Name: Pranav Sanjay Kalambe

Branch/Year: EXTC-Second Year

Roll no.: 1913023

Email: pranav.kalambe@somaiya.edu (mailto:pranav.kalambe@somaiya.edu)

Aim: *Classify the given text into different Categories such as Computer, Politics, Relegion, Science, Sports using multinomial navie bayes and tfidf*

In [1]:

```
import time
import sklearn.datasets as skd
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
start_time = time.time()
```

In [3]:

```
categories = ['computer','politics','religion','science','sports']
fileaddressoftrain = '20news-bydate-train'
fileaddressoftest = '20news-bydate-test'
encoder = 'ISO-8859-1'
```

In [4]:

```
news_test = skd.load_files(fileaddressoftest,encoding = encoder , categories = categories)
news_train = skd.load_files(fileaddressoftrain,encoding = encoder , categories = categories
```

Let's have a look at our training data elements

In [5]:

```
news_train.keys()
```

Out[5]:

```
dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])
```

In [6]:

```
news_train.target_names # Names of target file
```

Out[6]:

```
['computer', 'politics', 'religion', 'science', 'sports']
```

In [7]:

```
len(news_train.filenames) # These are the total no. of text document in dataset
```

Out[7]:

```
9537
```

Plotting number of input data

In [8]:

```
news_train.target # So here target names are labeled with numbers as target
computer = np.sum(news_train.target == 0)
politics = np.sum(news_train.target == 1)
religion = np.sum(news_train.target == 2)
science = np.sum(news_train.target == 3)
sports = np.sum(news_train.target == 4)
```

In [9]:

```
train_data_list = [computer,politics,religion,science,sports]
train_df = pd.DataFrame(train_data_list,categories)
train_df = train_df.transpose()
train_df.index = ["Samples"]
train_df
```

Out[9]:

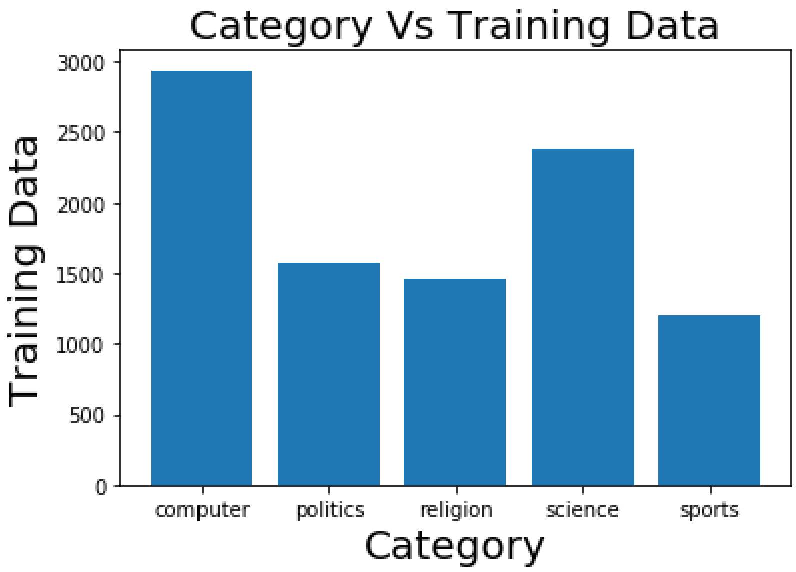
	computer	politics	religion	science	sports
Samples	2936	1575	1456	2373	1197

In [10]:

```
plt.bar(news_train.target_names,train_data_list)
plt.xlabel('Category',fontsize = 20)
plt.ylabel('Training Data',fontsize = 20)
plt.title('Category Vs Training Data', fontsize = 20)
```

Out[10]:

```
Text(0.5, 1.0, 'Category Vs Training Data')
```



Convert the using CountVectorizer

It basically counts the unique words and then gives the frequency of those words

In [11]:

```
from sklearn.feature_extraction.text import CountVectorizer
count = CountVectorizer()
count.fit(news_train.data)
x_train_transform = count.transform(news_train.data)
```

In [12]:

```
x_train_tf = count.fit_transform(news_train.data)
type(x_train_tf)
```

Out[12]:

```
scipy.sparse.csr.csr_matrix
```

In [13]:

```
x_train_tf.shape
```

Out[13]:

```
(9537, 121620)
```

```
So here 9537 samples and 121360 unique words that is features
```

Term Frequency: This summarizes how often a given word appears within a document.

Inverse Document Frequency: This downscales words that appear a lot across documents.

Here every word is givem weight depending on their importance in the document

In [14]:

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf = TfidfTransformer()
tfidf.fit(x_train_tf)
x_train_tfidf = tfidf.transform(x_train_tf)
```

In [15]:

```
print(x_train_tfidf) # These are values of Tfidf
```

```
  (0, 119972)	0.05104820354101887
  (0, 119949)	0.05644088513070946
  (0, 118319)	0.057913296489725506
  (0, 117329)	0.03875018096562748
  (0, 116957)	0.03252891101264183
  (0, 116951)	0.027003025574690278
  (0, 116763)	0.04798865098323608
  (0, 116756)	0.017963115481053022
  (0, 116724)	0.019553717811371784
  (0, 116368)	0.021101711280599237
  (0, 116346)	0.04122887793639142
  (0, 116179)	0.09429876073228571
  (0, 116178)	0.062487023922651826
  (0, 116176)	0.062487023922651826
  (0, 116170)	0.11470631573275727
  (0, 116169)	0.057913296489725506
  (0, 116084)	0.013601880273786048
  (0, 116003)	0.0413805989136269
  (0, 115919)	0.02563196612597486
  (0, 115832)	0.02065602316762973
  (0, 115826)	0.014845845243365468
  (0, 115789)	0.09526377750851737
  (0, 115575)	0.062487023922651826
  (0, 115132)	0.012335815588370078
  (0, 114291)	0.04876584162387283
  :	:
  (9536, 61067)	0.05848559812674219
  (9536, 59632)	0.029454685272633743
  (9536, 58269)	0.22987882529179968
  (9536, 57800)	0.03942429855823775
  (9536, 56225)	0.04561879712275488
  (9536, 53278)	0.057229437351977934
  (9536, 52934)	0.013095904698205808
  (9536, 51483)	0.07990365816883659
  (9536, 49503)	0.09599069953934083
  (9536, 49367)	0.07304989433774324
  (9536, 48527)	0.19900771828349104
  (9536, 47529)	0.04954071477667163
  (9536, 46816)	0.03703782371562454
  (9536, 45070)	0.0564543055330558
  (9536, 44782)	0.025568065438070567
  (9536, 42997)	0.22987882529179968
  (9536, 42496)	0.06730193096661063
  (9536, 41565)	0.13810129071728014
  (9536, 39199)	0.05050781132426433
  (9536, 37984)	0.04927504673502748
  (9536, 34714)	0.04598977938155798
  (9536, 34179)	0.12401680206434292
  (9536, 27359)	0.022980278616298717
  (9536, 24572)	0.03774687487553313
  (9536, 7487)	0.04363771350043012
```

Training model using navie bayes multinomialNB

In [16]:

```
from sklearn.naive_bayes import MultinomialNB
classify = MultinomialNB() # object created for classification
model = classify.fit(x_train_tfidf,news_train.target) # making model by matching tfidf data
```

```
Thats all model has been created, now let's check it how it works
```

Checking model on a custom input

In [17]:

```
# Have a look at our target names with their index no. in list
print(news_train.target_names)
tempdict = dict(enumerate(news_train.target_names))
print(dict(enumerate(news_train.target_names)))
```

```
['computer', 'politics', 'religion', 'science', 'sports']
{0: 'computer', 1: 'politics', 2: 'religion', 3: 'science', 4: 'sports'}
```

In [18]:

```
# This is for trying with custom input
# Since your model is trained now no need to fit
# only transforming through vectorizers and then predicting
pk = input("Enter custom input: ")
#custom_input = [pk,'God is Great','GPU makes computer run fast','science is good','ministe
custom_input = [pk]
x_count = count.transform(custom_input) # count is countvectorizer
x_tfidf = tfidf.transform(x_count) # tfidf is tfidfvectorizer
x_predict = classify.predict(x_tfidf) # classify is naive bayes multinominalnb object whuch
print(f"\nThe given news is under {tempdict[int(x_predict)]} category")
```

```
Enter custom input: h

The given news is under computer category
```

Let's work it on test data and find accuracy

In [19]:

```
news_test_count = count.transform(news_test.data)
news_test_tfidf = tfidf.transform(news_test_count)
news_test_predict = classify.predict(news_test_tfidf)
```

In [20]:

```
from sklearn import metrics
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(news_test.target,news_test_predict)
print('Accuracy of my model is',accuracy)
print('Accuracy in percentage is',round(100*accuracy,2),'%')
```

```
Accuracy of my model is 0.8382167611846251
Accuracy in percentage is 83.82 %
```

In [21]:

```
report = metrics.classification_report(news_test.target, news_test_predict, target_names=ne
confusion_matrix = metrics.confusion_matrix(news_test.target,news_test_predict)
print(report)
```

```
              precision    recall  f1-score   support

    computer       0.80      0.98      0.88      1955
    politics       0.95      0.82      0.89      1050
    religion       0.98      0.81      0.89       968
     science       0.72      0.84      0.78      1579
      sports       1.00      0.54      0.70       796

    accuracy                           0.84      6348
   macro avg       0.89      0.80      0.83      6348
weighted avg       0.86      0.84      0.84      6348
```

Making a dataframe of confusion matrix

In [22]:

```
import pandas as pd
df = pd.DataFrame(confusion_matrix)
```

In [23]:

```
df.index = news_test.target_names
df.columns = news_test.target_names
df
```

Out[23]:

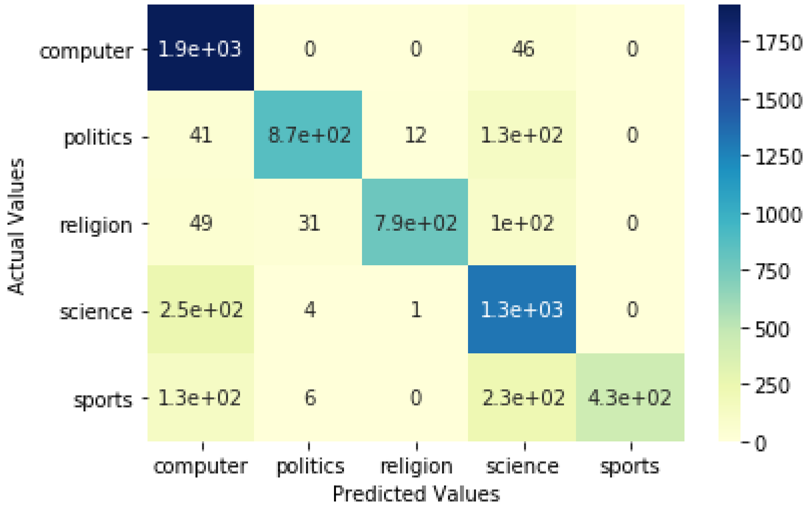
	computer	politics	religion	science	sports
computer	1909	0	0	46	0
politics	41	866	12	131	0
religion	49	31	786	102	0
science	247	4	1	1327	0
sports	127	6	0	230	433

In [24]:

```
import seaborn as sb
sb.heatmap(df,annot=True, cmap="YlGnBu")
plt.xlabel("Predicted Values",)
plt.ylabel("Actual Values")
```

Out[24]:

```
Text(33.0, 0.5, 'Actual Values')
```



Time Required

In [25]:

```
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 31.285295009613037 seconds ---
```

In [ ]: