

## **MID TERM PROJECT REPORT**

**TITLE:** Analysis by classification of RADAR signals returned from the ionosphere.

**NAME:** Sambandh Bhusan Dhal

**UIN 726006080**

### **Abstract:**

The data from a radar system was analyzed to identify whether the signal is “good” or “bad”. The dataset had 34 attributes and one response. Several classification methods were applied on the dataset to predict the response.

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

While some signals got lost through the ionosphere, some of them contained vague and irrelevant information in the form of noise. Thus, “good” radar signals were those showing evidence of some type of structure in the ionosphere, “bad” signals were those that did not. For this distinction of the bad signals from the good ones, classification methods were 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 this project group is in accordance to learning of ECEN 765 course and 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 16 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

We performed these analyses by splitting the dataset into training and test datasets.

## 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 subtracted data. Thirdly, after conducting Singular Value Decomposition on the covariance matrix, we get the Eigen values and the Eigen vectors.

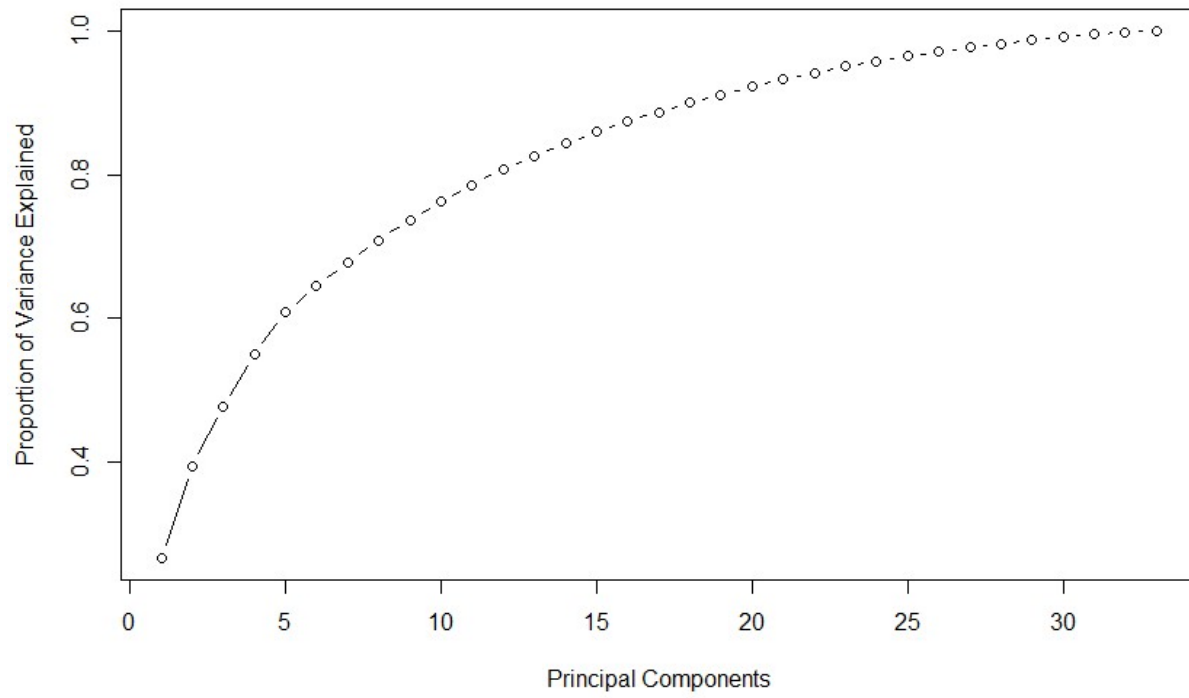
In our case, we have 34 predictors which gives us 34 Eigen values and 34 Eigen vectors but we consider only the first 5 Eigen vectors or principal components since those are the directions where we got the maximum scatter of our data.

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

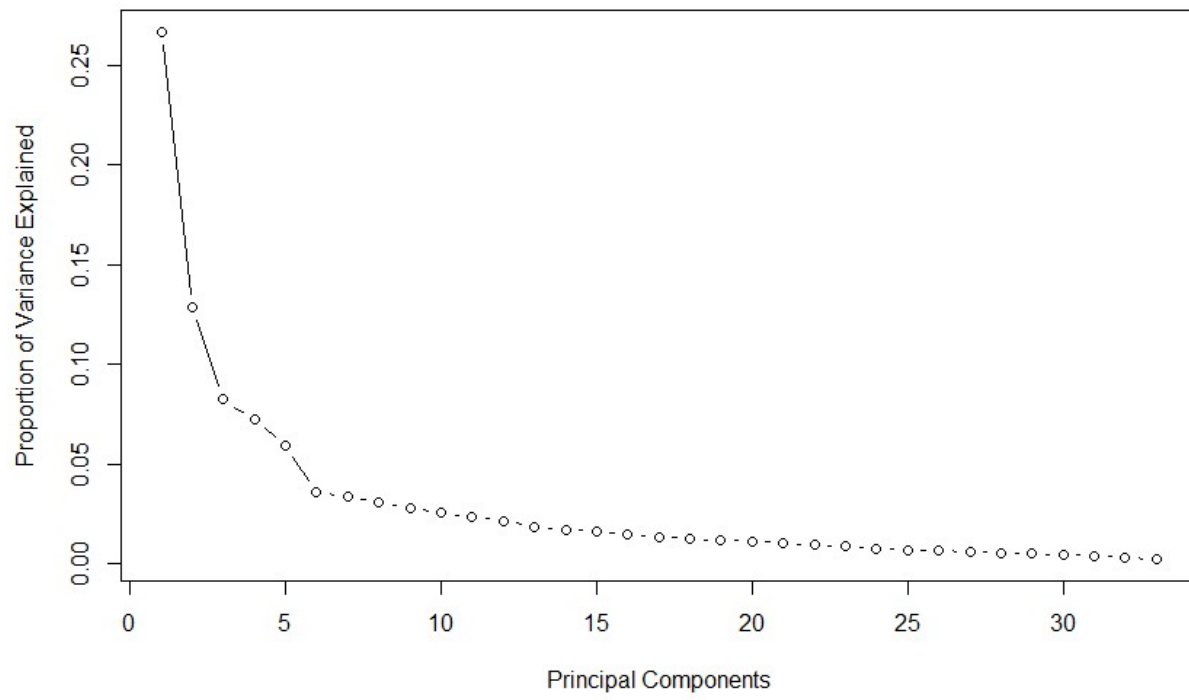
# Scree plot

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**



**Scree Plot**



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 (~60%) In the scree plot, the line starts to straighten after the 5<sup>th</sup> principal component. We can conclude that the 6<sup>th</sup> principal components onwards, explain very 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**

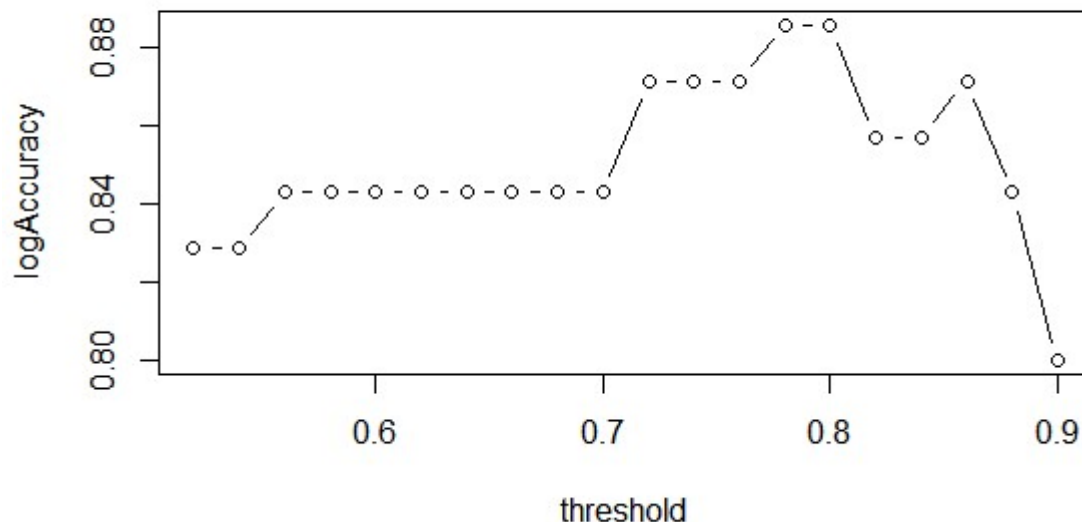
## (ii). Logistic Regression:

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**

So, we can conclude that from the classification techniques used until now, bagging provides the least testing error(8.5%) on our dataset and highest prediction accuracy (91.43%) and linear discriminant analysis provides the highest testing error ( ~ 20%).

#### Ways to address these challenges:

(i). We will perform a 5-step cross-validation on the dataset. 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.



(ii). We will implement 4 more classification algorithms on the dataset like Random Forest, Support Vector Classifier and Support Vector Machines ( both polynomial and radial) to find out which algorithm gives the highest prediction accuracy with lowest error rate.