Week12_Sentiment_Analysis

April 18, 2018

1 Sentiment Analysis in Python

Sentiment analysis is not topic analysis. Topic analysis is concerned with identifying topics shared among similar documents or reviews. Sentiment analysis is about measuring the emotional sentiment in documents or reviews. Topic analysis is concerned with all of the terms found in the corpus, apart from terms found in the stop list. Sentiment analysis is only concerned with terms that are know to carry emotional content, either positive or negative.

Terms such as good, happy, honest and love are examples of positive sentiment words. Terms such as bad, unhappy, dishonest and hate are their opposites, and are referred to as negative sentiment terms.

Sentiment analysis identifies and counts the sentiment words found in a document and then scores them to develop a score that on average reflects the emotional content of a document. High sentiment scores represent positive emotional content and low score represent negative emotional content.

1.0.1 Import Statements

All of the import statments have been used previously for topic analyis. The main package used is the sklearn CountVectorier which is used to prepare the term frequency matrix. In this example we will use word clouds to describe sentiment. The last four imports support display and developement of words clouds.

```
In [1]: import pandas as pd
        import numpy  as np
        import string
        from sklearn.feature_extraction.text import CountVectorizer

        import matplotlib.pyplot as plt
        from wordcloud import WordCloud, STOPWORDS
        from PIL import Image
        import random
```

1.0.2 The Data

There are two sets of data.

GMC_Complaints.xlsx, and afinn_sentiment_words.xlsx.

This first is a collection of complaints filed with the NTHSA regarding their automoble. Many of these were involved in accidents, and it's not surprising to find complaints with negative sentiment.

The second is a list of sentiment words prepared by A. Finn. These are words associated with emotions. The list contains these words and points associated with each word. Include are negative variations. The word 'good', for example, also includes 'not good'. In the Sentiment Word List used in this example, the word 'good' has 2 points since it is considered to convey positive sentiment. The term 'not good' conveys negative sentiment. It's score is -1.5 .

```
In [2]: # Increase column width to let pandy read large text columns
        pd.set_option('max_colwidth', 32000)
        # Read N=13,575 California Cabernet Savignon Reviews
        file_path = '/Users/Home/Desktop/python/Excel/'
        df = pd.read_excel(file_path+"GMC_Complaints.xlsx")
        sw = pd.read_excel(file_path+"afinn_sentiment_words.xlsx")
```

1.0.3 Sentiment Dictionary

The sentiment words and their points are captured in a Python dictionary for easy lookup during document processing.

The dictionary termed sentiment_dic use the individual sentiment words and phrases as the dictionary keys and the points as their value.

```
In [3]: # Setup Sentiment dictionary
        sentiment_dic = {}
        for i in range(len(sw)):
            sentiment_dic[sw.iloc[i][0]] = sw.iloc[i][1]

        print(sentiment_dic['good'], sentiment_dic['not good'])
```

```
3.0 -1.5
```

1.0.4 Create the Term Frequency Matrix

The term frequency matrix is created as before using the GMC_Complaints data file. However, there are a few parameters that are set differently from topic analysis.

The parameter max_df is used to discard terms that appear too frequently. A value of 0.5 would discard all terms that appear in more than half of all documents. In sentiment analysis, we do not want to discard any sentiment words. All of them must be counted, even if they appear in all documents. For that reason, it is necessary to set max_df to 1.0.

Similarly the parameter min_df controls rare words. By default all terms that appear in only one document are discared, and again no words should be discarded in sentiment analysis. Hence min_df is set to 1.

Finally the ngram_range parameter should be set to (1,2) which asks for the extraction of word pairs as well as individual words.

As part of collecting the term frequency matrix it will be necessary to do some preprocessing of the data. In particular words such as wasn't need to be changed to was not. This is important

since the word not changes the sentiment from positive to negative or visa versa. This is done using a user defined function for preprocessing. In this example it is called my_preprocessor</>.

```
In [4]: def my_preprocessor(s):
            # Preprocess String s
            s = s.lower()
            # Replace special characters with spaces
            s = s.replace('-', ' ')
            s = s.replace('_', ' ')
            s = s.replace(',', '. ')
            # Replace not contraction with not
            s = s.replace("'nt", " not")
            s = s.replace("n't", " not")
            return s

        cv = CountVectorizer(max_df=1.0, min_df=1, max_features=None, \
                             preprocessor=my_preprocessor, ngram_range=(1,2))
        tf    = cv.fit_transform(df['description'])
        terms = cv.get_feature_names()
        n_reviews = tf.shape[0]
        n_terms   = tf.shape[1]
        print('{:.<22s}{:>6d}'.format("Number of Reviews", n_reviews))
        print('{:.<22s}{:>6d}'.format("Number of Terms",   n_terms))
```

```
Number of Reviews...  2734
Number of Terms... 84235
```

1.0.5 Sentiment Scoring

Now that the term frequency matrix is created along with the dictionary of sentiment words, each document can be scored.

In python the term frequency matrix is stored in CSR format, Compressed Sparce. No zeros are stored in the term frequency matrix. This makes sentiment calculations fast and efficient.

This code not only calculates the average sentiment for each document it also find the overall average sentiment for the collection of complaints and the particular complaints that were most negative and postive among the complaints with at least 4 sentiment words.

```
In [5]: min_sentiment = +5
        max_sentiment = -5
        avg_sentiment, min, max = 0,0,0
        min_list, max_list = [],[]
        sentiment_score = [0]*n_reviews
        for i in range(n_reviews):
            # Iterate over the terms with nonzero scores
            n_sw = 0
            term_list = tf[i].nonzero()[1]
            if len(term_list)>0:
                for t in np.nditer(term_list):
```

```
                score = sentiment_dic.get(terms[t])
                if score != None:
                    sentiment_score[i] += score * tf[i,t]
                    n_sw += tf[i,t]
        if n_sw>0:
            sentiment_score[i] = sentiment_score[i]/n_sw
        if sentiment_score[i]==max_sentiment and n_sw>3:
            max_list.append(i)
        if sentiment_score[i]>max_sentiment and n_sw>3:
            max_sentiment=sentiment_score[i]
            max = i
            max_list = [i]

        if sentiment_score[i]==min_sentiment and n_sw>3:
            min_list.append(i)
        if sentiment_score[i]<min_sentiment and n_sw>3:
            min_sentiment=sentiment_score[i]
            min = i
            min_list = [i]
        avg_sentiment += sentiment_score[i]
    avg_sentiment = avg_sentiment/n_reviews
    print("\nCorpus Average Sentiment: ", avg_sentiment)
    print("\nMost Negative Reviews with 4 or more Sentiment Words:")
    for i in range(len(min_list)):
        print("{:<s}{:<d}{:<s}{:<5.2f}".format("    Review ", min_list[i],  \
              " Sentiment is ", min_sentiment))

    print("\nMost Positive Reviews with 4 or more Sentiment Words:")
    for i in range(len(max_list)):
        print("{:<s}{:<d}{:<s}{:<5.2f}".format("    Review ", max_list[i],  \
              " Sentiment is ", max_sentiment))
```

```
Corpus Average Sentiment:  -1.2721691806

Most Negative Reviews with 4 or more Sentiment Words:
    Review 2396 Sentiment is -2.80

Most Positive Reviews with 4 or more Sentiment Words:
    Review 349 Sentiment is 1.17
```

It is not surprising that the corpus sentiment is negative. On average, we can confirm that the complaints registered with the NTSHA have negative emotional content. That is they people submitting these are unhappy. However, there are some with positive sentiment.

1.0.6 Sentiment Words Clouds

It is intersting to ask what words produced these sentiments. The following code produces four word clouds, two for the most negative complain and two for the most positive.

The first of each pair is a cloud constructed using all of the words in the complaint. This will be busy since we have not removed any stop words. The second could will only use the sentiment words found in the complaint.

The first production of these clouds found several stop words that confused the image. These are included in the stop word list to produce a more meaningful word cloud.

```
In [6]: stopw = set(STOPWORDS)
        stopw.add("consumer")
        stopw.add("vehicle")
        stopw.add("information")
        stopw.add("request")
        stopw.add("records")
        stopw.add("issue")
        stopw.add("records")
        stopw.add("becomes")
        stopw.add("design")
        stopw.add("system")
```

The word clouds will use the circle mask instead of a rectangle cloud. In addition, the colors of the word will be change to shade of grey to make them easier to see.

```
In [7]: def shades_of_grey(word, font_size, position, orientation, random_state=None, \
                           **kwargs):
            return "hsl(0, 0%%, %d%%)" % random.randint(60,1000)
        circle_mask = np.array(Image.open("CircleMask.png"))
```

First the word clouds for the complaint with the most negative sentiment are displayed. The background color is maroon, and the words are colored in shades of grey. This combination makes them easier to read, but some still have very small fonts.

The first cloud is a cloud using all of the words in the review. As a result, the sentiment words are not obvious.

The second cloud only uses the sentiment words, now it is clearer why this review is negative.

```
In [8]: wcm = WordCloud(background_color="maroon", max_words=200, stopwords=stopw, \
                        mask=circle_mask, random_state=341)
        wcm.generate(df['description'].iloc[min_list[0]])
        print("\nMost Negative Review Sentiment=", min_sentiment)
        plt.imshow(wcm.recolor(color_func=shades_of_grey, random_state=3), \
                               interpolation="bilinear")
        plt.axis("off")
        plt.figure()

        s=""
        i = min_list[0]
        term_list = tf[i].nonzero()[1]
```

```
        if len(term_list)>0:
            for t in np.nditer(term_list):
                score = sentiment_dic.get(terms[t])
                if score != None:
                    s += terms[t] + " "
        wcm.generate(s)
        plt.imshow(wcm.recolor(color_func=shades_of_grey, random_state=3), \
                               interpolation="bilinear")
        plt.axis("off")
        plt.figure()
        plt.show()
```

```
Most Negative Review Sentiment= -2.8
```

```
<matplotlib.figure.Figure at 0x1a0ff832e8>
```

Next the same words clouds are produced for the complaint with the highest positive sentiment.

```
In [9]: wcm.generate(df['description'].iloc[max_list[0]])
        print("\nMost Positive Review Sentiment=", max_sentiment)
        plt.imshow(wcm.recolor(color_func=shades_of_grey, random_state=3), \
                               interpolation="bilinear")
        plt.axis("off")
        plt.figure()

        s=""
        i = max_list[0]
        term_list = tf[i].nonzero()[1]
        if len(term_list)>0:
            for t in np.nditer(term_list):
                score = sentiment_dic.get(terms[t])
                if score != None:
                    s += terms[t] + " "
        wcm.generate(s)
        plt.imshow(wcm.recolor(color_func=shades_of_grey, random_state=3), \
                               interpolation="bilinear")
        plt.axis("off")
        plt.figure()
        plt.show()
```

```
Most Positive Review Sentiment= 1.16666666667
```


```
<matplotlib.figure.Figure at 0x1a0fef7160>
```

1.0.7 Corpus Sentiment Word Cloud

It is useful to look at the sentiment words for individual documents. From this you can identify possible problems with stop words or synonyms. The synonyms can be handed inside the preprocess function - my_preprocessor. Stop words can be removed before producing the word cloud.

A final cloud showing the distribution of all of the sentiment words over the entire corpus might be useful. We will create a dictionary for all of the sentiment terms found in the corpus together with a count of how many were found. This can be set to wordcloud to produce a cloud displaying all of the sentiment words for this corpus.

This could have been created in the original loop over the entire data set, but the process is fairly fast because of the sparse term frequency matrix.

```
In [10]: corpus_sentiment = {}
         n_sw = 0
         for i in range(n_reviews):
             # Iterate over the terms with nonzero scores
             term_list = tf[i].nonzero()[1]
             if len(term_list)>0:
                 for t in np.nditer(term_list):
                     score = sentiment_dic.get(terms[t])
                     if score != None:
                         n_sw += tf[i,t]
                         current_count = corpus_sentiment.get(terms[t])
```

```
                    if current_count == None:
                        corpus_sentiment[terms[t]] = tf[i,t]
                    else:
                        corpus_sentiment[terms[t]] += tf[i,t]
        print("The Corpus contains a total of ", len(corpus_sentiment), " unique sentiment word
        print("The total number of sentiment words in the Corpus is", n_sw)
```

```
The Corpus contains a total of  754  unique sentiment words
The total number of sentiment words in the Corpus is 13900
```

Since the corpus_sentiment dictionary contains the sentiment for the entire corpus we can use it directly to produde a word cloud without tokenization or removing stop words.

Notice an important change. The previous word clouds operated on the text using the wcm.generate( ) command. That command tokenizes the text and removes stop words before displaying the cloud. Included in this process is developing a table of word counts for each term.

In this example, the table is already created in the form of a dictionary. With that we use the wcm.generate_from_frequencies command display the word cloud.

```
In [11]: wcm = WordCloud(background_color="maroon", max_words=200, \
                          mask=circle_mask, random_state=341)
         wcm.generate_from_frequencies(corpus_sentiment)
         plt.imshow(wcm.recolor(color_func=shades_of_grey, random_state=3), \
                                interpolation="bilinear")
         plt.axis("off")
         plt.figure()
         plt.show()
```


```
<matplotlib.figure.Figure at 0x1a100e3358>
```

The sentiment word failure has negative sentiment with a score of -2. It appears to be the sentiment word used most frequency. The second most used word is problem which also as a score of -2. The word no by is also a negative sentiment word, but with a score of only -1.