```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mo
```

```
# imports
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer #can try using tf-idf to see if i
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.feature_extraction import text
from sklearn import preprocessing #encode labels as integers from 0-7
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import spacy
##
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
from nltk.stem import PorterStemmer
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from sklearn.feature_extraction import text
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Data Preprocessing

```
#change filepath
pd_train_data = pd.read_csv('/content/gdrive/My Drive/ECSE 551 Project 2/train.csv')
pd_test_data = pd.read_csv('/content/gdrive/My Drive/ECSE 551 Project 2/test.csv')
#pd_train_data.head()
```

Double-click (or enter) to edit

```
#uncomment this block to run the logistic regression model
'''
'''Create Vocabulary'''
pf = pd_train_data.drop_duplicates(subset='body')
#print(pd_train_data['body'][9745]==pd_train_data['body'][9785])
#print(pf.shape)
#train_data = pd_train_data['body'].tolist()
#train_labels = pd_train_data['subreddit'].tolist()
train_data = pf['body'].tolist()
train_labels = pf['subreddit'].tolist()
sklearn_stopwords = text.ENGLISH_STOP_WORDS

vectorizer0 = TfidfVectorizer(min_df=.0004,stop_words=sklearn_stopwords,analyzer='word')
X0 = vectorizer0.fit_transform(train_data)
features=vectorizer0.get_feature_names()
train_data20 = X0.toarray()
train_data0 = SelectKBest(chi2,k=5000).fit(train_data20,train_labels)

new_features = []
for index in train_data0.get_support(indices=True):
        new_features.append(features[index])
print(len(new_features))
rpg_data=[]
rpg_label=[]
anime_data=[]
datascience_data=[]
datascience_label=[]
hardware_data=[]
hardware_label=[]
cars_data=[]
gamernews_data=[]
gamernews_label=[]
gamedev_data=[]
gamedev_label=[]
computers_data=[]
computers_label=[]
'''separate by classes'''
for lineNum in range(len(train_data)):
    if (train_labels[lineNum]=='rpg'):
        rpg_data.append(train_data[lineNum])
        rpg_label.append(train_labels[lineNum])
    #elif(train_labels[lineNum]=='anime'):
    #    anime_data.append(train_data[lineNum])
    elif(train_labels[lineNum]=='datascience'):
        datascience_data.append(train_data[lineNum])
        datascience_label.append(train_labels[lineNum])
    elif(train_labels[lineNum]=='hardware'):
```

```
    elif(train_labels[lineNum]=='hardware'):
        hardware_data.append(train_data[lineNum])
        hardware_label.append(train_labels[lineNum])
    #elif(train_labels[lineNum]=='cars'):
    #    cars_data.append(train_data[lineNum])
    elif(train_labels[lineNum]=='gamernews'):
        gamernews_data.append(train_data[lineNum])
        gamernews_label.append(train_labels[lineNum])
    elif(train_labels[lineNum]=='gamedev'):
        gamedev_data.append(train_data[lineNum])
        gamedev_label.append(train_labels[lineNum])
    elif(train_labels[lineNum]=='computers'):
        computers_data.append(train_data[lineNum])
        computers_label.append(train_labels[lineNum])
GDvsDS=list.copy(gamedev_data)
GDvsDSlabel=list.copy(gamedev_label)
for lineNum in range(len(datascience_data)):
    GDvsDS.append(datascience_data[lineNum])
    GDvsDSlabel.append(datascience_label[lineNum])

GNvsHW=list.copy(gamernews_data)
GNvsHWlabel=list.copy(gamernews_label)
for lineNum in range(len(hardware_data)):
    GNvsHW.append(hardware_data[lineNum])
    GNvsHWlabel.append(hardware_label[lineNum])

GNvsRPG=list.copy(gamernews_data)
GNvsRPGlabel=list.copy(gamernews_label)
for lineNum in range(len(gamedev_data)):
    GNvsHW.append(rpg_data[lineNum])
    GNvsHWlabel.append(rpg_label[lineNum])

PCvsHW=list.copy(computers_data)
PCvsHWlabel=list.copy(computers_label)
for lineNum in range(len(hardware_data)):
    GNvsHW.append(hardware_data[lineNum])
    GNvsHWlabel.append(hardware_label[lineNum])

'''vectorize each class separately'''
def getVocabulary(dataset,datalabel):
    vect = TfidfVectorizer(max_df=.9, stop_words=sklearn_stopwords, analyzer='word')
    Data=vect.fit_transform(dataset)
    features=vect.get_feature_names()
    bestData = SelectKBest(chi2,k=1000).fit(Data.toarray(),datalabel)
    indexes=bestData.get_support(indices=True)
    nfeatures = []
    for index in indexes:
        nfeatures.append(features[index])
    return nfeatures

vocabulary=getVocabulary(GNvsHW,GNvsHWlabel)
for lineNum in range(len(vocabulary)):
    features.append(vocabulary[lineNum])
```

```
  new_features.append(vocabulary[lineNum])
vocabulary=getVocabulary(GDvsDS,GDvsDSlabel)
for lineNum in range(len(vocabulary)):
    new_features.append(vocabulary[lineNum])
vocabulary=getVocabulary(GNvsHW,GNvsHWlabel)
for lineNum in range(len(vocabulary)):
    new_features.append(vocabulary[lineNum])
vocabulary=getVocabulary(PCvsHW,PCvsHWlabel)
for lineNum in range(len(vocabulary)):
    new_features.append(vocabulary[lineNum])
#for word in ['ps2','ps3','ps4','ps5','ps1','480p','pass']:
#  new_features.append(word)

words=set(new_features)'''
```

```
     5000
```

```
#comment this block out to run logistic regression model

#erase before final version
pf = pd_train_data.drop_duplicates(subset='body')
#print(pd_train_data['body'][9745]==pd_train_data['body'][9785])
#print(pf.shape)
#train_data = pd_train_data['body'].tolist()
#train_labels = pd_train_data['subreddit'].tolist()
train_data = pf['body'].tolist()
train_labels = pf['subreddit'].tolist()
sklearn_stopwords = text.ENGLISH_STOP_WORDS
vectorizer = CountVectorizer(strip_accents = ascii, max_features = 10000,binary=True,
                             stop_words='english',analyzer='word',ngram_range=(1, 1)) #shoulc
X = vectorizer.fit_transform(train_data)
train_data2 = X.toarray()
train_data = SelectKBest(f_classif,k=5000).fit_transform(train_data2,train_labels)
#to do the inverse of the result do not forget to use le object by using inverse_transform(y)
le = preprocessing.LabelEncoder()
train_labels = le.fit_transform(train_labels)


'''#uncomment this block to run logistic regression model
#vectorize text data
vectorizer = TfidfVectorizer(vocabulary=words,analyzer='word',binary=True)
X = vectorizer.fit_transform(train_data)
train_data = X.toarray()

#to do the inverse of the result do not forget to use le object by using inverse_transform(y)
le = preprocessing.LabelEncoder()
train_labels = le.fit_transform(train_labels)'''


#create hold out data for testing
X_train, X_test, y_train, y_test = train_test_split(train_data, train_labels, test_size=0.20,
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

```
(9105, 5917) (9105,) (2277, 5917) (2277,)
```

Naive Bayes Algorithm

```
#create Naivebayes class
class Naivebayes:
    def fit(self,X,y):
            #group training data into distinct classes and store as a list\n",
            grouped = []
            for i in np.unique(y):
              temp = []
              for j,k in zip(X,y):
                if k==i:
                  temp.append(j)
              grouped.append(temp)

            #calculate prior probabilities of each class and store in a list
            #use logs to avoid floating point underflow because we will be multiplying-
            #a lot of small numbers together and we dont want them getting approximated to ze

            self.prior_prob = np.array([np.log(len(i)/len(y)) for i in grouped])
            print()

            #calculate conditional probabilities for each class
            #variables needed:
            #(1) how many documents does word occur for a given class,(2) total no of documer
            #(3) add laplace smoothing constant

            #(1)
            word_occurs = np.array([np.array(i).sum(axis=0) for i in grouped])+1

            #(2)
            no_of_doc = np.array([[np.array(i).shape[0] for i in grouped]])+2

            #compute conditional probabilities by dividing. Use vectors instead of writing a
            self.cond_prob = word_occurs/no_of_doc.T

    def predict(self, X):
      #store result of log probabilities
      #select the max
      #assume X is binary encoded
      self.labs = []
      for i in X:
        result = []
        temp_yes = np.log(self.cond_prob)*i #zeros out non occuring words and calculates log
        temp_no = np.array(1-i)*np.log(1-self.cond_prob) #zeros out occuring words and calcul
        temp = temp_yes + temp_no
        for i in temp:
```

```
        for j in temp:
          result.append(np.array([j]).sum(axis=1))
        result = np.array(result).T+self.prior_prob
        self.labs.append(np.argmax(result))

    def accu_eval(self,y):
      self.acc = 0
      for i in range(len(y)):
        if self.labs[i]==y[i]:
          self.acc+=1
      print("The accuracy over the test data is",self.acc/len(y)*100,"%")
      return self.acc/len(y)

    def write_to_csv(self):
      #return csv file with labels
      self.labs = le.inverse_transform(self.labs)
      dt = {'subreddit':self.labs}
      df = pd.DataFrame(dt)
      df.to_csv('out.csv',columns=['subreddit'],index_label=['id'])


d = Naivebayes()
d.fit(X_train,y_train)
d.predict(X_test)
d.accu_eval(y_test)
```

```
The accuracy over the test data is 87.22002635046113 %
0.8722002635046113
```

```
test_data = pd.read_csv('/content/gdrive/My Drive/ECSE 551 Project 2/test.csv')
test_data = test_data['body'].tolist()
test_data = vectorizer.transform(test_data)
#x_new = SelectKBest(chi2,k=5000).fit(train_data2,train_labels)
#test_data = x_new.transform(test_data)
test_data = test_data.toarray()
print(test_data.shape)

td = Naivebayes()
td.fit(train_data, train_labels)
td.predict(test_data)
```

```
(2898, 5917)
```

```
#change filepath
td.write_to_csv()
s = pd.read_csv('./out.csv')
s.head()
```

	id	subreddit
0	0	datascience
1	1	anime
2	2	rpg
3	3	computers

Logistic Regression Model

```
def write_to_csv_sklearn(labs, model):
  #return csv file with labels
  labs = le.inverse_transform(labs)
  dt = {'subreddit': labs}
  df = pd.DataFrame(dt)
  filename = model + '_out.csv'
  df.to_csv(filename, columns=['subreddit'],index_label=['id'])
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

model = LogisticRegression(max_iter = 500)
logreg = model.fit(X_train, y_train)

predictions = logreg.predict(X_test)
acc = accuracy_score(y_test, predictions)
print('Logistic Regression Accuracy:', acc)

test_predictions = logreg.predict(test_data)

write_to_csv_sklearn(test_predictions, 'logreg')
```

```
     Logistic Regression Accuracy: 0.896354852876592
```

k-fold cross validation

```
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
```

```
#this loop will run n_splits number of times
def K_Fold_Eval(n_splits, model, X_train, y_train):
  kf = KFold(n_splits, shuffle = True)
  accs = []

  for train_ind, val_ind in kf.split(X_train):
    kf_x_train, kf_x_val = X_train[train_ind], X_train[val_ind]
    kf_y_train, kf_y_val = y_train[train_ind], y_train[val_ind]

    fitted = model.fit(kf_x_train, kf_y_train)
    predictions = fitted.predict(kf_x_val)
    acc = accuracy_score(kf_y_val, predictions)
    print(acc)
    accs.append(acc)

  av_acc = sum(accs)/len(accs)

  return av_acc

def K_Fold_Eval_NB(n_splits, model, X_train, y_train):
  kf = KFold(n_splits = n_splits, shuffle = True)
  accs = []

  for train_ind, val_ind in kf.split(X_train):
    kf_x_train, kf_x_val = X_train[train_ind], X_train[val_ind]
    kf_y_train, kf_y_val = y_train[train_ind], y_train[val_ind]

    #fitted = model.fit(kf_x_train, kf_y_train)
    model.fit(kf_x_train, kf_y_train)
    model.predict(kf_x_val)
    acc = model.accu_eval(kf_y_val)
    accs.append(acc)

  av_acc = sum(accs)/len(accs)

  return av_acc


#av_acc_log = K_Fold_Eval(10, LogisticRegression(max_iter = 500), X_train, y_train)
#print('K-fold Accuracy for Logistic Regression', av_acc_log)
```

```
#Logistic Regression K fold
av_acc_log = K_Fold_Eval(10, LogisticRegression(class_weight= None, max_iter = 500), train_da
print('K-fold Accuracy for Logistic Regression', av_acc_log)
```

```
    0.8823529411764706
    0.9025460930640913
    0.9024604569420035
    0.8804920913884007
    0.8927943760984183
    0.9130052724077329
```

```
0.8884007029876977
0.8927943760984183
0.8796133567662566
0.8945518453427065
K-fold Accuracy for Logistic Regression 0.8929011512272196
```

```
k_fold_NB = K_Fold_Eval_NB(10, Naivebayes(), train_data, train_labels)
print('K-fold Accuracy for Naivebayes', k_fold_NB)
```

```

The accuracy over the test data is 88.14749780509219 %

The accuracy over the test data is 87.00614574187884 %

The accuracy over the test data is 87.69771528998243 %

The accuracy over the test data is 86.55536028119508 %

The accuracy over the test data is 89.103690685413 %

The accuracy over the test data is 86.55536028119508 %

The accuracy over the test data is 87.17047451669596 %

The accuracy over the test data is 87.60984182776801 %

The accuracy over the test data is 88.22495606326889 %

The accuracy over the test data is 88.92794376098419 %
K-fold Accuracy for Naivebayes 0.8769989862534736
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.