MIDAS TASK3 - NLP

Name: Sarth Kulkarni
Institute: Pune Institute of Computer Technology

Assignment Details-
Use the given dataset to build a model to predict the category using description.

Steps followed to implement the model are:

Step1: Import all the necessary libraries like scikit-learn,nltk,re and importing the dataset. Run the code dataset.head() and dataset.tail() to see the first 5 entries and the last 5 entries.

Step2: Next, we see that the product_category_tree needs to be separated into its primary category for the model to predict this primary category using the description. The product category tree is then separated into its different levels by splitting it and getting the required primary category. Next, we drop the product_category_ tree and other columns such as url,pid,image,link from the dataset and keep only the description and the primary_category in the dataset.

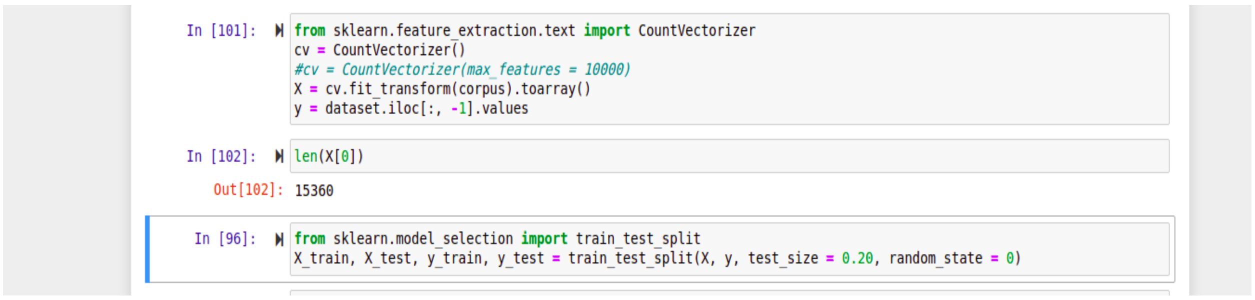
Step3: We now plot a graph to visualize data more clearly.In this graph plotted using matplotlib we find that that the most frequent primary category is Clothing owing up to 30% i.e 6000 from 20000 rows of data available.

Step4 : In this step and the next step we clean and deep clean our description column to predict the primary category further. We start by importing the nlp libraries and write a function that will take the string as input and return cleaned string as output. The cleaned string will have only alphabets,removed punctuations and single space between words.We apply this function to the description column of our dataset.

Step5 : In this step we will deep clean the description column and make a corpus of words from the description column of the dataset.We will simplify the description column using the re library and nltk library and make an empty list called corpus which will be the main words on which our model will be trained. We will download the stopwords which consists of all the articles like the,a,and basically the words which have no use in our prediction. These stopwords will also not be included in the corpus.

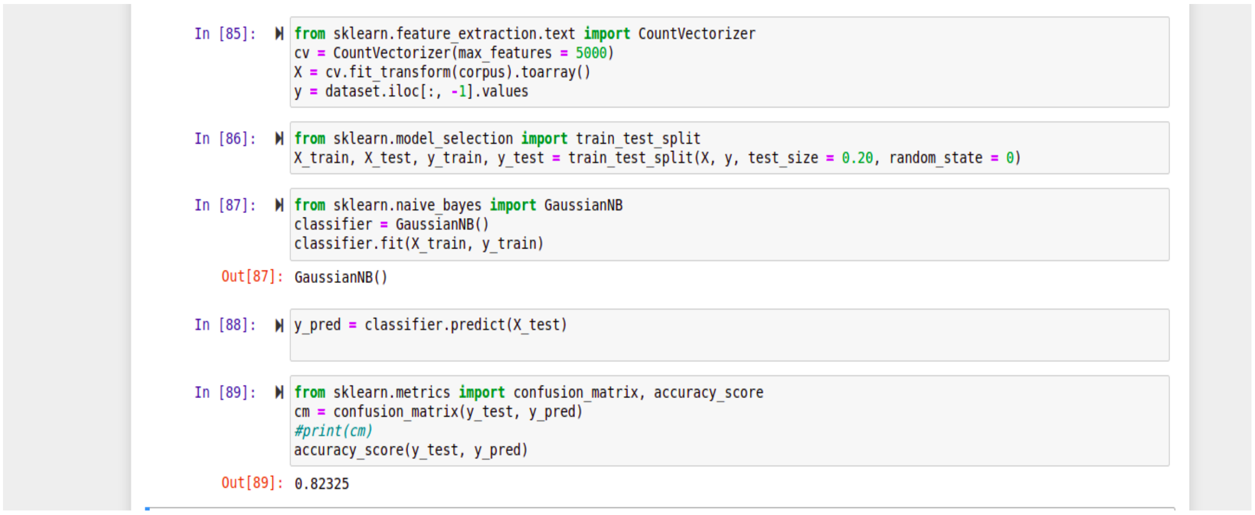
Next, we will import a Porter Stemmer class which will be useful to apply stemming to our descriptions.Stemming consists of taking only the root of the word to simplify the description and making the sparse matrix correct otherwise all the different forms of verb will have different columns making the sparse matrix complex.
We will iterate through all of the description again from the dataset[20000 here] and clean them and make a corpus ready to be trained .

Step6: Now,we will create a bag of words .From the feature_extraction module we will get to the text sub module and call the CountVectorizer class . cv is an instance of the CountVectorizer class which takes max_features as the only parameter. For example in our corpus there are

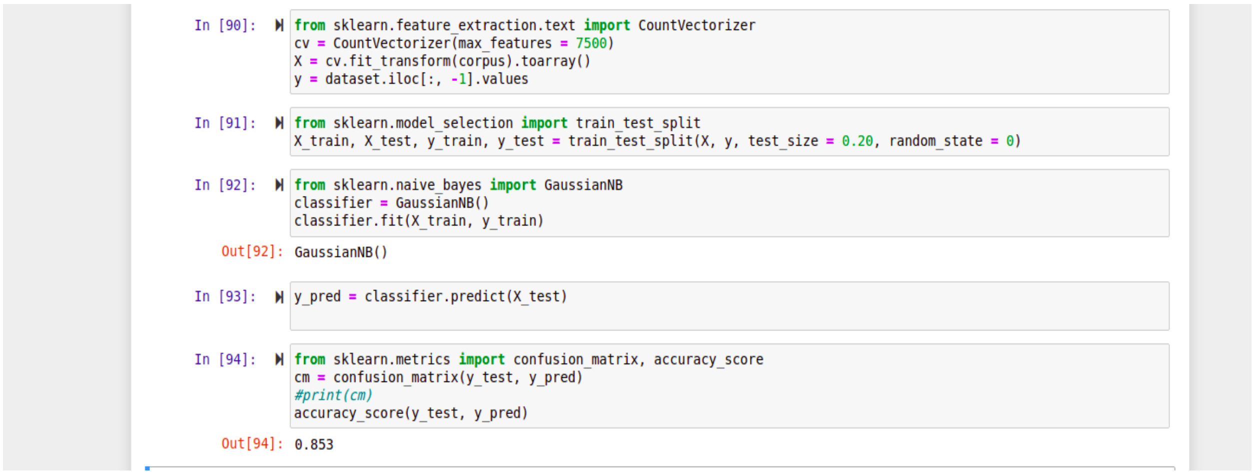


```
In [101]:  from sklearn.feature_extraction.text import CountVectorizer
            cv = CountVectorizer()
            #cv = CountVectorizer(max_features = 10000)
            X = cv.fit_transform(corpus).toarray()
            y = dataset.iloc[:, -1].values

In [102]:  len(X[0])

Out[102]:  15360

In [96]:   from sklearn.model_selection import train_test_split
           X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

words like dilli etc which are not important and so we have to decide what will be the number to be provided in the parameter. For that we first run the CountVectorizer without the max_features parameter and then run the code len(X[0]) to get the total features which here is 15360. So from these 15360, I have taken only the 14000 most important words in the bag of words. This is what we have to try and check for different values and with 14000 the results were considerably good. The examples of the code blocks of len(X[0]) and other accuracies with different number as max_feature parameter are shown below:



```
In [85]:  from sklearn.feature_extraction.text import CountVectorizer
          cv = CountVectorizer(max_features = 5000)
          X = cv.fit_transform(corpus).toarray()
          y = dataset.iloc[:, -1].values

In [86]:  from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

In [87]:  from sklearn.naive_bayes import GaussianNB
          classifier = GaussianNB()
          classifier.fit(X_train, y_train)

Out[87]:  GaussianNB()

In [88]:  y_pred = classifier.predict(X_test)

In [89]:  from sklearn.metrics import confusion_matrix, accuracy_score
          cm = confusion_matrix(y_test, y_pred)
          #print(cm)
          accuracy_score(y_test, y_pred)

Out[89]:  0.82325
```



```
In [90]:  from sklearn.feature_extraction.text import CountVectorizer
          cv = CountVectorizer(max_features = 7500)
          X = cv.fit_transform(corpus).toarray()
          y = dataset.iloc[:, -1].values

In [91]:  from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

In [92]:  from sklearn.naive_bayes import GaussianNB
          classifier = GaussianNB()
          classifier.fit(X_train, y_train)

Out[92]:  GaussianNB()

In [93]:  y_pred = classifier.predict(X_test)

In [94]:  from sklearn.metrics import confusion_matrix, accuracy_score
          cm = confusion_matrix(y_test, y_pred)
          #print(cm)
          accuracy_score(y_test, y_pred)

Out[94]:  0.853
```

After selecting the max_features parameter the corpus was fitted on X and the primary_category column on y. From here, the data was split as a training set and test set using a general 80-20 percent method.

Step7: After this, the naive bayes model was trained on the training set and predicted the test set results. We have trained this model on both the Gaussian Naive Bayes and the Multinomial Naive Bayes. Multinomial Naive Bayes uses term frequency i.e. the number of times a given term appears in a document whereas Gaussian Naive Bayes is used when the features have continuous values. We see that the result from Multinomial NB has an accuracy of 92% whereas the Gaussian Naive Bayes reports an accuracy of 86%.

To improve the accuracy of the model more we can work with the max_features parameter in Step6 , we can also include some other columns in our dataset to try if they improve the accuracy even further.

Other models we can try with can be GridSearchCV or Bernoulli Naive Bayes. We can also visualize this data more with the help of more plots from seaborn or matplotlib.