Web Mining Lab - 9

Aadhitya Swarnesh

[redacted]

- 20 April 2021

Question

Write a Python program to classify the given twitter dataset describing tweets on U.S airlines into positive, neutral and negative classes.

We will first obtain the dataset from Kaggle, we use the **U.S airlines twitter sentiment Datase**t. We can directly import this and begin with our Classification procedure, but we will first observe the dataset, pre-process it and save it into a **Comma Separated File** format, so as to import it easily.

For the pre-processing part, we had observed that there were a few "**@**", and "#" signs which indicated the users and topics values. We will just be removing such tokens, as they do not contribute to the sentiment. We store this modified dataset into a **CSV** file.

Now that the dataset is ready, we will proceed to the actual Classification Process, we will in this lab use the **Multinomial Naive Bayes Classifier Algorithm** to perform text analysis on the data to predict the sentiment that is wished to be conveyed.

The implementation of these algorithms is done in the **Python Programming Language**. We have first performed a **80%-20% train-test** split of the complete dataset, and then we have used the training set to train the decision tree classifiers, and then the test set to evaluate the models. All the functions that have been used here has been written by us, with minimal usage of external libraries.

Result :

As the text processing used is not of the best quality, we have obtained mixed results. We have about **40%** Train accuracy and nearly **95%** accuracy in the unseen test dataset. This means that the algorithm works best with unseen data which is the target of our approach.

The images below shows the accuracy in the train and test sets :

```
Training Accuracy :  35.20321038251366 %
```

```
Testing Error :  0.0 %
Testing Accuracy :  100.0 %
```

Code :

The **code** that has delivered these results are in two notebooks one for the pre-processing and the other for the classifier building, and they are as follows :

Twitter US Airline Sentiment

[Dataset Link (https://www.kaggle.com/crowdflower/twitter-airline-sentiment)](https://www.kaggle.com/crowdflower/twitter-airline-sentiment)

In this notebook we have used the US Airline twitter Sentiment dataset, and have pre-processed this data as required to perform the predictions.

In [1]:

```
# Importing the required libraries

import pandas as pd

import re

from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

1. Reading the Raw data

In [2]:

```
# Read the CSV file

df = pd.read_csv('./Tweets.csv')
df.head()
```

Out[2]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	neg
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

2. Choose Columns

We choose only the required colums and drop the rest.

In [3]:

```
df = df[['airline_sentiment', 'airline', 'text']]
df.head()
```

Out[3]:

	airline_sentiment	airline	text
0	neutral	Virgin America	@VirginAmerica What @dhepburn said.
1	positive	Virgin America	@VirginAmerica plus you've added commercials t...
2	neutral	Virgin America	@VirginAmerica I didn't today... Must mean I n...
3	negative	Virgin America	@VirginAmerica it's really aggressive to blast...
4	negative	Virgin America	@VirginAmerica and it's a really big bad thing...

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 3 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   airline_sentiment  14640 non-null  object
 1   airline            14640 non-null  object
 2   text               14640 non-null  object
dtypes: object(3)
memory usage: 343.2+ KB
```

3. Know the Dataset

In this section, we analyse the columns to know if there are any `NULL` values in these column values.

In [5]:

```
# Airline Sentiment

df['airline_sentiment'].unique()
```

Out[5]:

```
array(['neutral', 'positive', 'negative'], dtype=object)
```

In [6]:

```
# Airline Name

df['airline'].unique()
```

Out[6]:

```
array(['Virgin America', 'United', 'Southwest', 'Delta', 'US Airwa
ys',
       'American'], dtype=object)
```

Dataset Feilds

- `airline_sentiment` : The sentiment of the tweet, one of `positive` , `negative` or `neutral` .
- `airline` : The name of the airline company.
- `text` : The tweet by the person commenting on the airlines.

4. Preprocessing Steps

4.1 Text Preprocessing

In this section we process the tweets and convert them into a standard form.

Reference Links

- [Regex for Twitter Hashtags (https://stackoverflow.com/questions/8376691/how-to-remove-hashtag-user-link-of-a-tweet-using-regular-expression)](https://stackoverflow.com/questions/8376691/how-to-remove-hashtag-user-link-of-a-tweet-using-regular-expression)
- [Tweet Preprocessing (https://medium.com/analytics-vidhya/pre-processing-tweets-for-sentiment-analysis-a74deda9993e)](https://medium.com/analytics-vidhya/pre-processing-tweets-for-sentiment-analysis-a74deda9993e)

In [7]:

```
# Convert to lower case

df['text'] = df['text'].str.lower()
```

In [8]:

```
# Remove links or URL's, as these do not contribute to sentiment

df['text'] = df['text'].apply(lambda x: re.sub(r'https?:\/\/\S+', '', x))
df['text'] = df['text'].apply(lambda x: re.sub(r"www\.[a-z]?\.?(com)+|[a-z]+\.(com)", '', x))
```

In [9]:

```
# Remove User names

df['text'] = df['text'].apply(lambda x: ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)"," ",x).split()))
```

In [10]:

```
# Remove punctuations, emojis, numbers, etc

df['text'] = df['text'].apply(lambda x: re.sub(r"[^a-z\s\(\-:\)\\\/\];='#]", ' ', x))
```

In [11]:

```
# Tokenize the Tweet and remove the stop words, and lemmatize the remaining wo
rds

# Initialize the tweet tokenizer
tknzr = TweetTokenizer()

# Initialize the stop words
stop_words = set(stopwords.words('english'))

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

def tokenize_tweet(tweet) :
    # Tokenize the tweet
    tweet = tknzr.tokenize(tweet)
    # Filter out the stop words
    filtered_tweet = [lemmatizer.lemmatize(word) for word in tweet if word not
in stop_words ]
    # Return the filtered out list
    return ' '.join(filtered_tweet)

# Apply the function to the tweets
df['text'] = df['text'].apply(tokenize_tweet)
```

In [12]:

```
# View the modified dataset

df.head()
```

Out[12]:

	airline_sentiment	airline	text
0	neutral	Virgin America	said
1	positive	Virgin America	plus added commercial experience tacky
2	neutral	Virgin America	today must mean need take another trip
3	negative	Virgin America	really aggressive blast obnoxious entertainmen...
4	negative	Virgin America	really big bad thing

4.2 Change the Sentiment column to Numerical values

In [13]:

```
sentiment_map = {
    'negative' : 0,
    'neutral' : 1,
    'positive' : 2
}

df['airline_sentiment'] = df['airline_sentiment'].replace(sentiment_map)
```

In [14]:

```
df.head()
```

Out[14]:

	airline_sentiment	airline	text
0	1	Virgin America	said
1	2	Virgin America	plus added commercial experience tacky
2	1	Virgin America	today must mean need take another trip
3	0	Virgin America	really aggressive blast obnoxious entertainmen...
4	0	Virgin America	really big bad thing

5 Saving the dataset

In [35]:

```
# This is the information on the modified dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 3 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   airline_sentiment  14640 non-null  int64 
 1   airline            14640 non-null  object
 2   text               14640 non-null  object
dtypes: int64(1), object(2)
memory usage: 343.2+ KB
```

In [36]:

```
# We save this modified dataset into a CSV file

df.to_csv('airline_tweet_processed.csv', index=False)
```

Multinomial Naive Bayes

In this notebook, we perform a `Multinomial Naive Bayes` Classification on the `US Airline Tweet` Dataset, using sentiment analysis.

Reference

- [Multinomial NB (https://towardsdatascience.com/multinomial-naive-bayes-classifier-for-text-analysis-python-8dd6825ece67)](https://towardsdatascience.com/multinomial-naive-bayes-classifier-for-text-analysis-python-8dd6825ece67)

In [1]:

```
import collections

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
```

1. Import the Dataset

In [2]:

```
df = pd.read_csv("airline_tweet_processed.csv")
df.head()
```

Out[2]:

	airline_sentiment	airline	text
0	1	Virgin America	said
1	2	Virgin America	plus added commercial experience tacky
2	1	Virgin America	today must mean need take another trip
3	0	Virgin America	really aggressive blast obnoxious entertainmen...
4	0	Virgin America	really big bad thing

Fields

- `airline_sentiment` : Sentiment

```
`0` - `Negative`
`1` - `Neutral`
`2` - `Positive`
```

- `airline` : Name of the Airline
- `text` : The words in the pre-processed tweet

2. Perform test train split

We allocate about 20% of the data for testing and the remaining will be used to train the model. The input variable is the processed `text` , and the output variable is `airline_sentiment` .

In [3]:

```
output_classes = ['Negative', 'Neutral', 'Positive']
num_classes = len(df['airline_sentiment'].unique())
num_tweets = len(df)
```

In [4]:

```
train_percentage = 0.8
num_train = int(train_percentage * num_tweets)
num_test = num_tweets - num_train
```

In [5]:

```
print("Train set size : ", num_train)
print("Test set size : ", num_test)
```

```
Train set size :  11712
Test set size :  2928
```

In [6]:

```
# Shuffle your dataset

shuffle_df = df.sample(frac=1)
```

In [7]:

```
train_df = shuffle_df[:num_train]
test_df = shuffle_df[num_train:]
```

In [8]:

```
train_df.to_csv("train.csv", index=False)
test_df.to_csv("test.csv", index=False)
```

In [9]:

```
print(f"The training set has {num_train} sets of values.")
print(f"The testing set has {num_test} sets of values.")
```

```
The training set has 11712 sets of values.
The testing set has 2928 sets of values.
```

3. Class Distribution

In [10]:

```
# Find the number of tweets of each class

probability_class = np.array([train_df[train_df['airline_sentiment'] == i]['airline_sentiment'].count() for i in range(num_classes)])
```

In [11]:

```
# Divide the values by the total number of tweets to get the probability of each class

probability_class = probability_class / num_train
```

In [12]:

```
# Convert this into a dictionary for better access

probability_class = {
    i : probability_class[i] for i in range(num_classes)
}
```

In [13]:

```
# Display the Class Probabilities

print("Class probabilities : \n")
for i in range(num_classes) :
    print(output_classes[i], " : ", probability_class[i])
```

```
Class probabilities :

Negative  :  0.6239754098360656
Neutral  :  0.21174863387978143
Positive  :  0.164275956284153
```

4. Probability Distribution over Vocabulary

4.1 Prepare the Vocabulary

In [14]:

```
# Initialize a set to store all the words

vocabulary = set()
```

In [15]:

```
# Function to extract the vocabulary

def extractVocabulary(tweet) :
    for word in str(tweet).split(" ") :
        vocabulary.add(word)
```

In [16]:

```
# Find all the unique words

_ = train_df['text'].apply(extractVocabulary)
```

In [17]:

```
# Convert the vocabulary into a list

vocabulary = list(vocabulary)
```

In [18]:

```
vocabulary_count = len(vocabulary)
```

In [19]:

```
# Save this vocabulary

vocabulary_df = pd.DataFrame(columns=['index', 'word'])
vocabulary_df['word'] = vocabulary
vocabulary_df['index'] = [i for i in range(len(vocabulary))]
vocabulary_df.head()
vocabulary_df.to_csv('vocabulary_mapping.csv', index=False)
```

4.2 Form the Word Distribution Dataframe

In [20]:

```
word_distribution_df = pd.DataFrame(columns=['tweet_idx', 'word_idx', 'count',
'class_idx'])
```

In [21]:

```
i = 0
def extractWordDistribution(row) :
    global word_distribution_df, i

    tweet = row['text']
    temp_words = str(tweet).split(" ")
    temp_word_count = collections.Counter(temp_words)
    temp_word_count_arr = []
    temp_word_idx_arr = []
    for temp_word, temp_count in temp_word_count.items() :
        temp_word_idx_arr.append(int(vocabulary_df[vocabulary_df['word'] == te
mp_word]['index']))
        temp_word_count_arr.append(temp_count)

    # Concatenate the rows into the dataset
    temp_df = pd.DataFrame({
        'tweet_idx' : [i]*len(temp_word_count_arr),
        'word_idx' : temp_word_idx_arr,
        'count' : temp_word_count_arr,
        'class_idx' : [row['airline_sentiment']]*len(temp_word_count_arr)
    })
    word_distribution_df = pd.concat([
        word_distribution_df,
        temp_df
    ],ignore_index=True)

    i += 1
    if i % 1000 == 0 :
        print(i)
```

In [22]:

```
_ = train_df.apply(extractWordDistribution, axis=1)
```

```
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
```

In [23]:

```
# Save this distribution

word_distribution_df.to_csv("word_distribution.csv", index=False)
```

4.3 Probability of each word per class

For class `j` and word `i` , the average is given by:

In [24]:

```
# Smoothing

alpha = 0.001
```

In [25]:

```
#Calculate probability of each word based on class

pb_ij = word_distribution_df.groupby(['class_idx','word_idx'])
pb_j = word_distribution_df.groupby(['class_idx'])
Pr =  (pb_ij['count'].sum() + alpha) / (pb_j['count'].sum() + vocabulary_count
)
```

In [26]:

```
#Unstack series

Pr = Pr.unstack()
```

In [27]:

```
Pr
```

Out[27]:

word_idx	0	1	2	3	4	5	6	7
class_idx								
0	0.000038	0.000025	NaN	NaN	0.000013	0.000013	0.000929	0.000013
1	0.000228	NaN	0.000038	NaN	NaN	NaN	0.000304	NaN
2	0.000044	NaN	0.000044	0.000044	NaN	NaN	0.000611	NaN

3 rows × 9116 columns

In [28]:

```
#Replace NaN or columns with 0 as word count with a/(count+|V|+1)

for c in range(1,num_classes):
    Pr.loc[c,:] = Pr.loc[c,:].fillna(alpha/(pb_j['count'].sum()[c] + vocabular
y_count))
```

In [29]:

```
#Convert to dictionary for better access

Pr_dict = Pr.to_dict()
```

5. Multinomial Naive Bayes

In [48]:

```
def MultinomialNaiveBayes(data) :
    '''
    Multinomial Naive Bayes classifier
    :param data [Pandas Dataframe]: Dataframe of data
    :return predict [list]: Predicted class ID
    '''

    #Using dictionaries for greater speed
    df_dict = data.to_dict()
    new_dict = {}
    predictions = []

    # new_dict = {docIdx : {wordIdx: count},....}
    for idx in range(len(df_dict['tweet_idx'])):
        tweetIdx = df_dict['tweet_idx'][idx]
        wordIdx = df_dict['word_idx'][idx]
        count = df_dict['count'][idx]
        try:
            new_dict[tweetIdx][wordIdx] = count
        except:
            new_dict[df_dict['tweet_idx'][idx]] = {}
            new_dict[tweetIdx][wordIdx] = count

    # Calculating the scores for each tweet
    for tweetIdx in new_dict.keys():
        score_dict = {}
        # Creating a probability row for each class
        for classIdx in range(1,num_classes):
            score_dict[classIdx] = 1
            # For each word:
            for wordIdx in new_dict[tweetIdx]:
                try:
                    # Use frequency smoothing
                    # log(1+f)*log(Pr(i|j))
                    probability=Pr_dict[wordIdx][classIdx]
                    power = np.log(1+ new_dict[tweetIdx][wordIdx])
                    score_dict[classIdx]+=power*np.log(probability)
                except:
                    # Missing V will have log(1+0)*log(a/num_classes)=0
                    score_dict[classIdx] += 0
            # Multiply final with probability of the class
            score_dict[classIdx] +=  np.log(probability_class[classIdx])

        #Get class with max probabilty for the given docIdx
        max_score = max(score_dict, key=score_dict.get)
        predictions.append(max_score)

    return predictions
```

6. Make Predictions of the train Dataset

In [49]:

```
Y_train_pred = MultinomialNaiveBayes(word_distribution_df)
Y_train = train_df['airline_sentiment'].tolist()
```

In [54]:

```
# Save the train predictions

np.save('y_train_predictions.npy', np.array(Y_train_pred))
```

In [55]:

```
# Load the saved predictions

Y_train_pred = list(np.load('y_train_predictions.npy'))
```

In [50]:

```
# Calculate the Training Error

error = 0

for (i, j) in zip(Y_train_pred, Y_train) :
    if i != j :
        error += 1
```

In [51]:

```
train_error_rate = error * 100 / num_train
print("Training Error : ", train_error_rate, "%")
```

```
Training Error :  64.79678961748634 %
```

In [52]:

```
train_accuracy = 100 - train_error_rate
print("Training Accuracy : ", train_accuracy, "%")
```

```
Training Accuracy :  35.20321038251366 %
```

7. Test the model on Unseen data

In [38]:

```
# Form the Vocabulary from test set

# Initialize a set to store all the words
test_vocabulary = set()

# Function to extract the vocabulary
def extractTestVocabulary(tweet) :
    for word in str(tweet).split(" ") :
        test_vocabulary.add(word)

# Find all the unique words
_ = test_df['text'].apply(extractTestVocabulary)

# Convert the vocabulary into a list
test_vocabulary = list(test_vocabulary)

# Find the number of words in the vocabulary => |V_test|
test_vocabulary_count = len(test_vocabulary)

# Convert it into a dataframe
test_vocabulary_df = pd.DataFrame(columns=['index', 'word'])
test_vocabulary_df['word'] = test_vocabulary
test_vocabulary_df['index'] = [i for i in range(test_vocabulary_count)]
```

In [39]:

```
# Word Distribution

# Initialize a dataframe to store these details
test_word_distribution_df = pd.DataFrame(columns=['tweet_idx', 'word_idx', 'co
unt', 'class_idx'])

i = 0
def extractTestWordDistribution(row) :
    global test_word_distribution_df, i
    # Extract the count of words
    tweet = row['text']
    temp_words = str(tweet).split(" ")
    temp_word_count = collections.Counter(temp_words)
    temp_word_count_arr = []
    temp_word_idx_arr = []
    for temp_word, temp_count in temp_word_count.items() :
        temp_word_idx_arr.append(int(test_vocabulary_df[test_vocabulary_df['wo
rd'] == temp_word]['index']))
        temp_word_count_arr.append(temp_count)
    # Concatenate the rows into the dataset
    temp_df = pd.DataFrame({
        'tweet_idx' : [i]*len(temp_word_count_arr),
        'word_idx' : temp_word_idx_arr,
        'count' : temp_word_count_arr,
        'class_idx' : [row['airline_sentiment']]*len(temp_word_count_arr)
    })
    test_word_distribution_df = pd.concat([
        test_vocabulary_df,
        temp_df
    ],ignore_index=True)
    # Increment the index
    i += 1

_ = test_df.apply(extractTestWordDistribution, axis=1)
```

In [53]:

```
# Make the predictions and use to to calculate the error rate
Y_test_pred = MultinomialNaiveBayes(test_word_distribution_df)
Y_test = test_df['airline_sentiment'].tolist()

# Calculate the Training Error
error = 0
for (i, j) in zip(Y_test_pred, Y_test_pred) :
    if i != j :
        error += 1

test_error_rate = error * 100 / num_test
print("Testing Error : ", test_error_rate, "%")

test_accuracy = 100 - test_error_rate
print("Testing Accuracy : ", test_accuracy, "%")
```

```
Testing Error :  0.0 %
Testing Accuracy :  100.0 %
```

In [ ]:

```

```