Predicting_Sentiment_and_Helpfulness

May 20, 2017

1 Making predictions over amazon recommendation dataset

The purpose of this analysis is to make up a prediction model where we will be able to predict whether a recommendation is positive or negative. In this analysis, we will not focus on the Score, but only the positive/negative sentiment of the recommendation. In the end, we hope to find a "best" model for predicting the recommendation's sentiment.

1.1 Loading the data

As we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "postive". Otherwise, it will be set to "negative".

The data will be split into an training set and a test set with a test set ratio of 0.2

In [1]:
```
%matplotlib inline

import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
```

In [3]:
```
df = pd.read_table('reviews_2.csv',sep=',',header=None)
df.columns = ['ID','numDate','prod','overall','helpful','votes',
              'date','asin','summary','reviewText']
df.head()
```

Out[3]:

	ID	numDate	prod	overall	helpful	votes	date
0	0	20161030	NaN	2	0	0	on October 30, 2016
1	1	20161030	NaN	4	0	0	on October 30, 2016
2	2	20161029	NaN	5	0	0	on October 29, 2016
3	3	20161028	NaN	5	0	0	on October 28, 2016
4	4	20161027	NaN	5	0	0	on October 27, 2016

```
              asin                                          summary  \
0   A4TDOTZKPA79G  but product arrived in excellent condition.
1   AIR0R7XJECA87                      Not fun to put together
2  A273FNRW9W4RIP                         Get This Cool Trike!
3   AVRR1KJLVE3FF                                     Love it!
4   AN8M8X07W9NXX                                  Sweet ride.

                                          reviewText
0  whobbly to ride.  going around corners VERY ca...
1  Not fun to put together! You have to do a lot ...
2  All Senior Citizens need to own this trike!  V...
3  Needed metric tools to put it together, but on...
4  We bought this, added a boat seat, 80cc gas mo...
```

```
In [4]: messages=df;
        messages["Sentiment"] = messages["overall"].apply(
        lambda score: "positive" if score > 3 else "negative")

        messages["Usefulness"] = (messages["helpful"]/messages["votes"]).apply(
        lambda n: "useful" if n > 0.8 else "useless")
```

2 Extracting features from text data

SciKit cannot work with words, so we'll just assign a new dimention to each word and work with word counts. See more here: http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

```
In [8]: from sklearn.feature_extraction.text import
        CountVectorizer, TfidfTransformer

        import re
        import string
        import nltk

        cleanup_re = re.compile('[^a-z]+')
        def cleanup(sentence):
            sentence = sentence.lower()
            sentence = cleanup_re.sub(' ', sentence).strip()
            #sentence = " ".join(nltk.word_tokenize(sentence))
            return sentence

        messages["Summary_Clean"] = messages["summary"].apply(cleanup)
        # messages["Summary_Clean"] = messages["reviewText"].apply(cleanup)

        train, test = train_test_split(messages, test_size=0.2)
        print("%d items in training data, %d in test data
```

```
536 items in training data, 134 in test data
```

```
In [9]: from wordcloud import WordCloud, STOPWORDS
```

```
        # To cleanup stop words, add stop_words = STOPWORDS
        # But it seems to function better without it
        count_vect = CountVectorizer(min_df = 1, ngram_range = (1, 4))
        X_train_counts = count_vect.fit_transform(train["Summary_Clean"])

        tfidf_transformer = TfidfTransformer()
        X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

        X_new_counts = count_vect.transform(test["Summary_Clean"])
        X_test_tfidf = tfidf_transformer.transform(X_new_counts)

        y_train = train["Sentiment"]
        y_test = test["Sentiment"]

        prediction = dict()
```

```
In [10]: from wordcloud import WordCloud, STOPWORDS
         stopwords = set(STOPWORDS)

         #mpl.rcParams['figure.figsize']=(8.0,6.0)    #(6.0,4.0)
         mpl.rcParams['font.size']=12                #10
         mpl.rcParams['savefig.dpi']=100             #72
         mpl.rcParams['figure.subplot.bottom']=.1


         def show_wordcloud(data, title = None):
             wordcloud = WordCloud(
                 background_color='white',
                 stopwords=stopwords,
                 max_words=200,
                 max_font_size=40,
                 scale=3,
                 random_state=1
             ).generate(str(data))

             fig = plt.figure(1, figsize=(8, 8))
             plt.axis('off')
             if title:
                 fig.suptitle(title, fontsize=20)
                 fig.subplots_adjust(top=2.3)

             plt.imshow(wordcloud)
             plt.show()

         show_wordcloud(messages["Summary_Clean"])
```

In [11]: show_wordcloud(messages[messages.overall == 1]["Summary_Clean"], title = "Low scoring")

Low scoring

In [12]: show_wordcloud(messages[messages.overall == 5]["Summary_Clean"], title = "High scoring")

High scoring

In [13]:
```
from sklearn.pipeline import FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
```

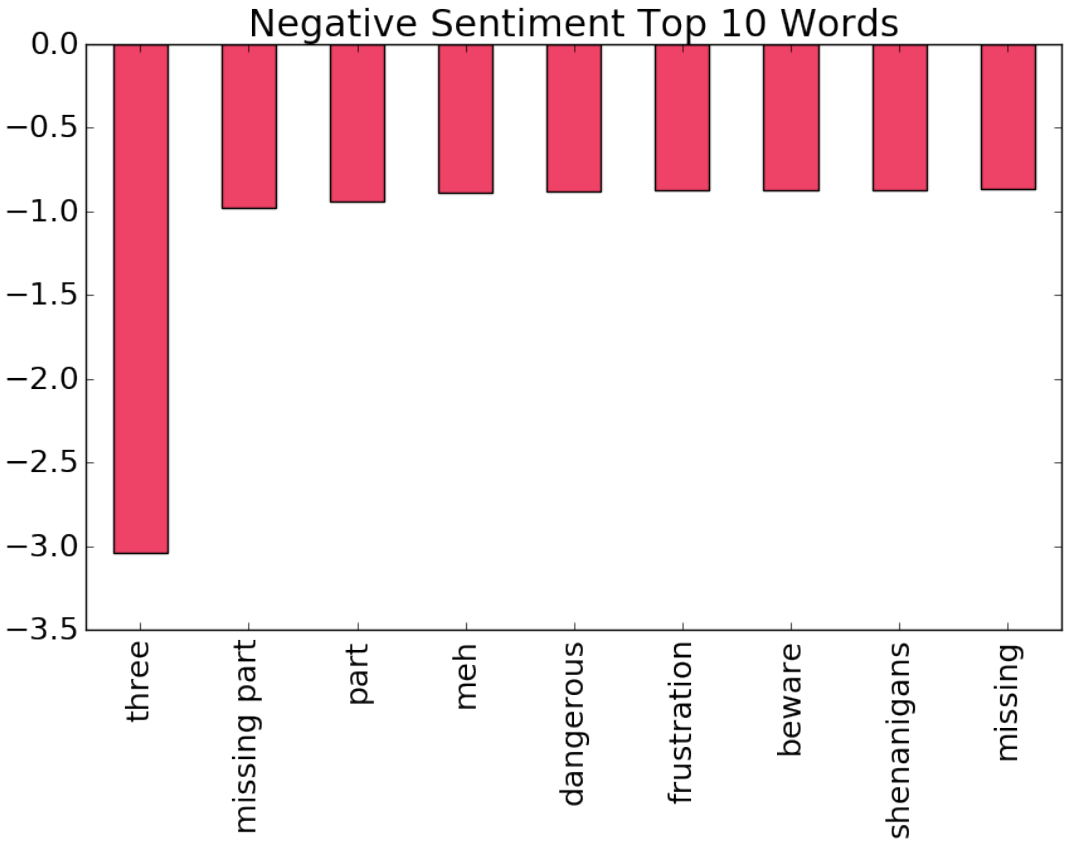
```
        # Useful to select only certain features in a dataset for
        #  forwarding through a pipeline

        class ItemSelector(BaseEstimator, TransformerMixin):
            def __init__(self, key):
                self.key = key
            def fit(self, x, y=None):
                return self
            def transform(self, data_dict):
                return data_dict[self.key]

        train_ufn2, test_ufn2 = train_test_split(messages, test_size=0.2)

        ufn_pipe2 = Pipeline([
           ('union', FeatureUnion(
               transformer_list = [
                   ('summary', Pipeline([
                       ('textsel', ItemSelector(key='Summary_Clean')),
                       ('vect',CountVectorizer(min_df = 1,ngram_range = (1, 4))),
                       ('tfidf', TfidfTransformer())])),
                  ('score', ItemSelector(key=['overall']))
               ],
               transformer_weights = {
                   'summary': 0.2,
                   'score': 0.8
               }
           )),
           ('model', LogisticRegression(C=1e5))
        ])

        ufn_result2 = ufn_pipe2.fit(train_ufn2, train_ufn2["Sentiment"])
```

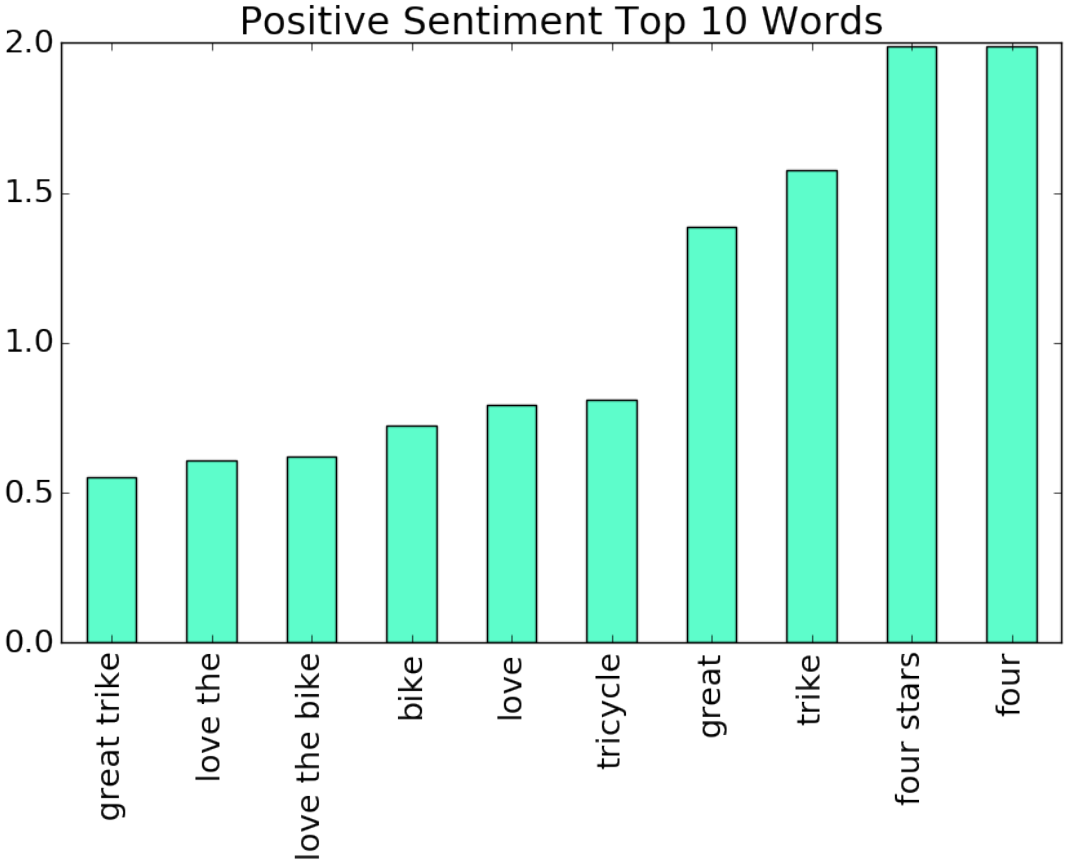
In [ ]:
```
ufn_summary_pipe = next(tr[1] for tr in ufn_result2.named_steps["union"]
ufn_words = ufn_summary_pipe.named_steps['vect'].get_feature_names()
ufn_features = ufn_words
ufn_feature_coefs = pd.DataFrame(
    data = list(zip(ufn_features, ufn_result2.named_steps['model'].
                    coef_[0])),columns = ['feature', 'coef'])
ufn_feature_coefs.sort_values(by='coef')
```

In [21]:
```
df=ufn_feature_coefs.sort_values(by='coef')
type(df)
word_least=df[1:10]
word_most=df[-10:]
```

In [25]:
```
%matplotlib inline
import matplotlib.pyplot as plt
word_least.plot(kind='bar',color='#EE4266',x=word_least['feature'],
legend=False,title='Negative Sentiment Top 10 Words',figsize=(10,6))
word_most.plot(kind='bar',color='#5DFDCB',x=word_most['feature'],
legend=False,title='Positive Sentiment Top 10 Words',figsize=(10,6))
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24179707208>
```


Negative Sentiment Top 10 Words
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
three
missing part
part
meh
dangerous
frustration
beware
shenanigans
missing


Positive Sentiment Top 10 Words
2.0
1.5
1.0
0.5
0.0
great trike
love the
love the bike
bike
love
tricycle
great
trike
four stars
four