Table of Contents

Section 3. Supervised Predictive Learning

3.1 Learning outcomes

The following topics were studied and practiced using some practice exercises and applied to the current project

- Prediction vs. interpretation
- Prediction accuracy vs. model interpretability trade-off
- Predictive modeling process
- Over fitting and tuning
    - Training data, Data splitting
- Supervised vs. Unsupervised learning
- Regression vs. Classification
- Regression
    - Ordinary linear regression
    - Non-linear regression
        - Decision trees (Regression Trees)
        - K-Nearest neighbor
    - Bias-Variance tradeoff
- Classification
    - Linear discriminant analysis
        - Using Bayes theorem for classification
    - Logistic regression
    - Quadratic Discriminant Analysis
    - K-Nearest neighbor
    - Classification Trees
- Estimating the Regression Coefficients
- Making predictions
- Resampling methods
    - Validation set

- Leave one out cross-validation
- k-fold CV

3.2 Project 3: A comparison of classification methods on Caravan Insurance Data

Linear Discriminant Analysis, Logistic Regression, Decision Trees, K-Nearest Neighbors

Objective:

The goal of this project is to consider 4 different classification approaches,

1) Linear discriminant analysis
2) Logistic regression
3) Decision Trees and
4) K-Nearest neighbors

apply them to a single dataset, and compare the performances in terms of ***"Percentage of individuals that are correctly predicted to buy insurance"***

Data used:

"Caravan" Insurance data available in ISLR package.

The data contains 5822 real customer records. Each record consists of 86 variables, containing sociodemographic data (variables 1-43) and product ownership (variables 44-86). The sociodemographic data is derived from zip codes. All customers living in areas with the same zip code have the same sociodemographic attributes. Variable 86 (Purchase) indicates whether the customer purchased a caravan insurance policy.

Tools used:

R Studio

Language:

R

Analysis:

We will divide this into four parts obviously, one for each model and then compare the results of each of the parts in the end

STEP 1: LOOKING AT THE DATA

The key to any kind of data analysis is to look at the data and understand what it conveys.

Lets look at the data:

```
library(ISLR)
summary(Caravan)
```

Below is part of the summary of 86 variables

```
##     MOSTYPE          MAANTHUI          MGEMOMV          MGEMLEEF
##  Min.   : 1.00   Min.   : 1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:10.00   1st Qu.: 1.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :30.00   Median : 1.000   Median :3.000   Median :3.000
##  Mean   :24.25   Mean   : 1.111   Mean   :2.679   Mean   :2.991
##  3rd Qu.:35.00   3rd Qu.: 1.000   3rd Qu.:3.000   3rd Qu.:3.000
##  Max.   :41.00   Max.   :10.000   Max.   :5.000   Max.   :6.000
##     MOSHOOFD          MGODRK           MGODPR           MGODOV
##  Min.   : 1.000   Min.   :0.0000   Min.   :0.000   Min.   :0.00
##  1st Qu.: 3.000   1st Qu.:0.0000   1st Qu.:4.000   1st Qu.:0.00
##  Median : 7.000   Median :0.0000   Median :5.000   Median :1.00
##  Mean   : 5.774   Mean   :0.6965   Mean   :4.627   Mean   :1.07
##  3rd Qu.: 8.000   3rd Qu.:1.0000   3rd Qu.:6.000   3rd Qu.:2.00
##  Max.   :10.000   Max.   :9.0000   Max.   :9.000   Max.   :5.00
```

....

....

```
##     AINBOED            ABYSTAND         Purchase
##  Min.   :0.000000   Min.   :0.00000   No :5474
##  1st Qu.:0.000000   1st Qu.:0.00000   Yes: 348
##  Median :0.000000   Median :0.00000
##  Mean   :0.007901   Mean   :0.01426
##  3rd Qu.:0.000000   3rd Qu.:0.00000
##  Max.   :2.000000   Max.   :2.00000
```

```
str(Caravan)
```

```
## 'data.frame':    5822 obs. of  86 variables:
##  $ MOSTYPE : num  33 37 37 9 40 23 39 33 33 11 ...
##  $ MAANTHUI: num  1 1 1 1 1 1 2 1 1 2 ...
##  $ MGEMOMV : num  3 2 2 3 4 2 3 2 2 3 ...
##  $ MGEMLEEF: num  2 2 2 3 2 1 2 3 4 3 ...
##  $ MOSHOOFD: num  8 8 8 3 10 5 9 8 8 3 ...
##  $ MGODRK  : num  0 1 0 2 1 0 2 0 0 3 ...
##  $ MGODPR  : num  5 4 4 3 4 5 2 7 1 5 ...
##  $ MGODOV  : num  1 1 2 2 1 0 0 0 3 0 ...
```

...

....

```
##  $ AFIETS  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ AINBOED : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ABYSTAND: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purchase: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

```r
head(Caravan, 5)
```

```
##   MSKB1 MSKB2 MSKC MSKD MHHUUR MHKOOP MAUT1 MAUT2 MAUT0 MZFONDS MZPART
## 1     1     2    6    1      1      8     8     0     1       8      1
## 2     2     3    5    0      2      7     7     1     2       6      3
## 3     5     0    4    0      7      2     7     0     2       9      0
## 4     2     1    4    0      5      4     9     0     0       7      2
## 5     0     0    0    0      4      5     6     2     1       5      4
```

```r
help(Caravan)
dim(Caravan)
```

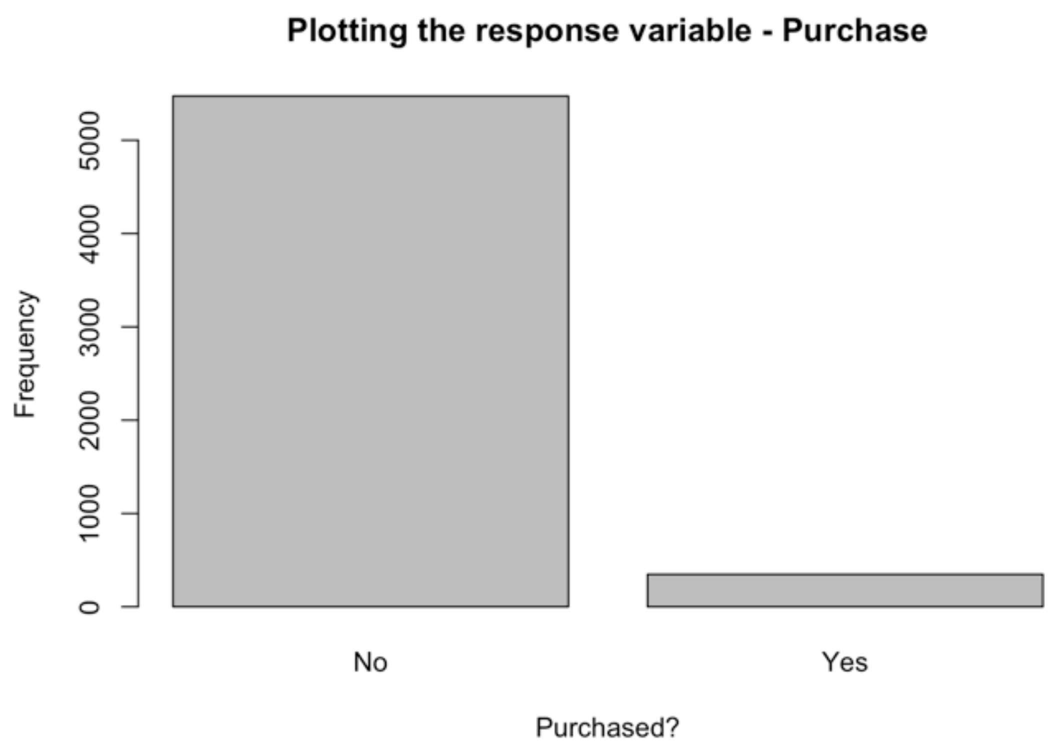
```
## [1] 5822   86
```

```r
summary(Caravan$Purchase)
```

```
##   No  Yes
## 5474  348
```

From this info, we can say 348/5822 = 0.05977 ≈ 6%:

```
plot(Caravan$Purchase, xlab = "Purchased?", ylab = "Frequency", main = "Plottin
g the response variable - Purchase")
```



In this data, only 6% of individuals have purchased Caravan insurance

STEP 2: SAMPLING: SETTING ASIDE 10% OF THE DATA FOR TESTING

```
set.seed(111)
training <- sample(1:nrow(Caravan), 0.9*nrow(Caravan))
train.set <- Caravan[training,]
test.set <- Caravan[-training,]
dim(train.set)
```

```
## [1] 5239   86
```

```
dim(test.set)
```

```
## [1] 583  86
```

Cases per class:

```
table(test.set$Purchase)
```

```
##   No Yes
## 539   44
```

3.2.1 Part 1: LINEAR DISCRIMINANT ANALYSIS

STEP 1: APPLYING LINEAR DISCRIMINANT ANALYSIS (LDA)

#For LDA, we need to supply prior probabilities for the classes involved. Lets assume the prior probability of a client purchasing is 0.5 (we are hence giving them equal prior probabilities)

```
#LDA functions are present in the MASS library in R
library(MASS)
LDA.fit <- lda(Purchase~., data = train.set, prior=priors)
summary(LDA.fit)
```

```
##         Length Class  Mode
## prior     2    -none- numeric
## counts    2    -none- numeric
## means   170    -none- numeric
## scaling  85    -none- numeric
## lev       2    -none- character
## svd       1    -none- numeric
## N         1    -none- numeric
## call      4    -none- call
## terms     3    terms  call
## xlevels   0    -none- list
```

```
LDA.fit
```

```
## Call:
## lda(Purchase ~ ., data = train.set, prior = priors)
##
## Prior probabilities of groups:
##  No Yes
## 0.5 0.5
```

…

….

STEP 2: PREDICTING THE RESPONSE "PURCHASE" USING THE LDA MODEL ON THE TEST DATA

```
LDA.predictions <- predict(LDA.fit, newdata = test.set, type = "class")

summary(LDA.predictions)
```

```
##           Length Class  Mode
## class      583   factor numeric
## posterior 1166   -none- numeric
## x          583   -none- numeric
```

*#class - gives the 'yes', 'No' classes each test observation was predicted to be belonging to #posterior - will give us the posterior probabilities of both classes calculated for each of the test observation (hence 1165*2 = 2330 probability values)*

STEP 3: CHECK THE CORRECTNESS OF OUR MODEL

#The correctness of our model can be understood by looking at how well our predictions match with the actual response values.

#we can do this by creating a contingency table to count the total correct guess and wrong guesses per class

```
LDA.contingency <- table(LDA.predictions$class, test.set$Purchase)
LDA.contingency
```

```
##        No Yes
##    No  409  18
##    Yes 130  26
```

Here, diagonal elements represent the corrects and off-diagonal represent the mistakes Columns represent actual responses recorded, and rows represent the predicted responses.

STEP 4: COMPUTE THE FRACTION OF INDIVIDUALS THAT ARE CORRECTLY PREDICTED TO BUY INSURANCE

#Total Prediction accuracy is (sum of diagonal elements)/total rows which is also same as below

```
mean(test.set$Purchase==LDA.predictions$class)
```

```
## [1] 0.7461407
```

IMPORTANT NOTE: *The company would like to try to sell insurance only to customers who are likely to buy it. So the overall Accuracy rate is not of interest. Instead, the fraction of individuals that are correctly predicted to buy insurance is of interest.*

Therefore, from the contingency table above the true positive prediction accuracy is:

```
#"Yes" class prediction accuracy
lda.Acc <- 26/(130+26)
lda.Acc*100
```

```
## [1] 16.66667
```

So, Among 156 predicted customers, 26, or 16.7 %, actually do purchase insurance.

3.2.2 Part 2: LOGISTIC REGRESSION

STEP 1: FITTING LOGISTIC REGRESSION ON THE TRAINING DATA

we can use the glm() command by specifying family as binomial to predict $Purchase variable as a two-class problem

```
#Logistic regression fit
LogReg.fit <- glm(Purchase~., data = train.set, family = binomial)
```

```
summary(LogReg.fit)
```

```
##
## Call:
## glm(formula = Purchase ~ ., family = binomial, data = train.set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7110  -0.3623  -0.2413  -0.1577   3.2379
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.511e+02  1.124e+04   0.022  0.98217
## MOSTYPE      6.064e-02  4.991e-02   1.215  0.22437
## MAANTHUI    -5.833e-02  1.877e-01  -0.311  0.75592
## MGEMOMV     -5.153e-02  1.506e-01  -0.342  0.73218
## MGEMLEEF     2.188e-01  1.098e-01   1.993  0.04630 *
## MOSHOOFD    -2.328e-01  2.238e-01  -1.040  0.29838
```

STEP 2: PREDICTING THE RESPONSE "PURCHASE" USING THE LOGISTIC REGRESSION MODEL ON THE TEST DATA

```
LogReg.predictions <- predict(LogReg.fit, newdata = test.set, type = "response")
```

The above predictions has posterior predictions for the response variable - 'Purchase'. Deciding which class each test observation will belong to will depend on us. We can keep the importance equal and say all predictions > 0.5 will relate to Purchase = 'Yes'

```
length(LogReg.predictions)
```

```
## [1] 583
```

```
LogReg.output <- rep('No',length(LogReg.predictions))
LogReg.output[LogReg.predictions>0.5] = 'Yes'
```

Now we have got our predictions for Yes and No in LogReg.output variable.

STEP 3: CHECK THE CORRECTNESS OF OUR MODEL

Lets create the contingency table again.

```
LogReg.contingency <- table(LogReg.output, test.set$Purchase)
LogReg.contingency
```

```
## LogReg.output  No Yes
##           No  538  44
##           Yes   1   0
```

STEP 4: COMPUTE THE FRACTION OF INDIVIDUALS THAT ARE CORRECTLY PREDICTED TO BUY INSURANCE

```
#Total Accuracy
mean(test.set$Purchase==LogReg.output)
```

```
## [1] 0.922813
```

```
#"Yes" class prediction accuracy
0/(0+1)
```

So here, We predicted that 1 person will buy, but we were wrong about that.
Earlier, when we gave a probability cut of 0.5, it isn't required that we always give 50%.
Lets re-do the computation by giving LogReg.predictions cut off as 0.25 instead of 0.5 since our true purchase is only 6% anyway

STEP 5: RE-COMPUTATION BY INCREASING PROBABILITY CUTOFF TO IMPROVE THE MODEL

```
LogReg.output[LogReg.predictions>0.25] = 'Yes'
LogReg.contingency <- table(LogReg.output, test.set$Purchase)
LogReg.contingency
```

```
## LogReg.output  No Yes
##           No  532  43
##           Yes   7   1
```

```
#"Yes" class prediction accuracy
log.Acc <- 1/(7+1)
log.Acc*100
```

```
## [1] 12.5
```

So, True positive prediction accuracy truly got better after reducing the cut off

3.2.3 Part 3: DECISION TREES:

STEP 1: FITTING DECISION TREES TO OUR DATA

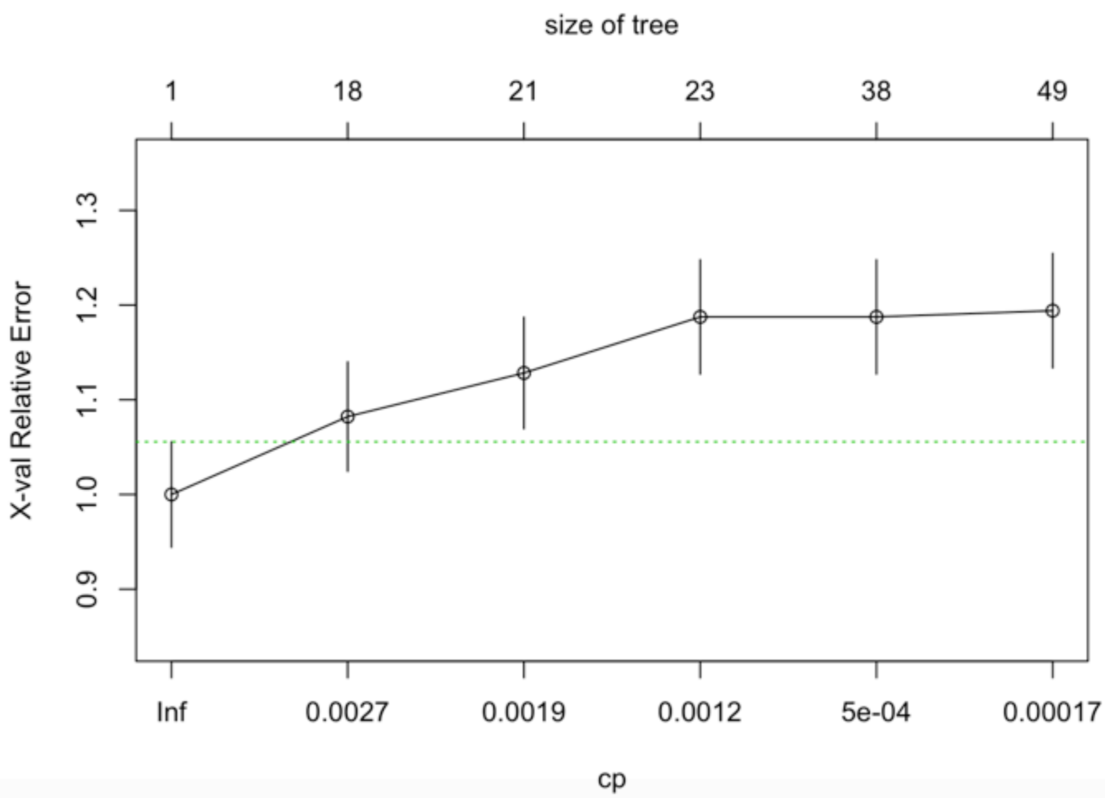
rpart is the library where method called rpart() will be used fit decision trees

```
library(rpart)
tree.fit <- rpart(Purchase~., data = train.set, method = "class", cp =0.0001)
```

STEP 2: PLOT AN INTERPRETABLE CLASSIFICATION DECISION TREE

Before we do anything, we can look at the graph of cross validation error as a function of size of tree using the methods plotcp()

```
plotcp(tree.fit, col = 3)
```



From this plot, we can already get a feel of the size of the tree (number of splits), the minimum cross validation error value and the corresponding complexity parameter value.

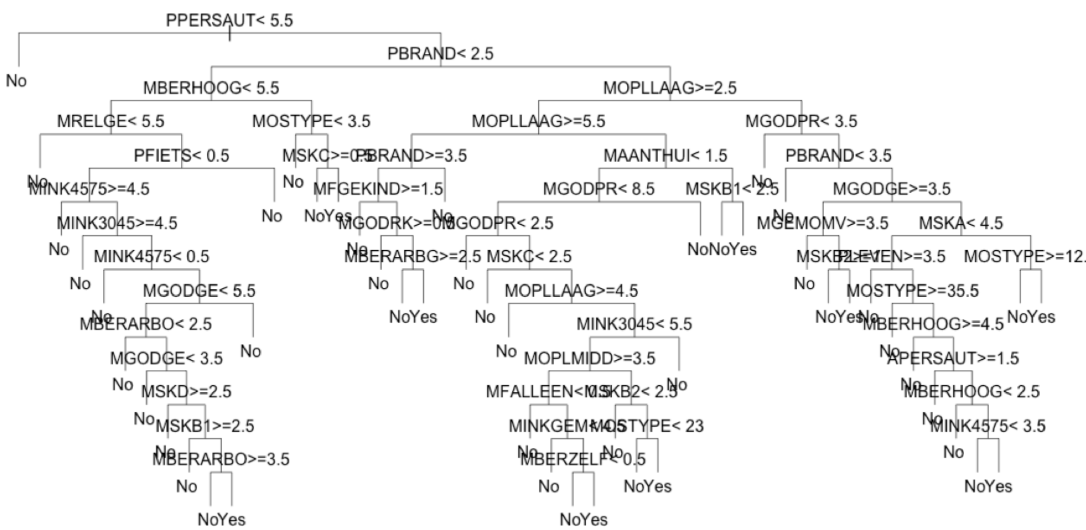
Next we can see the details of this graph through the object returned by rpart using the cptable

```
plotdetails <- tree.fit$cptable
plotdetails
```

```
##                CP nsplit rel error   xerror       xstd
## 1 0.0032894737      0 1.0000000 1.000000 0.05566505
## 2 0.0021929825     17 0.9309211 1.082237 0.05776183
## 3 0.0016447368     20 0.9243421 1.128289 0.05889385
## 4 0.0008223684     22 0.9210526 1.187500 0.06030825
## 5 0.0002990431     37 0.9078947 1.187500 0.06030825
## 6 0.0001000000     48 0.9046053 1.194079 0.06046268
```

Now, lets look at the decision tree:

```
plot(tree.fit, uniform = T, margin = 0.01)
text(tree.fit, pretty = 0)
```



The text at nodes show the splits and the labels at the terminal nodes show the class of the final classification of the terminal node

```
summary(tree.fit)
```

The summary of the tree will also give us detailed splits, variable importance and surrogate splits if any. A glimpse of summary is as follows:

```
## Call:
## rpart(formula = Purchase ~ ., data = train.set, method = "class",
##     cp = 1e-04)
##   n= 5239
```

```
## Variable importance
## PPERSAUT MBERHOOG APERSAUT   PBRAND MOPLLAAG     MSKA     MSKC    MSKB1
##        6        5        5        5        5        5        4        3
##  MOSTYPE   MGODGE   MGODPR MFGEKIND MFALLEEN MOPLMIDD MOSHOOFD MOPLHOOG
##        3        3        3        3        3        3        3        3
```

```
## Node number 1: 5239 observations,    complexity param=0.003289474
##   predicted class=No   expected loss=0.05802634  P(node) =1
##     class counts:  4935   304
##    probabilities: 0.942 0.058
##   left son=2 (3101 obs) right son=3 (2138 obs)
##   Primary splits:
##       PPERSAUT < 5.5  to the left,  improve=17.403970, (0 missing)
##       APERSAUT < 0.5  to the left,  improve=10.576750, (0 missing)
##       PBRAND   < 2.5  to the left,  improve= 7.935174, (0 missing)
##       PPLEZIER < 0.5  to the left,  improve= 6.753779, (0 missing)
##       APLEZIER < 0.5  to the left,  improve= 6.753779, (0 missing)
##   Surrogate splits:
##       APERSAUT < 0.5  to the left,  agree=0.896, adj=0.744, (0 split)
##       PBRAND   < 3.5  to the left,  agree=0.620, adj=0.069, (0 split)
##       PWAPART  < 1.5  to the left,  agree=0.603, adj=0.027, (0 split)
##       PTRACTOR < 1.5  to the left,  agree=0.601, adj=0.023, (0 split)
##       ATRACTOR < 0.5  to the left,  agree=0.601, adj=0.023, (0 split)
```

….

…..

```
## Node number 29926: 11 observations
##   predicted class=No   expected loss=0.09090909  P(node) =0.002099637
##     class counts:    10     1
##    probabilities: 0.909 0.091
##
## Node number 29927: 9 observations
##   predicted class=Yes  expected loss=0.4444444  P(node) =0.001717885
##     class counts:     4     5
##    probabilities: 0.444 0.556
```

Step 3: Predict the response variable and compute true positive accuracy rate

```r
tree.predictions <- predict(tree.fit, newdata = test.set, type = "class")

#creating the contingency matrix
tree.contingency <- table(tree.predictions, test.set$Purchase)
tree.contingency
```

```
## tree.predictions  No Yes
##              No  529  43
##              Yes  10   1
```

```r
#Accuracy
mean(tree.predictions == test.set$Purchase)
```

```
## [1] 0.9090909
```

```
#"Yes" class prediction accuracy
tree.Acc <- 1/(10+1)
tree.Acc*100
```

```
## [1] 9.090909
```

To our disappointment, classification Tree has performed not so well than Logistic regression as the prediction is only 9%

3.2.4 Part 4: K-NEAREST NEIGHBORS

We will now perform KNN using the knn() function from library(class). This function works differently than other models. We usually fit the model first, then use the model to make predictions on the test data. But for knn(), we need single command to form predictions

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it in space, the scale of the variables matters more than for any of the rest of the classifiers (for eg, it will consider a difference of $500 greater than a difference of 25 years, where as in reality we know that 25yrs difference is a lot)

Hence we need to standardize all the data (mean = 0 and standard deviation = 1)

Step 1: Scaling the predictors

```
X=scale(Caravan [,-86])
train.x <- X[training ,]
test.x <- X[-training,]
train.y <- Caravan$Purchase[training]
test.y <- Caravan$Purchase[-training]
```

Step 2: Predicting the response and computing test error using knn() with K=1

```
library(class)
?knn
set.seed(111)
KNN.predictions <- knn(train.x, test.x, train.y, k=1)

knn.contingency <- table(KNN.predictions, test.y)
knn.contingency
```

```
##                test.y
## KNN.predictions  No Yes
##             No  509  41
##             Yes  30   3
```

```
#"Yes" class prediction accuracy
3/(30+3)
```

```
## [1] 0.09090909
```

The accuracy here is 9%

STEP 3: PREDICTING THE RESPONSE AND COMPUTING TEST ERROR USING KNN() WITH K=4

```
KNN.predictions <- knn(train.x, test.x, train.y, k=4)

knn.contingency <- table(KNN.predictions, test.y)
knn.contingency
```

```
##                test.y
## KNN.predictions  No Yes
##             No  536  42
##             Yes   3   2
```

```
#"Yes" class prediction accuracy
knn.Acc <- 2/(3+2)
knn.Acc*100
```

```
## [1] 40
```

This has a whopping 40% of accuracy which is 31% increase from the k=1 prediction

STEP 4: PREDICTING THE RESPONSE AND COMPUTING TEST ERROR USING KNN() WITH K=5

```
KNN.predictions <- knn(train.x, test.x, train.y, k=5)

knn.contingency <- table(KNN.predictions, test.y)
knn.contingency
```

```
##                test.y
## KNN.predictions  No Yes
##             No  537  44
##             Yes   2   0
```

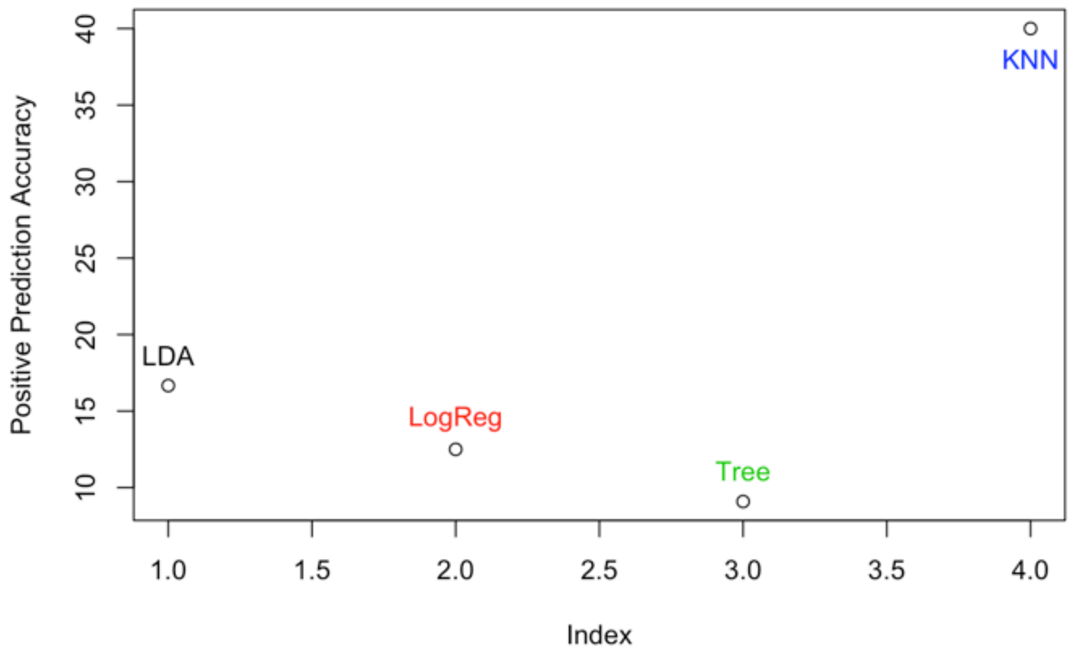
```
#"Yes" class prediction accuracy
0/(2+0)
```

```
## [1] 0
```

K=5 did not give desirable results, hence k=4 seems the apt, perfect value for the best accuracy

FINAL COMPARISON
Now that we have tested all four classifiers on the Caravan Insurance data, we can compare the True Positive Accuracy of these models and decide which one was the best suited model for the data and why.

```
plot(c(lda.Acc, log.Acc, tree.Acc, knn.Acc)*100, ylab = "Positive Prediction Acc
uracy")
text(c(lda.Acc+0.02, log.Acc+0.02, tree.Acc+0.02, knn.Acc-0.02)*100, labels = c(
"LDA", "LogReg", "Tree", "KNN"), col = c(1,2,3,4))
```



CONCLUSION: Clearly KNN outperformed all the remaining models followed by LDA.

- LDA works when data is assumed to follow normal distribution. It therefore did sincerely try to perform better but because the actual model perhaps wasn't linear, LDA could only perform so much assuming a linear model.
- Logistic Regression did not work for the same reasons. It probably would match LDA, if the cutoff threshold is further reduced.
- Classification trees do not perform well when there is little data available (in our case the number of "Yes" were too less – only 6%, to train a solid tree model). And they also usually need a bit of parameter tuning and complexity pruning. So, in my opinion, it could not perform well because of the that.
- KNN appears to have found some real patterns in the data and grouped the test data better than the rest of the models even though the "Yes" observations were sparse.