Sentiment Analysis on Imbalanced Airline Data

Haoming Jiang
School of Gifted Young
University of Science and Technology of China
jinghm@mail.ustc.edu.cn

Abstract

Sentiment analysis is a wildly studied topic in Natural Language Processing(NLP) area. Most researchers focus on the model and algorithm of text processing regardless of other data specific characters. In this work, I explore different models and analysis the airline data from multiple aspects (e.g. imbalance). Specifically, I first examine the property of this real world data and process it carefully for a learnable form. Secondly, I analysis different evaluation methods before I actually analysis the data. The following is the main part of my project, doing sentiment analysis with different models.

Index Terms

NLP, Sentiment Analysis, Class Imbalance, Online Learning, Bag of Words, Language Model, Model Evaluation, Deep Learning.

I. Introduction

A. Sentiment Analysis

The amount of user-generated content on the Internet has risen exponentially over the last decade. [1] These information continuously influence our life. As a result, even our decision-making unconsciously rely on these online resource. For example, we can see others' review and using experience when buying products online. We only buy the service or product when others' sentiment to it is positive.

Sentiment analysis is a computational way of revealing how the sentiment and opinions are expressed in language. It is a vital function allowing product provider to quickly access the customers' reviews. Especially when they want to test their new product, this technique will be quite useful for them to quickly adjust their product based on the market reflection.

Basically, sentiment analysis is to judge whether a given text is positive or negative, which is a classifying task. However, people sometimes express their opinion in a more complex way (i.e. comparative opinions, sarcasm), which make the judgement hard. [2] [3] Some researches even divide the opinion into different parts (i.e. Opinion targets, Sentiments, Opinion holders, Time) to discover the fine structure of the language. Here I only focus on the simplest one, sentiment analysis. Among sentiment analysis, we sometimes face the problem of binary sentiment (positive/ negative), while sometimes we have to handle fine grained sentiment (extreme positive/ positive/ neutral/ negative/ extreme negative), which is a multi-class problem and make the problem even harder. [4]. Although some of them face the situation of multi-class, few of them focus on class imbalance and online learning.

In this work, I both consider the multi-class and class imbalance problem. According to the characteristic of tweet language, I first carefully process the data and turn it into minable form. I also discuss the tactics of handling class imbalance, including model evaluation. After that, I analysis the performance of three different models (Bag of words, Language Model and Neural Language Model) under different circumstance. In addition, I also take online learning into consideration, which is helpful for revealing the properties of different models.

B. Data Set

The data set I used is collected from Kaggle (https://www.kaggle.com/crowdflower/twitter-airline-sentiment). It contains about 14,000 tweets with labeled sentiment about the airline. The labels are in three levels (positive/ negative/ neutral). Most of preliminary researches were done in this relatively small data set. In the following research, I also resorted to a larger data set from CrowdFlower(https://www.crowdflower.com/data-for-everyone/). This data set contains about 55,000 tweets. In the discussion of Neural Language Model (NLM), which is a semi-supervised way to do sentiment analysis. I also utilize the unlabeled data, which I collected from twitter API and can be accessed here(http://home.ustc.edu.cn/~jinghm/files/BHAMProject/Tweets_Unlabeled.csv)

Both the Kaggle data set and CrowdFlower data set are imbalanced. CrowdFlower data set has similar sentiment class distribution to the Kaggle data set. The detail are listed in Table I. Actually, Kaggle data set is a subset of CrowdFlower dataset. They are all labeled by CrowdFlower, which is a machine learning data spreading platform.

Haoming Jiang was a summer intern in Computer Science Department in University of Birmingham, under the supervision of Dr.Shuo Wang and Prof.Xin Yao.

	positive	neutral	negative	total
Kaggle	2334	3069	9082	14485
CrowdFlower	8359	10879	36123	55361
Unlabeled	-	-	-	58817

TABLE I
DATA SET SUMMARY

C. Paper Organization

In this section, I mainly introduced the problem both generally (i.e. sentiment analysis) and data specifically (i.e. multi-class, class imbalance). In Section II, I explained the construction of sentiment aware tokenizer, which is the first step of all sentiment analysis methods. Before the discussion of different model, I introduced the method of evaluating the model in Section III. After that I introduced different tactics of tackling the imbalance problem. Section VVIVII, introduced three different sentiment analysis models. Bag-of-words is the most basic and intuitive way to model language by taking word frequency as features. Language Model originally aims at another problem, modeling the probability under certain circumstance. Neural Language Model utilize the document vector which is generated from deep learning approach. Some preliminary results of online learning are presented in VIII. Finally the the conclusion is draw from the experiment results in SectoinIX

II. TWEET PREPROCESSING

The aim of this procedure follows the most important machine learning principle "Garbage in, Garbage out". Tokenizing, splitting a string into its desired constituent parts, is fundamental to all NLP tasks. I built my own tokenizer based on Christopher Potts's work [1] with slight modification (http://home.ustc.edu.cn/~jinghm/files/BHAMProject/ReadACleanT.py).

The most crude and reckless way to process the tweets is tokenize them with Whitespace. The drawback is obvious. For example, some words may cohere punctuation which make the word unidentifiable (e.g. "say:", "yes!"). A good tokenizer should extract important semantic blocks from the given tweet.

Specifically, we should pay attention to the following items when handling tweet data.

- **Emoticons**: They are common in tweets and always express strong emotion. They should be treated as an entirety instead of separated characters. (E.g. ">: $-D$", ":)", ":(")
- **Twitter mark-up**: "@someone" and "#topic" are common elements in tweets. They should be extract as an entirety. Also since "@someone" is meaningless in sentiment analysis, we can treated them the same via normalizing them as a unified token "MENTIONSOMEONE". Normalizing is helpful for reducing the total vocabulary and help improve the sentiment analysis system.
- **URLs**: We can normalize all URLs as unified token "SOMEWEBSITE". Unless we can access the information on the website, it is useless information.
- **Encoding**: Specifically, we should first ensure the sentence is in unicode. And also some HTML entities should be turn into corresponding unicode characters (e.g. replace "&amp" with "and")
- **Lengthening**: It is helpful to normalize the lengthening of words (e.g. replace "WOOOOOOOW" with "WOOOW")
- **Capitalization**: Preserving capitalization across all words can result in unnecessary sparseness. But be careful that not to change the capital in Emoticons.
- **Numbers**: Since the number in each tweet varies a lot, we can not identify the sentiment from them (i.e. we dont know if the number is high or low without context and common sense). We can normalize them with the token "ANUMBER".

All the above tokenizing methods can be realized with regular expression.

Stemming is a widely applied techniques in handling raw text. This could help reduce the vocabulary size, but it will also undermine the system's performance by decreasing about 1% overall accuracy in my preliminary test (the base line algorithm will be introduced in the following section). It is also widely acknowledged in NLP area. [5] [1]

By putting the pieces together, I extract as much as much sentiment information as possible. The tokenizer normalize some useless information (e.g. numbers, URLs, Twitter mark-up) to avoid unnecessary sparseness while keeping the sentence grammar structure.

III. MODEL EVALUATION

For most Machine Learning tasks, **accuracy** is the only criterion. However, here we face the problem of class imbalance and multi-class (3-class), which required a closer look to the performance of each class. The best way to describe the behavior of a classifier is using **Confusion Matrix**, a specific table layout that allows visualization of the performance of an algorithm. As Figure 1 shows.

In terms of single value criterion, we should choose some statistic which emphasize the performance on minority class. We can either choose the **ROC** type criterion or the **F-score** type criterion. Both of them are originally designed for 2-class situation. As a result we need to generalize them to muti-class situation.

		Predicted		
		Cat	Dog	Rabbit
Actual class	Cat	5	3	0
	Dog	2	3	1
	Rabbit	0	2	11

Fig. 1. Confusion Matrix

A. ROC Type Evaluation

Receiver Operating Characteristic (ROC), is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. Usually, we use **Area Under Curve (AUC)** as a single value criterion. The higher AUC is, the better the classifier is.

However, in the case of multi-class, ROC and AUC cannot be simply applied. A simple extension is **Mean of AUC (MAUC)** [6]. MAUC can be computed under One vs All scheme or One vs One scheme. Another useful extension is **Volume Undef the ROC Surface (VUS)**. [7]–[9] Firstly, we have to extend the definition of ROC to ROC Surface, which has similar property to ROC. After that, we can approximate the volume under the ROC surface in high dimension space. The volume can be used to evaluate the multi-class classifier.

The drawbacks of ROC type criterion is obvious. First, it is hard to calculate especially in multi-class situation and may also be time consuming. Secondly, it is also hard to be extended to every algorithm (e.g. Neural Network, Decision Tree, online learning).

B. F-score Type Evaluation

F-Score is weighted average of precision and recall.

$$F_\beta = (1+\beta^2)\frac{precision \times recall}{\beta^2 \times precision + recall}$$

The most common F-score is F1-score (i.e. $\beta = 1$). It also need to be extended to multi-class situation. A simple and widely used extension is **G-Mean**, which is the geometric mean of recalls of classes. In addition, the Harmonic Mean of recalls also can be used.

Finally, for simplicity, I choose overall accuracy and G-Mean as my criteria. By compare overall accuracy and G-Mean, we can figure out how much class-imbalance influence the performance. In addition, I access the performance of each class by examining confusion matrix.

IV. Class Imbalance Learning

Here I mainly focus on the sampling method of tackling the problem of imbalance.

A. Random Over Sampling and Under Sampling

Random Over Sampling simply randomly fetches the duplicate records from the minority class. Similarly, **Random Under Sampling** simply randomly removes a portion of the majority class. After the sampling, the data set will be much more balanced. However, over sampling sometimes may result in overfitting while under sampling discards useful and valuable data.

B. SMOTE

Synthetic Minority Over-sampling TEchnique (SMOTE) is an over-sampling method. [10]. It over sample the minority class by generating synthetic minority class examples. The essential steps are presented as follows.

It utilize the neighborhood information to generate new examples for minority class. Since it have to find k-neighbor first, it is very time-consuming with large high-dimension data, which is common in text mining.

Algorithm 1 SMOTE

```
Find k-neighbors for each record in minority class
for each synthetic sample do
    randomly choose a sample
    randomly choose its neighbor from kNN
    for each attribute do
        randomly choose a weighted value of the sample and its neighbor on this certain attribute
    end for
end for
```

IDEA: SMOTE remind me of the manifold learning, which aims at finding low dimension structure in high dimension space. Since some manifold learning also utilized the neighbor hood information (e.g. kNN graph), it may not accelerate the algorithm. However, when dealing with high dimension sparse data, it may help SMOTE generate better synthetic minority examples, which follows the same low dimension structure as the original data.

V. Bag of Words Model

Bag-Of-Word (BOW) is the most basic model for extracting feature from language. Each word in the vocabulary has a corresponding feature in the vector $\{w_1, w_2, ..., w_n\}$ as Table II shows. There exists various weighting methods. After obtaining the vector, we can take it as input of classifier to train the sentiment classifier model.

The advantage of this method is straight forward, it is easy to use and applied. Usually it can generate a reliable baseline. On the other hand, the drawbacks are also apparent. The BOW vector representation is really sparse with high-dimension (the same as the vocabulary size). Usually we have to trim the features, which have really low frequency (≤ 1500). The stop words which have really high frequency (e.g. less informative words, such as 'a','the') are also filtered. The feature dimension reduce to about 1,600. The data is still sparse with high demension. As a result, it is very time consuming. Some classifiers become unacceptablely slow and memory demanding, such as AdaBoost, Neural Network.

a	man	one	...	total	zoo
w_1	w_2	w_3	...	w_{n-1}	w_n

TABLE II
BOW vector

A. Different Weighting Methods

Here I compare 3 different weighting methods.

- **Term Frequency (TF)**$tf(t,d)$, the simplest weighting choice is to use the raw frequency of a term in a document, i.e. the number of times that term t occurs in document d.
- **Term FrequencyCInverse Document Frequency (TF-IDF)** The $tf-idf(t,d)$ value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. $tf-idf(t,d) = tf(t,d) \times \frac{N}{|\{d \in D: t \in d\}|}$, N is total number of documents in the corpus
- **Binary** For those words appear in the document. The value of corresponding feature is 1. For other words, the value is 0.

In terms of the classifier, I chose Naive Bayes. The data set I used is the Kaggle data set, which is relatively small , but it is enough for analysis. I randomly split a testing set which contains 1,000 records. The results listed in Table III are the average of ten runs. The running environment is Python.

	Accuracy	G-Mean
NB+tf	0.7672 ± 0.0149	0.6969 ± 0.0208
NB+tf-idf	0.7672 ± 0.0148	0.6969 ± 0.0208
NB+Binary	0.7727 ± 0.0118	0.7045 ± 0.0181

TABLE III
Experiment Results of Different Weighting method

The result also verified the conclusion "Word occurrence may matter more than word frequency" from [5].

B. Different Basic Classifiers

Here I present and compare the results of serval classifiers including **Naive Bayes (NB)** and **Maximum Entropy (MaxEnt)** , which are widely applied in sentiment analysis. In addition, I used other classifiers including **Decision Tree (DT)**,**Support**

Vector Machine (SVM), **AdaBoost.M1**. Among them, AdaBoost.M1 was failed, since I ran it for a whole afternoon and get nothing back. Decision Tree was also failed, because complicate tree structure need a large amount of training time and memory while simple tree is not far more better than guessing the majority('positive'). Support Vector Machine was also failed, because it is hard to choose kernel function. For linear kernel SVM, the performance is not far more better than guessing the majority('positive'). As a result, for decision tree and SVM, I only run the algorithm for one time and the results are not listed here.

The following work is done in R. As a result, the experiment setting is slightly different from the pervious result in Python. The weighting methods of different models are the same. Binary weighting is applied.

	Accuracy	G-Mean
NB	0.7624 ± 0.0120	0.6868 ± 0.0226
MaxEnt	0.7514 ± 0.0133	0.6729 ± 0.0240

TABLE IV
EXPERIMENT RESULTS OF DIFFERENT CLASSIFIERS

From the experiment results, the disadvantage of the high-dimension data is revealed. The large feature make the data hard to learn. For this data, Naive Bayes, which is simple, is good enough. Some classifier even can not be applied. Also, we can tell that it suffer from the class imbalance problem from the comparison of Accuracy and G-Mean.

C. BOW with Class Imbalance

Since Naive Bayes model is a reliable and simple light weight model as previous section described, I use it as my basic classifier. After that, I applied different methods of handling the class imbalance, such as Over Sampling (OS), Under Sampling (US) and SMOTE.

The following work is also done in R. As a result, the experiment setting is slightly different from the pervious result in Python. For Over Sampling, the ratio is the ratio of the total samples and the original samples of the two minority classes ('positive' and 'neutral'). For Under Sampling, the ratio is the ratio of the total samples and the original samples of the majority class ('negative').

	Accuracy	G-Mean
NB	0.7624 ± 0.0120	0.6868 ± 0.0226
NB + 150%OS	0.7652 ± 0.0107	0.6896 ± 0.0192
NB + 200%OS	0.7643 ± 0.0117	0.6881 ± 0.0224
NB + 250%OS	0.7696 ± 0.0128	0.6930 ± 0.0230
NB + 80%US	0.7559 ± 0.0122	0.6775 ± 0.0223
NB + 80%US	0.7437 ± 0.0137	0.6643 ± 0.0223

TABLE V
BAG OF WORDS WITH CLASS IMBALANCE LEARNING

By examining the confusion matrix, I found out the behavior of sampling method seems like just simply move the prediction from the majority class to the minority classes. Which means, the performance of majority class is undermined while the performance of the minority classes is improved. It is also can be explained by the mechanism behind Naive Bayes. The behavior of sampling method is just the same as enlarging the estimated probability of the word features in minority class via adding more appearance of these words.

For SMOTE, the final number of the two minority classes is similar to the majority class, while the majority class did not go through under sampling. (i.e. in training data, negative examples: 9082, neutral examples: 8757, positive examples: 8871) Since the data is really hard to learn, SMOTE become very time consuming. I only run SMOTE for one time, which nearly took me three hours. The overall Accuracy is 0.754 while the G-mean is 0.6856. After examining the confusion matrix of Naive Bayes and Naive Bayes + SMOTE, I found that Naive Bayes misbehaved.

	negative	neutral	positive
negative	518	40	22
neutral	77	126	27
positive	38	33	119

TABLE VI
CONFUSION MATRIX FOR NB

The SMOTE improve the performance of 'neutral' while the performance of 'negative' remain good. But an unusual thing happened in the class 'positive'. The performance drop significantly.

The bag-of-words feature has it own problem. SMOTE use the neighborhood information to generate new data for minority class. [10] However, the two neighborhood information sometimes can be unreliable. The nearness of vector representation, does not mean the nearness of their sentiment. Two near tweet can even generate an opposite sentiment tweet.

	negative	neutral	positive
negative	510	55	23
neutral	85	150	38
positive	25	20	94

TABLE VII
CONFUSION MATRIX FOR NB+SMOTE

Here I use some artificial examples to illustrate that.

- "This flight is the most lovely one I have ever seen" and "This flight is the most hateful one I have ever seen" are neighbors in bag-of-words model.
- Two positive tweets "It is not bad. ", "It is good" can generate "It is not good", which is a negative tweet.
- "It is not bad. It is good" and "It is not good. It is bad" are treated the same in unigram model. Although introducing bigram feature can help this situation, some longer sematic language structure still can not be captured.

Since the feature level can only be 0 or 1, the original SMOTE is not used to generate this kind of discrete feature. Although it can be easily extended to the discrete situation, the above situations are still inevitable.

D. Conclusion

From the above experiment results, we can tell that BOW model can give us a reliable sentiment classifier. However, the BOW's feature has several drawbacks. Firstly, the BOW vectors are sparse with high-dimension which make the algorithm very time consuming. As a result, some classifier become feeble and inapplicable. In addition, the BOW model losts the sentence structure, which contain useful sematic information and may be helpful for sentiment analysis. As a result, SMOTE, which is a powerful imbalance learning tactic, can not be applied. Thus, I resorted to more advanced language model as Section VI VII introduced.

VI. LANGUAGE MODEL

A. N-Gram Language Model Formulation

I have to introduce **Language Modeling (LM)** first. Instead of feature extraction, LM actually focuses on another major research topic in NLP, which aims at modeling the probability of a sentence or document under certain language environment. [11] [12] It is often used in machine translation, while little research was done for sentiment analysis.

A document or a sentence is a sequence of tokens ($\{w_1, w_2, ..., w_n\}$). The goal is to model the probability of a certain sequence:

$$P(w_1, w_2, w_3, ..., w_n)$$

By applying the chain rule, the probability can be expanded as the product of conditional probability:

$$P(w_1, w_2, w_3, ..., w_n) = P(w_n|w_1, w_2, w_3, ..., w_{n-1}) \cdot ... \cdot P(w_3|w_1, w_2) \cdot P(w_2|w_1) \cdot P(w_1)$$

In order to estimate the conditional probability, we can use the following formulation.

$$P(w_n|w_1, w_2, w_3, ..., w_{n-1}) = \frac{P(w_1, w_2, w_3, ..., w_{n-1}, w_n)}{P(w_1, w_2, w_3, ..., w_{n-1})}$$

We can use **Maximum Likelihood Estimation (MLE)** to estimate the probability.

$$P_{MLE}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n)}{N}$$

However the data is really sparse, $P(w_n|w_1, w_2, w_3, ..., w_{n-1})$ can not be estimated if the certain sequence is not observed. As a result, the **Markov Assumption** is introduced to simplify the model:

$$P(w_n|w_1, w_2, w_3, ..., w_{n-1}) = P(w_n|w_{n-1})$$

After applying Markov Assumption, the model is simplified, which is also called **2-Gram Language Model**:

$$P(w_1, w_2, w_3, ..., w_n) = P(w_n|w_{n-1}) \cdot ... \cdot P(w_3|w_2) \cdot P(w_2|w_1) \cdot P(w_1)$$

It is only an approximation of the real model, which is easier to be estimated. However, sometimes it will over simplify the model. If we have more data which could make the model more accurate, this simplification will limit the model from doing so. As a result, higher level models are introduced.

If each word appearance depend on two previous words, the model becomes **3-Gram LM**:

$$P(w_1, w_2, w_3, ..., w_n) = P(w_n|w_{n-1}, w_{n-2}) \cdot ... \cdot P(w_3|w_2, w_1) \cdot P(w_2|w_1) \cdot P(w_1)$$

If each word appearance depend on three previous words, the model becomes **4-Gram LM**:

$$P(w_1, w_2, w_3, ..., w_n) = P(w_n|w_{n-1}, w_{n-2}, w_{n-3}) \cdot \ldots \cdot P(w_3|w_2, w_1) \cdot P(w_2|w_1) \cdot P(w_1)$$

The definition of **N-Gram LM** are similar. In practical, we could use log likelihood to convert the products to sums, which is more computational friendly.

In the case of sentiment analysis, seldom research was done. We can build LM for each class. [13]. We can estimate a given sentence's probability under different LM. After that, we can assign the sentence to the class with highest probability. A interesting thing is that Bag-of-Word plus Naive Bayes is mathematically similar to 1-Gram LM, where words' appearance are independent to each other.

Here, I use the same experiment setting as Section V described. The programming environment is Python. The data set I used is the Kaggle data set. The Bag-of-Word model setting is Binary vectorization plus Naive Bayes classifier.

	Accuracy	G-Mean
BOW	0.7727 ± 0.0118	0.7045 ± 0.0181
2GramLM+MLE	0.7133 ± 0.0119	0.4625 ± 0.0102

TABLE VIII
COMPARISON BETWEEN LANGUAGE MODEL AND BAG-OF-WORD MODEL

The performance of Language Model is poor. It also suffer from severe class imbalance. The reason is quite obvious, that the data set is too small. For minority class, the Language Model only have two or three thousand training data. As a result, I collect a larger data set as the Section I described. I redo the previous experiment with this large data set. The total data contains 55,783 records. Each run of the algorithm, I randomly split a test set of 3,000 records.

	Accuracy	G-Mean
BOW	0.7897 ± 0.0057	0.7294 ± 0.0076
2GramLM+MLE	0.8220 ± 0.0059	0.7907 ± 0.0068
3GramLM+MLE	0.8283 ± 0.0063	0.7912 ± 0.0067

TABLE IX
COMPARISON BETWEEN LANGUAGE MODEL AND BAG-OF-WORD MODEL WITH LARGER DATA SET

The performance of both Language Model and BOW improved. The language models start outperform BOW. They do not suffer from severe class imbalance anymore.

B. Smoothing

The manifest failure of maximum likelihood estimation forces people to examine better estimators. [12] [14] Because we still have to face data sparseness in language model, different smoothing estimating methods are also examined.

a) Laplace Smoothing: The oldest solution is to employ Laplace's law. According to this law:

$$P_{Lap}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n) + 1}{N + B}$$

This process has the effect of giving a little bit of probability space to unseen events. However, in sparse data with large vocabulary, it gives far too much of the probability space to unseen events.

b) Lidstone's Smoothing: Because of the overestimation, a commonly adopted solution to the problem is Lidstone's law (λ is a small constant):

$$P_{Lap}(w_1 \cdots w_n) = \frac{C(w_1 \cdots w_n) + \lambda}{N + B\lambda}$$

In the following experiments, The λ I chose is 0.01. Laplace Smoothing can be taken as a special case of Lidstone's Smoothing.

Besides the above two basic smoothing, **The Held Out Estimator**, **The Cross Validation Estimator**, **Good-Turning estimator** [15],**Witten-Bell Estimator**,**Kneser-Ney Estimator** are also designed to handle the sparsity. According to [16], Kneser-Ney consistently has the best performance. As a result I also tested Kneser-Ney Smoothing. Kneser-Ney also perform well in my experiments. The following experiments were done under 3gram model, which is not too simple nor too complicated.

	Accuracy	G-Mean
BOW	0.7897 ± 0.0057	0.7294 ± 0.0076
3GramLM+MLE	0.8283 ± 0.0063	0.7912 ± 0.0067
3GramLM+Lidstone	0.8303 ± 0.0060	0.8021 ± 0.0102
3GramLM+KneserNey	0.8288 ± 0.0057	0.7910 ± 0.0112

TABLE X
EXPERIMENT RESULTS OF DIFFERENT LANGUAGE MODEL

There is no significant difference between different estimators.

C. Class Imbalance

In the case of sentiment analysis, different language model only models probability under different class, without considering the class imbalance problem:

$$P_{positive}(w_1w_2\cdots w_n)\ P_{negative}(w_1w_2\cdots w_n)\ P_{neutral}(w_1w_2\cdots w_n)$$

As a result, I proposed a **Bayesian Estimator** to handle the class imbalance problem. According to the perspective of Bayesian Analysis, which can minimize the posterior classification error [17], I applied Bayesian Estimator to estimate the posterior probability:

$$\hat{class}(w_1w_2\cdots w_n) = \mathbf{argmax}_c(P(c|w_1w_2\cdots w_n))$$

The previous work only simply compare the conditional probability. The actual probability we want to compare and estimate is posterior probability:

$$P(positive|w_1w_2\cdots w_n)\ P(negative|w_1w_2\cdots w_n)\ P(neutral|w_1w_2\cdots w_n)$$

They can be expanded:

$$P(class|w_1w_2\cdots w_n) = \frac{P(w_1w_2\cdots w_n|class)P(class)}{P(w_1w_2\cdots w_n, class)}$$

As a result, the Bayesian Estimator is:

$$\hat{class}(w_1w_2\cdots w_n) = \mathbf{argmax}_c(P(c|w_1w_2\cdots w_n)) = \mathbf{argmax}_c(P(w_1w_2\cdots w_n|c)P(c)$$

The prior probability is easy to be estimated:

$$P(class) = \frac{N_{class}}{N_{total}}$$

Simply comparing the conditional probability is the same as Bayesian Estimation with equal prior probability.

	Accuracy	G-Mean
BOW	0.7897 ± 0.0057	0.7294 ± 0.0076
3GramLM+KneserNey	0.8288 ± 0.0057	0.7910 ± 0.0113
3GramLM+KneserNey+Bayes	0.8288 ± 0.0064	0.7937 ± 0.0112

TABLE XI
EXPERIMENT RESULTS OF DIFFERENT LANGUAGE MODEL

It seems that Bayes Estimation slightly improved the performance. If we do Student's t test, t statistic is 0.318. Set significant level to be $\alpha = 0.05$. From t-distribution table, we know that the difference is not significant.

IDEA: Although the improvement is not significant, Bayesian Estimation seems worked on this task. It exists possibility to make Bayesian inference which aim at optimizing G-MEAN or F-Score (the Mean of recalls). However, it should be tested in another data set, where LM suffer from severe class imbalance.

D. Conclusion

In the work of language model, I found out that language model also could be applied in sentiment analysis under the support of big data, which is not a traditional way. However, because it is probabilistic model, we can not address class-imbalance via traditional Imbalance Learning. Instead, we only could resolve it in statistical way (e.g. Bayesian Analysis). I introduced Bayesian Estimation to this model, which consider the imbalanced class distribution. However,it is not enough. And I think it is an interesting topic that given the probabilistic model, how to make Bayesian inference which could address class imbalance problem. In order to do the corresponding research, maybe we should face other data set, since LM do not suffer from severe class-imbalance problem here.

VII. NEURAL LANGUAGE MODEL

Deep learning now is a research hot spot, it also plays an important role in NLP research. It nearly redefined the way of doing NLP. [18] With the help of deep learning, people extend our limitation on nearly every task in NLP. In this section, I mainly discussed the application of deep learning in sentiment analysis, which is also called **Neural Language Model (NLM)**.

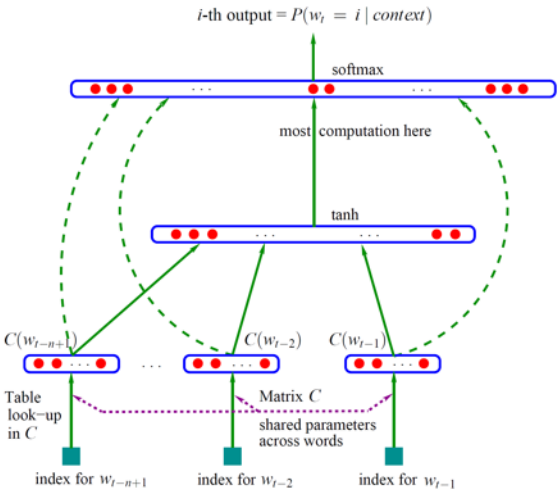


Fig. 2. Neural Network used in NLM [19]

A. Distributed Representation

The main idea of neural language model is to find a mapping from word into **Distributed Representation** in a fixed dimension continuous space. Use the distributed representation as feature, we can address the curse of dimensionality. Nearly all methods of obtaining distributed representation are based on training a Language Model, which is described in previous section.

In order to illustrate how deep learning can be applied, I have to introduce **word2vec** (mapping word to vector) first. This classical model is introduced by Bengio in 2003. [19]. It is also a N-Gram Language model, however it use a neural network to construct the estimator of the conditional probability as Figure 2 shows.

- The first layer map each word to a vector with fixed dimension.
- The second layer use tanh as activation function.
- The third layer use softmax to output the log probability of the next word.
- The objective function is the probability over the whole corpus.

After optimizing the whole neural network, the outputs of the first layer are extracted as word vectors. This word vector is so called Distributed Representation. Amazingly, these vectors contain sematic meaning. A great example to illustrate that is: 'king'-'man'+'women'='queen' . The operation is the operation of word vectors. Although the result vector is not exactly equal to the vector of 'queen', its nearest neighbor is 'queen'.

A lot of deep neural language models are developed. But the essential ideas are similar to Bengios work in 2003. By utilizing the word vector and neural language model, many researchers achieved a lot of the state-of-the-art results in many traditional NLP tasks, including sentiment analysis. I have mainly explored two of them. The first one is Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. The second one is Distributed Representations of Sentences and Documents.

B. RNN with Parsing Tree

The main idea behind this method is to parse a sentence into a **Parsing Tree** via parser (e.g. Stanford Parser). The parsing tree can help with capture the compositional meaning. Each word and parse node is represented by a fixed length vector as Figure 3 shows. And each word and parse node has a sentiment. A Recursive Neural Model is used to compute parent node vector in a bottom up fashion (i.e. taking its children's vectors as input). Also a classifier (e.g. softmax) is established to classify each vector to a sentiment. The sentiment of the root node represent the sentiment of the whole sentence. The given data contains sentiment labels for each node.

I used **coreNLP** to test this method on kaggle data set with a well trained model. This model was trained with movie review data (Stanford Sentiment Treebank). Since this method only can analyze sentence level sentiment, I have to do some modification to fit the airline tweet data, where each record may contain several sentences.

Experiment Results: Accuracy: 0.633 G-mean:0.5593

The above experiment proofed that Recursive Deep Model has the ability of generalization to other data. One drawback of this is that the vectors were trained from movie reviews, which is quite different from our airline data. In addition, this method can only learned sentence level sentiment. Owing to this limitation, I can not use the airline data, where each record may contain multiple sentence, to retrain this model. Even if the model can learn from multi-sentence data, I am not able to fully label the parsing tree for all the data.

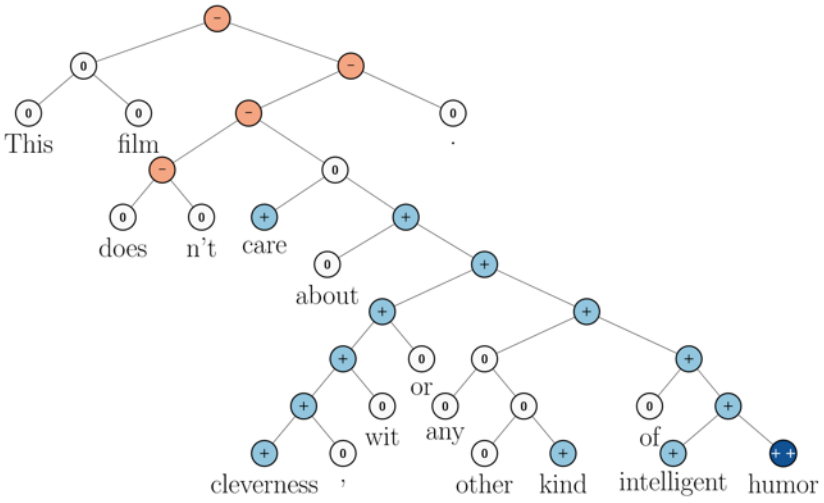


Fig. 3. Parsing Tree of Recursive Deep Model [4]

	negative	neutral	positive
negative	7327	1901	749
neutral	1308	830	570
positive	447	338	1015

TABLE XII
CONFUSION MATRIX FOR RECURSIVE DEEP MODEL

C. Distributed Representations of Documents, Doc2Vec

The other neural language model I mainly explored is **Distributed Representations of Documents (Doc2Vec)**. [20] Here, I only explained the essential idea behind Doc2Vec. It also make the use of concept language model. In language model, the conditional probability may varies from document to document. As a result we can take document into consideration by adding an additional document term in the neural network, as Figure 4 shows.

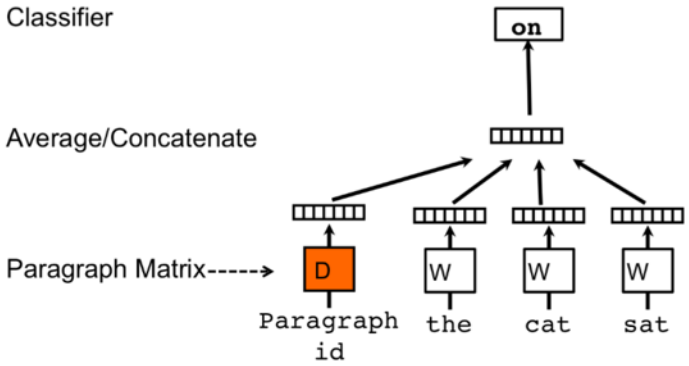


Fig. 4. Doc2Vec [20]

Compared to word2vec, the most important difference is taking document vector into the model. After optimization, we can get distributed representations for documents as well as words.

Since it is an unsupervised learning approach, we trained Doc2Vec with both training data and testing data on the same time to get the document vectors before sending vectors to classifier. It contradict the intuition. Since we do not have to use the label information, it is fine to utilize unlabeled testing data. Once we obtained document vectors, we could use vectors from training data to build sentiment classifier (e.g. softmax, neural network).

Experiment Setting: I used the GENSIM Python package [21] to implement this model. By following the parameter setting in [20], I choose the both DM model and DBOW model. In each model, the length of the document vector is set to be 400. After that, I concatenate them together. The finial model setting in GENSIM is

- $modelDM = Doc2Vec(size = 400, window = 3, min_count = 1, sample = 1e-4, negative = 5, workers = cores, dm = 1, dm_concat = 1)$

- $modelDBOW = Doc2Vec(size = 400, window = 3, min_count = 1, sample = 1e-4, negative = 5, workers = cores, dm = 0)$

Unlabeled Data After noticing that this model can utilize unlabeled data to get the document vectors. I collect other 55,000 unlabeled data from tweet api. I did experiments both with or without unlabeled data. In terms of classifier, I used SoftMax.

Neural Network Classifier Since the performance of SoftMax is not ideal, I tested different classifier. Among them **Neural Network(NN)** with well tuned parameters gave the best performance. When tuning the parameters for NN, we have to trade off regularization and fitting by checking the accuracy on training data and testing data. In terms of regularization, I applied drop out. The construction of NN is 2 fully connected layers. 100 nodes for the first layer, 20 nodes for the second layer. Both of them are with drop-out at the ratio of 0.3. I used Stochastic Gradient Descend to optimize it.

Class Imbalance In neural language model, the classifier also suffer from class imbalance. As a result, I applied SMOTE on the data. Since distributed representation can capture the sematic meaning in a continuous space, I expected SMOTE can behavior well and handle the problem mentioned in Bag-of-Word model.

	Accuracy	G-Mean
BOW	0.7897 ± 0.0057	0.7294 ± 0.0076
3GramLM+KneserNey	0.8288 ± 0.0057	0.7910 ± 0.0113
Doc2Vec+SoftMax	0.7525 ± 0.0055	0.5837 ± 0.0112
Doc2Vec+SoftMax+UnlabeledData	0.7766 ± 0.0056	0.6268 ± 0.0134
Doc2Vec+NN+UnlabeledData	0.8203 ± 0.080	0.7235 ± 0.0160
Doc2Vec+NN+UnlabeledData+SMOTE	0.8277 ± 0.066	0.7839 ± 0.0102

TABLE XIII
EXPERIMENT RESULTS OF NEURAL LANGUAGE MODEL

Here is three observations from the above experiments: 1. The unlabeled data can really improve the quality of the distributed representation. 2. Neural language model suffer from class imbalance. 3. SMOTE significantly improved the performance in terms of G-Mean by reducing the class imbalance.

D. Conclusion

From the experiment results in Table XIII, we can find out that neural language model really outperform the classical Bag-of-Words model. The low-dimension representation benefits classifier a lot. In terms of class imbalance learning, SMOTE behaved normally and indeed improved the performance. For the success of SMOTE, which utilize the neighborhood information, we could conclude that the nearness in the distributed vector space really means the sematic nearness at least for sentiment analysis. However, tuning parameter of the model is really hard. In addition, we can not make this system online. Although both classifier and doc2vec can be online updated, they can not be updated at the same time. Once Doc2Vec is updated, all document vectors are changed. As a result the classier can only be retrained to fit new vectors. Maybe we can pile up the sentiment classifier and Doc2Vec model together, which may leads to better classification results and feasibility of online learning.

VIII. ONLINE LEARNING

In this section I only briefly discussed the online learning in Bag-of-Word Model and traditional Language Model. For neural language model, which counters obstacles in online learning, it will not be discuss in this section. Since I am not going to analyze the time factor, I just choose a fixed testing data for online learning. For more discussion about class evolving, please refer to Pan Li's report. Although the time factor is not the main consideration here, it is meaningful to discuss online learning as a supplement of discussion of the size of data set.

A. Online Bag-of-Word

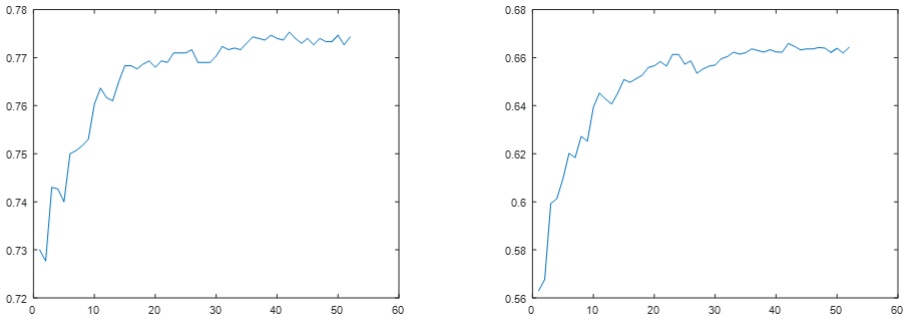
Experiment Setting: The BOW model took Naive Bayes as the classifier with Binary feature. Each step I took 1,000 new data to update the model.

From the online learning result, we can clearly observe that as the data set goes large, the performance slowly converges. It can be inferred another limitation of Bag-of-Word model that it oversimplified the model. It has already reach the limitation of model, we can not improve the model by adding more data.

B. Online Language Model

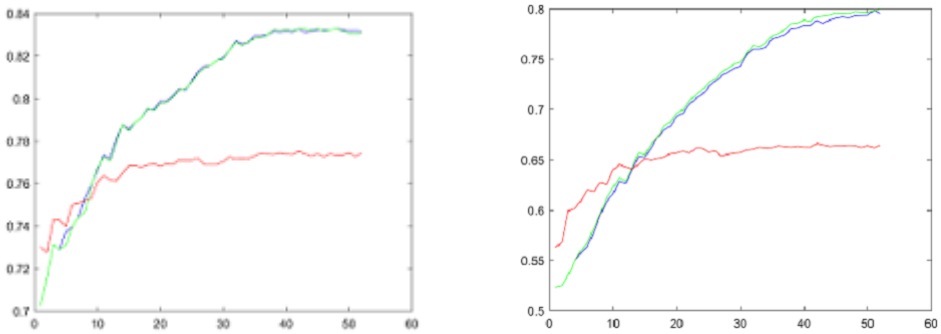
Experiment Setting: The LM model took KneserNey as the estimator. Each step I took 1,000 new data to update the model.

From the Figure 6, we can conclude that LM indeed is more potential than BOW. However, when dealing with small data BOW is more reliable. Also an interesting thing is that the Bayesian Estimator indeed slightly improve the classifier in terms of G-Mean. It can be inferred from the figure(b), that the green curve is always above the blue one.



(a) Overall Accuracy (b) G-Mean

Fig. 5. Online Bag-of-Word



(a) Overall Accuracy (b) G-Mean

Fig. 6. Online Language Model and Bag of Words. The green line is LM with Bayesian Estimator, while the blue is LM. The red one is BOW.

IX. Conclusion and Future Work

In conclusion, I analyzed three different models: Bag-of-Words, Language Model and Neural Language Model. Both of them I tried to handle the class imbalance problem. Bag-of-Words model is simple and reliable. But the problem of the feature prevent me from applying better classifier and SMOTE, a great class imbalance learning tactic. Language Model has the best performance. But it require a larger data set to support. Also it seems not suffer from severe class-imbalance problem. I proposed a Baysian Estimation to take nonequal class distribution into consideration. Neural Language Model is the latest research. It has great performance. Because it can map the tweet into distributed representation, which contains sematic meaning, I am able to apply classifier such as Neural Network. Also SMOTE can be successfully applied to handle class imbalance. If we have larger data, it may be more powerful than Language Model. However, the drawback of this method is that it may need larger data set in order to outperform traditional LM owing to that it is a more complicated model. Also it is not easy to do online learning based on its working pipeline. In addition, the performance relies on tuning the parameters.

Possible Future Work: Try to investigate online learning for Neural Language model on sentiment analysis. Also maybe there exists better Bayesian Inference for Language Model waiting for us to disclose.

Acknowledgment

The author would like to thank University of Science and Technology of China and Computer Science Department of University of Birmingham for giving me the intern opportunity. Also I would like to thank Dr.Shuo and Prof.Xin for kind advising.

References

[1] C. Potts, *Sentiment Symposium Tutorial*, 2011. [Online]. Available: http://sentiment.christopherpotts.net/
[2] B. Liu, *Sentiment Analysis tutorial AAAI 2011*, 2011. [Online]. Available: https://www.cs.uic.edu/~liub/FBS/Sentiment-Analysis-tutorial-AAAI-2011.pdf
[3] ——, *Web data mining: exploring hyperlinks, contents, and usage data*. Springer Science & Business Media, 2007.
[4] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, vol. 1631. Citeseer, 2013, p. 1642.

[5] R. Heimann and N. Danneman, "Social media mining with r," 2014.

[6] S. Wang and X. Yao, "Multiclass imbalance problems: Analysis and potential solutions," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, no. 99, pp. 1–12, 2012.

[7] C. Ferri, J. Hernández-Orallo, and M. A. Salido, "Volume under the roc surface for multi-class problems," in *European Conference on Machine Learning*. Springer, 2003, pp. 108–120.

[8] T. Landgrebe and R. Duin, "A simplified volume under the roc hypersurface," *SAIEE Africa Research Journal*, vol. 98, no. 3, pp. 94–100, 2007.

[9] X. He and E. C. Frey, "The meaning and use of the volume under a three-class roc surface (vus)," *IEEE transactions on medical imaging*, vol. 27, no. 5, pp. 577–588, 2008.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[11] C. D. Manning, P. Raghavan, and H. Schtze, "An introduction to information retrieval," *History*, vol. 43, no. 3, pp. 824–825, 2008.

[12] C. D. Manning and H. Schtze, *Foundations of Statistical Natural Language Processing, MIT Press*. MIT Press,, 1999.

[13] K.-L. Liu, W.-J. Li, and M. Guo, "Emoticon smoothed language models for twitter sentiment analysis." in *AAAI*, 2012.

[14] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," in *Meeting of the Association for Computational Linguistics, 24-27 June 1996, University of California, Santa Cruz, California, Usa, Proceedings*, 1996, pp. 359–393(35).

[15] W. A. Gale, "Good-turing smoothing without tears," *Journal of Quantitative Linguistics*, vol. 2, 2000.

[16] B. MacCartney, *NLP Lunch Tutorial: Smoothing*, 2005. [Online]. Available: http://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf

[17] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian data analysis*. Chapman & Hall/CRC Boca Raton, FL, USA, 2014, vol. 2.

[18] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[19] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[20] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents." in *ICML*, vol. 14, 2014, pp. 1188–1196.

[21] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[22] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," in *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*. IEEE, 2009, pp. 324–331.

[23] C. C. Aggarwal and C. Zhai, *Mining text data*. Springer Science & Business Media, 2012.

[24] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[25] D. J. Hand and R. J. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Machine learning*, vol. 45, no. 2, pp. 171–186, 2001.

[26] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python*. " O'Reilly Media, Inc.", 2009.