Beyond Trigram: Learning to Distinguish Fake Article 11-761 Project Report

Yanran Hao
yanranh@andrew.cmu.edu

Yulan Huang
yulanh@andrew.cmu.edu

Ang Lu
alu1@andrew.cmu.edu

Heqing Ya
heqingy@andrew.cmu.edu

April 24, 2016

Abstract

1 Introduction

With the development of Language Technology, people have construct models to formally describe the natural language we are using in daily life. Interestingly, some of the models have been powerful enough to generate fake articles that could even mix the false with the genuine. Three MIT students developed a program called SCIgen[1] that can automatically generate an SCI article. They eventually fooled reviewers in several IEEE conferences with the generated articles. Therefore, a challenge is before us: how can we tell the machine generated articles from human written ones. Finding such methods will not only avoid being fooled, but can also help us build more powerful language model to make fun of the incompetent paper reviewers.

In this project, we have extracted several features, and built several classification models to distinguish the fake generated by trigram models and true articles. According to our experiment result , we have reached about 90% accuracy in cross validation and development set.

2 Feature Generation

2.1 Semantic Feature

For semantic features, we've explored features related to content/stop words and the features about word pairs.

1. Features about content/stop words

Stop words are word which are filtered out before or after processing of natural language data. We use a stop words list 98 common stop words.

Content words are words such as nouns, most verbs, adjectives, and adverbs that refer to some object, action, or characteristic. We generate a content word list using the training data according to words? frequency. We retrieved the words ranks top (using a threshold) and removed stop words.

[1]website: https://pdos.csail.mit.edu/archive/scigen/

2. Feature about word pairs

We use Yule's measure of association as measuring the correlation between word pairs.[1] The measure of semantic association will be defined based on the contingency table below. For each word pair, C_{11} is the count of sentences in the training corpus that contains the word pair; C_{12} is the subtraction of the count of sentences contains word1 and C_{11}; C_{21} is the subtraction of the count of sentences contains word2 and C_{11}; C_{22} is the total number of sentences subtracting C_{11}, C_{12} and C_{21}.

Then we calculate the Q-statistics as the correlation value between word pairs:

$$Q = \frac{C_{11}C_{22} - C_{12}C_{21}}{C_{11}C_{22} + C_{12}C_{21}} \tag{1}$$

The higher value of Q, the stronger the correlation between the two words.

Based on the above two categories, we designed our features as:

- The ratio of stop words and content words.
- The general statistics of correlation value of word pairs.

  We calculate the mean, variance, maximum, minimum, mean and range of the correlation value of word pairs. Moreover, we created a subgroup of words pairs that only contains distant word pairs (distance $\geq$5) and calculated the above feature for this group too.
- Percentage of high correlation values.

  This features detects the percentage of pairs that have high correlation values. We use a threshold to determine what is the standard of high.
- Percentage of unseen pairs.

  Real articles would have a relative stable percentage of unseen pairs while fake articles will have either too high or too low percentage. We believe that this feature can help us distinguish between real and fake.
- Repetition of phrases.

  Similar to the previous one, real articles would have a relative stable percentage of phrases repetitions while fake articles will have either too much repetition or little repetitions. We calculate the percentage of repetitions and the length of longest repeated phrases (single words are treated as special phrases too).
- Coherence score.

  Coherence score are defined as the average correlation value of distant (distance $\geq$5) content pairs.
- Latent Semantic Analysis (LSA).

  Inspired by the indexing technology, we've applied the idea of LSA to create a new feature which we believe can help improving the classification performance.[2] Let $X_{m\times n}$ be a matrix where element X_{ij} describes the frequency of $Word_i$ in $Sentence_j$. Now, the dot product of $A_{n\times n} = X^T X$ contains all the dot products between the word vectors gives the correlation between the term over sentences. By using singular value decomposition (SVD), we can get

  $$A = U\Sigma V^T \tag{2}$$

  Here, $\Sigma_{n\times n}$ is the diagonal matrix contains the singular value. Based on LSA, when you select the top k largest singular values and their corresponding singular vectors from U and V, you get the

Word2 \ Word1	Yes	No
Yes	C_{11}	C_{12}
No	C_{21}	C_{22}

approximation to A. We use a threshold to select the top k singular values and calculate the A_k. We use

$$loss = \|A - A_k\| \tag{3}$$

as the feature which represent the information loss using approximation. For real articles, the loss should be small while fake articles should be relatively large.

2.2 Syntactic Feature

Based on the hypothesis that trigram model cannot capture the syntactic features of sentence and the fake sentence are less grammatical, we measures the grammaticality of a sentence and uses it as one of our feature. More precisely, we parsed both real and fake sentences in the training data using the Charniak Parser in NLTK and got the likelihood of the highest ranked parsing tree for each sentence. We derived the overall grammaticality score by

$$P_{Gram} = \frac{\sum_{i=1}^{N} L_i P(S_i)}{\sum_{i=1}^{N} L_i}$$

However, the syntactic feature is only able to improve the prediction a little bit in the training set and is very time-consuming. Thus, we abandoned it to be further used in the test set. The potential reason for the syntactic feature to be such useless might be that our data contains spoken scripts, which are much less grammatical than the official written language. Another reason might be that the parser is not accurate, since it was trained in mixcase corpus, while our corpus was all uppercase.

2.3 Statistical Feature

Since the trigram model cannot capture information in the quad-gram model, we uses the ratio of perplexity of trigram and quad-gram models as one of the features. Since our training data is very small, the variance of the estimation will be very big. Thus, we used the same BNC corpus to estimate the probabilities of both trigram and quad-gram models. While the two type of articles will both have low perplexity score for the tri-gram model, the real articles would have lower perplexity for the quad-gram model than the fake articles. This implies that the ratio of trigram to quad-gram perplexities would be lower for a fake article than for a real article. Figure 1 shows the distribution of the feature in real articles and fake articles in the training data. As we can see, there is a significant difference of the feature between real and fake articles, which indicates that this is a good feature for distinguish two types of articles.

3 Experiments

3.1 Data Description

The training set is 1000 articles, 500 real and 500 fake, of varying length. The development set is 200 articles, 100 fake, 100 real. Articles in development set is truncated to meet

Word2 \ Word1	Yes	No
Yes	C_{11}	C_{12}
No	C_{21}	C_{22}

Figure 1: Histogram for the ratio of perplexity of trigram and quadgram

#sentence	#article
1	20
2	10
3	10
4	10
5	10
7	10
10	10
15	10
20	10

Table 1: Article length distribution of dev set

the length distribution in Table 3.1. Besides these two sets, we also use a 100 million word corpus of Broadcast News articles as external source for generation of specific feature.

3.2 Data Preprocessing

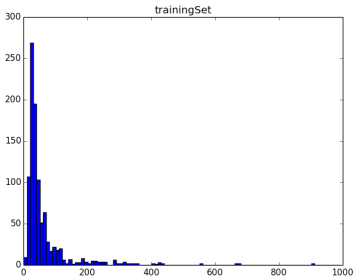
Articles from training set are truncated following the document length distribution in Table 3.1. The number of truncated training articles is 10065. Sentence per article distributions of original training set, truncated training set and development set are shown in Figure 2.

3.3 Classifier Choice

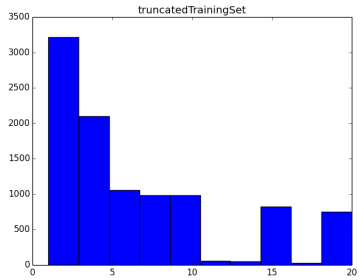
Popular classifiers for binary classification task are chosen as candidates, including KNN, logistic regression, SVM, gradient boosting (xgboost). Testing individual feature on development set and cross validation on training set, xgboost outperforms all the other algorithm by about 5%. Also it is very fast comparing to SVM. So we choose xgboost as the final classier.

3.4 Result

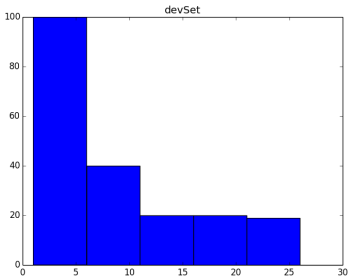
4 Conclusions



(a) Training set (original)



(b) Training set (truncated)



(c) Dev set

Figure 2: Article length distribution of different sets

Appendices

A Contribution

- Yanran Hao: Feature Implementation
- Yulan Huang: Feature Implementation
- Ang Lu: Classification Model Tuning
- Heqing Ya: Data Pre-Processing, Codebase Implementation

B Comments

This assignment gives us a chance to apply statistical method on classification problem, which is good practice for real problems that we will meet in the future. We analyzed the task together, and discussed what features are reasonable, what models are best for this 2-class classification problem, and how we could develop the program together.

We not only get a sense of how the language models work, but also learned techniques in team management[2] and software engineering[3].

For feature extraction part, we found that heuristic works better than our thought. Some simple features such as the repetition of phrases can lead to more than 80% accuracy by only using this very feature. We also found that the performance of language modeling is better than our thought. By using a combination of 3 to 5 features, we can easily get an accuracy of over 80%. However, we found that once achieved a nearly 90% accuracy, the improvement become really difficult. Only by adding reasonable features won't help, it might even lead to overfitting if we are not training on a massive corpus.

One thing worth mentioning is that, for the 4-gram perplexity feature, which has the best performance among all of ours, was almost abandoned for poor performance in the beginning, giving an accuracy less than 60%, which is significantly less than our expectation. We proposed several methods to locate the potential bug in the code of our models and feature extraction parts, but still couldn't figure out the reason. Then, according to the information we get from the previous methods, we went back to the theory itself, and finally realize that the training corpus we chose was not large enough to support a 4-gram model. Thus, we run our program on larger corpus[4] to get more reliable distribution. Then the accuracy was improved significantly. So aside from good features, the size of training data matters a lot since it matter a lot in the model training process.

[2]We used team management tool Teambition to share files and manage tasks

[3]We used Git for version control and Tmux to extract data and train model remotely on server clusters, and parallelize our programs to accelerate the calculation

[4]LM-train-100MW.txt.gz