4 - Sentiment Analysis

November 29, 2021

Table of Contents

0.1 Objective

Can we create a model that will accurately assess the sentiment of incoming review comments. This assumes we will not have a rating column.

[ ]:
```
#from google.colab import drive
#drive.mount('/content/drive')
```

0.1.1 Load the data into a dataframe

[ ]:
```
import pandas as pd
```

[ ]:
```
latuda = pd.read_csv('https://raw.githubusercontent.com/jimcody2014/nlp_cdc/
 ↪main/data/latuda.csv')
```

[ ]:
```
latuda.columns
```

[ ]:
```
latuda.head()
```

0.2 Pre-processing

[ ]:
```
!pip install contractions
!pip install pyspellchecker

import matplotlib.pyplot as plt
```

```python
import seaborn as sns
import numpy as np
import contractions
#from pyspellchecker import SpellChecker
import string
import re
from textblob import TextBlob, Word

import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')

from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.corpus import wordnet
```

[ ]:
```python
# Add a column 'target' based on the rating column
latuda['eval'] = latuda['rating'].apply(lambda x: 'Good' if x >= 6 else 'Bad')
#drop_columns = {'drugName', 'condition','date', 'usefulCount', 'rating'}
#latuda = latuda.drop(columns = drop_columns)
latuda['remove_ctr'] = latuda['review'].apply(lambda x: [contractions.fix(word) for word in x.split()])
# change no_contract back to a string
latuda["review_new"] = [' '.join(map(str, l)) for l in latuda['remove_ctr']]
latuda['tokenized'] = latuda['review_new'].apply(word_tokenize)
latuda['lower'] = latuda['tokenized'].apply(lambda x: [word.lower() for word in x])
punc = string.punctuation
latuda['lower'] = latuda['lower'].apply(lambda x: [word for word in x if word not in punc])
```

[ ]:
```python
def clean_text_round1(text):
    #text = text.lower()
    #text = re.sub('\[.*?\]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('[‘’""…]', '', text)
    text = re.sub('\n', '', text)
    return text

round1 = lambda x: clean_text_round1(x)

latuda.review = pd.DataFrame(latuda.review.apply(round1))
```

```
[ ]: stop_words = set(stopwords.words('english'))
     latuda['no_stopwords'] = latuda['lower'].apply(lambda x: [word for word in x if␣
      ↪word not in stop_words])
     latuda['pos_tags'] = latuda['no_stopwords'].apply(nltk.tag.pos_tag)
```

```
[ ]: def get_wordnet_pos(tag):
         if tag.startswith('J'):
             return wordnet.ADJ
         elif tag.startswith('V'):
             return wordnet.VERB
         elif tag.startswith('N'):
             return wordnet.NOUN
         elif tag.startswith('R'):
             return wordnet.ADV
         else:
             return wordnet.NOUN
```

```
[ ]: latuda['wordnet_pos'] = latuda['pos_tags'].apply(lambda x: [(word,␣
      ↪get_wordnet_pos(pos_tag)) for (word, pos_tag) in x])
     wnl = WordNetLemmatizer()
     latuda['lemmatized'] = latuda['wordnet_pos'].apply(lambda x: [wnl.
      ↪lemmatize(word, tag) for word, tag in x])
     latuda.columns
     drop_columns = {'remove_ctr', 'review_new', 'tokenized',␣
      ↪'lower','no_stopwords','pos_tags','wordnet_pos'}
     latuda2 = latuda.drop(columns = drop_columns)
```

0.3 Calculate Polarity

```
[ ]: sample = latuda.review.sample(1).iloc[0]
     print(sample)
```

```
[ ]: TextBlob(sample).sentiment
```

```
[ ]: pol = lambda x: TextBlob(x).sentiment.polarity
     sub = lambda x: TextBlob(x).sentiment.subjectivity

     latuda2['polarity'] = latuda2['review'].apply(pol)
     latuda2['subjectivity'] = latuda2['review'].apply(sub)
```

```
[ ]: latuda2.head()
```

```
[ ]: # A histogram of the polarity scores.
     num_bins = 50
     plt.figure(figsize=(10,6))
     n, bins, patches = plt.hist(latuda2.polarity, num_bins, facecolor='blue',␣
      ↪alpha=0.5)
```

```
plt.xlabel('Polarity')
plt.ylabel('Count')
plt.title('Histogram of polarity')
plt.show();
```

[ ]:
```
# Box plot of sentiment grouped by rating
plt.figure(figsize=(10,6))
sns.boxenplot(x='rating', y='polarity', data=latuda2)
plt.show();
```

[ ]:
```
plt.rcParams['figure.figsize'] = [10, 8]
sns.scatterplot(data = latuda2, x = 'polarity', y='subjectivity', hue = 'eval')
```

[ ]:
```
pd.set_option('max_colwidth', 400)
```

[ ]:
```
latuda2[latuda2.polarity == -1].review.head()
```

[ ]:
```
latuda2[latuda2.rating == 1].review.head()
```

[ ]:
```
latuda2[(latuda2.rating == 5) & (latuda2.polarity <= -0.2)].head(10)
```

[ ]:
```
latuda2[(latuda2.rating == 1) & (latuda2.polarity >= 0.5)].head(10)
```

Is it possible that the words used in medical reviews are too ambiguous for most common lexicons?

0.4 Sentiment Analysis using a logistic regression classifier

Tasks

- Change Good/Bad to 1/-1 (We've already classified the review as good or bad)
- visualize counts for 1/-1
- Create a summary of the review (leave this for later)
- Reduce columns to review and eval
- split the datset to train & test
- create a bag of words
- import the logistic regression from sklearn
- fit the model
- make a prediction
- determine accuracy

[ ]:
```
latuda.columns
```

[ ]:
```
latuda3 = latuda
latuda3['eval'] = latuda['rating'].apply(lambda x: 1 if x >= 6 else -1)
drop_columns = {'drugName', 'condition', 'rating', 'date', 'usefulCount',␣
↪'remove_ctr', 'review_new', 'tokenized', 'lower',
       'no_stopwords', 'pos_tags', 'wordnet_pos', 'lemmatized'}
latuda3 = latuda3.drop(columns = drop_columns)
```

```
latuda3.head()
```

[ ]:
```
# random split train and test data
index = latuda3.index
latuda3['random_number'] = np.random.randn(len(index))
train = latuda3[latuda3['random_number'] <= 0.8]
test = latuda3[latuda3['random_number'] > 0.8]
```

[ ]:
```
# count vectorizer:
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(token_pattern=r'\b\w+\b')
train_matrix = vectorizer.fit_transform(train['review'])
test_matrix = vectorizer.transform(test['review'])
```

[ ]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

[ ]:
```
X_train = train_matrix
X_test = test_matrix
y_train = train['eval']
y_test = test['eval']
```

[ ]:
```
lr.fit(X_train,y_train)
```

[ ]:
```
predictions = lr.predict(X_test)
```

[ ]:
```
# find accuracy, precision, recall:
from sklearn.metrics import confusion_matrix,classification_report
new = np.asarray(y_test)
confusion_matrix(predictions,y_test)
```

- Rows are predicted, columns are actual.

- Top left (pos,pos). True Positive. These are correctly predicted.
- Bottom right (neg,neg). True Negative. These are correctly predicted.
- bottom left (false negative), top right (false positive) are the number of incorrect predictions

[ ]:
```
print(classification_report(predictions,y_test))
```

[ ]:
```
y_test.shape
```

0.4.1 Sentiment Analysis using Deep Learning

[ ]:
```
latuda3.head()
```

[ ]:
```
latuda3.drop('random_number', axis=1, inplace=True)
```

[ ]:
```
from sklearn.model_selection import train_test_split
```

[ ]:
```
X=latuda3['review'].values
```

[ ]:
```
Y=latuda3['eval'].values
```

[ ]:
```
import seaborn as sns
sns.countplot(latuda3['eval'])
```

[ ]:
```
X_train, X_test, Y_train, Y_test= train_test_split(X,Y, test_size=0.3)
```

[ ]:
```
vec = CountVectorizer()
```

[ ]:
```
vec.fit(X_train)
```

[ ]:
```
x_train=vec.transform(X_train)
```

[ ]:
```
x_test=vec.transform(X_test)
```

[ ]:
```
x_train
```

0.4.2 This is the deep learning code

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
model = Sequential()
model.add(Dense(16, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

[ ]:
```
model.summary()
```

[ ]:
```
history = model.fit(x_train, Y_train,epochs=100,verbose=True,batch_size=16)
```

[ ]:
```
model.evaluate(x_train,Y_train)
```

[ ]:
```
model.evaluate(x_test,Y_test)
```

[ ]:
```

```