CODE ANALYSIS

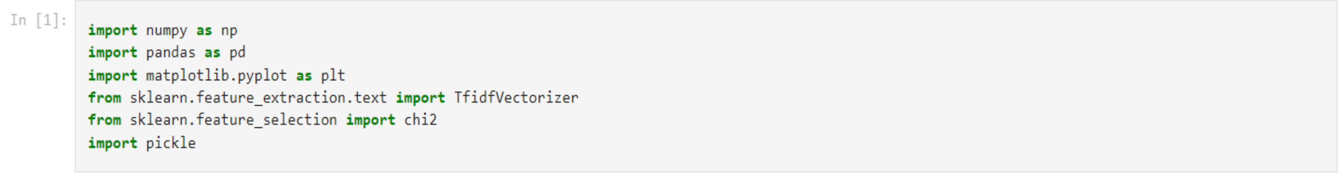
Preparation Steps

In this section, we look at each part of the code and understand the steps covered above. For this document, we stick to the text classification steps since it requires proper detailing. For processing steps, refer to the steps covered above and the code in link below.

For the BBC News classification code, please refer to the code link below.

Link for prediction steps code: https://github.com/dhrubasil/BBC-News-Classification/blob/master/News%20Classificaiton.ipynb

(A). Importing important packages and libraries:



```
In [1]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import chi2
import pickle
```

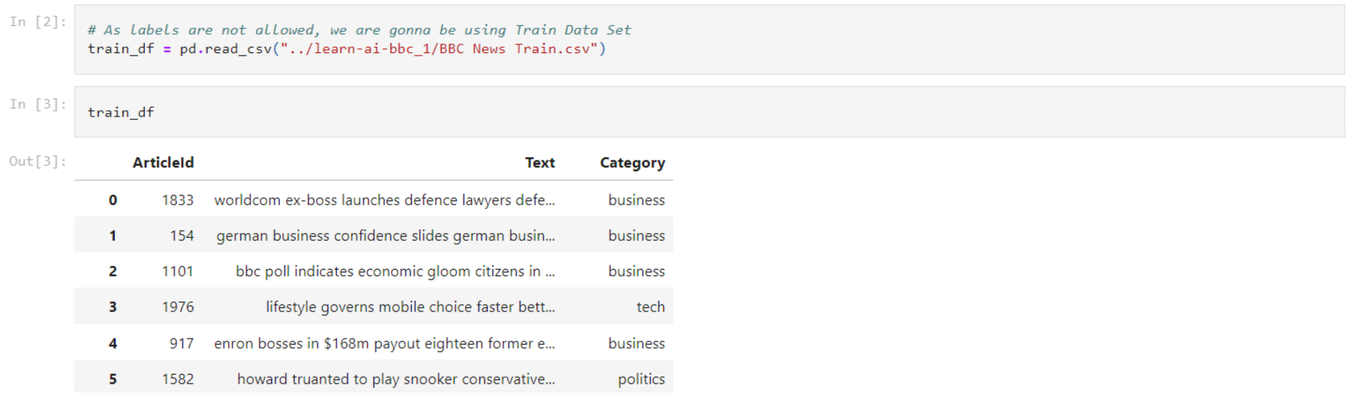
We have imported ‘**numpy**’ and ‘**pandas**’ for basic data manipulation in our project.

We have imported ‘**TfidVectorizer**’ to convert a collection of raw documents to a matrix of TF-IDF features to measure how relevant a word is to a document; it is equivalent to ‘**CountVectorizer**’.

We have imported ‘**chi2**’ to compute chi-squared stats between each non-negative feature such as term counts in document classification relative to the classes.

We have imported ‘**pickle**’ for serializing and de-serializing a Python object structure. Any object on python can be pickled so that it can be saved on disk. At first Python pickle serialize the object and then converts the object into a character stream so that this character stream contains all the information necessary to reconstruct the object in another python script.

(B). Loading the given Dataset from Kaggle:



```
In [2]:
# As labels are not allowed, we are gonna be using Train Data Set
train_df = pd.read_csv("../learn-ai-bbc_1/BBC News Train.csv")
```

```
In [3]:
train_df
```

Out[3]:

	ArticleId	Text	Category
0	1833	worldcom ex-boss launches defence lawyers defe...	business
1	154	german business confidence slides german busin...	business
2	1101	bbc poll indicates economic gloom citizens in ...	business
3	1976	lifestyle governs mobile choice faster bett...	tech
4	917	enron bosses in $168m payout eighteen former e...	business
5	1582	howard truanted to play snooker conservative...	politics

This training dataset is already in CSV format, and it has 1490 different texts, each labeled under one of 5 categories: entertainment, sport, tech, business, or politics. The dataframe has three columns, which is ‘*category*’ will be our **label**, ‘*ArticleId*’ and ‘*text*’ which will be our **input data** for **BERT**.

(C). Performing Exploratory Data Analysis (EDA) of the dataset:

1485	857	double eviction from big brother model caprice...	entertainment	4
1486	325	dj double act revamp chart show dj duo jk and ...	entertainment	4
1487	1590	weak dollar hits reuters revenues at media gro...	business	0
1488	1587	apple ipod family expands market apple has exp...	tech	1
1489	538	santy worm makes unwelcome visit thousands of ...	tech	1

1490 rows × 4 columns

```
In [7]:
# Frequency Distribution for Each Class
print (train_df["Category"].value_counts())
print (train_df["Label_Encoding"].value_counts())


# Based on frequency distribution  we can say that data is balanced, not suffering from class imbalance.
```

```
sport            346
business         336
politics         274
entertainment    273
tech             261
Name: Category, dtype: int64
3    346
0    336
2    274
4    273
1    261
Name: Label_Encoding, dtype: int64
```

Using ‘**value_counts()**’ we can see the unique values of the different categories and its label encoding (3: sport with 346 texts, 0: business with 336 texts, 2: politics with 274 texts, 4: entertainment with 273 texts and 1: tech with 261 texts).

```
In [11]:
# Preserving the Category Coding
category_labels_to_id = {"business":0,"tech":1,"politics":2,"sport":3,"entertainment":4}
id_to_category = {0:"business",1:"tech",2:"politics",3:"sport",4:"entertainment"}
```

```
In [12]:
# Check the number of Null in our Data Set
train_df.isnull().sum()
```

```
Out[12]: ArticleId         0
         Text              0
         Category          0
         Label_Encoding    0
         dtype: int64
```

We have checked if there’s any null or missing values in the training dataset using ‘**isnull().sum()**’.

(D). TF-IDF:

```
In [13]:
"""
Setting TF-IDF
--------------
Apply sublinear tf scaling, i.e. replace tf with 1 + log(tf).
min_df = Ignore all the words that have a document frequency less than min_df
"""

tfidf = TfidfVectorizer(sublinear_tf=True, min_df=7, norm='l2', encoding='utf-8', ngram_range=(1, 3),lowercase = True,stop_words='english')
```

```
In [14]:
# Training the tfidf feature
tfidf_feature = tfidf.fit_transform(train_df.Text).toarray()
```

```
In [74]:
with open('news_classification_tfidf_vectorizer', 'wb') as output:
    pickle.dump(tfidf, output)
```

Here, we have first initialized the TfidVectorizer function with these parameters:

1. **sublinear_tf = True**: To apply sublinear tf scaling, ie., replace tf with ‘1+log(tf)’.

It is not always true that multiple occurrences of a term in a document mean more significance of that term in proportion to the number of occurrences (ex., if a word ‘country’ occurs for 20 times, it doesn’t indicate that the word will have 20 times the relevance). It’s a method of **reducing the impact of repeated terms**.

Sublinear tf-scaling is modification of term frequency, which calculates weight as following,

$\boldsymbol{Wf}(\boldsymbol{t},\boldsymbol{d})$

a. if $tf_{t,d} = 0$

b. then return (0)

c. else return $(1 + \log{(tf_{t,d})})$

2. **min_df = 7**: 'min_df' is used for removing terms that appear too infrequently. Here, we ignore terms that appear in less than 7 documents.

3. **norm = l2**: Sum of squares of vector elements is 1. The cosine similarity between two vectors is their dot product when l2 norm has been applied.

4. **encoding='utf-8'**: The file has been encoded in 'utf-8' format to work upon.

5. **ngram_range = (1, 3)**: An n-gram is just a string of n words in a row. Here, the parameter 'ngram_range= (1,3), where 1 is the minimum and 3 is the maximum size of ngrams we will include in our features.

6. **lowercase = True**: It is used to convert all the characters to lowercase before tokenizing.

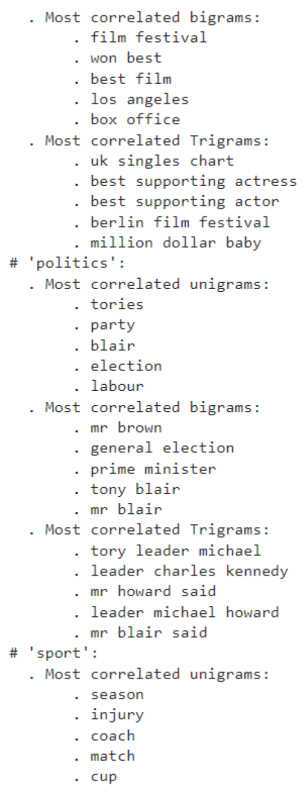
7. **stop_words = 'english'**: Some of the words detected in the vocabulary of unique n-grams have little value, such as "would", "you", or "your". These are so-called "stop words" and can safely be removed from the data. Stop words generally don't contribute very much and can massively bloat the size of your dataset, which increases model training times and causes various other issues. As a result, it's a common practice to remove stop words. The easiest way is to do this automatically by passing the argument (stop_words='english').

After initializing the 'Tfidf', we **train the dataset** by transforming the dataframe using '**fit_transform()**' and '**toarray()**' method converts the efficient representation of a sparse matrix that sklearn uses to our ordinary readable dense ndarray representation. Furthermore, we save the training data using '**pickle**'.

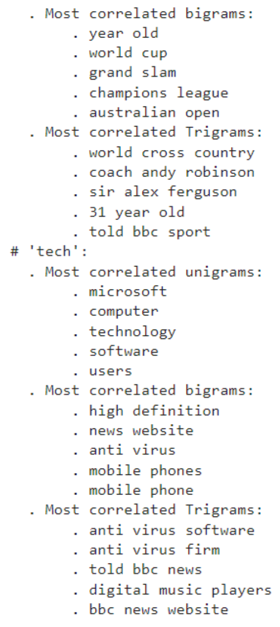
```
In [15]:
N = 5  # We are going to look for top 3 categories
labels = train_df.Label_Encoding

#For each category, find words that are highly corelated to it
for category, category_id in sorted(category_labels_to_id.items()):
  features_chi2 = chi2(tfidf_feature, labels == category_id)          # Do chi2 analyses of all items in this category
  indices = np.argsort(features_chi2[0])                              # Sorts the indices of features_chi2[0] - the chi-squared stats of each feat
  feature_names = np.array(tfidf.get_feature_names())[indices]        # Converts indices to feature names ( in increasing order of chi-squared sta
  unigrams = [v for v in feature_names if len(v.split(' ')) == 1]     # List of single word features ( in increasing order of chi-squared stat val
  bigrams = [v for v in feature_names if len(v.split(' ')) == 2]      # List for two-word features ( in increasing order of chi-squared stat value
  trigrams = [v for v in feature_names if len(v.split(" "))==3]
  print("# '{}':".format(category))
  print("  . Most correlated unigrams:\n       . {}".format('\n       . '.join(unigrams[-N:]))) # Print 3 unigrams with highest Chi squared stat
  print("  . Most correlated bigrams:\n       . {}".format('\n       . '.join(bigrams[-N:]))) # Print 3 bigrams with highest Chi squared stat
  print("  . Most correlated Trigrams:\n       . {}".format('\n       . '.join(trigrams[-N:]))) # Print 3 bigrams with highest Chi squared stat
```

```
# 'business':
  . Most correlated unigrams:
       . economy
       . oil
       . growth
       . bank
       . shares
  . Most correlated bigrams:
       . chief executive
       . oil prices
       . stock market
       . economic growth
       . analysts said
  . Most correlated Trigrams:
       . current account deficit
       . pre tax profits
       . chief financial officer
       . high oil prices
       . securities exchange commission
# 'entertainment':
  . Most correlated unigrams:
       . awards
       . album
       . singer
       . actor
       . film
```



```
  . Most correlated bigrams:
       . film festival
       . won best
       . best film
       . los angeles
       . box office
  . Most correlated Trigrams:
       . uk singles chart
       . best supporting actress
       . best supporting actor
       . berlin film festival
       . million dollar baby
# 'politics':
  . Most correlated unigrams:
       . tories
       . party
       . blair
       . election
       . labour
  . Most correlated bigrams:
       . mr brown
       . general election
       . prime minister
       . tony blair
       . mr blair
  . Most correlated Trigrams:
       . tory leader michael
       . leader charles kennedy
       . mr howard said
       . leader michael howard
       . mr blair said
# 'sport':
  . Most correlated unigrams:
       . season
       . injury
       . coach
       . match
       . cup
```



```
  . Most correlated bigrams:
       . year old
       . world cup
       . grand slam
       . champions league
       . australian open
  . Most correlated Trigrams:
       . world cross country
       . coach andy robinson
       . sir alex ferguson
       . 31 year old
       . told bbc sport
# 'tech':
  . Most correlated unigrams:
       . microsoft
       . computer
       . technology
       . software
       . users
  . Most correlated bigrams:
       . high definition
       . news website
       . anti virus
       . mobile phones
       . mobile phone
  . Most correlated Trigrams:
       . anti virus software
       . anti virus firm
       . told bbc news
       . digital music players
       . bbc news website
```

Now, we will look for top 5 categories where for each category, we find words that are highly correlated to it using '**chi square**' test which is a statistical method to check the relationship (or correlation) between the variables. We converts the indices to '**feature_names**' and get a list of **unigram, bigrams, and trigrams** (in increasing order of chi-squared stat values). Finally, printed the unigram, bigrams and trigrams with the highest value of chi-squared stat values.

(E). Splitting the dataset into Training and Test data and building the Random Forest Classifier:

```
In [16]:  # Train Test Split
          from sklearn.model_selection import train_test_split
          from sklearn.ensemble import RandomForestClassifier
          from sklearn.model_selection import cross_val_score

          model = RandomForestClassifier()
```

```
In [17]:  #Split Data
          X_train, X_test, y_train, y_test= train_test_split(tfidf_feature, labels, test_size=0.25, random_state=0)
```

```
In [18]:  model.fit(X_train,y_train)
```

```
Out[18]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                criterion='gini', max_depth=None, max_features='auto',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

We have imported '**RandomForestClassifier**' & '**train_test_split**' from scikit-learn library for classification and training, testing, and splitting the pre-processed data.

We have imported 'cross_val_score' from scikit-learn library to evaluate the test model over multiple folds of our dataset.

The final pre-processing step is to split our processed dataset into '**training**' and '**test**' sets. 'test_size' parameter specifies the ratio of the test set, in which we used to split up **25% of the data into the test set and 75% for training**. We have created our **random forest classifier** (**model**) and then train it on the train set.

The '**fit**' method of this class is called to **train the algorithm on the training data**, which is passed as parameter to the 'fit' method.

(F). Evaluating the Model:

```
In [75]:  with open('news_classification_rf_model', 'wb') as output:
              pickle.dump(model, output)
```

```
In [20]:  predicted_train = model.predict(X_train)
          predicted_test = model.predict(X_test)
```

```
In [21]:  from sklearn.metrics import classification_report
```

```
In [22]:  print (classification_report(y_test,predicted_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        86
           1       0.97      0.93      0.95        67
           2       0.94      0.94      0.94        63
           3       0.93      0.98      0.95        84
           4       0.95      0.96      0.95        73

    accuracy                           0.95       373
   macro avg       0.95      0.95      0.95       373
weighted avg       0.95      0.95      0.95       373
```

After training the random forest model, we saved the model using '**pickle**'. After our classifier is trained, we made predictions on our test data. To make predictions, the '**predict**' method of the 'RandomForestClassifier' class is used.

We have imported '**classification_report**' to evaluate and analyze the random forest model where in classification, some commonly used metrics are **'confusion matrix', 'accuracy score', 'precision', 'recall' & 'F1-score or F-measure'** which can be used using the Scikit-learn '**metric**' library (**accuracy_score, confusion_matrix, classification_report**).

From the classification report of 373 test instances, we can see that the **accuracy of the model is 95%,** which is quite good. From the **weighted average**: the model is **95% precise** with **95% recall** and **f1-score of 0.95**.