Eluvio_Challenge

March 3, 2020

Define a Problem

The first thing I wanted to do was analyse the dataset and see what information could be gleaned by a visual inspection of it.

Also I wanted to see if any patterns emerged from processing the given .csv file.

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20http
s%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.reado
nly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

[2]:
```
from textblob import TextBlob
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import nltk
import re
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
nltk.download('stopwords')
nltk.download('punkt')
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("english")
import csv
```

```
from scipy.stats import spearmanr
import sklearn.feature_extraction.text as text
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

[0]:
```
path = "drive/My Drive/Eluvio/Eluvio_DS_Challenge.csv"
df = pd.read_csv(path)
```

First I used Pandas library to visualize the data in a neat manner.

This enabled me to see the fields and features present. I quickly realized upvotes, titles and author could be fields I could utilize to find out the trends in the given data for effective understanding of it.

[4]:
```
df.head()
```

[4]:
```
   time_created date_created  up_votes  …  over_18    author   category
0    1201232046   2008-01-25         3  …    False     polar  worldnews
1    1201232075   2008-01-25         2  …    False     polar  worldnews
2    1201232523   2008-01-25         3  …    False     polar  worldnews
3    1201233290   2008-01-25         1  …    False   fadi420  worldnews
4    1201274720   2008-01-25         4  …    False  mhermans  worldnews

[5 rows x 8 columns]
```

[5]:
```
print('The size of dataset (number of samples) is %i' %len(df))
```

```
The size of dataset (number of samples) is 509236
```

From the block below, it can be gleaned that all the samples belong to the category **'worldnews'**, and all the samples also have **0 downvotes**. In case of a very large dataset, we can remove features which would not help us in any way for classification, or something else. Hence, when we use this data, we should remove or drop these two features.

[6]:
```
print('Number of samples belonging to the category world news is %i'␣
 ↪%sum(df['category'] == "worldnews"))
print('Number of samples having 0 downvotes is %i' %sum(df["down_votes"] == 0))
```

```
Number of samples belonging to the category world news is 509236
Number of samples having 0 downvotes is 509236
```

The block below shows that only a small number of articles are deemed to be for people above 18 years old. Majority of the articles can be read by all age groups.

```
[7]: print('The number of articles for people above 18 years old is %i and articles␣
 ↪for everyone above and below 18 years old is %i' %(sum(df["over_18"] ==␣
 ↪True), sum(df["over_18"] == False)))
```

```
The number of articles for people above 18 years old is 320 and articles for
everyone above and below 18 years old is 508916
```

The two code blocks below show that the articles meant for people above 18 years old gets much more upvotes per article as compared to articles meant for all age groups.

```
[8]: print('Number of upvotes per articles meant to be for people above 18 years old␣
 ↪is %g' %(df.groupby('over_18')['up_votes'].sum()[1]/sum(df["over_18"] ==␣
 ↪True)))
```

```
Number of upvotes per articles meant to be for people above 18 years old is
380.375
```

```
[9]: print('Number of upvotes per articles meant to be for all people is %g' %(df.
 ↪groupby('over_18')['up_votes'].sum()[0]/sum(df["over_18"] == False)))
```

```
Number of upvotes per articles meant to be for all people is 112.068
```

The code block below displays the top ten authors having the maximum number of upvotes.

```
[10]: df.groupby('author')['up_votes'].sum().sort_values(ascending=False).head(10)
```

```
[10]: author
maxwellhill        1985416
anutensil          1531544
Libertatea          832102
DoremusJessup       584380
Wagamaga            580121
NinjaDiscoJesus     492582
madazzahatter       428966
madam1              390541
davidreiss666       338306
kulkke              333311
Name: up_votes, dtype: int64
```

The code block below displays the top ten authors having the most number of posts.

```
[11]: df.groupby('author')['up_votes'].count().sort_values(ascending=False).head(10)
```

```
[11]: author
davidreiss666    8897
anutensil        5730
DoremusJessup    5037
maxwellhill      4023
igeldard         4013
```

```
readerseven      3170
twolf1           2923
madam1           2658
nimobo           2564
madazzahatter    2503
Name: up_votes, dtype: int64
```

The block below displays the top ten authors sorted by the average number of upvotes per post. This lets us know which authors are likely to get more upvotes on their posts.

[12]:
```
((df.groupby('author')['up_votes'].sum())/(df.groupby('author')['up_votes'].
↪count())).sort_values(ascending=False).head(10)
```

[12]:
```
author
navysealassulter       12333.0
seapiglet              11288.0
DawgsOnTopUGA          10515.0
Flamo_the_Idiot_Boy    10289.0
haunted_cheesecake      9408.0
bendertheoffender22     8781.0
crippledrejex           8601.0
FlandersNed             8446.0
lesseva96               8404.0
sverdrupian             8262.0
Name: up_votes, dtype: float64
```

The above output is verified by the following code block. It can be seen that the author who is number 1 on the list preceding this block is 'navysealassulter', and the following block displays the instances where this author's name has appeared. As we can see, he has only one post garnering 12333 upvotes, and this number also aligns with the number mentioned above.

[13]:
```
df[df['author'] == 'navysealassulter']
```

[13]:
```
        time_created date_created  …            author   category
391318    1440367768   2015-08-23  …  navysealassulter  worldnews

 [1 rows x 8 columns]
```

Now that we have finished our visual inspection, we will move on to training a model using basic classifiers like SVM and Logisitic Regression to see how much the title effects the upvotes. Basically, based on title, we will check if the post belongs in the posts having upvotes greater than 85% of the total data.

[0]:
```
df1 = pd.read_csv(path)
```

[15]:
```
df1.head()
```

[15]:
```
   time_created date_created  up_votes  …  over_18    author   category
0    1201232046   2008-01-25         3  …    False     polar  worldnews
1    1201232075   2008-01-25         2  …    False     polar  worldnews
2    1201232523   2008-01-25         3  …    False     polar  worldnews
3    1201233290   2008-01-25         1  …    False   fadi420  worldnews
4    1201274720   2008-01-25         4  …    False  mhermans  worldnews

[5 rows x 8 columns]
```

As we saw before, downvotes and categories are basically redundant features. To make our model simpler, we will also remove time created and date created for the posts.

[0]:
```
df1 = df1.drop("category", axis = 1)
df1 = df1.drop("down_votes", axis = 1)
df1 = df1.drop("time_created", axis = 1)
df1 = df1.drop("date_created", axis = 1)
```

[17]:
```
df1.head()
```

[17]:
```
   up_votes                                          title  over_18    author
0         3             Scores killed in Pakistan clashes    False     polar
1         2              Japan resumes refuelling mission    False     polar
2         3                US presses Egypt on Gaza border    False     polar
3         1      Jump-start economy: Give health care to all    False   fadi420
4         4  Council of Europe bashes EU&UN terror blacklist    False  mhermans
```

Below we will tokenize and stem the dataset using predefined libraries.

[0]:
```
#main function
def ts(title):
    stemmed = []
    tokenized = []
    for i in title:
        stemmed1 = tokenstem(i)
        tokenized1 = token(i)
        stemmed.extend(stemmed1)
        tokenized.extend(tokenized1)
    return stemmed, tokenized

#side functions
def tokenstem(text):
    words1 = []
    words = [word for sent in nltk.sent_tokenize(text) for word in nltk.
 ↪word_tokenize(sent)] #tokenize sentences then word
    for token in words:
        if re.search('[a-zA-Z]', token): #check if it is a word
            words1.append(token)
    stems = [stemmer.stem(t) for t in words1]
```

```
    return stems

def token(text):
    words2 = []
    words = [word for sent in nltk.sent_tokenize(text) for word in nltk.
 ↪word_tokenize(sent)] #tokenize sentences then word
    for token in words:
        if re.search('[a-zA-Z]', token):
            words2.append(token)
    return words2
```

[19]:
```
titles = df1.title.str.lower() #to make it lower case
stemmedop, tokenizedop = ts(titles)
#To remove repitions for better output
words = zip(stemmedop, tokenizedop)
words = list(set(words))
stemmed2, tokenized2 = zip(*words)
merged = pd.DataFrame({'words': tokenized2}, index = stemmed2)  #to put words␣
 ↪under a specific stem
#Using NLTK to get stopwords to remove it from our list
stopwords = nltk.corpus.stopwords.words('english')
stop_words = text.ENGLISH_STOP_WORDS.union(stopwords)
# tf-idf vectorizer
tfidf_vectorizer = TfidfVectorizer(min_df =10**-3 ,analyzer = 'word',␣
 ↪max_features=len(set(stemmed2)), stop_words=stop_words, tokenizer=tokenstem,␣
 ↪ngram_range=(1,3))
tfidf1 = tfidf_vectorizer.fit_transform(titles)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/text.py:385:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'s", 'abov', 'afterward',
'alon', 'alreadi', 'alway', 'ani', 'anoth', 'anyon', 'anyth', 'anywher',
'becam', 'becaus', 'becom', 'befor', 'besid', 'cri', 'describ', 'doe', 'dure',
'els', 'elsewher', 'empti', 'everi', 'everyon', 'everyth', 'everywher', 'fifti',
'forti', 'henc', 'hereaft', 'herebi', 'howev', 'hundr', 'inde', 'mani',
'meanwhil', 'moreov', "n't", 'need', 'nobodi', 'noon', 'noth', 'nowher', 'onc',
'onli', 'otherwis', 'ourselv', 'perhap', 'pleas', 'sever', 'sha', 'sinc',
'sincer', 'sixti', 'someon', 'someth', 'sometim', 'somewher', 'themselv',
'thenc', 'thereaft', 'therebi', 'therefor', 'togeth', 'twelv', 'twenti', 'veri',
'whatev', 'whenc', 'whenev', 'wherea', 'whereaft', 'wherebi', 'wherev', 'whi',
'wo', 'yourselv'] not in stop_words.
  'stop_words.' % sorted(inconsistent))
```

[0]:
```
modified = np.percentile(df1['up_votes'], 85)
op = [1 if i > modified else 0 for i in df['up_votes']]
op = np.array(op)
```

```
x_train, x_test, y_train, y_test = train_test_split(tfidf1, op, test_size = 0.
↪2, shuffle = True, random_state = 0)
```

Here, we first try the Linear SVM classifier and check output.

[21]:
```
clf = LinearSVC()
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print('Classification accuracy on Test Set is %g' %(clf.score(x_test,␣
↪y_test)*100))
print('\n Some statistics of the Linear SVM model are:\n')
print(classification_report(y_test, y_predict))
```

```
Classification accuracy on Test Set is 85.0758

 Some statistics of the Linear SVM model are:

              precision    recall  f1-score   support

           0       0.85      1.00      0.92     86634
           1       0.56      0.00      0.01     15214

    accuracy                           0.85    101848
   macro avg       0.71      0.50      0.46    101848
weighted avg       0.81      0.85      0.78    101848
```

Next we check the classification accuracy of Logistic Regression.

[22]:
```
clf = LogisticRegression(max_iter=10000)
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print('Classification accuracy on Test Set is %g' %(clf.score(x_test,␣
↪y_test)*100))
print('\n Some statistics of the Logistic Regression model are:\n')
print(classification_report(y_test, y_predict))
```

```
Classification accuracy on Test Set is 85.0935

 Some statistics of the Logistic Regression model are:

              precision    recall  f1-score   support

           0       0.85      1.00      0.92     86634
           1       0.53      0.02      0.04     15214

    accuracy                           0.85    101848
   macro avg       0.69      0.51      0.48    101848
```

```
weighted avg       0.80      0.85      0.79    101848
```

The accuracy of Linear SVM and Logistic Regression is same.

Also, using only the text, we have got a very high classification accuracy (~85%). This shows that there is a very strong correlation between the titles and the number of upvotes.

Using more computing power, we could probably see better results with more features.