Question 1.

a.

Linear threshold function is

f(x) = 1 if - >= 0 and

f(x) = -1 otherwise

For the above said linear threshold function to be valid with the given condition the values that “w” and “” should take are w=1 and =m.

b.

P(y=0) = (C(8,0) + C(8,1) + C(8,2))/256

= (1 + 8 + 28)/256

= 37/256

P(y=1) = 1 - p(y=0)

= 1 - 37/256

= 219/256

= p (=1|y=1) = /219= 120/219

Also =======

= p (=1|y=0) = /37= 8/37

=======

= log((1-120/219)/(1-8/37))\*8

= 4.6136

W1 = log (/(1-) - log(/(1-)

W1 = 1.095

W1 = W2 = W3 = W4 = W5 = W6 = W7 = W8

Once we have the values for the parameters i.e the and substituting these values to the given equation results in learned naive bayes hypothesis

C. It does not exactly learn the same target function. Weights should have been 1 but the naive bayes classifier computes as 1.095 and value should have been 3 but the value learnt is 4.6136

Naïve Bayes requires naïve bayes assumptions be satisfied where’s linear function does not make any assumption. With this argument we can say the hypothesis space for Naïve Bayes is more restrictive whereas linear function is more expressive. Because of the restrictive nature of Naïve Bayes the hypothesis learned by Naïve bayes is not the actual target function.

https://pdfs.semanticscholar.org/8880/2dffb42b21e5e5113e7d69a7e05c3321a99b.pdf

Question 2:

1. Submitted the code on github : https://github.ccs.neu.edu/cs6140-03-spring2017/kattaseshasai09
2. I have not done anything interesting. I have used the algorithm and data as-is.
3. Confusion matrix for the Naive Bayes output

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Class** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **Accuracy** |
| **1** | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 4/6=.66 |
| **2** | 0 | 39 | 1 | 0 | 0 | 0 | 1 | 4 | 39/45=.86 |
| **3** | 0 | 4 | 1 | 0 | 0 | 0 | 0 | 1 | 1/6=.166 |
| **4** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | ½=.5 |
| **5** | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 12/12=1 |
| **6** | 0 | 4 | 1 | 0 | 0 | 22 | 0 | 4 | 22/31=.7096 |
| **7** | 1 | 5 | 1 | 0 | 0 | 2 | 5 | 3 | 5/17=.29411 |
| **8** | 3 | 2 | 1 | 0 | 0 | 1 | 1 | 143 | 143/151=.9470 |

Over all accuracy =227/270=.8407

1. From the confusion matrix we see that overall accuracy we get is 84% which is decent I guess. In this problem we are dealing with class conditional probabilities.

This is clearly seen in the confusion matrix.

In the confusion matrix the diagonal elements are total correct predictions and all other elements are wrong predictions made by the model.

So there might be few words/attributes that are more probable in occurring in a particular class than in other. For example there were totally 151 training examples that belonged to class 8. Out of the 151 examples the model predicted 143 examples as class 8 and other 8 examples as other classes. This might be because there might be few words whose class conditional probability is higher for other class than class 8 because of which they were misclassified. This reasoning holds true for other classes too.

So to get better accuracy maybe we could have further pre-processed the data such that the probability of attribute in class is representative of the class.

Question 3.

1. Submitted the code on github: <https://github.ccs.neu.edu/cs6140-03-spring2017/kattaseshasai09>
2. With respect to data I have used it as-is. I have represented the given data as a matrix in numpy. Rows represent each training/test example while columns represent the whole lexicon. I have set the learning rate by trial and error. I have tried random learning rates and for random number of rounds and which ever gave me highest accuracy I chose that value. This might not be interesting but this is how I choose the learning rate and number of rounds. For a learning rate of .015625 and rounds of 20 I was getting accuracy of 92.
3. Confusion Matrix for

|  |  |  |  |
| --- | --- | --- | --- |
| Class | 2.0 | 6.0 | Accuracy |
| 2.0 | 41 | 4 | 41/45 |
| 6.0 | 2 | 29 | 29/31 |

Over all accuracy = 70/76=.9210

1. From the confusion matrix we see that overall accuracy we get is 92% which is good I guess. In this problem we are dealing with learning the weights.

In the confusion matrix the diagonal elements are the correct predictions and all other elements are wrong predictions made by the model.

Model used to make predictions is

σ(x) = 1/ 1+exp(−wT x−w0)

we have used the gradient descent algorithm to estimates the weights for the model by minimizing the loss.

Each of the weight is associated with the attribute. It is difficult to get 100% accuracy but maybe by using more training examples we could have estimated the weights which could have possibly resulted in better accuracy.