Project 2 Report

By,

Anupkumar Nagaraj Joshi

101880602

Aimee Ciane Nyambo

101880199

Naïve Bayes Method:

1. Description about the code:

   Flow of the program:

   Steps followed:

   1. Load all the csv files and create sparse matrix and files got from training.
   2. Extract each testing document for prediction
   3. Find value for each class for the words list obtained.
   4. The class with highest value will be the class predicted.
   5. Finally, prediction is converted to a csv file.

   This code explanation can be divided into three parts:

   a. Training: Following are the functions used to train our method. Training data set is used for these functions.

      1. countWordOccurenceOfCategory(category,word) :This function takes category(1-20) and wordId(1-61189) and returns its count. That is total number of words of particular wordId belonging to that particular category.

      2. getWordOccurenceForAllCategory():This function iterates 20 times for each word ID and returns list of lists which has count of the wordID category wise. That is, rows indicate category and columns indicate wordIDs and cell value indicates total count of that wordID (column id) belonging to particular category (row id).

         For Eg: wordOccurenceCategory.txt is the file got from above function

      3. totalNumberOfWordsOfCat(category): This function accepts category and return total number of words present in that category. This value is used while calculating MAP.

      4. totalDocsOfACat(category): This function accepts category and returns total number of documents with that category. This is used to find MLE.

      5. wordProb(): this function returns total number of words for each wordID. This value is used in calculating weightage for each word.

         For Eg: IGList.txt has the file got from above function.

   b. Testing: Following are the functions used to extract each document and predict its class .And output to a csv file of required format.

1. findWordsInTesting() : This functions extracts individual word IDs from each document using sparse matrix and calls calculateProbability(wordlist,rowCount) function to predict the class.

2. calculateProbability(wordlist,rowCount) : This iterates through all classes and wordlist obtained to calculate MAP and MLE and final calculate value for each class.Class with highest value is the one predicted.

3. findWeightOfWords(): It calculates entropy(IG) for each of the word recursing through each of the classes. The total value for each wordId is appended in the list.

   For Eg: IGList.txt is the file output by above function.

4. findTop100Weights() : This function returns list of the 100 words with highest information gain.

   This is more explained in the answer for the Q6.

c. Other functions:
1. outputFormat(): it makes the list of list containing ID and respective predicted class.
2. listOfListToCSV(): it converts list got from above function to csv file.
3. plotMap(): it creates graph comparing B values and accuracy. [For Q3]
4. Main function(): Which imports all csv data and convert to sparse matrix .It has all the required variables, list initialized and function calls.

2. Accuracies under various setting:

Beta Values	Accuracy
0.00001	0.86713
0.0001	**0.86802**
0.001	0.86772
0.01	0.85975
0.1	0.82993

Files used are : CSV files with names : finalOutput(0.00001), finalOutput(0.0001), finalOutput(0.001), finalOutput(0.1),finalOutput(0.1)

We can see that accuracy is lower for higher and lower values of beta. It gives better accuracy when beta value is 0.0001. Reason being, when beta is low it relatively decreases the importance of the feature. When Beta is high it overestimates the importance of the feature. To avoid these cases, beta is required to be optimum.

Logistics Regression:

1. Description about the code:

   Flow of the program:

   Steps followed:

   1. Train all the matrix required. That is delta matrix, X matrix and initialize W matrix.
   2. Update W for n iterations using the given formula for a learning rate and penalty rate.
   3. Find matrix exp(W*Xt), where W is the updated W and Xt is the transpose of the testing data matrix.
   4. Finally, find argmax for each of the column. The index of that highest value in each column would be predicted class.
   5. Output the predicted class to required format.

      The functions used for the implementation can be divided into three parts:

      a. Training functions: Below are the functions used to train, that is get the delta, X and inititalize W matrix using sparse matrix.

         1. createDeltaMatrix () and deltaFunction() : This functions help to create delta matrix. This has classes as rows and examples as columns.
            For eg: deltaMatrix(train) is obtained from above functions.

         2. exMatrixAndYMatrix (): This is used create X matrix and Y matrix. X contains m examples and n+1 column where n are attributes. Y Matrix is column matrix which has all the classes.
            For eg: exMatrix(Train).txt is got from above function.

         3. initializeMatrix(N,M) : It is used to initialize matrix (null matrix). N is the number of rows and M is the number of columns.

         4. update(numberOfIterations): It is used to update W for given numberOfIterations. This is used for prediction.

         5. getNormalizedMatrix(): It is used to calculate exp(W.Xt) for every above iteration. The resultant matrix is normalized by dividing each cell by sum of respective column values.

b. Testing: Now classes are predicted using updated W matrix.

1. getMatrixForPrediction() and predict() : These functions find exp(W.Xt) for testing document. The resultant matrix is checked for highest value column wise using argmax. Row with highest value would be class predicted.

c. Other functions:

5. outputFormat(): it makes the list of list containing ID and respective predicted class.
6. listOfListToCSV(): it converts list got from above function to csv file.
7. Main function(): Which imports all csv data and convert to sparse matrix .It has all the required variables, list initialized and function calls.

2. Accuracies under various setting:

Learning rate	Penalty term	Iterations	Accuracy
0.01	0.01	1000	.80458
0.004	0.004	1500	.85346
0.001	0.001	2000	.82145

We can see that factors that matter for the accuracy are:

1. Learning rate: This value represents how fast the model learns. Higher value results in no convergence while lower results in slow convergence. Hence optimum value should be selected.
2. Number of iteration(gd): The appropriate value is reached after certain iteration. Higher number of iterations would give no better result because it saturates after certain iteration.
3. Penalty rate: The penalty rate should be optimum. Because initially it avoids overfitting without affecting bias but after certain point it increases bias resulting in underfitting.

   Above mentioned effects can be seen in the table.

Naïve bayes vs logistic regression:

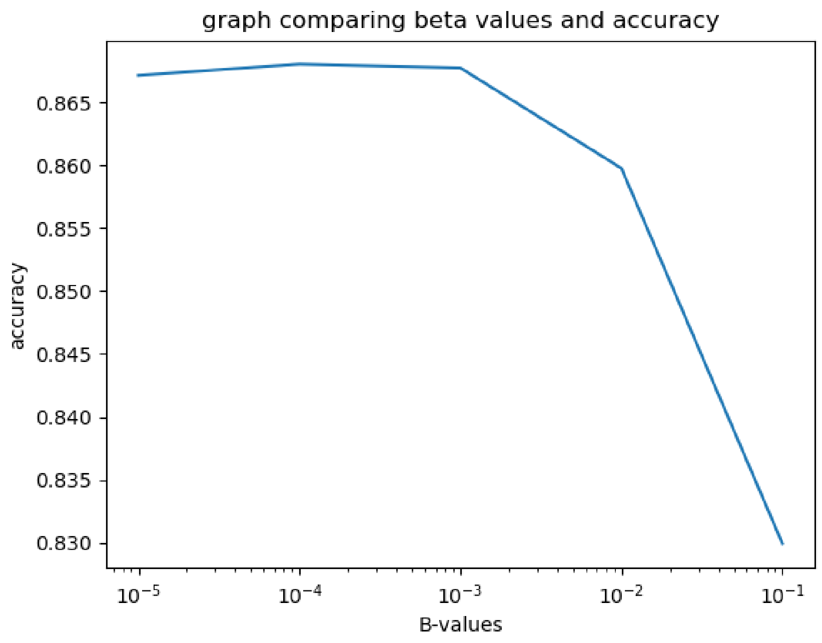
1. Naïve Bayes:
   Accuracy: This assumes all features are conditionally independent. Hence it has higher bias and lower variance comparing to logistic regression. Hence accuracy may be lower because in real data there can be some dependency between features.
   Performance: This does not involve gradient descent and hence no need for convergence. This deals with simple calculations. Ergo, performance is better than logistic regression.

2. Logistic regression:
   Accuracy: This calculates weight of each feature by using gradient descent. Hence accuracy is can be higher
   Performance: Since it uses gradient descent, convergence can be issue. And it depends on many factors like learning rate, penalty rate and iterations. Hence performance is lower compared to Naïve Bayes.

Questions and Answers:

A.1 Given that there are 1000 documents and 1000 each. Words occur from 50000. It would be difficult to estimate parameters because features are less. There are no enough words to distinguish class from others. It means a lot of words may be common in all the classes. Odds of few words being in particular class is less.

A.2 Below graph compares Beta with accuracy



Accuracy drop for lower and higher beta values

A.3 . Factors for logistic regression:

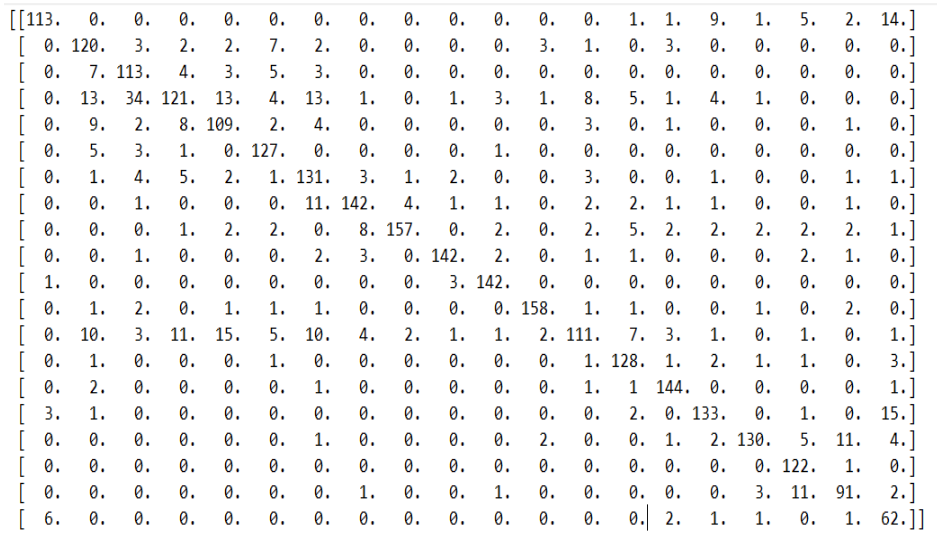
Learning rate	Penalty term	Iterations	Accuracy
0.01	0.01	1000	.80458
0.004	**0.004**	**1500**	**.85346**
0.001	0.001	2000	.82145

We can see that factors that matter for the accuracy are:

1. Learning rate: This value represents how fast the model learns. Higher value results in no convergence while lower results in slow convergence. Hence optimum value should be selected.
2. Number of iteration(gd): The appropriate value is reached after certain iteration. Higher number of iterations would give no better result because it saturates after certain iteration.
3. Penalty rate: The penalty rate should be optimum. Because initially it avoids overfitting without affecting bias but after certain point it increases bias resulting in underfitting.

   The sweet spot would optimum values for all factors. Learning rate and penalty rate =0.04 and iteration = 1500
   Above mentioned effects can be seen in the table.

A.4 Confusion matrix : Below is the matrix which contains counts of predictions. Rows are the predicted classes and columns contains actual classes.



```
[[113.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   1.   1.   9.   1.   5.   2.  14.]
 [  0. 120.   3.   2.   2.   7.   2.   0.   0.   0.   0.   3.   1.   0.   3.   0.   0.   0.   0.   0.]
 [  0.   7. 113.   4.   3.   5.   3.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.  13.  34. 121.  13.   4.  13.   1.   0.   1.   3.   1.   8.   5.   1.   4.   1.   0.   0.   0.]
 [  0.   9.   2.   8. 109.   2.   4.   0.   0.   0.   0.   0.   3.   0.   1.   0.   0.   0.   1.   0.]
 [  0.   5.   3.   1.   0. 127.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   1.   4.   5.   2.   1. 131.   3.   1.   2.   0.   0.   3.   0.   0.   1.   0.   0.   1.   1.]
 [  0.   0.   1.   0.   0.   0.  11. 142.   4.   1.   1.   0.   2.   2.   1.   1.   0.   0.   1.   0.]
 [  0.   0.   0.   1.   2.   2.   0.   8. 157.   0.   2.   0.   2.   5.   2.   2.   2.   2.   2.   1.]
 [  0.   0.   1.   0.   0.   0.   2.   3.   0. 142.   2.   0.   1.   1.   0.   0.   0.   2.   1.   0.]
 [  1.   0.   0.   0.   0.   0.   0.   0.   0.   3. 142.   0.   0.   0.   0.   0.   0.   0.   0.   0.]
 [  0.   1.   2.   0.   1.   1.   1.   0.   0.   0.   0. 158.   1.   1.   0.   0.   1.   0.   2.   0.]
 [  0.  10.   3.  11.  15.   5.  10.   4.   2.   1.   1.   2. 111.   7.   3.   1.   0.   1.   0.   1.]
 [  0.   1.   0.   0.   0.   1.   0.   0.   0.   0.   0.   0.   1. 128.   1.   2.   1.   1.   0.   3.]
 [  0.   2.   0.   0.   0.   0.   1.   0.   0.   0.   0.   0.   1.   1  144.   0.   0.   0.   0.   1.]
 [  3.   1.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   2.   0. 133.   0.   1.   0.  15.]
 [  0.   0.   0.   0.   0.   0.   1.   0.   0.   0.   0.   2.   0.   0.   1.   2. 130.   5.  11.   4.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0. 122.   1.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   1.   0.   0.   0.   0.   0.   3.  11.  91.   2.]
 [  6.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   2.   1.   1.   0.   1.  62.]]
```

A.5 Below is the list of total counts of confused classes for each class.

[10.0, 50.0, 53.0, 32.0, 38.0, 28.0, 48.0, 20.0, 7.0, 8.0, 11.0, 8.0, 23.0, 25.0, 16.0, 23.0, 10.0, 28.0, 23.0, 42.0]

We can see that classes 2,3,7 and 20 are more confusing than others. The reason may be these classes have documents having words of lower weights. Those words may be having words with least information gain and more entropy. Hence more difficulty for the model to classify.

A.6 This can be accomplished using entropy and IG. Firstly, calculate entropy for entire class. Entropy of each of the branches. That is, one where given a word it belongs to class and other is not given word and it belongs to class. Then add all entropies to find information gain. Then words with highest IG are the words which help most in categorizing.

Entropy and IG expression:

Entropy of class = sumAllClass(probability of class * log(probability of class))

Entropy of class when given word = probability of word * sumAllClass ((probability of class given word) *log(probability of class given word))

Entropy of class without word = probability of not given word * sumAllClass ((probability of class not given word) *log(probability of class not given word))

IG = - entropy of class + Entropy of class when given word + Entropy of class without word

A.7 Used the above method to find 100 words with highest IG. Below is the list of such wordIDs:

[772, 772, 831, 877, 1174, 1619, 2040, 2582, 2593, 2638, 2640, 3295, 3883, 3920, 3921, 3939, 4135, 4135, 4149, 4166, 4219, 4219, 4220, 4406, 4475, 4504, 4505, 4728, 4795, 4816, 4953, 4953, 4957, 5030, 5051, 5063, 5070, 5265, 5577, 5577, 5682, 5703, 5756, 5777, 5946, 6305, 6307, 6311, 6371, 6390, 6592, 6620, 6636, 6636, 6653, 6660, 6720, 7005, 7040, 7277, 7514, 7570, 7571, 7655, 7681, 7773, 7773, 7844, 7876, 7975, 7987, 8011, 8640, 8640, 8704, 8721, 8831, 8831, 8832, 9123, 9131, 9144, 9144, 9159, 9169, 9192, 9194, 9219, 9219, 9227, 9244, 9245, 9245, 9248, 9252, 9254, 9254, 9255, 9263, 9264]

A.8 The bias in data can be seen if we analyze above words. The words with highest IG influence more on categorization. These words increase bias in likelihood functions. They tend to lean towards few classes which results in bias while categorization.