Fake News Detection Using Natural Language Processing

nvzf61

1 Introduction

We compare the performance of two feature extraction techniques and two classification techniques on the task of fake news detection. We justify our implementational choices and proposed method throughout. We conclude with a discussion of ethical considerations.

2 Problem Definition

2.1 Problem Outline

We use the FNC dataset [1] comprising 49,972 pairs of article headlines and bodies with their associated stances - agree/disagree/discuss/unrelated. To classify the stance of an article given the content of the article's headline and body we use a two-step approach:

Task-1: Classify all articles as related/unrelated.

Task-2: Classify the related articles as agree/disagree/discuss.

Task-3: Classify all articles as agree/disagree/discuss/unrelated by using the best models from Task-1 and Task-2 consecutively

For feature extraction we use **TF-IDF** and **BERT** (specifically **SBERT**).

For classification we use **RFC** (random forest classifier) and **MLP** (multi-layer perceptron).

2.2 Problem Framework

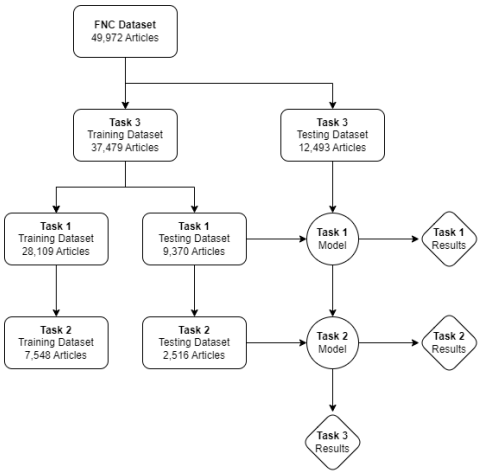

Figure 1: For brevity, we do not explicitly define the partitioning of the FNC dataset with respect to each task but instead provide a visual representation

2.3 Dataset Analysis

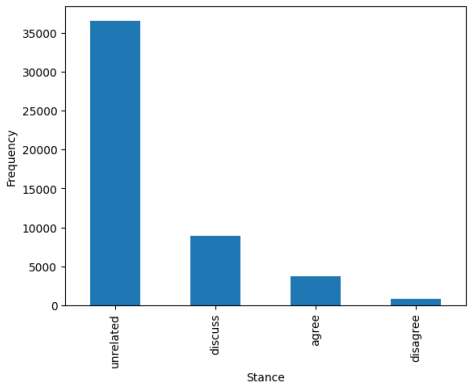

Figure 2: Bar plot highlighting the imbalance of stances within the NFC dataset

Our dataset contains imbalanced data. To mitigate this bias, one option is to undersample the majority classes, oversample the minority classes, or synthetically create data using SMOTE. We employ a different solution that retains all elements in the dataset by simply hyperparameter tuning our classification models for balanced accuracy and training with balanced class weights. Breaking the problem down into sub-tasks also helps.

$$\begin{gathered}\text{Balanced Accuracy} = \frac{\sum_{i=1}^{n} \text{Recall}(i)}{n} \\ \text{where for each class i} \\ \text{Recall}(i) = \frac{TP}{TP + FN}\end{gathered} \tag{1}$$

2.4 TF-IDF

TF-IDF is a statistical metric that is calculated by multiplying the term frequency and inverse document frequency for a given word in a document. The term frequency evaluates how common the given word is in a document whilst the inverse document frequency evaluates how common the given word is with respect to the corpus. Intuitively, this gives a higher score to the more information rich (unique) words in a document to create a more informative embedding.

$$\begin{gathered}tf(w,d) = \log(1 + f(w,d)) \\ idf(w,D) = \log(\frac{N}{f(w,D)}) \\ tfidf(w,d,D) = tf(w,d) \cdot idf(w,D)\end{gathered} \tag{2}$$

where N is the number of documents in the corpus D
and $f(w,d)$ is the frequency of word w in document d
and $f(w,D)$ is the frequency of word in the corpus D

By calculating the TF-IDF score for all words in the vocabulary for a given document we can produce document embedding. This allows for intuitive comparison between documents since the order of elements (corresponding to words) in the vectors is always the same.

However, this produces a memory/computationally expensive sparse vector since most words in the vocabulary aren't present in a specific document. Moreover, TF-IDF does not take into account the context of words or perform well with domain-specific knowledge.

2.5 BERT and SBERT

BERT is a bidirectional transformer-based masked language model that can be used for word encoding (among other things). We use SBERT, a sentence-level variant of BERT (a document can be considered as a long sentence), since it vectorises documents into the same format as TF-IDF (a single vector) so that we may use the same framework for both approaches.

In comparison, BERT as a word encoder produces a list of vectors for each document where each vector corresponds to the embedding of each consecutive word. If we still wished to use BERT as a word encoder, we could consider taking the average of the word embeddings for a document to produce a document embedding.

BERT is pre-trained on the language modelling objective to predict masked words based on their semantic context. SBERT fine-tunes this BERT model on the sentence similarity task using a siamese architecture that contains two identical BERT architectures to process two sentences as pairs simultaneously. A max pooling layer is used to give fixed-size representations for input sentences of varying length. SBERT is trained on the triplet loss function to learn embeddings that minimise/maximise the distance between similar/different sentences and to prevent training on all possible combinations of sentences.

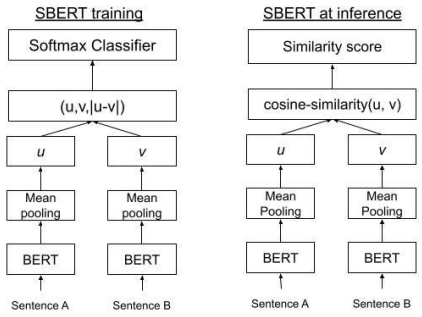

Figure 3: SBERT architecture at training and inference

Most importantly, BERT considers the context of words (thanks to the bidirectional transformers). BERT also produces a memory/computationally efficient dense vector. This should make a BERT embedding perform better and quicker then a TF-IDF embedding.

Since BERT is a pre-trained model, it can also be fine-tuned for a specific task - however this may be computationally expensive. More importantly, BERT loses the interpretability that TF-IDF has making it harder to understand its decision making process.

3 Proposed Solutions

3.1 Preprocessing

We concatenate the headline and body text to produce our training data - we refer to this concatenation as the article. Therefore, it is possible that our classification is using information other than the relationship between the headline and the body to determine the stance of the article e.g. it might use only the body content to classify the stance.

Although not strictly necessary, we preprocess the text content of our articles for our TF-IDF approach (see code). Conversely, BERT does not require preprocessing and in fact benefits from non-textual contextual information.

3.2 Feature Extraction

We fit our TF-IDF model to the training corpus which is then used to vectorise both training and testing data. Each vectorised article is given by a 74,214-dimensional sparse vector.

BERT is a pre-trained model so does not require fitting to any dataset. We use the paraphrase-MiniLM-L3-v2 SBERT transformer from huggingface [2] given its relatively low encoding speed and size with respect to its performance. Each vectorised article is given by a 768-dimensional dense vector.

3.3 Classification

For our ML classification approach we use RFC. This model was chosen from other ML algorithms by direct comparison of performance using the lazypredict package [3]. Note, this package does not perform hyperparameter tuning so it is possible that there is a model that performs better than RFC after hyperparameter tuning.

For our DL classification approach we use MLP. We choose MLP over more sophisticated algorithms such as LSTM or RNN since it suits the nature of our data - LSTM and RNN both require sequential data but our document embeddings are not sequential.

Instead, if we were to use BERT as a word encoder it would provide sequential information since consecutive words in the articles are embedded consecutively. Alternatively, we could still use TF-IDF or SBERT as document encoders to leverage sequential information by dividing each article into a sequence of consecutive paragraphs or sentences to be encoded separately. However, given the high performance we achieve using merely document encoders we consider this superfluous.

3.4 Imbalanced Improvements

We attempt hyperparameter tuning our classification models for balanced accuracy but find little improvement. We don't train our classifiers with balanced class weights for comparability since this isn't possible with sklearn's MLP implementation. Therefore, all our above classifiers have the default hyperparameters and nothing to mitigate the imbalanced data.

Instead, we create a custom MLP classifier with balanced class weights (for Task-2 only since Task-1 has near perfect performance already). This model has 2 hidden layers of size 512, ReLU activation functions, (notably) dropout with probability 0.2, and optimises the cross-entropy loss (with balanced class weights).

4 Results

4.1 Task-1 - Related/Unrelated Classification

	precision	recall	f1-score	support
related	0.96	0.82	0.89	2505
unrelated	0.94	0.99	0.96	6865
accuracy			0.94	9370
macro avg	0.95	0.90	0.92	9370
weighted avg	0.94	0.94	0.94	9370

Balanced accuracy score: 0.9048761908084777

(a) TF-IDF RFC

	precision	recall	f1-score	support
related	0.99	0.85	0.92	2505
unrelated	0.95	1.00	0.97	6865
accuracy			0.96	9370
macro avg	0.97	0.92	0.94	9370
weighted avg	0.96	0.96	0.96	9370

Balanced accuracy score: 0.9243835708044944

(b) SBERT RFC

	precision	recall	f1-score	support
related	0.99	0.97	0.98	2505
unrelated	0.99	0.99	0.99	6865
accuracy			0.99	9370
macro avg	0.99	0.98	0.99	9370
weighted avg	0.99	0.99	0.99	9370

Balanced accuracy score: 0.9834787816937138

(c) TF-IDF MLP

	precision	recall	f1-score	support
related	0.98	0.99	0.99	2505
unrelated	1.00	0.99	1.00	6865
accuracy			0.99	9370
macro avg	0.99	0.99	0.99	9370
weighted avg	0.99	0.99	0.99	9370

Balanced accuracy score: 0.9936017258999845

(d) SBERT MLP

Figure 4: Performance of our feature extraction and classification techniques on Task-1

4.2 Task-2 - Agree/Disagree/Discuss Classification

	precision	recall	f1-score	support
agree	0.75	0.73	0.74	725
disagree	0.52	0.43	0.47	129
discuss	0.89	0.91	0.90	1662
accuracy			0.83	2516
macro avg	0.72	0.69	0.70	2516
weighted avg	0.83	0.83	0.83	2516

Balanced accuracy score: 0.6874662495243297

(a) TF-IDF RFC

	precision	recall	f1-score	support
agree	0.86	0.77	0.81	725
disagree	0.76	0.46	0.57	129
discuss	0.90	0.97	0.93	1662
accuracy			0.89	2516
macro avg	0.84	0.73	0.77	2516
weighted avg	0.88	0.89	0.88	2516

Balanced accuracy score: 0.7327715293278277

(b) SBERT RFC

	precision	recall	f1-score	support
agree	0.92	0.92	0.92	725
disagree	0.71	0.73	0.72	129
discuss	0.98	0.98	0.98	1662
accuracy			0.95	2516
macro avg	0.87	0.87	0.87	2516
weighted avg	0.95	0.95	0.95	2516

Balanced accuracy score: 0.8744950914686221

(c) TF-IDF MLP

	precision	recall	f1-score	support
agree	0.89	0.87	0.88	725
disagree	0.72	0.63	0.67	129
discuss	0.95	0.96	0.95	1662
accuracy			0.92	2516
macro avg	0.85	0.82	0.84	2516
weighted avg	0.92	0.92	0.92	2516

Balanced accuracy score: 0.8218956558717254

(d) SBERT MLP

	precision	recall	f1-score	support
0	0.90	0.94	0.92	725
1	0.77	0.77	0.77	129
2	0.98	0.97	0.97	1662
accuracy			0.95	2516
macro avg	0.88	0.89	0.89	2516
weighted avg	0.95	0.95	0.95	2516

Balanced accuracy score: 0.8910779340131518

(e) TF-IDF Custom MLP

	precision	recall	f1-score	support
0	0.88	0.92	0.90	725
1	0.68	0.78	0.72	129
2	0.97	0.94	0.95	1662
accuracy			0.93	2516
macro avg	0.84	0.88	0.86	2516
weighted avg	0.93	0.93	0.93	2516

Balanced accuracy score: 0.878141214003862

(f) SBERT Custom MLP

Figure 5: Performance of our feature extraction and classification techniques on Task-2

4.3 Task-3 - Full Pipeline

	precision	recall	f1-score	support
agree	0.87	0.92	0.89	910
disagree	0.79	0.77	0.78	230
discuss	0.97	0.96	0.97	2223
unrelated	1.00	0.99	1.00	9130
accuracy			0.98	12493
macro avg	0.91	0.91	0.91	12493
weighted avg	0.98	0.98	0.98	12493

Balanced accuracy score: 0.9110864945236757

Figure 6: Performance of our best feature extraction and classification techniques from Task-1 (SBERT with MLP) and Task-2 (TF-IDF with custom MLP) on Task-3

5 Analysis and Discussion

We see that our DL classification approach (MLP) always performs better than our ML approach (RFC). This is expected since DL is generally able to learn more complex patterns.

We also see that our SBERT feature extraction approach almost always performs better than our TF-IDF approach. This is expected since SBERT produces a more information rich embedding as it considers the context of words. We find that our MLP and custom MLP classifiers perform worse on SBERT than on TF-IDF for Task-2. Since the difference is marginal, this may simply be due to training/testing stochasticity. It could also be due to the size of the SBERT model since we selected the smallest model for the quickest encoding speeds.

For our full pipeline (Task-3), we achieve a very high accuracy of 98%. This metric is biased towards the majority class so our balanced accuracy score of 91% is more appropriate. We see that the precision/recall/f1-score for the majority class (unrelated) are all nearly perfect thanks to the high performance in task-1. However, we also see that the precision/recall/f1-score are all worst over the minority class (disagree). This is likely due to the fact that there are simply too few instances of this class to train on despite our efforts to mitigate this by hyperparameter tuning for balanced accuracy and training with balanced class weights. We may want to consider oversampling/undersampling/SMOTE.

6 Ethical Implications

Primarily, we are subject to biases present in the FNC dataset. We are also subject to biases present in the training data for SBERT. These biases could cause unfair false positive/negative fake news classifications. Given the current training data, we are also only able to classify English articles. Moreover, the gathering of further data raises possible privacy concerns.

A possible future misuse case for fake news detection is censorship and limitation of free speech where legitimate sources are either accidentally of purposefully (maliciously) labelled as fake news. Similarly, it could be used for propoganda where fake news is forcefully labelled as true. Therefore, this could result in complex legal implications. As such, transparency of the development and deployment of these models is important.

7 Conclusion

We achieve a very high success rate in classifying the stance of articles to detect fake news. We see that BERT consistently outperforms TF-IDF as a feature extraction technique and MLP consistently outperforms RFC as a classification technique. Despite our efforts, there is still room for improving performance over the minority classes. Nevertheless, our model provides valuable fake news detection so long as we respect the ethical considerations stated.

References

[1] *Fake news challenge stage 1 (FNC-I): Stance detection.* URL: `http://www.fakenewschallenge.org/`.

[2] *Pretrained models.* URL: `https://www.sbert.net/docs/pretrained_models.html`.

[3] *Lazypredict.* URL: `https://pypi.org/project/lazypredict/`.