07	56.08
bow,lc	56.45	56.30	56.33	56.30
bow,pc	56.40	56.34	56.35	56.34
bow,lc,pc	56.69	56.57	56.59	56.57
bow,lm	56.35	56.26	56.27	56.26
bow,lc,pc,lm	56.35	56.26	56.27	56.22
bow,swr	56.05	56.00	55.99	56.00
bow,lc,pc,swr	56.45	56.34	56.34	56.34
bow,lc,pc,lm, swr	55.85	55.74	55.74	55.74

Table 3: Pre-process steps performance

Adding punctuations and lowercasing the words, slightly improves performance. Other processes show minor changes, but their combinations bring the performance down.

(b) Contextual Features: Using bigrams (bi) and trigrams (tri), alongside unigrams (u), increases the context (Table 4). Higher order ngrams are not explored as the texts are of limited length.

	p %	*r %*	*f1 %*	*a %*
u	56.69	56.57	56.59	56.57
u,bi	58.67	58.82	58.68	58.82
u,tri	60.35	60.57	60.37	60.57
u,bi,tri	59.73	60.00	59.76	60.00
u,bi,tri,lm	59.96	60.23	59.99	60.23

Table 4: Ngram combinations performance

The results incline us to adding trigrams alone. Nevertheless, to avoid overfitting the data and considering the increase upon adding bigrams, a combination of all three would be better suited. Stemming the tokens in n-grams increases the performance and decreases the feature space.

(c) Ngram Weights and Normalisation: In order to optimize the combination of uni, bi, tri-grams, we use the weights (0.1, 0.1, 0.2) respectively. Figure 1 captures the darkest points in the bottom left corner, indicating that weights in the region improve accuracy.

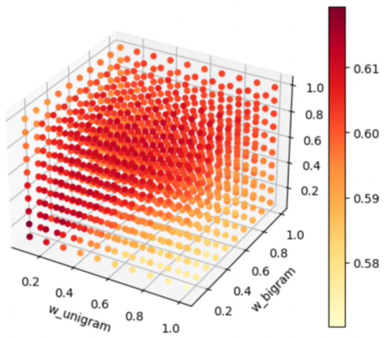

Figure 1: Weights and accuracy (colour)

To avoid skewing, we can normalise values by dividing them with sum(values), count(words) or count(features). Contrasting to our reason for skew in error analysis, none of the techniques help (Table 5). Thus, the vectors need not be normalized.

	p %	*r %*	*f1 %*	*a %*
count	59.96	60.23	59.99	60.23
ngram – wts	61.45	61.93	61.26	61.93
/ sum(values)	59.47	58.82	51.15	58.82
/ cnt(words)	60.59	61.04	58.58	61.04
*/ cnt(feature)**	31.98	56.52	40.84	56.52

Table 5: Weights and normalisation performance (* failed to converge)

6. Other Metadata in the File

Table 6 shows the impact of using metadata as features – adding all columns generically as binary features (g), dividing the subjects into individual features (sb, nsb – normal), using the counts as attribute-value mappings (cnt, ncnt – normal) and applying weights (w).

	p %	*r %*	*f1 %*	*a %*
statement	61.45	61.93	61.26	61.93
g	69.37	69.48	69.36	69.48
g,sb	68.97	69.06	68.95	69.06
g,nsb	69.19	69.26	69.17	69.26
*g,nsb,cnt**	60.53	60.77	60.35	60.77
g,nsb,ncnt	70.27	70.36	70.26	70.36
w(g),nsb,ncnt w=0.1	73.85	73.95	73.82	73.95
ncnt	74.59	74.67	74.54	74.67

Table 6: Metadata inclusion performance (* failed to converge)

Including all the metadata as features results in a performance leap. Although, after some pre-processing on the features and normalizing the values, we observe increase in performance even with w=0.1 applied to all features, except counts. Hence, we conclude that 'total_barely_true_counts', 'total_false_counts', 'total_half_true_counts', 'total_mostly_true_counts' and 'total_pants_on_fire_counts' are the most significant metadata features.

Conclusion

	p %	*r %*	*f1 %*
Training	74.59	74.67	74.54
Testing	74.46	74.38	74.11

Table 7: Training and testing results

The classifier is behaving as expected in both training and testing phases (Table 7). There is a considerable improvement in the overall performance from Table 1, however this was achieved by a trade-off with processing time because of pre-processing steps and increase in the number of features passed to LSVM.