Implementation of Selected Machine Learning Algorithms on Two Separate UCI Repository Datasets

by

Akeem Ajede

A report submitted in partial fulfillment of the
requirements for a Masters in
Data Science

Auburn, Alabama
Spring, 2020

Supervisor

Bo Liu, PhD

Department of Computer Science & Engineering

ABSTRACT

This analytical study was done using the R-programming language and the two datasets used were sourced from the UC Irvine (UCI) Machine Learning Repository. The first dataset comprises of two separate files, representing the training and testing data, and both data have a binary data structure. The other dataset is a single multiclass data from which the training and testing data were sampled. Machine learning algorithms, comprising of Decision Tree, Support Vector Machine (SVM), and Perceptron, were implemented on the datasets, and the classification accuracy of each algorithms were reported. Last, the prediction accuracy of each algorithm considered in the analysis of both datasets were summarized in a tabular format.

Keyword: Machine Learning, Decision Tree, SVM, Perceptron, Algorithm

INTRODUCTION

Machine Learning (ML) is a branch of data science that requires minimal amount of human intervention to find subtle patterns in large databases and make analytical decisions. It can also be considered as a branch of artificial intelligence based on the idea that machines can learn from data beyond the capability of human.

There are several algorithms used in ML, depending on the specific task to be accomplished. In this project, three unique ML algorithms, including Decision Tree, SVM, and Perceptron, which are useful in data classification tasks were deployed in analyzing two different UCI ML Repository dataset.

Decision trees are built using a recursive partitioning, which is commonly referred to as split and conquer because the algorithm splits the data into subsets, which are further split until the algorithm deduce that the data within the subsets are sufficiently homogenous, or until the specified stopping criterion is satisfied [1]. SVM can be described as a surface that creates a decision boundary, also called hyperplane, between data points into homogenous partitions on either side, while maximizing the margin between the support vectors and the separating hyperplane.

Perceptron is one of the oldest ML algorithm, and it was originally designed by Frank Rosenblatt. A perceptron is a binary classification algorithm that makes its predictions based on a linear predictor function that combines a set of weights with their corresponding feature vector.

PURPOSE OF STUDY & SCOPE

The aim of this study is divided into two-fold. The first fold is to implement binary classification in the UCI's "a4a" dataset, and the second fold involves the classification of the multi-class UCI's "Iris" dataset. The programming language to be used is R and the ML algorithms to be implemented in both datasets are Decision Tree, SVM, and Perceptron.

DATA DESCRIPTION

The "a4a" [2] and "Iris" [3] raw datasets were stored in sparse format, which requires pre-processing before they could be deployed in ML algorithms. The preprocessing was done in R and the code used can be found in appendix A.

DATA ANALYSIS

Binary Classification ("a4a" dataset)

- **Decision Tree**

The decision tree algorithm implemented in the binary classification task is the C5.0 algorithm, which was developed by the famous computer scientist, J. Ross Quinlan as an improved version of C4.5 and the early Iterative Dichotomizer 3 (ID3).

Choosing the best splitting candidate is a major challenge that determines the efficiency of a decision tree. C5.0 uses the entropy concept to split the trees. High entropy indicates a diverse subset that provides little information on which class the subset belongs to. Entropy can be mathematically expressed as shown in equation (1).

$$Entropy\ (S) = \sum_{i=1}^{c} -p_i log_2(pi) \qquad (1)$$

Where "S" represents the segment of data, "c" refers to the number of class levels, and "p_i" is proportion of values falling into the ith class level. A quick illustration of the concept of entropy is shown below. To determine the optimal feature to split upon, the algorithm computes the difference in homogeneity from splitting on each feature, which is a measure of "information gain."

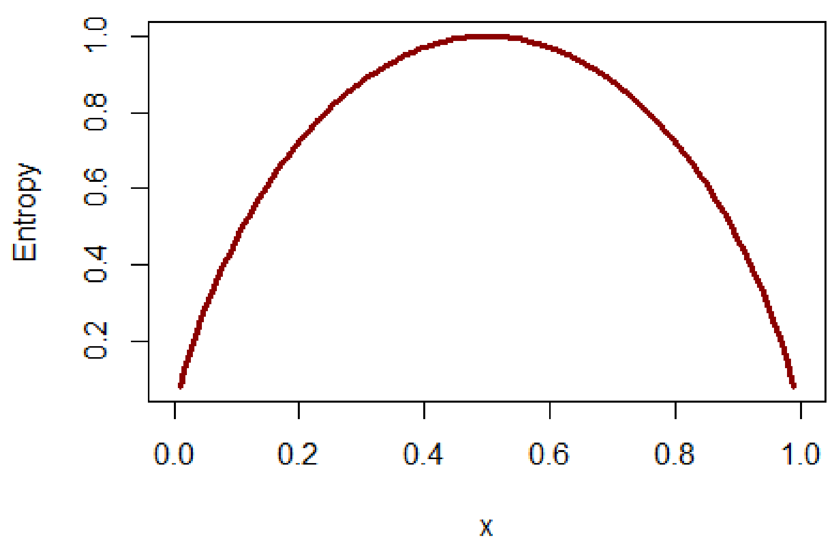

Figure 1. Illustration of Entropy

The higher the information gain, the better a feature is in producing homogenous subsets after splitting. Using this concept, the decision algorithm was written, and a summary of the attribute importance is shown in Figure 2.

```
##  100.00% w40
##   54.44% w23
##   52.92% w29
##   50.47% w74
##   48.90% w39
##   35.49% w76
##   32.63% w1
##   30.81% w35
##   25.87% w51
##   22.30% w9
##   22.15% w52
##   19.05% w47
##   16.25% w78
##    1.99% w3
##    1.05% w4
##    0.88% w82
##    0.61% w14
```

Figure 2. Attribute Importance

The confusion matrix of the decision tree implementation is shown in Figure 3. The resulting classification accuracy is approximately 83%.

```
##              | actual
##    predicted |        -1 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##           -1 |     19263 |      1864 |     21127 |
##              |     0.693 |     0.067 |           |
## -------------|-----------|-----------|-----------|
##            1 |      2857 |      3796 |      6653 |
##              |     0.103 |     0.137 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     22120 |      5660 |     27780 |
## -------------|-----------|-----------|-----------|
```

Figure 3. Confusion Matrix of the Binary Classification

Model performance improvement was executed by boosting the decision tree. C5.0 algorithm improved upon the C4.5 algorithm through the adoption of adaptive boosting. The resulting confusion matrix is shown in Figure 4. The classification accuracy slightly increased by approximately 1.3% to yield a new classification accuracy of 84.3%

```
##              | actual
##    predicted |        -1 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##           -1 |     19468 |      1659 |     21127 |
##              |     0.701 |     0.060 |           |
## -------------|-----------|-----------|-----------|
##            1 |      2714 |      3939 |      6653 |
##              |     0.098 |     0.142 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     22182 |      5598 |     27780 |
## -------------|-----------|-----------|-----------|
```

Figure 4. Confusion Matrix of the Boosted Binary Classification Decision Tree

- **SVM**

The critical feature of SVM is the ability to map a non-linear data classification problem into a higher dimensional space using kernels such that non-linear relationship then appears to be linear. Kernels include the linear, sigmoid, gaussian, and the Radial Base Function (RBF) kernels. Application of the right kernel is a form of trial and error because the suitability depends on the type and size of the data. The outcome (confusion matrix) of the SVM binary classification is shown in Figure 5. The linear kernel was used because it returned the highest prediction accuracy (i.e., 84.4%). The accuracy of the SVM binary classifier on the dataset is almost the same as the prediction accuracy of the Decision Tree.

```
##          Actual
## Predicted    -1     1
##        -1 19456  2649
##         1  1671  4004
```

Figure 5. Confusion Matrix of the SVM Binary Classifier

- **Perceptron**

The perceptron algorithm was implemented on the entire training dataset, but the resulting weight vector has 123 rows. For ease of implementation on the testing dataset, four (4) features were selected at random and used to train the perceptron. The weight vector obtained was used in the testing dataset, and the resulting accuracy was 50.7%. The prediction accuracy reinforces the superiority of the SVM and Decision Tree as a binary classifier.

From Figure 6, it can be deduced that the perceptron algorithm did not converge to zero.

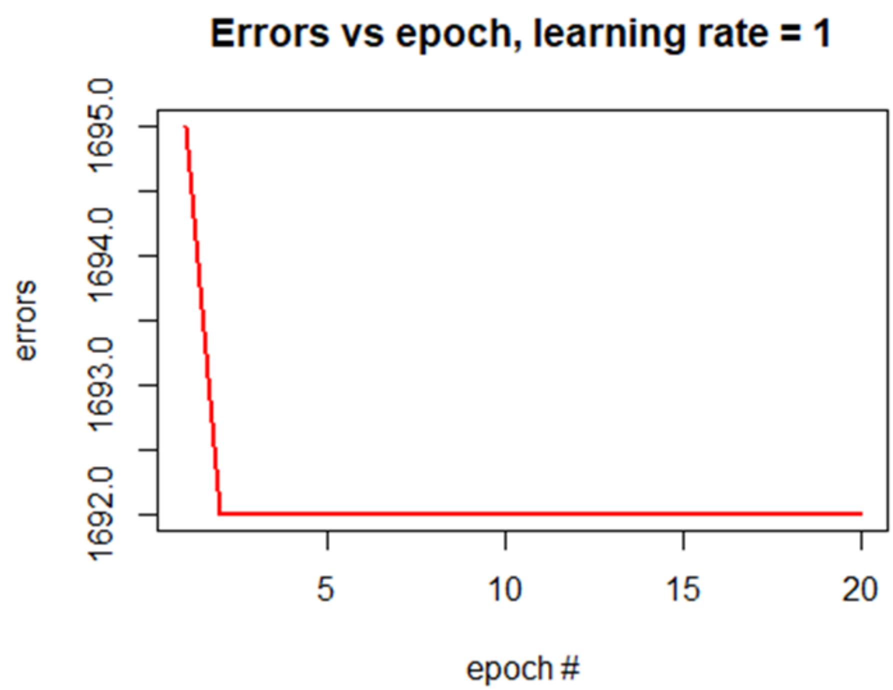

Figure 6. Graph of Error vs Epoch # - a4a

Multi-Class Classification ("Iris" dataset)

- **Decision Tree**

The summary of the attribute importance of the multi-class decision tree algorithm is shown in Figure 7.

```
##  Attribute usage:
##
##  100.00% Petal.Length
##   66.67% Petal.Width
```

Figure 7. Attribute Importance of the Iris Dataset Features

The confusion matrix of the multi-class decision tree is shown in Figure 8. The resulting classification accuracy is approximately 97%. Only one data point was misclassified.

```
##              | actual
##    predicted |     setosa | versicolor | virginica |  Row Total |
## -------------|------------|------------|------------|------------|
##       setosa |         10 |          0 |          0 |         10 |
##              |      0.333 |      0.000 |      0.000 |            |
## -------------|------------|------------|------------|------------|
##   versicolor |          0 |         11 |          0 |         11 |
##              |      0.000 |      0.367 |      0.000 |            |
## -------------|------------|------------|------------|------------|
##    virginica |          0 |          1 |          8 |          9 |
##              |      0.000 |      0.033 |      0.267 |            |
## -------------|------------|------------|------------|------------|
## Column Total |         10 |         12 |          8 |         30 |
## -------------|------------|------------|------------|------------|
```

Figure 8. Confusion Matrix of the Multi-Class Classification

- **SVM**

Figure 9 presents the outcome (confusion matrix) of the SVM binary classification. The classification accuracy is 100%. No data point was misclassified!

```
##                Actual
## Predicted     setosa versicolor virginica
##    setosa         10          0         0
##    versicolor      0         11         0
##    virginica       0          0         9
```

Figure 9. Confusion Matrix of the SVM Multi-Class Classifier

- **Perceptron**

The Iris dataset was plotted to evaluate the separability of the data points as shown in Fig. 10. Generally, a perceptron ML algorithm is a binary classifier, but it is implemented in the case of a multi-class data by deploying the 1 vs (n-1) technique. Since the Iris dataset has three classes, two different perceptron were strategically implemented.

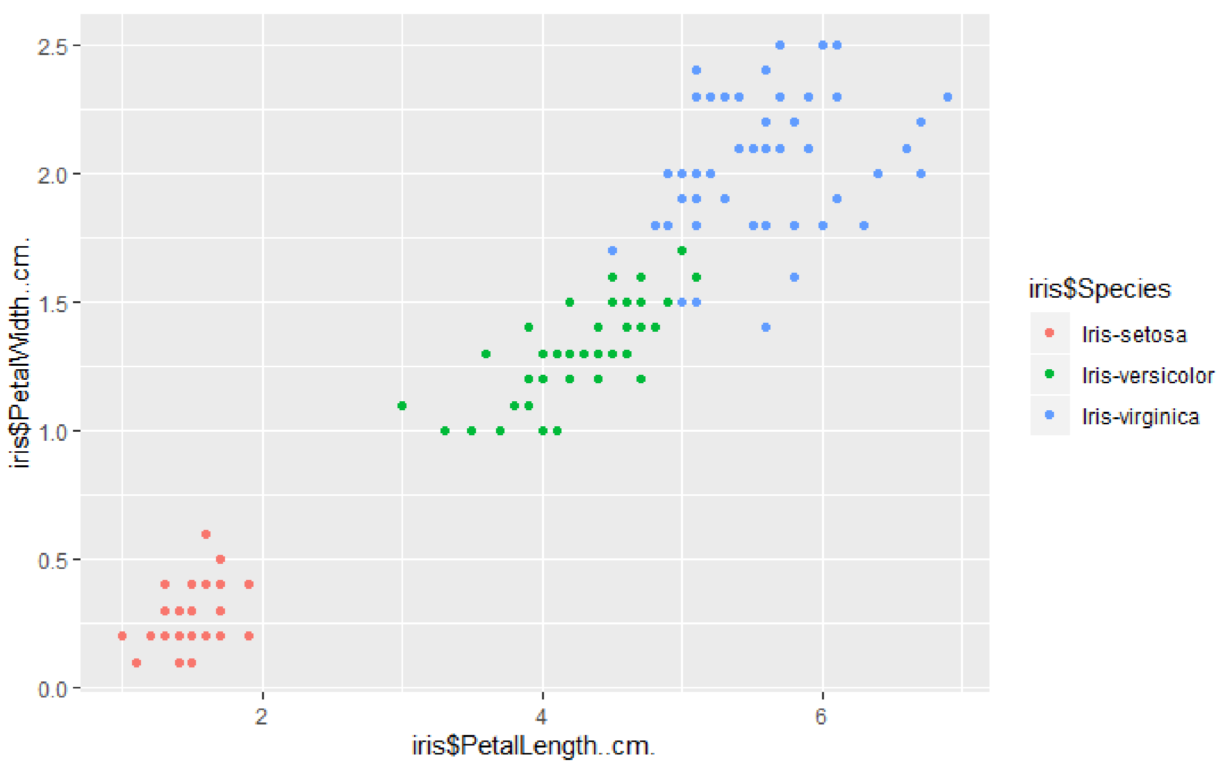

Figure 10. Graph of Petal Width Vs Petal Length

The first perceptron is a classifier that classifies Iris-Setosa specie from the other species, as illustrated in Figure 11. The perceptron was implemented, and it converged to zero, as shown in Figure 12. The prediction accuracy of the first perceptron is 53.3%.

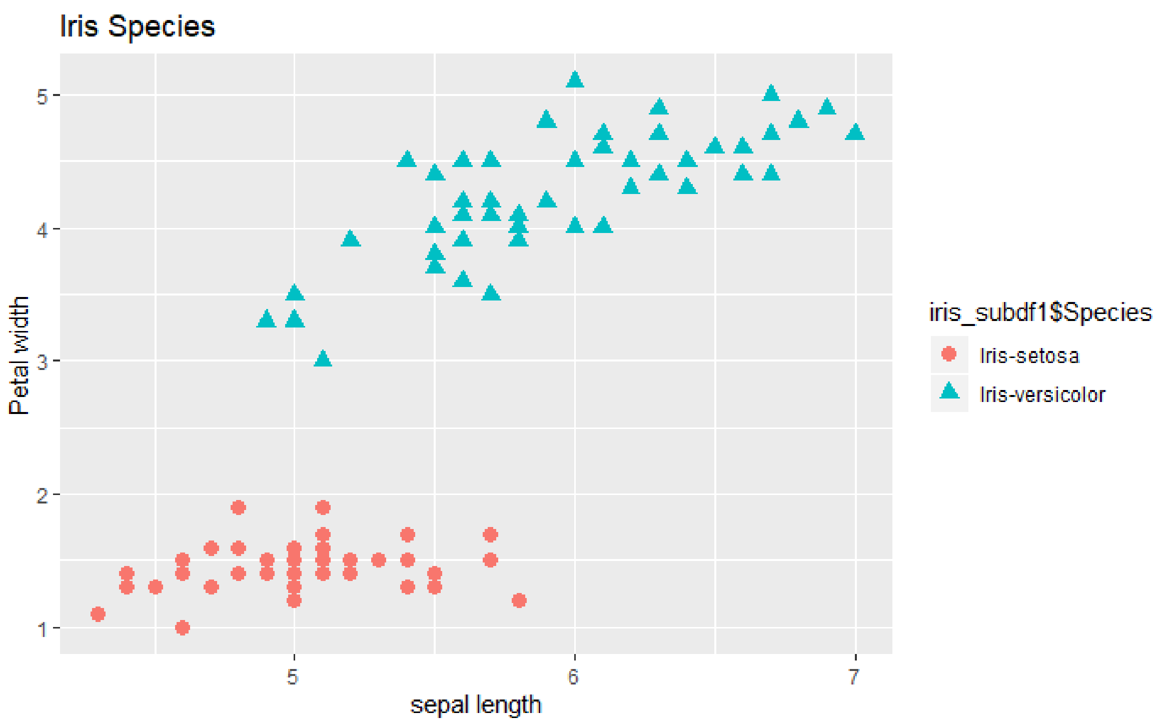

Figure 11. Graph of Petal Width Vs Sepal Length (Setosa Vs Others)

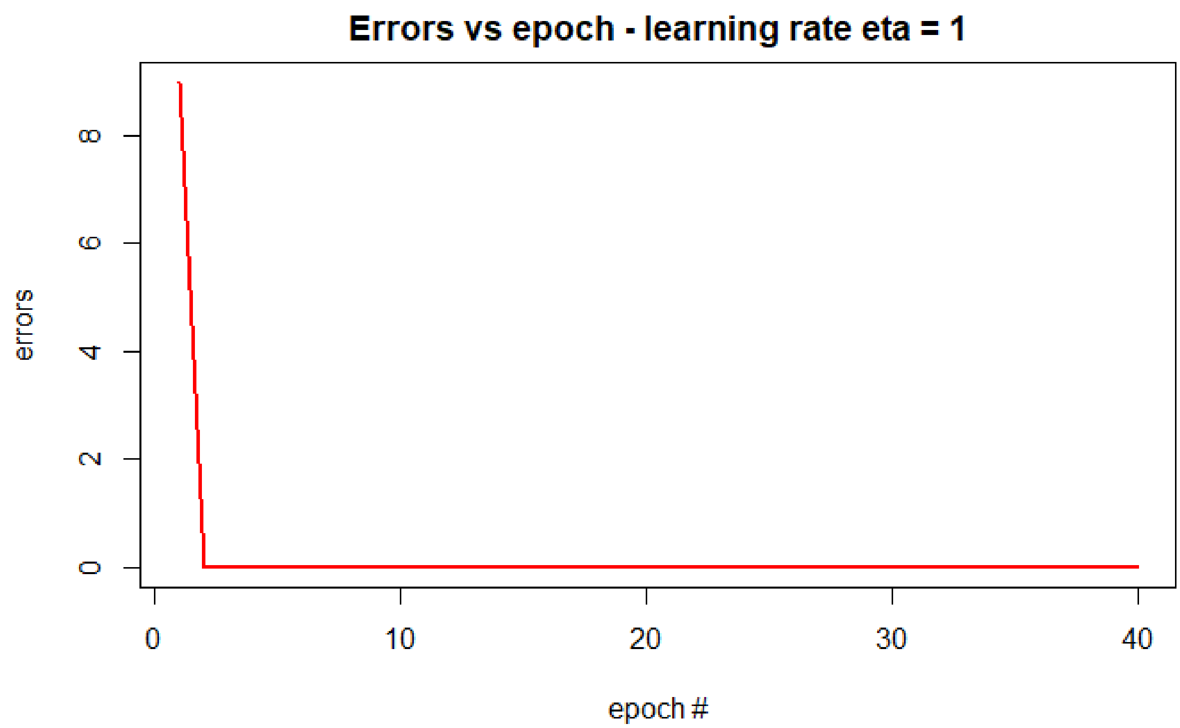

Figure 12. Graph of Error vs Epoch # - Setosa Vs Others

The second perceptron is a classifier that classifies Iris-Virginica specie from the other species, as illustrated in Figure 13. Figure 13 suggested that Iris-virginica is not linearly separable from the other species. The perceptron was implemented, and as expected, it did not converge to zero, as shown in Figure 14. The prediction accuracy of the second perceptron is 28.2%.

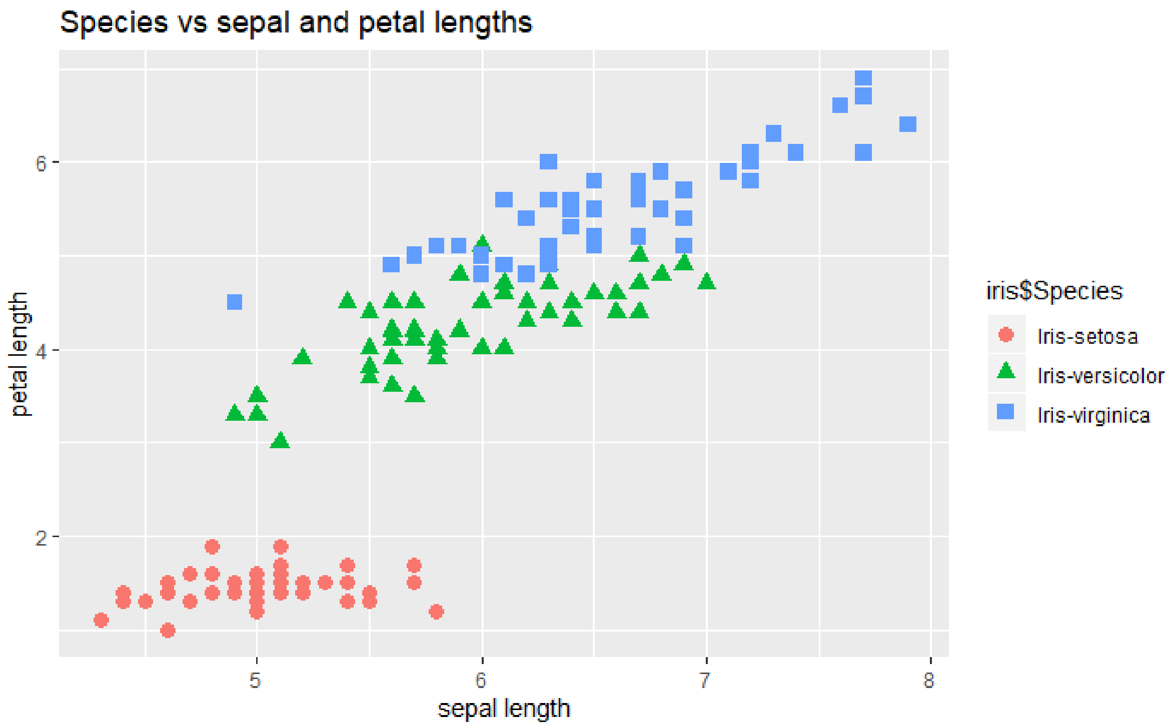

Figure 13. Graph of Petal Length Vs Sepal Length (Virginica Vs Others)

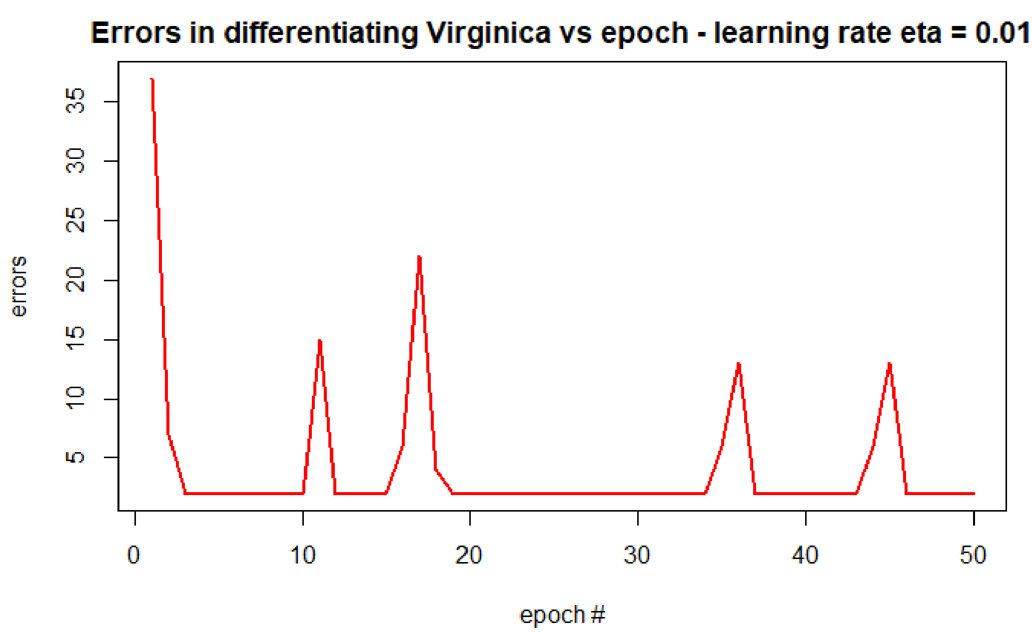

Figure 14. Graph of Error vs Epoch # - Virginica Vs Others

The prediction accuracies of the three ML algorithms used in both datasets are summarized in Table 1.

Table 1. Summary of Prediction Accuracy

	Prediction Accuracy (%)		
Dataset	**Decision Tree**	**SVM**	**Perceptron**
a4a	84.3	84.4	50.7
Iris	97	100	(53.3; 28.2)

CONCLUSION

The analysis results showed that SVM and Decision Tree ML algorithms are superior in performance to Perceptron in both binary and multi-class classification tasks. In instances where interpretability is highly desirable, Decision Tree becomes the preferred choice since the classification accuracy is high and the interpretability is still within human comprehension.

REFERENCES

[1] Brett Lantz. (2015). Machine Learning with R. ISBN 978-1-78439-390-8

[2] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#a4a

[3] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#iris

APPENDIX: A

Pre-Processing

```
read.libsvm = function( filename, dimensionality ) {

  content = readLines(filename )
  num_lines = length( content )
  yx = matrix( 0, num_lines, dimensionality + 1 )

  # loop over lines
  for ( i in 1:num_lines ) {

    # split by spaces
    line = as.vector( strsplit( content[i], ' ' )[[1]])

    # save label
    yx[i,1] = as.numeric( line[[1]] )

    # loop over values
    for ( j in 2:length( line )) {

      # split by colon
      index_value = strsplit( line[j], ':' )[[1]]

      index = as.numeric( index_value[1] ) + 1 # +1 because label goes first


      value = as.numeric( index_value[2] )

      yx[i, index] = value
    }
  }

  return( yx )
}
```

APPENDIX: B

Other Source Codes

```
train1 <- read.csv("a4aTraining.csv", header = TRUE)
names(train1)
```

```
#Class variables function
Classes <- function(data){
  Class_variables <- sapply(data, function(x) class(x))
  return(Class_variables)
}
```

```
test1 <- read.csv("a4aTesting.csv", header = TRUE)
```

```
# Decision Tree (Binary Classification)
#Decision Tree Algo using the C5.0 algorithm by J. Ross Quinlanb (Industry St
andard) - Divide and Conquer

curve(-x * log2(x) - (1 - x) * log2(1 - x),
      col = "darkred", xlab = "x", ylab = "Entropy", lwd = 3) #Illustration o
f entropy; 50-50 split results in maximum entropy


library(dplyr)

a4a_train <- train1 %>%
  mutate_at(vars(Label),
            funs(factor))   #Transforms the label integer variable to a facto
r variable

a4a_test <- test1 %>%
  mutate_at(vars(Label),
            funs(factor))
library(C50)

model <- C5.0(a4a_train[-1], a4a_train$Label) #Decision tree model
model
```

```
#summary(model)


# Model performance evaluation


model_pred <- predict(model, a4a_test)

library(gmodels)
```

```
#Confusion Matrix

CrossTable(a4a_test$Label, model_pred, prop.chisq = FALSE, prop.c = FALSE, pr
op.r = FALSE,
           dnn = c('predicted', 'actual'))

(mean(model_pred == a4a_test$Label))*100 #Classification Accuracy is approx.
83%

model_boost <- C5.0(a4a_train[-1], a4a_train$Label, trials = 10)
model_boost

model_boost_pred <- predict(model_boost, a4a_test)
CrossTable(a4a_test$Label, model_boost_pred, prop.r = F, prop.c = F, prop.chi
sq = F,
           dnn = c("predicted","actual"))

(mean(model_boost_pred == a4a_test$Label))*100 #Classification Accuracy is ap
prox. 84.3%

#Boosting the tree barely produced a significant improvement in the tree

library(e1071)
```

```
set.seed(7)
```

```
svm_model <- svm(a4a_train$Label~., data = a4a_train, kernel = "linear",

scale = TRUE) #linear Kernel
```

```
summary(svm_model)

#Confusion Matrix
pred1 <- predict(svm_model, a4a_test)
pred_table <- table(Predicted = pred1, Actual = a4a_test$Label)
pred_table

library(dplyr)
a4a_train <- train1 %>%
  mutate_at(vars(Label),
            funs(factor))   #Transforms the label integer variable to a facto
r variable

a4a_test <- test1 %>%
  mutate_at(vars(Label),
            funs(factor))


#Perceptron Algorithm

# write function that takes in the data frame, learning rate - eta, and numbe
r of epochs - n.iter and updates the weight factor.
# To obtain the final weight and the number of epochs required for the weight
```

```
to converge

#Here we separate the attributes from the class

library(optimbase)

x <- a4a_train[-1]
names(x) <- tolower(names(x))

# create species labels
y <- train1$Label

perceptron <- function(x, y, eta, n_iter) {

  # initialize weight vector
  weight <- rep(0, dim(x)[2] + 1)
  errors <- rep(0, n_iter)


  # loop over number of epochs niter
  for (jj in 1:n_iter) {

    # loop through training data set
    for (ii in 1:length(y)) {

      # Predict binary label using Heaviside activation
      # function
      z <- sum(weight[2:length(weight)] *
                 as.numeric(x[ii,])) + weight[1]
      if(z < 0) {
        y_pred <- -1
      } else {
        y_pred <- 1
      }

      # Change weight - the formula doesn't do anything
      # if the predicted value is correct
      weight_diff <- eta * (y[ii] - y_pred) *
        c(1, as.numeric(x[ii, ]))
      weight <- weight + weight_diff

      # Update error function
      if ((y[ii] - y_pred) != 0.0) {
        errors[jj] <- errors[jj] + 1
      }

    }
  }

  # weight to decide between the two species
```

```
  print(weight)
  return(errors)
}

err_train_a4a <- perceptron(x,y,1,8)

# Model evaluation

#Due to the constraint of the huge data set, randomly selected subset of the
data will be used in developing the perceptron


#Keeping all the attributes

a4a_train_s <- a4a_train[, c(37,45,83,122)] #attributes


x <- a4a_train_s    #attributes
y <- train1$Label        #class values

#str(train1)

# compute and plot error

a4a_train_err <- perceptron(x, y, 1, 20)

## [1] -2 -2 -2  2 -4

#Visualization

plot(1:20, a4a_train_err, type="l", lwd=2, col="red", xlab="epoch #", ylab="e
rrors")
title("Errors vs epoch, learning rate = 1")

w_a4a <- c(-2,-2,-2,2,-4)  #Weight of the perceptron


# Model Evaluation

#Let us test the accuracy of the perceptron



a4a_test[,125]<- 1 #Initialize
a4a_test[a4a_test[,1] == "Label",125]<- -1

x <- a4a_test[, c(37,45,83,122)]  #attributes
y <- a4a_test[,125]        #class values

a4a_test[,1] <- 1
```

```
w_a4t <- c(-2,-2,-2,2)
colnames(x) <- NULL
p1<-zeros(27780, 1)
for (ii in 1:27780) {
  p1[ii,1]<- w_a4t%*%as.double(x[ii,])
}
p1[p1 >= 0] = 1
p1[p1< 0] = -1

pred_accuracy = (sum(p1==y)/27780)*100
pred_accuracy  #Class accuracy is 50.7%

iris = read.csv("Iris - data.csv", header = TRUE)
names(iris)
```

```
set.seed(7)
iris_sampling <- sample(150,120)
str(iris_sampling) #Looks randomized

iris_train <- iris[iris_sampling,]
iris_test <- iris[-iris_sampling,]

iris_model <- C5.0(iris_train[-5], iris_train$Species)
iris_model
summary(iris_model) #Training error is 2.5%

#Evaluate Model Performance

iris_pred <- predict(iris_model, iris_test)
library(gmodels)
CrossTable(iris_test$Species, iris_pred, prop.r = FALSE,
           prop.c = FALSE, prop.chisq = FALSE,
           dnn = c("predicted", "actual"))

(mean(iris_pred == iris_test$Species))*100 #Classification Accuracy is approx
. 97%

# Support Vector Machine (Multi-Class Classification) - Finding optimal separ
ating hyperplane while maximizing margin

#Visualization of the Iris data

library(e1071)

set.seed(7)
iris_model1 <- svm(iris_train$Species~., data = iris_train, kernel = "linear"
) #linear Kernel
#summary(iris_model1)
```

```r
#Confusion Matrix
pred2 <- predict(iris_model1, iris_test)
pred_table1 <- table(Predicted = pred2, Actual = iris_test$Species)
pred_table1

(mean(pred2 == iris_test$Species))*100 #Classification Accuracy is 100%(RBF),
100%(Linear), 96.7%(Sigmoid)
```

```r
#Perceptron Algorithm


#summary(iris)

#create sub-dataframe


iris_subdf1 <- iris[1:100, c(1,2,3,4,5)]

names(iris_subdf1)
```

```r
#generate a training a training and testing data set from the iris sub-frame

set.seed(7)
sbf_sample <- sample(100,70)

str(sbf_sample) #looks randomized

iris_subdf1_train <- iris_subdf1[sbf_sample,] #70 observations
iris_subdf1_test <- iris_subdf1[-sbf_sample,] #30 observations

#str(iris_subdf1_test)


iris_subdf1_train[, 6] <- 1      #initialize
iris_subdf1_train[iris_subdf1_train[, 5] == "Iris-setosa", 6] <- -1  #setosa
is now -1

x <- iris_subdf1_train[, c(1,2,3,4)]    #attributes
y <- iris_subdf1_train[, 6]          #class values
tail(y)

# write function that takes in the data frame, learning rate - eta, and numbe
r of epochs - n.iter and updates the weight factor.
# To obtain the final weight and the number of epochs required for the weight
to converge

#Here we separate the attributes from the class

perceptron_iris <- function(x, y, eta, n_iter) {

  # initialize weight vector
  weight <- rep(0, dim(x)[2] + 1)
```

```
  errors <- rep(0, n_iter)


  # loop over number of epochs niter
  for (jj in 1:n_iter) {

    # loop through training data set
    for (ii in 1:length(y)) {

      # Predict binary label using Heaviside activation
      # function
      z <- sum(weight[2:length(weight)] *
                 as.numeric(x[ii,])) + weight[1]
      if(z < 0) {
        y_pred <- -1
      } else {
        y_pred <- 1
      }

  # Change weight - the formula doesn't do anything
  # if the predicted value is correct
      weight_diff <- eta * (y[ii] - y_pred) *
        c(1, as.numeric(x[ii, ]))
      weight <- weight + weight_diff

      # Update error function
      if ((y[ii] - y_pred) != 0.0) {
        errors[jj] <- errors[jj] + 1
      }

    }
  }

  # weight to decide between the two species
  print(weight)
  return(errors)
}

iris_subdf1_train_err <- perceptron_iris(x,y,1,40)
plot(1:40,iris_subdf1_train_err, type="l", lwd=2, col="red", xlab="epoch #",
ylab="errors")
title("Errors vs epoch - learning rate eta = 1")

library(optimbase)

# Perceptron evaluation

w1 <- c(-2.0,-5.2,-11.8,18.2,8.2)     #weight for classifying setosa vs other
s
```

```r
#Let us test the accuracy of the first perceptron


iris_subdf1_test[, 6] <- 1     #initialize
iris_subdf1_test[iris_subdf1_test[, 5] == "Iris-setosa", 6] <- -1  #setosa is
now -1

x <- iris_subdf1_test[, c(1,2,3,4)]    #attributes
y <- iris_subdf1_test[, 6]          #class values

x[,5] <- 1

colnames(x) <- NULL
p1<-zeros(30, 1)
for (ii in 1:30) {
  p1[ii,1]<-w1%*%as.double(x[ii,])
}
p1[p1 >= 0] = 1
p1[p1< 0] = -1

pred_accuracy = sum(p1==y)/30
pred_accuracy  #Class accuracy of 53.3% on classifying setosa vs others

#Hyperplane for iris-viginica versus iris-setosa OR iris-vesicolor


#Keeping all the attributes

iris_subdf2 <- iris[, c(1,2,3,4,5)]
names(iris_subdf2) <- c("sepal", "petal", "species")


set.seed(7)
sbf_sample2 <- sample(150,111)

str(sbf_sample2) #looks randomized

iris_subdf2_train <- iris_subdf2[sbf_sample2,] #111 observations
iris_subdf2_test <- iris_subdf2[-sbf_sample2,] #39 observations

#str(iris_subdf2_test)

# Training the second perceptron


iris_subdf2_train[, 6] <- 1     #initialize
iris_subdf2_train[iris_subdf2_train[, 5] == "Iris-virginica", 6] <- -1  #Virg
```

```
inica is now 1

x <- iris_subdf2_train[, c(1,2,3,4)]    #attributes
y <- iris_subdf2_train[, 6]          #class values


# compute and plot error

irissubdf2_train_err <- perceptron_iris(x, y, 0.01, 50)

#Visualization

plot(1:50, irissubdf2_train_err, type="l", lwd=2, col="red", xlab="epoch #",
ylab="errors")
title("Errors in differentiating Virginica vs epoch - learning rate eta = 0.0
1") #Minimum error is 2, but the weight converged

w2 <- c(0.180,0.490,0.768,-0.970,-0.468)  #Weight of the second perceptron

# Model Evaluation

#Let us test the accuracy of the second perceptron


iris_subdf2_test[, 6] <- 1     #initialize
iris_subdf2_test[iris_subdf2_test[, 5] == "Iris-virginica", 6] <- -1  #Virgin
ica is now -1

x <- iris_subdf2_test[, c(1,2,3,4)]    #attributes
y <- iris_subdf2_test[, 6]          #class values

x[,5] <- 1

colnames(x) <- NULL
p1<-zeros(39, 1)
for (ii in 1:39) {
  p1[ii,1]<-w1%*%as.double(x[ii,])
}
p1[p1 >= 0] = 1
p1[p1< 0] = -1

pred_accuracy = sum(p1==y)/39
pred_accuracy  #Class accuracy of 28.2% on classifying Virginica vs others
```