CSE 802: Pattern Recognition and Analysis

Project Report

By: Vishal Asnani

Introduction

News headlines are very important part of our daily lives. From classic newspapers to the digital social media, the importance of news media is very much in our lives. With the introduction of internet and social media, there exists a large amount of information which is being stored in the electronic digital format. Because of the digital media, it has now become easy to collect and analyze such type of data and extract some interesting facts and insights that could help in decision-making.

News information was not easily and quickly available until the beginning of last decade. But now news is easily accessible via content providers such as online news services and social media. A huge amount of information exists in form of text in various diverse areas whose analysis can be beneficial in several areas. Classification of the text data is very challenging as the features are not directly available. The data must be preprocessed to extract the features so that the unstructured data can be converted to structured information. Classifying the news into different categories as different people want to write different kinds of news. A user does not want to read about politics if he is interested in sports. So, the classification of different news is an important task.

Dataset

This dataset contains around 200k news headlines from the year 2012 to 2018 obtained from HuffPost. The model trained on this dataset could be used to identify tags for untracked news articles or to identify the type of language used in different news articles.

The dataset could be found on: https://www.kaggle.com/rmisra/news-category-dataset

The dataset contains the following columns:

1. Category
2. Headline
3. Authors
4. Link of the news
5. Short description

6. date

The dataset contains different categories of news. In total, the dataset contains the following categories:

POLITICS: 32739
WELLNESS: 17827
ENTERTAINMENT: 16058
TRAVEL: 9887
STYLE & BEAUTY: 9649
PARENTING: 8677
HEALTHY LIVING: 6694
QUEER VOICES: 6314
FOOD & DRINK: 6226
BUSINESS: 5937
COMEDY: 5175
SPORTS: 4884
BLACK VOICES: 4528
HOME & LIVING: 4195
PARENTS: 3955
THE WORLDPOST: 3664
WEDDINGS: 3651
WOMEN: 3490
IMPACT: 3459
DIVORCE: 3426
CRIME: 3405
MEDIA: 2815
WEIRD NEWS: 2670
GREEN: 2622
WORLDPOST: 2579
RELIGION: 2556
STYLE: 2254
SCIENCE: 2178
WORLD NEWS: 2177
TASTE: 2096
TECH: 2082
MONEY: 1707
ARTS: 1509
FIFTY: 1401
GOOD NEWS: 1398
ARTS & CULTURE: 1339
ENVIRONMENT: 1323
COLLEGE: 1144
LATINO VOICES: 1129
CULTURE & ARTS: 1030

Below is shown the histogram showing number of news in different categories of the dataset.

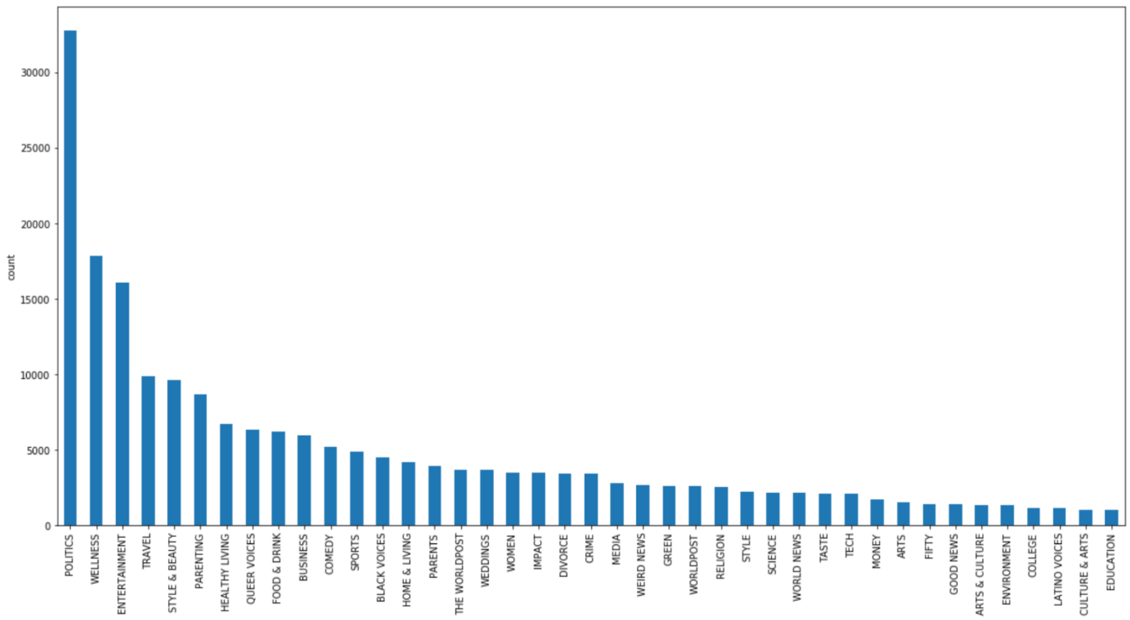


Figure 1: Histogram showing different categories in datset

The pie chart below shows the percentage of every category in the dataset:

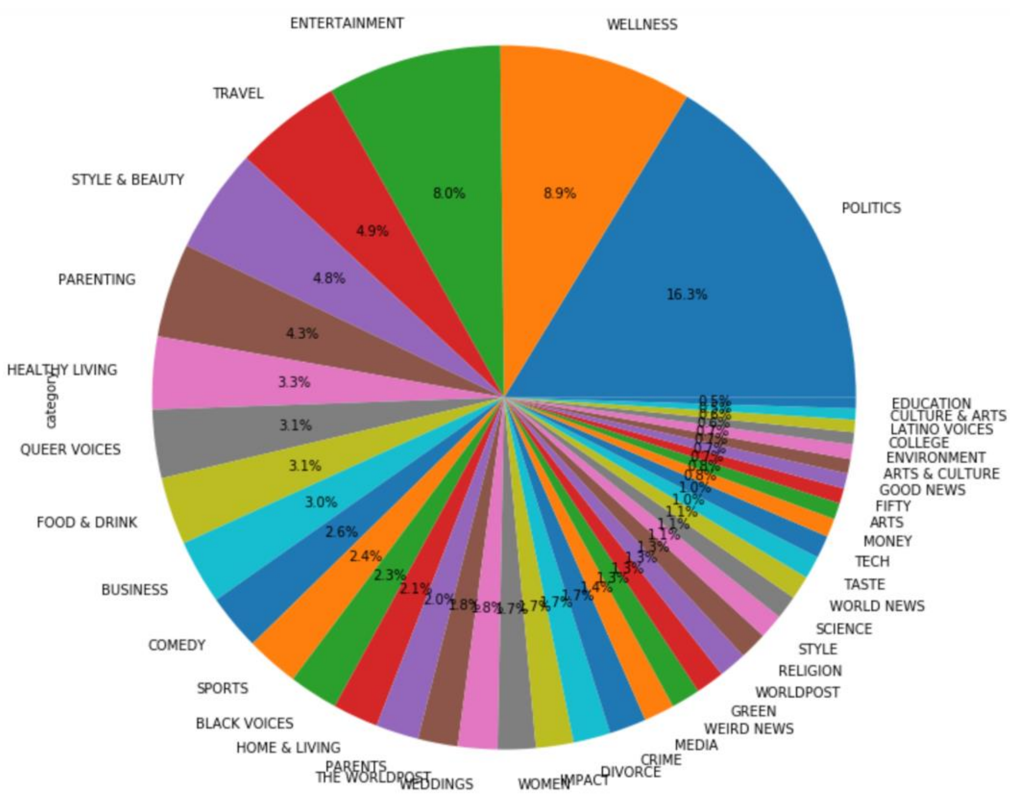


Figure 2: Pi chart showing percentage of samples in different categories in dataset

For this project purpose, I have chosen the top 12 categories as these categories contains maximum number of news. Below is the pie chart which shows the percentage distribution of the categories selected for this project.

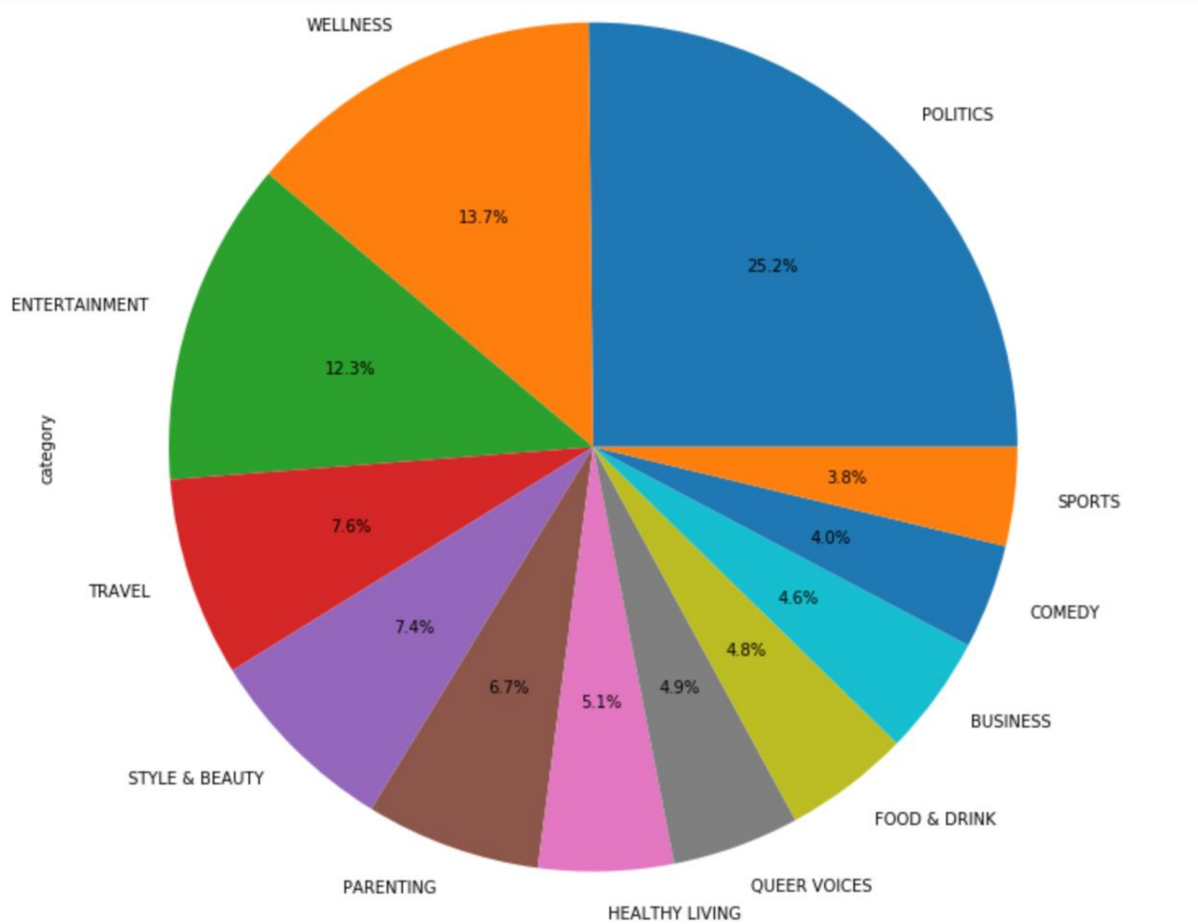


Figure 3: Pi chart showing percentage of samples in different categories used in this project

Preprocessing the data

The news data contains the text that includes the headline of the news and the description of the news. For the analysis of this project, these two columns of the dataset have been combined as the headline and description both help in classifying the news into different categories.

We have the text data of the news and the labels associated with every news. The text data and the labels must be converted into numbers for them to pass to the machine learning models.

Raw data contain numerical value, punctuation, special character etc. These values can hamper the performance of model so before applying any text featurization first we need to convert raw data into meaningful data which is also called as text preprocessing.

Following are the steps taken to preprocess the data:

1. Removal of noisy data:
   In regular sentences Noisy data can be defined as text file header, footer, HTML, XML. As these types of data are not meaningful and does not provide any information this noisy data sjould be removed from the text data. In python HTML, XML can be removed by BeautifulSoup library while markup, header can be removed by using regular expression. This step was not done in this project as the data the news dataset does not contain any kind of noisy data like header, footer, etc. This dataset contains only 2-3 lines of text description of every news.

2. Tokenization:
   In tokenization, we convert group of sentences into token. It is basically splitting whole senetence into small chunk of individual words. Tokenization in python can be done by python's NLTK library's word_tokenize () function.

3. Remove stop words:
   There would be many stop words in the text data like 'me', 'my', 'myself', 'we', 'you', 'he', etc. These words have low predictive power and are unnecessary for text classification. So, these words must be removed from the data. I imported a list of the most frequently used words from the NL Toolkit using the command "from nltk.corpus import stopwords".

4. Stemming & Lemmatizing:
   We must transform some words into their original root form. Stemming cuts off prefixes and/or endings of words based on common ones. Lemmatizing, on the other hand, maps common words into one base. Unlike stemming, it always still returns a proper word. In this project, lemmatization is used for transforming the words to their original root form.

   The previous four steps can be thought of as the normalization of the text data which means to remove unnecessary data and making the text data readable.

5. Vectorizer

   Vectorizer techniques are used to convert the text data into a vector. There were two categories that were adopted for converting the text data into vectors in this project. These two methods are discussed below:

   (a) Tf-IDF vecorizer

In information retrieval, tf–idf or TFIDF, short for term frequency–inverse document frequency, is a method used to indicate how important a word is to a document in a collection. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. Some of the words appear on the document more frequent to others. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word. tf–idf is one of the most popular term-weighting schemes today.

The tf–idf is the product of two terms, term frequency and inverse document frequency. There are several methods to calculate both terms.
For term frequency, the number of times a word appears in a document id divided by the total number of words in the document. Every document has its own term frequency. Term frequency is given by the following equation:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse data frequency is defined as the log of the number of documents divided by the number of documents that contain the word w. Inverse data frequency determines the weight of rare words across all documents in the corpus.

$$idf(w) = log(\frac{N}{df_t})$$

The TF-IDF is simply the TF multiplied by IDF:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

TFIDF can be visualized as a method of feature extraction. I have used TFIDFvectorizer from the Scikit-learn library of feature extraction algorithms.
The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow me to encode new documents.

(b) Doc2Vec :
Doc2vec is a natural language processing tool for representing documents as a vector and is a generalization of the word2vec method. word2vec is used to generate representation vectors out of words. The goal of doc2vec is to create a numeric representation of a

document which can be used to pass to the classifier, regardless of its length. The doc2vec can be represented as below:

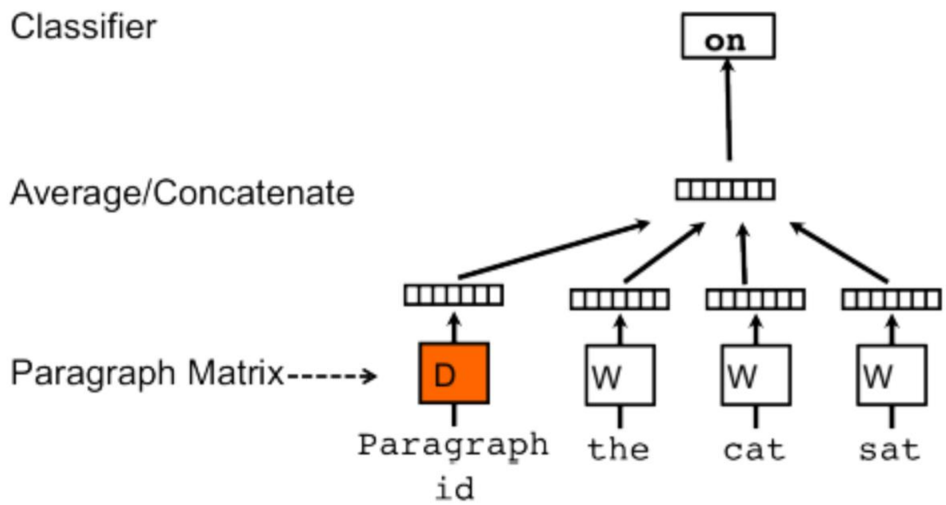


Figure 4: Doc2vec model architecture

In this project, I have used gensim implementation of doc2vec.

Different classifiers used:

Different classifier techniques have been used to classify this news data. Below are techniques used in this project:

1. Naïve bayes classifier:
   Bayes classifier which use the probability from a distribution cannot be applied into the text related problems. When the preprocessing is done, we get a very sparse feature vector as the sentence contains only some words. If this feature vector is used to calculate the probability by assuming some distribution, then the output pf the probability is very small and close to zero. So, the naïve bayes classifier is used.

   Naive Bayes is based on applying Bayes theorem with an assumption that every feature is independent of the others, in order to predict the category of a given sample. They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output.

First, we have to calculate each category class distribution i.e. priors.

$$\pi_j = \frac{class_j}{\sum_{n=1}^{20} class_n}$$

Secondly, for calculating our probability, we will find the average of each word for a given class. The probability of word i given class j is the count that the word occurred in documents of class j, divided by the sum of the counts of each word in our vocabulary in class j refers the above probability. For class j and word i, the average is given by:

$$P(i|j) = \frac{word_{ij}}{word_j}$$

There might be the case that a word doesn't appear in the whole class of the document. Since we are calculating the overall probability of the class by multiplying individual probabilities for each word, we would end up with an overall probability of 0 for that class. We can use a Smoothing Algorithm like Laplace smoothing. We modify our conditional word probability by adding a small constant to the numerator and adding total number of unique words in the class.

$$P(i|j) = \frac{word_{ij} + \alpha}{word_j + |V| + 1}, \ \alpha = 0.001$$

where V is an array of all the words in the vocabulary.
Finally, for class j, word i at a word frequency of f:

$$Pr(j) \propto \pi_j \prod_{i=1}^{|V|} Pr(i|j)^{f_i}$$

We find the class for which this probability is maximum and assign that class to the test sample.

2. Estimating class-conditional PDFs assuming a Multi-Variate Gaussian density function:
The data here is split into the training and testing data. I have used the feature vector of doc2vec preprocessing method here as the TFIDF method is giving zero probability of the

class. Doc2vec gives a 100-dimensional feature vector with non-zero entries from which mean and covariance can be calculated unlike TFIDF method.

The training data has been used for estimating the mean and the covariance of the individual class in the training data. Here, I am taking the case when the data is modeled as a gaussian distribution.

The mean and the covariance have been calculated by using the following equation:

$$\mu_{ML} = \frac{1}{N} \sum_{i} \mathbf{x}_i$$

$$\Sigma_{ML} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \overline{x})(\mathbf{x}_i - \overline{x})^T$$

where x is the feature vector.

This mean and covariance calculation is done for every class.

Now, the probability of every sample in the test data is calculated and the class for which the probability is maximum is taken as the class for test sample.

3. Using non-parametric density estimation:
   I have used the K nearest neighbor classifier for this dataset. The feature vector for from TFIDF preprocessing is used. The distance of every feature vector in test data from every feature vector in training data has been calculated. We then choose the K lowest distance points from the training set for one sample of the test data sample. The most frequent class in these K nearest neighbors list is assigned to the test sample data.

4. Decision tree classifier:
   A Decision Tree is a Supervised Machine Learning algorithm where the data is continuously split according to a certain parameter. Decision Tree consists of Nodes, Edges or branch and leaf nodes. Nodes are used to test the value of a certain attribute. Branches correspond to the outcome of a test and connect this output to the next node or leaf. Leaf nodes are the terminal nodes that predict the outcome. I have used the classifier from the Scikit learn library. Below is a flow chart of the structure of the decision trees.

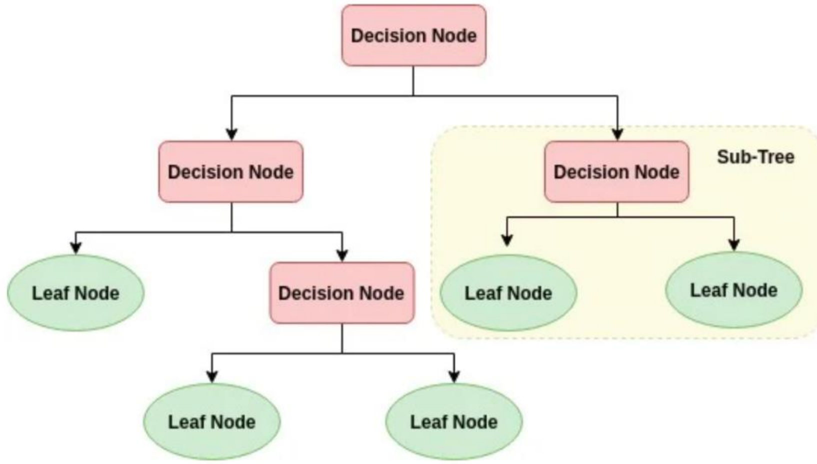


Figure 5: Decision tree architecture

5. Linear support vector classifier:

   SVM or Support Vector Machine is a linear model for classification and regression problems. Support Vector Machine finds a line or a hyperplane which separates the data into classes.

   According to the SVM algorithm, we find the points closest to the line from both the classes. These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin. The goal of Support vector machine is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane that is given as the output for the SVM.

   Thus, SVM tries to make a decision boundary in such a way that the separation between the classes is as much as possible.

6. Bagging classifier:

   A Bagging classifier is an ensemble learning based method that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions to form a final prediction. The predictions are formed using either by voting or averaging. Each base classifier is trained in parallel with a separate independent training set which is generated by randomly drawing, with replacement, N examples from the original training dataset – where N is the size of the original training set.

Bagging reduces overfitting by averaging or voting to form the predictions, however, this leads to an increase in bias, which is compensated by the reduction in variance because of bias-variance tradeoff.

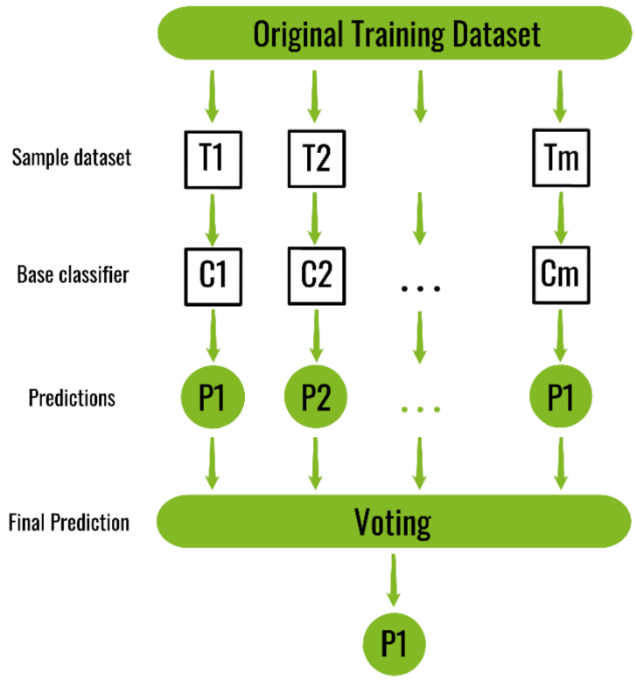


Figure 6: Bagging classifier architecture

7. Multi-class logistic regression:
   Logistic regression is one of the most fundamental and widely used Machine Learning Algorithms. I have used multi-class logistic regression for classifying the dataset. We force the output layer to be a discrete probability distribution over the k classes. To be a valid probability distribution, we will want the output to (i) contain only non-negative values, and (ii) sum to 1. We accomplish this by using the SoftMax function.

$$\operatorname{softmax}(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\sum_{i=1}^{k} e^{z_i}}$$

8. Recurrent neural network model:
   A recurrent neural network is used when the sequence of data is important for classifying the data. This sequence learning is important as sometimes, the model should remember the previous sample outputs as it influences the current output. In our dataset, the text dataset can be passed to the RNN model as a sequential data. The text sentence can be considered as an order of words so one word in the starting has an effect on the word occurring later.

Recurrent means the output at the current time step becomes the input to the next time step. At each element of the sequence, the model considers not just the current input, but what it remembers about the preceding elements.

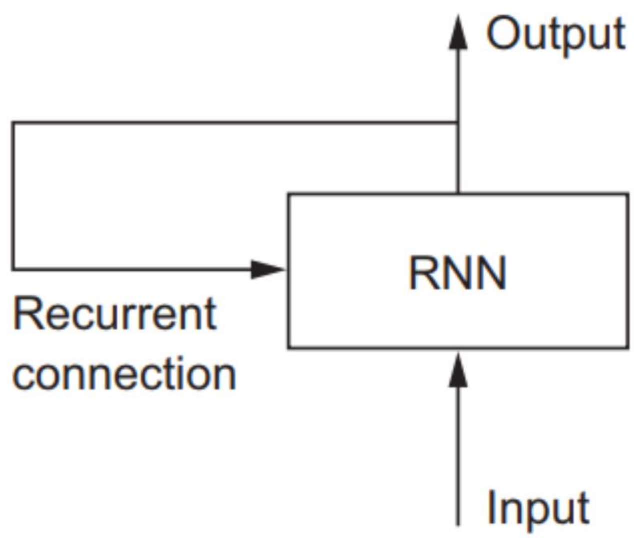


Figure 7: RNN model architecture

The memory part of the model is possible because of the inclusion of a layer of memory cell in the model which is able to store the output of the previous time stamps to be able to use them as the input at the current timestamp.

The most popular used in the RNN model is the Long Short-Term Memory (LSTM). At each time step the LSTM considers the current word, the carry, and the cell state. LSTM maintains a cell state as well as a carry for ensuring that the data (information in the form of a gradient) is not lost as the sequence is processed.

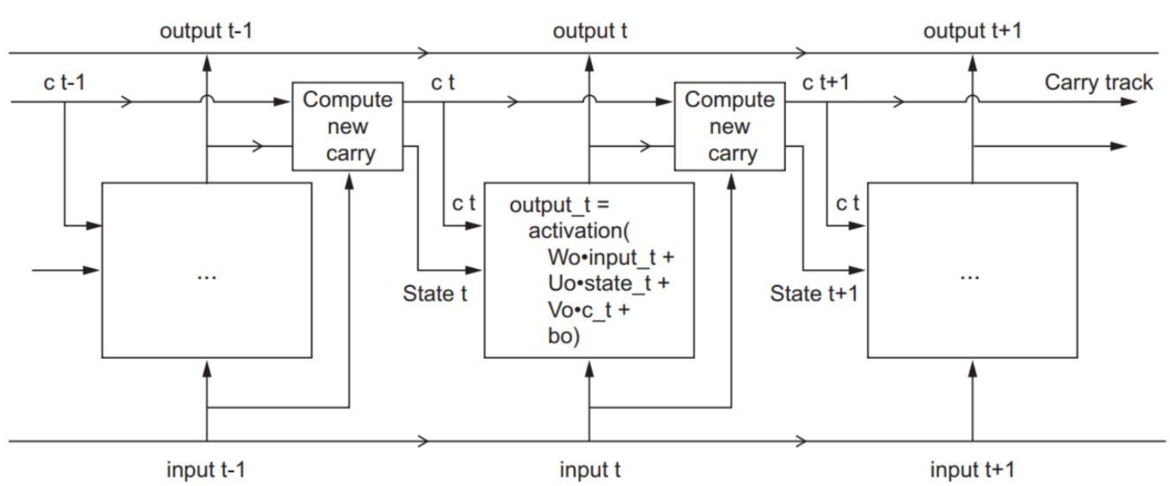


Figure 8: LSTM layer architecture

The embedding matrix has been created using the doc2vector preprocessing. This matrix contains a 100-dimensional feature vector for every sample in the training set. The RNN model used in the project has an LSTM network with 32 hidden layers. It is followed by a fully connected layer and then sigmoid function is used to calculate the probability of every class. The class having highest probability is assigned to that particular test sample.

9. AdaBoost classifier:
   An AdaBoost classifier is an ensemble learning method that begins by fitting a classifier on the original dataset and then fits more copies of the classifier on the same dataset. AdaBoost classifier builds a strong classifier by combining many bad performing classifiers to get high accuracy strong classifier. The weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on these cases and therefore help in increasing accuracy.

Results and discussion:

Below are the results of the different classifiers discussed above:

1. Naïve Bayes classifier:

   The naïve Bayes classifier algorithm is implemented from scratch using some of the code of previous homeworks of the course but most of the code I wrote again as the concept of naïve Bayes is little different from what I implemented in homeorks. The Naïve Bayes algorithm performed very poorly on the news dataset.
   The accuracy of the classifier was 26.27%. The confusion matrix is given below:

```
[[ 297   58  142   41   99  158  455   80   54  108  163  300]
 [  67  201  343   42   67  128  322  109   76  160  111  105]
 [ 161  439 1408   83  130  380  675  429  190  607  355  453]
 [  99   56  154  387  116  175  200   61   22  151  312  331]
 [ 126   68  181   74  305  258  319  108   66  115  182  434]
 [ 156  114  298   70  190  463  352  166   80  188  277  435]
 [ 931  633 1278   55  424  490 4113  888  356  321  544  772]
 [ 101   84  339   12   71  174  414  354   82  100  177  173]
 [  72   72  271   28   47   76  282  101  279   98  105  159]
 [  61   70  519   51   54  281  227  126   37 1129  283  346]
 [ 174   69  310  119  135  265  485  164   65  226  794  436]
 [ 312   98  465  159  694  738  783  244   87  316  498 1542]]
```

The naïve Bayes classifier is not working properly as the output of the feature vector from TFIDF preprocessing is a vary sparse vector. So even after applying Laplace smoothing, the probabilities are very less, and the classifier is not able to classify the samples properly.

2. Estimating class-conditional PDFs assuming a Multi-Variate Gaussian density function: The classifier is implemented from scratch using some of the code of previous homeworks of the course.
   The feature vector from doc3vec was used for this classifier as it was giving nonzero feature values unlike TFIDF which gave a very sparse vector.
   For the split of 2:1 as the train:test ratio, the accuracy in this case was found to be 72.96%.
   The confusion matrix is given below:

```
[[  814     5    53    37    15    56   583     7    10    27    64   290]
 [    7   548   447    29     5    81   375    11    23    32    56   109]
 [   10   143  4160    23     8    79   399    56    28   157    68   140]
 [   11     8    49  1599     2    42    23     1     3    22   136   191]
 [   20     5    69    50   194    82   235     7    11    16    46  1486]
 [   14    11   170    33    27  1711   133    19    13    63    73   541]
 [  108    51   179     8    21    39 10073    53    37    18    67   164]
 [    7     7   310     4     5    87   409  1043    32    34    44   106]
 [    4    21   147     4     6    17   235    16  1048    11    58    58]
 [   26     4   209    23     1    51    48     8     9  2506    57   178]
 [   30     3   103    92     4    57   190     8    11    43  2595   154]
 [   47     3   105   131    75   176   176    13    26    37    96  5026]]
```

   For the split of 4:1 as the train:test ratio, the accuracy in this case was found to be 73.63%.
   The confusion matrix is given below:

```
[[ 512    2   26   18    4   22  357    6   10   10   37  184]
 [   6  353  243   20    1   55  202    5   17   15   28   71]
 [   8   68 2549   16    1   50  227   28   17  110   43  108]
 [   5    2   25 1008    1   13   11    3    2   13   90   95]
 [  13    2   36   36  100   63  141    2    3    7   21  923]
 [  19   10   99   22    9 1064   88    9    7   36   44  296]
 [  86   37  110    5    6   28 6127   28   21    8   37   84]
 [   3    7  161    1    3   46  246  695   13    9   26   81]
 [   2    6  111    4    5   11  138   10  625   10   20   35]
 [  16    5  134   12    1   24   41    4    6 1529   47   88]
 [   9    5   57   45    3   22  100    4   11   28 1569  100]
 [  22    2   69   89   44   95  127    6   13   20   52 3023]]
```

   For the split of 3:2 as the train:test ratio, the accuracy in this case was found to be 72.91%.
   The confusion matrix is given below:

```
[[ 1013     5    55    31    18    58   695     7    18    28    73   379]
 [   11   656   516    37     7   113   440    15    33    36    65   137]
 [   13   169  5050    28    11    97   502    60    41   179    78   177]
 [   17     5    49  1957     2    35    33     4     4    28   191   214]
 [   24     8    69    57   224   109   275    13    12    24    46  1801]
 [   24    16   213    65    22  2055   152    19    18    70    69   694]
 [  153    76   211    14    33    46 12100    62    37    13    96   181]
 [    6    14   340     4    16   104   476  1329    32    37    45   147]
 [    9    16   190     7     2    26   300    18  1237    14    56    85]
 [   37     6   282    27     3    66    79    10    11  3039    84   213]
 [   35    10   121   108     6    60   196    10    18    51  3206   200]
 [   41     8   105   164   116   210   244    15    26    51   100  6068]]
```

3. Using non-parametric density estimation:
   The KNN classifier is implemented from scratch using some of the code of previous homeworks of the course. The feature vectors from TFIDF vectorizer were used for this classifier.
   For KNN, I have taken 20000 samples for training and 10000 samples for testing the classifier. I have tested with different values of K.

   For K=20:
   The accuracy was found to be 56.68%. The confusion matrix is shown below:

```
[[ 105   61   32    6   32    1  166    0    2    0   12   10]
 [    1  217   76    4    9    4   83    2    0    2    5    3]
 [    3  189  879    6   17    7  132    2    2   20    7    2]
 [    1   85   32  288   27    4   23    0    1    3   11   11]
 [    5   69   38    7  182    6   76    1    1    2   12  117]
 [    6  193   71   10   76  213   73    0    1   10   13   39]
 [   18   86   61    3   30    4 2206    7    3    3   23   12]
 [    3   77   80    4   20    6  116  124    2    1    7    6]
 [    1   83   69    1   12    1   92    0  113    1    8    3]
 [    7  105  149    9    9    6   47    0    2  408    3   12]
 [    3   97   67   28   32    4   84    1    2    7  418    4]
 [   17  221   88   27  320   22  161    2    6    5   20  515]]
```

   For K=5:
   The accuracy was found to be 4.81%. The confusion matrix is shown below:

```
[[   0  453    0    0    0    0    0    0    0    0    0    0]
 [   0  385    0    0    0    0    0    0    0    0    0    0]
 [   0 1248    1    0    0    0    0    0    0    0    0    0]
 [   0  465    0    1    0    0    0    0    0    0    0    0]
 [   0  506    0    0    0    0    0    0    0    0    0    0]
 [   0  643    0    0    0    2    0    0    0    0    0    0]
 [   0 2432    0    0    0    0   20    0    0    0    0    0]
 [   0  509    0    0    0    0    0    3    0    0    0    0]
 [   0  373    0    0    0    0    0    0    0    0    0    0]
 [   0  739    0    0    0    0    0    0    0   45    0    0]
 [   0  772    0    0    0    0    0    0    0    0    4    0]
 [   0 1379    0    0    0    0    0    0    0    0    0   20]]
```

Then, For K=20, I increased my training set samples to be 40000. For this experiment, I got very low accuracy. The extra 20000 samples were maybe of class 2 as most of the samples were categorized as class 2.

The accuracy in this case was found to be 5.18%. The confusion matrix is given below:

```
[[   1  426    0    0    0    0    0    0    0    0    0    0]
 [   0  406    0    0    0    0    0    0    0    0    0    0]
 [   0 1262    4    0    0    0    0    0    0    0    0    0]
 [   0  483    0    3    0    0    0    0    0    0    0    0]
 [   0  514    0    0    1    0    0    0    0    0    0    1]
 [   0  696    0    0    0    9    0    0    0    0    0    0]
 [   0 2432    0    0    0    0   24    0    0    0    0    0]
 [   0  443    0    0    0    0    0    3    0    0    0    0]
 [   0  383    0    0    0    0    1    0    0    0    0    0]
 [   0  720    1    0    0    0    0    0    0   36    0    0]
 [   0  743    0    0    0    0    0    0    0    0    4    0]
 [   0 1377    0    0    0    0    0    0    0    0    0   27]]
```

4. Decision tree classifier:

   The feature vectors from TFIDF vectorizer were used for this classifier.

   For the split of 2:1 as the train:test ratio, the accuracy was found to be 59.85%. The confusion matrix is given by:

```
[[ 710   59  111   59  106   50  394   40   38   51  103  274]
 [  44  602  270   43   66   60  301   30   59   55   54   85]
 [  85  282 3118   82  134  201  495  130  165  137  167  301]
 [  47   53  106 1199   63   61   72   16   18   70  139  203]
 [  91   95  151   74  509  102  213   37   46   32   81  793]
 [  51   52  200   54   84 1612  144   54   48   93  102  320]
 [ 306  253  369   54  186  143 8524  177  119   41  233  368]
 [  26   37  183   13   43   84  200 1240   43   38   53  112]
 [  33   75  214   27   41   46  195   40  776   30   79   92]
 [  43   57  197   73   36   88   74   20   41 2183  205  146]
 [  92   51  167  153   71  110  209   34   51  156 1935  270]
 [ 208   95  260  209  636  314  356   64   85  143  271 3281]]
```

For the split of 4:1 as the train:test ratio, the accuracy was found to be 60.67%. The confusion matrix is given below:

```
[[ 414   37   84   23   65   20  248   24   39   17   60  157]
 [  33  364  178   28   38   43  162   15   33   34   34   54]
 [  47  181 1978   33   87  129  287   63   86   78   87  169]
 [  28   27   87  721   36   41   37   11   14   49   91  126]
 [  60   58   99   49  311   68  142   20   27   27   53  433]
 [  30   30  120   25   61  986   96   37   30   45   56  187]
 [ 197  146  242   31   97   73 5225   97   72   36  111  250]
 [  21   37  128    5   23   36  124  786   11   20   35   65]
 [  25   42  129   10   26   34  110   30  457   25   33   56]
 [  28   30  111   35   25   50   43   16   24 1330  105  110]
 [  57   33  104   87   40   58  112   19   36  104 1133  170]
 [ 129   63  167  115  337  183  186   37   51   86  129 2079]]
```

For the split of 3:2 as the train:test ratio, the accuracy was found to be 59.05%. The confusion matrix is given below:

```
[[   847     66    162     46    122     77    437     30     66     68    111    348]
 [    54    690    405     62     78     64    358     38     68     59     86    104]
 [   111    329   3701    108    174    290    645    123    168    174    211    371]
 [    57     62    164   1432     83     58     90     19     26    120    176    252]
 [    96    114    195     84    634    139    251     42     38     41     98    930]
 [    55     59    243     88    115   1965    156     49     36    123    112    416]
 [   388    338    529     79    212    201  10151    201    155     72    219    477]
 [    46     44    266     30     53    101    252   1478     42     36     62    140]
 [    58     64    259     31     50     63    220     36    893     55     94    137]
 [    57     53    234     92     58    109    101     24     41   2601    254    233]
 [   112     86    224    177     85    122    264     39     67    193   2278    374]
 [   241     99    333    250    747    346    452     75     98    174    279   4054]]
```

5. Linear support vector classifier:
   The feature vectors from TFIDF vectorizer were used for this classifier.
   The confusion matrix is given below:

```
[[1100   15   60   38   64   48  349   18   24   36   53  156]
 [  25  766  330   40   40   66  255   20   32   39   41   69]
 [  34  211 4101   28   41  120  295   77   65  120   71  108]
 [  25   22   39 1660   34   41   19    6   11   33   85  112]
 [  69   26   84   62  545  102  176   16   21   16   42 1062]
 [  29   23  102   43   60 2009   95   18   23   59   58  289]
 [ 183   74  169   22   82   67 9808   97   61   31   94  130]
 [  10   14  168    9   11   72  205 1448   32   25   26   68]
 [  12   29   98    8   21   24  123   20 1188    9   53   40]
 [  24   12  128   13   12   44   29   14   13 2685   57   89]
 [  40   20   93  108   22   54  104   16   18   71 2645   99]
 [ 114   35  107  114  339  213  123   29   47   65   96 4629]]
```

For the split of 4:1 as the train:test ratio, the accuracy was found to be 76.86%. The confusion matrix is given below:

```
[[ 666   14   28   19   23   21  208   14   14   14   26  107]
 [  20  479  177   19   26   36  151   11   22   22   28   40]
 [  20  111 2512   22   21   65  167   43   42   59   49   62]
 [  12   16   25  985   18   16   16    3    3   14   55   64]
 [  33   14   35   34  325   70   93    8   17   13   19  643]
 [  22   22   53   29   30 1266   59   14   16   28   33  168]
 [ 123   43  116   14   41   34 6063   66   31   10   47   82]
 [   6   11  107    2    4   58  118  888   15   11   10   27]
 [   4   11   67    3   13   20   61   14  757    6   15   32]
 [  23   13   71   12   10   26   21   10    5 1667   30   46]
 [  21   10   57   65   14   25   59    7   21   45 1582   52]
 [  75   15   53   66  190  127   82   20   31   35   64 2805]]
```

For the split of 3:2 as the train:test ratio, the accuracy was found to be 75.92%. The confusion matrix is given below:

```
[[ 1349    17    77    40    93    53   414    10    24    36    68   199]
 [   41   926   373    48    51    72   299    25    48    40    61    82]
 [   35   272  4965    40    49   133   387    90    89   129    84   132]
 [   29    17    53  2044    31    38    24     3    12    25   127   136]
 [   71    35    89    59   695   128   202    24    23    20    46  1270]
 [   43    42   135    65    64  2450   101    28    25    68    48   348]
 [  259   110   237    24    81    89 11733   127    67    20   118   157]
 [   13    18   196     9    22    92   241  1782    28    36    36    77]
 [   16    29   120     7    15    32   161    21  1438    15    44    62]
 [   39    17   151    26    19    58    50    17    15  3286    65   114]
 [   45    17   115   126    31    49   128    20    44    64  3246   136]
 [  114    30   120   151   415   254   181    40    55    80   123  5585]]
```

6. Bagging classifier:

The feature vectors from TFIDF vectorizer were used for this classifier.
For the split of 2:1 as the train:test ratio, the accuracy was found to be 65.58%. The confusion matrix is given below:

```
[[ 809   39  125   44   79   38  403    7   32   34   77  308]
 [  46  623  315   29   57   58  312   12   44   54   35   84]
 [  76  203 3501   53   83  228  503   46   86  114  130  274]
 [  33   37  112 1318   31   52   53    4   17   95   86  209]
 [  71   62  180   68  463  116  226    9   26   26   61  916]
 [  31   37  145   26   41 1965  121   13   18   75   47  295]
 [ 249  101  337   33  126  136 9067  117   87   28  154  338]
 [  33   21  230   12   34   98  201 1250   26   34   29  104]
 [  31   60  250   15   42   46  184   11  834   34   52   89]
 [  48   37  201   39   20   78   59    8   27 2366  110  170]
 [  65   53  185  130   34  103  200    8   37  195 2039  250]
 [ 148   65  255  170  442  264  299   17   59  113  172 3918]]
```

7. Logistic regression:

The feature vectors from TFIDF vectorizer were used for this classifier.
For the split of 2:1 as the train:test ratio, the accuracy was found to be 76.26%. The confusion matrix is given below:

```
[[ 1023     6    85    29    24    35   440     7    15    30    55   212]
 [   16   679   395    22    15    65   316    16    26    45    38    90]
 [   23   156  4235    15    11   121   369    40    37    97    53   114]
 [   20    13    72  1622     9    38    34     1     8    42    78   150]
 [   42     8   113    50   313    97   210    11    16    14    40  1307]
 [   22    14   101    31    25  2045   117    10    18    48    51   326]
 [  116    33   152    17    27    69 10058    72    40    20    74   140]
 [    6     7   218     4     4    80   242  1346    25    29    25   102]
 [    5    14   163     5     9    27   182    15  1097    14    43    51]
 [   25    12   159    15     8    33    44     8    13  2639    41   123]
 [   27     2   122    87     3    47   131     5    11    62  2672   121]
 [   65     9   113   103    87   173   179    11    41    44    80  5006]]
```

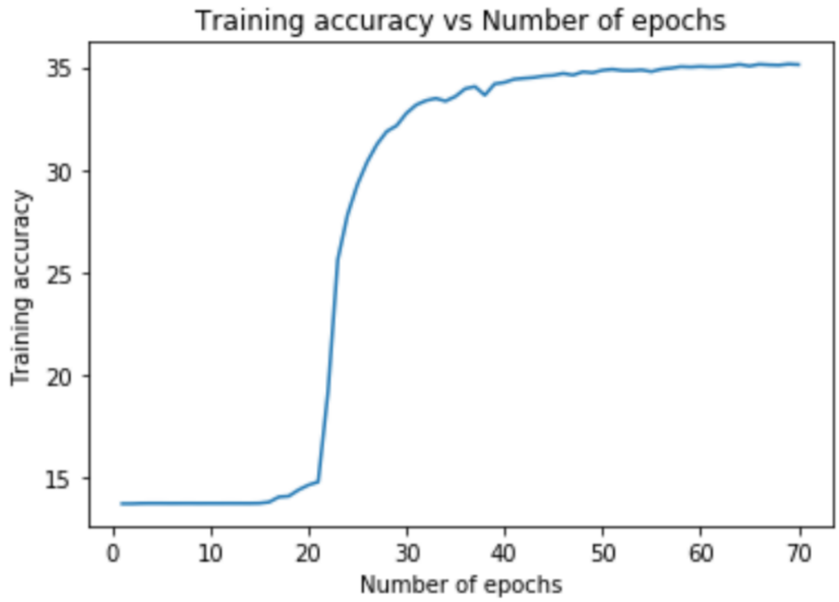
For the split of 4:1 as the train:test ratio, the accuracy was found to be 77.09%. The confusion matrix is given below:

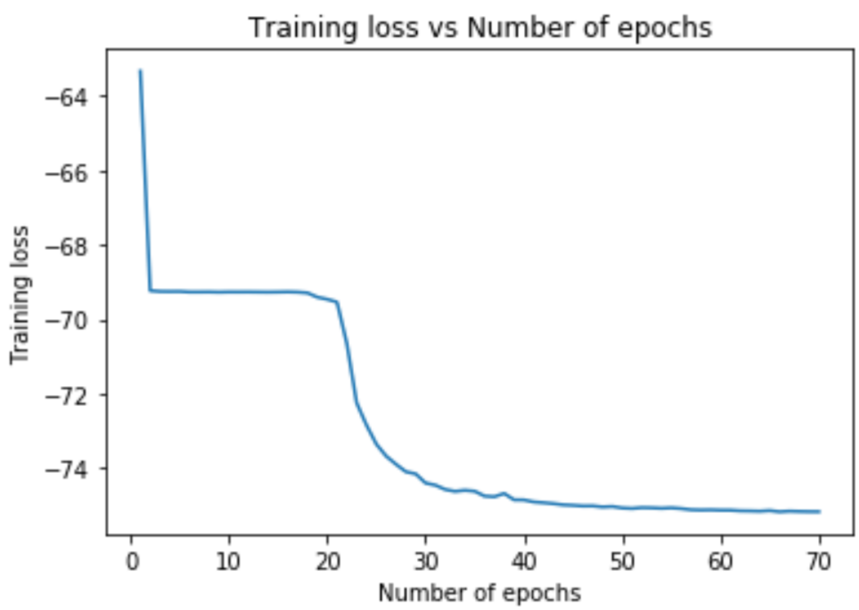
```
[[ 610    6   38   18   11   19  262    7   10   12   27  134]
 [  17  422  221   18    9   33  194    3   17   19   23   55]
 [   7   90 2587   10    7   62  219   22   24   55   30   60]
 [  10    4   29  983    6   15   22    0    3   18   50   87]
 [  25    3   38   29  208   65  100    7   11    8   21  789]
 [  20   10   66   23   14 1259   69   11   10   28   33  197]
 [  80   22   87   10   18   37 6211   51   22    8   38   86]
 [   8    9  127    2    5   55  147  831   13   19    8   33]
 [   2    8   90    3    6   23  109    8  700    9   13   32]
 [  22    8   98    8    3   18   32    5    8 1638   29   65]
 [  11    5   57   56    7   17   67    3   15   36 1614   70]
 [  45    6   58   57   60  110  105   13   21   29   67 2992]]
```

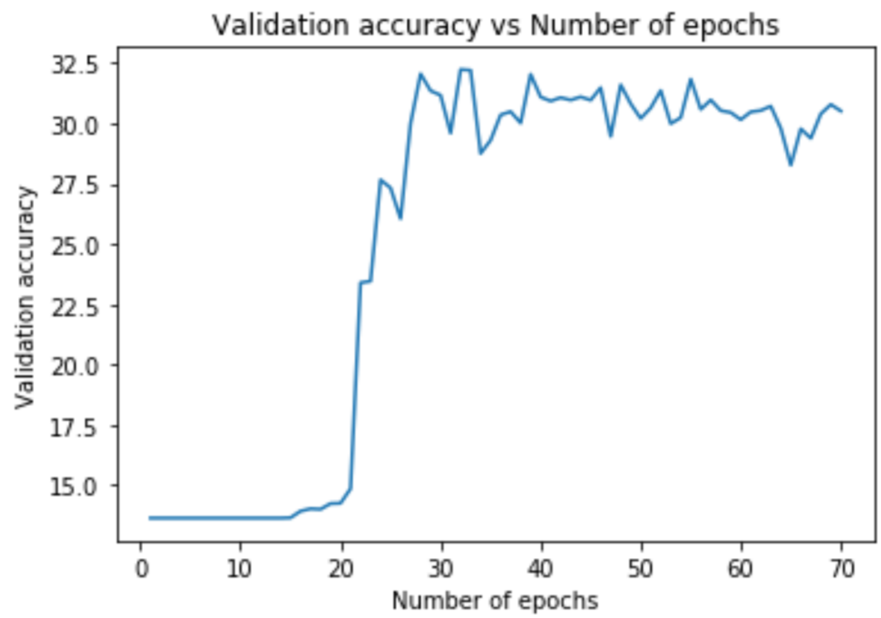
For the split of 3:2 as the train:test ratio, the accuracy was found to be 75.89%. The confusion matrix is given below:

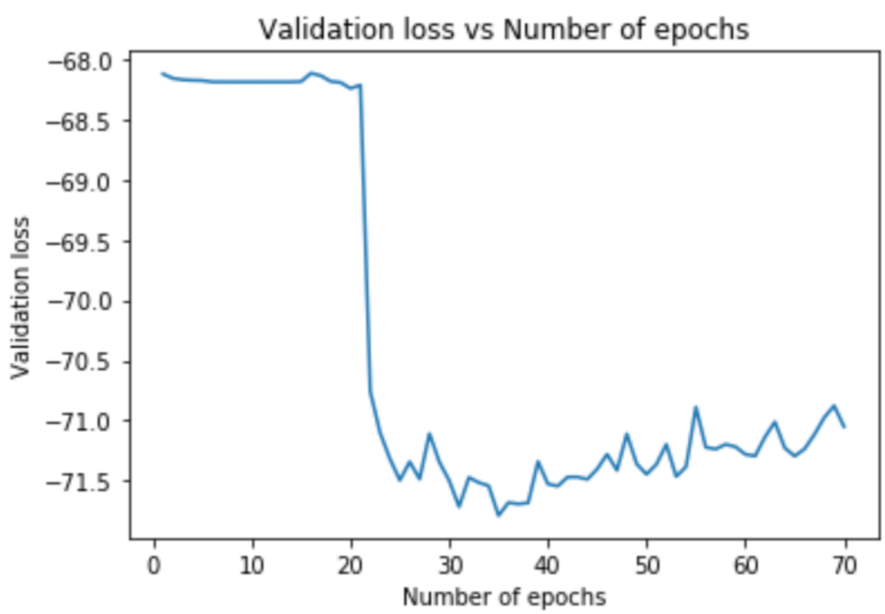
```
[[ 1234     3    99    32    39    41   528     9    19    35    59   282]
 [   22   784   465    24    15    71   418    14    39    42    51   121]
 [   21   182  5110    23    11   128   520    43    39   112    63   153]
 [   24    11    71  1976    16    33    40     0    12    44   122   190]
 [   35    10   110    55   414   123   238    11    18    22    46  1580]
 [   21    19   139    44    26  2451   121    11    18    68    46   453]
 [  157    53   211    18    33    94 12044   100    37    11    88   176]
 [    9     9   278     7    12   101   291  1634    25    37    31   116]
 [    7    17   193     8     6    36   221    13  1328    19    37    75]
 [   37    16   200    15     6    46    83     7    13  3219    50   165]
 [   33     7   158   118     8    46   148     8    21    65  3253   156]
 [   62     8   121   137   128   197   220    15    50    65   109  6036]]
```

8. RNN model:
   The feature vectors from doc2vec vectorizer were used for this classifier.
   The RNN model was trained for 70 epochs. The following results were obtained for the split of 7:3 as the train:test ratio:


Training accuracy vs Number of epochs
Training accuracy
Number of epochs
15
20
25
30
35
0
10
20
30
40
50
60
70


Training loss vs Number of epochs
Training loss
Number of epochs
−64
−66
−68
−70
−72
−74
0
10
20
30
40
50
60
70





The best training accuracy achieved was 35.15% and best validation accuracy was found to be 30.78%.

9. AdaBoost classifier:
   The feature vectors from TFIDF vectorizer were used for this classifier.
   For the split of 2:1 as the train:test ratio, the accuracy was found to be 39.26%. The confusion matrix is given below:

```
[[  200     1     0     1     0    49  1712     7     0    24     6    35]
 [    8     3     0     1     0    50  1553    11     0    59     2    10]
 [   24     3     1     0     0   305  4797    26     0   157     5     7]
 [    5     0     0   504     0    36  1116     0     0   151     5   234]
 [    9     0     0    11     0   117  1955     6     0     7    10    66]
 [    8     0     0    16     0  1860   857     5     0    70     7    22]
 [   59    14     0     1     0   192 10331   153     0    41    30    29]
 [    2     1     0     0     0    64   776  1192     0    21     2     4]
 [    5     0     0     0     0    46  1499    11     0    42     0     2]
 [    7     1     0     5     0    90  1014     2     0  2048     6     6]
 [   15     0     0    20     0    93  2036     5     0   564   428    54]
 [   33     0     0    37     0   266  5162     6     0    78    11   285]]
```

For the split of 4:1 as the train:test ratio, the accuracy was found to be 39.33%. The confusion matrix is given below:

```
[[   0    1    0    4    0   18 1121    5    2   13    2   22]
 [   0    4    1    1    0   41  908    7    3   45    2    4]
 [   0    0    1    2    0  184 2934   15    0   82    5    2]
 [   0    0    0  306    0   27  698    0    0   92    3  142]
 [   0    0    0    3    0   69 1219    4    0    2    3   47]
 [   0    0    0    6    0 1096  537    5    0   45    5    9]
 [   0    4    1    1    0  133 6276   94    6   25   15   22]
 [   0    0    0    0    0   38  479  750    5   17    2    0]
 [   0    0    0    0    0   33  826   10   85   20    1    2]
 [   0    0    0    4    0   46  589    3    1 1259    3    2]
 [   0    1    0    9    0   59 1251    3    1  311  275   43]
 [   0    0    0   23    0  158 3134    4    3   47   13  180]]
```

For the split of 3:2 as the train:test ratio, the accuracy was found to be 39.26%. The confusion matrix is given below:

```
[[    0     0     0     2     0    60  2241     7     5    30     5    30]
 [    0     6     1     2     0    61  1888    10     7    77     3    11]
 [    0     2     1     3     0   379  5808    29     1   166     5    11]
 [    0     0     0   636     0    38  1393     0     0   183    10   279]
 [    0     0     0    10     0   151  2392     6     3    13    19    68]
 [    0     0     0    22     0  2236  1038     6     0    87     6    22]
 [    0    12     1     2     0   255 12446   172    10    52    29    43]
 [    0     2     0     0     0   100   945  1468     3    28     1     3]
 [    0     0     0     0     0    57  1658    12   179    47     2     5]
 [    0     2     0     7     0    98  1200     6     1  2531     6     6]
 [    0     1     0    18     0   126  2546    12     2   671   563    82]
 [    0     0     0    42     0   310  6285    11     7   108    27   358]]
```

Summary of the results reported above is given below:

Table 1: Summary of all classifiers

Classifier	Train:test split ratio	Accuracy (%)
Naïve bayes	2:1	26.27
Estimating class-conditional PDFs	2:1	72.96
	4:1	73.63
	3:2	72.91
Using non-parametric density function	2:1	K=20: 56.68
		K=5: 4.81
	4:1	K=20: 5.18
Bagging classifier	2:1	65.58
AdaBoost classifier	2:1	39.26
	4:1	39.33
	3:2	39.26
Decision tree classifier	2:1	59.85
	4:1	60.67
	3:2	59.05
Linear support vector classifier	2:1	75.91
	4:1	76.86
	3:2	75.92
Logistic regression classifier	2:1	76.26
	4:1	77.09
	3:2	75.89
Recurrent neural network model	7:3	Train=35.15
		Valid=30.78%

As we can see from the above table, different models have been trained on different train:test split ratio. The different train:test split ratio used were 2:1, 4:1 and 3:2. I tried 8 different classifiers on this dataset. First three classifiers i.e. Naïve Bayes, estimating class conditional PDFs and KNN classifier were implemented from scratch using some of the code implemented in previous homeworks of the course. Still the code of naïve bayes classifier was written again as its concept is little different from what was implemented in the homeworks.

Some of the classifiers performed very poorly on the dataset. Naïve bayes and recurrent neural network model performed very poorly on the dataset. Naïve bayes achieved an accuracy of 26.27% while the RNN model achieved a validation accuracy of 30.78%. However, for Naïve bayes, the number of features extracted by using the TFIDF vectorizer were 10000. There were memory issues if I extracted more features. So, I believe that the naïve bayes algorithm if tested with more features can give comparable results to other

classifiers. For RNN classifier, the model was trained for 70 epochs. The model can give better accuracy if trained for number of epochs.
For KNN classifier, we can see that the high vale of K is able to better classify the data. For K=5, 2nd class which is the “COMEDY” category was the output in most of the samples. But as the value of K is increased to 20, the model was able to classify the data much better than the case of K=5. But still the accuracy was very low i.e. 56.68%. Now the number of training examples were increased, but it resulted in a very low accuracy of 5.18% with most of the samples getting misclassified as class 2 which is “COMEDY”.

The Bayesian classifier when the class condition PDFs were estimated performed good on the dataset when compared to other classifiers. The accuracy was also approximately constant when the train:test split ratio was changed. It was seen that the examples were mostly misclassified as class 3 and 8 which were “ENTRTAINMENT” and “WELLNESS”. This might be because these three categories have higher percentage of news samples in the dataset.

The AdaBoost classifier also performed very poorly on the dataset. For all the train:test split ratios, it gave approximately constant but very poor accuracy. It was seen that most if the samples were getting misclassified as classes 6 and 7 which were “PARENTING” and “POLITICS”.

Remaining four classifiers: Decision tree, linear support vector, bagging and logistic regression were implemented the functions from the Scikit learn library. All of them performed well on the dataset. The bagging classifier and decision tree classifier had the lowest accuracy among all of them with an accuracy of around 60%. Linear SVC and logistic regression performed very good on the dataset. It was seen that the examples were mostly misclassified as class 3, 7 and 8 which were “ENTRTAINMENT”, “POLITICS” and “WELLNESS”. This might be because these three categories have higher percentage of news samples in the dataset. Also, the accuracy was approximately constant when the train:test split ratio was changed.

KNN, Naïve Bayes and RNN had the worst performance on the data. So, these classifiers are not useful for classifying this data and are unstable for this dataset.

The mean and variance of the accuracy was also analysed for the classifiers tested on multiple train:test split ratios. These are given below:

Table 2: Mean and variance of accuracy of classifiers on different train:test split

Classifier	Mean accuracy	Variance of accuracy
Estimating class-conditional PDFs	73.16	0.1077
AdaBoost	39.28	0.0011
Decision tree classifier	59.85	0.4374
Linear support vector classifier	76.23	0.1984
Logistic regression classifier	76.41	0.2517

The variance of the decision tree classifier is highest among all classifiers. Its mean accuracy also is lowest. So, decision tree classifier is not a good and stable classifier. For this dataset, estimating class-conditional PDFs classifier has the lowest variance and a good accuracy also on the data.

Summary and Conclusion:

The news category classification was done in the dataset by using the news data available on Kaggle. The dataset initially had 41 categories out of which only 12 categories were taken for this project. The dataset first required preprocessing before passing the data to the classifier. Tokenization, lemmatization and vectorization was done on the text data to convert them into vectors. Two vectorizers were used in this dataset: TFIDF and doc2vec. Tokenization and lemmatization can be thought of as normalizing of the text data

The preprocessed data was then passed on to different classifiers. In total, nine classifiers were tested on the dataset and their performance on the dataset was observed. Naïve bayes, KNN, RNN and AdaBoost performed very poorly on the dataset. Bayesian classifier by estimating conditional PDFs, Linear SVC and logistic regression classifier performed very well on the dataset. In most of the cases, it was seen that the examples were mostly misclassified as class 3, 6, 7 and 8 which were “ENTRTAINMENT”, “PARENTING”, “POLITICS” and “WELLNESS”. This might be because these four categories have higher percentage of news samples in the dataset.

References:

1. Pattern Classification book by Richard O Duda and Peter E Hart and David G Stork
2. Pattern recognition and analysis lecture slides
3. https://www.kaggle.com/
4. https://www.kaggle.com/rmisra/news-category-dataset
5. https://rishabhmisra.github.io/publications/
6. https://towardsdatascience.com/
7. https://medium.com/
8. https://stackoverflow.com/
9. https://web.stanford.edu/~jurafsky/slp3/slides/7_NB.pdf

10. https://scikit-learn.org/

Pi chart

In [ ]:

```python

import re
import pandas as pd # CSV file I/O (pd.read_csv)
from nltk.corpus import stopwords
import numpy as np
import sklearn
import nltk
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score ,confusion_matrix

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
news = pd.read_excel('filtered_data.xlsx')
#remove_columns_list = ['authors', 'date', 'link', 'short_description', 'headline']
#news['information'] = news[['headline', 'short_description']].apply(lambda x: ' '.jc
# Dataset dimension(row, columns)
# To display entire text
pd.set_option('display.max_colwidth', -1)

fig, ax = plt.subplots(1, 1, figsize=(12,12))
news['category'].value_counts().plot.pie( autopct = '%1.1f%%')
```

In [ ]:

```python
fig, ax = plt.subplots(1, 1, figsize=(12,12))
news['category'].value_counts().plot.pie( autopct = '%1.1f%%')
```

Histogram

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn
import warnings
warnings.filterwarnings('ignore')

data = pd.read_json('News_Category_Dataset_v2.json', lines=True)
print(data)

%matplotlib inline
data1 = data[["category", "text"]]
print(data1)
data1.category.value_counts().plot.bar(figsize = (20,10))
plt.ylabel("count")
```

Gaussian estimate conditional PDFs

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn
import warnings
from sklearn import model_selection
import gensim


fil_data=pd.read_excel('filtered_data.xlsx')
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(fil_data[['text']
                                                                                fil_data[
print(X_train.shape)
```

In [ ]:

```
from gensim.test.utils import common_texts
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(common_texts)]
model = Doc2Vec(documents, vector_size=100, window=1, min_count=1, workers=4,dm=1)
```

In [ ]:

```
from gensim.test.utils import get_tmpfile

fname = get_tmpfile("my_doc2vec_model")

model.save(fname)
model = Doc2Vec.load(fname)  # you can continue training with the loaded model!
```

In [ ]:

```
X_train_no=np.zeros((87144,500))

for i in range(87144):
    X_train_no[i,:]= model.infer_vector(np.asarray(X_train)[i])
```

In [ ]:

```
Y_train = np.array(Y_train);
Y_test = np.array(Y_test);
```

In [ ]:

```
Y_trainno=np.zeros((Y_train.size))
for i in range(Y_train.size):
    if Y_train[i]=="BUSINESS":
        Y_trainno[i]=1
    elif Y_train[i]=="COMEDY":
        Y_trainno[i]=2
    elif Y_train[i]=="ENTERTAINMENT":
        Y_trainno[i]=3
    elif Y_train[i]=="FOOD & DRINK":
        Y_trainno[i]=4
    elif Y_train[i]=="HEALTHY LIVING":
        Y_trainno[i]=5
    elif Y_train[i]=="PARENTING":
        Y_trainno[i]=6
    elif Y_train[i]=="POLITICS":
        Y_trainno[i]=7
    elif Y_train[i]=="QUEER VOICES":
        Y_trainno[i]=8
    elif Y_train[i]=="SPORTS":
        Y_trainno[i]=9
    elif Y_train[i]=="STYLE & BEAUTY":
        Y_trainno[i]=10
    elif Y_train[i]=="TRAVEL":
        Y_trainno[i]=11
    elif Y_train[i]=="WELLNESS":
        Y_trainno[i]=12

```

In [ ]:

```
x1_train=[]
x2_train=[]
x3_train=[]
x4_train=[]
x5_train=[]
x6_train=[]
x7_train=[]
x8_train=[]
x9_train=[]
x10_train=[]
x11_train=[]
x12_train=[]
x1_test=[]
x2_test=[]
x3_test=[]
x4_test=[]
x5_test=[]
x6_test=[]
x7_test=[]
x8_test=[]
x9_test=[]
x10_test=[]
x11_test=[]
x12_test=[]
y1_train=[]
y2_train=[]
y3_train=[]
y4_train=[]
y5_train=[]
y6_train=[]
y7_train=[]
y8_train=[]
y9_train=[]
y10_train=[]
y11_train=[]
y12_train=[]
y1_test=[]
y2_test=[]
y3_test=[]
y4_test=[]
y5_test=[]
y6_test=[]
y7_test=[]
y8_test=[]
y9_test=[]
y10_test=[]
y11_test=[]
y12_test=[]

```

In [ ]:

```
for i in range(Y_train.size):
    if Y_train[i]=="BUSINESS":
        x1_train.append(X_train_no[i,:])
        y1_train.append(Y_trainno[i])
    elif Y_train[i]=="COMEDY":
        x2_train.append(X_train_no[i,:])
        y2_train.append(Y_trainno[i])
    elif Y_train[i]=="ENTERTAINMENT":
        x3_train.append(X_train_no[i,:])
        y3_train.append(Y_trainno[i])
    elif Y_train[i]=="FOOD & DRINK":
        x4_train.append(X_train_no[i,:])
        y4_train.append(Y_trainno[i])
    elif Y_train[i]=="HEALTHY LIVING":
        x5_train.append(X_train_no[i,:])
        y5_train.append(Y_trainno[i])
    elif Y_train[i]=="PARENTING":
        x6_train.append(X_train_no[i,:])
        y6_train.append(Y_trainno[i])
    elif Y_train[i]=="POLITICS":
        x7_train.append(X_train_no[i,:])
        y7_train.append(Y_trainno[i])
    elif Y_train[i]=="QUEER VOICES":
        x8_train.append(X_train_no[i,:])
        y8_train.append(Y_trainno[i])
    elif Y_train[i]=="SPORTS":
        x9_train.append(X_train_no[i,:])
        y9_train.append(Y_trainno[i])
    elif Y_train[i]=="STYLE & BEAUTY":
        x10_train.append(X_train_no[i,:])
        y10_train.append(Y_trainno[i])
    elif Y_train[i]=="TRAVEL":
        x11_train.append(X_train_no[i,:])
        y11_train.append(Y_trainno[i])
    elif Y_train[i]=="WELLNESS":
        x12_train.append(X_train_no[i,:])
        y12_train.append(Y_trainno[i])
```

In [ ]:

```
Y_testno=np.zeros((Y_test.size))
for i in range(Y_test.size):
    if Y_test[i]=="BUSINESS":
        Y_testno[i]=1
    elif Y_test[i]=="COMEDY":
        Y_testno[i]=2
    elif Y_test[i]=="ENTERTAINMENT":
        Y_testno[i]=3
    elif Y_test[i]=="FOOD & DRINK":
        Y_testno[i]=4
    elif Y_test[i]=="HEALTHY LIVING":
        Y_testno[i]=5
    elif Y_test[i]=="PARENTING":
        Y_testno[i]=6
    elif Y_test[i]=="POLITICS":
        Y_testno[i]=7
    elif Y_test[i]=="QUEER VOICES":
        Y_testno[i]=8
    elif Y_test[i]=="SPORTS":
        Y_testno[i]=9
    elif Y_test[i]=="STYLE & BEAUTY":
        Y_testno[i]=10
    elif Y_test[i]=="TRAVEL":
        Y_testno[i]=11
    elif Y_test[i]=="WELLNESS":
        Y_testno[i]=12

```

In [ ]:

```
total=len(x1_train)+len(x2_train)+len(x3_train)+len(x4_train)+len(x5_train)+len(x6_tr
print(total)
total=len(y1_train)+len(y2_train)+len(y3_train)+len(y4_train)+len(y5_train)+len(y6_tr
print(total)
```

In [ ]:

```

mu1_es=(1/len(x1_train))*(np.sum(x1_train,axis=0));
mu2_es=(1/len(x2_train))*(np.sum(x2_train,axis=0));
mu3_es=(1/len(x3_train))*(np.sum(x3_train,axis=0));
mu4_es=(1/len(x4_train))*(np.sum(x4_train,axis=0));
mu5_es=(1/len(x5_train))*(np.sum(x5_train,axis=0));
mu6_es=(1/len(x6_train))*(np.sum(x6_train,axis=0));
mu7_es=(1/len(x7_train))*(np.sum(x7_train,axis=0));
mu8_es=(1/len(x8_train))*(np.sum(x8_train,axis=0));
mu9_es=(1/len(x9_train))*(np.sum(x9_train,axis=0));
mu10_es=(1/len(x10_train))*(np.sum(x10_train,axis=0));
mu11_es=(1/len(x11_train))*(np.sum(x11_train,axis=0));
mu12_es=(1/len(x12_train))*(np.sum(x12_train,axis=0));

#Estimated variance
cov1_es=(1/len(x1_train))*np.dot((np.transpose(x1_train-mu1_es)),(x1_train-mu1_es));
cov2_es=(1/len(x2_train))*np.dot((np.transpose(x2_train-mu2_es)),(x2_train-mu2_es));
cov3_es=(1/len(x3_train))*np.dot((np.transpose(x3_train-mu3_es)),(x3_train-mu3_es));
cov4_es=(1/len(x4_train))*np.dot((np.transpose(x4_train-mu4_es)),(x4_train-mu4_es));
cov5_es=(1/len(x5_train))*np.dot((np.transpose(x5_train-mu5_es)),(x5_train-mu5_es));
cov6_es=(1/len(x6_train))*np.dot((np.transpose(x6_train-mu6_es)),(x6_train-mu6_es));
cov7_es=(1/len(x7_train))*np.dot((np.transpose(x7_train-mu7_es)),(x7_train-mu7_es));
cov8_es=(1/len(x8_train))*np.dot((np.transpose(x8_train-mu8_es)),(x8_train-mu8_es));
cov9_es=(1/len(x9_train))*np.dot((np.transpose(x9_train-mu9_es)),(x9_train-mu9_es));
cov10_es=(1/len(x10_train))*np.dot((np.transpose(x10_train-mu10_es)),(x10_train-mu10_
cov11_es=(1/len(x11_train))*np.dot((np.transpose(x11_train-mu11_es)),(x11_train-mu11_
cov12_es=(1/len(x12_train))*np.dot((np.transpose(x12_train-mu12_es)),(x12_train-mu12_

```

In [ ]:

```
X_test_no=np.zeros((42923,500))

for i in range(42923):
    X_test_no[i,:]= model.infer_vector(np.asarray(X_test)[i])
```

In [ ]:

```
prob_prior=np.zeros((12))
prob_prior[0]=len(x1_train)/total
prob_prior[1]=len(x2_train)/total
prob_prior[2]=len(x3_train)/total
prob_prior[3]=len(x4_train)/total
prob_prior[4]=len(x5_train)/total
prob_prior[5]=len(x6_train)/total
prob_prior[6]=len(x7_train)/total
prob_prior[7]=len(x8_train)/total
prob_prior[8]=len(x9_train)/total
prob_prior[9]=len(x10_train)/total
prob_prior[10]=len(x11_train)/total
prob_prior[11]=len(x12_train)/total
print(prob_prior)
```

In [ ]:

```
from sklearn.metrics import confusion_matrix,accuracy_score
from scipy.stats import multivariate_normal
y_pred=np.zeros((42923))
for i in range(42923):
    p1=multivariate_normal.pdf(X_test_no[i,:],mu1_es,cov1_es)*prob_prior[0]
    p2=multivariate_normal.pdf(X_test_no[i,:],mu2_es,cov2_es)*prob_prior[1]
    p3=multivariate_normal.pdf(X_test_no[i,:],mu3_es,cov3_es)*prob_prior[2]
    p4=multivariate_normal.pdf(X_test_no[i,:],mu4_es,cov4_es)*prob_prior[3]
    p5=multivariate_normal.pdf(X_test_no[i,:],mu5_es,cov5_es)*prob_prior[4]
    p6=multivariate_normal.pdf(X_test_no[i,:],mu6_es,cov6_es)*prob_prior[5]
    p7=multivariate_normal.pdf(X_test_no[i,:],mu7_es,cov7_es)*prob_prior[6]
    p8=multivariate_normal.pdf(X_test_no[i,:],mu8_es,cov8_es)*prob_prior[7]
    p9=multivariate_normal.pdf(X_test_no[i,:],mu9_es,cov9_es)*prob_prior[8]
    p10=multivariate_normal.pdf(X_test_no[i,:],mu10_es,cov10_es)*prob_prior[9]
    p11=multivariate_normal.pdf(X_test_no[i,:],mu11_es,cov11_es)*prob_prior[10]
    p12=multivariate_normal.pdf(X_test_no[i,:],mu12_es,cov12_es)*prob_prior[11]
    prob=np.column_stack((p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12))
    print(prob)
    try:
        y_pred[i]=(np.where(prob[0,:] == np.amax(prob[0,:]))[0])+1
    except:
        y_pred[i]=1
print(confusion_matrix(Y_testno, y_pred))
print(accuracy_score(Y_testno, y_pred))
```

In [ ]:

```
print(accuracy_score(Y_testno, y_pred))
```

Naive bayes

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn
import warnings
fil_data=pd.read_excel('filtered_data.xlsx')
```

In [ ]:

```
import sklearn
from sklearn import model_selection
X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(fil_data[
                                                                            fil_data[
```

In [ ]:

```
#preprocessing the data
from nltk.stem import PorterStemmer, WordNetLemmatizer
import sklearn.model_selection
import re
from nltk.corpus import stopwords
import numpy as np
import sklearn
import nltk
X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(fil_data[
                                                                                    fil_data[

X_train = np.array(X_train);
X_test = np.array(X_test);
Y_train = np.array(Y_train);
Y_test = np.array(Y_test);

procText_train = []
procText_test = []
number_train = len(X_train)
number_test = len(X_test)

lemmetizer = WordNetLemmatizer()
stemmer = PorterStemmer()
def get_words(headlines_list):
    headlines = headlines_list[0]
    headlines_only_letters = re.sub('[^a-zA-Z]', ' ', headlines)
    words = nltk.word_tokenize(headlines_only_letters.lower())
    stops = set(stopwords.words('english'))
    meaningful_words = [lemmetizer.lemmatize(w) for w in words if w not in stops]
    return ' '.join(meaningful_words )

for i in range(number_train):
    proctext = get_words(X_train[i]) #Processing the data and getting words with no s
    procText_train.append( proctext )
print("train words done")
for i in range(number_test):
    proctext = get_words(X_test[i]) #Processing the data and getting words with no sp
    procText_test.append( proctext )
print("test words done")
vectorize = sklearn.feature_extraction.text.TfidfVectorizer(analyzer = "word", max_fe
tfidwords_train = vectorize.fit_transform(procText_train)
X_train = tfidwords_train.toarray()

tfidwords_test = vectorize.transform(procText_test)
X_test = tfidwords_test.toarray()
print("vectorizer done")
```

In [ ]:

```
x1_train=[]
x2_train=[]
x3_train=[]
x4_train=[]
x5_train=[]
x6_train=[]
x7_train=[]
x8_train=[]
x9_train=[]
x10_train=[]
x11_train=[]
x12_train=[]
x1_test=[]
x2_test=[]
x3_test=[]
x4_test=[]
x5_test=[]
x6_test=[]
x7_test=[]
x8_test=[]
x9_test=[]
x10_test=[]
x11_test=[]
x12_test=[]
y1_train=[]
y2_train=[]
y3_train=[]
y4_train=[]
y5_train=[]
y6_train=[]
y7_train=[]
y8_train=[]
y9_train=[]
y10_train=[]
y11_train=[]
y12_train=[]
y1_test=[]
y2_test=[]
y3_test=[]
y4_test=[]
y5_test=[]
y6_test=[]
y7_test=[]
y8_test=[]
y9_test=[]
y10_test=[]
y11_test=[]
y12_test=[]

```

In [ ]:

```
for i in range(Y_train.size):
    if Y_train[i]=="BUSINESS":
        x1_train.append(X_train[i,:])
        y1_train.append(Y_train[i])
    elif Y_train[i]=="COMEDY":
        x2_train.append(X_train[i,:])
        y2_train.append(Y_train[i])
    elif Y_train[i]=="ENTERTAINMENT":
        x3_train.append(X_train[i,:])
        y3_train.append(Y_train[i])
    elif Y_train[i]=="FOOD & DRINK":
        x4_train.append(X_train[i,:])
        y4_train.append(Y_train[i])
    elif Y_train[i]=="HEALTHY LIVING":
        x5_train.append(X_train[i,:])
        y5_train.append(Y_train[i])
    elif Y_train[i]=="PARENTING":
        print(2)
        x6_train.append(X_train[i,:])
        y6_train.append(Y_train[i])
    elif Y_train[i]=="POLITICS":
        print(1)
        x7_train.append(X_train[i,:])
        y7_train.append(Y_train[i])
    elif Y_train[i]=="QUEER VOICES":
        x8_train.append(X_train[i,:])
        y8_train.append(Y_train[i])
    elif Y_train[i]=="SPORTS":
        x9_train.append(X_train[i,:])
        y9_train.append(Y_train[i])
    elif Y_train[i]=="STYLE & BEAUTY":
        x10_train.append(X_train[i,:])
        y10_train.append(Y_train[i])
    elif Y_train[i]=="TRAVEL":
        x11_train.append(X_train[i,:])
        y11_train.append(Y_train[i])
    elif Y_train[i]=="WELLNESS":
        x12_train.append(X_train[i,:])
        y12_train.append(Y_train[i])
```

In [ ]:

```
for i in range(Y_test.size):
    if Y_test[i]=="BUSINESS":
        x1_test.append(X_test[i,:])
        y1_test.append(Y_test[i])
    elif Y_test[i]=="COMEDY":
        x2_test.append(X_test[i,:])
        y2_test.append(Y_test[i])
    elif Y_test[i]=="ENTERTAINMENT":
        x3_test.append(X_test[i,:])
        y3_test.append(Y_test[i])
    elif Y_test[i]=="FOOD & DRINK":
        x4_test.append(X_test[i,:])
        y4_test.append(Y_test[i])
    elif Y_test[i]=="HEALTHY LIVING":
        x5_test.append(X_test[i,:])
        y5_test.append(Y_test[i])
    elif Y_test[i]=="PARENTING":
        x6_test.append(X_test[i,:])
        y6_test.append(Y_test[i])
    elif Y_test[i]=="POLITICS":
        x7_test.append(X_test[i,:])
        y7_test.append(Y_test[i])
    elif Y_test[i]=="QUEER VOICES":
        x8_test.append(X_test[i,:])
        y8_test.append(Y_test[i])
    elif Y_test[i]=="SPORTS":
        x9_test.append(X_test[i,:])
        y9_test.append(Y_test[i])
    elif Y_test[i]=="STYLE & BEAUTY":
        x10_test.append(X_test[i,:])
        y10_test.append(Y_test[i])
    elif Y_test[i]=="TRAVEL":
        x11_test.append(X_test[i,:])
        y11_test.append(Y_test[i])
    elif Y_test[i]=="WELLNESS":
        x12_test.append(X_test[i,:])
        y12_test.append(Y_test[i])
```

In [ ]:

```
total=len(x1_train)+len(x2_train)+len(x3_train)+len(x4_train)+len(x5_train)+len(x6_tr
print(total)
```

In [ ]:

```
print(len(x1_train))
print(len(x2_train))
print(len(x3_train))
print(len(x4_train))
print(len(x5_train))
print(len(x6_train))
print(len(x7_train))
print(len(x8_train))
print(len(x9_train))
print(len(x10_train))
print(len(x11_train))
print(len(x12_train))
```

In [ ]:

```
x1_train=np.array(x1_train)
x2_train=np.array(x2_train)
x3_train=np.array(x3_train)
x4_train=np.array(x4_train)
x5_train=np.array(x5_train)
x6_train=np.array(x6_train)
x7_train=np.array(x7_train)
x8_train=np.array(x8_train)
x9_train=np.array(x9_train)
x10_train=np.array(x10_train)
x11_train=np.array(x11_train)
x12_train=np.array(x12_train)
x1_test=np.array(x1_test)
x2_test=np.array(x2_test)
x3_test=np.array(x3_test)
x4_test=np.array(x4_test)
x5_test=np.array(x5_test)
x6_test=np.array(x6_test)
x7_test=np.array(x7_test)
x8_test=np.array(x8_test)
x9_test=np.array(x9_test)
x10_test=np.array(x10_test)
x11_test=np.array(x11_test)
x12_test=np.array(x12_test)
```

In [ ]:

```
count_words=np.zeros((10000))
for i in range(x1_train.shape[1]):
    for j in range(X_train.shape[0]):
        if X_train[j,i]!=0:
            count_words[i]+=1;
print(count_words)
```

In [ ]:

```
prob=np.zeros((12,10000))
for i in range(x1_train.shape[1]):
    count=0
    for j in range(x1_train.shape[0]):
        if x1_train[j,i]!=0:
            count=+1
    prob[0,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x2_train.shape[0]):
        if x2_train[j,i]!=0:
            count=+1
    prob[1,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x3_train.shape[0]):
        if x3_train[j,i]!=0:
            count=+1
    prob[2,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x4_train.shape[0]):
        if x4_train[j,i]!=0:
            count=+1
    prob[3,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x5_train.shape[0]):
        if x5_train[j,i]!=0:
            count=+1
    prob[4,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x6_train.shape[0]):
        if x6_train[j,i]!=0:
            count=+1
    prob[5,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x7_train.shape[0]):
        if x7_train[j,i]!=0:
            count=+1
    prob[6,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x8_train.shape[0]):
        if x8_train[j,i]!=0:
            count=+1
    prob[7,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x9_train.shape[0]):
        if x9_train[j,i]!=0:
            count=+1
    prob[8,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x10_train.shape[0]):
        if x10_train[j,i]!=0:
            count=+1
    prob[9,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x11_train.shape[0]):
        if x11_train[j,i]!=0:
            count=+1
    prob[10,i]=(count+1)/(100+count_words[i])
    count=0
    for j in range(x12_train.shape[0]):
```

```
        if x12_train[j,i]!=0:
            count=+1
    prob[11,i]=(count+1)/(100+count_words[i])

print(prob[:,0])
```

In [ ]:

```
prob_prior=np.zeros((12))
prob_prior[0]=len(x1_train)/total
prob_prior[1]=len(x2_train)/total
prob_prior[2]=len(x3_train)/total
prob_prior[3]=len(x4_train)/total
prob_prior[4]=len(x5_train)/total
prob_prior[5]=len(x6_train)/total
prob_prior[6]=len(x7_train)/total
prob_prior[7]=len(x8_train)/total
prob_prior[8]=len(x9_train)/total
prob_prior[9]=len(x10_train)/total
prob_prior[10]=len(x11_train)/total
prob_prior[11]=len(x12_train)/total
print(prob_prior)
```

In [ ]:

```
post_prob=np.ones((X_test.shape[0],12))
for i in range(X_test.shape[0]):
    for j in range(10000):
        for k in range(12):
            if X_test[i,j]!=0:
                post_prob[i,k]=post_prob[i,k]*prob[k,j]

```

In [ ]:

```
post_prob_prior=post_prob*prob_prior
```

In [ ]:

```
y_pred=np.zeros((X_test.shape[0]))
for i in range(X_test.shape[0]):
    try:
        y_pred[i]=(np.where(post_prob[i,:] == np.amax(post_prob[i,:]))[0])+1
    except:
        print(i)
        y_pred[i]=np.random.choice((np.where(post_prob[i,:] == np.amax(post_prob[i,:]
```

In [ ]:

```
Y_testno=np.zeros((Y_test.size))
for i in range(Y_test.size):
    if Y_test[i]=="BUSINESS":
        Y_testno[i]=1
    elif Y_test[i]=="COMEDY":
        Y_testno[i]=2
    elif Y_test[i]=="ENTERTAINMENT":
        Y_testno[i]=3
    elif Y_test[i]=="FOOD & DRINK":
        Y_testno[i]=4
    elif Y_test[i]=="HEALTHY LIVING":
        Y_testno[i]=5
    elif Y_test[i]=="PARENTING":
        Y_testno[i]=6
    elif Y_test[i]=="POLITICS":
        Y_testno[i]=7
    elif Y_test[i]=="QUEER VOICES":
        Y_testno[i]=8
    elif Y_test[i]=="SPORTS":
        Y_testno[i]=9
    elif Y_test[i]=="STYLE & BEAUTY":
        Y_testno[i]=10
    elif Y_test[i]=="TRAVEL":
        Y_testno[i]=11
    elif Y_test[i]=="WELLNESS":
        Y_testno[i]=12

```

In [ ]:

```
Y_trainno=np.zeros((Y_train.size))
for i in range(Y_train.size):
    if Y_train[i]=="BUSINESS":
        Y_trainno[i]=1
    elif Y_train[i]=="COMEDY":
        Y_trainno[i]=2
    elif Y_train[i]=="ENTERTAINMENT":
        Y_trainno[i]=3
    elif Y_train[i]=="FOOD & DRINK":
        Y_trainno[i]=4
    elif Y_train[i]=="HEALTHY LIVING":
        Y_trainno[i]=5
    elif Y_train[i]=="PARENTING":
        Y_trainno[i]=6
    elif Y_train[i]=="POLITICS":
        Y_trainno[i]=7
    elif Y_train[i]=="QUEER VOICES":
        Y_trainno[i]=8
    elif Y_train[i]=="SPORTS":
        Y_trainno[i]=9
    elif Y_train[i]=="STYLE & BEAUTY":
        Y_trainno[i]=10
    elif Y_train[i]=="TRAVEL":
        Y_trainno[i]=11
    elif Y_train[i]=="WELLNESS":
        Y_trainno[i]=12

```

In [ ]:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_testno, np.asarray(y_pred)))
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_testno, np.asarray(y_pred)))
```

KNN

In [ ]:

```
from scipy.spatial import distance
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix

K={20}
#K={3}
count=0
err=np.zeros((6))
for k in K:
    print(k)
    y_pred=np.zeros((10000))
    for i in range(10000):
        neighbors=[]
        dist=np.zeros((20000,2))
        for j in range(20000):
            dist[j,0]=distance.euclidean(X_train[j,:], X_test[i,:])
            dist[j,1]=Y_trainno[j]
        dist=dist[dist[:,0].argsort()]
        for value in range(k):
            neighbors.append(dist[value,1])
        y_pred[i]=np.bincount(neighbors).argmax()
        if Y_testno[i]!=y_pred[i]:
            err[count]=err[count]+1
    print(confusion_matrix(Y_testno, y_pred))
    print(accuracy_score(Y_testno[0:10000], y_pred))
    count=count+1

```

AdaBoost, Linear SVC, Logistic regression, bagging and decision tree classifier

In [ ]:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
clf = AdaBoostClassifier(n_estimators=20)
#scores = cross_val_score(clf, X_train, Y_train, cv=5)

y_pred = clf.fit(X_train, Y_train).predict(X_test)

from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix

model = LinearSVC()
model.fit(X_train,Y_train)
Y_predict = model.predict(X_test)
accuracy = accuracy_score(Y_test,Y_predict)*100
print(format(accuracy, '.2f'))
print(confusion_matrix(Y_test,Y_predict))

from sklearn.linear_model import LogisticRegression
logistic_Regression = LogisticRegression()
logistic_Regression.fit(X_train,Y_train)
Y_predict = logistic_Regression.predict(X_test)
accuracy = accuracy_score(Y_test,Y_predict)*100
print(format(accuracy, '.2f'))
print(confusion_matrix(Y_test,Y_predict))

from sklearn.ensemble import BaggingClassifier
model = BaggingClassifier(random_state=0, n_estimators=10)
model.fit(X_train, Y_train)
prediction = model.predict(X_test)
print('Accuracy of bagged KNN is :',accuracy_score(prediction, Y_test))
print(confusion_matrix(Y_test,Y_predict))

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
prediction_decision_tree = model.predict(X_test)
print('The accuracy of Decision Tree is', accuracy_score(prediction_decision_tree, Y_
print(confusion_matrix(Y_test,prediction_decision_tree))
```