Initital Setup

The Assignment is implemented using the R Language to build a Linear Regression model.

```
setwd('/Users/Rushi/Documents/Masters/ML/Project 1') #Set the working directory
library('MASS')
data<-read.csv('iris.txt',header=FALSE) #read data from the dataset
newdata<-data
newdata$V5<-NULL #dropping the 5th column from the dataset
A<-data.matrix(newdata, rownames.force = NA) #create matrix for remaining data
transpose<-t(A) #transpose of A
Y<-matrix(data$V5)
Y<-sapply(data$V5,switch,'Iris-setosa'=1,'Iris-versicolor'=2,'Iris-virginica'=3,'undefined'="")
#set integer values to each of the 3 classifications as follows:
#1.Iris-setosa
#2.Iris-virginica
#3.Iris-versicolor
```

1. **Training a model via Linear Regression**

```
#Linear Regression Function
linear_regression <- function(A, Y){
  transpose<-t(A)
  beta<-ginv(transpose%*%A)%*%transpose%*%Y
  return(beta)
  }
```

Here a function is made to implement Linear Regression using the formula Beta = $(AA^T)^{-1}A^TY$.

2. **Using the trained model to do the classification**

```
#Classification function
classify <- function(test, beta, length_class){
  #No round -
    #Yop = test*beta
    Yop <- round(test%*%beta)
    Yop[ Yop > length_class ] <- length_class
    Yop[Yop < 1] <- 1
    return(Yop)
}
```

The classification function is used to classify and predict the data using test and train data. The predicted values are rounded to their nearest classification value.

3. K-Fold Cross Validation

```
#K-fold Cross Validation Function to find error rate
KFCV <- function(A,Y,k){
 randindex<-sample(length(Y))
 size<-length(Y)/k
 error <- 0
 class_name <- unique(Y)
 for(i in 1:k){
  test <- randindex[ (size*(i-1)+1): (size*i) ] #Selects test data with given K one at a time
  train <- setdiff( 1:length(Y),  test) #Rest of the folds are used as a training data
  beta <- linear_regression(A[train, ], Y[train])
  Ycalc <- classify(A[test, ], beta, length(class_name))
  error <- error + sum_of_squared_error(Ycalc, Y[test])
 }
  error <- error/k
  return(error)
}
```

Here a K Fold Cross Validation function is built to Divide the Dataset in K Folds where K is the size provided by the user. The function takes the input matrix, output matrix and k as arguments. Then we divide the dataset into test and train and take test data and rest as the train data to train the model.

```
#sum of squared error function
sum_of_squared_error = function(Ycalc, Yact){
 error = Ycalc - Yact;
 error = sum(error * error)/length(Ycalc);
 return(error)
}
```

This function is used to find the error rate by taking the difference of the actual Y value from the predicted Y values.

```
beta<-linear_regression(A,Y)
print(paste("Beta :", beta))
k<-10;
KVError<-KFCV(A, Y, k)
print(paste("Average Kfold Error : ", KVError))
```

In the code above we perform linear regression on A and Y matrix and then find the average K-Fold-Error rate K here is 10 which is ideal for this case as it takes less time to perform lower number of folds and the error rate is also less which is as shown below:

```
> print(paste("Average Kfold Error : ", KVError))
[1] "Average Kfold Error :  0.04"
> print(paste(avg))
[1] "0.0405999999999994"
>
```

Hence if the error rate is 4% the accuracy can be said to be 96% for 10-Fold cross validation.

```
#Finding the ideal value
error <- 0;
N <- 1000;
for (i in 1:N){
  error = error + KFCV(A, Y, k);
}
avg = error / N;
print(paste(avg))
```

The code above is used to find the ideal value by running K fold for N times which is 1000 in this case. By running this KFold for 1000 times on with k=10 we find the error rate which is efficient as well as ideal to calculate the error rate.
If we take k=30 the error is reduced little bit but takes lot of time to perform 30 fold cross validation and hence is no very efficient.