IDS 575 Statistical Models and Methods

Term project- Mid Report

Project Title: Automated product score generation based on sentiment analysis on customer reviews

Contents:

PRE-PROCESSING RAW TEXT

Objective 1 of my project is to generate features from raw text. Features in text analysis are parameters that quantify the importance of a term in a given document or among a collection of documents. The user generated text collected under the “reviewText” column of Amazon product data consists of words, numbers, punctuations and special characters. Just like any other statistical modelling process, cleaning the raw text and making it usable for modelling consumes the maximum time than the actual modelling.

Pre-processing starts with converting the entire document to lower case to support the following steps involved in feature generation.

1. TOKENIZATION: The user reviews are on an average 2 sentences long. In order to break the running sentences, I have used the Natural Language Tool Kit (NLTK) package in python. The method used to tokenize is:

```
nltk.word_tokenize(sentence)
```

Running the sample of 5000 reviews through the tokenizer produced **204862 individual tokens**.

2. STEMMING: The process of reducing all derived words to their root form is called stemming. This will avoid the effect of plurals, adverbs while assigning weights to words based on their occurrence rate. There are different types of Stemmers (Porter stemmer, Snowball stemmer etc) or Lemmetizers to identify and eliminate all derived words. The stemmer used in my text is PorterStemmer. Examples of stemmer results are *forgotten* is reduced to *forget*. Below is the method used:

```
PorterStemmer().stem(words)
```

The 204862 individual tokens obtained in Step 1 are now reduced to **199518 stemmed tokens.**

3. Removing Punctuation: A targeted regex function to retain only words that had alphabets at the start and end helped remove almost 90% of the punctuations that were tokenized. The remaining punctuations like "\" and "-" between words needed regex functions specific to the symbols.

```
[re.sub(r'(\b)-(\b)',",x)for words in file
[re.sub(r'[a-z]\.' '.x)for words in file]
```

The regex functions removed 4442 punctuation tokens resulting in **195076 punctuations free list of tokens.**

4. Removing Stopwords: Words that do not carry meanings but are required to form meaningful sentences in English are called stopwords. Examples: "and", "of", "the" etc. Stop words rank the highest when the frequency of tokens in a document is calculated. NLTK provides methods to eliminate stop words in different commonly used languages.

```
stopwords.words('english')
```

Removing stop words are duplicate tokens resulted in a **vocabulary set of size 166888 tokens.**

5. N-Grams: Removing stopwords, punctuations and stemming rips off the contextual meaning of the tokens, thus the central meaning of a document is fragmented and lost. N-gram construction is a method to combine processed tokens to retain the core meaning of the parent sentence. For computational purposes, I have used Bi-grams (pair of words) as part of my vocabulary set.

```
bigrams = list(nltk.bigrams(tokens))
```

Adding bi-grams increased the vocabulary set size to **169042** (unique uni-grams + bi-grams).

Feature Generation:

Term Frequency- A straight forward method of finding the importance of a term in a document is by counting the number of times the term has occurred. TF can be expressed as $\boldsymbol{tf(t,d) = c(t,d)}$ where t – term; d – document; c(t,d) – count/frequency of the term in the document.

Document Frequency- In a collection of documents I.e., each review being considered as a document here, the number of documents a particular term appears is called Document Frequency of the term which is given by $\boldsymbol{df(t) / N}$ where N is the total number of documents in the collection.

Inverse Document Frequency- Assigning higher weightage to terms that have low document frequency intuitively mean such terms are highly discriminative of themes or central ideas of documents. This is the Inverse Document Frequency given by the expression below.

$$IDF(t) = 1 + \log(\frac{N}{df(t)})$$

Where log is for normalization purposes.

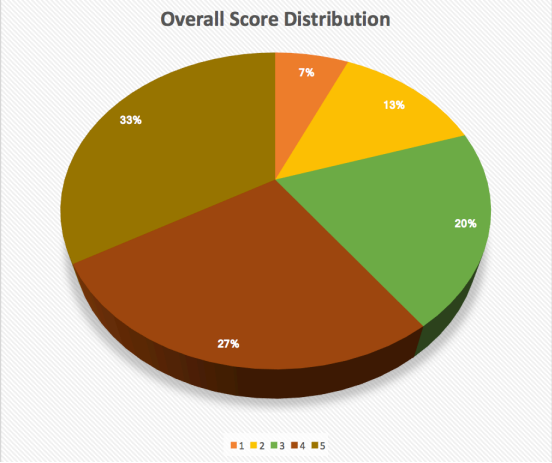
TF-IDF Matrix- Each term in our vocabulary set takes a column position to represent a feature and each document takes a row in forming the TF-IDF matrix. The value in each cell is the combination of term frequency and inverse document frequency. The objective behind taking a combined value is that tf denotes high significance within a document and idf denotes rare occurrence that can be highly discriminative. The expression for combined tf-idf values is $\boldsymbol{w(t,d) = TF(t,d) \times IDF(t)}$**.** The method used to construct this

matrix is *TfidfVectorizer* that is imported from the package *sklearn.feature_extraction.text.*

```
TfidfVectorizer(tokenizer=lambda i:i, lowercase =False, max_df=0.99,
min_df=0.01, use_idf='True')
```

INITIAL STEPS FOR OBJECTIVE 2

My objective 2 is to build a logistic regression model to test the hypothesis- *Sentiment analysis of customer reviews is a direct measure of the overall product score.* With a critical assumption that the overall score in training data is a standardized representation of the scores with very minimal subjectivity, an exploratory analysis on the existing distribution of overall scores is very insightful.



The distribution of scores is not equal as shown by the chart above. Thus, sampling the original data to represent a normal distribution of scores will help the learning algorithm learn the coefficients without bias.