ASSIGNMENT 4
Implementation of Naive Bayes Algorithm

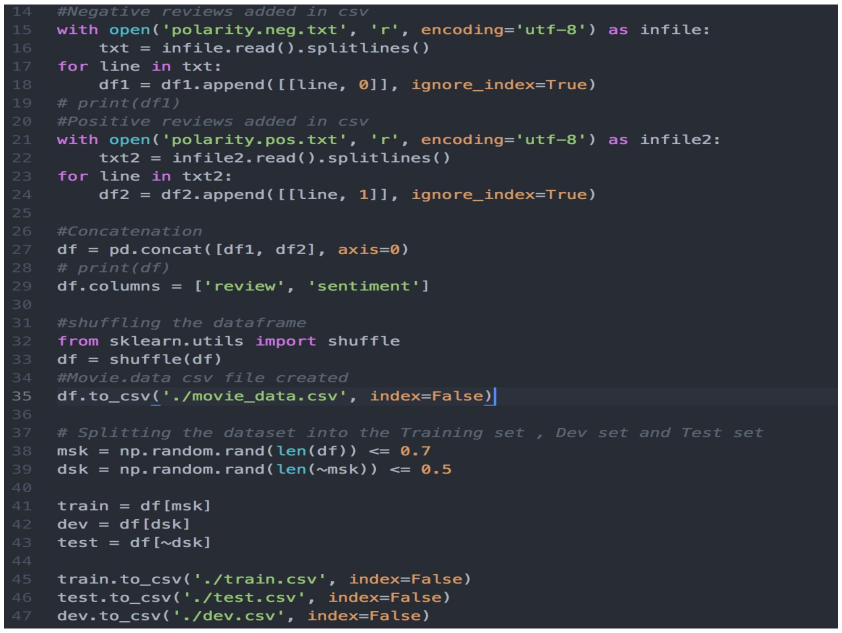
The data given to us are the two text files of positive & negative reviews respectively of Movies Review Data. Our goal was to implement Naive Bayes Classifier that performs the movie review classification automatically. Naive Bayes classifier is based on Bayes' theorem, which is:

P(H | x) = P(H) P(x | H) / P(x)

- P(H|x) is the posterior probability of hypothesis (H or target) given predictor (x or attribute).
- P(H) is the prior probability of hypothesis
- P(x|H) is the likelihood which is the probability of predictor given hypothesis.
- P(x) is the prior probability of predictor.

STEP 1: Splitting Data

- The code file is : **DataDivision.py**
- First I took the two text files and combine them into a file named : movie_data.csv. The combined file consists of two columns - review, sentiment. Sentiment column consists of labels as 0 (negative) and 1(positive) according to the reviews. I then shuffled the data and stored it.
- Second I divided the movie_data.csv into train.csv (70%), dev.csv (15%)and test.csv(15%).

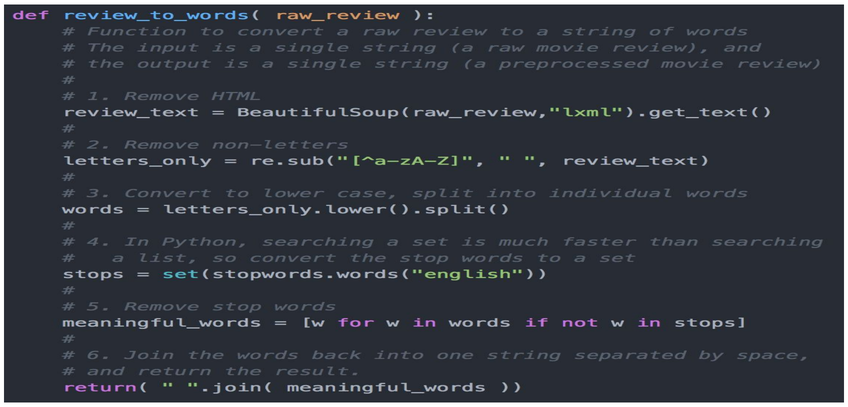


```python
#Negative reviews added in csv
with open('polarity.neg.txt', 'r', encoding='utf-8') as infile:
    txt = infile.read().splitlines()
for line in txt:
    df1 = df1.append([[line, 0]], ignore_index=True)
# print(df1)
#Positive reviews added in csv
with open('polarity.pos.txt', 'r', encoding='utf-8') as infile2:
    txt2 = infile2.read().splitlines()
for line in txt2:
    df2 = df2.append([[line, 1]], ignore_index=True)

#Concatenation
df = pd.concat([df1, df2], axis=0)
# print(df)
df.columns = ['review', 'sentiment']

#shuffling the dataframe
from sklearn.utils import shuffle
df = shuffle(df)
#Movie.data csv file created
df.to_csv('./movie_data.csv', index=False)

# Splitting the dataset into the Training set , Dev set and Test set
msk = np.random.rand(len(df)) <= 0.7
dsk = np.random.rand(len(~msk)) <= 0.5

train = df[msk]
dev = df[dsk]
test = df[~dsk]

train.to_csv('./train.csv', index=False)
test.to_csv('./test.csv', index=False)
dev.to_csv('./dev.csv', index=False)
```

STEP 2: Data Cleaning and Preprocessing

- To clean the data i implemented a function : *`review_to_words`* which removes any html and non letters, convert to lowercase and split into individual words and removed stopwords

STEP 3:

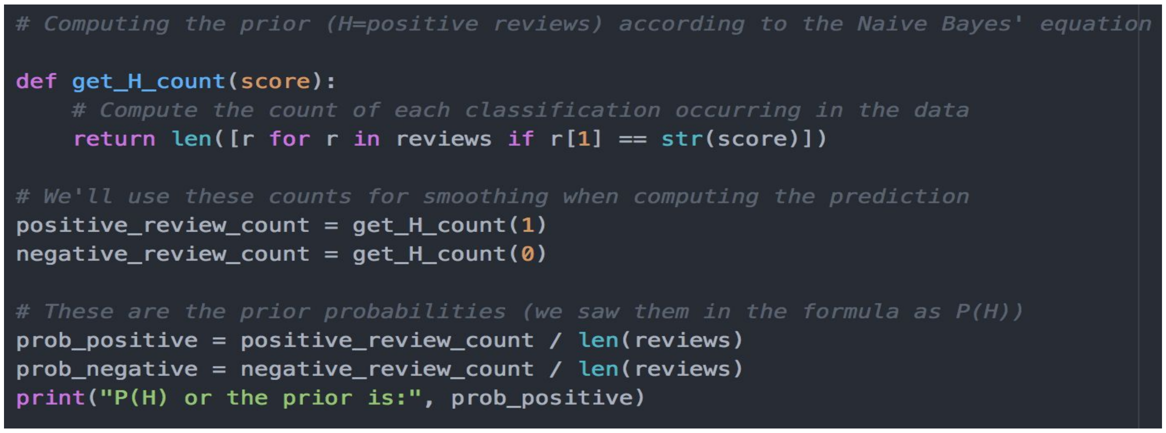


```
def review_to_words( raw_review ):
    # Function to convert a raw review to a string of words
    # The input is a single string (a raw movie review), and
    # the output is a single string (a preprocessed movie review)
    #
    # 1. Remove HTML
    review_text = BeautifulSoup(raw_review,"lxml").get_text()
    #
    # 2. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)
    #
    # 3. Convert to lower case, split into individual words
    words = letters_only.lower().split()
    #
    # 4. In Python, searching a set is much faster than searching
    #   a list, so convert the stop words to a set
    stops = set(stopwords.words("english"))
    #
    # 5. Remove stop words
    meaningful_words = [w for w in words if not w in stops]
    #
    # 6. Join the words back into one string separated by space,
    # and return the result.
    return( " ".join( meaningful_words ))
```

Algorithm

Implementation

- The code file is: **NaiveBayes.py**
- Now i implemented a function *`get_H_count`* which calculates the *`Prior Probability (P(H))`* for negative & positive review count. The result of which was *0.49429300389418557*



```
# Computing the prior (H=positive reviews) according to the Naive Bayes' equation

def get_H_count(score):
    # Compute the count of each classification occurring in the data
    return len([r for r in reviews if r[1] == str(score)])

# We'll use these counts for smoothing when computing the prediction
positive_review_count = get_H_count(1)
negative_review_count = get_H_count(0)

# These are the prior probabilities (we saw them in the formula as P(H))
prob_positive = positive_review_count / len(reviews)
prob_negative = negative_review_count / len(reviews)
print("P(H) or the prior is:", prob_positive)
```

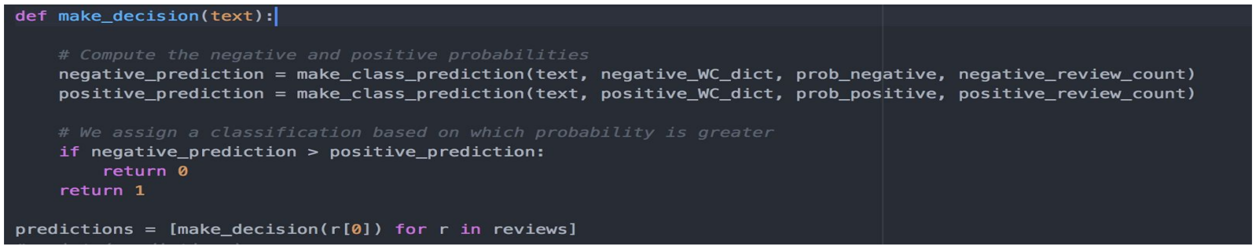
- For Obtaining *`P(xi | H)`* `: the likelihood` i implemented two functions:

  - *`count_text`* *:* This functions counts the no of words and generate the Word Count dictionary for negative and positive reviews respectively

  - *`make_class_prediction`* : Implements the formula by calculating For every word in the text, the number of times that word occurred in the text and then Multiply it with the probability of that word in Hypothesis. Next smooth it by adding 1 by the following code:

```
for word in text_WC_dict:
        prediction *=  text_WC_dict.get(word,0) *
((H_WC_dict.get(word, 0) + 1) / (sum(H_WC_dict.values()) + H_count))
```

- For the Final decision I implemented the function `make_decision` which computes the negative and positive probabilities and assign the classification according to which probability is greater

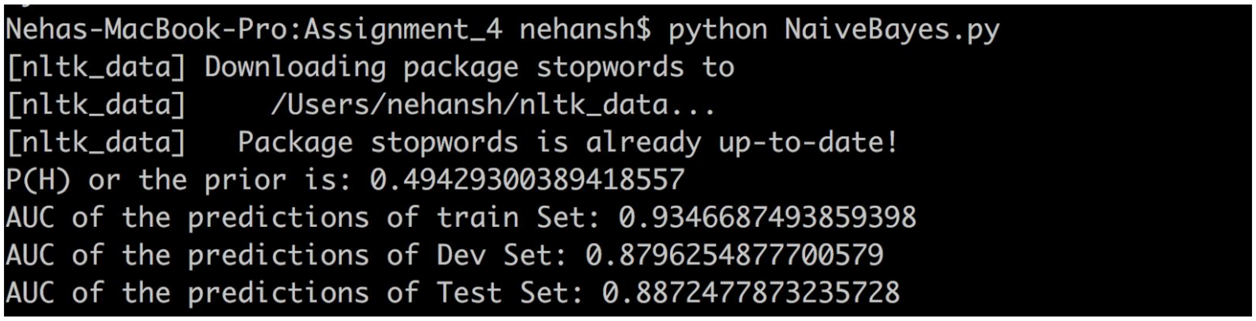


```
def make_decision(text):

    # Compute the negative and positive probabilities
    negative_prediction = make_class_prediction(text, negative_WC_dict, prob_negative, negative_review_count)
    positive_prediction = make_class_prediction(text, positive_WC_dict, prob_positive, positive_review_count)

    # We assign a classification based on which probability is greater
    if negative_prediction > positive_prediction:
        return 0
    return 1

predictions = [make_decision(r[0]) for r in reviews]
```

STEP 4: Error Analysis of the Algorithm

- For error analysis I generate the ROC Curve which gives the fpr, tpr & thresholds. Using which I measure the area under the curve which defines the accuracy for my algorithm. The closer to 1 it is the better the prediction.
- I applied the algorithm and did the analysis for the training set, dev set and test set. And got the below as my AUC for each sets.

```
AUC of the predictions of train Set: 0.9346687493859398
AUC of the predictions of Dev Set: 0.8796254877700579
AUC of the predictions of Test Set: 0.8872477873235728
```



```
Nehas-MacBook-Pro:Assignment_4 nehansh$ python NaiveBayes.py
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/nehansh/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
P(H) or the prior is: 0.494293003894185557
AUC of the predictions of train Set: 0.9346687493859398
AUC of the predictions of Dev Set: 0.8796254877700579
AUC of the predictions of Test Set: 0.8872477873235728
```

Based on the above result I conclude the algorithm implemented classified the reviews with 88% accuracy for the Test set.