# Analyze Reviews Texts and Predict Items Rating

## I. Introduction

With the rapid development of information technology and the applications in the past tenth years, the Internet has not only brought great changes to society and had a profound impact on consumer behavior patterns. Many people prefer to look for the reviews of one item before they plan to buy one product. Reviews are one important part of the information of one item. What is more, people will express their feelings and evaluation for one item in the review text, so it contains a good deal of useful information. By analyzing review text, it can provide more comprehensive information for consumers, and also provide merchants with understanding of customer needs and product decision support.
For this project, I will try to analyze the review text of the items. I will extract features from the review text and find one proper classifier to do the classification work. Then use the classifier to predict the rating of the item based on the item's review text. At the end of the project, I will have one trained classifier. We can enter the review text and the classifier will provide the predicted rating of this review text.

## II. Prior work

For this project, I will focus on analyzing the review text and find whether this review text is positive or negative or neutral. There are three papers I have cited.
The first paper is A Survey on Sentiment Analysis Algorithms for Opinion Mining. This paper introduces overall methods for the sentiment analysis and the details of different approaches.[1]
The second paper is Sentiment Analysis of Twitter Data. This paper introduces the methods about how to preprocess the data, how to do features extraction and analysis. [2] This provides me with some ideas to do the review text analysis.
Third paper is Third paper is Beyond the Stars: Improving Rating Predictions using Review Text Content. This paper introduces how to review text classification and how to do the rating prediction.[3] This provides me with some ideas to do the review text analysis and rating prediction.

## III. Methods

## 1. Obtain the dataset and description

I use the Amazon dataset which is downloaded from Amazon's official dataset website. The dataset is.JSON file.
There are several attributes in the file and I will mainly use two attributes: reviewText and overall. (reviewText is the text of the review, overall is the rating of the product.)

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.
He is having a wonderful time playing these old hymns.  The music  is
at times hard to read because we think the book was published for
singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


## 2. Preprocess the review text

I will get the review text content and the corresponding rating from the.JSON file.

```
content = one['reviewText']  # review text content
rating_score = one['overall']  # ratings
documents.append((content, rating_score))  # content and rating pair
filter_words = do_preprocess_data(content)  # do pre process
```

I will then preprocess for the review text.

Firstly, do word tokenization which will divide the sentence into words.

Secondly, lower these words which make the words as lowercase form.

Thirdly, I will do lemmatization which is to normalize the words and make it easy to analyze the text content. For example, words like playing, played, plays will be transferred to word play. Words like am, are, will be transferred to word be.

Fourth since I want to filter those unuseful characters and get only alpha words.

Fifth, I will get the stop words list which is not useful for the following analysis. I got the stop word list from NLTT corpus. But there are some useful words like: not, never. And I reserve these words.

Sixth, I filter these stop words.

The following is the core code to do preprocess work.

```
# make preprocess
def do_preprocess_data(content):
    tokens_content = do_word_tokenize(content)  # word tokenize
    lower_words = lower_work(tokens_content)  # lower the words
    words_tags = pos_tag(lower_words)  # lemmatization
    word_net_le = WordNetLemmatizer()
    words = []
    for tag in words_tags:
        word_pos = get_pos(tag[1]) or wordnet.NOUN
        words.append(word_net_le.lemmatize(tag[0], pos=word_pos))
    alpha_words = alphabetical_words(words)  # filter non-alpha characters
    stopwords = get_stopwords_list()  # get stopwords list
    filtered_words = [w for w in alpha_words if w not in stopwords]  # filter stop words
    return filtered_words
```

## 3. Show the word cloud of the overall words

For here, I get the overall words list, rating 1 and 2 review text words list, rating 3 review text words list, rating 4 and 5 review text words list. Then I draw the word cloud graph for different words list.

The graphs will be shown in the results.

## 4. Features extraction

For this step, I will extract the features from the documents. I have received all the word lists so far, so I can calculate the frequent words and use the top 2000 frequent words as features.

```
# get set of words for features
def word_features_generate(all_words_list):
    all_words = nltk.FreqDist(all_words_list)
    word_items = all_words.most_common(2000)
    word_features = [word for (word, freq) in word_items]
    return word_features
```

Then I use bags of words model. For this model, it will count each features words' number and get a matrix which represents the features of each document.

```
# get feature
def linear_document_features(document, word_features, features, index):
    document_words = set(document)
    num = 0
    for word in word_features:
        if word in document_words:
            features[index][num] = features[index][num] + 1
        num = num + 1
```

## 5. Three models

For this step, I have three models: linear regression, logistic regression and SVM model. Firstly, I will get the feature of each document and the corresponding ratings. And then I will divide the data into two sets: train set and test set. For the train set, it will be used to train the classifier to let the classifier learn from the data. And for the test set, it will be used to test the classifier and evaluate whether the classifier is good or not.

```
# use the feature to train the classifier
def linear_classifier_train(documents, word_features, features):
    index = 0
    rating_values = []
    for (d, c) in documents:
        linear_document_features(d, word_features, features,index)  # get features
        rating_values.append(c)
        index = index + 1
    X_train, X_test, Y_train, Y_test = features[2000:], features[:2000],  rating_values[2000:], rating_values[:2000]  # d
```

Following I will introduce the three models.

### 5.1. linear regression model

The multi variable linear regression model uses multiple variables to determine one variable. It will construct a regression equation that uses multiple independent variables to estimate one variable to explain and predict the value of the variable. The normal formula for the linear regression model is as follows:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \mu$$

For here, Y is dependent variable, X is independent variable and is the constant term.
For the implementation of this model, I will use the sklearn's linear model.

```
model = linear_model.LinearRegression()  # linear regression model
model.fit(X_train, Y_train)
```

### 5.2. Logistic regression model

The main idea of linear regression is to fit a straight line from historical data and use this straight line to predict new data. For logistic regression, it is also based on the idea of linear regression. The formula is as follows:

$$h_\theta(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta^T x}}$$

The Sigmoid function is:

$$y = \frac{1}{1+e^{-x}}$$

Logistic Regression algorithm maps the linear function's result to the sigmoid function.
For the implementation of this model, I will use the sklearn's logistical regression model.

```
model = linear_model.LogisticRegression()  # logistic regression model
model.fit(X_train, Y_train)
```

### 5.3. SVM model

SVM is a two-class classification model. Its basic model is defined as the linear classifier with the largest interval in the feature space. The learning strategy is to maximize the interval, which can be finally transformed into a solution of a convex quadratic programming problem.
For the implementation of this model, I will use the sklearn's SVM model.

```
model = svm.SVR()  # SVM model
model.fit(X_train, Y_train)
```

## 6. Evaluations of these three models

For the evaluation, I use mean square error and mean absolute error to evaluate whether the model is fit to the data or not.
For the linear regression model, the MSE and MAE are as follows:

```
MSE is: 0.7819114455942594
MAE is: 0.6807069615039792
```

For the logistics model, the MSE and MAE are as follows:

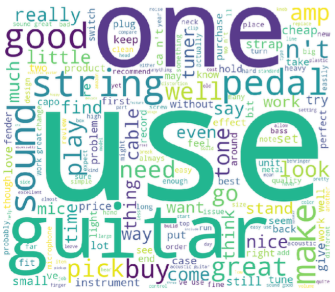
```
MSE is: 1.1475
MAE is: 0.5225
```

For SVM, the MSE and MAE are as follows:

```
MSE is: 0.9569033834908092
MAE is: 0.534736979945943
```

# IV. Results

I got the overall word list and got the word cloud graph. Following is the graph:




I also got the word cloud for rating 4 and 5 review text. Following is the graph:

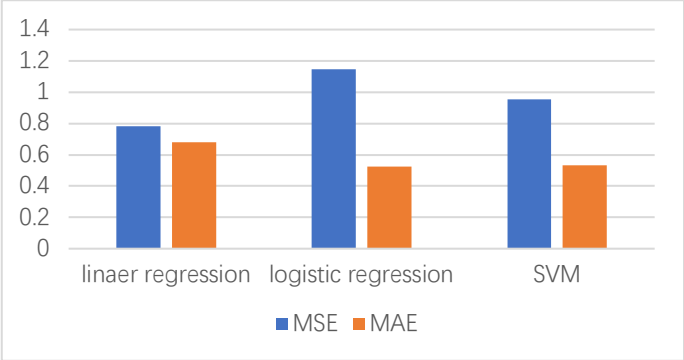

I got the word cloud for rating 1 and 2 review text. Following is the graph:


From above word cloud graphs, there are so many similar words among the positive words set and negative words set. We can try to use the TF-IDF method to do the feature extraction instead of bags of words.

We also have MAE and MSE evaluations for the three models.

I use TF-IDF which is term frequency and inverse frequency. This will decrease the weight of those words which appear frequently in all categories or in many categories.

Bags of words feature:




TF-IDF feature:


From the above evaluation, the logistic regression's MAE is the least, and linear regression's MSE is the least. I will use logistic regression as the final classifier.

If we enter a sentence and get the sentence's features, then we can use the classifier to analyze whether the sentence is negative, neutral or positive and predict the rating. For example, if we enter sentences: I like this guitar very well, the classifier will return its prediction: 4.63.

For this analysis, I just consider the review text and do not consider other interesting attributes like summary of the product, time slice. And in the future, I will consider these attributes and predict the rating according to more attributes.

# V. Reference

[1]. A Survey on Sentiment Analysis Algorithms for Opinion Mining. Vidisha M. Pradhan, Jay Vala, Prem Balani. International Journal of Computer Applications (0975 – 8887) Volume 133 — No.9, January 2016.

[2]. Sentiment Analysis of Twitter Data. Apoorv Agarwal Boyi Xie Ilia Vovsha Owen Rambow Rebecca Passonneau. Department of Computer Science Columbia University, New York, NY 10027 USA

[3]. Beyond the Stars: Improving Rating Predictions using Review Text Content. Gayatree Ganu, Noemie Elhadad, Amelie Marian. WebDB - 2009 - Citeseer