ECEN 765-Machine Learning with Networks
Fall 2017
Final Project Report

Analysis by Classification of RADAR Signals Returned from Ionosphere

Instructor: Dr. Xiaoning Qian

***Submitted by**:*
Sambandh Bhusan Dhal
UIN: 726006080

Abstract:

This report analyses data from a radar system to identify whether the signal is “good” or “bad”. The dataset has 34 attributes and one response. Firstly, Principal Component Analysis (PCA) is used to find the directions in which maximum scattering occurred. We choose the first five principal components in our case since it explained 70% of the variance in our data (as obtained from the scree plot). Several classification methods are then applied on the dataset to predict the response taking the first 20% as testing dataset and the rest 80% as training dataset. Here, we observe that some of the classifiers like Support Vector Classifier, Linear Discriminant Analysis and Support Vector Machine with polynomial kernel yielded about 20% error on the testing data. That is why, we decided to go for 5-fold cross-validation to generate multiple test and training datasets from the original data and test all methods. The Support Vector Machine method with radial kernel achieved the best mean accuracy of 92%.

Motivation:

In the study of radar signals in ionospheric research, it is necessary to check whether the signals returned from radar are useful for further analysis or not. Based on the available results obtained in this dataset, the main objective is to find out which of these signals can be termed as “good”.

While some signals get lost through the ionosphere, some of them contain vague and irrelevant information in the form of noise. Thus, “good” radar signals are those showing evidence of some type of structure in the ionosphere, “bad” signals are those that do not. For this distinction of the bad signals from the good ones, classification methods can be applied on the radar generated signals.

Thus, the problem then becomes to identify the classification model that most accurately predicts quality of signal based on pulses received from the ionosphere by antennas in the radar system.

The dataset:

The dataset consists of 34 attributes. These describe the pulses received by antennas in the radar system at Goose Bay, Labrador. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number. There are 351 observations of data, with the response being either “g” or “b” standing for “good” or “bad”.

The chosen “Ionosphere” data set has comparatively fewer observations to require high computational time. Additional combinational classification techniques are proposed by Marina Skurichina and Robert P W Duin (2000 and 2002). These techniques provide good insight into different classification methods used for analysis along with their combined effects. The former gives a statistical overview and application algorithm to study the sample size effect on classifier accuracy while latter provides an approach to boosting in LDA and corresponding statistical improvements. The training data for “Ionosphere” data set has significant number of observations as compared to predictors and can be evaluated with or without combinational techniques.

There is abundant literature available for readers for learning exploratory data analysis followed by classification models and their improvements. However, the approach followed by me is in accordance to learning of ECEN 765 course and is pertinent to the chosen data set.

The proposed approach:

The first step in working with the dataset was making sure it was clean and fit to work with. All the observations from the radar signals were not useful. Hence we first made sure there weren't any missing values and deal with any which come up.

The number of attributes were relatively high 34. i.e. ~10% of the number of observations. By conducting Principal Component Analysis (PCA), we represented the data using the principal components that explain most of the variance in the data.

Since the data was from 17 high-frequency antennas belonging to the same system, the antenna from which the data was obtained was inconsequential. Hence, we discarded the predictors in their original form and used Principal Components for all our subsequent analysis.

After we had the appropriate number of attributes we needed, we continued to perform various classification methods on the dataset viz.

- K Nearest Neighbors (KNN)
- Logistic Regression
- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA) and
- Bagging
- Random Forest
- Support Vector Classifier (SVC)
- Support Vector Machine (Polynomial and Radial)

We performed these analyses by splitting the dataset into training and testing data sets. Then we perform this splitting 5 times over. Thus, we find 5 different error rates for each method. Finally, we calculate the mean of this error to arrive at the best method. In other words, we perform 5-fold cross-validation to find the best method.

Note: We are not using decision trees because we are already using bagging and random forest which are aggregates of trees.

Analysis of the datasets:

1. **Libraries used:**

```
library(class)
library(MASS)
library(e1071)
library(tree)
library(randomForest)
```

2. **Code to load the dataset:**

```
df <- read.csv("ionosphere.csv", header = FALSE)#to ignore the column
                                                  titles of the predictors
colnames(df)[35] <- "SignalType" # 35th column stores the signaltype
                                   i.e. g for good and b for bad
```

3. **Code to find missing values if any in our dataset:**

```
MissingVals <- apply(df, 2, is.na)
which(MissingVals==TRUE)


xdf <- df[,-2]
```

Here, we did not find any missing values in the dataset "ionosphere.csv" but the value of the second predictor was zero throughout for all the observations. That is why, we decided to remove the second predictor from our dataset.

4. **Implementing Principal Component Analysis (PCA) on our dataset:**

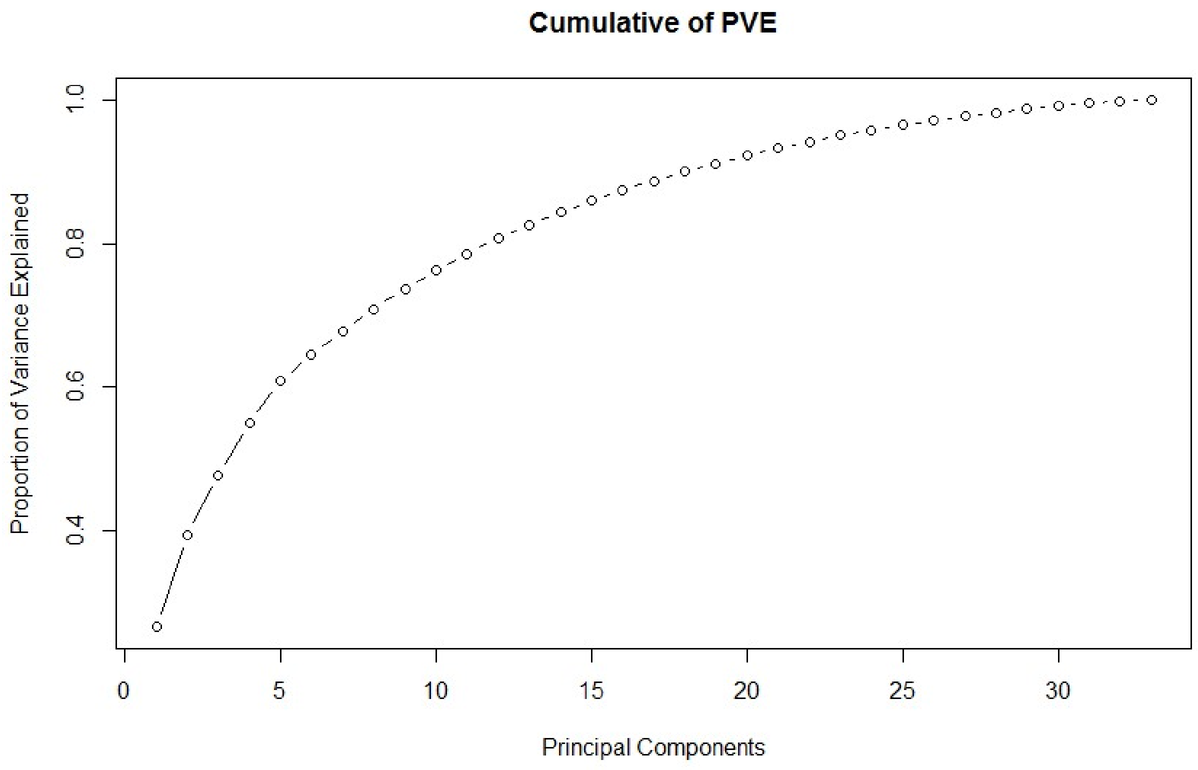
It is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components (or sometimes, principal modes of variation).The PCA algorithm first computes the data means of all the predictors. Secondly, it computes the covariance matrix of the mean substracted data. Thirdly, after conducting Singular Value Decomposition on the covariance matrix, we get the Eigen values and the Eigen vectors.

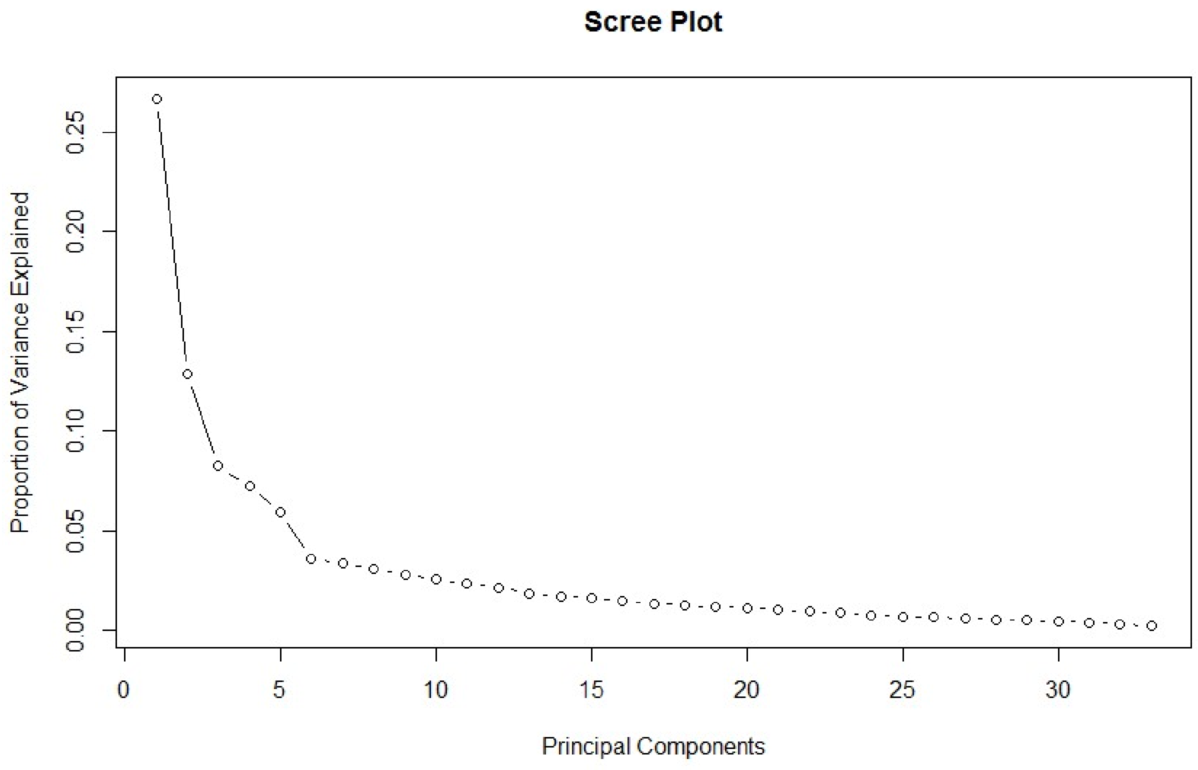
In our case, we have 34 predictors which gives us 34 Eigen values and 34 Eigen vectors but we consider only the first 5 Eigen vectors or principal components since those are the directions where we got the maximum scatter of our data.

```
prOut <- prcomp(xdf, scale = TRUE)# runs PCA on the dataset


# Scree plot

prvar <- prOut$sdev^2
pve <- prvar/sum(prvar)
plot(pve, type = "b", ylab = "Proportion of Variance Explained", xlab =
"Principal Components", main = "Scree Plot")
plot(cumsum(pve), type = "b", ylab = "Proportion of Variance Explained", xlab
= "Principal Components", main = "Cumulative of PVE")
```


Cumulative of PVE
Proportion of Variance Explained
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Principal Components


Scree Plot
Proportion of Variance Explained
0.00
0.05
0.10
0.15
0.20
0.25
0
5
10
15
20
25
30
Principal Components

By studying the scree plot, we can see the proportion of variance obtained. Looking at the cumulative plot, we can see that 5 principal components explain most of the variability (~70%). In the scree plot, the line starts to straighten after the 5th principal component. We can conclude that 6th principal component onwards, it explains a very small proportion and are likely not important. Hence, we decided to use 5 principal components for our analysis.

5. Creating test and training datasets:

In order to test the methods that we will employ, we split the data into training and test datasets. We choose the first 20% of the data as the test set and the rest as the training set.

```
z <- prOut$x[,1:5]

k = 5
testSample <- 1:as.integer(nrow(z)/k)
ztrain <- z[-testSample,]
ztest <- z[testSample,]
signalTrain <- df$SignalType[-testSample]# stores the signaltype of
                                          training dataset
signalTest <- as.character(df$SignalType[testSample])# stores the signaltype

                                                     of testing dataset
```

6. Classification Techniques:

We will attempt to apply several classification techniques to the data in order to predict the response i.e. the nature of signal ("good" or "bad").

(i). KNN (K Nearest Neighbours):

Here, we use KNN as a non-parametric method for classification of data. The input consists of k closest training samples in the sample space. In *k-NN classification*, the output is a class membership. 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 (*k* is a positive integer, typically small).

In this case, we consider the value of k to be 3 and implement the classification algorithm.

```
knnPreds <- as.character(knn(ztrain,ztest,signalTrain,k=3))#returns the
                                          signaltype after knn is applied
predCheck <- cbind(knnPreds,signalTest)#combines the returned signaltype
                                       and signalTest by columns
colnames(predCheck)[1] <- "knn" #renames knnPreds column as knn
knnAccuracy <- mean(predCheck[,"knn"] == predCheck[,"signalTest"])#returns
                                            the accuracy of knn algorithm
```

Test error: 0.1286
Accuracy: 0.8714

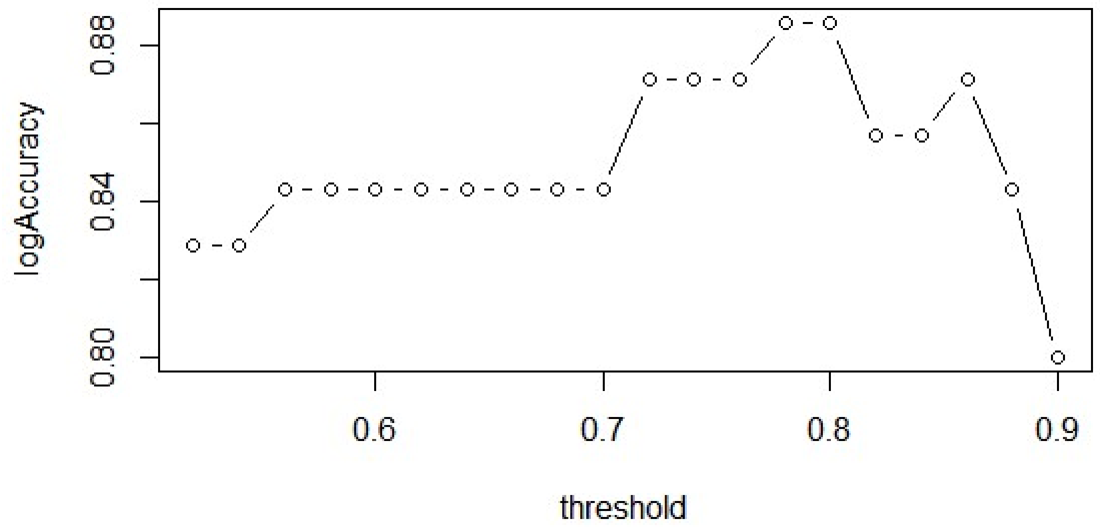
(ii). Logistic Regression:

Here, we use logistic regression because it is a typical regression model where the dependent variable is categorical. In this case, we consider the case of a binary dependent variable where the output can take only two values either 0 or 1, which represent the outcomes that whether the signal is good or bad.

The outcome is based on a logarithmic transformation of regression on the predictors.

```
ztrain <- data.frame(ztrain,signalTrain)#creates dataframe with trainingdata
                                         and corresponding label
ztest <- as.data.frame(ztest)
logAccuracy <- rep(0,20)
threshold <- rep(0,20)
logPredCheck <- as.character(signalTest)

for(i in 1:20)#runs the loop increasing threshold from 0.5 to 0.9 with an
increment of 0.02
{
  threshold[i] <- 0.5 +i*0.02
  logFit <- glm(signalTrain~.,data = ztrain, family = "binomial")
  logProbs <- predict(logFit,ztest,type = "response")
  logPreds <- rep("b",nrow(ztest))
  logPreds[logProbs>threshold[i]] = "g"
  logPredCheck <- cbind(logPredCheck, logPreds)
  colnames(logPredCheck)[1] <- "signalTest"
  colnames(logPredCheck)[length(colnames(logPredCheck))] <-
paste0("Logistic",i)
  logAccuracy[i] <- mean(logPredCheck[,paste0("Logistic",i)] ==
logPredCheck[,"signalTest"])
  }
plot(threshold, logAccuracy, type = 'b')
predCheck <-
cbind(as.character(logPredCheck[,paste0("Logistic",which.max(logAccuracy))]),
predCheck)
colnames(predCheck)[1] <- "Logistic"
logAccuracy <- logAccuracy[which.max(logAccuracy)]
```



Test error: 0.1142
Accuracy: 0.8858

In the above code, we looped through threshold values i.e. the value of probability above which a signal can be classified as “good”. The values of threshold ranged between 0.5 and 0.9 with increments of 0.02 to find threshold that gives the lowest test error. This value was then selected to predict the response using logistic regression.
From the above plot, we can note that when the value of threshold reached 0.8, we got the maximum accuracy for this model of logistic regression.

(iii). Linear Discriminant Analysis (LDA) :

LDA is used to provide maximum separability between classes. The main aim of LDA algorithm is to maximize the mean between the classes and minimize the variance of the data points inside each class. In our case, we have 2 classes. So, the maximum number of Eigen vectors to get separability is 1.

```
ldaFit <- lda(signalTrain~.,data=ztrain)
ldaPreds <- predict(ldaFit,ztest)

predCheck <- cbind(as.character(ldaPreds$class),predCheck)
colnames(predCheck)[1] <- "lda"
ldaAccuracy <- mean(predCheck[,"lda"] == predCheck[,"signalTest"])
```

Test error: 0.2000
Accuracy: 0.8000

(iv). Quadriatic Discriminant Analysis (QDA):

QDA works the same way as that of LDA except for the fact that a quadriatic surface is used to discriminate two or more classes.

```
qdaFit <- qda(signalTrain~.,data=ztrain)
qdaPreds <- predict(qdaFit,ztest)
predCheck <- cbind(as.character(qdaPreds$class),predCheck)
colnames(predCheck)[1] <- "qda"
qdaAccuracy <- mean(predCheck[,"qda"] == predCheck[,"signalTest"])
```

Test error: 0.1142
Accuracy: 0.8858

Therefore, we can observe that the accuracy of the model is significantly increased by 8% as compared to that of LDA.

(v). Bagging:

In this method of classification, bootstrap aggregation is used i.e. it draws a random subset of data by sampling and then draws N' out of N samples with replacement. For each, it generates a predictor on that resampling data. The final classification of the data is decided by majority vote.
The only disadvantage is that it uses more computation per prediction.

Here, we use 100 decision trees for our analysis.

```
p <- ncol(z)
set.seed(1)
bagMod <-
randomForest(signalTrain~.,data=ztrain,mtry=p,ntree=100,importance=TRUE)
bagPreds <- predict(bagMod, ztest)
predCheck <- cbind(as.character(bagPreds),predCheck)
colnames(predCheck)[1] <- "Bagging"
bagAccuracy <- mean(predCheck[,"Bagging"] == predCheck[,"signalTest"])
```

Test error: 0.0857
Accuracy: 0.9143

(vi).Random Forest:

Random Forests is the same as bagging except that the number of predictors can be specified. In bagging all predictors are used. If number of predictors is p, the convention is to use $\sqrt{p}$ predictors are used. In our case, p = 5, Hence we will use 2 predictors.

```
set.seed(1)
rfMod <-
randomForest(signalTrain~.,data=ztrain,mtry=2,ntree=100,importance=TRUE)
rfPreds <- predict(rfMod, ztest)
predCheck <- cbind(as.character(rfPreds),predCheck)
colnames(predCheck)[1] <- "Random Forest"
```

```
rfAccuracy <- mean(predCheck[,"Random Forest"] == predCheck[,"signalTest"])
```

Test error: 0.0571
Accuracy: 0.9429

(vii).Support Vector Classifier:

Support vector machining (SVM) is a discriminative classifier formally defined by a separating hyperplane i.e. using training data it outputs an optimal hyperplane which categorizes new examples.

The support vector classifier uses a linear boundary to separate classes. In this method, we will use multiple values of “cost” or the amount by which the responses from training data are allowed to violate the linear boundary. We will select the best model and apply it to the test data.

```
i <- -3:2
costs <- 10^i
gammas <- seq(0.5,5,by = 0.5)
degrees <- i[5:6]

set.seed(1)
tuneOutSVC = tune(svm,signalTrain~.,data=ztrain, kernel="linear",
ranges=list(cost=costs))
bestModSVC <- tuneOutSVC$best.model
svcPreds <- predict(bestModSVC, ztest)
predCheck <- cbind(as.character(svcPreds),predCheck)
colnames(predCheck)[1] <- "SVC"
svcAccuracy <- mean(predCheck[,"SVC"] == predCheck[,"signalTest"])
```

Test error: 0.2000
Accuracy: 0.8000

(viii). Support Vector Machine

The radial and polynomial support vector classifiers uses a radial and polynomial kernel respectively to separate classes. In this method, we will use multiple values of “cost” or the amount by which the responses from training data are allowed to violate the kernel boundary. We will also check for the best value of gamma for the radial kernel and degree of the polynomial kernel. We will select the best model and apply it to the test data.

```
#Radial SVM
set.seed(1)
tuneOutRadial = tune(svm,signalTrain~.,data=ztrain,
kernel="radial",ranges=list(cost=costs,gamma=gammas))
bestModRadial <- tuneOutRadial$best.model
svmRadialPreds <- predict(bestModRadial, ztest)
predCheck <- cbind(as.character(svmRadialPreds),predCheck)
colnames(predCheck)[1] <- "SVM Radial"
svmRadialAccuracy <- mean(predCheck[,"SVM Radial"] ==
predCheck[,"signalTest"])
```

Test error: 0.0428

Accuracy: 0.9572

```
#Polynomial SVM
set.seed(1)
tuneOutPoly = tune(svm,signalTrain~.,data=ztrain,
kernel="polynomial",ranges=list(cost=costs,degree=degrees))
bestModPoly <- tuneOutPoly$best.model
svmPolyPreds <- predict(bestModPoly, ztest)
predCheck <- cbind(as.character(svmPolyPreds),predCheck)
colnames(predCheck)[1] <- "SVM Poly"
svmPolyAccuracy <- mean(predCheck[,"SVM Poly"] == predCheck[,"signalTest"])
```

Test error: 0.2000
Accuracy: 0.8000

After running the above classification algorithms taking the first 20% of data as testing dataset and the rest as training dataset, we can observe that some of the classification algorithms like Linear Discriminant Analysis, Support Vector Classifier and Support Vector Kernel with polynomial kernel yielded just about 80% accuracy on the testing data.
So, in order to avoid overfitting, we go for cross-validation of the data.

7. Cross-Validation

Cross-validation is a method of estimating error from the same data set by creating multiple test and training data sets. In our case, we will divide the dataset into 5 non-overlapping test data sets and in each case, we use the remaining data as the training data set. This way we will calculate 5 different error rates which we will eventually average to find the method with lowest mean error rate.

```
cvAll <- function(cvCount) {

  # Creating training and test data sets
  k = 5
  testFactor <- as.integer(nrow(z)/k)
  if(cvCount==k){
    testSample <- ((cvCount-1)*testFactor+1):nrow(z)
    } else {
  testSample <- ((cvCount-1)*testFactor+1):((cvCount)*testFactor)
  }



  # The comments below are only placeholders for actual code written above
  # for each method. The code is identical. Not reproduced to save space.

  # KNN
  # Cleaning data for other methods
  # Logistic Regression
  # LDA
  # QDA
  # Random Forest
  # SVM

  accuracyTable <- c(svmPolyAccuracy, svmRadialAccuracy, svcAccuracy,
rfAccuracy, bagAccuracy, qdaAccuracy, ldaAccuracy, logAccuracy, knnAccuracy)
  errorTable <- 1 - accuracyTable
  names(errorTable) <- colnames(predCheck)[-length(colnames(predCheck))]

  return(errorTable)
}

cvCount <- 1:k
cvErrors <- lapply(cvCount, cvAll)
cvErrorTable <- data.frame()

for(j in 1:k){
  cvErrorsUnlisted <- unlist(cvErrors[j])
  cvErrorTable <- rbind(cvErrorTable,cvErrorsUnlisted)
}

names(cvErrorTable) <- names(unlist(cvErrors[1]))
meancverrors <- apply(cvErrorTable,2, mean)
```

Error table:

Runs	SVM poly	SVM Radial	SVC	Random Forest	Bagging	QDA	LDA	Logistic	KNN
1	0.2000	0.0428	0.2000	0.0571	0.0857	0.1142	0.2000	0.1142	0.1286
2	0.2285	0.1571	0.2428	0.1714	0.2000	0.1142	0.2285	0.2142	0.2000
3	0.1885	0.0857	0.1428	0.0857	0.1571	0.1428	0.1428	0.1285	0.1143
4	0.1285	0.0571	0.1142	0.0571	0.0857	0.0428	0.0857	0.1428	0.1000
5	0.0422	0.0281	0.0422	0.0140	0.0281	0.05633	0.0563	0.0845	0.0423
MEAN	0.1570	0.0742	0.1485	0.0771	0.1113	0.0941	0.1427	0.1369	0.1170

Results:

After using various classification techniques like SVM, SVC, random forest, Bagging, LDA, QDA, KNN and log regression and running 5 different iterations to get different test errors, we see that radial SVM gives us the lowest test error at 7.42% (accuracy: 92.58%) followed by Random Forest with an error rate of: 7.71% (accuracy: 92.29%).
It can also be observed that after conducting cross-validation on the dataset, most of the classifiers improved their performance as compared to the previous case.

Conclusion and Discussions:

After looking at the results we can safely state that the random forest and radial SVM models are best suited for our dataset. While the methods discussed in this report are rather primitive, they provide good results with about 92% mean test accuracy for the best model while the maximum test accuracy is over 98%. The selected methods i.e. SVM with radial kernel and random forest can now be employed on any subsequent data along with principal component analysis to predict the nature of the signal received by the RADAR system.

Further improvements can be by made by observing more data and training the models further. The best models from this report can also be stacked to generate more robust models. Sigillito et al suggests using more sophisticated methods such as feedforward neural network in multilayer and single layer networks to achieve around 98% test accuracy.

References:

1. Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, *10*(3), 262-266
2. Kim, H., & Park, H. (2004, April). Data reduction in support vector machines by a kernelized ionic interaction model. In *Proceedings of the 2004 SIAM International Conference on Data Mining* (pp. 507-511). Society for Industrial and Applied Mathematics.
3. Skurichina, M., Kuncheva, L. I., & Duin, R. P. (2002, June). Bagging and boosting for the nearest mean classifier: Effects of sample size on diversity and accuracy. In *International Workshop on Multiple Classifier Systems* (pp. 62-71). Springer Berlin Heidelberg.
4. Skurichina, M., & Duin, R. P. (2000, June). Boosting in linear discriminant analysis. In *International Workshop on Multiple Classifier Systems* (pp. 190-199). Springer Berlin Heidelberg.