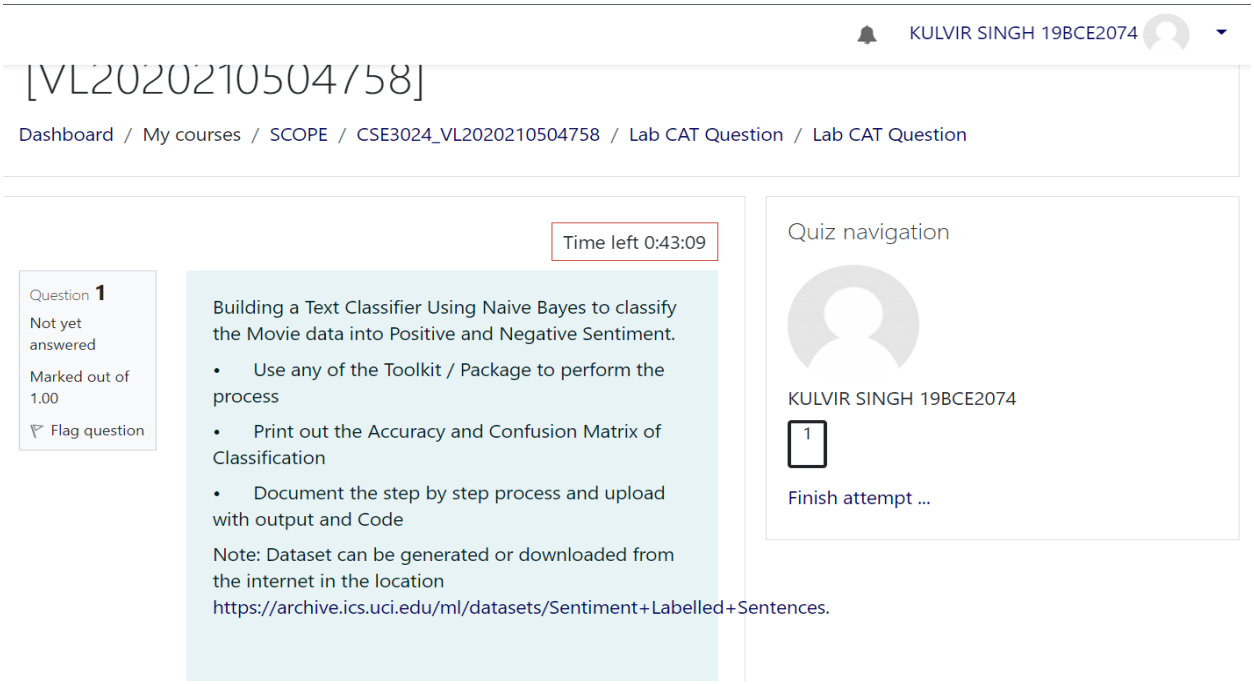
Web Mining Lab CAT

Name: Kulvir Singh

Register Number: 19BCE2074

Question:



Dataset Link :

https://archive.ics.uci.edu/ml/machine-learning-databases/00331/sentiment%20labelled%20sentences.zip
Imbd_labelled.txt is used

Procedure

1) Download data set from the given link
2) Import the necessary packages into the code using the import statement
3) Using nltk download the stop words
4) Train the txt file as a data frame with sentiment and review as headers using pandas
5) Train each of the headers as two separate entities
6) Create a tokenizer class with appropriate functions used to assist the tokenizing of the reviews of the document
7) Use Naive Bayes classifier to classify the movie data into positive and negative sentiments
8) Display the accuracy of the classification
9) Display the confusion matrix of the classification
10) Display the classification report

Code

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import string
import re
import matplotlib.pyplot as plt
import math
from matplotlib import rc
from google.colab import drive
from sklearn.model_selection import train_test_split
from collections import Counter, defaultdict
from bs4 import BeautifulSoup
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud

%matplotlib inline

sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
nltk.download('stopwords')
train = pd.read_csv("imdb_labelled.txt", delimiter="\t", names=['review','
sentiment'])
train.head()
X = train['review'].values
y = train['sentiment'].values
class Tokenizer:

  def clean(self, text):
      no_html = BeautifulSoup(text).get_text()
      clean = re.sub("[^a-z\s]+", " ", no_html, flags=re.IGNORECASE)
      return re.sub("(\s+)", " ", clean)
```

```
  def tokenize(self, text):
      clean = self.clean(text).lower()
      stopwords_en = stopwords.words("english")
      return [w for w in re.split("\W+", clean) if not w in stopwords_en]
class MultinomialNaiveBayes:

    def __init__(self, classes, tokenizer):
      self.tokenizer = tokenizer
      self.classes = classes

    def group_by_class(self, X, y):
      data = dict()
      for c in self.classes:
        data[c] = X[np.where(y == c)]
      return data

    def fit(self, X, y):
        self.n_class_items = {}
        self.log_class_priors = {}
        self.word_counts = {}
        self.vocab = set()

        n = len(X)

        grouped_data = self.group_by_class(X, y)

        for c, data in grouped_data.items():
          self.n_class_items[c] = len(data)
          self.log_class_priors[c] = math.log(self.n_class_items[c] / n)
          self.word_counts[c] = defaultdict(lambda: 0)

          for text in data:
            counts = Counter(self.tokenizer.tokenize(text))
            for word, count in counts.items():
                if word not in self.vocab:
                    self.vocab.add(word)

                self.word_counts[c][word] += count

        return self

    def laplace_smoothing(self, word, text_class):
      num = self.word_counts[text_class][word] + 1
      denom = self.n_class_items[text_class] + len(self.vocab)
```

```
        return math.log(num / denom)

    def predict(self, X):
        result = []
        for text in X:

            class_scores = {c: self.log_class_priors[c] for c in self.classe
s}

            words = set(self.tokenizer.tokenize(text))
            for word in words:
                if word not in self.vocab: continue

                for c in self.classes:

                    log_w_given_c = self.laplace_smoothing(word, c)
                    class_scores[c] += log_w_given_c

            result.append(max(class_scores, key=class_scores.get))

        return result
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=RANDOM_SEED)
MNB = MultinomialNaiveBayes(
    classes=np.unique(y),
    tokenizer=Tokenizer()
).fit(X_train, y_train)
y_hat = MNB.predict(X_test)
print("Accuracy = ",accuracy_score(y_test, y_hat))
cnf_matrix = confusion_matrix(y_test, y_hat)
print("Confusion Matrix =\n",cnf_matrix)
print("Classification Report = \n",classification_report(y_test, y_hat))
```

Output

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
KULVIR SINGH
19BCE2074
_____OUTPUT_____
Accuracy =  0.8066666666666666
Confusion Matrix =
 [[59 17]
 [12 62]]
Classification Report =
              precision    recall  f1-score   support

           0       0.83      0.78      0.80        76
           1       0.78      0.84      0.81        74

    accuracy                           0.81       150
   macro avg       0.81      0.81      0.81       150
weighted avg       0.81      0.81      0.81       150
```