StudentName : Vallabh Vaibhav Patil
Student Id : 22116880

Part 2 :

To solve the given classification problem of predicting category of news article.

GitHub Repository Link : https://github.com/vallabhpatil777/News-category-prediction-.git

Import all the required python libraries as shown in code snippet below.

```
# Importing required Python libraries.

import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import os
import seaborn as sns


# Downloading the set of stopwords, which we will be using for preprocessing of our dataset
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))


# Using Porterstemmer for stemming, i.e converting the word to its original form.
ps = PorterStemmer()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\MSI\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

a) To download the corpus of stopwords required for pre-processing of data we are using nltk.download("stopwords").
b) We are creating instance of Porterstemmer class to perform the stemming on each word from the text to obtain its original form i.e to remove suffixes, for example, raining-rain. In this case we are using Porterstemmer as it is more faster for pre-processing of data.

Next step is to load the dataset. We will be performing below steps to load data.

a) Provide the path for dataset folder.
b) Create two lists for storing article data and categories.
c) Now, iterate each category folder and store the name of this category folder in category name list. Iterate over each ".txt" file and store its content in text list.

Data is stored in two lists texts and categories using these we will convert the data into .csv file. For this we are first adding these two lists into a pandas Dataframe and then using "to_csv()" function to convert the dataframe to csv file. Read the csv file using pandas function "read_csv()".

```
[ ]  # Loading Dataset

     root_dir = "C:/Users/MSI/Downloads/bbc/bbc"

     texts = []
     categories = []

     for category in os.listdir(root_dir):
         category_dir = os.path.join(root_dir, category)
         if os.path.isdir(category_dir):
             for file_name in os.listdir(category_dir):
                 file_path = os.path.join(category_dir, file_name)
                 if os.path.isfile(file_path) and file_name.endswith('.txt'):
                     with open(file_path, 'r', encoding='unicode-escape') as file:
                         content = file.read()
                     texts.append(content)
                     categories.append(category)

     df = pd.DataFrame({'text': texts, 'category': categories})

     df.to_csv('C:/Users/MSI/Downloads/bbc/news_article.csv', index=False)



     df = pd.read_csv('C:/Users/MSI/Downloads/bbc/news_article.csv')
```

```
[ ]  df.head()
```

	text	category
0	Ad sales boost Time Warner profit\n\nQuarterly...	business
1	Dollar gains on Greenspan speech\n\nThe dollar...	business
2	Yukos unit buyer faces loan claim\n\nThe owner...	business
3	High fuel prices hit BA's profits\n\nBritish A...	business
4	Pernod takeover talk lifts Domecq\n\nShares in...	business

As we have loaded our data now its time to pre-process the data. We have created a preprocess_text function which takes text (articles) data as parameter.

a) In this function, first we are converting all text data into lowercase format as it will be easy for model to train when the data is in single case lower/upper.

b) Then using join() function and python list comprehension we have to remove the all the punctuations from our text data as punctuations are not required for training.

c) Then we are removing all the words which are stop words and are less relevant for training the model as these are commonly used words like the, is, are and so on.

d) Next we are performing stemming on each word and converting it into its original form. i.e raining-rain and rains-rain, in this case both raining and rains will be considered same as they play equal role no matter the context, this might improve accuracy of our model.

e) Then we have to apply "preprocess_text" function on each row of text column using "apply()" function.

f) Finally we are splitting the feature set "processed_text"and target label "category" into variable X and y respectively.

```
[ ]  # Preprocessing Data

     def preprocess_text(text):
         text = text.lower()
         text = ''.join([char for char in text if char.isalpha() or char.isspace()])
         text = ' '.join([word for word in text.split() if word not in stop_words])
         text = ' '.join([ps.stem(word) for word in text.split()])
         return text

     df['processed_text'] = df['text'].apply(preprocess_text)
     X= df['processed_text']
     y = df["category"]
```

```
[ ]  X.head()

     0    ad sale boost time warner profit quarterli pro...
     1    dollar gain greenspan speech dollar hit highes...
     2    yuko unit buyer face loan claim owner embattl ...
     3    high fuel price hit ba profit british airway b...
     4    pernod takeov talk lift domecq share uk drink ...
     Name: processed_text, dtype: object
```

```
[ ]  y.head()

     0    business
     1    business
     2    business
     3    business
     4    business
     Name: category, dtype: object
```

Now we are done with pre-processing of our data. Its time to perform feature extraction and feature selection. For this problem we are using three NLP features which are :

1. **Term frequency Inverse document frequency (TFIDF) :** This is a type of feature where it considers the word frequency in an article and its inverse document frequency, i.e how common the word is across all other documents (articles).

2. **Count Vectorizer :** This is a type of feature where it considers only the frequency of each word in a document. This helps in knowing which word is more frequent in which type of category based on which accuracy of classification model can be improved.

3. **N-gram Vectorizer :** This is a type of feature where it considers the sequence of n words depending on the n parameter we provide and captures the combinations of these n consecutive words. In this case we are taking combination of 2 consecutive words i.e bigram. This feature is a variation of CountVectorizer and thus it is applied using CountVectorizer() function with ngram_range as parameter.

```
count_vectorizer = CountVectorizer()
tfidf_vectorizer = TfidfVectorizer()
ngram_vectorizer = CountVectorizer(ngram_range=(2,2))

X_count = count_vectorizer.fit_transform(X)
X_tfidf = tfidf_vectorizer.fit_transform(X)
X_ngrams = ngram_vectorizer.fit_transform(X)

X_combined = pd.concat([pd.DataFrame(X_tfidf.toarray()),pd.DataFrame(X_count.toarray()),pd.DataFrame(X_ngrams.toarray())],axis=1)

selector = SelectKBest(chi2,k=500)
X_selected = selector.fit_transform(X_combined,y)
```

Steps for feature extraction and selection:

1. By creating instances of CountVectorizer(), TfidfVectorizer() and CountVectorizer(ngram_range(2,2)) we will perform feature extraction on X variable which is our feature set and store it in three different variables respectively.

2. Then we will use concat() function to concatenate all three features which we extracted and store it in a single dataframe “X_combined”

3. As we have combined all the extracted features, we have to create instance of inbuilt SelectKBest(chi2,k= 500) class where chi2 belongs to Chi-squared test, it measures the dependency between feature and target variable. Feature with highest chi-sqaure score are considered more relevant and selected. And k=500 such features are selected for training the model.

After feature extraction and selection now its time to split the dataset into Training, Development (validation) and Test set. We are pre-defining these three sets and this is done using train_test_split() as seen in code snippet below. we have splitted the dataset as follows :

Training : 70%
Development (validation): 15%
Testing : 15%

```
[ ]  # Splitting our dataset into Training, Development (Validation) and Test set.

     X_train, X_val, y_train, y_val = train_test_split(X_selected,y, test_size=0.3, random_state=23)
     X_val , X_test, y_val, y_test = train_test_split(X_val, y_val, test_size=0.5, random_state=23)
```

Next step is to train and evaluate our machine learning model.

1. In this classification problem we are using Support Vector Machine Classifier as our ML model. so first we have create instance model of SVC() class. And define our parameter_grid which will contain all the hyperparameters you want to tune so asto find the best hyperparameter set. SVC has C, gamma and kernel hyperparameters through which we can tune our model accuracy.

2. Next, we are using crossvalidation KFold to perform random folds of training set and evaluate the performance of model based on each fold considered as validation set.

3. We are using GridsearchCV to find the best hyperparameters for the model from the param_grid provided based on scoring as accuracy. The set of hyperparameters with higher accuracy score are considered best. This accuracy score is evaluated on the development (validation) set. The significance of development set in model training is that it is used to tune the hyperparameters of our model and find the best model.

4. In the code snippet below it can be seen that we are performing Gridsearch and tuning hyperparameters C, gamma and kernel (linear,rpf or polynomial).

```
[ ]  # Training model and using development set for tuning hyperparameters for better accuracy.

     model = SVC()


     param_grid = {'C' : [0.1,0.5,1,10] , 'gamma' : [0.01,0.1,1,1.5], 'kernel' : ['linear','rbf', 'poly']}

     cv = KFold(n_splits=5, shuffle=True, random_state=23)
     grid_search = GridSearchCV(estimator = model, param_grid=param_grid, cv=cv, scoring='accuracy', verbose=3)

     grid_search.fit(X_train, y_train)

     best_model = grid_search.best_estimator_
     val_score = best_model.score(X_val,y_val)

     print("Best parameters:", grid_search.best_params_)

     Fitting 5 folds for each of 48 candidates, totalling 240 fits
     [CV 1/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.955 total time=   6.6s
     [CV 2/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.958 total time=   0.0s
     [CV 3/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.968 total time=   0.0s
     [CV 4/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.958 total time=   0.0s
     [CV 5/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.955 total time=   0.0s
     [CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.591 total time=   0.4s
     [CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.543 total time=   0.3s
     [CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.556 total time=   0.3s
     [CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.534 total time=   0.3s
     [CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.673 total time=   0.3s
     [CV 1/5] END ....C=0.1, gamma=0.01, kernel=poly;, score=0.668 total time=   0.1s
```

```
Best parameters: {'C': 0.1, 'gamma': 0.01, 'kernel': 'linear'}
```

5. Thus after successfully training and evaluating our model we are able to get the best model with its hyperparameters set as seen in figure above.

Now as we have trained our model, we can perform predictions on test data and evaluate its performance on test data to obtain its classification report with accuracy, macro-averaged recall,precision,F1-score. This can be done using instance of best_model and applying predict() function and for accuracy score we can use accuracy_score() function with parameters as actual and predicted labels. Similarly, to generate complete evaluation report
we are using classification_report() function. As seen in figure below we are able to get model accuracy of 94% and also macro-averaged recall,precision,F1-score is 94%.

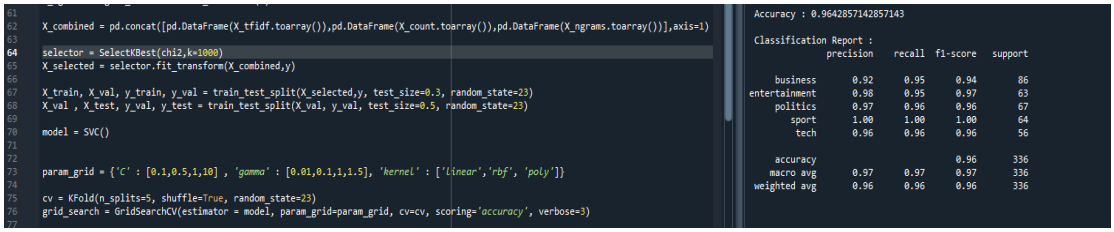
```
# Performing predictions and evaluating models performance based on best_model found during gridsearch.

y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("\n Accuracy :", accuracy)

print("\n Classification Report :")
print(classification_report(y_test, y_pred))
```

```
 Accuracy : 0.9434523809523809

 Classification Report :
               precision    recall  f1-score   support

     business       0.93      0.93      0.93        86
entertainment       0.95      0.97      0.96        63
     politics       0.94      0.90      0.92        67
        sport       0.98      0.98      0.98        64
         tech       0.91      0.95      0.93        56

     accuracy                           0.94       336
    macro avg       0.94      0.94      0.94       336
 weighted avg       0.94      0.94      0.94       336
```

Critical Reflection on improvements in future:

1. The current model is only trained on the dataset “bbc_news” so it might be a case that provided dataset is biased so model can predict biased outputs. For this large amount of data from difference data source should be feeded to ML model to reduce the bias.

2. The model accuracy can be improved to 96% if we increase the number of k features to 1000 in feature selection.



```
X_combined = pd.concat([pd.DataFrame(X_tfidf.toarray()),pd.DataFrame(X_count.toarray()),pd.DataFrame(X_ngrams.toarray())],axis=1)

selector = SelectKBest(chi2,k=1000)
X_selected = selector.fit_transform(X_combined,y)

X_train, X_val, y_train, y_val = train_test_split(X_selected,y, test_size=0.3, random_state=23)
X_val , X_test, y_val, y_test = train_test_split(X_val, y_val, test_size=0.5, random_state=23)

model = SVC()


param_grid = {'C' : [0.1,0.5,1,10] , 'gamma' : [0.01,0.1,1,1.5], 'kernel' : ['linear','rbf', 'poly']}

cv = KFold(n_splits=5, shuffle=True, random_state=23)
grid_search = GridSearchCV(estimator = model, param_grid=param_grid, cv=cv, scoring='accuracy', verbose=3)
```

```
Accuracy : 0.9642857142857143

Classification Report :
              precision    recall  f1-score   support

    business       0.92      0.95      0.94        86
entertainment      0.98      0.95      0.97        63
    politics       0.97      0.96      0.96        67
       sport       1.00      1.00      1.00        64
        tech       0.96      0.96      0.96        56

    accuracy                           0.96       336
   macro avg       0.97      0.97      0.97       336
weighted avg       0.96      0.96      0.96       336
```

3. Also extraction of features can be improved in current model by implementing features like POS tagging, Word2Vec or GloVe

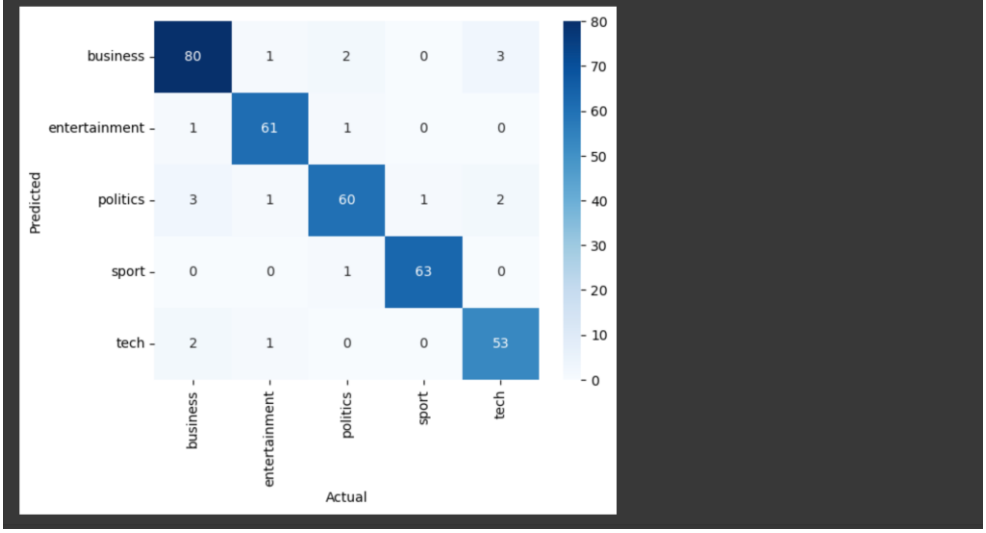
4. Model ensembling can also be used where multiple classifier models like Random forest classifier, Decision tree classifier and so on can be trained to produce more improved results.

5. It might also be a case that the preprocessing techniques used are creating biased output due to chosen stopwords and stemmer, resulting in filtering out words which might be relevant to specific category.

Error Analysis :

Based on the confusion matrix below, we can see the errors in model prediction.

```
# Confusion matrix which gives clear understanding between the actual and predicted labels.

confusion_mat = confusion_matrix(y_test,y_pred)

sns.heatmap(confusion_mat,annot=True, cmap='Blues',xticklabels=best_model.classes_, yticklabels=best_model.classes_)
plt.xlabel("Actual")
plt.ylabel("Predicted")

plt.show()
```



Lets take an example of False positive error :

Model is classifying some categories as “business” category. This type of error can be improved if :

1. The data size is increased to reduce the bias and improves training performance.
2. The feature extraction technique Named Entity Recognition can also help in improving model performance as it will help model to train on relevant feature by identifying entities which belongs to specific category of article.
3. The specific category where misclassification is seen more, in this case the model can be feed with more training data of such category.