Amazon Reviews' Rating Prediction Based on Word Vectors

Jie Cui

College of Computer and Information Science, Northeastern University
Boston, MA 02115
cui.jie@husky.neu.edu

Abstract. Customer reviews containing language that would indicate the ratings. However, sentiment analysis is a challenging work in natural language processing. The language is often obscured by sarcasm, ambiguity and so on. With the assumption that the rating of a review may be embodied by key words, this paper applied word2vec to 568,454 amazon food reviews. After the word2vec model was established, word vectors were used in two ways to construct features for prediction, averaging word vectors and clustering words according to their vectors. By comparing the effect of word2vec model on predicting review ratings with a traditional way of using random forest models in terms of the number of key words in a particular review, the results showed that the word2vec model is superior to the latter and is more efficient. In addition, to collect plenty of various sentimental text dataset will improve the performance of word to vector model.

Keywords: Rating prediction, Word2Vec, Random Forest

1 Introduction

Words are treated as atomic units in many current NLP systems and techniques, where no notion of similarity between words, as these are represented as indices in a vocabulary. These kinds of techniques has some advantages. First, they are simple. For example, N-grams are very traditioanl model used for statistical language modeling. They are sequences of characters or words extracted from a text. The main motivation behind this approach is that similar words will have a high proportion of N-grams in common. Second, experiments show that simple models trained on huge amount of data can get better results than some complex systems trained on less data. Nowadays, it is possible to train N-gram on very huge amounts of data (trillions of words).
However, there are many limits of these simple techniques when applied to many applications. For instance, in automatic speech recognition, the size of relevant in-domain data is small - the performance usually depends on the size of high quality transcribed speech data (often just millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, there are situations where simple scaling up of the basic

techniques will limit us to make significant progress, thus we have to explore more advanced techniques.
In my project, i mainly explored a more complex model - word2vec, which is a technique that transforms words into vectors, so that words with similar meaning can end up laying close to each other. And we can use this characteristic to construct word features for corpora and applied to some natural language processing or artificial intelligence tasks, like prediction and classification. My project is a classification project, I used three methods to generate different word features and applied these features individually to a same model to predict. Then i compared the performance of each method. The three methods are: bag of words, which is a traditional approach of feature construction for text mining, and it computes the frequency count in terms of words; averaging word vectors - in which i averaged the multi-dimensional word vectors of all the words in a particular review to be a feature of the review; word clustering, in which i clustered the words according to their vectors by K-mean. All these three methods gave pretty good results, especially the two methods that used word2vec to produce word features.
The development of words represented as continuous vectors has taken a long time, mostly based on neural network language models (NNLM). In such systems, the word vectors are randomly initialized and then trained to predict optimally the contexts in which the corresponding words tend to appear. Since words with similar meanings generally appear in similar contexts, their resulting word embeddings are semantically close after training. This work has been followed by many others.Word2vec was created by a team of researchers led by Tomas Mikolov at Google. Subsequently, many other researchers has analyzed and explained the algorithm. Embedding vectors created using the Word2vec algorithm have many advantages compared to earlier algorithms like Latent Semantic Analysis.

2 Related Work

In this section, I present a brief review from two perspectives of the related work, Amazon review sentiment classification and constructing word features for sentiment classification.

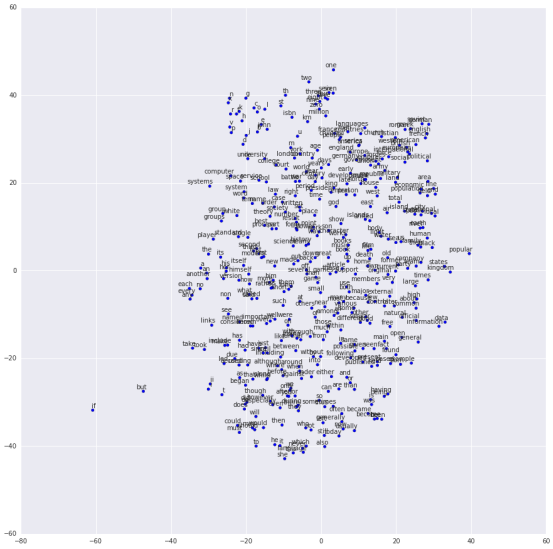
2.1 Amazon Review Sentiment Classification

In recent years, many students has been working on Amazon review sentiment classification as their project, which identifies the sentiment represented in stars scaling from 1 to 5 of pieces of Amazon reviews. The methods applied in Amazon sentiment classification follow traditional sentiment classification approaches in general. Many methods on sentiment classification attach great importance to choices of features. Since the performance of sentiment classifier is heavily dominated by the choice of feature representation of reviews. Here i focused on

comparing the performance of different word features produced by three methods.

2.2 Constructing Word Features for Sentiment Classification

The first method i used is bag-of-word representation, representing each word as a one-hot vector. Because it will be time-consuming if i choose to use the feature's length to be the size of the vocabulary, i ordered the vocabulary according to each word's frequency in the training data and select the first 40th most frequent words as key words, which means the length of my feature of each piece of review is 40. However, the one-hot word representation cannot capture sufficient complex linguistic characteristics of words.
To learn word embeddings from raw text efficiently in terms of computation, Word2vec is a pretty good predictive model for this purpose. It comes in two flavors, the continuous bag-of-words model and the skip-gram model. Nowadays there are many toolkits like gensim that can help us easily build a Word2vec model, of which you can choose the number of dimensions of word vectors. In my project, i chose 300 dimensional vector. After training has finished, we can visualize the learned embedding using t-SNE. Below is a graph of learned vectors projected down to 2 dimensions using t-SNE dimensionality reduction technique. We can find that vectors capture general and helpful semantic information not only about words, but also their relationships to one another.



3 Experiment

I conducted experiments to evaluate the performance of three different word features produced to be employed in random forest model to predict the stars associated with each review.

3.1 Experiment Setup and Datasets

I used the Amazon Fine Food Reviews dataset consisting of 568,454 food reviews Amazon users left up to October 2012. This dataset is from a platform named Kaggle for predictive modeling and analytics competitions. Finding predictive relationships from data is usually done with training training dataset on a model, and a test dataset is used for evaluating whether the discovered relationships hold. With a set of dataset, common proportions are 70%/30% training /test to split the dataset. Here i chose to split my dataset into 80%/20% training /test. I wrote a python snippet to split my dataset into two CSV files, and put them under the 'data' folder, named them `test_data.csv` and `train_data.csv`. The columns in each table are: Id, Score, Text. 'Score' is the rating between 1 and 5, 'Text' is the text of each piece of review.

	Train	Test
Number	454,763	113,691
Percentage	0.8	0.2
File	`train_data.csv`	`test_data.csv`
Score	1 to 5	1 to 5

Table 1. Statistics of the Amazon Review rating classification dataset.

Since a Word2vec model can perform better if it is trained on larger training dataset, i downloaded a dataset of movie reviews from http://www.cs.cornell.edu, and i used this dataset with the training dataset i split from original Amazon review dataset to train the Word2vec model. The movie reviews' dataset is named `review_polarity` under the 'data' folder.

3.2 Baseline Methods

Below are the methods that i used in my project:

1. The Bag of Words representation: In text analysis, raw data is a sequence of symbols that cannot be directly used in the algorithms, since most algorithms require numerical feature vectors with a fixed size instead of the raw text with variable length. In order to address this, i used CountVectorizer in scikit-learn, which provides utilities to extract numerical features from

text. Thus, a corpus can be represented by a matrix with one row per review and one column per token (word) in the first 40 most frequent words (in my project) occurring in the corpus. The vectorization is called the general process of turning a corpus into numerical feature vectors. The steps (tokenization, counting and normalization) to realize this is called the Bag of Words representation.

2. Word2vec: This technique uses a shallow neural network to learn the representation of words in corpus. In the project, i used Gensim's Word2vec API which requires some parameters to initialize the model. There are several parameters that i defined on my own.

   (a) size: this denotes the number of dimensions present in the vectorial forms. I repeated the initialization for different numbers and pick 300 which yields better results than other numbers i had tried. A good heuristic that frequently used is the square-root of the length of the vocabulary.

   (b) `min_count`: Words with frequency count less than min_count number of times are ignored in the calculations. This reduces noise in the semantic space. Generally, the more extensive your corpus is , the higher this number can be. Here i chose 40.

   (c) window: Only terms that occur within a window-neighborhood of a term, in a sentence, are associated with it during training. I chose 10 in my project.

   (d) workers: Number of threads to run in parallel. More workers, faster we train. Here i chose 4.

3. K-mean: K-mean is one of the most popular clustering algorithms. It stores k centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid. Here in my project, i used the scikit-learn to perform K-mean. And i set k to be 1/20 of the vocabulary size, and the feature that i applied to the K-mean model is the word vectors i acquired by Word2vec.
4. Random Forest: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. I used the random forest classifier in scikit-learn.

   (a) `n_estimators`: The number of trees in the forest. For all the three methods i mentioned above that were used to produce word features, i used this random forest model to predict the ratings of Amazon reviews from test dataset.

3.3 Program Description

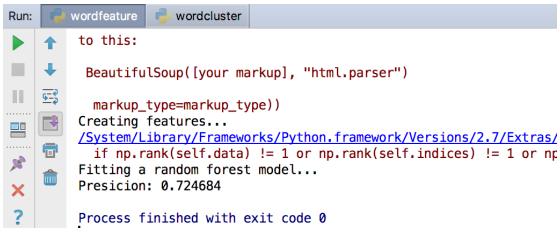
1. Text Preprocessing
   Before we use the text dataset, a common issues in natural language processing is how to clean the dataset, especially when we deal with text from online, because there are HTML tags, abbreviations and punctuation in the text dataset. The code in the python file named 'cleanwords.py' is used to do this work. And to remove HTML markup, i used the Beautiful Soup library. Since stop words are frequently occurring words that don't carry much meaning, i imported a stop word list from Natural Language Toolkit (NLTK). The second function `text_to_sentences` is typically used for Word2vec model, since it expects a sequence of sentences as its input. To split a paragraph into sentences, i used NLTK's punkt tokenizer for sentence splitting. Importantly, it is better not to remove stop words to train Word2vec, because the algorithm relies on the broader context of the sentence in order to produce high-quality word vectors.

2. Imputation of Missing Values
   Many real world datasets contain missing values because of various reasons, and these missing values are often encoded as blanks, NaNs or other placeholders. Such dataset is incompatible with the Random Forest estimator i used from scikit-learn, which assumes that all values in an array are numerical. A good strategy that can be used to solve this problem is to impute the missing values, to infer them from the known part of the data. So i used the Imputer class in scikit-learn, which either uses the mean, the median or the most frequent value of the row or column in which the missing values are located.

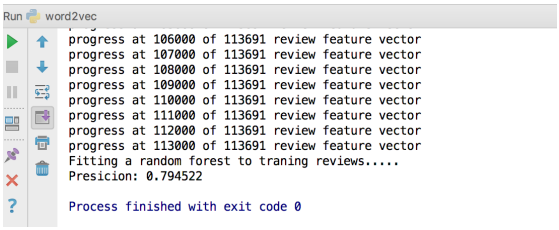
3. Evaluation Results
   To compare those three methods for classification, we need a way to evaluate the result of each methods. In my project, i used the method `accuracy_score()` of sklearn.metrics. It calculates how accuracy the classification is, and returns accuracy classification score. This is a pretty convenient method, all i need to do is to enter two parameters in the method: the true score value for each test review in test dataset, and the predicted score value.

3.4 Results and Analysis

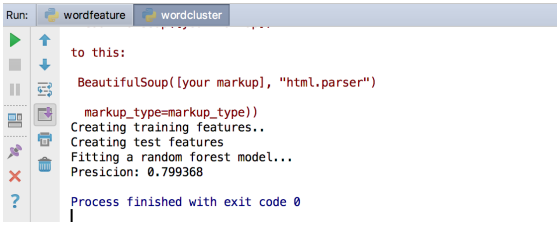
1. The Bag of Words:
   It takes 1 hour to finish running the code for the classification using method of Bag of Words in wordfeature.py. And the result is showing below:



2. Averaging word vectors:
   It takes 40 minutes to finish running the code for the classification using method of averaging the word vectors in word2vec.py. And the result is showing below:



3. Word clustering:
   It takes 50 minutes to finish running the code for the classification using method of word clustering in wordcluster.py. And the result is showing below:



1. The Bag of Words takes longest time and gets the worst result, with accuracy of 0.725. To improve the accuracy, we can add more key words by change the `max_features` parameter of CountVectorizer to a greater value. However, this may cause the code running for very long time.

2. The method of averaging word vectors outperform the other two methods in running time. It takes less time with an accuracy of 0.794.

3. Word clustering takes 50 minutes with the highest accuracy among these three methods that produce word features. The accuracy of word clustering method is 0.799. To improve the accuracy, we can increase the number of clusters. Trial and error suggested that small cluster gives better results than large clusters with many words. A trade-off is that this may take more time to run the code.

4. For all these three methods, most of the time was cost on fitting the random forest model.

4 Conclusion

Sentiment analysis is a challenging subject in natural language processing. There are many ways to solve this problem. In this paper, i used three methods and compared the performance of each method. Google Word2vec focuses on the meaning of words. It attempts to understand meaning and semantic relationships among words. By using Word2vec model from gensim, i found the word to vector method is computationally more efficient than traditional simple methods like bag of words. With the strong power of analyzing similarities among words, there are many areas that Word2vec can be employed to make significant progress. An extension of Word2vec to construct embeddings from entire documents instead of the individual words has been proposed, which is a technique called Doc2vec.

References

1. Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean: Efficient Estimation of Word Representations in Vector Space
2. P Majumder, M Mitra, B.B. Chaudhuri.: N-gram: a language independent approach to IR and NLP, Computer vision and pattern recognition Unit, Indian Statistical Institute, Kolkata
3. Feature Extraction, `http://scikit-learn.org`
4. Building Word Embeddings for Solving Natural Language Processing, `http://infoscience.epfl.ch`
5. Vector Representations of Words, `https://www.tensorflow.org`
6. Generating a Word2vec model from a block of Text using Gensim (Python), `https://codesachin.wordpress.com`
7. sklearn.ensemble.RandomForestClassifier, `http://scikit-learn.org`
8. Preprocessing data, `http://scikit-learn.org/stable/modules/preprocessing.html`
9. Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, Bing Qin: Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification
10. Word2vec, `https://en.wikipedia.org/wiki/Word2vec`
11. Bag of Words Meets Bags of Popcorn, `https://www.kaggle.com/`