Data Classification on Zoo Dataset

By

Soumya Yamsani (700693403)

Ashwini Bhoomi Brahmanand(700693129)

Vineesha Paladugu(700691238)

2. Contribution to the project by each group member:

Soumya Yamsani:

Performed all the required classification techniques on Zoo Data set. Rechecked the solutions with teammates. Contribution in designing the project report and presentation .

Ashwini Bhoomi Brahmanand:

Performed all the required classification techniques on Zoo Data set. Rechecked the solutions with teammates. Contribution in designing the project report and presentation.

Vineesha Paladugu:

Performed Decision Tree techniques for Holdout, RandomForest and Bagging on Zoo Data Set. Rechecked the solutions with teammates

3. INTRODUCTION:

This is the smallest dataset belongs to Richard Forsyth . He donated this dataset to the UCI repository in 1990. This is a simple dataset containing 101 animals from a zoo. There are 18 different variables which describes the animals. Among them most of the attributes are Boolean valued attributes. This dataset does not contain any missing values and do not have uniformly distributed data. The main aim of this dataset is to classify a wide range of animals into 7 classes by the help of attributes that are related to animal characteristics There are 7 Class Types by which the animals are differentiated ,they are: Mammal, Bird, Reptile, Fish, Amphibian, Bug and Invertebrate.

The major purpose for this dataset is based on the variables, we can predict the classification of the animals. It dataset is perfect for the new Datamining Learners.

Information about Zoo DataSet:

Class: Set of animals

1 -- (41) aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose, opossum, oryx, platypus, polecat, pony, porpoise, puma, pussycat, raccoon, reindeer, seal, sealion, squirrel, vampire, vole, wallaby, wolf

2 -- (20) chicken, crow, dove, duck, flamingo, gull, hawk, kiwi, lark, ostrich, parakeet, penguin, pheasant, rhea, skimmer, skua, sparrow, swan, vulture, wren

3 -- (5) pitviper, seasnake, slowworm, tortoise, tuatara

4 -- (13) bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna

5 -- (4) frog, frog, newt, toad

6 -- (8) flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp

7 -- (10) clam, crab, crayfish, lobster, octopus, scorpion, seawasp, slug, starfish, worm.

Attribute Information:

1. animal name	Unique for each instance
2. hair	Boolean
3.feathers	Boolean
4. eggs	Boolean
5. milk	Boolean
6. airborne	Boolean
7. aquatic	Boolean
8. predator	Boolean
9. toothed	Boolean
10. backbone	Boolean
11. breathes	Boolean
12. venomous	Boolean
13. fins	Boolean
14. legs	Numeric (set of values: {0,2,4,5,6,8})
15. tail	Boolean
16. domestic	Boolean
17. catsize	Boolean
18. type	Numeric (integer values in range [1,7])

4. DATA PREPROCESSING:

- Initially when we read the zoo data set we found that there are 2 instances of "frog" . So we deleted the duplicate before performing any classification techniques on it.
- We dropped the column "animal name" as it didn't add value.
- We discretized the class label in our dataset which is "type" from ranges[0-7] to Boolean "Yes" and "No".

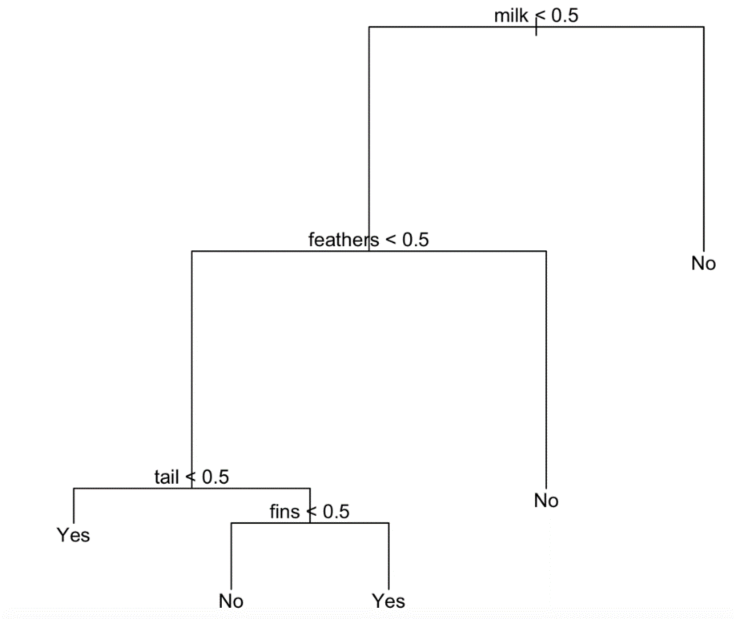
5. HOLDOUT METHOD:

Holdout is one of the methods to evaluate the classification accuracy technique. In this method initially the zoo dataset is randomly partitioned into two different independent sets. Training set for model construction and Test set for accuracy estimation. And also, the technique of Random sampling is implemented in this method, in which the hold out method is repeated on the same dataset for k number of times, where accuracy is the average of all accuracies which are obtained.

<u>Library used:</u> library(tree)

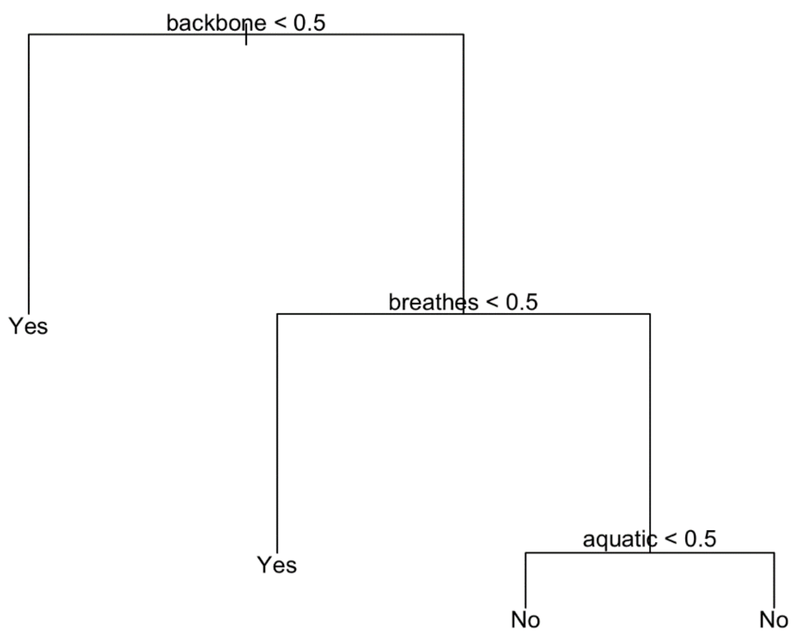
<u>Hold-Out Implementation:</u>

```
> library(tree)
>
> #To load the dataset
> zoo=read.csv("/Users/ashwinibhoomi/Desktop/SEM-3/Data Mining/Project/zoo.data",header=F)
>
> #Variable headers
> names(zoo)=c("animal", "hair", "feathers", "eggs", "milk", "airborne","aquatic", "predator", "toothed", "backbone", "breathes", "venomous",
+ "fins", "legs", "tail", "domestic", "size", "type")
>
> #To attach dataset
> attach(zoo)
>
> #To check missing values
> zoo=na.omit(zoo)
>
> #Deleting duplicate data
> zoo=zoo[-c(26),]
> rownames(zoo)=NULL
>
>
> #Dropping the column animal name
> zoo$animal=NULL
>
> #Discretization of class label
> response=ifelse(zoo$type<=3,"No","Yes")
> zoo=data.frame(zoo,response)
> zoo=zoo[,-17]
>
> #To check row and column dimensions
> dim(zoo)
[1] 100  17
>
> #To create a decision tree with response as the class label based on all other attributes
> tree.zoo=tree(response~.,zoo)
>
> #Summary of the created tree
> summary(tree.zoo)

Classification tree:
tree(formula = response ~ ., data = zoo)
Variables actually used in tree construction:
[1] "milk"     "feathers" "tail"     "fins"
Number of terminal nodes:  5
Residual mean deviance:  0.08817 = 8.376 / 95
Misclassification error rate: 0.02 = 2 / 100
>
> #To display the tree structures and node labels
> plot(tree.zoo)
> text(tree.zoo,pretty=0)
> tree.zoo
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 100 128.200 No ( 0.6600 0.3400 )
   2) milk < 0.5 59  80.410 Yes ( 0.4237 0.5763 )
     4) feathers < 0.5 39  29.870 Yes ( 0.1282 0.8718 )
       8) tail < 0.5 19   0.000 Yes ( 0.0000 1.0000 ) *
       9) tail > 0.5 20  22.490 Yes ( 0.2500 0.7500 )
        18) fins < 0.5 7   8.376 No ( 0.7143 0.2857 ) *
        19) fins > 0.5 13   0.000 Yes ( 0.0000 1.0000 ) *
     5) feathers > 0.5 20   0.000 No ( 1.0000 0.0000 ) *
   3) milk > 0.5 41   0.000 No ( 1.0000 0.0000 ) *
>
> #Testing the model using predict function
> tree.pred=predict(tree.zoo,type="class")
> tree.pred
  [1] No  No  Yes No  No  No  No  Yes Yes No  No  No  Yes Yes Yes Yes No  No  Yes No  No  No  No  No  Yes Yes No  No  No  Yes No  No  No  Yes No  No  No  Yes Yes
 [40] Yes No  Yes No  No  No  Yes No  No  No  No  Yes No  Yes No  No  No  No  No  No  Yes Yes No  No  No  No  No  No  No  No  No  No  No  Yes No  No  No  Yes No
 [79] No  No  Yes Yes No  No  Yes Yes No  Yes Yes No  No  Yes No  No  No  No  Yes No  Yes No
Levels: No Yes
```

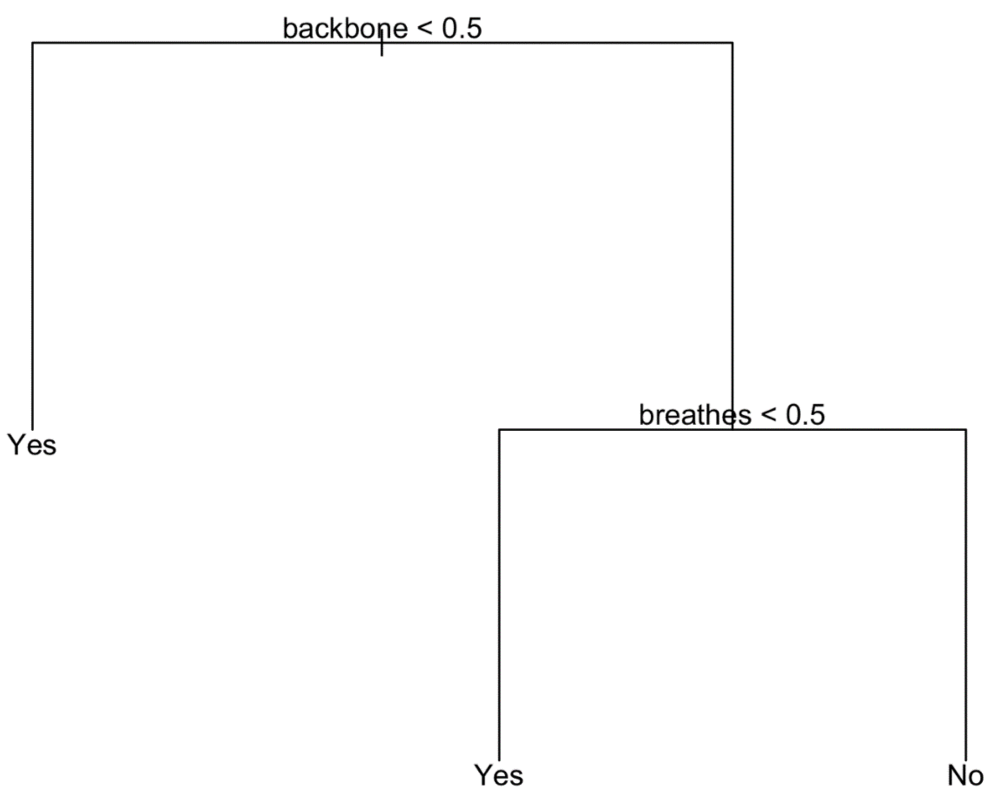



```
> #Confusion matrix
> table(tree.pred,response)
         response
tree.pred No Yes
      No  66   2
      Yes  0  32
>
> #Correct prediction rate
> mean(tree.pred==response)
[1] 0.98
>
> #Error prediction rate
> mean(tree.pred!=response)
[1] 0.02
>
> #Random select a sample of 60 observations of the data set as a training set and the rest
> #of the data set as a test set
> set.seed(123)
> train=sample(1:nrow(zoo), 60)
> zoo.train=zoo[train,]
> zoo.test=zoo[-train,]
> response.train=response[train]
> response.test=response[-train]
> tree.zoo=tree(response~.,zoo.train)
>
> #Train error
> tree.pred=predict(tree.zoo,zoo.train,type="class")
> table(tree.pred,response.train)
         response.train
tree.pred No Yes
      No  36   2
      Yes  1  21
> mean(tree.pred!=response.train)
[1] 0.05
>
> #Test error
> tree.pred=predict(tree.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  29   1
      Yes  0  10
> mean(tree.pred!=response.test)
[1] 0.025
>
```

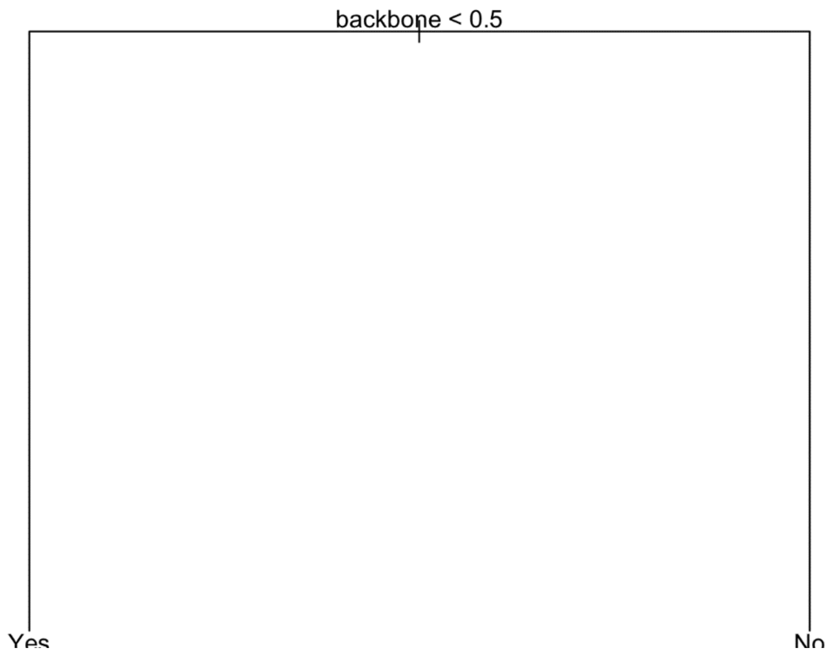
```
> set.seed(123)
> cv.zoo=cv.tree(tree.zoo,FUN=prune.misclass)
> cv.zoo
$size
[1] 4 3 2 1

$dev
[1]  5  5 13 24

$k
[1] -Inf    0    7   13

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
>
> plot(cv.zoo$size ,cv.zoo$dev ,type="b")
>
>
> #Check the tree with size 4
> set.seed(123)
> prune.zoo=prune.misclass(tree.zoo,best=4)
> plot(prune.zoo)
> text(prune.zoo,pretty=0)
> tree.pred=predict(prune.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  29   1
      Yes  0  10
> mean(tree.pred!=response.test)
[1] 0.025
```



```
> #Check the tree with size 3
> set.seed(123)
> prune.zoo=prune.misclass(tree.zoo,best=3)
> plot(prune.zoo)
> text(prune.zoo,pretty=0)
> tree.pred=predict(prune.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  29   1
      Yes  0  10
> mean(tree.pred!=response.test)
[1] 0.025
```



```
> set.seed(123)
> prune.zoo=prune.misclass(tree.zoo,best=2)
> plot(prune.zoo)
> text(prune.zoo,pretty=0)
> tree.pred=predict(prune.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  29   6
      Yes  0   5
> mean(tree.pred!=response.test)
[1] 0.15
>
```



<u>Test Error rates:</u>

	Tree Size	Test Error Rate
Before random sampling	-	0.025
After Random sampling	4	0.025
After Random sampling	3	0.025
After Random sampling	2	0.15

6. BAGGING:

Even Bagging is also one of the methods to evaluate the classification accuracy technique. It is the bootstrap aggregation technique. In this the zoo dataset is divided into few number of tuples at each iteration on which a bootstrap technique is implemented. From each training set we get a classifier model where models return its class prediction.

<u>Library used:</u> library(randomForest)

<u>Bagging Implementation:</u>

```
> #Check with 50 trees
> set.seed(123)
> tree.zoo=randomForest(response~.,zoo.train, ntree=30,mtry=7)
> tree.pred=predict(tree.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  29   1
      Yes  0  10
> mean(tree.pred!=response.test)
[1] 0.025
>
> #Check with 20 trees
> set.seed(123)
> tree.zoo=randomForest(response~.,zoo.train, ntree=10,mtry=7)
> tree.pred=predict(tree.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  28   1
      Yes  1  10
> mean(tree.pred!=response.test)
[1] 0.05
> #Check with 10 trees
> set.seed(123)
> tree.zoo=randomForest(response~.,zoo.train, ntree=8,mtry=7)
> tree.pred=predict(tree.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  27   1
      Yes  2  10
> mean(tree.pred!=response.test)
[1] 0.075
```

Test Error rates:

parameters	Test Error Rate
ntree: 30, mtry:7	0.025
ntree: 10, mtry:7	0.05
ntree: 8, mtry:7	0.075

7. RANDOM FOREST:

In this technique each classifier is a decision tree classifier. This decision tree is generated only when at each node the attributes are randomly selected to determine the split. And as the part of the classification process each tree will vote and class which is the most popular will be returned.

Library used: library(tree), library(randomForest)

Random Forest Implementation:

```
> #mtry=4
> set.seed(123)
> tree.zoo=randomForest(response~.,zoo.train, ntree=50, mtry=4)
> tree.pred=predict(tree.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  29   1
      Yes  0  10
> mean(tree.pred!=response.test)
[1] 0.025
>
> #mtry=5
> set.seed(123)
> tree.zoo=randomForest(response~.,zoo.train, ntree=50, mtry=5)
> tree.pred=predict(tree.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  28   1
      Yes  1  10
> mean(tree.pred!=response.test)
[1] 0.05
>
> #mtry=3
> set.seed(123)
> tree.zoo=randomForest(response~.,zoo.train, ntree=50, mtry=3)
> tree.pred=predict(tree.zoo,zoo.test,type="class")
> table(tree.pred,response.test)
         response.test
tree.pred No Yes
      No  29   1
      Yes  0  10
> mean(tree.pred!=response.test)
[1] 0.025
```

Test Error rates:

parameters	Test Error Rate
ntree: 50, mtry:4	0.025
ntree: 50, mtry:5	0.05
ntree: 50, mtry:3	0.025

8. BOOSTING:

Boosting is a technique in which once we get the classifier model, the weights are updated to allow the subsequent classifier model which has more priority for training tuples that were misclassified by initial classifier model.

Library used: library(tree), library(gbm)

Boosting Implementation:

```
> set.seed(123)
> train=sample(1:nrow(zoo),60)
> zoo.train=zoo[train,]
> zoo.test=zoo[-train,]
> class.label.test=class.label[-train]
>
> #Check with 40 trees, n.trees =no. of trees
> set.seed(123)
> tree.zoo=gbm(class.label~., zoo.train, distribution="bernoulli",n.trees=40)
> tree.pred.prob=predict(tree.zoo, zoo.test, n.trees=40, type="response")
> tree.pred=ifelse(tree.pred.prob>0.5, "Yes", "No")
> table(class.label.test, tree.pred)
                tree.pred
class.label.test No Yes
             No  29   0
             Yes  3   8
> mean(tree.pred!=class.label.test)
[1] 0.075
>
> #Check with 15 trees
> set.seed(123)
> tree.zoo=gbm(class.label~., zoo.train, distribution="bernoulli",n.trees=15)
> tree.pred.prob=predict(tree.zoo, zoo.test, n.trees=15, type="response")
> tree.pred=ifelse(tree.pred.prob>0.5, "Yes", "No")
> table(class.label.test, tree.pred)
                tree.pred
class.label.test No Yes
             No  29   0
             Yes  3   8
> mean(tree.pred!=class.label.test)
[1] 0.075
>
> #Check with 10 trees
> set.seed(123)
> tree.zoo=gbm(class.label~., zoo.train, distribution="bernoulli",n.trees=10)
> tree.pred.prob=predict(tree.zoo, zoo.test, n.trees=10, type="response")
> tree.pred=ifelse(tree.pred.prob>0.5, "Yes", "No")
> table(class.label.test, tree.pred)
                tree.pred
class.label.test No Yes
             No  29   0
             Yes  7   4
> mean(tree.pred!=class.label.test)
[1] 0.175
```

Test Error rates:

parameters	Test Error Rate
ntree: 40	0.075
ntree: 15	0.075
ntree: 10	0.175

9. NAÏVE BAYES CLASSIFIER

The e1071 library contains implementations for different classification methods including Support Vector Machine and Naive Bayes classification.

Library used: e1071

Naïve Bayes Implementation:

```
> #Fitting the Naive Bayes model
> Naive_Bayes_Model=naiveBayes(response~., zoo)
>
> #Understanding the model summary
> Naive_Bayes_Model

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
  No  Yes
0.66 0.34

Conditional probabilities:
     hair
Y          [,1]      [,2]
  No  0.5909091 0.4954337
  Yes 0.1176471 0.3270350

     feathers
Y          [,1]      [,2]
  No  0.3030303 0.4630899
  Yes 0.0000000 0.0000000

     eggs
Y          [,1]      [,2]
  No  0.3787879 0.4888024
  Yes 0.9705882 0.1714986

     milk
Y          [,1]      [,2]
  No  0.6212121 0.4888024
  Yes 0.0000000 0.0000000

     airborne
Y          [,1]      [,2]
  No  0.2727273 0.4487746
  Yes 0.1764706 0.3869530

     aquatic
Y          [,1]      [,2]
  No  0.1969697 0.4007569
  Yes 0.6470588 0.4850713

     predator
Y          [,1]      [,2]
  No  0.5303030 0.5029053
  Yes 0.5882353 0.4995542

     toothed
Y          [,1]      [,2]
  No  0.6666667 0.4750169
  Yes 0.4705882 0.5066404

     backbone
Y          [,1]      [,2]
  No  1.0000000 0.0000000
  Yes 0.4705882 0.5066404

     breathes
Y          [,1]      [,2]
  No  0.9848485 0.1230915
  Yes 0.4117647 0.4995542

     venomous
Y           [,1]      [,2]
  No  0.03030303 0.1727334
  Yes 0.17647059 0.3869530
```

```
    fins
Y          [,1]      [,2]
  No  0.06060606 0.2404347
  Yes 0.38235294 0.4932702

    legs
Y        [,1]     [,2]
  No  2.818182 1.311594
  Yes 2.852941 3.016443

    tail
Y         [,1]      [,2]
  No  0.9090909 0.2896827
  Yes 0.4411765 0.5039947

    domestic
Y          [,1]      [,2]
  No  0.16666667 0.3755338
  Yes 0.05882353 0.2388326

    size
Y         [,1]      [,2]
  No  0.5909091 0.4954337
  Yes 0.1470588 0.3594906

>
> #Predicting dataset
> NB_Predictions=predict(Naive_Bayes_Model,zoo)
>
> #Confusion matrix for accuracy
> table(NB_Predictions,response)
              response
NB_Predictions No Yes
           No  61   0
           Yes  5  34
> mean(NB_Predictions!=response)
[1] 0.05
>
> #Train and test set
> set.seed(123)
> train=sample(1:nrow(zoo),70)
> trainSet=zoo[train,]
> testSet=zoo[-train,]
> test.label=response[-train]
> NB_2=naiveBayes(response~.,trainSet)
> NB_Predictions_2=predict(NB_2,testSet)
> table(NB_Predictions_2,test.label)
                test.label
NB_Predictions_2 No Yes
             No  20   0
             Yes  2   8
> mean(NB_Predictions_2!=test.label)
[1] 0.06666667
```

Test Error rates:

parameters	Test Error Rate
Before Sampling	0.05
After Sampling	0.06666667

10. SUPPORT VECTOR MACHINE USING LINEAR KERNEL WITH DIFFERENT COSTS

The main objective of the linear Support vector machine is to maximize the margin to feasible extent. The Decision boundary will only depend on the Support vectors. It will not change only when the dataset has same support vectors. . Initially we find the optimal cost for the zoo dataset then we implement this technique using linear kernel and the optimal cost. In the next step we find the train and test error rates.

Library used: library(e1071)

SVM Implementation for Linear Kernel:

```
> #Fitting the model
> svmfit=svm(response~.,data=zoo.train,kernel="linear",cost=0.01)
> summary(svmfit)

Call:
svm(formula = response ~ ., data = zoo.train, kernel = "linear", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  34

 ( 16 18 )


Number of Classes:  2

Levels:
 No Yes



>
> #Training error rate
> svm1.pred=predict(svmfit,newdata=zoo.train)
> table(svm1.pred,response.train)
         response.train
svm1.pred No Yes
      No  41   2
      Yes  0  17
> mean(svm1.pred!=response.train)
[1] 0.03333333
>
> #Testing error rate
> svm1.pred=predict(svmfit,newdata=zoo.test)
> table(svm1.pred,response.test)
         response.test
svm1.pred No Yes
      No  25   2
      Yes  0  13
> mean(svm1.pred!=response.test)
[1] 0.05
>
```

```
> set.seed(123)
> tune.out=tune(svm, response~., data=zoo, kernel="linear",ranges=list(cost=c(0.01,0.1,1,10,100)))
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.03

- Detailed performance results:
   cost error dispersion
1 1e-02  0.03 0.04830459
2 1e-01  0.03 0.04830459
3 1e+00  0.03 0.04830459
4 1e+01  0.03 0.04830459
5 1e+02  0.03 0.04830459

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

Call:
best.tune(method = svm, train.x = response ~ ., data = zoo, ranges = list(cost = c(0.01, 0.1, 1, 10, 100)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  53

 ( 27 26 )


Number of Classes:  2

Levels:
 No Yes



>
> #To find the training error
> pred=predict(tune.out$best.model, newdata=zoo.train)
> table(response.train, pred)
              pred
response.train No Yes
           No  41   0
           Yes  1  18
> mean(pred!=response.train)
[1] 0.01666667
>
>
> #To find the testing error
> pred=predict(tune.out$best.model, newdata=zoo.test)
> table(response.test, pred)
             pred
response.test No Yes
          No  24   1
          Yes  0  15
> mean(pred!=response.test)
[1] 0.025
```

Test Error rates:

	Train Error Rate	Test Error Rate
SVM - Linear Kernel **(cost=c(0.01,0.1,1,10,100)** **Best cost:0.01** **Train set – 60%** **Test set – 40%**	0.01666	0.025

11. SUPPORT VECTOR MACHINE USING RADIAL KERNEL WITH DIFFERENT COSTS AND GAMMAS

Similarly, if the decision boundary is not linear this scenario comes to existence. The main logic here is to transform data into higher dimensional space. Initially we find the optimal cost for the zoo dataset then we implement this technique using radial kernel, the optimal cost and different gamma values. In the next step we find the train and test error rates.

Library used: library(e1071)

SVM Implementation for Radial Kernel:

```
> set.seed(123)
>
> svmfit=svm(response~.,data=zoo.train,kernel="radial",gamma=1,cost=0.01)
> summary(svmfit)

Call:
svm(formula = response ~ ., data = zoo.train, kernel = "radial", gamma = 1, cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01

Number of Support Vectors:  47

 ( 26 21 )


Number of Classes:  2

Levels:
 No Yes



>
> svm1.pred=predict(svmfit,newdata=zoo.train)
> table(svm1.pred,response.train)
         response.train
svm1.pred No Yes
      No  39  21
      Yes  0   0
> mean(svm1.pred!=response.train)
[1] 0.35
>
> svm2.pred = predict(svmfit,newdata=zoo.test)
> table(svm2.pred,response.test)
         response.test
svm2.pred No Yes
      No  27  13
      Yes  0   0
> mean(svm2.pred!=response.test)
[1] 0.325
>
> ##different cost and gammas
> tune.out=tune(svm, response~., data=zoo.train, kernel="radial",ranges=list(cost=c(0.001,0.01,0.1,1,10), gamma=c(1,2,3,4,5)))
```

```
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1     1

- best performance: 0.15

- Detailed performance results:
    cost gamma     error dispersion
1  1e-03     1 0.3500000  0.2415229
2  1e-02     1 0.3500000  0.2415229
3  1e-01     1 0.3500000  0.2415229
4  1e+00     1 0.1500000  0.1657382
5  1e+01     1 0.1500000  0.1657382
6  1e-03     2 0.3500000  0.2415229
7  1e-02     2 0.3500000  0.2415229
8  1e-01     2 0.3500000  0.2415229
9  1e+00     2 0.1500000  0.1657382
10 1e+01     2 0.1500000  0.1657382
11 1e-03     3 0.3500000  0.2415229
12 1e-02     3 0.3500000  0.2415229
13 1e-01     3 0.3500000  0.2415229
14 1e+00     3 0.1500000  0.1657382
15 1e+01     3 0.1500000  0.1657382
16 1e-03     4 0.3500000  0.2415229
17 1e-02     4 0.3500000  0.2415229
18 1e-01     4 0.3500000  0.2415229
19 1e+00     4 0.1500000  0.1657382
20 1e+01     4 0.1500000  0.1657382
21 1e-03     5 0.3500000  0.2415229
22 1e-02     5 0.3500000  0.2415229
23 1e-01     5 0.3500000  0.2415229
24 1e+00     5 0.1833333  0.1657382
25 1e+01     5 0.1500000  0.1657382

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

Call:
best.tune(method = svm, train.x = response ~ ., data = zoo.train, ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10), gamma = c(1, 2, 3, 4, 5)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  47

 ( 26 21 )


Number of Classes:  2

Levels:
 No Yes

> #Test error
> pred=predict(tune.out$best.model, newdata=zoo.test)
> table(response.test, pred)
             pred
response.test No Yes
          No  27   0
          Yes  5   8
> mean(pred!=response.test)
[1] 0.125
>
```

Test Error rates:

	Test Error Rate
SVM - Radial Kernel **(cost=c(0.001,0.01,0.1,1,10)** **(Gamma= c(1, 2, 3,4,5))** **Best cost: 1** **Best gamma=1** **Train set – 60%** **Test set – 40%**	0.125

12. SUPPORT VECTOR MACHINE USING POLYNOMIAL KERNEL WITH DIFFERENT COSTS AND GAMMAS

Similarly, if the decision boundary is not linear this scenario comes to existence. The main logic here is to transform data into higher dimensional space. Initially we find the optimal cost for the zoo dataset thenwe implement this technique using polynomial kernel, the optimal cost and different gamma values. In the next step we find the train and test error rates.

Library used: library(e1071)

SVM Implementation for Polynomial Kernel:

```
[1] 0.025
> set.seed(123)
>
> svmfit=svm(response~.,data=zoo.train,kernel="polynomial",degree=3,cost=0.01)
> summary(svmfit)

Call:
svm(formula = response ~ ., data = zoo.train, kernel = "polynomial", degree = 3, cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.01
     degree:  3
     coef.0:  0

Number of Support Vectors:  46

 ( 25 21 )


Number of Classes:  2

Levels:
 No Yes



>
> svm1.pred=predict(svmfit,newdata=zoo.train)
> table(svm1.pred,response.train)
         response.train
svm1.pred No Yes
      No  39  21
      Yes  0   0
> mean(svm1.pred!=response.train)
[1] 0.35
>
> svm2.pred = predict(svmfit,newdata=zoo.test)
> table(svm2.pred,response.test)
         response.test
svm2.pred No Yes
      No  27  13
      Yes  0   0
> mean(svm2.pred!=response.test)
[1] 0.325
>
> tune.out=tune(svm, response~., data=zoo.train, kernel="polynomial",degree=3,ranges=list(cost=c(0.001,0.01,0.1,1,10), gamma=c(0.2,0.5,1,2,3)))
```

```
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
 0.01   0.2

- best performance: 0.05

- Detailed performance results:
    cost gamma error dispersion
1  1e-03   0.2  0.35  0.2415229
2  1e-02   0.2  0.05  0.1124914
3  1e-01   0.2  0.05  0.1124914
4  1e+00   0.2  0.05  0.1124914
5  1e+01   0.2  0.05  0.1124914
6  1e-03   0.5  0.05  0.1124914
7  1e-02   0.5  0.05  0.1124914
8  1e-01   0.5  0.05  0.1124914
9  1e+00   0.5  0.05  0.1124914
10 1e+01   0.5  0.05  0.1124914
11 1e-03   1.0  0.05  0.1124914
12 1e-02   1.0  0.05  0.1124914
13 1e-01   1.0  0.05  0.1124914
14 1e+00   1.0  0.05  0.1124914
15 1e+01   1.0  0.05  0.1124914
16 1e-03   2.0  0.05  0.1124914
17 1e-02   2.0  0.05  0.1124914
18 1e-01   2.0  0.05  0.1124914
19 1e+00   2.0  0.05  0.1124914
20 1e+01   2.0  0.05  0.1124914
21 1e-03   3.0  0.05  0.1124914
22 1e-02   3.0  0.05  0.1124914
23 1e-01   3.0  0.05  0.1124914
24 1e+00   3.0  0.05  0.1124914
25 1e+01   3.0  0.05  0.1124914

>
> #Train error
> pred=predict(tune.out$best.model, newdata=zoo.train)
> table(response.train, pred)
              pred
response.train No Yes
           No  39   0
           Yes  2  19
> mean(pred!=response.train)
[1] 0.03333333
>
> #Test error
> pred=predict(tune.out$best.model, newdata=zoo.test)
> table(response.test, pred)
             pred
response.test No Yes
          No  27   0
          Yes  1  12
> mean(pred!=response.test)
[1] 0.025
```

Test Error rates:

	Test Error Rate
SVM - Polynomial Kernel **cost=c(0.001,0.01,0.1,1,10),** **degree=2** **gamma=c(0.2,0.5,1,2,3)** **Best gamma=0.2** **Best Cost Value: 1** **Train set – 60%** **Test set – 40%**	0.025

13. COMPARISION OF MULTIPLE CLASSIFICATION TECHNIQUES

Name of the Classifier	Parameters	Testing error
Hold-out Method **Train set – 60%** **Test set – 40%**	**Tree Size: 4** **Tree Size: 3** **Tree Size: 2**	**0.025** **0.025** **0.15**
Bagging **Train set – 60%** **Test set – 40%**	**ntree: 30, mtry=7** **ntree: 10, mtry=7** **ntree: 8, mtry=7**	**0.025** **0.05** **0.075**
Random Forest **Train set – 60%** **Test set – 40%**	**ntree:50, Mtry: 4** **ntree:50, Mtry: 5** **ntree:50, Mtry: 3**	**0.025** **0.05** **0.025**
Boosting **Train set – 60%** **Test set – 40%**	**n.trees: 40** **n.trees: 15** **n.trees: 10**	**0.075** **0.075** **0.175**
Naïve Bayes **Train set – 70%** **Test set – 30%**	**-**	**Before sampling:** **0.05** **After Sampling:** **0.066667**

SVM - Linear Kernel (cost=c(0.01,0.1,1,10,100) Train set – 60% Test set – 40%	Best cost value: 0.01	0.025
SVM – Radial Linear Kernel Cost=(0.001, 0.01,0.1,1,10), gamma=c(1,2,3,4,5) Best gamma=1 Train set – 60% Test set – 40%	Best cost value: 1	0.125
SVM - Polynomial Kernel cost=c(0.001,0.01,0.1,1,10), degree=2 gamma=c(0.2,0.5,1,2,3) Best gamma=0.2 Train set – 60% Test set – 40%	Best cost value: 1	0.025

14. POTENTIAL PERFORMANCE ISSUES AND POSSIBLE FUTURE STUDY

- For each characteristic, we analyzed how the results vary whenever test mode is changed
- Decision Tree: The model performed better with training set and test set in 60-40%.
    - Bagging had issues with larger tree size and performed better with small tree size of 50 and less
    - Random Forest performed worst with mtry value = 5 and Boosting with tree size =40
- Naïve Bayes performance was weak with 60-40% and improved on 70-30%
- SVM with Linear Kernel have performed best with cost value 0.01.
- SVM with Radial Kernel have performed best with cost value 1 & gamma value 1
- SVM with Polynomial Kernel have performed best with cost value 1 & gamma value 0.2

15. CONCLUSION

This project studied the performance of a variety of classification techniques on a zoo dataset, by varying the training size.

By examining different classification techniques on zoo dataset, we observed that Random Forest and SVM classification methods has less Test Error rate when compared to other classification methods.

16. REFERENCES

[1]. Richard Forsyth -- Donor: Richard S. Forsyth 8 Grosvenor Avenue Mapperley Park Nottingham NG3 5DX 0602-621676

https://data.world/uci/zoo/workspace/project-summary?agentid=uci&datasetid=zoo

[2]. https://www.kaggle.com/uciml/zoo-animal-classification

[3]. https://rdrr.io/cran/VDA/man/zoo.html

[4]. http://tunedit.org/repo/UCI/zoo.arff

[5]. https://sci2s.ugr.es/keel/dataset.php?cod=69

17. APPENDIX OF R CODES:

```
library(tree)

#To load the dataset
# Please use header=F
zoo=read.csv("C:/Users/group/SEM-3/Data Mining/Project/zoo.data", header=F)

#Variable headers
names(zoo)=c("animal", "hair", "feathers", "eggs", "milk", "airborne","aquatic", "predator",
"toothed", "backbone", "breathes", "venomous",
"fins", "legs", "tail", "domestic", "size", "type")

#To attach dataset
attach(zoo)

#To check missing values
zoo=na.omit(zoo)

#Deleting duplicate data
zoo=zoo[-c(26),]
rownames(zoo)=NULL
```

```
#Dropping the column animal name

zoo$animal=NULL


#Discretization of class label

response=ifelse(zoo$type<=3,"No","Yes")

zoo=data.frame(zoo,response)

zoo=zoo[,-17]


#To check row and column dimensions

dim(zoo)


#------------------------Hold-out Method------------------------------------

#To create a decision tree with response as the class label based on all other attributes

tree.zoo=tree(response~.,zoo)


#Summary of the created tree

summary(tree.zoo)


#To display the tree structures and node labels

plot(tree.zoo)

text(tree.zoo,pretty=0)

tree.zoo


#Testing the model using predict function
```

```
tree.pred=predict(tree.zoo,type="class")
tree.pred

#Confusion matrix
table(tree.pred,response)

#Correct prediction rate
mean(tree.pred==response)

#Error prediction rate
mean(tree.pred!=response)

#Random select a sample of 60 observations of the data set as a training set and the rest
#of the data set as a test set
set.seed(123)
train=sample(1:nrow(zoo), 60)
zoo.train=zoo[train,]
zoo.test=zoo[-train,]
response.train=response[train]
response.test=response[-train]
tree.zoo=tree(response~.,zoo.train)

#Train error
tree.pred=predict(tree.zoo,zoo.train,type="class")
```

```
table(tree.pred,response.train)

mean(tree.pred!=response.train)


#Test error

tree.pred=predict(tree.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)


#Cross validation to understand optimal level of tree complexity

set.seed(123)

cv.zoo=cv.tree(tree.zoo,FUN=prune.misclass)

cv.zoo


plot(cv.zoo$size ,cv.zoo$dev ,type="b")



#Check the tree with size 4

set.seed(123)

prune.zoo=prune.misclass(tree.zoo,best=4)

plot(prune.zoo)

text(prune.zoo,pretty=0)

tree.pred=predict(prune.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)
```

```
#Check the tree with size 3

set.seed(123)

prune.zoo=prune.misclass(tree.zoo,best=3)

plot(prune.zoo)

text(prune.zoo,pretty=0)

tree.pred=predict(prune.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)


#Check the tree with size 2

set.seed(123)

prune.zoo=prune.misclass(tree.zoo,best=2)

plot(prune.zoo)

text(prune.zoo,pretty=0)

tree.pred=predict(prune.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)



#------------------------Bagging Method-----------------------------------


##Decision tree using bagging
```

```
#Package includes randomForest() to perform both bagging and random forest

library(randomForest)


#bagging - special case of a random forest with m = p

#ntree indicates the number of trees are generated by bagging

#mtry  indicates the number of variables are used at each split.


#Check with 50 trees

set.seed(123)

tree.zoo=randomForest(response~.,zoo.train, ntree=30,mtry=7)

tree.pred=predict(tree.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)


#Check with 20 trees

set.seed(123)

tree.zoo=randomForest(response~.,zoo.train, ntree=10,mtry=7)

tree.pred=predict(tree.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)



#Check with 10 trees

set.seed(123)
```

```
tree.zoo=randomForest(response~.,zoo.train, ntree=8,mtry=7)

tree.pred=predict(tree.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)


#------------------------RandomForest Method-----------------------------------

#Decision tree using RandomForest

#By default, randomForest() uses about sqrt(p) variables when building a random forest of
classification trees. sqrt(16)=4


#mtry=4

set.seed(123)

tree.zoo=randomForest(response~.,zoo.train, ntree=50, mtry=4)

tree.pred=predict(tree.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)


#mtry=5

set.seed(123)

tree.zoo=randomForest(response~.,zoo.train, ntree=50, mtry=5)

tree.pred=predict(tree.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)
```

```
#mtry=3

set.seed(123)

tree.zoo=randomForest(response~.,zoo.train, ntree=50, mtry=3)

tree.pred=predict(tree.zoo,zoo.test,type="class")

table(tree.pred,response.test)

mean(tree.pred!=response.test)


#-----------------------Boosting Method----------------------------------

##Decision tree using boosting

library(tree)

library(gbm)

zoo=read.csv("/Users/ashwinibhoomi/Desktop/SEM-3/Data Mining/Project/zoo.data",header=F)

names(zoo)=c("animal", "hair", "feathers", "eggs", "milk", "airborne","aquatic", "predator",
"toothed", "backbone", "breathes", "venomous",

"fins", "legs", "tail", "domestic", "size", "type")


attach(zoo)


zoo=na.omit(zoo)


#Deleting duplicate data

zoo=zoo[-c(26),]

rownames(zoo)=NULL
```

```
#Dropping the column animal name
zoo$animal=NULL

#Discretization of class label
class.label=ifelse(zoo$type<=3,"No","Yes")
zoo=data.frame(zoo,class.label)
zoo=zoo[,-17]

#For binary classification, the response variable should be 0 or 1 if using Bernoulli distribution.
zoo$class.label=ifelse(zoo$class.label=="Yes",1,0)
zoo$class.label

set.seed(123)
train=sample(1:nrow(zoo),60)
zoo.train=zoo[train,]
zoo.test=zoo[-train,]
class.label.test=class.label[-train]

#Check with 40 trees, n.trees =no. of trees
set.seed(123)
tree.zoo=gbm(class.label~., zoo.train, distribution="bernoulli",n.trees=40)
tree.pred.prob=predict(tree.zoo, zoo.test, n.trees=40, type="response")
tree.pred=ifelse(tree.pred.prob>0.5, "Yes", "No")
table(class.label.test, tree.pred)
```

```
mean(tree.pred!=class.label.test)


#Check with 15 trees

set.seed(123)

tree.zoo=gbm(class.label~., zoo.train, distribution="bernoulli",n.trees=15)

tree.pred.prob=predict(tree.zoo, zoo.test, n.trees=15, type="response")

tree.pred=ifelse(tree.pred.prob>0.5, "Yes", "No")

table(class.label.test, tree.pred)

mean(tree.pred!=class.label.test)


#Check with 10 trees

set.seed(123)

tree.zoo=gbm(class.label~., zoo.train, distribution="bernoulli",n.trees=10)

tree.pred.prob=predict(tree.zoo, zoo.test, n.trees=10, type="response")

tree.pred=ifelse(tree.pred.prob>0.5, "Yes", "No")

table(class.label.test, tree.pred)

mean(tree.pred!=class.label.test)


#------------------------Naïve Bayes Method-----------------------------------

##Naïve Bayes classifier

#The e1071 library contains implementations for Naive Bayes classification and Support Vector Machine.

library(e1071)
```

```
zoo=read.csv("/Users/ashwinibhoomi/Desktop/SEM-3/Data Mining/Project/zoo.data",header=F)

names(zoo)=c("animal", "hair", "feathers", "eggs", "milk", "airborne","aquatic", "predator",
"toothed", "backbone", "breathes", "venomous",

"fins", "legs", "tail", "domestic", "size", "type")


zoo=zoo[-c(26),]

rownames(zoo)=NULL

zoo$animal=NULL


response=ifelse(zoo$type<=3,"No","Yes")

zoo=data.frame(zoo,response)

zoo=zoo[,-17]


dim(zoo)

attach(zoo)


#Fitting the Naive Bayes model

Naive_Bayes_Model=naiveBayes(response~., zoo)


#Understanding the model summary

Naive_Bayes_Model


#Predicting dataset

NB_Predictions=predict(Naive_Bayes_Model,zoo)
```

#Confusion matrix for accuracy

table(NB_Predictions,response)

mean(NB_Predictions!=response)

#Train and test set

set.seed(123)

train=sample(1:nrow(zoo),70)

trainSet=zoo[train,]

testSet=zoo[-train,]

test.label=response[-train]

NB_2=naiveBayes(response~.,trainSet)

NB_Predictions_2=predict(NB_2,testSet)

table(NB_Predictions_2,test.label)

mean(NB_Predictions_2!=test.label)

#------------------------SVM-Linear Method-----------------------------------

##Support vector machine using liner kernel with different costs

library(e1071)

zoo=read.csv("/Users/ashwinibhoomi/Desktop/SEM-3/Data Mining/Project/zoo.data",header=F)

names(zoo)=c("animal", "hair", "feathers", "eggs", "milk", "airborne","aquatic", "predator", "toothed", "backbone", "breathes", "venomous",

"fins", "legs", "tail", "domestic", "size", "type")

```
zoo=zoo[-c(26),]

rownames(zoo)=NULL

zoo$animal=NULL


response=ifelse(zoo$type<=3,"No","Yes")

zoo=data.frame(zoo,response)

zoo=zoo[,-17]


train=sample(1:nrow(zoo), 60)

zoo.train=zoo[train,]

zoo.test=zoo[-train,]


response.train=response[train]

response.test=response[-train]


#Fitting the model

svmfit=svm(response~.,data=zoo.train,kernel="linear",cost=0.01)

summary(svmfit)


#Training error rate

svm1.pred=predict(svmfit,newdata=zoo.train)

table(svm1.pred,response.train)

mean(svm1.pred!=response.train)
```

```
#Testing error rate

svm1.pred=predict(svmfit,newdata=zoo.test)

table(svm1.pred,response.test)

mean(svm1.pred!=response.test)



#Using tune() for cross validation

set.seed(123)

tune.out=tune(svm, response~., data=zoo, kernel="linear",ranges=list(cost=c(0.01,0.1,1,10,100)))

summary(tune.out)


bestmod=tune.out$best.model

summary(bestmod)


#To find the training error

pred=predict(tune.out$best.model, newdata=zoo.train)

table(response.train, pred)

mean(pred!=response.train)



#To find the testing error

pred=predict(tune.out$best.model, newdata=zoo.test)

table(response.test, pred)
```

```
mean(pred!=response.test)

#------------------------SVM-Radial Method--------------------------------
#Support Vector Machine with radial kernel and default gamma before tune-out
set.seed(123)

svmfit=svm(response~.,data=zoo.train,kernel="radial",gamma=1,cost=0.01)
summary(svmfit)

svm1.pred=predict(svmfit,newdata=zoo.train)
table(svm1.pred,response.train)
mean(svm1.pred!=response.train)

svm2.pred = predict(svmfit,newdata=zoo.test)
table(svm2.pred,response.test)
mean(svm2.pred!=response.test)

##different cost and gammas
tune.out=tune(svm, response~., data=zoo.train,
kernel="radial",ranges=list(cost=c(0.001,0.01,0.1,1,10), gamma=c(1,2,3,4,5)))
summary(tune.out)

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

```
#Test error
pred=predict(tune.out$best.model, newdata=zoo.test)
table(response.test, pred)
mean(pred!=response.test)

#------------------------SVM-Polynomial Method-----------------------------------
#Support Vector Machine with polynomial kernel and default gamma before tune-out
set.seed(123)

svmfit=svm(response~.,data=zoo.train,kernel="polynomial",degree=2,cost=0.01)
summary(svmfit)

svm1.pred=predict(svmfit,newdata=zoo.train)
table(svm1.pred,response.train)
mean(svm1.pred!=response.train)

svm2.pred = predict(svmfit,newdata=zoo.test)
table(svm2.pred,response.test)
mean(svm2.pred!=response.test)

##different cost and gammas
```

```
tune.out=tune(svm, response~., data=zoo.train,
kernel="polynomial",degree=2,ranges=list(cost=c(0.001,0.01,0.1,1,10), gamma=c(0.2,0.5,1,2,3)))

summary(tune.out)



#Test error

pred=predict(tune.out$best.model, newdata=zoo.test)

table(response.test, pred)

mean(pred!=response.test)
```