11761 - Language and Statistics Course Project Detection of Real and Fake Articles

Anusha Prakash, Anushree Kumar, Srividya Pranavi, Varun Bharadhwaj

1 Introduction

The goal of this project is to distinguish between real broadcast news articles and fake ones. The fake articles were generated from a 3-gram language model trained over a 100MW corpus that is similar to the one from which the real articles were drawn.The real and fake articles can be of various lengths and the aim is to classify them accurately as real or fake along with the confidence (probability) of the classification. We have access to a training dataset of 1000 articles - 500 real and 500 fake articles and a development dataset of 200 articles - 100 real and 100 fake articles. We also have access to a 100MW broadcast news corpus. The performance is based on a **hard metric** - accuracy of the classification decision and a **soft metric** - perplexity of correct label.

By looking at the articles, although the task seems trivial for a human who can easily identify the differences and shortcomings of fake articles with respect to real articles based on his knowledge of Language, Grammar and Semantics, the same task is not very straightforward for a machine. It is essential to teach the machine the various aspects that distinguish these two types of articles by choosing the features that best reflect the difference between trigram model generated fake articles and real ones.

In this report, we model a various features spanning Semantic, Syntactic, Statistical and Language Model based features. We analyze the relative performance of all features as a whole as well as observe individual clusters of features from the sets enumerated above. Grid search is then performed on a pipeline involving Recursive Feature Elimination to select the best features among our set and evaluating our model's performance over 4 classifiers: Logistic Regression, Adaboost Ensemble Classifier, Random Forest Classifier and Gaussian Naive Bayes.

Out of these classifiers, we observe that **Adaboost** and **Logistic Regression** give the best classification results on the development set with development accuracy of **0.93** and **0.92** respectively, while the log-likelihood was in the range of **-0.7** and **-0.14** respectively. We choose the Logistic Regression model as our final model considering both soft and hard metrics for best performance.

The rest of the report is organized as follows : Section 2 speaks about the data preprocessing and balancing, Section 3 speaks about the various features experimented with, Section 4 speaks about the feature selection performed, Section 5 speaks about the various classifiers employed and experimented with and finally Section 6 details the experiment results and final metric values obtained.

2 Data Preprocessing - Balancing The Training Dataset

For Machine Learning algorithms to perform best, it is essential that both the train and test data come from identical and independent distribution (i.i.d). But the Training and Development Set that we have, clearly do not come from the same distribution since the distribution of number of sentences in each article is very different in these two sets.

Since the composition of the test data is given to us, we wanted to leverage that information by creating a balanced training data that is more like the test data. This gives an impression that both training and test data come from the ‘same’ distribution. We adapted an algorithm for balancing the training dataset by chopping the articles to contain same length distribution as test dataset. We employed a sampling mechanism where articles according to the expected length distribution are sampled and if the length of the article is more, the article is trimmed and the remaining partial article is added back to the original pool. This is repeated iteratively until all our original training data is converted to the new train data having same distribution as test dataset.

After running the algorithm completely, the initial 1000 articles in the training dataset was transformed into 9269 articles, with the articles’ length distribution in the training set and the development set being identical. The following table illustrates the distribution of the new training dataset in detail :

Balanced Training Dataset

# of sentences	**Real articles**	**Fake articles**	**Percentage**
1	**940**	**914**	**20**
2	**478**	**449**	**10**
3	**453**	**474**	**10**
4	**478**	**449**	**10**
5	**471**	**456**	**10**
7	**461**	**467**	**10**
10	**468**	**459**	**10**
15	**463**	**463**	**10**
20	**469**	**457**	**10**

Our intuition was that by creating and using this new train dataset, we can provide the machine learning algorithm with a better sense of how the test data will be, and as a result it’s accuracy will be more realistic and the model can generalize better. We were able to prove this experimentally, wherein the 5-Fold mean CV accuracy was 98% on our old train dataset and overall accuracy was 92% on our test dataset, while 5-Fold mean CV accuracy was 90% on our new train dataset and overall accuracy wa 92% on our test dataset. This error between the between train and test data in the new train data case is a better estimate of the expected error on any new dataset. Here we can also observe that the train accuracy is lesser than the test accuracy as the train : test division is not a standard 80 : 20 percentage.

3 Feature Extraction

The bigger goal here is to determine which features best discover and exploit the deficiencies of a trigram model like lack of global coherence and context, grammar violation etc. in order to better classify the articles as real or fake. The various features we experimented with can be classified as follows :

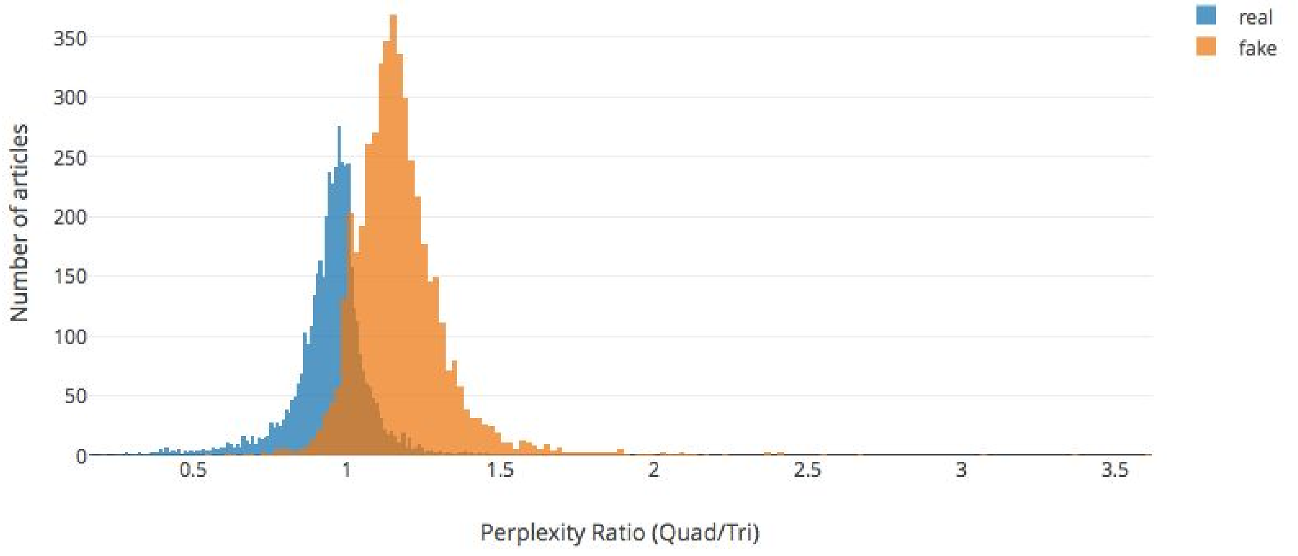
- Language Model Features
- Statistical Features

- Semantic Features
- Syntactic Features

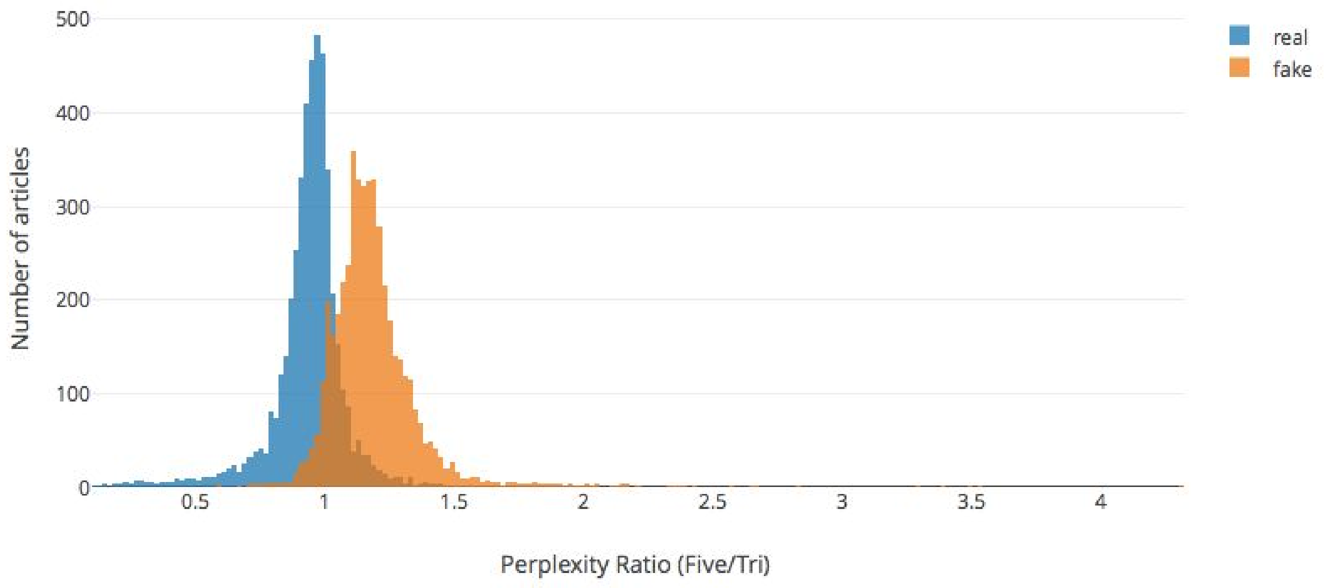
3.1 Language Model Features

3.1.1 Text Language Model Perplexity Ratios

Since, it is mentioned that trigram model is used to generate ‘fake’ articles, a higher gram language model should be able to capture the long distance word dependencies better in a real article. Both ‘real’ and ‘fake’ articles would have similar and low perplexity score on a trigram model as the articles are themselves generated from a trigram. When we consider quadgram language model, the real articles would have lower perplexity score than the fake articles. This implies that the ratio of quadgram to trigram perplexities would be higher for a fake article than for a real article. In other words, this ratio is similar to computing the likelihood ratio of an article w.r.t the trigram and quadgram models. A quad-gram language model is a better estimator of real sentences than a trigram model. The histogram plot of the ratio of quad-gram perplexity to trigram perplexity is shown below.

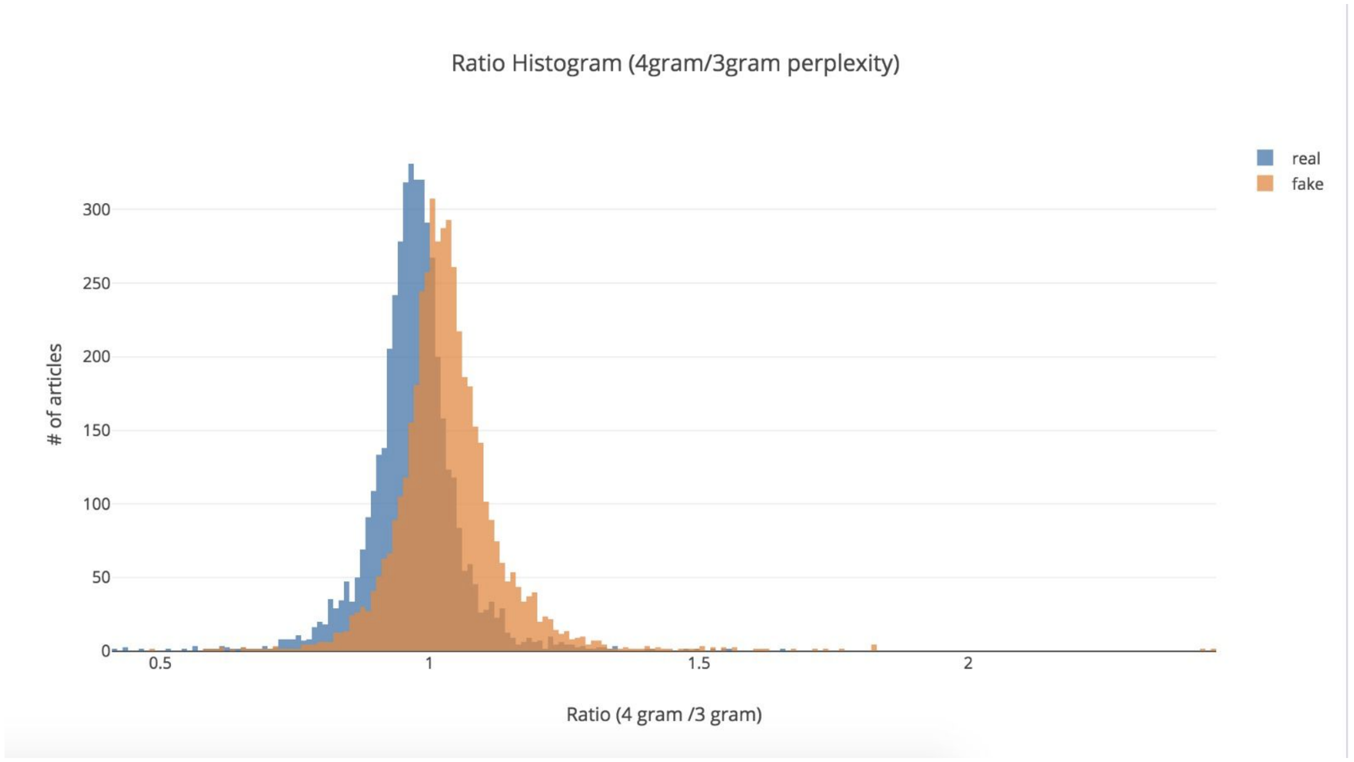


As we go to higher gram, we move closer to the truth. The perplexity is lower for a quadgram and 5-gram for real articles. Hence the ratio would be higher for fake articles and lower for true or real articles. Out of curiosity we tried for 6-gram language model as well. These models are trained on the 100 Million word corpus using ken-lm package which does Kneser-Ney smoothing. The histogram plot of the ratio of 5-gram perplexity to trigram perplexity is shown below.



3.1.2 POS Language Model Perplexity Ratios

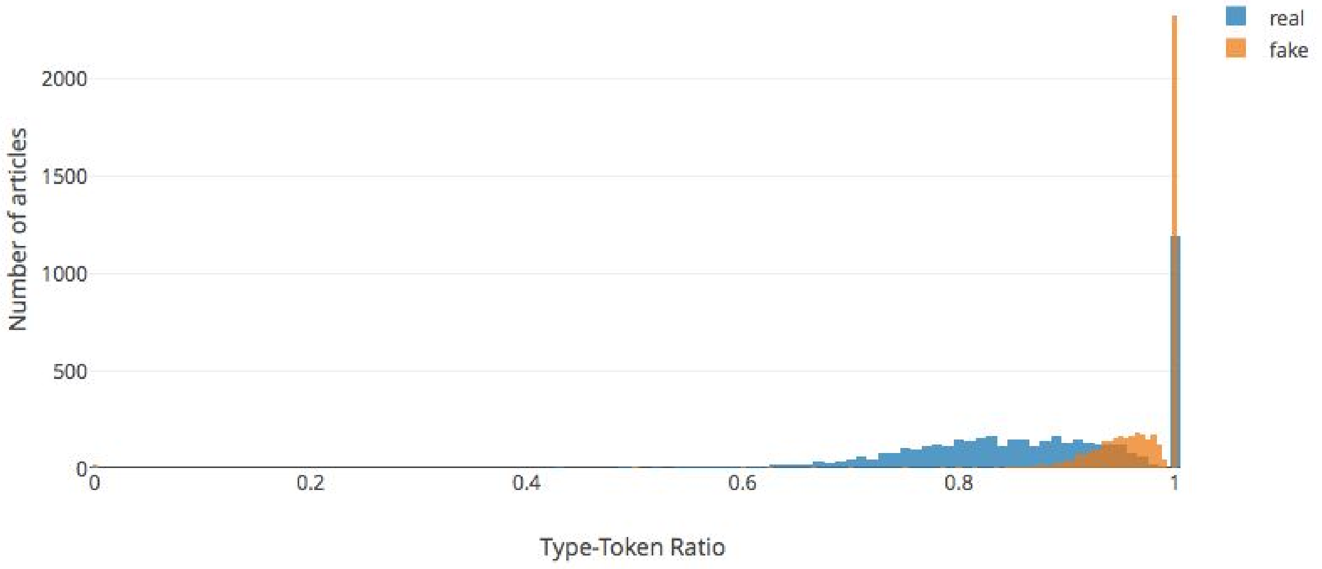
To further boost our model, we also built POS language models after POS tagging the 100 Million word corpus using bllip-parser. We employed the ken-lm package which does Kneser-Ney smoothing for building the language model. The intuition for the perplexity ratio for text language model remains the same for POS language model as well. The ratio of quadgram to trigram perplexities would be higher for a fake article than for a real article. The histogram plot of the ratio of quad-gram perplexity to trigram perplexity is shown below.

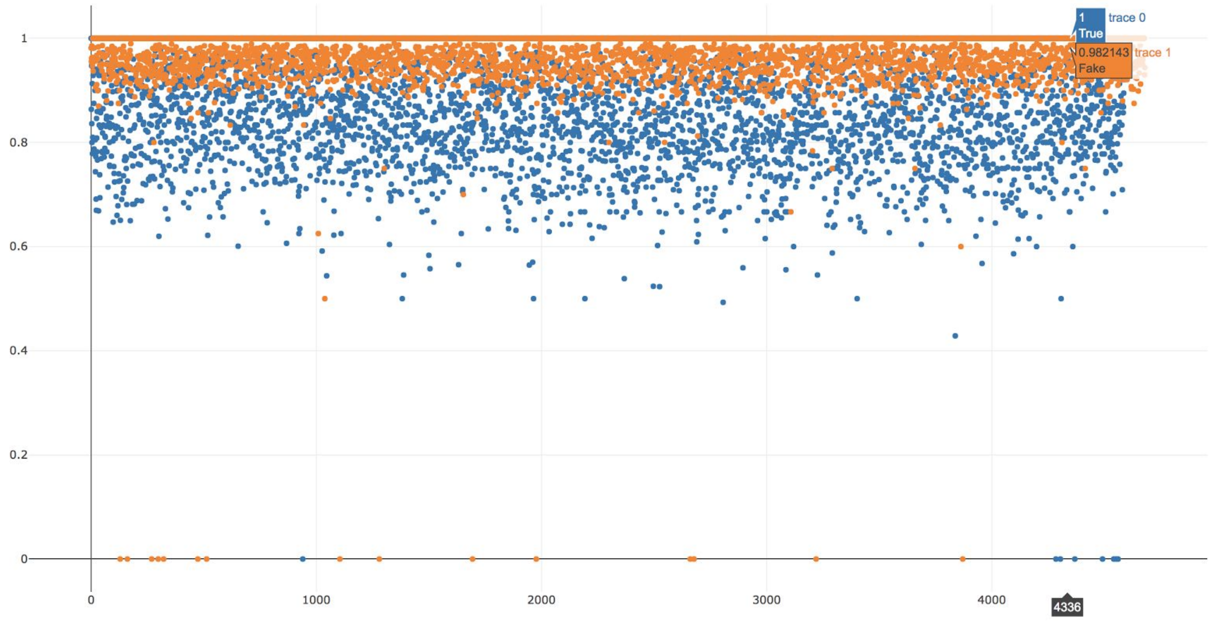


3.2 Statistical Features

3.2.1 Type to Token Ratio (TTR)

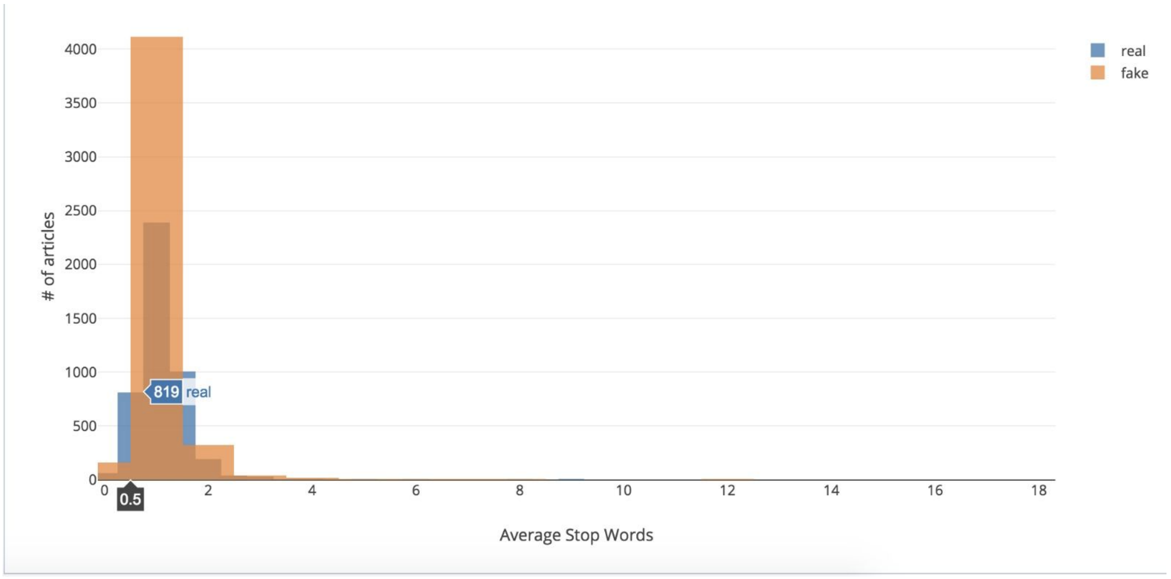
On average a human vocabulary is limited and its is much lesser than the complete Vocabulary of a language. When an article is produced by human i.e real article, the number of unique types is expected to be lesser than the case produced by machine. As humans we tend to use same words repeatedly, with less variation. So, we calculated number of unique types and total number of tokens in each article and calculated Type to Token ratio, after removing the stopwords from the articles. This ratio is expected to be higher for fake articles. The pictorial depiction of the separation / classification can be seen from the histogram plot and scatterplot below.





3.2.2 Stop Words to Content Words Ratio

Stop words are non-informative words that connect the content words. Our intuition was that since a machine cannot really distinguish between stop words and content words, the ratio of stop word to content word distribution in fake articles will not be uniform and will be in fact be higher than real articles, as the fake article does not take into consideration global grammar and semantics and as a result may use stop words more often than actually required / makes sense. But in real articles, the stop words are not randomly used in the sentence, but usually follow a uniform ratio with respect to content words. We used a list of 207 stop words to test the same intuition. The plot for the feature is shown below. But as it can be seen from the plot, this feature was not distinguishing between real and fake articles. It means that although global coherence is missing in fake articles, this ratio distribution is still not much different from real articles.



3.2.3 Rare Words tf/idf

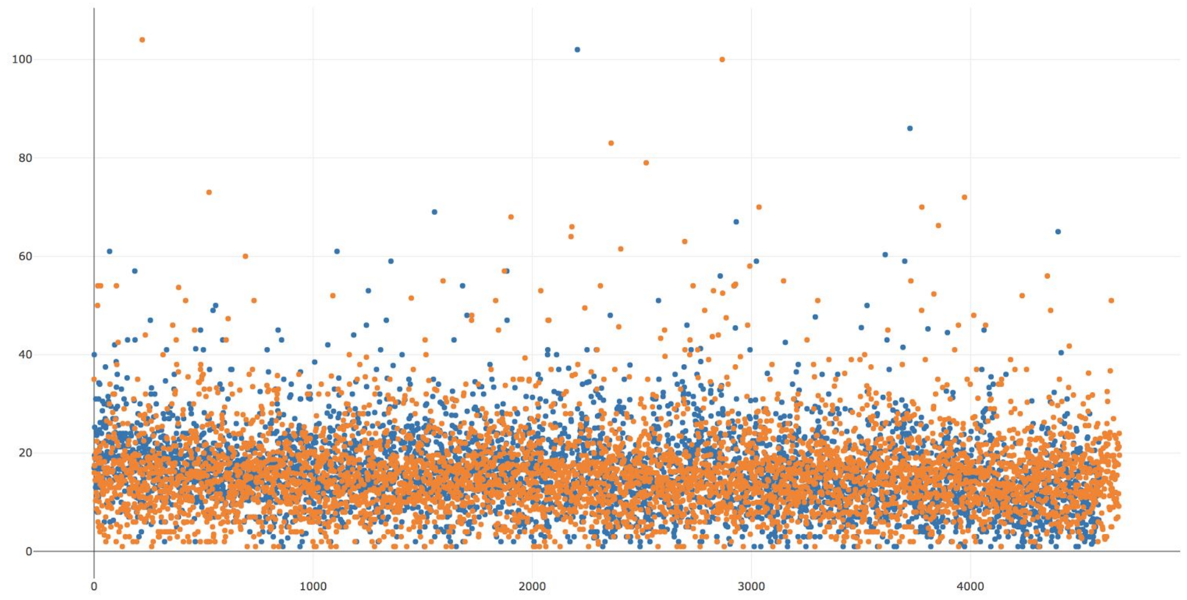
When rare words such as proper nouns etc. occur more than once in an article, it reflects the semantic coherence of the article and is most likely a real article. Such rare words are not as likely to occur in fake articles. From 100MW word corpus, we set a threshold of 1000 and selected all the words that occurred lesser than 1000 times as rare words. Then in the article, for any rare word present more than once, we calculated a tf/idf score based on the frequency of occurrence of the word in the article and its frequency in the original 100MW corpus and summed up this score and further normalized it. The drawback we observed after calculating this feature was that for smaller articles, like single / double sentence articles, neither the real nor fake articles contained any rare words, and as a result the value of this feature was zero in a lot of cases. Due to this, the feature was unable to distinguish between real and fake articles and as a result we decided not to use this feature.

3.2.4 Average Sentence Length

Each sentence in the article was tokenized and the sentence length was calculated to be the number of tokens of the sentence. This was normalized across the article by dividing by the number of sentences.

$$\text{Average Sentence Length} = \sum_{Sentence\ \varepsilon\ Article} (\text{\# Tokens in the Sentence}) / (\text{\# Sentences})$$

Our assumption was that there may be a distinction with respect to the average sentence length between fake and real articles as human generated real sentences tend to be shorter than machine generated fake sentences. Unfortunately we couldn't find much distinction between the two and the same is depicted in the scatterplot below. We feel it may be because while generating the fake articles using the trigram model the sentence length distribution was also learnt and hence followed by the machine.



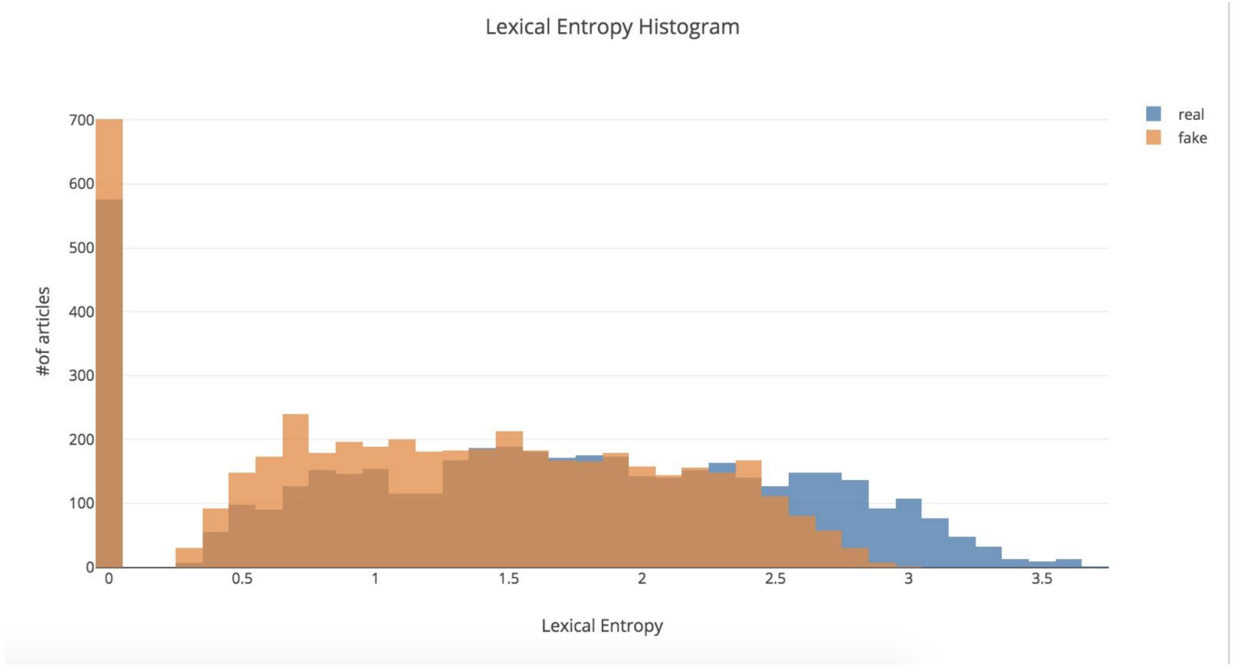
3.2.5 Entropy

Every author has a unique style of writing and lexical entropy best captures the structural diversity of the author language. This is the rationale behind choosing this feature. We calculated entropy as follows:
Let W_i be the no. of unique words that occurs i times. Let N be the total size of the Vocabulary and H be the lexical entropy given by :

$$\text{Let } p_i = \frac{i * w_i}{N}$$

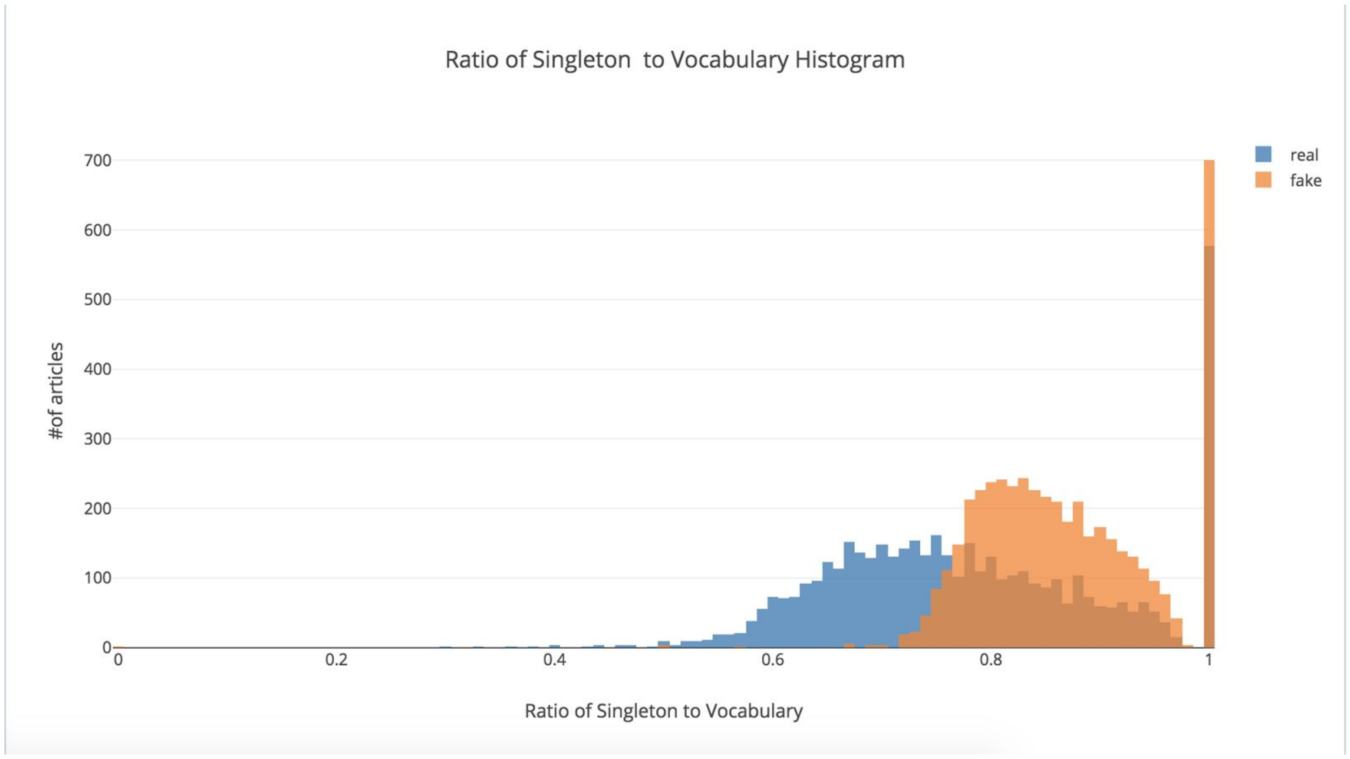
$$H = -\sum_i p_i * log(p_i)$$

The histogram plot below depicts the separation of this feature for real and fake articles along with the number of articles with that particular lexical entropy value. We are slightly surprised that it didn't add as much value to the classification as we initially thought, but it did improve the accuracy to certain extent.

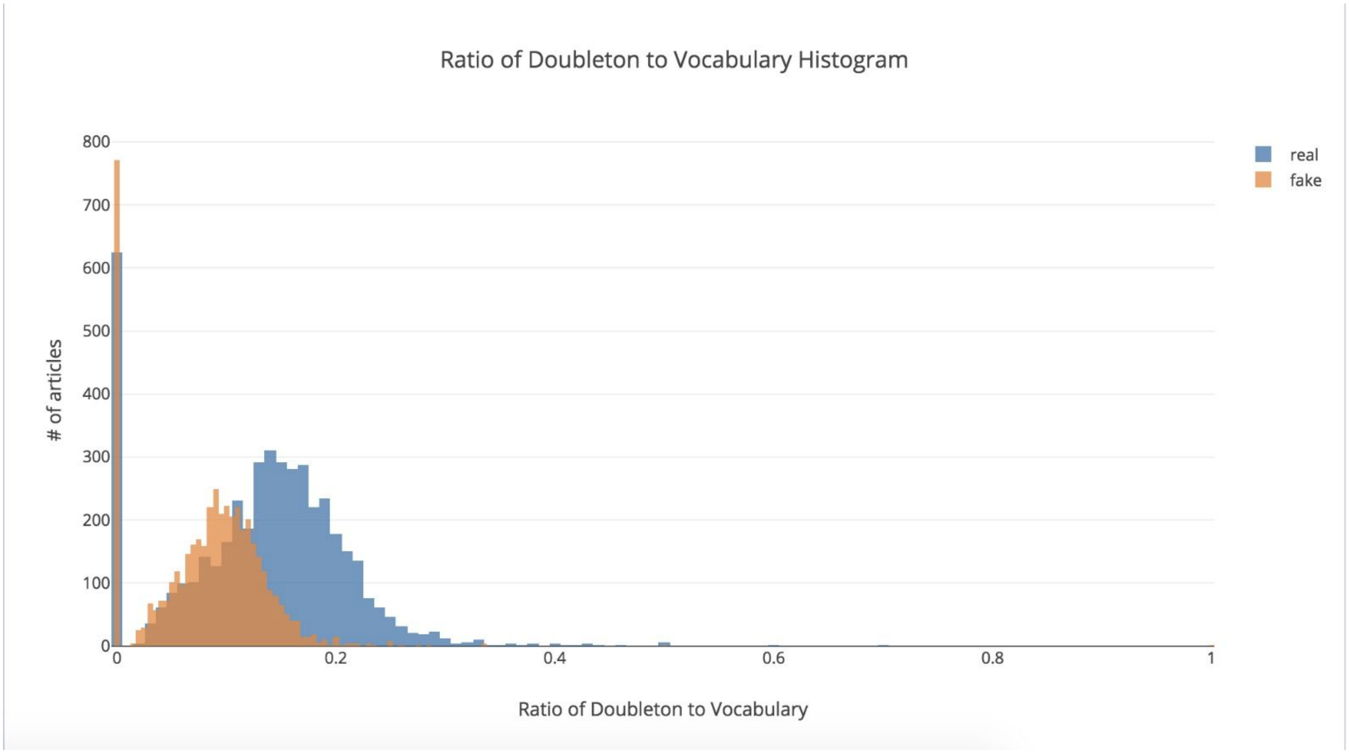


3.2.5 Proportion of Singletons and Doubletons

We tried calculating the richness of the vocabulary by calculating the proportion of singletons in the vocabulary. Basically, this measures how many rare words are being used in the article. The intuition is that a real article would have lesser singletons compared to the fake articles and hence the ratio is less for real articles compared to the fake articles. The histogram below depicts that this intuition is correct.



We tried calculating the richness of the vocabulary by calculating the proportion of singletons and doubletons in the vocabulary. Basically, these measure how many rare words are being used in the article. The intuition is that a real article is semantically coherent. Since the whole article would be about a specific topic, words related to the topic would occur more often in real than in fake. Therefore the number of doubletons are more in real as compared to fake. The histogram below depicts this intuition



3.3 Semantic Features

3.3.1 LDA (Latent Dirichlet allocation) Topic Modeling

To extract the topics in the articles, we used gensim toolkit which internally uses LDA to identify the topics in an unsupervised manner. We identified 100 topics and 10 most frequent words for each topic from the 100MW corpus. We then used the functionality of get_topics() for each article. Our intuition was that since the real articles are more coherent and talk about a specific topic, the probability of the top most topic will be higher for real articles compared to fake articles where the content is not very coherent and the article is not talking about any specific topic. We discarded this approach as LDA took too long to train and get the topics out and it did not add much improvement in the performance either.

3.3.2 LSA (Latent Semantic Analysis)

Real articles tend to be coherent with respect to the topic under discussion. In order to capture this aspect, we performed latent semantic analysis to analyze the relationship between the sentences and the terms in the article. A document-term matrix was constructed where columns represented the sentences and rows represented the unique words in a given article. Each element of the matrix was the term frequency of the word in the sentence.To obtain word to word correlations and document to document correlations , we transformed the term matrix D as follows:

<u>Word to word correlations</u>

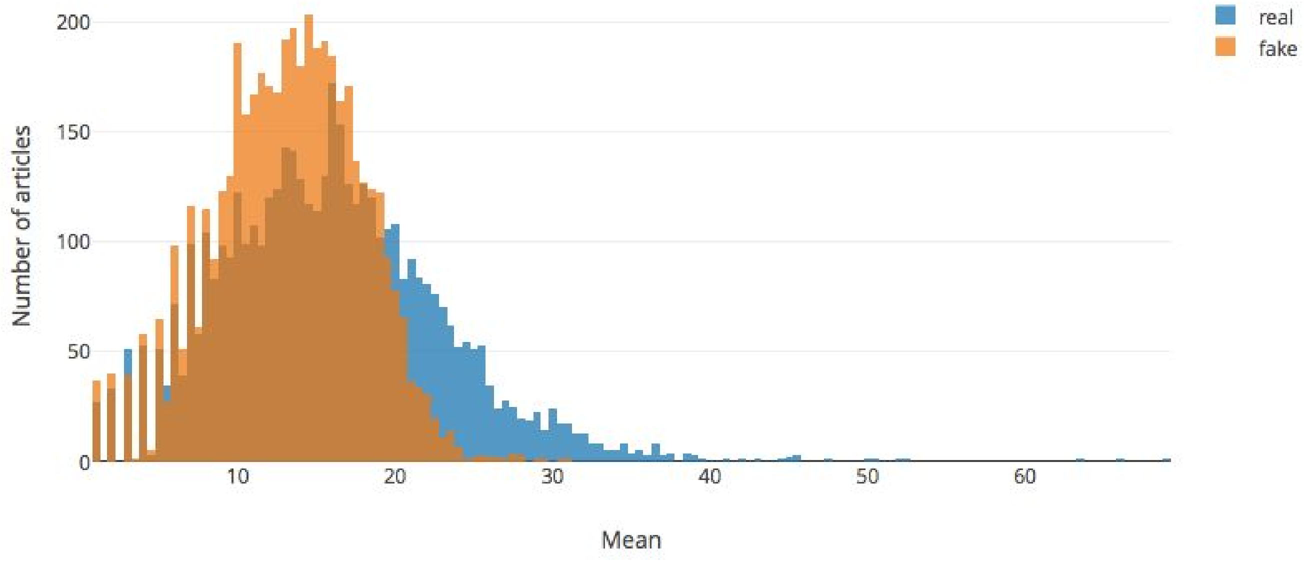
$$A_{w2w} = DD^T \ (W \times W,\ W = vocabulary\ size\ of\ the\ article)$$

<u>Document to Document correlations</u>

$$A_{d2d} = D^T D \ (N \times N,\ N = number\ of\ sentences\ in\ \ the\ article)$$

Further, a truncated Singular Value Decomposition was performed on the A matrices. For the truncation, only the top K eigenvalues (where K = top 10% of the vocabulary size of the article) were retained. For a real article the truncated SVD matrix would be less sparse as compared to the fake articles. With this intuition , the mean, median, minimum and maximum values from the reconstructed matrix would tend to be higher for real articles. Also, the ratio of the sum of the truncated eigenvector to the sum of the initial eigenvector would be higher in case of real articles

The above process was carried out for both A_{w2w} and A_{d2d}. We found out that our statistics on A_{w2w} performed better as compared to A_{d2d}. Below are the plots with statistical measures on the reconstructed eigenvalue matrix for A_{w2w}.



3.3.3 Word2Vec

For the task of classifying articles as True or Fake, contextual dependencies within articles can play a major role in determining the authenticity of the article. In this regard we explore Word2Vec, a word embedding model trained using a neural network to plot words on vector space. The salient feature of this model is that words occurring in similar contexts tend to be more closer in the vector space. Thus, a noteworthy point is that these word2vec features capture the contextual essence of words in the paragraph and not the individual meaning of the word itself which would probably be the reason why Apple, Macbook and Mango may occur close to each other.

In Word2Vec, each word is assigned a dense vector representation that can then be reduced to smaller dimensions using methods like t-Stochastic Neighbor Embedding (t-SNE). Given that our task is to classify News Articles, we chose to employ the pre-trained vectors from the Google-News Dataset. The Google-News model was trained on a total of 100 billion words (tokens). The model contains 300 dimensional vectors for each word with vectors for around 3 million words and phrases. Below are two t-SNE plots for the words in our training data True and Fake articles respectively.

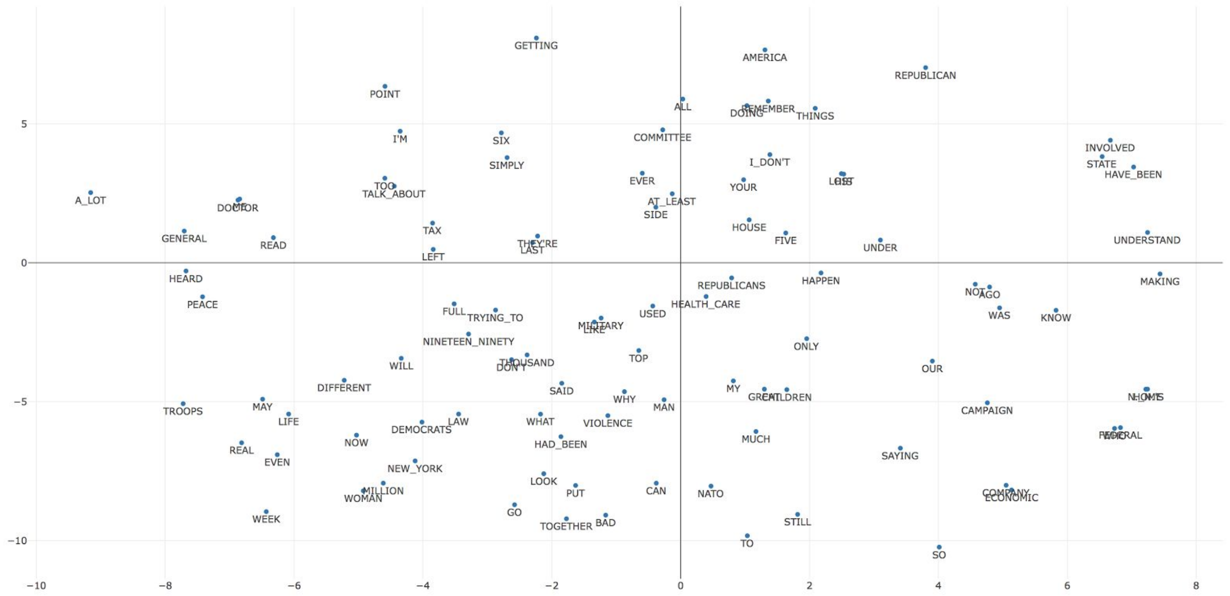


Figure 1: Word2Vec embeddings for true articles

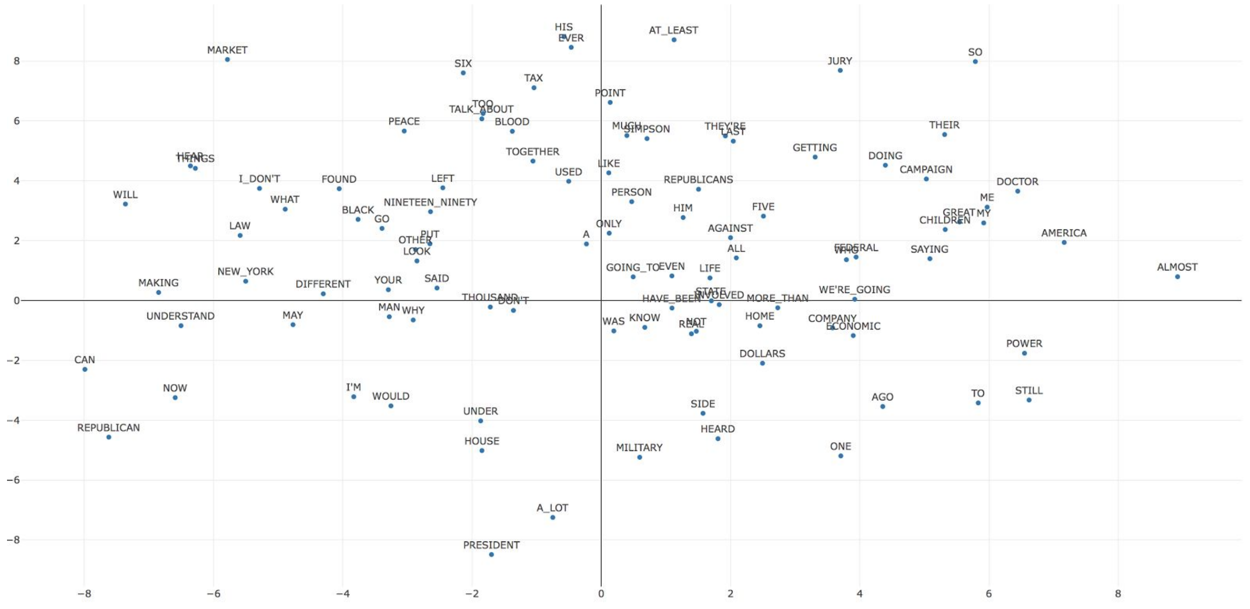


Figure 2: Word2Vec embeddings for fake articles

For ease of viewing, only the first 100 words in each category have been plotted. The figures provide some good insights about the word2vec model. Observing the embeddings for the true articles show interesting context based clusters in the vector space. For example, Republicans and Health_Care are clustered together, while Democrats and Law seem to be close to each other. We can also see that Company and Economic are close to each other too. What is important to note here is that since both the true articles and fake articles' word embeddings come from a common source (Google-News), the embeddings of words present in both corpus may be similar. However, considering the article as a whole, we believe that fake articles wont behave as consistently as true articles since it will be modelled only by a trigram i.e, its previous two words. On observing the fake articles, we are able to see more randomness in the embeddings with not much information between words near each other.

These pre-trained vectors were used on our model to classify Real and Fake articles. We used the mean embedding for each article as well as the variance of the embeddings as two separate features. For variance, we hypothesised that given that a fake article was randomly generated from a trained trigram model, the contextual integrity of fake articles would be very weak and hence words of an article when taken together would have a high variance. On the other hand, the true article would be coherent enough to have a low variance.

This feature was supplied to our machine learning classifiers and results observed. We observed that variance as a feature yielded a 5-fold cross-validation mean accuracy of 0.6 on the training dataset and when tested on the development set, yielded an accuracy of 0.515. On the other hand, the mean embeddings yielded a 5-fold cross-validation mean accuracy of 0.722 on training data and when tested on the development set, yielded an accuracy of 0.6. Clearly, both these features don't seem as informative as expected as hence does seem to be a good feature for selection.

3.3.4 Long Short-Term Memory Networks

In recent times, deep learning methods have proven to be useful in sequence modelling. Recurrent Neural Networks in particular have proven to give useful results in this domain. For this task, we also tried a Long Short-Term Memory Network (LSTM) that has been proven to model long-range dependencies efficiently.

The LSTM for our task was implemented using keras and the architecture of the implemented model was as follows. The input is initially tokenized and converted to sequences of integer encoded words. This is then passed through an embedding layer that converts integer encoded words into a dense vector representation. This dense representation is then supplied to an LSTM with dimensions of 128 (dropout=0.2) for training which is subsequently passed through a dense fully connected layer with sigmoid activation to classify articles. We tried to optimize the binary_crossentropy loss function with adam optimizer.

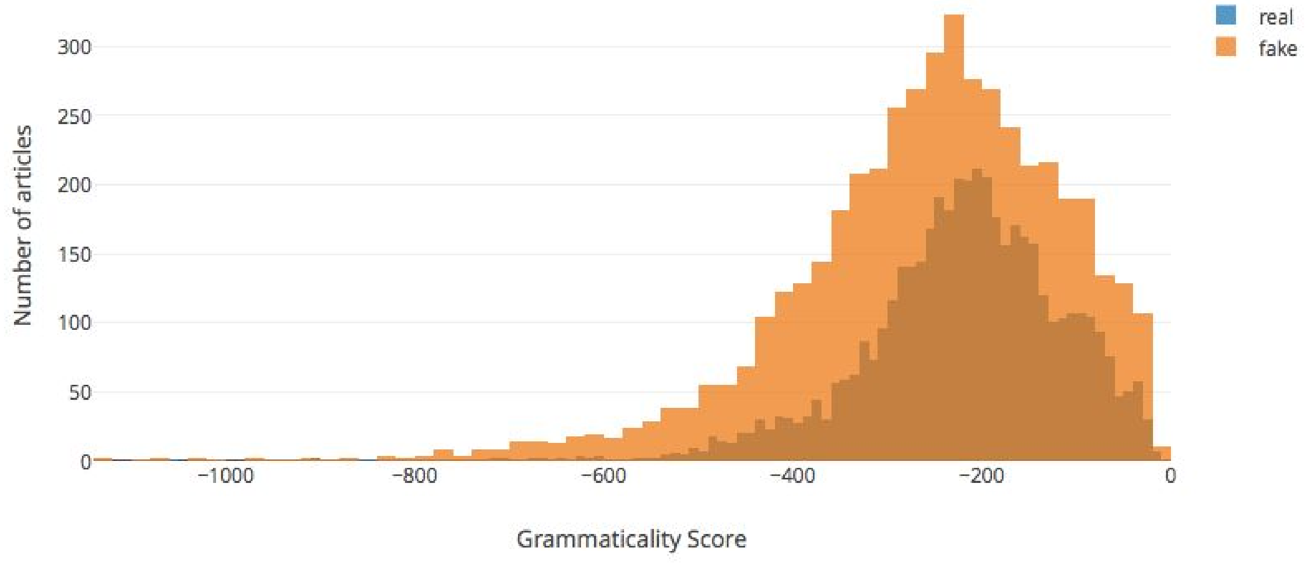
The model was created with a validation_split of 0.25 which meant a quarter of the data would be used for validation with loss calculated on every epoch. An early stopping criterion was also used with a patience of 5. This meant that in case after 5 epochs, if the validation loss consecutively keeps increasing instead of decreasing, we stop training then. This is done to prevent overfitting.

On training the model for a total of 100 epochs, we found that the model kept increasing its training accuracy moving from 0.66 to 0.71, 0.78, 0.82, 0.85 and so on.. However, after the third and fourth epoch, the validation loss started increasing continuously and the early stopping criterion stopped the model from training further. Finally, after a total of 8 epochs, we obtained a strictly increasing training accuracy of 88.61 (which would have kept increasing if more epochs were run), whereas the test accuracy was a mere 0.53, just above random.

3.4 Syntactic Features

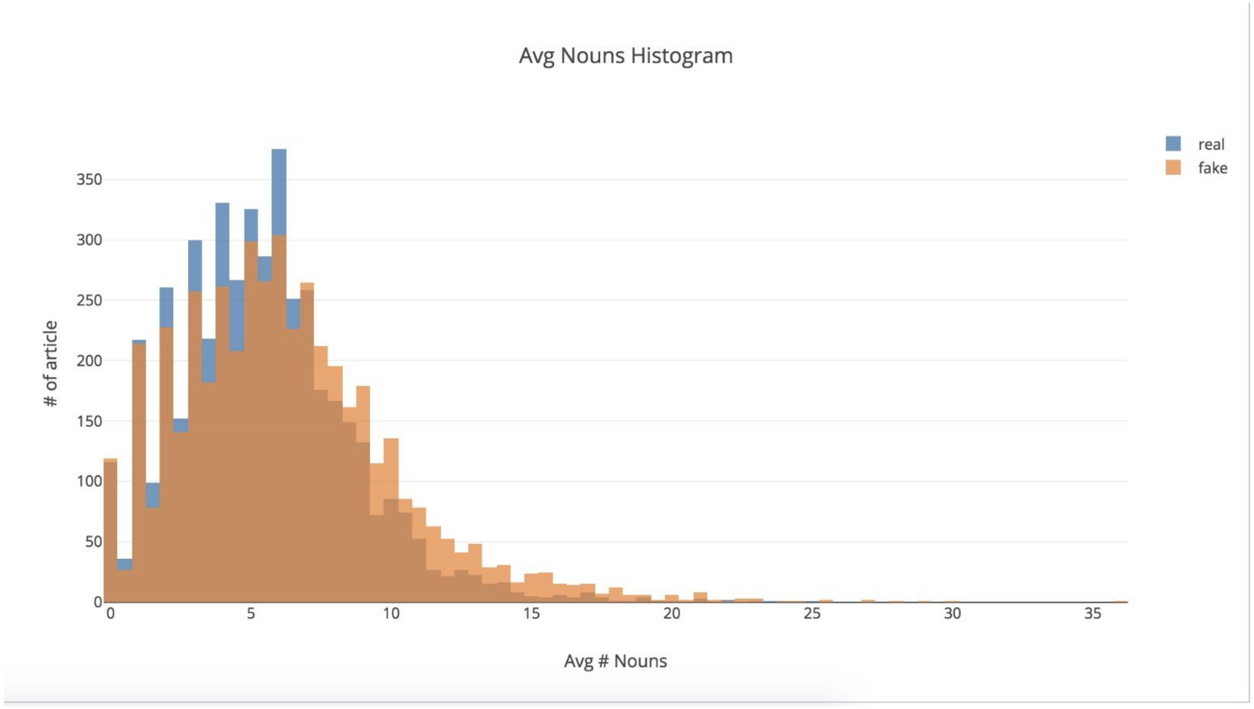
3.4.1 Grammaticality Score

Real articles are more grammatical than fake articles. In order to obtain an objective measure of this, we used the bllipparser which returns a list of all possible parses for a given sentence and the best parse along with the score. The bllipparser is trained on the Wall Street Journal Corpus (WSJ). The parser score is the average log likelihood returned that can be used to distinguish between real and fake articles. For a given article the parser score was obtained for each sentence and a weighted average of the scores was taken over the entire article resulting in the grammaticality score of the article as a whole. Our expectation was that the best parse score will be lower for fake articles which are not very grammatical compared to real articles. Below is the histogram plot for the feature. While we observed the more number of real articles had a higher grammaticality score, we did not see a distinction between real and fake articles. We thus feel that it is because the parser score was not reliable enough to obtain the grammaticality of the article. Also, since the parser was trained on Wall Street Journal Corpus, the grammar rules checked may not be in sync with the grammar rules specific to our corpus.



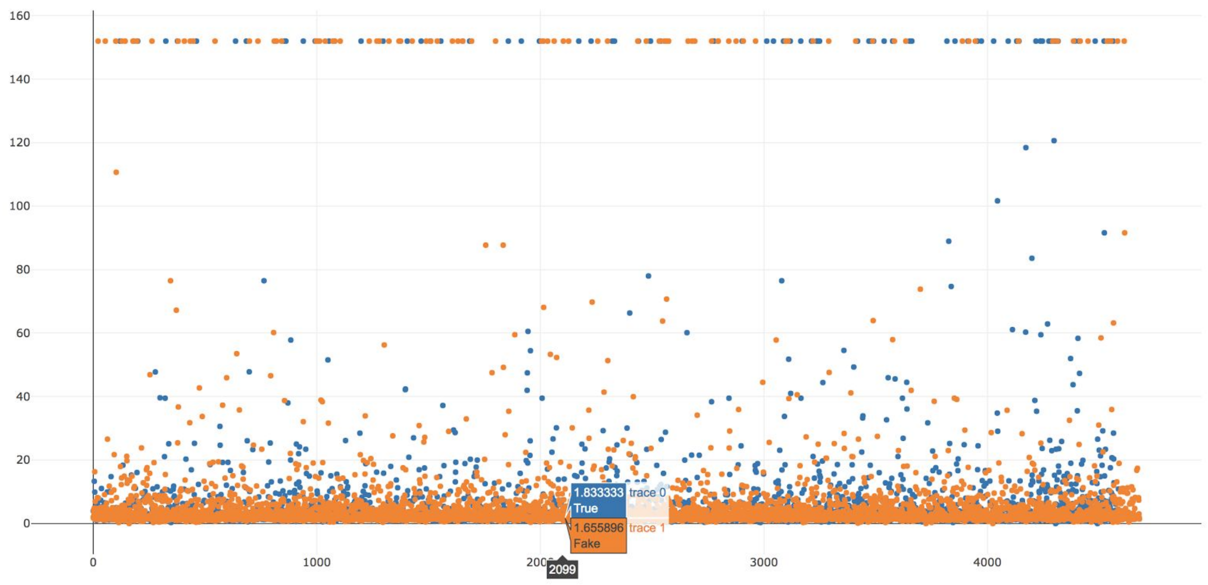
3.4.2 Noun / Verb to Token Ratio

We POS tagged the entire dataset using bllip-parser and our intuition was that in a sentence the ratio of number of nouns / verbs to the total number of tokens will be higher for real articles than fake articles. This is because fake articles don't have a global coherence, the words used may be of less frequent POS tags and not necessarily the more frequent nouns and verbs. But based on the histogram plot below, we realized that our intuition was not strong enough.



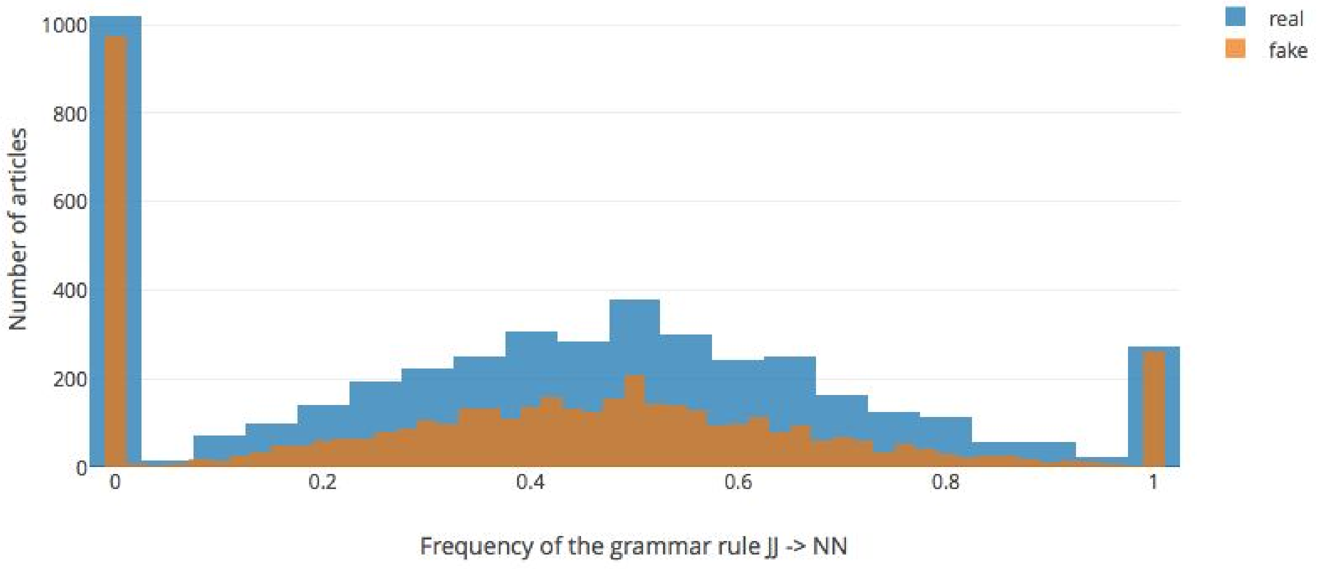
3.4.3 Content To Non-Content POS Tag Ratio

In any real article, the number of content words will always be more than the non-content / supporting / connector words, but this may not be the case in fake articles which do not really distinguish between the content and non-content words. We again used the bllip-parser tagged POS data here. Keeping this in mind, we based our intuition that the ratio of content to non-content POS tags will be higher for real articles and lower for fake articles. Here we took the content POS tags as JJ*, NN*, VB* and RB* and the non-content POS tags as PRP*, IN*, CC* and DT*. Our expectation was that this feature will be higher for real articles than fake articles. The distribution of the feature is plotted below. From the scatter plot below we can see no clear distinction for this feature. This could be because the ratio of content words to function words was approximately equally distributed for sentences from both real and fake articles (although this distribution would be random without any semantic coherence).



3.4.4 Adjective followed by Noun

In real sentences, an adjective is always followed by a noun. By looking at the POS tagged data, we could clearly make the distinction that this rule was strictly followed by real articles where in, in the case of fake articles an adjective followed by a noun was only by chance and was not consistent always. The adjectives were followed by Parts of Speech other than noun most of the time. Hence we calculated the feature of number of JJ* followed by NN* to number of times JJ* occurred in a sentence and normalized this measure across the article. Our expectation was that this feature will be higher for real articles than fake articles. The distribution of the feature is plotted below. Our intuition was incorrect as the number of times JJ* was followed by NN* turned out to be the similar and less distinguishing for real and fake articles.



4 Feature Selection

For feature selection, we used Recursive Feature Elimination (RFE). RFE recursively starts from a big set of features and keeps reducing it until it finds the best combination of features that yield the best results. For our task, when we performed feature selection, we found that the following features performed best sentlen, ttr, stopwords, trifive_ratio, triquad_ratio, entropy, singleton prob, doubleton prob, svd median. A more detailed metrics is presented in the section below.

5 Classification Algorithms

The classifiers on which we model our experiments are as follows:

- Logistic Regression
- Adaboost
- RandomForest
- GaussianNB

Grid search was performed on the above 4 classifiers using varied hyper parameters. Grid search was also applied on feature selection to find out the best number of features to use for the task as well. The results of the grid search for each classifier was obtained as follows:

Logistic Regression

Feature Selection: Best number of features: 7
Best Parameters of Logistic Regression: C = 100.0
Best Features: ['ttr', 'trifive_ratio', 'triquad_ratio', 'entropy', 'singleton prob', 'doubleton prob', ‘svd median']
Best Score on Training: 0.904196784982
Log-Likelihood on dev: -0.145545312308
Accuracy on Dev: 0.92

Random Forest

Feature Selection: Best Number of features: 9
Best Parameters of Random Forest: criterion = ‘entropy’, n_estimators = 50
Best Features: ['noun', 'sentlen', 'ttr', 'stopwords', 'trifive_ratio', 'triquad_ratio', 'entropy', 'singleton prob', 'doubleton prob”]
Best Score on Training: 0.903441579458
Log-Likelihood on dev: -0.0825215647508
Accuracy on Dev: 0.905

Adaboost Classifier

Feature Selection: Best Number of features: 9
Best Parameters of Random Forest: 0.897615708275
Best Features: ['sentlen', 'ttr', 'stopwords', 'trifive_ratio', 'triquad_ratio', 'entropy', 'singleton prob', 'doubleton prob', 'svd median']
Best Score on Training: 0.897615708275
Log-likelihood on Dev: -0.713115434725
Accuracy on Dev: 0.93

Gaussian Naive Bayes

(No grid search as GNB has no parameters other than prior)
5-fold cross validation mean accuracy: 0.863199924863
Log likelihood on Dev: -0.112260474941
Accuracy on Dev: 0.875

Statistical Features

	Logistic Regression	**Random Forest**	**AdaBoost**	**Gaussian Naive Bayes**
Train Accuracy	0.751644087195	0.759628283378	0.775704550547	0.711381205835
Dev Log-Likelihood	-0.40999426018	-0.256917838608	-0.814017729964	-0.488092134396
Dev Accuracy	0.73	0.77	0.745	0.745

Language Model Features

	Logistic Regression	**Random Forest**	**AdaBoost**	**Gaussian Naive Bayes**
Train Accuracy	0.859202678455	0.841940718226	0.855857392994	0.852299733221
Dev Log-Likelihood	-0.243418303454	-0.15149851223	-0.749648505905	-0.216137776863
Dev Accuracy	0.88	0.865	0.885	0.855

Semantic Features

	Logistic Regression	Random Forest	AdaBoost	Gaussian Naive Bayes
Train Accuracy	0.603832920813	0.576758827312	0.639758630909	0.624653471522
Dev Log-Likelihood	-0.797490571412	-0.570187621488	-0.942752934787	-0.774837546615
Dev Accuracy	0.655	0.565	0.545	0.64

Syntactic Features

	Logistic Regression	Random Forest	AdaBoost	Gaussian Naive Bayes
Train Accuracy	0.47416690134	0.496600806874	0.505010331838	0.461217434236
Dev Log-Likelihood	-0.983531755666	-0.733155017022	-0.985744030877	-0.959315444476
Dev Accuracy	0.48	0.565	0.57	0.545

6 Results

After completion of experiments, we found that **logistic regression** yielded the best accuracy with an accuracy of **0.92** and a log-likelihood value of **-0.1455** on the Development Set and an accuracy of **0.9041** on the train set. Furthermore, on grid search, we obtained the inverse-regularization (C value) of 100.0 and recursive feature elimination yielded 7 to be the optimum number of features for this task with this classifier. The best features were:

- Type-Token Ratio
- Pentagram to Trigram perplexity ratio
- Quadgram to Trigram perplexity ratio
- Entropy
- Singleton to Token Ratio
- Doubleton to token Ratio
- Singular Valued Decomposition Median

We obtained a relatively high accuracy of **0.93** on the development set with the **Adaboost Ensemble classifier** along with **0.8976** on the training set. However, the resulting log-likelihood on the development set was found to be -0.713 as compared to the much lower log likelihood of -0.1455 with logistic regression with only a small compromise on accuracy.

7 Contribution and Comments

The project on the whole had been a great learning experience applying the concepts learned over the course of time with Language and Statistics. Overall conclusion is that though seemingly trivial, statistical and language model features best classify the real articles from the fake ones. Some features like grammatical score, Syntactical features from PoS, though intuitively made more sense couldn't make it to the final list due to computational complexity and inability to distinguish the articles to acceptable level. Latent Semantic Analysis that was initially performed to find correlations between sentences performed better when instead of sentences, correlations between words were captured, showing a stronger semantic coherence between words in real articles.

Anusha Prakash

Data Preprocessing Pipeline, Text Language Model Features using SLM toolkit - tri/quad gram perplexities, Latent Dirichlet allocation (LDA) Topic Modeling, Recursive Feature Elimination (RFE) - Feature Selection, TTR Ratio, Rare Words tf/idf, Stop Words to Content Words Ratio, Average Sentence Length, Noun / Verb to Token Ratio, Content To Non-Content POS Tag Ratio, Adjective followed by Noun, Created Fake Corpus for training 'fake language model'

Anushree Kumar

Latent Semantic Analysis using Singular Value Decomposition for term document frequency matrix and tfidf matrix., Inter sentence coherence, word to word correlations - Loss,Mean, Median, Min, Max features of truncated svd eigenvectors. Grammaticality Score feature and POS tagging of the dataset.

Srividya Pranavi

Balancing the data set (creating our own training data), Language Model features using Ken-LM , trigram perplexity, quad gram perplexity, five gram perplexity, trained PoS language models and calculated bigram, trigram and quad - gram perplexity ratios for PoS Language Model, Lexical Entropy per article, ratio of Singletons and doubletons to the Vocabulary. Created Fake Corpus for training 'fake language model'.

Varun Bharadhwaj Lakshminarasimhan

LSTM using Keras - Data Preprocessing, Tokenization, Embedding and LSTM, Word2Vec Embedding (Mean and Variance) (trained on gensim Word2Vec and Google-News word2vec vectors) and t-Stochastic Neighbour Embedding dimensionality reduction and visualisation, Grid Search - Feature Selection: SelectKbest (Univariate Feature Selection (Chi-Square) and Recursive Feature Elimination and Hyper-parameter tuning over estimators.

8 References

1) LSA - http://www.cs.cmu.edu/~shilpaa/IJCNLP_shilpa.pdf
2) KenLM : https://kheafield.com/code/kenlm/
3) Language Tool Kit http://www.speech.cs.cmu.edu/SLM
4) Bllip - parser https://github.com/BLLIP/bllip-parser
5) Detecting Fake from Real Article- Amr Ahmed, Mengqiu Wang, Yi Wu