**Implementation of naïve bayes algorithm**

**Classification and Evaluation**

For classification and evaluation ,I implemented NaiveBayes.Train and NaiveBayes.PredictLabel

and applied smoothing hyperparameter with various value of ALPHA. NaiveBayes.Train calculates and initializes the variable. NaiveBayes.PredictLabel calculates the class of all words.

|  |  |
| --- | --- |
| ALPHA value | Accuracy |
| 1 | 0.82284 |
| 0.5 | 0.82284 |
| 0.1 | 0.82284 |
| 5.0 | 0.82284 |
| 10.0 | 0.82284 |

Below is the code of NaiveBayes.Train and NaiveBayes.PredictLabel :

def Train(self, X, Y):

# TODO: Estimate Naive Bayes model parameters

positive\_indices = np.argwhere(Y == 1.0).flatten()

negative\_indices = np.argwhere(Y == -1.0).flatten()

self.num\_positive\_reviews = len(positive\_indices)

self.num\_negative\_reviews = len(negative\_indices)

self.count\_positive = csr\_matrix.sum(X[np.ix\_(positive\_indices)], axis=0)

self.count\_negative = csr\_matrix.sum(X[np.ix\_(negative\_indices)], axis=0)

self.total\_positive\_words = csr\_matrix.sum(X[np.ix\_(positive\_indices)])

self.total\_negative\_words = csr\_matrix.sum(X[np.ix\_(negative\_indices)])

self.deno\_pos = float(self.total\_positive\_words + self.ALPHA \* X.shape[1])

self.deno\_neg = float(self.total\_negative\_words + self.ALPHA \* X.shape[1])

return

def PredictLabel(self, X):

# TODO: Implement Naive Bayes Classification

self.P\_positive\_review = (float(self.num\_positive\_reviews)/(self.num\_positive\_reviews + self.num\_negative\_reviews))

self.P\_negative\_review = (float(self.num\_negative\_reviews)/(self.num\_positive\_reviews + self.num\_negative\_reviews))

pred\_labels = []

sh = X.shape[0]

for i in range(sh):

z = X[i].nonzero()

Positive\_label\_val = log(self.P\_positive\_review)

Negative\_label\_val = log(self.P\_negative\_review)

for j in range(len(z[0])):

rowvalue = X[i,z[1][j]]

Positive\_val = log(self.count\_positive[0, z[1][j]]+ self.ALPHA) - log(self.deno\_pos)

Positive\_label\_val = Positive\_label\_val + rowvalue \* Positive\_val

Negative\_val = log(self.count\_negative[0, z[1][j]]+ self.ALPHA) - log(self.deno\_neg)

Negative\_label\_val = Negative\_label\_val + rowvalue \* Negative\_val

# print(self.Positive\_label\_val)

# print(self.Negative\_label\_val)

if Positive\_label\_val > Negative\_label\_val:

pred\_labels.append(1.0)

else:

pred\_labels.append(-1.0)

return pred\_labels

# return 1

**Probability Prediction**

We implemented NaiveBayes.LogSum and NaiveBayes.PredictProb for predicting the positive and negative class probabilities of first 10 reviews:

|  |  |  |
| --- | --- | --- |
| Sentence no. | Predicted Positive Probability | Predicted negative probability |
| 1. | 0.39999999999999997 | 0.6 |
| 2. | 0.33333333333333337 | 0.6666666666666667 |
| 3. | 0.33333333333333337 | 0.6666666666666667 |
| 4. | 0.6666666666666667 | 0.33333333333333337 |
| 5. | 0.25 | 0.7500000000000001 |
| 6. | 0.33333333333333337 | 0.6666666666666667 |
| 7. | 0.5 | 0.5 |
| 8. | 0.5 | 0.5 |
| 9. | 0.5 | 0.5 |
| 10 | 0.33333333333333337 | 0.6666666666666667 |

Below is the code that I implemented for NaiveBayes.LogSum and NaiveBayes.PredictProb :

def LogSum(self, logx, logy):

# TO Do: Return log(x+y), avoiding numerical underflow/overflow.

m = max(logx, logy)

return m + log(exp(logx - m) + exp(logy - m))

def PredictProb(self, test, indexes):

for i in indexes:

# TO DO: Predict the probability of the i\_th review in test being positive review

# TO DO: Use the LogSum function to avoid underflow/overflow

predicted\_label = 0

z = test.X[i].nonzero()

positive\_val = self.P\_positive\_review

negative\_val = self.P\_negative\_review

for j in range(len(z[0])):

datavalue = test.X[i,z[1][j]]

predicted\_negative\_value = log((self.count\_negative[0, z[1][j]]+self.ALPHA))

negative\_val = negative\_val + datavalue \* predicted\_negative\_value

predicted\_positive\_value = log((self.count\_positive[0, z[1][j]]+self.ALPHA))

positive\_val = positive\_val + datavalue \* predicted\_positive\_value

predicted\_prob\_positive = exp(predicted\_positive\_value - self.LogSum(predicted\_positive\_value,predicted\_negative\_value))

predicted\_prob\_negative = exp(predicted\_negative\_value - self.LogSum(predicted\_positive\_value,predicted\_negative\_value))

if predicted\_positive\_value > predicted\_negative\_value:

predicted\_label = 1.0

else:

predicted\_label = -1.0

# print test.Y[i], test.X\_reviews[i]

# TO DO: Comment the line above, and uncomment the line below

print(test.Y[i], predicted\_label, predicted\_prob\_positive, predicted\_prob\_negative, test.X\_reviews[i])