'PREDICTION OF HILL-VALLEY USING CLASSIFICATION METHODS'

Instructor

Dr. Na Zou

(Professor, ISEN 613 - Engineering Data Analysis)

Project Team

SR. NO.	STUDENT NAME	UIN	CONTRIBUTION
1.	Mr. Rajesh Subramanian	128003720	20%
2.	Mr. Rupak Ghawghawe	928000506	20%
3.	Mr. Akshay Kadu	827007175	20%
4.	Mr. Rutwhij Shukla	228001239	20%
5.	Mr. Swapnil Kale	727004928	20%

INDUSTRIAL & SYSTEMS ENGINEERING

TEXAS A&M UNIVERSITY

INDEX

List of Figures & Tables

1. IMPORTANCE

Geography provokes and answers questions about the natural and human worlds, using different scales of enquiry to view them from different perspectives. It develops knowledge of places and environments throughout the world, an understanding of maps, and a range of investigative and problem solving skills both inside and outside the classroom. To understand the geography of past times and how geography has played important roles in the evolution of people, their ideas, places and environments. To develop a mental map of the community, country and the world so that you can understand the "where" of places and events.

Studying geography invites us to participate fully in the excitement, enjoyment and challenge of this dynamic world. Also it draws on personal experience, to help us better understand the places we live in, why they matter and how they are connected to a globalized world. Geography draws from across the physical, cultural, economic and political spheres to illuminate key issues for the present and the future, explored at all scales from the personal to the local and the global. Geography is a focus within the curriculum for understanding and resolving issues about the environment and sustainable development. It is also an important link between the natural and social sciences.

As we study geography, we encounter different societies and cultures. The subject helps develop significant elements of the skills framework, with a strong emphasis on utilising maps and visual images as well as new technologies including Geographical Information. These transferable geographical skills help to equip us for lifelong learning as responsible global citizens.

2. OBJECTIVE

The main objective of this project is to leverage the supervised learning methods to build relation between the given predictors values to set-up a model which can be further used to classify any observation based on its attribute values into predefined classes. We are provide with the geographical data with 100 attribute, which plotted as ordinate will create a bump or dip profile.

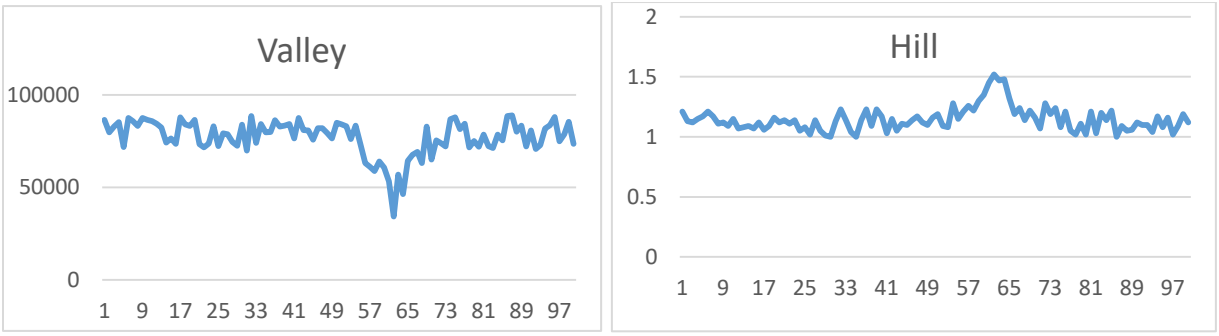


Fig 1: 2-D representation of observation data as Hill & Valley

Valleys in low lying areas have an average slope whereas in the mountains, they are deep and narrow. Erosion by rivers is a main valley-forming process, such as movement of the earth's crust and glaciers, also have an essential importance of agribusiness. Valley are useful for exploration for petroleum and groundwater, high quality sand for hardware and chips for computation purposes. With this dataset we can effectively build a model which will co-relate these parameters and help to identify the locations enriched with minerals and commercial materials.

On the other hand hill or high lying areas are elevations osf the earth's surface that have distinct summits, but are lower in elevation than mountains. Hills may be formed by a buildup of rock debris or sand deposited by glaciers and wind. Hills may be created by faults which are slight cracks in the earth which can cause earthquakes. Hills provide a major advantage to an army, giving them an elevated firing position and forcing an opposing army to charge uphill to attack them. We can use the model to build predictions based on the given data set and use this for tree plantation and local weather predictions.

3. SCOPE OF WORK

The techniques suggested in this project support the implementation of data analytical (classification) techniques obtained over the span of this course to take care of imposed problems in an effort to aid the detection of hills or valley.

We have a set of training observations (X_1, Y_1)..... (X_n, Y_n) that we can use to build a classifier. As the classifier should work well on both training as well as test data, we divided our data into test and train datasets.

In supervised learning, a single method does not dominates the entire classification process, as each method has its own advantage over another and can predict the results better. Linear Discriminant Analysis (LDA) dominates Classification Tree for a dataset for which the distribution is linear, whereas K-Nearest Neighbor performs well

on a non-linear data. Treating the data as black box to get the interaction between the attribute and classes without understanding the underlying relation distribution of the data makes selection of the best method for analysis difficult. Hence, we have attempted to carefully describe the model, intuition, assumptions, and trade-offs behind each of the methods that we consider.

We implemented different classification models such as Classification Tree, Random Forest, Boosting, Linear Determinant Analysis (LDA), K-Nearest Neighbors (KNN) and Support Vector Machines (Radial and Polynomial), Quadratic Discriminant Analysis (QDA), Cross Validation, Subset Selection, Support Vector Classification (SVC) on each training as well as test data and the methods were compared against each other on parameters like Test Error Rate, Sensitivity, Specificity, Accuracy and False Positive rate. The method with highest accuracy will be used for further analysis of the data in future.

4. Project Approach

- **Data Description :**

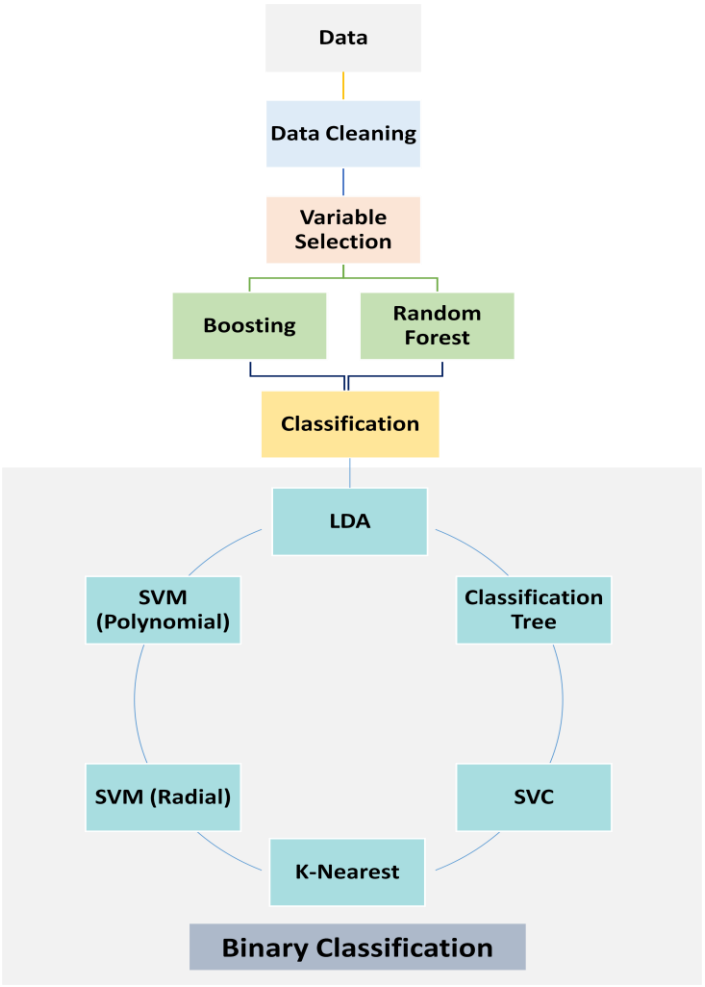
The dataset 'Hill-Valley' is taken from the UCI Repository. There are 606 rows of observations, each row contains measurement of 100 attributes which represents the parameters to predict the nature of a geographical area. Depending upon the geographical feature observed, the data set is classified in 2 different classes: Class 0 which represents 'valleys' and Class 1 represents 'hills'.

- **Reason for approach**

In ISEN 613 course, we came across various methods which utilizes different statistical models to represent real life problem. But we also realized that there is no single method which dominates other. Every statistical method provide results based on certain assumptions which are necessary to reduce the noise in the data which may have adverse effect on the prediction accuracy of the method. A particular model which provides significant predication accuracy may not provide same accuracy on a different type of data. This forms the basis of our approach throughout this project and hence we have applied all our class learning's to find out best fit model which will solve the problem we have chosen.

5. Methodology :

Initially, the data was cleaned to remove all non-significant observations. After that, we split the data into training data set and test data set to proceed further. The training data has 80% of the total observations whereas the test data has 20% of the total observations. We used the training data to train the models and test data was used to test the prediction accuracy of the model



. Fig 2: Methodology flowchart

The training dataset was used for training different classification models like Linear Determinant Analysis, Quadratic Determinant Analysis, Logistic Regression, K - nearest neighbors, Random Forest, Boosting for Variable Selection, Support Vector Classification, Support Vector Machine with Radial kernel, Support Vector Machine with Polynomial Kernel, Classification tree, Pruning of Classification tree. After training, the test error rate was determined using cross-validation.

6. IMPLEMENTATION

I. Variable Selection

- **Random Forest:**

The classification using random forest is done using the following code. The best mtry is calculated and then rest of the parameters are evaluated based on that result.

```
rf.hill_valley = randomForest(class ~ ., data = hill_valley.train,
mtry = 10,
n.trees = 50, importance = TRUE)
best.tree    =    tuneRF(hill_valley.train,    hill_valley.train$class,
stepFactor = 1.1, improve = 0.05)
```

Result:

For mtry= 10, OOB error= 0.83%; for mtry= 11, OOB= 0.41%; for mtry= 12, OOB= 1.65%. Hence, on the basis of these results, the best mtry value is taken to be 11 as it has the least OOB error.

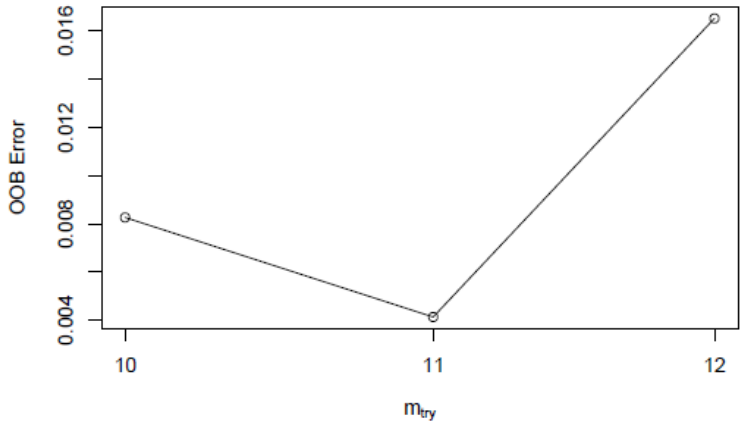


Fig 3: mtry vs OOB error for random forest

```
prediction = predict(rf.hill_valley, hill_valley.test, type = "class")
table(predicted = prediction, actual = hill_valley.test$class)
```

Result of predicted class:

predicted	0	1

0	30	25
1	30	37

a) To calculate Error Rate for Random Forest

```
error.rf = mean(prediction!=hill_valley.test$class)
error.rf
```

Error rate= 45.08%

After calculating the performance parameters on the whole tree, we then decided to do variable selection based on Gini index as follows.

```
importance(rf.hill_valley)

varImpPlot(rf.hill_valley)
```

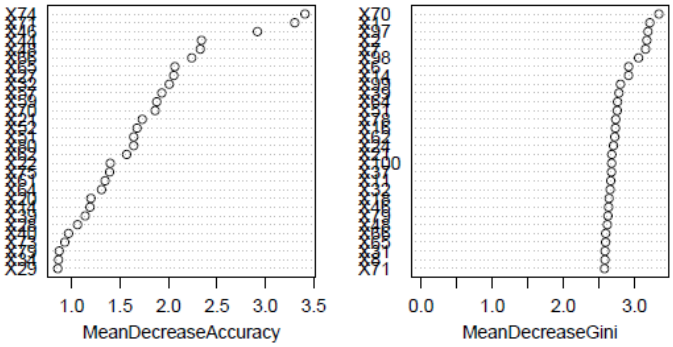


Fig 4: Variable selection using random forest

Based on above graph, the variables with least decrease in Gini index are X7, X2, X97, X1, X70.

Cross Validation:

Before applying any classification method to the dataset, we need to define a subset of test and train observation for fitting a test model and to evaluate the efficiency and error rate of the statistical model. We decided to use the cross-validation method throughout all models so that the results could be compared for all based on test and train data.

```
rf.hill_valley_cv = randomForest(class ~ X7+X2+X97+X1+X70, data =
hill_valley.train, mtry = 10, n.trees = 50, importance = TRUE)
best.tree = tuneRF(hill_valley.train, hill_valley.train$class, stepFactor =
1.1, improve = 0.05)
```

```
result: for mtry=, OOB= 2.69%, for mtry= 11, OOB = 0.62%, for mtry= 12, OOB
error= 0.62%.
Hence, choosing mtry =12 as it gives minimum mtry error, following parameters
are evaluated-

pred.rf = predict(rf.hill_valley_cv, hill_valley.test, type = "class")
table(pred.rf, hill_valley.test$class)
```

pred.rf	0	1
0	29	21
1	31	41

a) Test error:

```
error.rf = mean(prediction!=hill_valley.test$class)
error.rf
```

##***Error rate= 45.08%***

b) Accuracy:

```
accuracy.rf = mean(prediction==hill_valley.test$class)
accuracy.rf
```

##Accuracy = 54.91%

- **Boosting:**

Bagging and Random forests are implemented on each bootstrapped sample fitting an individual tree on each sample and them combining all to get one single predictive model. Boosting works on the same principle. But unlike random forests, it grows trees sequentially. Each tree is grown from information obtained from previous model. Since in statistical learning, methods that learn slowly tends to perform well hence giving us a better performance from boosting compared to random forests.

We applied boosting using the following tuning parameters-
Number of trees used= n.trees= 5000
Shrinkage parameters= λ= 0.019
Number of splits in each tree= 10

```
boost_hill_valley = gbm(class~.,data = hill_valley.train, distribution=
"gaussian",n.trees = 5000 , interaction.depth = 4,
shrinkage = 0.019, verbose = F)
summary(boost_hill_valley)
```

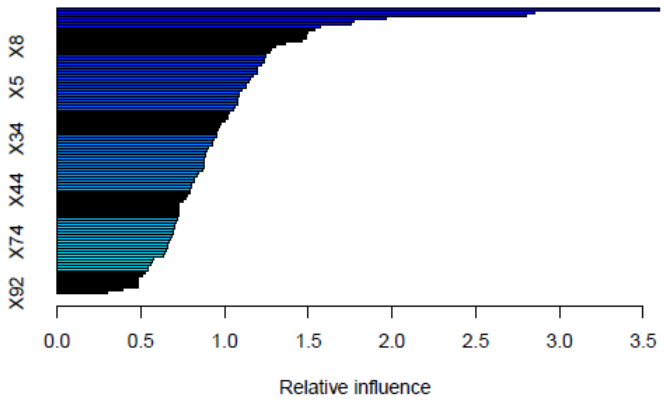


Fig 5: Variable selection using Boosting

Result: ***The variables which have maximum effect on response are- X1, X3, X2, X6, X11, X7, X70 considering cut-off 1.6.***

```
yhat.boost=predict (boost_hill_valley,newdata =hill_valley.test,
n.trees = 5000)
yhat.boost = ifelse(yhat.boost< 1.02, yhat.boost==0, 1)
error.boost = mean(yhat.boost!=hill_valley.test$class)
error.boost
```

Result : ***The error rate considering all variables in boosting is calculated to be 47.54%.***

Now calculating the error rate using only the significant variables-

```
set.seed(1)
boost_hill_valley_new = gbm(class~ X1+X3+X2+X6+X11+X7+X70
,data=hill_valley.train, distribution = "gaussian", n.trees =5000 ,
interaction.depth =4)
summary(boost_hill_valley_new)
```

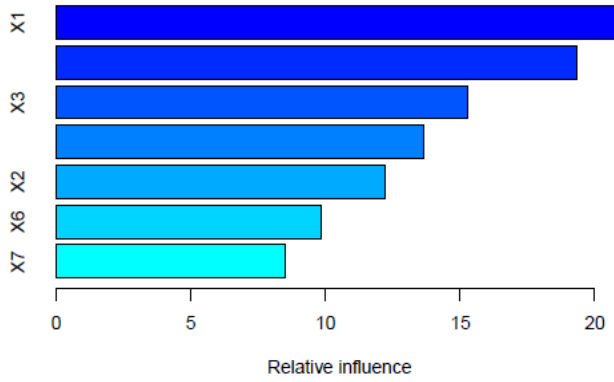


Fig 6: Influence of significant predictors on response

```
yhat.boost = predict (boost_hill_valley_new,newdata = hill_valley.test,
n.trees =5000)
yhat.boost_new = ifelse(yhat.boost< 1.02, yhat.boost==0, 1)
error.boost_new = mean(yhat.boost_new!=hill_valley.test$class)
error.boost_new
```

Result: ***The error rate is 46.72%. The error rate with boosting taking in consideration only the significant variables obtained during the first case in boosting is 45%. Therefore, considering fewer variables with higher relative influence gives a better prediction performance.***

II. Linear Discriminant Analysis:

The LDA is applied to the dataset as follows-

```
lda.fit =lda(class~., data = hill_valley.train)
lda.pred = predict(lda.fit, hill_valley.test)
names(lda.pred)

lda.class = lda.pred$class
View(lda.class)
table(lda.class,hill_valley.test$class)
```

Result:

Class	0	1
0	56	39
1	4	23

a) To Calculate the error rate for LDA:

```
error.lda = mean(lda.class!=hill_valley.test$class)
error.lda
```

Error rate = 35.24%

b) To calculate Accuracy for LDA

```
accuracy.lda = mean(lda.class==hill_valley.test$class)
accuracy.lda
```

Accuracy= 64.75%

c) Plotting ROC Curve:

```
LDA.pred = lda.pred$posterior[,2]
roc.curve = function(s,print=FALSE){
```

```
Ps = (LDA.pred>s)*1
FP=sum((Ps==1)*(hill_valley.test$class==0))/sum(hill_valley.test$cla
ss==0)
TP=sum((Ps==1)*(hill_valley.test$class==1))/sum(hill_valley.test$cla
ss==1)
if(print==TRUE){
print(table(Observed=hill_valley.test$class, Predicted=Ps))
}
vect = c(FP, TP)
names(vect) = c("FPR", "TPR")
return(vect)
}
threshold = 0.5
roc.curve(threshold, print=TRUE)
ROC.curve = Vectorize(roc.curve)
M.ROC = ROC.curve(seq(0, 1, by=0.01))
plot(M.ROC[1,],M.ROC[2,],col = "grey", lwd = 2, type = "l" ,xlab =
"False positive rate" ,ylab = "True positive rate")
```

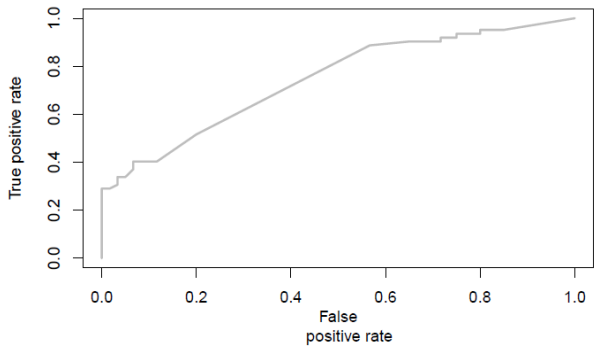


Fig 7 : ROC curve for LDA

Calculating Area under curve:

```
roc_obj = roc(hill_valley.test$class, lda.pred$posterior[,2])
auc(roc_obj)
```

Area under curve = 77.53 Sq. units

III. Quadratic Discriminant Analysis:

The QDA is applied to the dataset as follows-

```
qda.fit = qda(class~ ., data = hill_valley.train)
qda.fit
qda.pred = predict (qda.fit, hill_valley.test)
names(qda.pred)
as.data.frame(qda.pred)
qda.class = qda.pred$class
```

```
View(qda.class)
table(qda.class, hill_valley.test$class)
```

Results:

qda.class	0	1
0	31	30
1	29	32

a) To Calculate the error rate for QDA:

```
error.qda = mean(qda.class!=hill_valley.test$class)
error.qda
```

Error rate = 48.36%

b) Plotting ROC Curve:

```
QDA.pred = qda.pred$posterior[,2]
roc.curve = function(s, print = FALSE){
Ps = (QDA.pred>s)*1
FP =
sum((Ps==1)*(hill_valley.test$class==0))/sum(hill_valley.test$class==0)
TP =
sum((Ps==1)*(hill_valley.test$class==1))/sum(hill_valley.test$class==1)
if(print==TRUE){
print(table(Observed = hill_valley.test$class, Predicted=Ps))
}

vect = c(FP, TP)
names(vect) = c("FPR", "TPR")
return(vect)
}

threshold = 0.5
roc.curve(threshold, print=TRUE)

ROC.curve = Vectorize(roc.curve)
M.ROC = ROC.curve(seq(0, 1, by = 0.01))


plot(M.ROC[1,], M.ROC[2,], col="grey", lwd=2, type="l", xlab =
"False positive rate", ylab = "True positive rate")
```

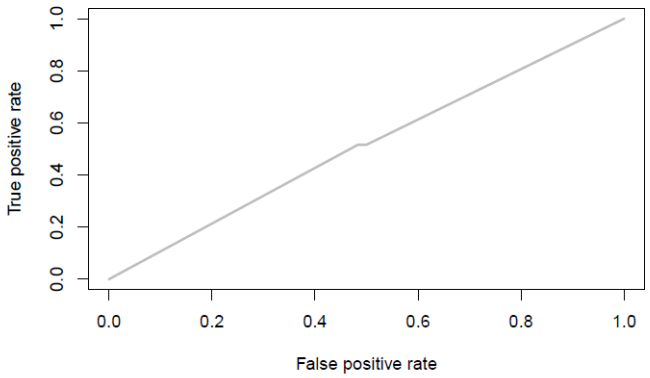


Fig 8 : ROC curve for QDA

c) Calculating area under curve:

```
roc_obj = roc(hill_valley.test$class, qda.pred$posterior[,2])
auc(roc_obj)
```

Area under curve = 0.4832 sq units

IV. Logistic Regression:

The Logistic Regression is applied to the dataset as follows-

```
glm.fit = glm(class ~., data = hill_valley.train, family
="binomial")
summary(glm.fit)
glm.probs = predict(glm.fit, hill_valley.test, type = "response")
glm.pred = predict(glm.fit, hill_valley.test)
glm.pred

glm.pred = rep("0",length(glm.probs))
glm.pred[glm.probs > 0.5] = 1
table(glm.pred, hill_valley.test$class)
```

Results:

glm.pred	0	1
0	18	6
1	42	56

a) To Calculate the error rate for Logistic Regression

```
error.glm = mean(glm.pred!=hill_valley.test$class)
error.glm
```

Error rate = 39.34%

b) To calculate Accuracy for Logistic Regression

```
accuracy.glm = mean(glm.pred==hill_valley.test$class)
accuracy.glm
```

Accuracy= 60.65%

c) Plotting ROC Curve:

```
roc.curve = function(s,print = FALSE){
Ps = (glm.probs>s)*1
FP=sum((Ps==1)*(hill_valley.test$class==0))/sum(hill_valley.test$cla
ss==0)
TP=sum((Ps==1)*(hill_valley.test$class==1))/sum(hill_valley.test$cla
ss==1)
if(print==TRUE){
print(table(Observed=hill_valley.test$class,Predicted=Ps))
}
vect=c(FP,TP)
names(vect)=c("FPR","TPR")
return(vect)
}
threshold=0.5
roc.curve(threshold,print=TRUE)
ROC.curve=Vectorize(roc.curve)
M.ROC=ROC.curve(seq(0,1,by=0.01))
plot(M.ROC[1,],M.ROC[2,],col="grey",lwd=2,type="l" ,xlab="False
positive rate" ,ylab="True positive rate")
```

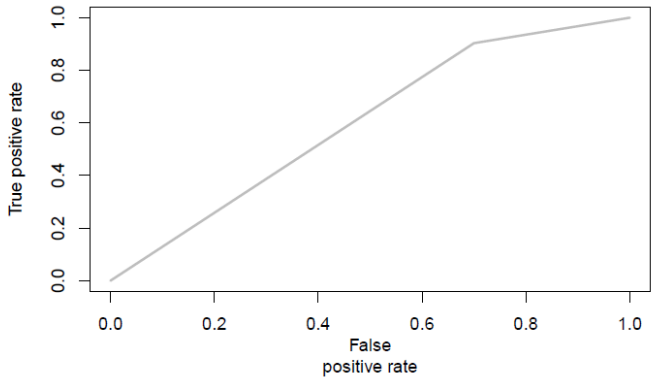


Fig 9 : ROC curve for Logistic Regression

d) Calculating area under curve:

```
roc_obj = roc(hill_valley.test$class, glm.pred$posterior[,2])
auc(roc_obj)
```

Area under curve = 5.06 sq units

V. K- Nearest Neighbors:

When the KNN algorithm is applied to a classification problem, the output is assignment of the class to each of the observations. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. We have used k= 5 for performing KNN analysis. The following code was used to train the KNN model.

```
knn.train = hill_valley.train
knn.test = hill_valley.test
train.class = class[train]

knn.pred = knn(data.frame(knn.train),data.frame(knn.test),
train.class, k=5)
table(knn.pred,hill_valley.test$class)
```

knn.pred	0	1
0	31	27
1	29	35

```
as.data.frame(hill_valley.test)
```

a) The test error obtained by using the KNN is as follows-

```
error.knn = mean(knn.pred!=hill_valley.test$class)
error.knn
```

Error rate = 45.90%

b) The accuracy using KNN is calculated as-

```
accuracy.knn = mean(knn.pred==hill_valley.test$class)
accuracy.knn
```

Accuracy= 54.09%

VI. Support Vector Classifier:

We used linear kernel to fit a support vector classifier as follows-

```
svc.fit = svm(class ~ ., data = hill_valley.train, kernel =
"linear", cost = 10, scale = FALSE)
set.seed(1)
```

The tune() function is used to perform the 10-fold cross validation to determine the best value of cost function C.

```
tune.out = tune(svm,class ~ .,data = hill_valley.train,
kernel = "linear", ranges = list(cost=c(0.01, 0.1, 1, 5,10,100)))
summary(tune.out)
```

The best cost was obtained to be 100.
Now the best model is obtained using this cost.

```
bestmod = tune.out$best.model
summary(bestmod)
```

Number of support vectors = 336.

The training error determined from confusion matrix is as follows-

```
ypred = predict(bestmod, hill_valley.test, type ="class")
table(ypred, hill_valley.test$class)
```

ypred	0	1
0	54	43
1	6	19

a) Error Rate:

```
error.svc = mean(ypred!= hill_valley.test$class)
error.svc
```

The test error rate obtained was 40.16%.

b) Accuracy:

```
accuracy.svc = mean(ypred== hill_valley.test$class)
accuracy.svc
```

The accuracy obtained was 59.83%

VII. Support Vector Machine (Radial Kernel)

It has an advantage over SVC because it helps classify data using radial and polynomial kernels. The data which is not linearly separated will be highly misclassified using a linear leading to high error rates. Hence, in such situations, using SVM is a better choice. The Support Vector Margin using Radial kernel was implemented as follows

```
svm.fit = svm(class ~ ., data = hill_valley.train, kernel =
"radial", gamma = 1, cost = 1)
```

The best value of cost c is calculated by choosing different values of gamma and c.

```
tune.out = tune(svm, class ~ ., data = hill_valley.train,
kernel = "radial", ranges = list(cost=c(0.1,1,5,10,100), gamma =
c(0.5,1,2,3,4)))
summary(tune.out)
```

The best parameters are obtained to be gamma=2 and cost= 100.

```
bestmod.rad = tune.out$best.model
summary(bestmod.rad)
```

No. of support vectors obtained = 430. And the error rate for best model is obtained to be 39.26%.

Now we calculate other performance parameters using best model-

```
pred.svm.rad = predict(bestmod.rad, hill_valley.test, type =
"class")
table(pred.svm.rad, hill_valley.test$class)
```

```
pred.svm.rad     0      1
           0     45     32
           1     15     3
```

a) The test error rate using the cross validation for SVM is obtained below-

```
error.svm = mean(pred.svm.rad!=hill_valley.test$class)
error.svm
```

The test error is obtained to be 38.52%.

b) Accuracy:

```
accuracy.svm = mean(pred.svm.rad==hill_valley.test$class)
accuracy.svm
```

The accuracy is obtained as 61.47%.

VIII. Support vector margin (Polynomial Kernel)

The SVM is an extension of the SVC that results from enlarging the feature space in a specific way, using kernels. In this method, we have used polynomial kernel.

```
svm.fit.poly = svm(class ~ ., data = hill_valley.train, kernel =
"polynomial", gamma = 1, cost = 1)

tune.out.poly = tune(svm, class ~ ., data = hill_valley.train, kernel =
"polynomial",

ranges = list(cost=c(0.1,1,5,10,100), gamma = c(0.5,1,2,3,4)))
summary(tune.out.poly)
```

The best values using cross validation is obtained as c= 10 and gamma= 3 and error rate = 39.26%.

```
bestmod.poly = tune.out.poly$best.model
summary(bestmod.poly)
```

The best model is obtained with support vectors = 194.

Using this model, we further calculate the performance parameters-

```
pred.svm.poly = predict(bestmod.poly, hill_valley.test, type = "class")
table(pred.svm.poly, hill_valley.test$class)
```

pred.svm.poly	0	1
0	58	20
1	2	42

a) Test error:

```
error.svm.poly = mean(pred.svm.poly!=hill_valley.test$class)
error.svm.poly
```

The test error rate is obtained to be 18.3%.

b) Accuracy:

```
accuracy.svm.poly = mean(pred.svm.poly==hill_valley.test$class)
accuracy.svm.poly
```

The accuracy is obtained as 81.96%.

IX. Classification trees

For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs. The classification tree is applied to the given dataset as follows-

```
tree.hill_valley = tree(class ~ ., hill_valley.train)
summary(tree.hill_valley)
```

Results:
Number of terminal nodes: 7
Residual mean deviance: 1.289
Misclassification error rate: 0.4008

```
tree.pred = predict(tree.hill_valley, hill_valley.test, type ="class")
table(tree.pred, hill_valley.test$class)
```

Using the predicted response of the model we calculate the performance parameters using the confusion matrix-

tree.pred	0	1

```
0      36     31
1      24     31
```

a) Error rate:

```
error.tree = mean(tree.pred!=hill_valley.test$class)
error.tree
```

The error rate is obtained as 45.08%.

b) Accuracy:

```
accuracy.tree = mean(tree.pred==hill_valley.test$class)
accuracy.tree
```

The Accuracy is obtained to be 54.91%.

c) Plotting the tree:

```
plot(tree.hill_valley)
text(tree.hill_valley, pretty = 0)
```

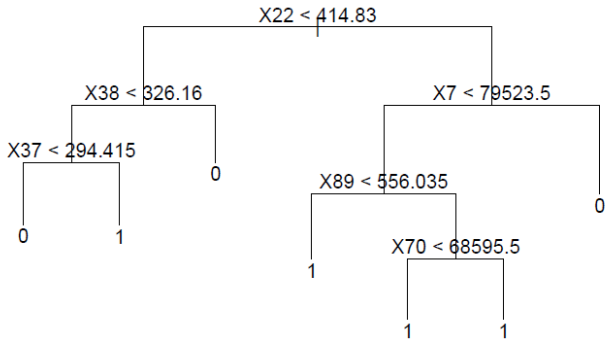


Fig 10: Classification Tree

d) Pruning the tree:

```
prune = cv.tree(tree.hill_valley, FUN = prune.misclass)
plot(prune$size, prune$dev, type="b")
```

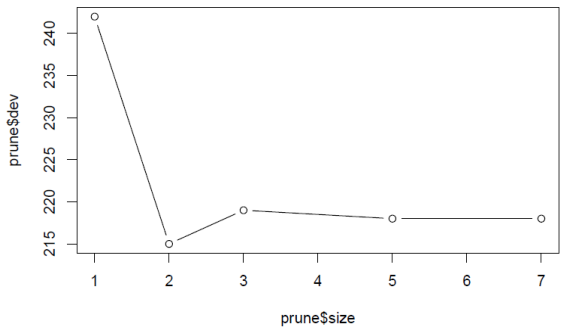


Fig 11 : Tree pruning

X. Neural Network

Neural networks are an arrangement of calculations, demonstrated freely after the human mind, that are intended to perceive designs. They decipher tactile information through a sort of machine recognition, marking or grouping crude information. The examples they perceive are numerical, contained in vectors, into which all certifiable information, be it pictures, sound, content or time arrangement, must be deciphered

```
require(neuralnet)

require(nnet)

require(ggplot2)

set.seed(1)
train <- cbind(hill_valley.train[, 1:100],
class.ind(as.factor(hill_valley.train$class)))
## Set labels name
names(train) <- c(names(hill_valley.train)[1:100],"l1","l2")

## Scale data
scl <- function(x){ (x - min(x))/(max(x) - min(x)) }
train[, 1:100] <- data.frame(lapply(train[, 1:100], scl))
head(train)

##  Set up formula
n <- names(train)
f <- as.formula(paste("l1 + l2  ~", paste(n[!n %in% c("l1","l2")], collapse =
" + ")))
f

nn <- neuralnet(f,
                data = train,
                hidden =
c(97,94,91,90,87,84,81,78,75,72,69,66,63,60,57,54,51,48,45,42,39,36,33,30,27,
24,21,18,15,12,9,6,3),
                act.fct = "logistic",
                linear.output = FALSE,
                lifesign = "minimal")

## Compute predictions
pr.nn <- compute(nn, train[, 1:100])
# Extract results
pr.nn_ <- pr.nn$net.result
head(pr.nn_)

# Accuracy (training set)
original_values <- max.col(train[, 101])
```

```
pr.nn_2 <- max.col(pr.nn_)
mean(pr.nn_2 == original_values)

##Set seed for reproducibility purposes
set.seed(500)
# 10 fold cross validationk <- 10
# Results from cv
outs <- NULL
##Train test split proportions
proportion <- 0.95 # Set to 0.995 for LOOCV
for(i in 1:k)
{
  index <- sample(1:nrow(train), round(proportion*nrow(train)))
  train_cv <- train[index, ]
  test_cv <- train[-index, ]
  nn_cv <- neuralnet(f,
                     data = train_cv,
                     hidden =
c(97,94,91,90,87,84,81,78,75,72,69,66,63,60,57,54,51,48,45,42,39,36,33,30,27,
24,21,18,15,12,9,6,3),
                     act.fct = "logistic",
                     linear.output = FALSE)

  ## Compute predictions
  pr.nn <- compute(nn_cv, test_cv[, 1:100])
  # Extract results
  pr.nn_ <- pr.nn$net.result
  # Accuracy (test set)
  original_values <- max.col(test_cv[, 101])
  pr.nn_2 <- max.col(pr.nn_)
  outs[i] <- mean(pr.nn_2 == original_values)
}

a<-mean(outs)
```

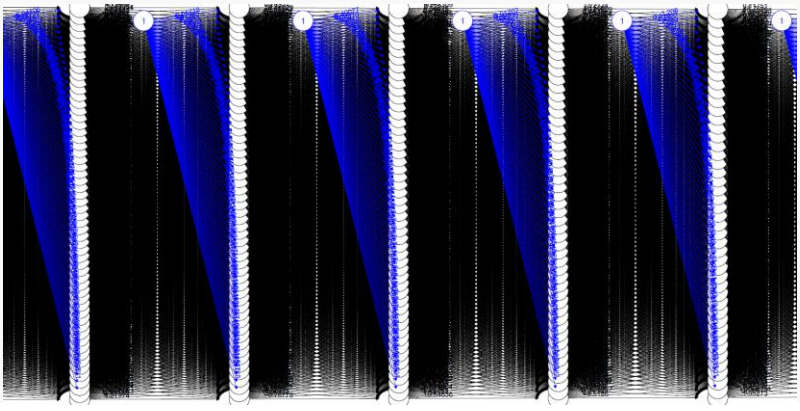


Fig 12 : Neural Network

Table 1: Comparison of performance measures of different classification methods

Sr. No	Classification Method	Test Performance
1	LDA	Error rate = 35.24% ; Sensitivity= 37.09% Specificity= 93.33 % ; AUC =0.7753 sq.units
2	QDA	Error rate = 48.36% ;Sensitivity= 51.61% Specificity= 51.66 % ; AUC =0.4836 sq.units
3	Logistic Regression	Error rate = 39.34% ;Sensitivity= 90.32% Specificity= 30 %
4	KNN	Error rate = 45.90% ;Sensitivity= 56.45% Specificity= 51.66 %
5	Random Forest	Error rate = 45.08% ;Sensitivity= 58.06% Specificity= 50 %
6	Boosting	Error rate = 47.54%
7	SVC	Error rate = 40.16% ;Sensitivity= 30.64% Specificity= 90 %
8	SVM (Radial)	Error rate = 39.26% ;Sensitivity= 48.38% Specificity= 75 %
9	SVM (Polynomial)	Error rate = 18.03% ;Sensitivity= 67.74% Specificity= 96.67 %
10	Classification Tree	Error rate = 45.08% ;Sensitivity= 50.0% Specificity= 60.0 %
11	Neural Network	Accuracy= 90%

Table 2: Table of Confusion Matrix

<table>
<tr><th>Method</th><th>Test Performance</th><th>Method</th><th>Test Performance</th></tr>
<tr>
<td>LDA</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>58</td><td>39</td></tr>
<tr><td>1</td><td>4</td><td>23</td></tr>
</table>
False positive rate: 6.67 %
</td>
<td>Random Forest</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>30</td><td>25</td></tr>
<tr><td>1</td><td>30</td><td>37</td></tr>
</table>
False Positive rate: 50.00 %
</td>
</tr>
<tr>
<td>QDA</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>31</td><td>30</td></tr>
<tr><td>1</td><td>29</td><td>32</td></tr>
</table>
False positive rate: 48.33 %
</td>
<td>SVC</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>54</td><td>43</td></tr>
<tr><td>1</td><td>6</td><td>19</td></tr>
</table>
False Positive rate: 10.0 %
</td>
</tr>
</table>

<table>
<tr>
<td>Logistic Regression</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>18</td><td>6</td></tr>
<tr><td>1</td><td>42</td><td>56</td></tr>
</table>
False positive rate: 70.00 %
</td>
<td>SVM (Radial)</td>
<td>
Error = 0.3555556
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>45</td><td>32</td></tr>
<tr><td>1</td><td>15</td><td>30</td></tr>
</table>
False Positive rate: 25.00%
</td>
</tr>
<tr>
<td>KNN</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>31</td><td>27</td></tr>
<tr><td>1</td><td>29</td><td>35</td></tr>
</table>
False Positive rate: 48.33 %
</td>
<td>SVM (Poly)</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>58</td><td>20</td></tr>
<tr><td>1</td><td>2</td><td>42</td></tr>
</table>
False Positive rate: 3.33 %
</td>
</tr>
<tr>
<td>Classification Tree</td>
<td>
<table>
<tr><th></th><th colspan="2">Actual</th></tr>
<tr><th>Predicted</th><th>0</th><th>1</th></tr>
<tr><td>0</td><td>36</td><td>31</td></tr>
<tr><td>1</td><td>24</td><td>31</td></tr>
</table>
False Positive rate: 40.00 %
</td>
<td></td>
<td></td>
</tr>
</table>

7. EXECUTIVE SUMMARY

The dataset that we chose for this project is a data of Hill and valley evaluation. This data was collected on the basis of different topographical parameters at different places around the world. This data gives an interpretation of the presence of hill or valley region at that particular place. The geographical importance and prediction of a particular data value can be huge and the prediction results can be used for many more applications apart from the physical ones. Some of the major real-life applications are as follows-

1. The determination of catchment areas for dams, bunds and irrigation facilities, etc.
2. Farming land determination depending upon the type of topography required for a particular plantation.
3. The type of building structure and material to be used depending upon the geographical land type.
4. Determination of regions wherein different types of renewable energy can be used, for example, in hilly areas, wind energy has high development potential and hydropower will have high potential in valley or low lying areas.

From the above example, it can easily be seen that the predicting the type of topography helps serve many industries in real life.

In the existing literature available, there are no methods available to predict the topography of the region. The innovative classification techniques that we used in the project, is way more far sighted and provides better prediction accuracies as compared to any other work done till now. Moreover, the project work not only works towards plain prediction but also takes into account interpretability of the data by taking into account methods like tree classification, random forest, etc. Also, the inclusion of modern data prediction techniques like neural network helps give more robust and dependable results.

The objective we set at the beginning of the project was to clean and classify the data into two classes and perform various classification analysis on it. By looking at the results of the project, we can conclude that we have been successful in achieving the objectives set and achieve the best possible results using all the techniques applied. The future scope of work can be to iterate the methods further and trying to improve the accuracies further and reduce the error rates involved.

8. CONCLUSION

- From the above report it can be seen that deploying different prediction methods can help detect whether a given terrain is a hill or a valley.
- Each method has its own advantages or disadvantages like a method might have high interpretability but lower accuracy or vice versa. Depending upon the requirement a suitable method can be chosen.
- However, for the given scenario accuracy is a more vital criteria and hence the model built using Neural Network seems to be the most preferable one.
- Giving an accuracy of about 90% Neural Network outperforms most of the other methods.
- The presence of large number of observations as well as hidden layers helps neural network better understand the small intricacies in the data
- At the same time with further accumulation of data a better understanding of the actual relationship between the output and the inputs can be obtained thus enabling a further scope of development.

REFERENCES

[1] *http://archive.ics.uci.edu/ml/datasets/hill-valley*

[2] Gareth James, Daniela Witten, Trevor Hastie, Robert, Tibshirani, “An Introduction to statistical learning” © Springer Science+Business Media New York 2013

[3] *https://pdfs.semanticscholar.org/f90a/76920679944ebe48142c74daa7e317f605c1.pdf*

[4] S. Fritsch and F. Günther. neuralnet: Training of Neural Networks. R Foundation for Statistical Computing, 2008. R package version 1.33

[5] Intrator O. and Intrator N. (1993) Using Neural Nets for Interpretation of Nonlinear Models. Proceedings of the Statistical Computing Section, 244-249 San Francisco: American Statistical Society (eds).

APPENDIX

```
library(dplyr)
hill_valley =
read.csv("file:///C:/Users/Akshay/Desktop/hill
_valley.csv")
attach(hill_valley)
dim(hill_valley)
is.factor(hill_valley$class)
hill_valley$class =
as.factor(hill_valley$class)

# Data Cleaning
which(is.na(hill_valley))
neg.value = apply(hill_valley, 1,
function(row) any(row < 0)
neg.value

#SPLITTING DATA INTO TRAINING DATA AND TEST
DATA

library (ISLR)
set.seed (1)
train =
sample(nrow(hill_valley),nrow(hill_valley)*0.8
)
hill_valley.train = hill_valley[train, ]
hill_valley.test = hill_valley[-train, ]
nrow(hill_valley.test)

dim(hill_valley.train)
dim(hill_valley.test)

#Linear Discriminant Analysis (LDA)
library(MASS)
hill_valley = unlist(hill_valley)
lda.fit =lda(class~., data =
hill_valley.train)
lda.pred = predict(lda.fit, hill_valley.test)
names(lda.pred)
as.data.frame(lda.pred)
lda.pred$class
lda.pred$posterior
lda.class = lda.pred$class
View(lda.class)
table(lda.class,hill_valley.test$class)
error.lda =
mean(lda.class!=hill_valley.test$class)
error.lda
accuracy.lda =
mean(lda.class==hill_valley.test$class)
accuracy.lda
TPR.lda = (23)/(23+39)
TPR.lda
sensitivity.lda = TPR.lda
sensitivity.lda
FPR.lda = (4)/(56+4)
FPR.lda
specificity.lda = (1-FPR.lda)
specificity.lda
LDA.pred =  lda.pred$posterior[,2]
roc.curve = function(s,print=FALSE){
  Ps = (LDA.pred>s)*1
FP=sum((Ps==1)*(hill_valley.test$class==0))/su
m(hill_valley.test$class==0)

TP=sum((Ps==1)*(hill_valley.test$class==1))/su
m(hill_valley.test$class==1)
  if(print==TRUE){

print(table(Observed=hill_valley.test$class,
Predicted=Ps))
  }
  vect = c(FP, TP)
  names(vect) = c("FPR", "TPR")
  return(vect)
}
threshold = 0.5
roc.curve(threshold, print=TRUE)
ROC.curve = Vectorize(roc.curve)
M.ROC = ROC.curve(seq(0, 1, by=0.01))
plot(M.ROC[1,],M.ROC[2,],col = "grey", lwd =
2, type = "l",xlab = "False positive rate"
,ylab = "True positive rate")

#Quadratic Discriminant Analysis
library(pROC)
roc_obj = roc(hill_valley.test$class,
lda.pred$posterior[,2])
auc(roc_obj)
qda.fit = qda(class~ ., data =
hill_valley.train)
qda.fit
qda.pred$class
qda.pred$posterior
qda.class = qda.pred$class
View(qda.class)
table(qda.class,
hill_valley.test$class)error.qda =
mean(qda.class!=hill_valley.test$class)
error.qda
accuracy.qda =
mean(qda.class==hill_valley.test$class)
accuracy.qda
TPR.qda = (32)/(32+30)
TPR.qda
sensitivity.qda = TPR.qda
sensitivity.qda
FPR.qda = (29)/(29+31)
FPR.qda
specificity.qda = (1-FPR.qda)
specificity.qda
QDA.pred =  qda.pred$posterior[,2]
roc.curve = function(s, print = FALSE){
  Ps = (QDA.pred>s)*1
  FP =
sum((Ps==1)*(hill_valley.test$class==0))/sum(h
ill_valley.test$class==0)
  TP =
sum((Ps==1)*(hill_valley.test$class==1))/sum(h
ill_valley.test$class==1)
  if(print==TRUE){
    print(table(Observed =
hill_valley.test$class, Predicted=Ps))
  }
```

```
  vect = c(FP, TP)
  names(vect) = c("FPR", "TPR")
  return(vect)
}
threshold = 0.5
roc.curve(threshold, print=TRUE)
ROC.curve = Vectorize(roc.curve)
M.ROC = ROC.curve(seq(0, 1, by = 0.01))
plot(M.ROC[1,], M.ROC[2,], col="grey", lwd=2,
type="l", xlab = "False positive rate", ylab =
"True positive rate")
library(pROC)
roc_obj = roc(hill_valley.test$class,
qda.pred$posterior[,2])
auc(roc_obj)
glm.fit = glm(class ~., data =
hill_valley.train, family ="binomial")
summary(glm.fit)
glm.probs = predict(glm.fit, hill_valley.test,
type = "response")
glm.pred = predict(glm.fit, hill_valley.test)
glm.pred
glm.pred = rep("0",length(glm.probs))
glm.pred[glm.probs > 0.5] = 1
table(glm.pred, hill_valley.test$class)
error.glm =
mean(glm.pred!=hill_valley.test$class)
error.glm
accuracy.glm =
mean(glm.pred==hill_valley.test$class)
accuracy.glm
TPR.glm = (56)/(56+6)
TPR.glm
sensitivity.glm = TPR.glm
sensitivity.glm
FPR.glm = (42)/(18+42)
FPR.glm
specificity.glm = (1-FPR.glm)
specificity.glm
roc.curve = function(s,print = FALSE){
  Ps = (glm.probs>s)*1

FP=sum((Ps==1)*(hill_valley.test$class==0))/su
m(hill_valley.test$class==0)

TP=sum((Ps==1)*(hill_valley.test$class==1))/su
m(hill_valley.test$class==1)
  if(print==TRUE){

print(table(Observed=hill_valley.test$class,Pr
edicted=Ps))
  }
  vect=c(FP,TP)
  names(vect)=c("FPR","TPR")
  return(vect)
}
threshold=0.5
roc.curve(threshold,print=TRUE)
ROC.curve=Vectorize(roc.curve)
M.ROC=ROC.curve(seq(0,1,by=0.01))
plot(M.ROC[1,],M.ROC[2,],col="grey",lwd=2,type
="l"
     ,xlab="False
positive rate"
     ,ylab="True positive rate")

#KNN

library(class)
knn.train = hill_valley.train   knn.test =
hill_valley.test
train.class = class[train]
knn.pred = knn(knn.train, knn.test,
train.class, k=1)
table(knn.pred,
hill_valley.test$class)knn.pred =
knn(data.frame(knn.train),data.frame(knn.test)
, train.class, k=5)
table(knn.pred,hill_valley.test$class)
as.data.frame(hill_valley.test)
error.knn =
mean(knn.pred!=hill_valley.test$class)
error.knn
accuracy.knn =
mean(knn.pred==hill_valley.test$class)
accuracy.knn
TPR.knn = (35)/(35+27)
TPR.knn
sensitivity.knn = TPR.knn
sensitivity.knn
FPR.knn = (29)/(29+31)
FPR.knn
specificity.knn = (1-FPR.knn)
specificity.knn

#Random Forest

library(randomForest)
set.seed(1)
rf.hill_valley = randomForest(class ~ ., data
= hill_valley.train, mtry = 10,  n.trees = 50,
importance = TRUE)
best.tree = tuneRF(hill_valley.train,
hill_valley.train$class, stepFactor = 1.1,
improve = 0.05)
prediction = predict(rf.hill_valley,
hill_valley.test, type = "class")
table(predicted = prediction, actual =
hill_valley.test$class)
error.rf =
mean(prediction!=hill_valley.test$class)
error.rf
accuracy.rf =
mean(prediction==hill_valley.test$class)
accuracy.rf
TPR.rf = (36)/(36+26)
TPR.rf
sensitivity.rf = TPR.rf
sensitivity.rf
FPR.rf = (30)/(30+30)
FPR.rf
specificity.rf = (1-FPR.rf)
specificity.rf
importance(rf.hill_valley)

#Random Forest
```

```
library(randomForest)
set.seed(1)
rf.hill_valley_cv = randomForest(class ~
X7+X2+X97+X1+X70,  data = hill_valley.train,
mtry = 10, n.trees = 50, importance = TRUE)
best.tree = tuneRF(hill_valley.train,
hill_valley.train$class, stepFactor = 1.1,
improve = 0.05)
pred.rf = predict(rf.hill_valley_cv,
hill_valley.test, type = "class")
table(pred.rf, hill_valley.test$class)
error.rf =
mean(prediction!=hill_valley.test$class)
error.rf
accuracy.rf =
mean(prediction==hill_valley.test$class)
accuracy.rf

#Boosting
library (gbm)
set.seed (1)
boost_hill_valley = gbm(class~.,data =
hill_valley.train,
distribution="gaussian",n.trees = 5000 ,
interaction.depth = 4, shrinkage = 0.019,
verbose = F)
summary(boost_hill_valley)
yhat.boost=predict (boost_hill_valley,newdata
=hill_valley.test,n.trees = 5000)
yhat.boost = ifelse(yhat.boost< 1.02,
yhat.boost==0, 1)
error.boost =
mean(yhat.boost!=hill_valley.test$class)
error.boost
set.seed(1)
boost_hill_valley_new = gbm(class~
X1+X3+X2+X6+X11+X7+X70,data=hill_valley.train,
distribution = "gaussian",n.trees =5000 ,
interaction.depth =4)
summary(boost_hill_valley_new)
yhat.boost = predict
(boost_hill_valley_new,newdata =
hill_valley.test, n.trees =5000)
yhat.boost_new = ifelse(yhat.boost< 1.02,
yhat.boost==0, 1)
error.boost_new =
mean(yhat.boost_new!=hill_valley.test$class)
error.boost_new

#SVC by using Kernel
library(e1071)
svc.fit = svm(class ~ ., data =
hill_valley.train, kernel = "linear",
              cost = 10, scale = FALSE)
set.seed(1)
tune.out = tune(svm,class ~ .,data =
hill_valley.train, kernel = "linear", ranges =
list(cost=c(0.01, 0.1, 1, 5,10,100)))
summary(tune.out)
ypred = predict(bestmod, hill_valley.test,
type ="class")
table(ypred, hill_valley.test$class)
error.svc = mean(ypred!=
hill_valley.test$class)
error.svc
accuracy.svc = mean(ypred==
hill_valley.test$class)
accuracy.svc

#SVM using Radial

library(e1071)
set.seed(1)

tune.out = tune(svm, class ~ ., data =
hill_valley.train, kernel = "radial", ranges =
list(cost=c(0.1,1,5,10,100), gamma =
c(0.5,1,2,3,4)))
summary(tune.out)
pred.svm.rad = predict(bestmod.rad,
hill_valley.test, type = "class")


TPR.svc = (19)/(19+43)
TPR.svc
sensitivity.svc = TPR.svc
sensitivity.svc
FPR.svc = (6)/(6+54)
FPR.svc
svm.fit = svm(class ~ ., data =
hill_valley.train, kernel = "radial", gamma =
1, cost = 1)
bestmod.rad = tune.out$best.model
summary(bestmod.rad)
specificity.svc = (1-FPR.svc)
specificity.svc

#SVM using polynomial kernel

library(e1071)
set.seed(1)
svm.fit.poly = svm(class ~ ., data =
hill_valley.train, kernel = "polynomial",
gamma = 1, cost = 1)
tune.out.poly = tune(svm, class ~ ., data =
hill_valley.train, kernel = "polynomial",
ranges = list(cost=c(0.1,1,5,10,100), gamma =
c(0.5,1,2,3,4)))
summary(tune.out.poly)
bestmod.poly = tune.out.poly$best.model
summary(bestmod.poly)
pred.svm.poly = predict(bestmod.poly,
hill_valley.test, type = "class")
table(pred.svm.poly, hill_valley.test$class)
error.svm.poly =
mean(pred.svm.poly!=hill_valley.test$class)
error.svm.poly
accuracy.svm.poly =
mean(pred.svm.poly==hill_valley.test$class)
accuracy.svm.poly
TPR.svm.poly = (42)/(42+20)
TPR.svm.poly
sensitivity.svm.poly = TPR.svm.poly
sensitivity.svm.poly
```

```
FPR.svm.poly = (2)/(2+58)
FPR.svm.poly
specificity.svm.poly = (1-FPR.svm.poly)
specificity.svm.poly

#Classification using Tree

library(tree)
library(ISLR)
tree.hill_valley = tree(class ~ .,
hill_valley.train)
summary(tree.hill_valley)
tree.pred = predict(tree.hill_valley,
hill_valley.test, type ="class")
table(tree.pred, hill_valley.test$class)
error.tree =
mean(tree.pred!=hill_valley.test$class)
error.tree
accuracy.tree =
mean(tree.pred==hill_valley.test$class)
accuracy.tree
TPR.tree = (31)/(31+31)
TPR.tree
sensitivity.tree = TPR.tree
sensitivity.tree
error.svm =
mean(pred.svm.rad!=hill_valley.test$class)
error.svm


sensitivity.svm = TPR.svm
sensitivity.svm
FPR.svm = (15)/(15+45)
FPR.svm
specificity.svm = (1-FPR.svm)
specificity.svm

#Neural Net

require(neuralnet)
require(nnet)
require(ggplot2)
set.seed(1)
train <- cbind(hill_valley.train[, 1:100],
class.ind(as.factor(hi
ll_valley.train$class)))
names(train) <-
c(names(hill_valley.train)[1:100],"l1","l2")
scl <- function(x){ (x - min(x))/(max(x) -
min(x)) }
train[, 1:100] <- data.frame(lapply(train[,
1:100], scl))
head(train)
n <- names(train)
f <- as.formula(paste("l1 + l2  ~", paste(n[!n
%in% c("l1","l2")], collapse = " + ")))nn <-
neuralnet(f,
                data = train,
                hidden =
c(97,94,91,90,87,84,81,78,75,72,69,66,63,60,57
,54,51,48,45,42,39,36,33,30,27,24,21,18,15,12,
9,6,3),
                act.fct = "logistic",
```

```
FPR.tree = (24)/(24+36)
FPR.tree
specificity.tree = (1-FPR.tree)
specificity.tree
plot(tree.hill_valley)
text(tree.hill_valley, pretty = 0)

set.seed(1)
prune = cv.tree(tree.hill_valley, FUN =
prune.misclass)

#Tree Pruning


plot(prune$size, prune$dev, type="b

accuracy.svm =
mean(pred.svm.rad==hill_valley.test$class)
accuracy.svm
table(pred.svm.rad, hill_valley.test$class)
```

```
                linear.output = FALSE,
                lifesign = "minimal")
pr.nn <- compute(nn, train[, 1:100])
pr.nn_ <- pr.nn$net.result
head(pr.nn_)
original_values <- max.col(train[, 101])
pr.nn_2 <- max.col(pr.nn_)
mean(pr.nn_2 == original_values)
set.seed(500)
k <- 10
outs <- NULL
proportion <- 0.95 # Set to 0.995 for LOOCV
for(i in 1:k)
{
  index <- sample(1:nrow(train),
round(proportion*nrow(train)))
  train_cv <- train[index, ]
  test_cv <- train[-index, ]
  nn_cv <- neuralnet(f,
                     data = train_cv,
                     hidden =
c(97,94,91,90,87,84,81,78,75,72,69,66,63,60,57
,54,51,48,45,42,39,36,33,30,27,24,21,18,15,12,
9,6,3),
                     act.fct = "logistic",
                     linear.output = FALSE)
  pr.nn <- compute(nn_cv, test_cv[, 1:100])
  pr.nn_ <- pr.nn$net.result
  original_values <- max.col(test_cv[, 101])
  pr.nn_2 <- max.col(pr.nn_)
  outs[i] <- mean(pr.nn_2 == original_values)
}
a<-mean(outs)
a
```