Sanket Nitin Wagh – A20330391

Problem Statement:

To perform experiments using datasets based on Generative Learning and Gaussian Discriminant Analysis. Perform cross validation to check and compare results. To determine the accuracy of the model used. Parameters were changed accordingly so as to understand how the model works.

Proposed Solution:

To first use entire data. Determine results for that. Followed 10 fold cross validation to test on a portion of the data used and check the results. Determine accuracy of the model used along with Precision and Recall values for each class. Use confusion matrix to obtain results.

Implementation Details:

Step by step procedure used to determine values. Very few inbuilt functions used which do not come in the middle of what is actually being asked. Each component is performed on a separate file.

Results and Discussion:

1. 1 Dimension – 2 Class.

For this I have used Iris Data set. For the feature I have used Sepal Length which is the first feature of the four given features in iris and is stored in X. The class labels are fed to labels list Y. 2 classes have been used: iris-setosa (Class0) and iris-versicolor(Class1).

The alpha, mean and covariance values are as follows:

```
count [50, 50]
alpha: [0.5, 0.5]

mean of classes: [3.4180000000000006, 2.7700000000000005]
covariance: [0.14227600000000001, 0.096500000000000016]
```

Membership function defined as: $g_j(x) = -\log(\sigma_j) – (X-\mu_j)^2/2\sigma_j^2 + \log(\alpha_j)$

Discriminant function is basically checking which membership function is better via comparison.

Test error before cross validation: `0.664068347083`
Test error after cross validation: `0.586579871235`

Confusion matrix for entire data:

```
  0   1
0 [ 0 50]
1 [ 0 50]
```

This means that all the elements in class0 have been incorrectly classified. Hence accuracy of the model is 50%. I think it will remain the same after cross validation.

➔ Values after cross validation. 10-fold cross validation performed with random shuffling set to true:

```
Confusion matrix obtained for every fold; for each of the class.
[[[0 7]
  [0 3]]
 [[0 6]
  [0 4]]
 [[0 5]
  [0 5]]
 [[0 8]
  [0 2]]
 [[0 3]
  [0 7]]
 [[0 5]
  [0 5]]
 [[0 3]
  [0 7]]
 [[0 4]
  [0 6]]
 [[0 5]
  [0 5]]
 [[0 4]
  [0 6]]]
```

Using which we predict on unknown test data. Use the predictions to determine membership functions and obtain important results.

Precision values for class0 and class1 respectively:

```
[[0, 0.5], [0, 0.1], [0, 0.3], [0, 0.6], [0, 0.5], [0, 0.8], [0, 0.6], [0, 0.
5], [0, 0.5], [0, 0.6]]
Out[36]:
      [iris-setosa; iris-versicolor]
array([ 0. ,  0.5])
```

Recall values for class0 and class1 respectively:

```
[[0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0
, 1.0], [0.0, 1.0], [0.0, 1.0], [0.0, 1.0]]
Out[38]:
      [iris-setosa; iris-versicolor]
array([ 0.,  1.])
```

From this we can say that class1 classification has performed better. Using these values we will now determine Fmeasure values:

```
[[0, 0.6666666666666666], [0, 0.18181818181818182], [0, 0.4615384615384615],
[0, 0.7499999999999999], [0, 0.6666666666666666], [0, 0.888888888888889], [0,
0.7499999999999999], [0, 0.6666666666666666], [0, 0.6666666666666666], [0, 0.
7499999999999999]]
Out[40]:
      [iris-setosa; iris-versicolor]
array([ 0.        ,  0.64489122])
```

We can observe that class1 provides fmeasure score of 0.65 approximately. Class0 is 0 since it has been incorrectly been classified in the first place.

Accuracy of the model is 0.5 , i.e., 50% as expected.

Everytime the program is run due to random shuffling values will tend to change but the results remain more or less the same. Accuracy of the model remains to be 0.5.

2. N-Dimension 2-Class:

For this also I have used Iris Data set. For the features X, I have used all the four given features in iris and is stored in X. The class labels are fed to labels list Y. 2 classes have been used: iris-setosa (Class0) and iris-versicolor(Class1).

The alpha, mean and covariance values are as follows:

```
alpha: [0.5, 0.5]
```

Mean values are obtained for each of the features. I have put it as a mean set:

There are 4 mean values (μ) for each feature, two sets of mean values for each class.

```
                      [SepalLength; SepalWidth; PetalLength; PetalWidth]
Class0 mean set:      [ 5.006  3.418  1.464  0.244]
Class1 mean set:      [ 5.936  2.77   4.26   1.326]
```

Since this is the case we will now have two **covariance matrices instead of values as follows:**

Covariance matrix for class0:

```
                       [SepalLength; SepalWidth; PetalLength; PetalWidth]
SepalLength            [[ 0.121764  0.098292  0.015816  0.010336]
SepalWidth             [ 0.098292  0.142276  0.011448  0.011208]
PetalLength            [ 0.015816  0.011448  0.029504  0.005584]
PetalWidth             [ 0.010336  0.011208  0.005584  0.011264]]
```

Covariance matrix for class1:

```
                       [SepalLength; SepalWidth; PetalLength; PetalWidth]
SepalLength            [[ 0.261104  0.08348   0.17924   0.054664]
SepalWidth              [ 0.08348   0.0965    0.081     0.04038 ]
PetalLength             [ 0.17924   0.081     0.2164    0.07164 ]
PetalWidth              [ 0.054664  0.04038   0.07164   0.038324]]
```

The confusion matrix obtained using the entire data:

```
  0   1
0 [50  0]
1 [ 0 50]
```

This means that the predictions will be better than the ones in the previous experiment.

- ➔ **Values after cross validation. 10-fold cross validation performed with random shuffling set to true:**

```
Confusion matrix obtained for every fold; for each of the class.

[[[6 0]
  [0 4]]
 [[5 1]
  [0 4]]
 [[5 2]
  [0 3]]
 [[2 2]
  [0 6]]
 [[3 0]
  [0 7]]
 [[4 0]
  [0 6]]
 [[5 0]
  [0 5]]
 [[6 0]
```

```
  [0 4]]
 [[2 0]
  [0 8]]
 [[7 0]
  [0 3]]]
```

Precision values for class0 and class1 respectively:

```
[[1.0, 1.0], [1.0, 0.8], [1.0, 0.6], [1.0, 0.75], [1.0, 1.0], [1.0, 1.0], [1.
0, 1.0], [1.0, 1.0], [1.0, 1.0], [1.0, 1.0]]
Out[81]:
array([ 1.   ,  0.915])
```

Recall values for class0 and class1 respectively:

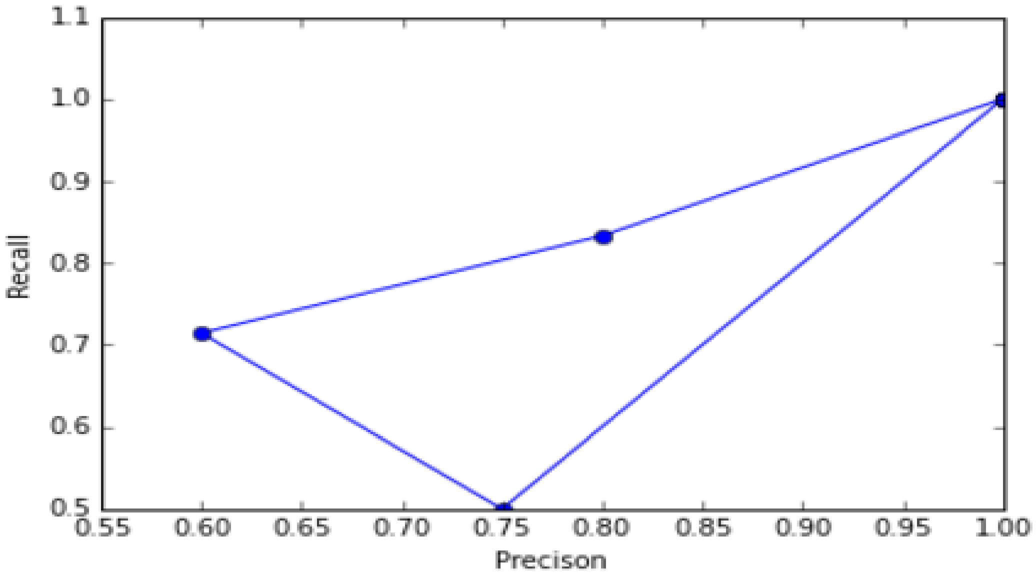
```
[[1.0, 1.0], [0.8333333333333334, 1.0], [0.7142857142857143, 1.0], [0.5, 1.0]
, [1.0, 1.0], [1.0, 1.0], [1.0, 1.0], [1.0, 1.0], [1.0, 1.0], [1.0, 1.0]]
Out[83]:
            [iris-setosa; iris-versicolor]
        array([ 0.9047619,  1.        ])
```

From this we can say that class1 classification has performed better. Using these values we will now determine Fmeasure values:

```
[[1.0, 1.0], [0.9090909090909091, 0.888888888888889], [0.8333333333333333, 0.
7499999999999999], [0.6666666666666666, 0.8571428571428571], [1.0, 1.0], [1.0
, 1.0], [1.0, 1.0], [1.0, 1.0], [1.0, 1.0], [1.0, 1.0]]
Out[85]:
      [iris-setosa; iris-versicolor]
        array([ 0.94090909,  0.94960317])
```

Accuracy of the model after cross validation is 0.95 , i.e., 95%. This shows that having more features is clearly better for generative learning.

The precision vs recall graph is as shown:



Area under the graph is 0.95749999999999991
This value is equal to the accuracy obtained for the model.

3. N-Dimension N-Class:

For this I have used Iris Data set. For the feature I have used Sepal Length which is the first feature of the four given features in iris and is stored in X. The class labels are fed to labels list Y. All 3 classes have been used: iris-setosa (Class0), iris-versicolor(Class1) and iris-virgininca(Class2)

The alpha, mean and covariance values are as follows:

```
Alpha: [0.3333333333333333, 0.3333333333333333, 0.3333333333333333]
```

There are 4 mean values (μ) for each feature, two sets of mean values for each class.

```
            [SepalLength; SepalWidth; PetalLength; PetalWidth]

Class0:     [ 5.006  3.418  1.464  0.244]

Class1:     [ 5.936  2.77   4.26   1.326]

Class2:     [ 6.588  2.974  5.552  2.026]
```

Covariance values:

Class0 – iris-setosa:

```
                [SepalLength; SepalWidth; PetalLength; PetalWidth]
SepalLength     [[ 0.121764  0.098292  0.015816  0.010336]
SepalWidth       [ 0.098292  0.142276  0.011448  0.011208]
PetalLength      [ 0.015816  0.011448  0.029504  0.005584]
PetalWidth       [ 0.010336  0.011208  0.005584  0.011264]]
```

Class1 – iris-versicolor

```
                [SepalLength; SepalWidth; PetalLength; PetalWidth]
SepalLength     [[ 0.261104  0.08348   0.17924   0.054664]
SepalWidth      [ 0.08348   0.0965    0.081     0.04038 ]
PetalLength     [ 0.17924   0.081     0.2164    0.07164 ]
PetalWidth      [ 0.054664  0.04038   0.07164   0.038324]]
```

Class2 – iris-virginica

```
                [SepalLength; SepalWidth; PetalLength; PetalWidth]
SepalLength     [[ 0.396256  0.091888  0.297224  0.048112]
SepalWidth       [ 0.091888  0.101924  0.069952  0.046676]
PetalLength      [ 0.297224  0.069952  0.298496  0.047848]
PetalWidth       [ 0.048112  0.046676  0.047848  0.073924]]
```

Confusion matrix of the entire data

```
         0    1   2
        0[50,  0,  0]
        1[ 0, 48,  2]
        2[ 0,  1, 49]
```

Accuracy of the model with entire data: 98%

- ➔ **Results after Cross Validation (10 fold with random shuffling):**

Confusion matrices obtained are present in the ipynb notebook. Here is an example of a matrix obtained after a fold:

```
  0  1  2
0[6 0 1]
1[0 2 4]
2[0 0 2]
```

Precision values for class0, class1, class2 respectively:

```
[[1.0, 1.0, 0.5714285714285714], [1.0, 1.0, 0.2857142857142857], [1.0, 1.0, 0
.6], [1.0, 1.0, 0.875], [1.0, 1.0, 0.6666666666666666], [1.0, 0, 0.6666666666
666666], [1.0, 1.0, 0.8888888888888888], [1.0, 1.0, 0.42857142857142855], [1.
0, 1.0, 0.7777777777777778], [1.0, 1.0, 0.6666666666666666]]
Out[40]:
array([ 1.        ,  0.9       ,  0.6427381])
```

Recall values for class0, class1, class2 respectively:

```
[[1.0, 0.5714285714285714, 1.0], [0.8571428571428571, 0.3333333333333333, 1.0
], [1.0, 0.7142857142857143, 1.0], [1.0, 0.8333333333333334, 1.0], [1.0, 0.75
, 1.0], [1.0, 0.0, 1.0], [1.0, 0.6666666666666666, 1.0], [0.75, 0.5, 1.0], [1
.0, 0.3333333333333333, 1.0], [0.8333333333333334, 0.3333333333333333, 1.0]]
Out[42]:
array([ 0.94404762,  0.50357143,  1.        ])
```

F-measure values for class0, class1, class2 respectively:

```
[[4.0, 2.2857142857142856, 4.0], [3.4285714285714284, 1.3333333333333333, 4.0
], [4.0, 2.857142857142857, 4.0], [4.0, 3.3333333333333335, 4.0], [4.0, 3.0,
4.0], [4.0, 0, 4.0], [4.0, 2.6666666666666665, 4.0], [3.0, 2.0, 4.0], [4.0, 1
.3333333333333333, 4.0], [3.3333333333333335, 1.3333333333333333, 4.0]]
Out[45]:
array([ 3.77619048,  2.01428571,  4.        ])
```

The accuracy of the model is: `0.826666666667`

Results and conclusion:

- **From the above experiments we can infer that when single feature is used the results are definitely inconclusive. We cannot say that the model is bad even though the accuracy seems to be very bad. This is so because only one feature is being used. The result could have been better or worse based on which feature is used.**
- **When it comes to the second part, we notice there is a vast improvement. This is so since there is more data available to cross validate. This cannot be considered as an overall conclusion since we are using a small dataset.**
- **We can deduce from the second part's confusion matrix on whole data that the classification is very accurate. This can be said to be the same for the third experiments where all the classes are used.**
- **The last part where we use all features and all classes provides the most conclusive r esult. We can say the overall the model has performed good but not excellent.**
- **Hence we can conclude by saying that the model developed is good enough to predict new/unknown instances of data.**

4. Naïve Bayes Gaussian Discriminant Analysis: Bernoulli Distribution:

The procedure followed here is to determine the likelihood (Alpha) and use it to define the membership function g(x). We use the formula $g_j(x) = [\sum_{j=1}^{m} Xjlog(\alpha_{j|y=i}) + (1-X_j)\log(1- \alpha_{j|y=i})] + \log(\alpha_i)$ to determine membership functions for the two classes.

I have used Spambase dataset which determines whether a mail is spam or not. The likelihood is calculated using and $\alpha_{j|y=i} = \sum_{j=1}^{m} I(y=i) \,.Xj^{(i)} / \sum_{j=1}^{m} I(y=i)$, i.e., likelihood of i given y=1.

The prior probability value is:

```
Class0: 0.605955227125
Class1: 0.394044772875
```

Confusion matrix using entire data:

```
  0     1
0[2591  197]
1[ 305 1508]
```

From this we can say that the classification good with a few elements misclassified.

Results after 10 fold cross validation:
An example of confusion matrix after a single fold for class 0 is:

```
 0     1
0[270,  16]
1[ 23, 152]
```

An example of confusion matrix after a single fold for class 1 is:

```
0      1
0[263,  26]
1[ 30, 141]
```

Both classification is not 100% accurate, but is quite good.

Precision Values are:
array([0.89421012, 0.88457924]) … for [class0, class1] respectively.

Recall Values are:
array([0.92922302, 0.83269602]) … for [class0, class1] respectively.

Fmeasure values are:
array([0.91127255, 0.85764651]) … for[class0, class1] respectively.

Accuracy of the model is: 0.890887956239

5. Naïve Bayes Gaussian Discriminant Analysis: Binomial Distribution:

In this part I have used the same dataset hence the prior and alpha values are the same. Confus ion matrix for the entire data also remains the same. The only difference is that after cross valid ation there is a slight improvement since we are considering the wordcount. In this case the wo rd count has been replaced by randomizing the data given. This will work as a wordcount as we use the frequency of ones and zeroes.

Likelihood is calculated using : $\alpha_{j|y=i} = \sum_{j=1}^{m} I(y = i) . Xj^{(i)} / \sum_{j=1}^{m} I(y = i).P$

The membership function is determined using: $g_j(x) =[\sum_{j=1}^{n} \log \binom{P}{Xj} . B_1. B_2] + \log(\alpha_l)$

```
The accuracy of the model is 0.890892671885
The precision, recall fmeasure values respectively are (for class0 and
class1 respectively) :
array([ 0.89418221,  0.88515246])
array([ 0.93056853,  0.82942413])
array([ 0.91195348,  0.85621282])
```

This is more or less the same as the accuracy obtained for Bernoulli distribution. Performance will vary based on the parameters input, size of the data and most importantly the type of data used.

→ Derivation

Parameter estimate equation for Naive Bayes with Binomial features

→ Parameter given by : $\alpha_{j|y=1}$ for $j = 1 \,---\, n$

$\alpha_i = P(y=i)$

$l(\alpha) = \text{Log}\, P(X_i^{(1)} \,---\, x_i^{(m)} \mid \alpha)$

The samples are assumed to be IID samples

then $\text{Log} \prod_{i=1}^{n} P(X^{(i)} \mid \alpha)$

→ Since distribution is Binomial (say)

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \log \binom{P^{(i)}}{x_j^{(i)}} \alpha_{p|y=y^{(i)}}^{x_j^{(i)}} \cdot (1-\alpha_{p|y=y^{(i)}})^{P-x_j^{(i)}}$$

$$\Rightarrow \sum_{i=1}^{m} \sum_{j=1}^{n} \binom{P^{(i)}}{x_j^{(i)}} x_j^{(i)} \log \alpha_{p|y=y^{(i)}} + (P - x_j^{(i)}) \log(1-\alpha_{p|y=y^{(i)}})$$

→ To maximize likelihood 'α' :

$$\frac{\partial l}{\partial \alpha_{p(y=1}} = 0$$

$$= \sum_{p=1}^{n} \frac{\partial}{\partial \alpha_{k|u=1}} x_k^{(i)} \cdot \log \alpha_{k|y=1} + P - X_k^{(i)} \log(1-\alpha_{k|y=1}) = 0$$

$$= \sum_{i=1}^{m} X_k^{(i)} \frac{1}{\alpha_{k|y=1}} + P - X_k^{(i)} \frac{1}{1-\alpha_{k|y=1}} (-1) = 0$$

$$\Rightarrow \frac{1}{\alpha_{k|y=1}} \sum_{i=1}^{m} X_k^{(i)} = \frac{1}{1-\alpha_{k|y=1}} \sum P - X_k^{(i)}$$

$$\Rightarrow \alpha_{k|y=1} = \sum_{i=1}^{m} \frac{X_k^{(i)}}{X_k^{(i)} + P - X_k^{(i)}}$$

$$\boxed{\alpha_{k|y=1} = 1 \sum^{n} X_k^{(i)}}$$