Building a Twitter Sentiment Analysis using Python

Introduction:-

- The ability to categorise opinions expressed in the text of tweets—particularly the ability to decide whether the writer's attitude is positive, negative, or neutral—is extremely valuable is the process of sentiment analysis.

- We will use the process known as *sentiment analysis* to categorize the opinions of people on Twitter.

- There are different ordinal scales used to categorize tweets. Today we will be dealing with a three-point ordinal scale includes Negative, Neutral, and Positive. The dataset for this analysis is taken from the kaggle. website https://www.kaggle.com/kazanova/sentiment140

Sentiment Analysis is a type of binary classifies problem. It can be solved using two techniques.

- Rule based Techniques:
  - In rule based techniques rules are drawn by experts are used to assign a label to a problem instance.
  - It uses sentiwordnet and VADER algorithms

- Machine learning Technique
  - In machine learning techniques label is assigned based on patterns displayed in aggregate data
  - This techniques uses Naive Bayes, logistic regression and NLTK libraries.

Explanation of the code

A) Content of the Dataset

1. target: the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)

2. ids: The id of the tweet (2087)

3. date: the date of the tweet (Sat May 16 23:58:44 UTC 2009)

4. flag: The query (lyx). If there is no query, then this value is NO_QUERY.

5. user: the user that tweeted (robotickilldozr)

6. text: the text of the tweet (Lyx is cool)

Sentiment analysis involves natural language processing because it deals with human-written text. Hence the libraries used are:

```
#data manupilation libraries
import pandas as pd
import numpy as np
import re
import string

#text processing libraries and feature extraction using sklea
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer

#machine learning libraries(sklearn)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

Step 2: After we run all the libraries need we need to load the data set.

Step 3: Remove the unwanted columns like ‘t_id’, ‘created_at’, ‘query’, ‘user’ as they are not necessary the sentiment analysis.

Step 4: After finishing the loading and removal of unwanted data columns we need to download and load the english stop words using the following code

```
#download the english stop words
nltk.download("stopwords")
spw = set(stopwords.words('english'))
```

Step 5: When the stopwords are downloaded we need to download the ‘punkt’ library for tokenization.

```
#download the punkt to tokanize the words
import nltk
nltk.download('punkt')
```

Step 6: Preprocess the data

a) Letter casing: Converting all letters to either uppercase or lowercase. In this code we convert all the letters to lowercase.
b) Tokenizing: Turning the tweets into tokens.
c) Noise removal: Eliminating unwanted characters, such as HTML tags, punctuation marks, special characters, white spaces etc.
d) Stopword removal: Removes words which does not make any sense and does not contribute to the machine learning model.
e) Normalization: Normalization generally refers to a series of related tasks meant to put all text on the same level. Converting text to lowercase, removing special characters, and removing stopwords will remove basic inconsistencies. Normalization improves text matching.

Code for preprocessing of data.

```
def preprocess_tweet_text(tweet):

    # covert all the text in lowercase
    tweet = tweet.lower()

    # Remove urls
    tweet = re.sub(r"http\S+|www\S+|https\S+", '', tweet, flags=re.MULTILINE)

    # Remove user @ references and '#' from tweet
    tweet = re.sub(r'\@\w+|\#','', tweet)

    # Remove punctuations
    tweet = tweet.translate(str.maketrans('', '', string.punctuation))

    # Remove stopwords
    tweet_tokens = word_tokenize(tweet)
    filtered_words = [word for word in tweet_tokens if word not in spw]

    return " ".join(filtered_words)
```

Step 7: After the data is preprocessed we have to now vectorize the data. Vectorizing is the process to convert tokens to numbers. It is an important step because the machine learning algorithm works with numbers and not text.

```
def get_feature_vector(train_fit):
    vector = TfidfVectorizer(sublinear_tf=True)
    vector.fit(train_fit)
    return vector
```

Step 8: Finally after the dataset is processed, this is the final result

```
#Preprocess data
dataset.text = dataset['text'].apply(preprocess_tweet_text)
dataset.text.head()
```

```
0    upset cant update facebook texting might cry r...
1    dived many times ball managed save 50 rest go ...
2                    whole body feels itchy like fire
3                            behaving im mad cant see
4                                          whole crew
Name: text, dtype: object
```

Step 8:
Now we need to split the dataset to train the machine learning models and after that to test them.
Before Splitting the dataset we need to vectorize it by calling the vector function done previously.
After the data is vectorized we need to transform it anf then split the data set.
The code for this process is as follows.

```
# Split dataset into Train, Test
# Same tf vector will be used for Testing sentiments on unseen trending data
tf_vector = get_feature_vector(np.array(dataset.iloc[:, 1]).ravel())
X = tf_vector.transform(np.array(dataset.iloc[:, 1]).ravel())
y = np.array(dataset.iloc[:, 0]).ravel()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=30)
```

Here we have split the data where the test dataset has 20% and the random state be assigned as 30.

Step 9: After the dataset is split into test data and train data we will need to train our machine learning models using the train dataset.
Here i have shown two different types of machine learning algorithms and their accuracy score.

A) Logistic Regression:

```
# Training Logistics Regression model
LR_model = LogisticRegression(solver='lbfgs')
LR_model.fit(X_train, y_train)
y_predict_lr = LR_model.predict(X_test)
print(accuracy_score(y_test, y_predict_lr))
```

```
0.784925
```

B) Naive Bayes:

```
# Training Naive Bayes model
NB_model = MultinomialNB()
NB_model.fit(X_train, y_train)
y_predict_nb = NB_model.predict(X_test)
print(accuracy_score(y_test, y_predict_nb))
```

```
0.7664125
```

Conclusion:-

From the above code snippets it is clear that we need to use the logistic Regression model as it's accuracy score of 78.49% is higher than the accuracy score of the Naive bayes model which has an accuracy score of 76.64%

Report By:
Name: Riya Bhavsar
E-mail address: rymbhavsar@gmail.com