Text Technologies for Data Science Coursework 3 Report

Group 14

February 2019

Abstract

For this project we set out to discover methods that can be used to identify propaganda both on the article level and sentence level. Using an autoencoder we successfully reduce the dimensionality of our data set for document level classification and end up producing a random forest model over this lower dimensional latent space. Furthermore, we explore using a Convolutional Neural Network with different word embeddings and achieve an F1 score of 0.97 on test data in this classification task.
We then compare the effectiveness of using various features at the sentence level in order to identify characteristics of propaganda and subsequently inform what kind of features might be useful in propaganda detection models and use the most effective feature to supplement our sentence classification models.

1 Introduction

Studies have shown that fake news spread faster than the truth on social media websites like Twitter [13]. Subsequently, a large body of research has been done on identifying fake news articles and finding false facts.
However, there's been relatively little focus on identifying propaganda and bias in news articles. Although fake news might certainly contain propagandistic sentiments, it is important to differentiate between the two. Biased and propagandistic news might not necessarily contain inaccurate information, but might cherry pick facts or might present a one-sided story, which could influence the public perception of important events.
A recently organized datathon focused on precisely this problem [2]. For this project we focused on the task of classifying propagandistic news articles(Section 2) and comparing features for propagandistic sentences(Section 3). For these tasks 2 datasets were used, provided by the datathon organizers. The datasets are described in the relevant sections.

2 Document level classification

2.1 Data description

The Data provided for this challenge was in the form of a text - 35993 articles. Each article includes the article text, a unique ID and a label ('propaganda'/'non-propaganda'). The data includes 4012 propaganda articles and 31972 non-propaganda articles (11% propaganda). The data is inherently noisy. This means that it comes from news outlets that have been classified by humans as propagandistic or not by looking at a large sample of the outlet articles. Therefore each article gets the label of its respective news outlet. However, not all articles from this outlet might be propagadistic(or not). It's argued that a big enough training set could deal with this noise.

2.2 Preproccessing steps

2.2.1 Tokenisation, stopping and stemming

The first steps we took to preprocess our data were to perform tokenisation, stopping and stemming. Using the NLTK [12] python package this could be done easily in one go. First we extracted the text portion of each article, then we applied standard NLTK tokenisation. Next we removed any words appearing in the NLTK list of English stop words in order to reduce the size of our data a little and removed words that shouldn't have any effect on the classification of the article. Finally we used the NLTK PorterStemmer to stem each word so that words with the same meaning but with different tenses or plurality are classified as the same which should enable our model to learn the significance of a word as a whole rather than each different suffix separately. After tokenisation, stemming and stopping, we one-hot-encoded the data into a matrix for classification. The shape of the resulting matrix was (35993,207185(+ 2 class vectors)).

2.2.2 Dimensionality reduction

In order to implement models on our dataset we initially explored some of the models proposed by teams in the hackathon from which our dataset came. The methods suggested by these teams were achieving high F1 scores in some cases. However, due to the size of the dataset we were unable to run similar methods given our own compute resources. We therefore set out to try and achieve similar classification levels on our personal computers using dimensionality reduction techniques to allow us to quickly process all the data and experiment with different classification techniques.
Given the computing power constraints we had, the one hot encoded matrix was too large to allow us to run classification algorithms at this point and some of the classification algorithms we wished to test were not suited to such high-dimensional data. In order to reduce the dimensionality of the data set we employed an autoencoder to learn a latent space model of our dataset. We chose this over principle component analysis as it is easy to implement batch

training so as not to use too much memory and cause the OS to kill our python script.
Our autoencoder was able to learn a latent space representation of the data by training the network such that the input and output are the same data and designing the network architecture to have a bottleneck in the middle such that in order to reconstruct the data for output the network must learn a reduced dimensionality representation of the whole data set at the bottleneck (figure 1). The encoding step puts the data into a latent space representation, and the decoding step attempts to reassemble the original data from the latent space.

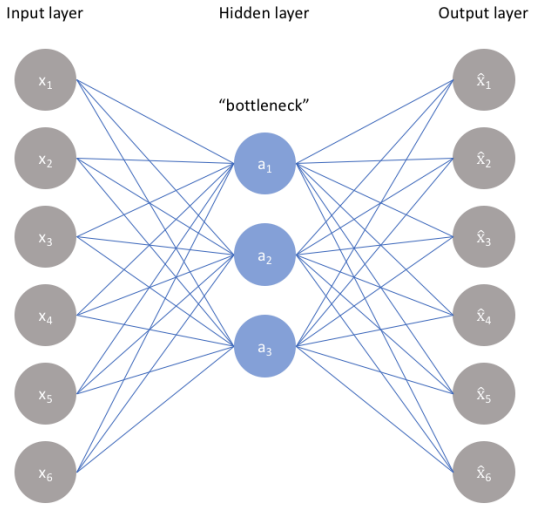

Figure 1: Autoencoder architecture diagram [6]

The network architecture we used included 2 hidden layers of size 256 and 128 respectively trained for 1000 epochs with a resulting Root Mean Squared Error in reconstruction of 0.0130. Our training input and output is the one-hot-encoded representation of the articles without the class labels in order that class labels are not explicitly inserted into the latent space. The training progression can be seen in figure 2. We can see that the network struggles to find any sort of pattern in the data set for around 150 epochs and then rapidly improves over the next 400 epochs.
After training the network we put all our data into this new latent space representation. In order to do this we simply ran the encoding step of the network on each article's one-hot-encoded representation in order to force it into 128 dimensions. We used this data from here onward for document level classification.

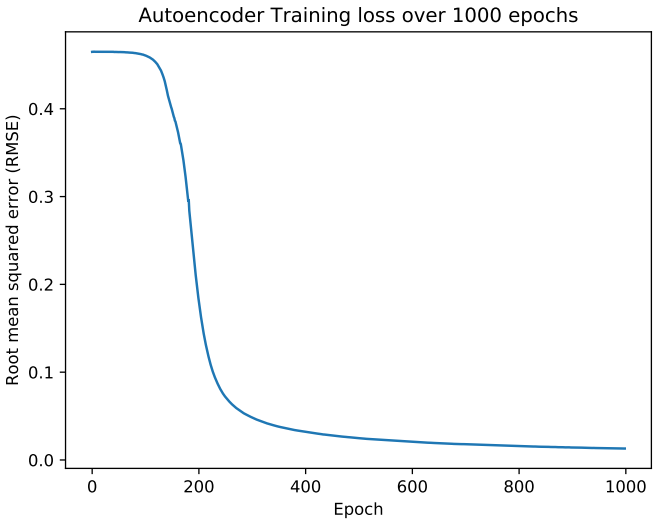

Figure 2: Autoencoder training

2.3 Experiments

2.3.1 Methodology

First we split our data into training and test sets, due to the class imbalance in the data we ensure that we have examples of both classes in the training and test sets by splitting the data by class and then splitting these sets into training and test sets before merging the sets. finally we resample the minority class ('propaganda') in each set so that we now have class balanced training and test sets. We split the data before resampling to ensure no data point was put into both the training and test sets.
The datathon [2] has a leaderboard of entries which is ranked by F1 score on the Propaganda class. We therefore tried to find models which have the best F1 score ourselves but display results for precision and recall as well so that we can see the trade-offs of each model.
The models we employed were from the Scikit learn package [8] in order to allow us to quickly compare the results of different classification approaches and allow fine-tuning easily. Seeds were used for reproducibility.

2.3.2 Baseline

Our baseline model is just random choice of either of the two categories. As expected the classifier results are all around 0.5

Baseline results: Precision: 0.492848, Recall: 0.495700, F1 Score: 0.494270.

2.3.3 Naive Bayes Classification

We next tested a Gaussian Naive Bayes classifier on the data due to its success in binary classification problems in the past such as Spam/Ham classification and it's short training time.

Naive bayes results: Precision: 0.645233, Recall: 0.588428, F1 Score: 0.615523

Naive bayes gives us results significantly better than the baseline model which we would expect on problems which contain any structure whatsoever.

2.3.4 Support vector machines

We also explored both a Linear Support Vector Classifier and a Support Vector Classifier using a radial basis function to transform the data.

Linear SVM results: Precision: 0.683943, Recall: 0.693354, F1 Score: 0.688616
RBF SVM results: Precision: 0.709299, Recall: 0.727600, F1 Score: 0.718333

We see that both of these models improve on the Naive Bayes classifier and the that the Radial Basis function improves classification somewhat this suggests a non linear decision boundary in the data.

2.3.5 Multi-layer Perceptron

We next used a multi-layer perceptron on the basis that we think we may have some success learning a non linear model of our data for classification due to the abstract features learned by the autoencoder not necessarily being linearly related and the improvement that radial basis functions made to our SVM model. We used the Scikit learn off-the-shelf classifier with layer sizes 128,256 after some trial and error. The Adam learning rule was used with an adaptive learning rate which started at 0.1.

Perceptron results: Precision: 0.654691, Recall: 0.769351, F1 Score: 0.707405

Our perceptron shows an increase in recall over our other models, but a decrease in precision, we suspect the increase in recall is due to the perceptron being able to create a non linear decision boundary and thus effectively recall more correct data points from random areas of the decision space.

2.3.6 K-nearest-Neighbours

Next we tried a KNN model as our data is normalized and not too high dimensional. We use 5 nearest neighbours a K-d tree to speed up search.

KNN results: Precision: 0.827765, Recall: 0.848475, F1 Score: 0.837992

We see that the KNN algorithm provides the best recall so far and the best precision as well. We think this is due to setting K to 5 so that noise in the data is less likely to affect the prediction.

2.3.7 Random Forest

Finally we try a Random Forest approach. We used 50 trees of max depth 10 in order to try not to overfit on the data too much.

Random Forest results: Precision: 0.955692, Recall: 0.853323, F1 Score: 0.901611

We see that the Random Forest approach achieves the best results by far in terms of precision, we think this is due to the averaging mechanism employed. And the recall is slightly better than the KNN algorithm.

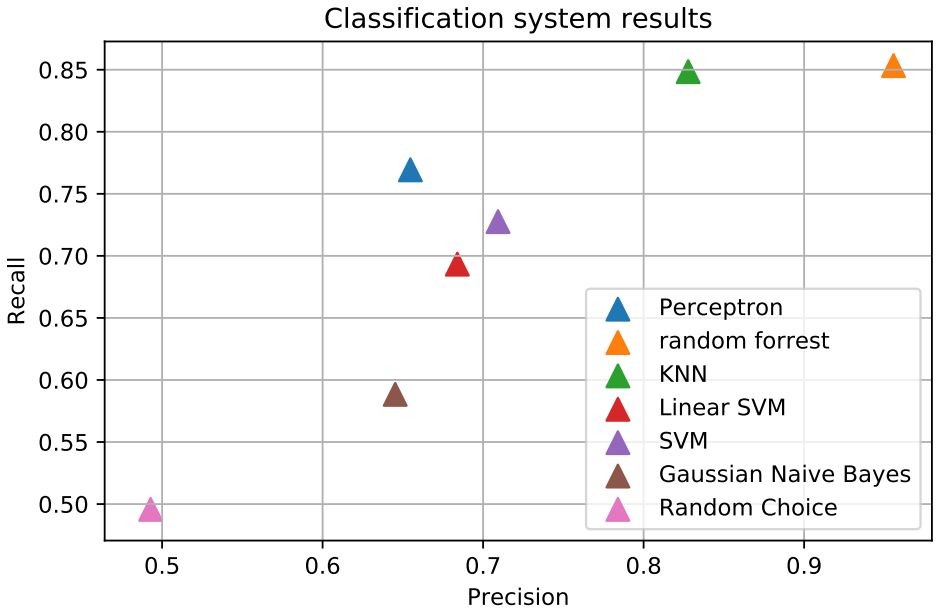

Figure 3: Classification system results

2.4 Revisiting Perceptrons

A different approach we had to document level classification was to once again visit the idea of using a multi-layer perceptron although this time without reducing the dimensionality of the data. Instead, we decided to transform the data using different word embedding methods and then feed it into a Convolutional Neural Network.

2.4.1 Preprocessing Steps

As mentioned before, there is a class imbalance in the data. When training Convolutional Neural Networks, it has been shown that the best results are achieved when an equal amount of each class exists in the training data. [5] Thus, we selected only 8 000 documents which include an equal sample of non-propagandistic and propagandistic documents. During training, we used 6 800 samples with 70% being used for training and 30% for validation. The remaining 1 200 documents were used as a final test set.

The data is then transformed into a numeric sequence according to an index fitted on the vocabulary. This vocabulary takes the 100 000 most frequent words in the data and thus all other words are treated as unknown. As the documents have varying lengths, all the documents are padded with 0s to fit the length of the longest document.

2.4.2 Word2Vec Representation

For embedding the text we used the Gensim implementation [11] of the Word2Vec embedding [7] . Word2Vec embedding transforms the word index into a vector representing the word. The logic being that similar words will have similar vectors. This allows us to do less preprocessing, such as stemming, on the words before transforming them as different inflections of words will have similar representations provided the same preprocessing is done when training the embeddings. We trained the embedding on the Text 8 Corpus developed by Matt Mahoney , which contains a subset of Wikipedia text to create 200 dimension representations.

Simple Model The first model we tested with was a simple convolutional architecture which had a convolutional layer with a kernel size of 2, followed by a pooling layer, dropout and then densification to reach a prediction.

Simple CNN (W2V) Results: Precision: 0.777947, Recall: 0.842454, F1 Score: 0.808917

We see that this result proved to be worse than the previous methods we had explored. However, like the previous perceptron we explored, the recall achieved on the test set was very high.

Complex Model In an effort to improve the precision and overall performance of the model, we tweaked it by adding another input branch. This was also a Convolutional layer although the kernel size was extended to 3, thus representing a method looking at trigrams too rather than just bigrams. The two input layers were then concatenated and the model is the same otherwise

Complex CNN (W2V) Results: Precision: 0.883333, Recall: 0.878939, F1 Score: 0.881131

As seen above, the precision on the test set actually increased when using the more complex model. Furthermore, we see that the recall is now also improved.

2.4.3 GloVe Representation

Despite providing worse results than the random forest, the convolutional approach was showing promise, especially with regards to recall. After some analysis of the results and further exploration of the Word2Vec implementation, we decided to use another embedding for training, namely GloVe [9].

This embedding is an alternation of Word2Vec. For this embedding, we used a pre-trained embedding of a subset of twitter tweets much larger than what we had used for training our Word2Vec embedding. We stuck with a 200 dimension representation for each word as we believed this had proved effective beforehand and increasing this would lead to too large an increase in training time.

Simple Model We first trained the original simple convolutional model as this was the best performing model previously, now using the GloVe embeddings.

Simple CNN (GloVe) Results: Precision: 0.903759, Recall: 0.996683, F1 Score: 0.947950

We see that these are the best results achieved so far and show that the GloVe embeddings are far more effective.

Complex Model As it had performed better overall than the simple model when dealing with Word2Vec embeddings, we decided to revisit the more complex, double input model and test it using the GloVe embeddings.

Complex CNN (GloVe) Results: Precision: 0.947702, Recall: 0.991708, F1 Score: 0.969206

We see that these are the best results achieved overall. Despite performing worse on precision for W2V embeddings, the complex model performs better than the more simple model for GloVe embeddings and achieves the highest F1 Score of all the systems tested.

2.5 Discussion

From our results we can see that the Convolutional Neural Network with GloVe embedding is clearly best suited to this task in terms of precision and recall. With the more complex model working best overall, however the running time taken to train is not insignificant even for the protracted sample used for training. Conversely, the random forest model performs well but has the downside of taking much longer to classify new instances.

Model	Precision	Recall	F1 Score
Baseline	0.493	0.496	0.494
Naive Bayes	0.645	0.588	0.616
SVM (Linear)	0.684	0.693	0.689
SVM (RBF)	0.709	0.728	0.718
Perceptron	0.655	0.769	0.707
K-nearest Neighbours	0.828	0.848	0.838
Random Forest	0.956	0.853	0.902
Simple CNN (W2V)	0.883	0.879	0.881
Complex CNN (W2V)	0.828	0.848	0.838
Simple CNN (GloVe)	0.904	**0.997**	0.948
Complex CNN (GloVe)	**0.948**	0.992	**0.969**

Table 1: Classification results

The drawbacks of all our systems are however that they will become less and less accurate over time due to Heap's law. For the systems in 2.3, the autoencoder used to reduce the dimensionality of the data is unable to handle new words in the corpus so they would have to be discarded. While the same problem is prevalent for the systems in 2.4 due to the word embedding that takes place. This could cause problems as language evolves and the prevalence of words from our original corpus decreases in news articles and thus to our classification system the data becomes more noisy with unknown words. This could be solved with periodic retraining of the autoencoder and embeddings on the new data such that the latent space is updated. This would also involve retraining the random forest model and convolutional neural networks on the new latent space. We could also try training the autoencoder on many more news articles in the first place to decrease the effect of a growing corpus, this is advantageous as training the autoencoder does not require labelled data whereas retraining our classifiers does. For the convolutional networks, we could also use a different type of embedding such as FastText [4], which creates a new vector for unseen words rather than discarding them.

3 Sentence level classification

3.1 Data description

For the task of classifying sentences as propagandistic or not the datathon organizers have constructed a very high-quality, no-noise(or very low noise) dataset, which consists of sentences and a label(propaganda or not). 300 articles have been analysed and each sentence has been annotated by professionals - independently by 2 people and revised by a third person. Each sentence is checked for propagandistic techniques. The total number of propagandistic techniques have been revised so as to include the number of most frequent ones, some techniques have been grouped or definitions have been revised. The annotation and

definitions have been done by the Qatar Computing Research Institute and A Data Pro - a professional annotating company.

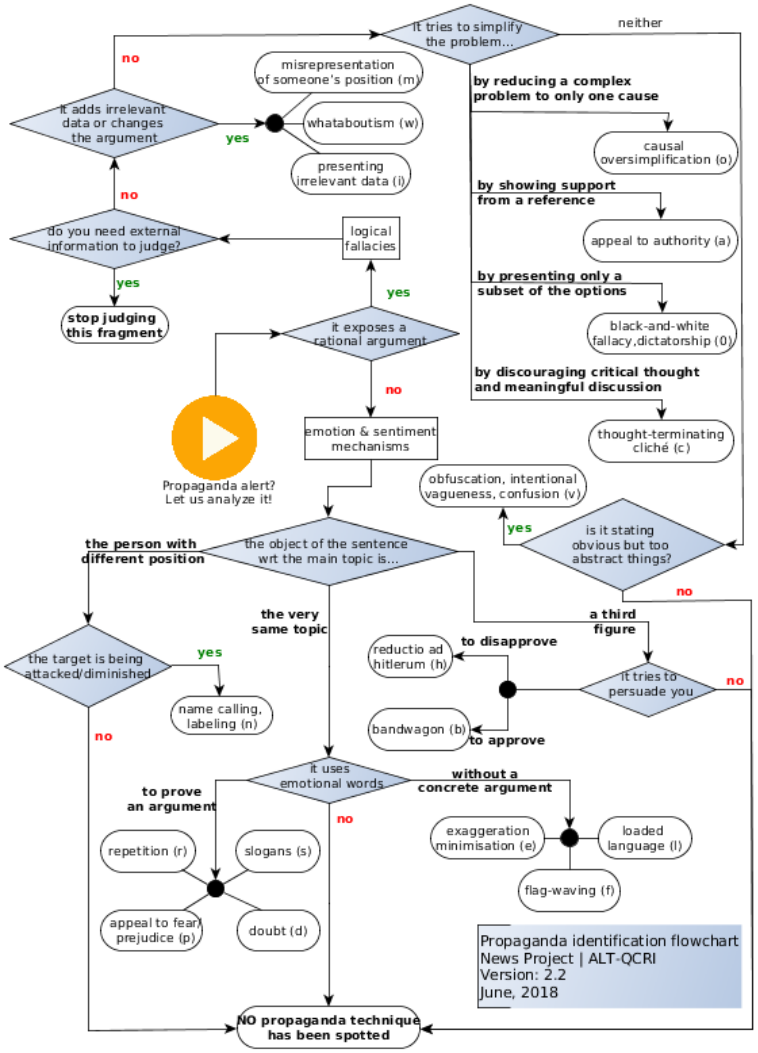

Figure 4: Propaganda identification flowchart. [1]

Examples of techniques include presentation of irrelevant data, misrepresentation of someone's position and whataboutism. For the full list refer to the website [3]. In total, there are 14263 sentences, 3938 out of which are propaganda. Both the article level and the sentence level datasets can be found on the datathon's website.

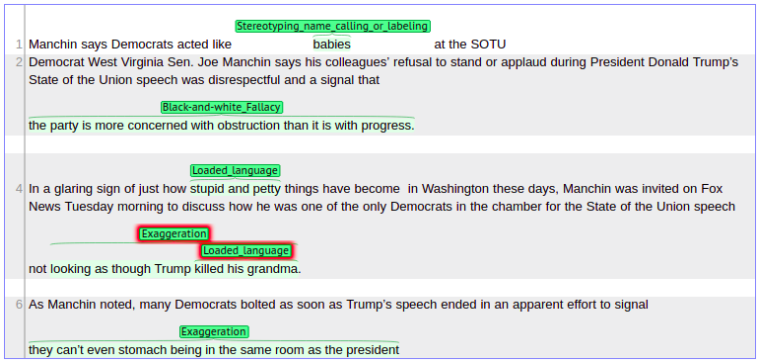

Figure 5: Propagandistic techniques examples. [1]

3.2 Experiments

3.2.1 Methodology and Feature extraction

In addition to building a well-performing model, we tackled the task of comparing the performance of various low-level features in order to investigate properties of propaganda. This would help with finding attributes which might be useful in supplementing newer language representation features(Word2Vec, Glove, FastText, Doc2Vec etc.) or more traditional ones(such as Tf-Idf). The following features were extracted for analysis from the sentences:

- **Total Counts**
  Propagandistic sentences might be deliberately long or obfuscated in order to confuse the reader. The total counts feature are 2 columns counting the total number of words and characters in a sentence.

- **Stop Words**
  Number of stop words in a sentence. The repetition(repeating the same message continuously) fallacy may possibly be correlated with the number of stop words in a sentence(for example - repetition of "and").

- **Unique Words**
  The number of unique words in a sentence.

- **Lexical features**
  Information about various lexical attributes:
  1. Number of capitalized words in a sentence. It's possible that propagandistic sentences contain more authoritative names helping to identify the Appeal to Authority fallacy or capitalized rhetoric to drive a point home, possibly identifying slogans.
  2. The number of adjectives and adverbs. Sentences containing a high number of adjectives and adverbs may indicate high level of subjectivity.

3. The number of proper nouns - may relate again to the Appeal to Authority fallacy.
4. Other features such as counts of other POS(nouns, verbs, determiners, prepositions, personal pronouns etc.), the average word length, number of hyphened words, number of quotes, the number of very short and the number of very long words and others.

- **Type of Sentence**
  Whether a sentence is a statement, question or an exclamation.

- **Loaded Words**
  Normalized number of words in a sentence that occur in a list of loaded words. The list has been manually compiled and contains words that have a high emotional content or degree of subjectivity, such as "stupid", "frightening", "shatter", or more politically, socially or debate-oriented such as "extremism", "populism", "white privilege", "sjw", etc.

- **Exclamation Words**
  Number of exclamations such as "wow", "uh", "oh", "ah", and others.

- **Lexical Character Counts**
  Counts of characters such as digits, commas, parentheses, apostrophes, paragraphs etc.

- **Readability Features**
  Various scores indicating how easy a text is to read. Standard ones include the Flesch Reading Ease, the Smog index, and Coleman–Liau index.

- **Ambiguous words**
  The sum of the number of synonyms of the words found in the sentences using WordNet. Ambiguity may also have high correlation with propaganda.

- **Emotions**
  Emotions extracted from IBM's Natural Understanding API. The degree of joy, fear, disgust, anger and sadness is available for each sentence.

- **Sentiment 1**
  The sentiment of the sentence with respect to the synonyms in WordNet of its constituent words.

- **Sentiment 2**
  The sentiment of the sentence using an Unsupervised Sentiment Neuron trained on Amazon reviews by Open AI. [10]

- **Subjectivity and Polarity**
  Using the Python library TextBlob, we extract subjectivity and polarity scores for the sentences.

3.3 Baseline to compare against the features

Using these features we gathered results by splitting the training and test set with an 80/20 split, such that the proportion of classes in both sets is the same. We evaluated the performance of the different features using a Linear Support Vector Classifier, due to its fast training times, with a default regularization parameter of 1, and balanced class weights. As the data is skewed, we concentrate more on the f1 score, than the accuracy. F1 scores were gathered again on the minority class.
As before, precision and recall for each feature are also plotted.

Feature	Precision	Recall	F1 Score
Total Counts	0.0	0.0	0.0
Stop Words	0.39	0.39	0.39
Unique Words	0.0	0.0	0.0
Lexical Features	0.34	0.63	0.44
Sentence Type	0.29	0.82	0.43
Loaded Words	0.38	0.40	0.39
Lexical Character Counts	0.31	0.81	0.45
Readability Features	0.42	0.28	0.34
Ambiguous Words	0.28	0.55	0.37
Emotions	**0.37**	**0.66**	**0.48**
Sentiment 1	0.31	0.46	0.37
Sentiment 2	0.32	0.54	0.40
Subjectivity and Polarity	0.34	0.52	0.41
Exclamation Words	0.33	0.0	0.0

Table 2: Baseline SVM Classifier against the different features

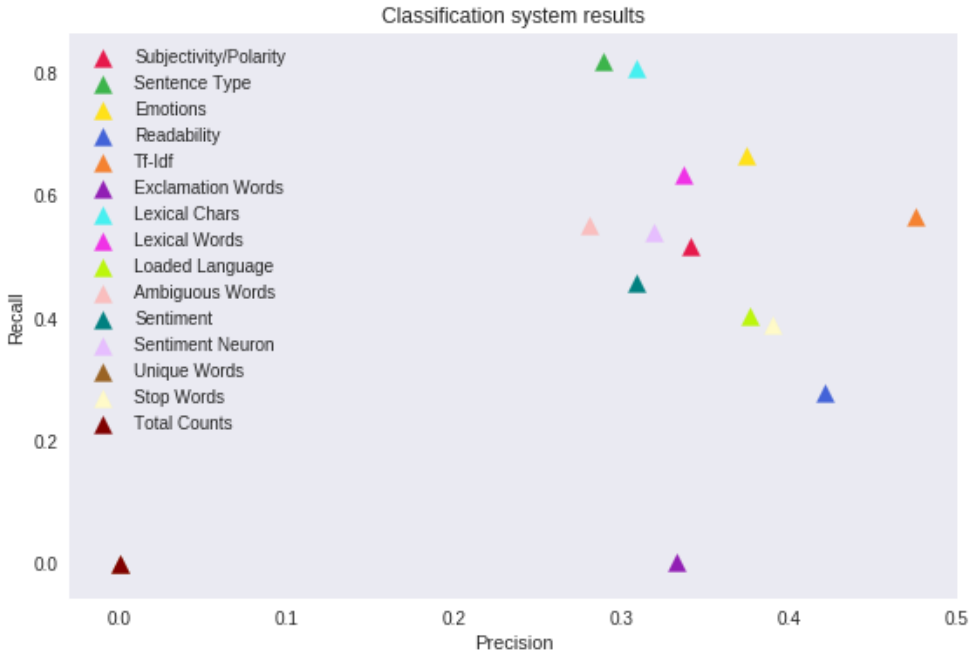

Figure 6: Baseline classification. This was used to select the most important feature. Tf-Idf performed only slightly better than emotions.

3.4 Other models compared against the baseline

3.4.1 Data pre-processing and splitting the data

Stop words were applied to the tokenized sentences, followed by Porter Stemming. They were then converted to their respective Doc2Vec Embeddings (having 10 dimensions). A count of all the thirty five nltk pos tags (like NN, CC) was then appended to every sentence before the preprocessing of the data. This was done to not lose the grammatical context of the original sentences. The sentence no of the sentence in the corresponding article was also given. The article no was not included as it did not convey any special meaning.
As we saw before, that that usage of emotional attributes tend to model well, we also append the numerical values of the 5 emotional attributes to the data. This appended data array was then passed to a two hidden layer neural network. The data was split into two parts, training and validation, in the ratio of 80:20. The ratio of non-propaganda to propaganda labels in both the sets were kept similar (27:73), to represent an actual scenario.

3.4.2 Using Logistic Regression with emotions

Logistic Regression was applied to the appended data. The PosTag count values, and emotions were used as features
The following scores were obtained:

```
For training

Precision: 0.680821581280176
Recall: 0.5717098226890684
F1: 0.5653396669692063
For validation

Precision: 0.6653280330537124
Recall: 0.5679798707817447
F1: 0.5656317149886332
```

This is an a definite improvement to our baseline

3.4.3 Using K-Nearest Neighbours Classification with Emotions

K-Nearest Neighbours with neighbours of size 3 was applied.The PosTag count values, and emotions were used as features:

```
Precision: 0.8150979276287147
Recall: 0.8853065128089584
F1: 0.8219652103978099
For validation

Precision: 0.5541095683137556
Recall: 0.5690381348297253
F1: 0.5440381917889026
```

There is a lot of over fitting. With a neighbour size of 2, there is underfitting, and increasing the neighbour size, does not improve the validation score.

3.4.4 Using Random Forest Classification with Emotions

The PosTag count values, and emotions were used as features
Random Forest Classification only seems to over-fit the data more:

```
For training

Precision: 0.9335147655462391
Recall: 0.9190258213619221
F1: 0.9258884661779716
```

```
For validation

Precision: 0.5775846133343651
Recall: 0.5696832608303946
F1: 0.572331894101405
```

3.4.5 Using Doc2Vec, PosTags and Neural Network

The initial layer had 175 neurons, with a dropout of 0.5. An activation function of relu was given, to eliminate the negative values. The second layer also had 175 neurons, with a dropout of 0.5 and relu activation. For the output layer, we used the sigmoid function. Binary cross entropy was used as the loss function and the rmsprop optimiser with the default learning rate was employed to optimise the algorithm in 10 epochs. The scores of the model are as follows:

```
For training

Precision: 0.7191060473373541
Recall: 1.0
F1: 0.82643346677842
Accuracy: 0.7191060473373541

For validation

Precision: 0.7430774623203645
Recall: 1.0
F1: 0.8451232763077141
Accuracy: 0.7430774623203645
```

As we can use, the neural network was under fitting with the training data, but with a vastly improved precision, recall and f1. Trying to tune the hyperparar-maters, did not seem to affect the model. Using a single or even a three layer neural network, did not help with the under-fitting, but we had found a model we looked to improve upon.

3.4.6 Using Long Short-Term Memory with Doc2Vec and PosTags

We then proceeded to pass our appended data through a LSTM (Long Short Term Memory). We passed our data through an LSTM having 175 neurons, and relu activation function. We then flattened our data and passed it through a one layer hidden layer, similiar to the one that we used earlier. As before, we fit the neural network using the balanced class weights.
The scores of the model are as follows:

```
For training
```

```
Precision: 0.7242916825254375
Recall: 0.9907047804983743
F1: 0.8270446840866317
Accuracy: 0.7234005258649614

For validation

Precision: 0.7481575139229899
Recall: 0.989601422726711
F1: 0.8444051558704824
Accuracy: 0.7441289870311952
```

We managed to improve the model, but there was still some under-fitting

3.4.7 Using the Keras embedding in our LSTM

We then decided to use the keras embedding for text, before passing it through our LSTM. We used a the fit_on_texts function on our original sentences only, and padded the data to have a dimension of 50. We then passed the padded sequence to the embedded layer, having a vocabulary size of 1200 and output dimension of 100. We then passed this data to a LSTM, with the same specifications as above.
We padded our validation data in the same value, and then tested our trained model on it. The scores of the model are as follows:
or training

```
Precision: 0.7594832291632342
Recall: 0.9853250294584229
F1: 0.8501454054734667
Accuracy: 0.7707274320353341

For validation

Precision: 0.7501297164205664
Recall: 0.9583619482426153
F1: 0.8333367870182394
Accuracy: 0.7308096740273397
```

We not only managed to improve our model, but we overcame the problem of under-fitting

3.4.8 Using Convoluted Neural Network and LSTM With Word Embeddings

As we saw before, the keras embedding on padded sequences, work better than doc2vec. As a final comparison, we first used this embedding, and then passed

our data to a 1 dimensional convoluted neural network, and an LSTM afterwards. A filter of 175, kernel-size of 5 and relu activation function was used. A pool size of 4 was used. The data was then passed through a LSTM (having 175 neurons and relu activation function), before being passed to the output layer having sigmoid activation function. As before binary cross entropy was used as the loss function, and RMSProp was used to optimise the loss value. The scores of the model are as follows:

```
For training

Precision: 0.7596727901127754
Recall: 0.9714273364161525
F1: 0.8459418138634417
Accuracy: 0.7670464504402421

For validation

Precision: 0.7497233782907641
Recall: 0.9513095911870368
F1: 0.830600403586146
Accuracy: 0.727304591657904
```

3.5 Results

Feature	Precision	Recall	F1 Score
Baseline with Emotions	0.37	0.66	0.48
Logistic Regression with Emotions	0.66	0.56	0.56
K-Nearest Neighbours with Emotions	0.55	0.56	0.54
Random Forest Classification with Emotions	0.57	0.56	0.57
Neural Netwok with Doc2Vec & PosTags with Emotions	0.74	1	0.845
LSTM with Doc2Vec & PosTags and Emotions	0.74	0.98	0.84
LSTM with Keras Embedding Layer	0.75	0.95	0.83
CNN & LSTM with Keras Embedding Layer	**0.74**	**0.95**	**0.83**

Table 3: Validation scores of different models

The CNN & LSTM model using Keras Embedding works the best, dealing with the previous under fitting, and not over-fitting the data. In this case, we used only 3 epochs, as we did not want the data to over-fit. Alternatively, we could have used early-stopping to deal with this problem, and get a better generalised result.

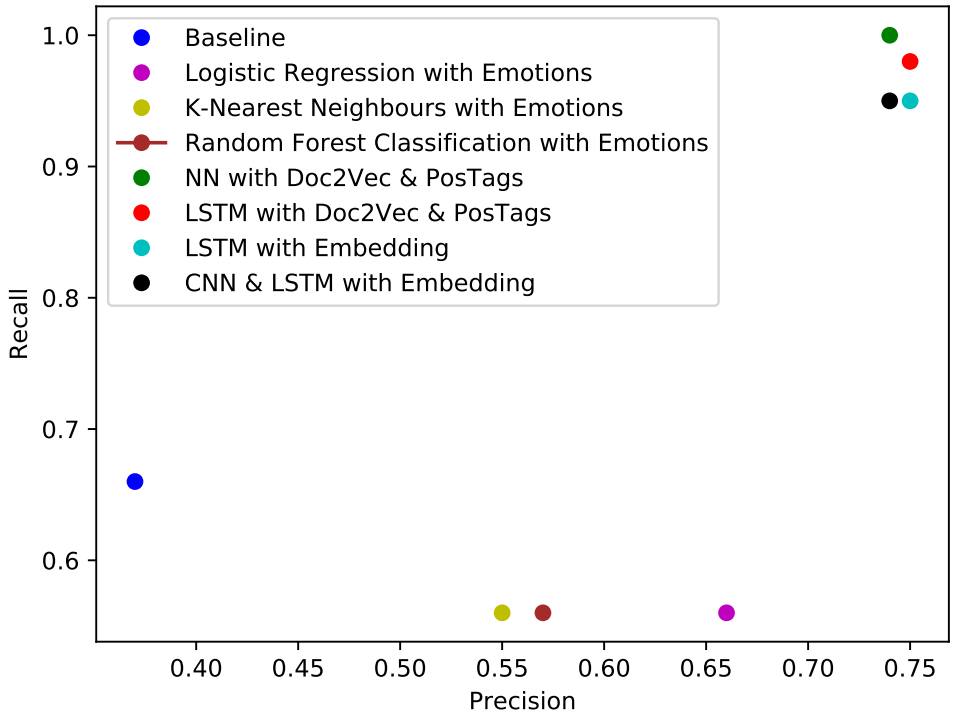

Figure 7: Classification system results

3.6 Discussion

We observe that using the Keras Word Embedding, and using CNN and LSTM to the model works well. One of the early challenges with the data, was the ratio of propaganda:non-propaganda (around 27:73). We then decided to not use accuracy as a metric, since f1 would convey more about our model in this case.

Lexical features was the second best performing feature. Counting a variety of characters might bring additional information to a model that is trained using only word or document representations.

Other useful features might be counts of the various parts of speech(lexical features), the type of sentence and the subjectivity/polarity score. Appending these features to our data set (appended with emotions and tags), and then passing them through a neural network might help with the under-fitting that we experienced earlier.

We could also have passed our appended data set through the embedding layer, which may have learned more things and generalised better. Alternatively, we could have tried using a combination of different features and different word embedding like glove.

4 Individual Contributions

4.1 Max Girkins - s1461092

Implementation of all the document level classification models found in Section 2.3, including preprocessing, dimensionality reduction, fitting the various models and running all the experiments and discussion of the best systems.

4.2 Philip van Biljon - s1545259

Desinging and implementing all the document level classification models found in Section 2.4, including preprocessing the text, training the embeddings, exploration of different models and fine-tuning them and discussion on the best systems for the task.

4.3 Deyan Petrov - s1453370

Proposal of the project idea and motivating the problem. Preparation and preprocessing of the sentence level data from different files into 1 dataframe. Finding the annotation methodology. Removal of the empty lines classified as non-propaganda. Extracting low-level features and comparing their performance. Finding the top-performing features.

4.4 Tanya Sanatani- s1835874

For the sentence-level classification, making and implementing various models to improve on the baseline as found in section 3.4

References

[1] Annotation methodology. http://propaganda.qcri.org/annotations/definitions.html. Accessed: 2019-02-01.

[2] Hack the news datathon. https://www.datasciencesociety.net/hack-news-datathon-case-propaganda-detection/. Accessed: 2019-02-01.

[3] Propagandistic techniques. http://propaganda.qcri.org/annotations/definitions.html. Accessed: 2019-02-01.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[5] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *CoRR*, abs/1710.05381, 2017.

[6] Jeremy Jordan. *Autoencoder diagram.* Mar 2018.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[10] Alec Radford, Rafal Józefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *CoRR*, abs/1704.01444, 2017.

[11] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[12] Ewan Klein Steven Bird, Edward Loper. NLTK: The natural language toolkit, 2001–.

[13] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.