Indian Institute of Technology, Kanpur

Data Science Lab (MTH312A) HOME WORK-4 REPORT

Group Members:
Mrinmoy Saha (221352)
Sudesh Kumari (221440)
Vijay Soren (211163)
Manjeet Chaudhary (221346)
Pravin Raman Bharti (221375)

Question 1: Download Heart Disease and Breast Cancer Wisconsin (Diagnostic) data from `https://archive.ics.uci.edu/datasets`. Apply depth-based classifier, SVM-based classifier, K-NN-based classifier (choose K with proper justification), Kernel density function-based classifier on the aforementioned two data sets and compute empirical misclassification probability for each classifier. Note that the final result will depend on the choice of training and test data set.

Answer :

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or nonlinear classification, regression, and even outlier detection tasks. SVMs can be used for a variety of tasks, such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection. SVMs are adaptable and efficient in a variety of applications because they can manage high-dimensional data and nonlinear relationships.

SVM algorithms are very effective as we try to find the maximum separating hyperplane between the different classes available in the target feature. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three. Let's consider two independent variables x_1 and x_2, and one dependent variable which is either a blue circle or a red circle. From the figure above,

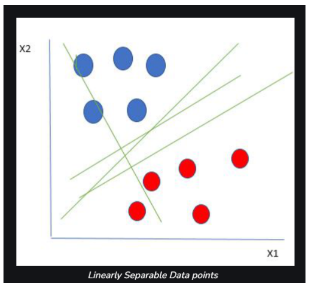


Figure 1: Linearly Separable Data points

it's very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x_1 and x_2) that segregate our data points or do a classification between red and blue circles. So how do we choose the best line or, in general, the best hyperplane that segregates our data points?

How does SVM work? One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes. So we choose the

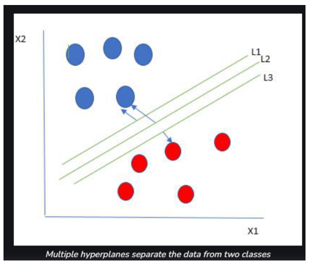
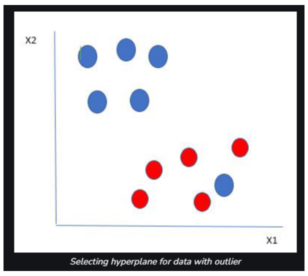


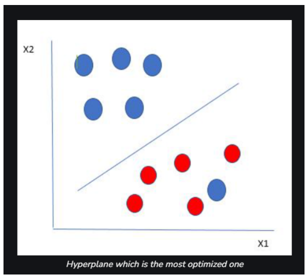
Figure 2: Multiple hyperplanes separate the data from two classes

hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane/hard margin. So from the above figure, we choose L_2.

Let's consider a scenario like shown below:



(a) Selecting hyperplane for data with outlier



(b) Hyperplane which is the most optimized one

Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of the red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.

So in this type of data point, what SVM does is it finds the maximum margin as done with previous data sets, along with that it adds a penalty each time a point crosses the margin. So the margins in these types of cases are called soft margins. When there is a soft margin to the data set, the SVM tries to minimize $\left(\frac{1}{\text{margin}} + \lambda \sum \text{penalty}\right)$. Hinge loss is a commonly used penalty. If there are no violations, there is no hinge loss. If there are violations, the hinge loss is proportional to the distance of the violation.

Till now, we were talking about linearly separable data (the group of blue balls and red balls are separable by a straight line/linear line). What do we do if the data are not linearly separable? Say, our data is shown in the figure above. SVM solves this by

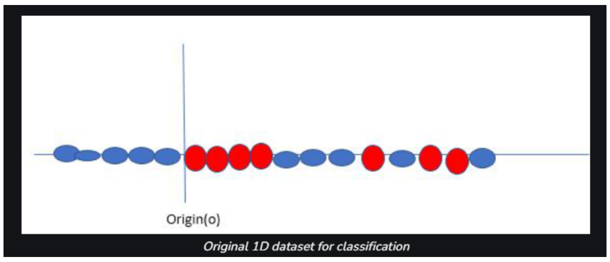


Figure 4: Original 1D dataset for classification

creating a new variable using a kernel. We call a point xi on the line and we create a new variable yi as a function of distance from origin o.so if we plot this we get something like as shown below. In this case, the new variable y is created as a function of distance from

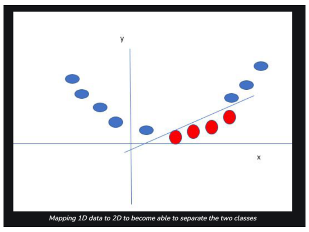

y
x
Mapping 1D data to 2D to become able to separate the two classes

Figure 5: Mapping 1D data to 2D to become able to separate the two classes

the origin. A non-linear function that creates a new variable is referred to as a kernel.

Mathematical intuition of Support Vector Machine:

Consider a binary classification problem with two classes, labeled as $+1$ and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyperplane can be written as:

$$w^T x + b = 0$$

The vector w represents the normal vector to the hyperplane, i.e., the direction perpendicular to the hyperplane. The parameter b in the equation represents the offset or distance of the hyperplane from the origin along the normal vector w.

The distance between a data point x_i and the decision boundary can be calculated as:

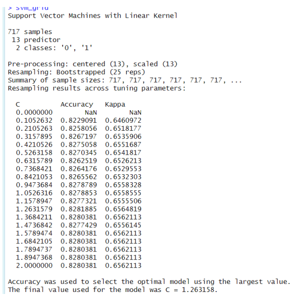
$$d_i = \frac{w^T x_i + b}{\|w\|}$$

where $\|w\|$ represents the Euclidean norm of the weight vector w.

For a Linear SVM classifier:

$$\hat{y} = \begin{cases} 1 & : \ w^T x + b \geq 0 \\ 0 & : \ w^T x + b < 0 \end{cases}$$

Model Description for heart data:



```
> svm_grid
Support Vector Machines with Linear Kernel

717 samples
 13 predictor
  2 classes: '0', '1'

Pre-processing: centered (13), scaled (13)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 717, 717, 717, 717, 717, 717, ...
Resampling results across tuning parameters:

  C          Accuracy   Kappa
  0.0000000        NaN        NaN
  0.1052632  0.8229091  0.6460972
  0.2105263  0.8258056  0.6518177
  0.3157895  0.8267197  0.6535906
  0.4210526  0.8275058  0.6551687
  0.5263158  0.8270345  0.6541817
  0.6315789  0.8262519  0.6526213
  0.7368421  0.8264176  0.6529553
  0.8421053  0.8265562  0.6532303
  0.9473684  0.8278789  0.6558328
  1.0526316  0.8278853  0.6558555
  1.1578947  0.8277321  0.6555506
  1.2631579  0.8281885  0.6564819
  1.3684211  0.8280381  0.6562113
  1.4736842  0.8277429  0.6556145
  1.5789474  0.8280381  0.6562113
  1.6842105  0.8280381  0.6562113
  1.7894737  0.8280381  0.6562113
  1.8947368  0.8280381  0.6562113
  2.0000000  0.8280381  0.6562113

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 1.263158.
```

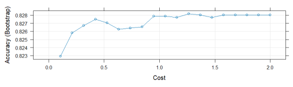
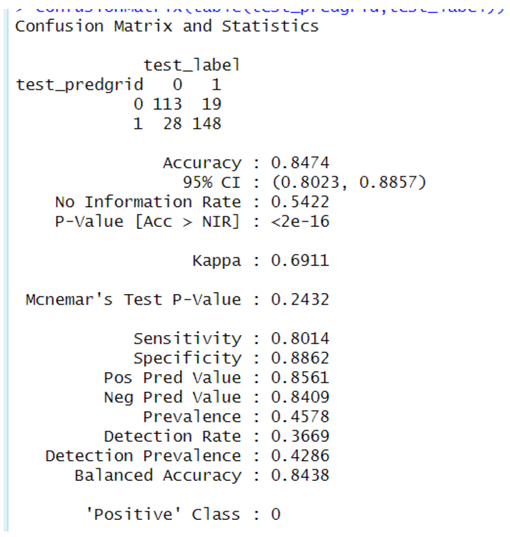


Figure 6: Plot of "C" cost vs Accuracy

Plot of "C" cost vs Accuracy

Confusion Matrix



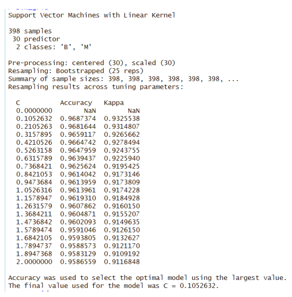
```
Confusion Matrix and Statistics

             test_label
test_predgrid   0   1
            0 113  19
            1  28 148

               Accuracy : 0.8474
                 95% CI : (0.8023, 0.8857)
    No Information Rate : 0.5422
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6911

 Mcnemar's Test P-Value : 0.2432

            Sensitivity : 0.8014
            Specificity : 0.8862
         Pos Pred Value : 0.8561
         Neg Pred Value : 0.8409
             Prevalence : 0.4578
         Detection Rate : 0.3669
   Detection Prevalence : 0.4286
      Balanced Accuracy : 0.8438

       'Positive' Class : 0
```

Figure 7: Confusion Matrix

Conclusion:

Final value of C =1.263158 with Accuracy = 0.8281
Probability of Misclassification = 28/141 + 19/167 = 0.312354

Model description for Breast Cancer data



```
Support Vector Machines with Linear Kernel

398 samples
 30 predictor
  2 classes: 'B', 'M'

Pre-processing: centered (30), scaled (30)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 398, 398, 398, 398, 398, 398, ...
Resampling results across tuning parameters:

  C          Accuracy   Kappa
  0.0000000        NaN        NaN
  0.1052632  0.9687374  0.9325538
  0.2105263  0.9681644  0.9314807
  0.3157895  0.9659117  0.9265662
  0.4210526  0.9664742  0.9278494
  0.5263158  0.9647959  0.9243755
  0.6315789  0.9639437  0.9225940
  0.7368421  0.9625624  0.9195425
  0.8421053  0.9614042  0.9173146
  0.9473684  0.9613959  0.9173809
  1.0526316  0.9613961  0.9174228
  1.1578947  0.9619310  0.9184928
  1.2631579  0.9607862  0.9160150
  1.3684211  0.9604871  0.9155207
  1.4736842  0.9602093  0.9149635
  1.5789474  0.9591046  0.9126150
  1.6842105  0.9593805  0.9132627
  1.7894737  0.9588573  0.9121170
  1.8947368  0.9583129  0.9109192
  2.0000000  0.9586559  0.9116848

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.1052632.
```

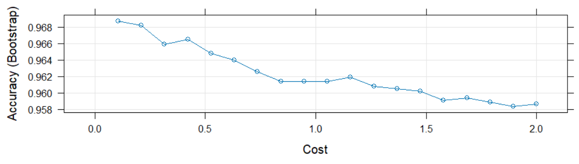
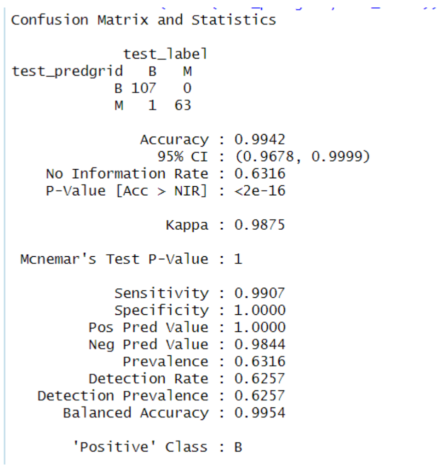


Figure 8: Plot of "C" cost vs Accuracy



```
Confusion Matrix and Statistics

             test_label
test_predgrid   B   M
            B 107   0
            M   1  63

               Accuracy : 0.9942
                 95% CI : (0.9678, 0.9999)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9875

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9907
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9844
             Prevalence : 0.6316
         Detection Rate : 0.6257
   Detection Prevalence : 0.6257
      Balanced Accuracy : 0.9954

       'Positive' Class : B
```

Figure 9: Confusion Matrix

Plot of "C" cost vs Accuracy

Confusion Matrix

Conclusion:

Final value of C = 0.1052632 with Accuracy = 0.9687
Probability of Misclassification = 1/108 = 0.0092

K-Nearest Neighbors:

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

As an example, consider the following table of data points containing two features: Now, given another set of data points (also called testing data), allocate these points to a group by analyzing the training set. Note that the unclassified points are marked as 'White'.

How to choose the value of k for KNN Algorithm?

The value of k is very crucial in the KNN algorithm to define the number of neighbors in the algorithm. The value of k in the k-nearest neighbors (k-NN) algorithm should be chosen based on the input data. If the input data has more outliers or noise, a higher

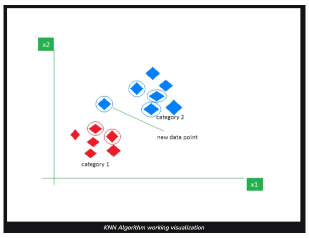
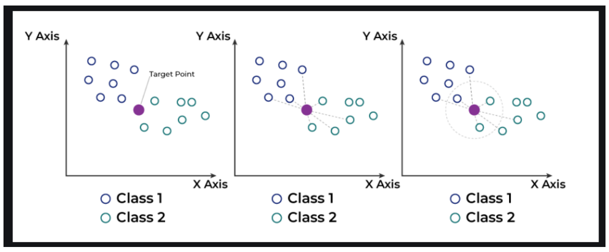


Figure 10: KNN Algorithm working visualization

value of k would be better. It is recommended to choose an odd value for k to avoid ties in classification. Cross-validation methods can help in selecting the best k value for the given dataset.

Workings of KNN algorithm:

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity, where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step-by-Step explanation of how KNN works:

Step 1: Selecting the optimal value of K K represents the number of nearest neighbors that need to be considered while making predictions.

Step 2: Calculating distance To measure the similarity between the target and training data points, Euclidean distance is used. The distance is calculated between each of the data points in the dataset and the target point.

Step 3: Finding Nearest Neighbors The k data points with the smallest distances to the target point are the nearest neighbors.

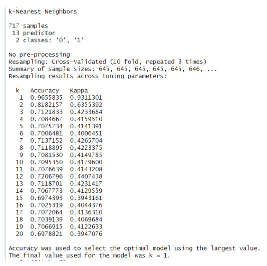
Step 4: Voting for Classification or Taking Average for Regression In the classification problem, the class labels are determined by performing majority voting. The class with the most occurrences among the neighbors becomes the predicted class for the target data point. In the regression problem, the class label is calculated by taking the average of the target values of the K nearest neighbors. The calculated average value becomes the predicted output for the target data point.

Let X be the training dataset with n data points, where each data point is represented by a d-dimensional feature vector X_i, and Y be the corresponding labels or values for each data point in X. Given a new data point x, the algorithm calculates the distance between x and each data point X_i in X using a distance metric, such as Euclidean distance:

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{i_j})^2}$$

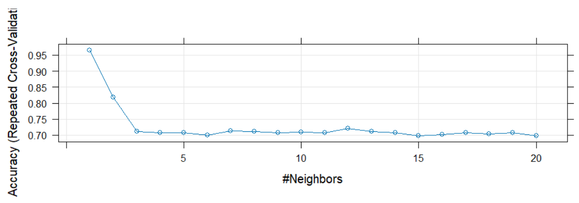
The algorithm selects the K data points from X that have the shortest distances to x. For classification tasks, the algorithm assigns the label y that is most frequent among the K nearest neighbors to x. For regression tasks, the algorithm calculates the average or weighted average of the values y of the K nearest neighbors and assigns it as the predicted value for x.

Model Description for heart data

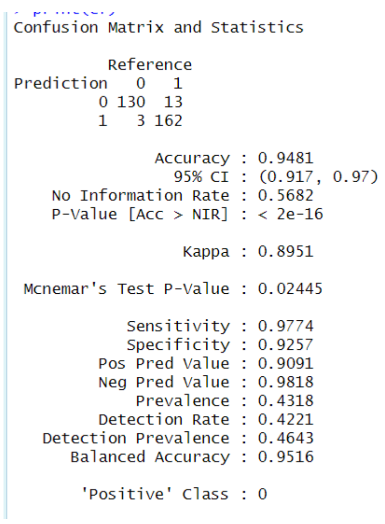


```
k-Nearest Neighbors

717 samples
 13 predictor
  2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 645, 645, 645, 645, 645, 646, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.9655835  0.9311301
   2  0.8182157  0.6355392
   3  0.7121833  0.4233684
   4  0.7084667  0.4159510
   5  0.7075734  0.4141391
   6  0.7006481  0.4006451
   7  0.7137152  0.4265704
   8  0.7118895  0.4223375
   9  0.7081530  0.4149785
  10  0.7095350  0.4179600
  11  0.7076639  0.4143208
  12  0.7206796  0.4407438
  13  0.7118701  0.4231417
  14  0.7067773  0.4129559
  15  0.6974393  0.3943161
  16  0.7025319  0.4044376
  17  0.7072064  0.4136310
  18  0.7039139  0.4069684
  19  0.7066915  0.4122633
  20  0.6978821  0.3947076

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 1.
```

Plot of different K vs accuracy for heart data



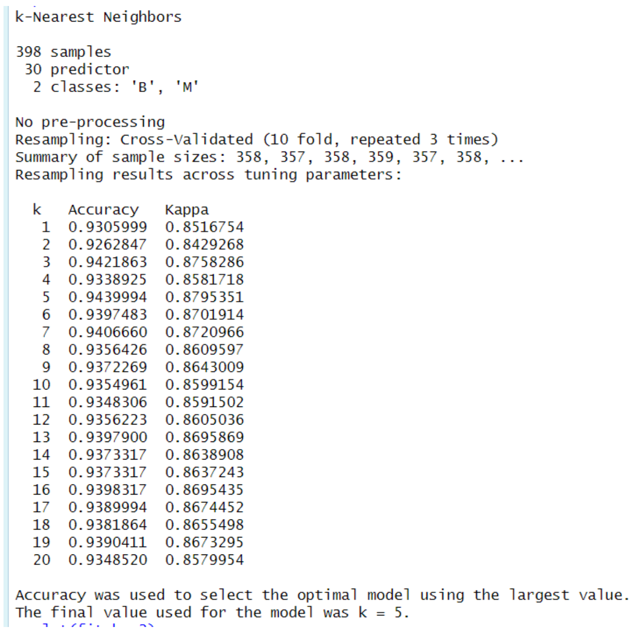
Confusion Matrix



```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 130  13
         1   3 162

               Accuracy : 0.9481
                 95% CI : (0.917, 0.97)
    No Information Rate : 0.5682
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8951

 Mcnemar's Test P-Value : 0.02445

            Sensitivity : 0.9774
            Specificity : 0.9257
         Pos Pred Value : 0.9091
         Neg Pred Value : 0.9818
             Prevalence : 0.4318
         Detection Rate : 0.4221
   Detection Prevalence : 0.4643
      Balanced Accuracy : 0.9516

       'Positive' Class : 0
```

Conclusion:

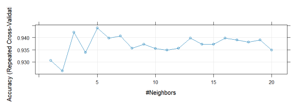
Final value of k = 1 with Accuracy = 0.9655
Probability of Misclassification = 3/133 + 13/175 = 0.096



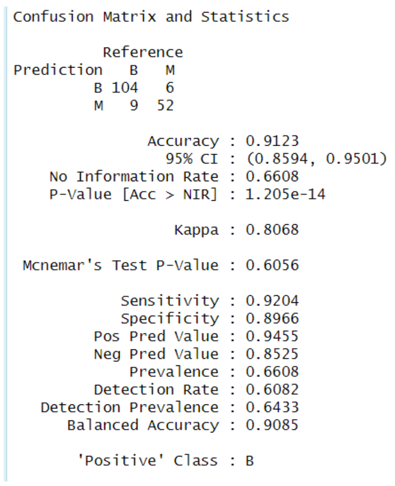
```
k-Nearest Neighbors

398 samples
 30 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 358, 357, 358, 359, 357, 358, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.9305999  0.8516754
   2  0.9262847  0.8429268
   3  0.9421863  0.8758286
   4  0.9338925  0.8581718
   5  0.9439994  0.8795351
   6  0.9397483  0.8701914
   7  0.9406660  0.8720966
   8  0.9356426  0.8609597
   9  0.9372269  0.8643009
  10  0.9354961  0.8599154
  11  0.9348306  0.8591502
  12  0.9356223  0.8605036
  13  0.9397900  0.8695869
  14  0.9373317  0.8638908
  15  0.9373317  0.8637243
  16  0.9398317  0.8695435
  17  0.9389994  0.8674452
  18  0.9381864  0.8655498
  19  0.9390411  0.8673295
  20  0.9348520  0.8579954

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
```

Model Description for Breast Cancer data

Plot of different K vs accuracy for breast cancer data



Confusion Matrix



```
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 104   6
         M   9  52

               Accuracy : 0.9123
                 95% CI : (0.8594, 0.9501)
    No Information Rate : 0.6608
    P-Value [Acc > NIR] : 1.205e-14

                  Kappa : 0.8068

 Mcnemar's Test P-Value : 0.6056

            Sensitivity : 0.9204
            Specificity : 0.8966
         Pos Pred Value : 0.9455
         Neg Pred Value : 0.8525
             Prevalence : 0.6608
         Detection Rate : 0.6082
   Detection Prevalence : 0.6433
      Balanced Accuracy : 0.9085

       'Positive' Class : B
```

Conclusion

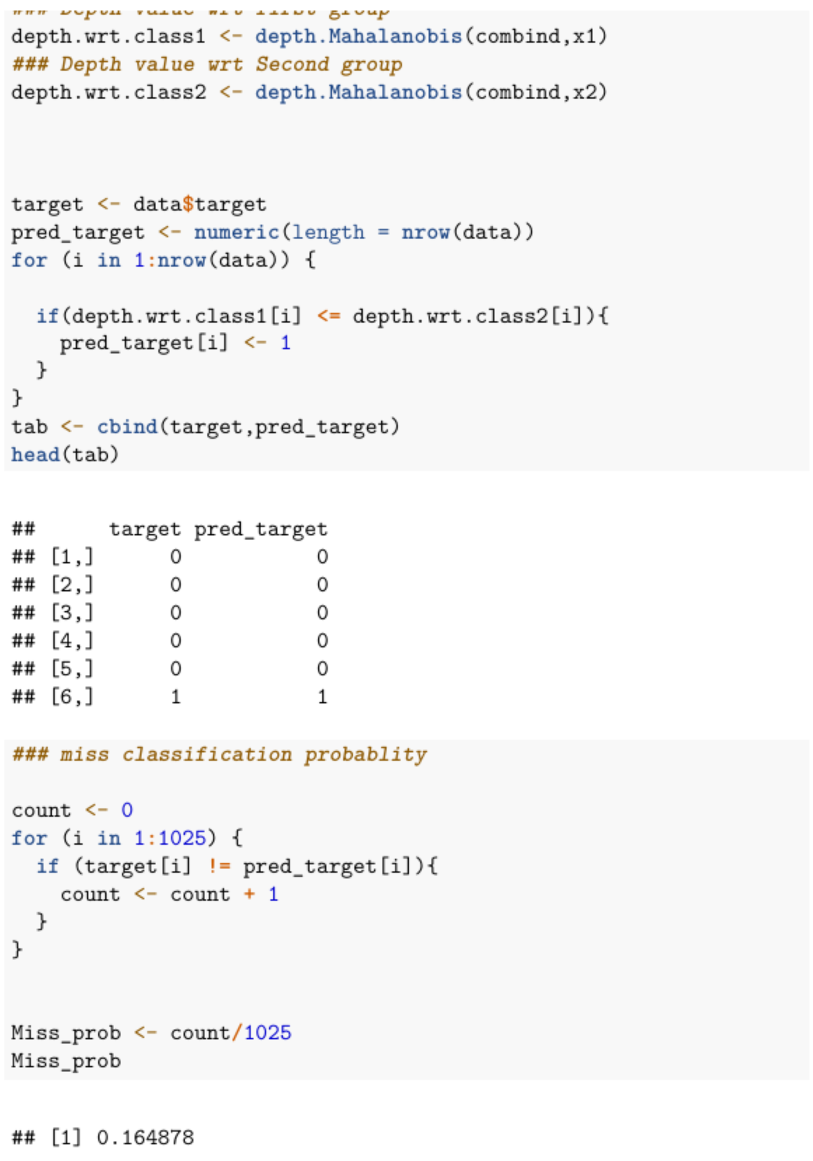
Final value of k $= 5$with Accuracy $= 0.940666$
Probability of Misclassification $= 9/113 + 6/58 = 0.1830$

Data Depth Classifier:

Suppose that $X = \{x_1, \ldots, x_{n_1}\}$ and $Y = \{y_1, \ldots, y_{n_2}\}$ are two data sets obtained from the distribution functions F_1 and F_2, respectively, where F_1 and F_2 are unknown and defined on $\mathbb{R}^d$ $(d \geq 1)$. Let f_1 and f_2 be the probability density functions of F_1 and F_2, respectively. Using these training data sets X and Y, one can classify a future observation x in the following ways.

Depth-based classification: Compute $D_X(x)$ and $D_Y(x)$. Classify x to the class associated with X if $D_X(x) > D_Y(x)$ and classify x to the class associated with Y if $D_Y(x) > D_X(x)$. Here $D_X(x)$ and $D_Y(x)$ denote the depth of x relative to data sets X and Y, respectively, considering that the priors are equal.

for Heart data set:



```
depth.wrt.class1 <- depth.Mahalanobis(combind,x1)
### Depth value wrt Second group
depth.wrt.class2 <- depth.Mahalanobis(combind,x2)



target <- data$target
pred_target <- numeric(length = nrow(data))
for (i in 1:nrow(data)) {

  if(depth.wrt.class1[i] <= depth.wrt.class2[i]){
    pred_target[i] <- 1
  }
}
tab <- cbind(target,pred_target)
head(tab)
```

```
##      target pred_target
## [1,]      0           0
## [2,]      0           0
## [3,]      0           0
## [4,]      0           0
## [5,]      0           0
## [6,]      1           1
```

```
### miss classification probablity

count <- 0
for (i in 1:1025) {
  if (target[i] != pred_target[i]){
    count <- count + 1
  }
}


Miss_prob <- count/1025
Miss_prob
```

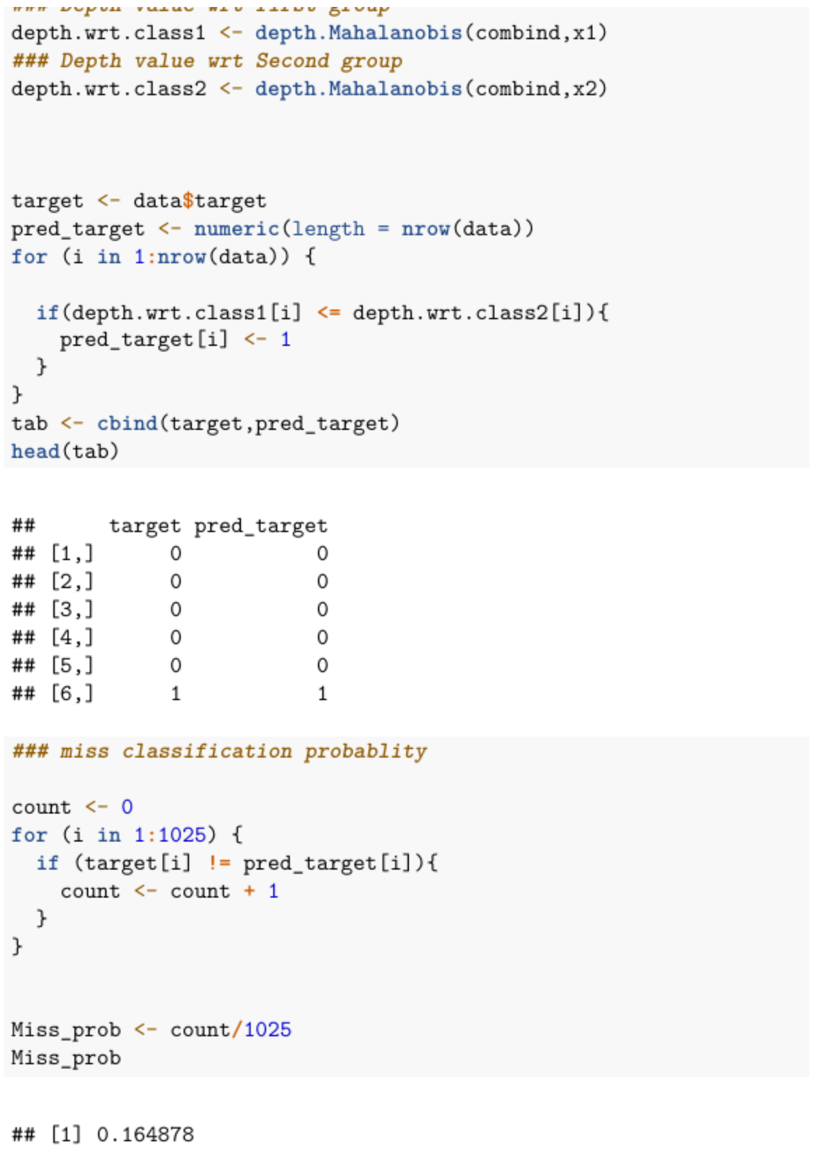
```
## [1] 0.164878
```

Figure 11: Probabiliry of Misclassification for heart data set

Conclusion

Probability of Misclassification = 0.164878 with Accuracy = 0.835122

for Cancer Breast data



```
depth.wrt.class1 <- depth.Mahalanobis(combind,x1)
### Depth value wrt Second group
depth.wrt.class2 <- depth.Mahalanobis(combind,x2)



target <- data$target
pred_target <- numeric(length = nrow(data))
for (i in 1:nrow(data)) {

  if(depth.wrt.class1[i] <= depth.wrt.class2[i]){
    pred_target[i] <- 1
  }
}
tab <- cbind(target,pred_target)
head(tab)
```

```
##      target pred_target
## [1,]      0           0
## [2,]      0           0
## [3,]      0           0
## [4,]      0           0
## [5,]      0           0
## [6,]      1           1
```

```
### miss classification probablity

count <- 0
for (i in 1:1025) {
  if (target[i] != pred_target[i]){
    count <- count + 1
  }
}


Miss_prob <- count/1025
Miss_prob
```

```
## [1] 0.164878
```

Figure 12: Probability of Misclassification for heart data set

Conclusion

Probability of Misclassification = 0.05560976 with Accuracy = 0.944391

Kernel densiry Estimation

In statistics, kernel density estimation (KDE) is the application of kernel smoothing for probability density estimation, i.e., a non-parametric method to estimate the probability density function of a random variable based on kernels as weights. KDE answers a fundamental data smoothing problem where inferences about the population are made based on a finite data sample. Let $(x_1, x_2, \ldots, x_n)$ be independent and identically distributed samples drawn from some univariate distribution with an unknown density f at any given point x. We are interested in estimating the shape of this function f. Its kernel density estimator is given by:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where K is the kernel—a non-negative function—and $h > 0$ is a smoothing parameter called the bandwidth. A kernel with subscript h is called the scaled kernel and defined as $K_h(x) = \frac{1}{h}K\left(\frac{x}{h}\right)$. Intuitively, one wants to choose h as small as the data will allow; however, there is always a trade-off between the bias of the estimator and its variance. The choice of bandwidth is discussed in more detail below.

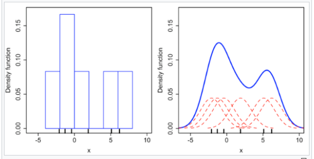
A range of kernel functions are commonly used: uniform, triangular, biweight, triweight, Epanechnikov, normal, and others. The Epanechnikov kernel is optimal in a mean square error sense, though the loss of efficiency is small for the kernels listed previously. Due to its convenient mathematical properties, the normal kernel is often used, which means $K(x) = \phi(x)$, where ϕ is the standard normal density function.

Example

Kernel density estimates are closely related to histograms but can be endowed with properties such as smoothness or continuity by using a suitable kernel. The diagram below, based on these 6 data points, illustrates this relationship:

Sample	Value
1	-2.1
2	-1.3
3	-0.4
4	1.9
5	5.1
6	6.2

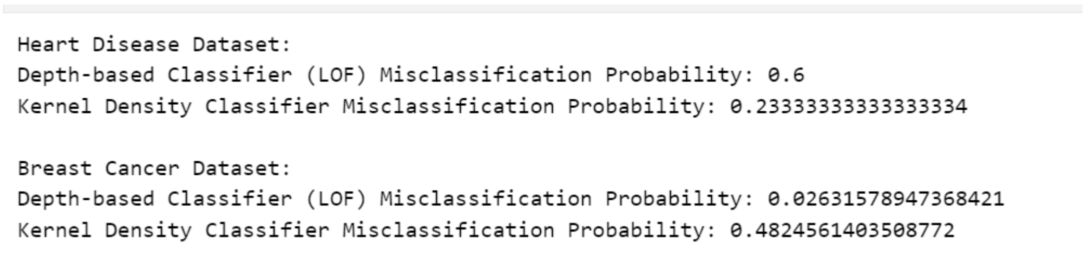
For the histogram, first, the horizontal axis is divided into sub-intervals or bins which cover the range of the data: In this case, six bins each of width 2. Whenever a data point falls inside this interval, a box of height $\frac{1}{12}$ is placed there. If more than one data point falls inside the same bin, the boxes are stacked on top of each other. For the kernel density estimate, normal kernels with a standard deviation of 1.5 (indicated by the red dashed lines) are placed on each of the data points x_i. The kernels are summed to make the kernel density estimate (solid blue curve).



Comparison of the histogram (left) and kernel density estimate (right) constructed using the same data. The six individual kernels are the red dashed curves, the kernel density estimate the blue curves. The data points are the rug plot on the horizontal axis.

Figure 13: Kernel Density Estimation

Conclusion



```
Heart Disease Dataset:
Depth-based Classifier (LOF) Misclassification Probability: 0.6
Kernel Density Classifier Misclassification Probability: 0.23333333333333334

Breast Cancer Dataset:
Depth-based Classifier (LOF) Misclassification Probability: 0.02631578947368421
Kernel Density Classifier Misclassification Probability: 0.4824561403508772
```

Figure 14: Probability of Misclassification for Heart and Brest cancer data set