Lab-10

Implementation of Naive Bayes Classifier, Decision Tree, SVM, and KNN

Objective: The objective of this lab is to understand and implement four different classification algorithms in R: Naive Bayes Classifier, Decision Tree, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN).

Theory:

1. Naive Bayes Classifier:

- The naive Bayes classifier is based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event.
- Despite its "naive" assumption of feature independence, Naive Bayes often performs surprisingly well in practice, especially for text classification and spam filtering.
- The classifier calculates the probability of each class given the input data using the joint probability distribution of the features and the class and then selects the class with the highest probability as the prediction.

2. Decision Tree:

- Decision trees recursively split the data based on feature values to create a tree-like structure of decision nodes and leaf nodes.
- The splits are chosen based on criteria such as Gini impurity or entropy, aiming to maximize the homogeneity (or purity) of the target variable within each subset.
- Decision trees are easy to interpret and visualize, making them useful for gaining insights into the decision-making process.

3. Support Vector Machine (SVM):

- SVM is a powerful classification algorithm that finds the hyperplane in the feature space that best separates the classes.
- It aims to maximize the margin between the closest points (support vectors) of different classes while minimizing the classification error.
- SVM can handle non-linear decision boundaries through techniques like the kernel trick, which maps the input data into a higher-dimensional space where it can be linearly separated.

4. K-Nearest Neighbors (KNN):

- KNN is a simple and intuitive classification algorithm that classifies a new data point based on the majority vote of its k nearest neighbors in the training dataset.

- The choice of k influences the model's bias-variance trade-off, with smaller values of k leading to more complex decision boundaries and potentially overfitting.

- KNN does not require model training, making it computationally inexpensive, but it can be sensitive to the choice of distance metric and the number of neighbors.

Key Concepts:

1. **Bias-Variance Trade-off:** All these algorithms exhibit a trade-off between bias and variance. Naive Bayes has low variance but can suffer from high bias due to its simplifying assumption. Decision trees are prone to overfitting, leading to high variance, especially with deep trees. SVM aims to find the optimal balance between bias and variance by maximizing the margin. KNN's performance depends on the choice of K; smaller K values lead to lower bias but higher variance.

2. **Model Interpretability:** Decision trees and Naive Bayes classifiers offer high interpretability, as their decision-making process can be easily visualized and understood. SVM and KNN, while powerful, are often considered "black-box" models with less interpretability.

3. **Scalability:** Naive Bayes and KNN are typically more scalable than Decision Trees and SVM. Naive Bayes and KNN have linear or near-linear time complexity during prediction, while Decision Trees and SVM can have higher time complexity, especially with large datasets.

Implementation in R:

We will use R programming language and relevant libraries to implement each classification algorithm:

- **For Naive Bayes Classifier**: `naivebayes` package.
- **For Decision Tree:** `rpart` package.
- **For SVM:** `e1071` package.
- **For KNN:** `class` package.

We will follow a structured approach to load the dataset, preprocess the data (if necessary), train the model, make predictions, and evaluate the model's performance using appropriate metrics such as accuracy.

(A) Implementing Naïve Bayes Classifier

```
> # Load required library
> library(rpart)
> # Load the dataset
> data(iris)
> # Split the dataset into training and testing sets
> set.seed(123)
> train_index1 <- sample(1:nrow(iris), 0.7*nrow(iris))
> train_data1 <- iris[train_index1, ]
> test_data1 <- iris[-train_index1, ]
> # Train the decision tree classifier
> dt_model <- rpart(Species ~ ., data = train_data1, method = "class")
> print(dt_model)
n= 105

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 105 68 virginica (0.34285714 0.30476190 0.35238095)
  2) Petal.Length< 2.45 36  0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 69 32 virginica (0.00000000 0.46376812 0.53623188)
    6) Petal.Width< 1.75 35  4 versicolor (0.00000000 0.88571429 0.11428571) *
    7) Petal.Width>=1.75 34  1 virginica (0.00000000 0.02941176 0.97058824) *
> # Make predictions on the testing set
> predictions1 <- predict(dt_model, test_data1, type = "class")
> # Evaluate
> accuracy1 <- sum(predictions1 == test_data1$Species) / nrow(test_data1)
> cat("Accuracy:", accuracy1)
Accuracy: 0.9777778
```

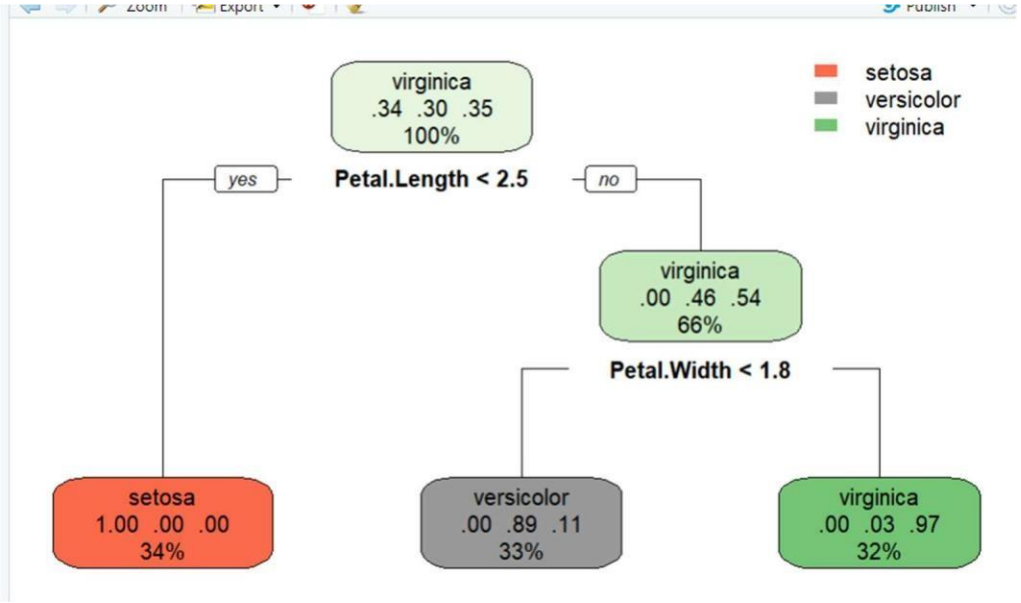
(B) Implementing Decision Tree

```
> #decision tree
> install.packages("rpart")
> install.packages("rpart.plot")
> install.packages("partykit")
package 'libcoin' successfully unpacked and MD5 sums checked
package 'mvtnorm' successfully unpacked and MD5 sums checked
package 'Formula' successfully unpacked and MD5 sums checked
package 'inum' successfully unpacked and MD5 sums checked
package 'partykit' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\1212k\AppData\Local\Temp\RtmpcLueQa\downloaded_packages
> library(rpart)
Warning message:
package 'rpart' was built under R version 4.3.3
> library(rpart.plot)
Warning message:
package 'rpart.plot' was built under R version 4.3.3
> data(iris)
```

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
> set.seed(123)  # for reproducibility
> train_indices <- sample(1:nrow(iris), 0.7 * nrow(iris))
> train_data <- iris[train_indices, ]
> test_data <- iris[-train_indices, ]
> model <- rpart(Species ~ ., data = train_data)
> rpart.plot(model)
```




```
> #test
> predictions <- predict(model, newdata = test_data, type = "class")
> table(predictions, test_data$Species)
```

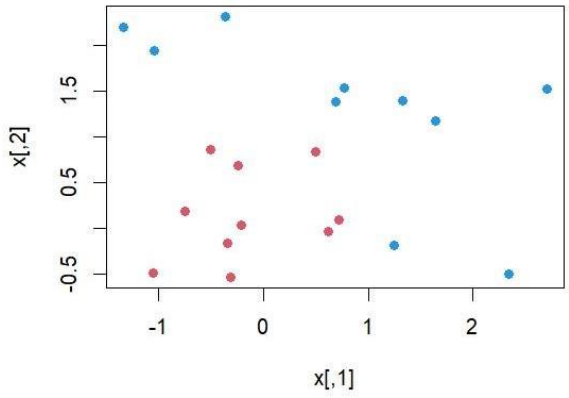
```
predictions  setosa versicolor virginica
  setosa         14          0         0
  versicolor      0         18         1
  virginica       0          0        12
```

```
> accuracy <- sum(predictions == test_data$Species) / nrow(test_data)
> print(paste("Accuracy:", accuracy))
```

```
[1] "Accuracy: 0.977777777777778"
>
```

(C) Implementing SVM (Support Vector Machine)

```
> set.seed(10111)
> x = matrix(rnorm(40), 20, 2)
> y = rep(c(-1, 1), c(10, 10))
> x[y == 1,] = x[y == 1,] + 1
> plot(x, col = y + 3, pch = 19)
```

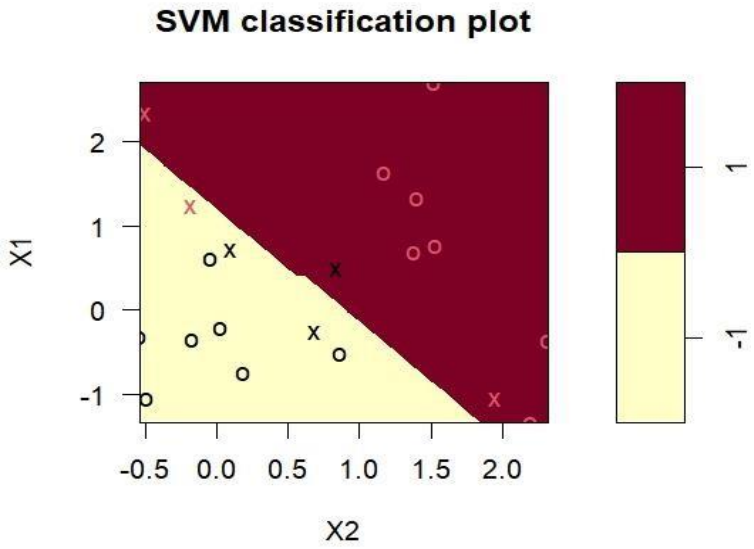



```
> library(e1071)
> dat = data.frame(x, y = as.factor(y))
> svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
> print(svmfit)
Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  6

> plot(svmfit, dat)
```

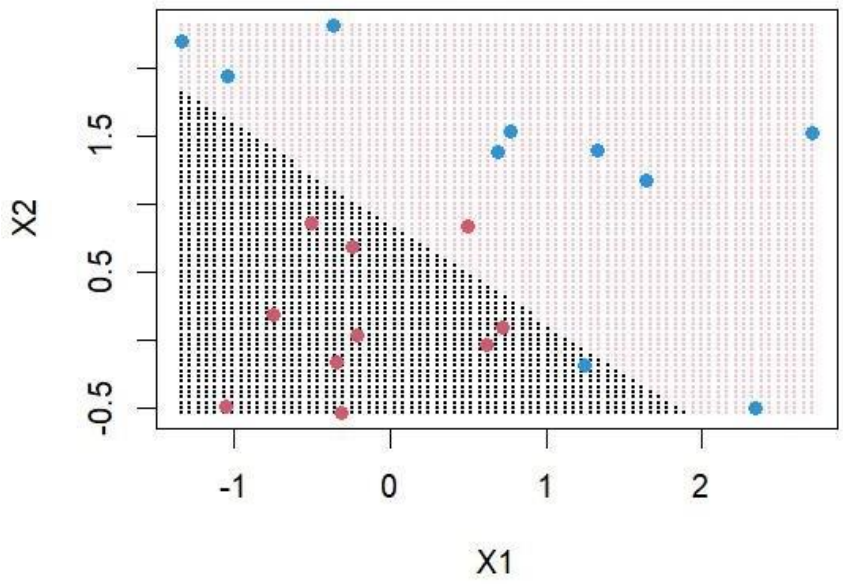



```
> make.grid = function(x, n = 75) {
+   grange = apply(x, 2, range)
```

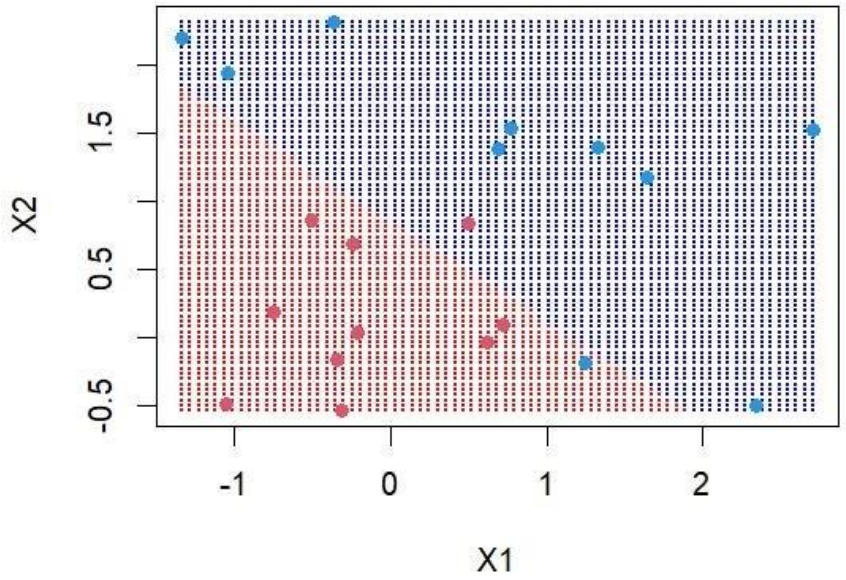
```
+   x1 = seq(from = grange[1,1], to = grange[2,1], length = n)
+   x2 = seq(from = grange[1,2], to = grange[2,2], length = n)
+   expand.grid(X1 = x1, X2 = x2)
+ }
> xgrid = make.grid(x)
> xgrid[1:10,]
```

	X1	X2
1	-1.3406379	-0.5400074
2	-1.2859572	-0.5400074
3	-1.2312766	-0.5400074
4	-1.1765959	-0.5400074
5	-1.1219153	-0.5400074
6	-1.0672346	-0.5400074
7	-1.0125540	-0.5400074
8	-0.9578733	-0.5400074
9	-0.9031927	-0.5400074
10	-0.8485120	-0.5400074

```
> ygrid = predict(svmfit, xgrid)
> plot(xgrid, col = c("black","pink")[as.numeric(ygrid)], pch = 20, cex = .2)
> points(x, col = y + 3, pch = 19)
> points(x[svmfit$index,], pch = 5, cex = 2)
> beta = drop(t(svmfit$coefs)%*%x[svmfit$index,])
> beta0 = svmfit$rho
```



```
> plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = .2)
> points(x, col = y + 3, pch = 19)
> points(x[svmfit$index,], pch = 5, cex = 2)
> abline(beta0 / beta[2], -beta[1] / beta[2])
> abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
> abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
>
```



(D) Implementing KNN (K- Nearest neighbour)

On google colab

```
suppressPackageStartupMessages(library(tidyverse))

data <- read_csv('loan_data.csv', show_col_types = FALSE)
data <- subset(data, select = -c(purpose))
head(data,3)
```

A tibble: 3 × 13

credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	0.1189	829.10	11.35041	19.48	737	5639.958	28854	52.1	0	0	0	0
1	0.1071	228.22	11.08214	14.29	707	2760.000	33623	76.7	0	0	0	0
1	0.1357	366.86	10.37349	11.63	682	4710.000	3511	25.6	1	0	0	0

```
install.packages("caTools")
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependency 'bitops'
```

```
library(caTools)
set.seed(255)

split = sample.split(data$not.fully.paid, SplitRatio = 0.75)
train = subset(data, split == TRUE)
test = subset(data, split == FALSE)
train_scaled = scale(train[-13])
test_scaled = scale(test[-13])
library(class)
test_pred <- knn(
        train = train_scaled,
        test = test_scaled,
        cl = train$not.fully.paid,
        k=10
        )
actual <- test$not.fully.paid
cm <- table(actual,test_pred)
cm
```

```
        test_pred
actual    0    1
     0 1988   23
     1  373   10
```

```
accuracy <- sum(diag(cm))/length(actual)
sprintf("Accuracy: %.2f%%", accuracy*100)
```

```
'Accuracy: 83.46%'
```