Lab 2 Block 1

Ahmed Alhasan

12/4/2019

Assignment 2. Analysis of credit scoring

2.2 Deviance vs Gini

"Deviance"

```
## [1] "Training data"
```

```
## $`Confusion Matrix`
##       Predicted
## Actual bad good
##   bad   61   86
##   good  20  333
##
## $`Missclassification Rate`
## [1] 0.212
```

```
## [1] "Testing data"
```

```
## $`Confusion Matrix`
##       Predicted
## Actual bad good
##   bad   28   48
##   good  19  155
##
## $`Missclassification Rate`
## [1] 0.268
```

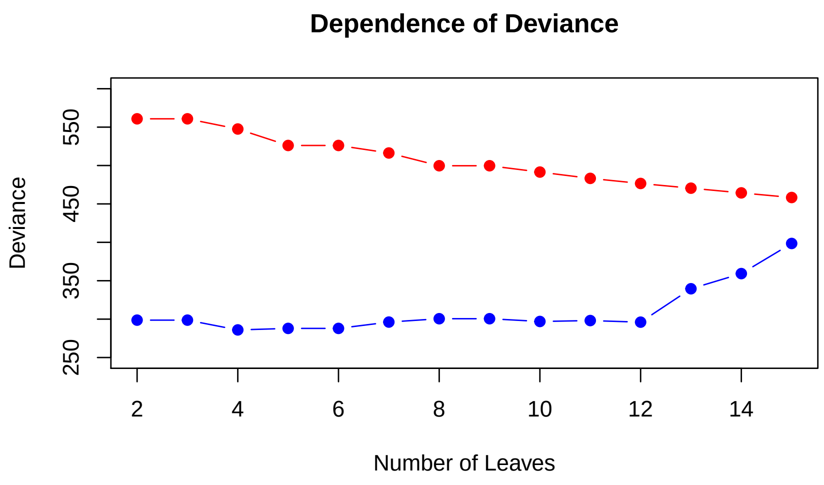
"Gini"

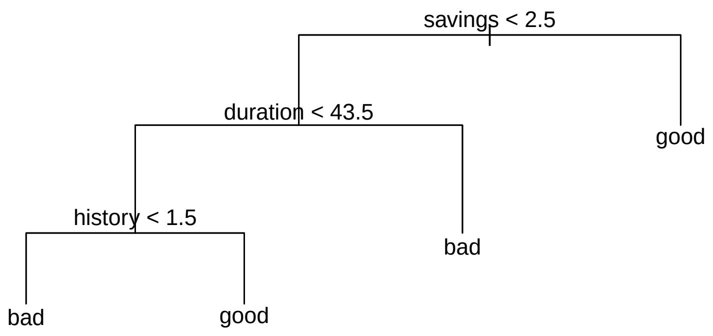
```
## [1] "Training data"
```

```
## $`Confusion Matrix`
##       Predicted
## Actual bad good
##   bad   66   81
##   good  38  315
##
## $`Missclassification Rate`
## [1] 0.238
## [1] "Testing data"

## $`Confusion Matrix`
##       Predicted
## Actual bad good
##   bad   18   58
##   good  35  139
##
## $`Missclassification Rate`
## [1] 0.372
```

- Using Deviance measurement "cross-entropy" give better error rates with less complexity

2.3 Optimal Tree





```
##
```

```
## Classification tree:
## snip.tree(tree = my_tree, nodes = c(5L, 3L, 9L))
## Variables actually used in tree construction:
## [1] "savings"  "duration" "history"
## Number of terminal nodes:  4
## Residual mean deviance:  1.117 = 547.5 / 490
## Misclassification error rate: 0.251 = 124 / 494
```

```
## $`Confusion Matrix`
##       Predeicted
## Actual bad good
##   bad   18   58
##   good   6  168
##
## $`Misclassification Rate`
## [1] 0.256
```

- The optimal tree “which is a pruned deviance tree” is constructed from 4 leaves and 3 nodes using 3 variables “savings”, “duration” & “history” in order of their importance.
- Missclassification rate of the otimal tree using the test data is 0.256 which is a slightly better than the whole tree produeced by deviance measurment.
- The nodes (5, 3 & 9) are the roots of sub-trees from the main tree that has been snipped off.
- The 0.251 error rate in the summary is for the training data.

2.4 Naive Bayes

```
## [1] "Training data"
```

```
## $`Confusion Matrix`
##       Predicted
## Actual bad good
##   bad   95   52
##   good  98  255
##
## $`Missclassification Rate`
## [1] 0.3
```

```
## [1] "Testing data"
```

```
## $`Confusion Matrix`
##       Predicted
## Actual bad good
##   bad   46   30
##   good  49  125
##
## $`Missclassification Rate`
## [1] 0.316
```

- Naive Bayes error rate was higher in both training and testing data than the optimal tree, however rate of False-Positive is less than optimal tree.

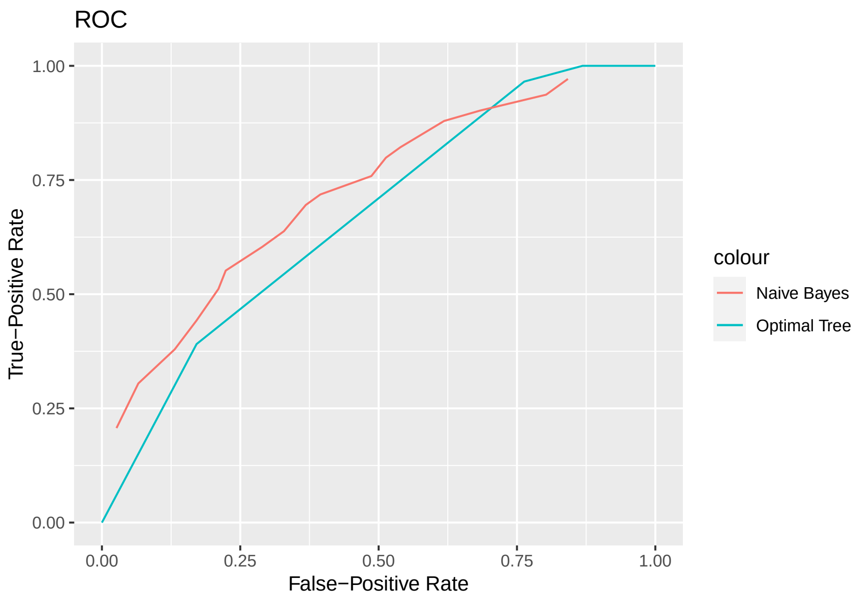
2.5 Using Thresholds

TPR & FPR for Optimal Tree

TPR	1	1	1	1.00	1.00	1.00	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.39	0.39	0	0	0
FPR	1	1	1	0.87	0.87	0.87	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.76	0.17	0.17	0	0	0

TPR & FPR for Naive Bayes

TPR	0.97	0.94	0.93	0.90	0.88	0.82	0.80	0.76	0.74	0.72	0.70	0.64	0.60	0.55	0.51	0.44	0.38	0
FPR	0.84	0.80	0.76	0.68	0.62	0.54	0.51	0.49	0.43	0.39	0.37	0.33	0.29	0.22	0.21	0.17	0.13	0



- From the ROC Naive Bayes performs better due to it's larger AUC, this is because in all thresholds used the percentage of FPR to TPR in Naive Bayes is smaller than its percentage in the optimal tree.

2.6 Applying Loss Penalty

```
## [1] "Training data"
```

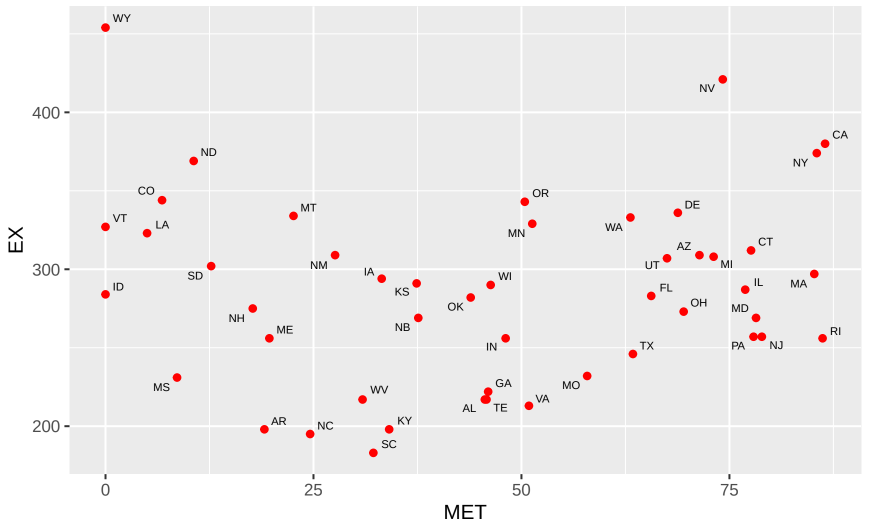
```
## $`Confusion Matrix`
##       Predicted
## Actual Bad Good
##   Bad  137   10
##   Good 263   90
```

```
##
## $`Missclassification Rate`
## [1] 0.546

## [1] "Testing data"

## $`Confusion Matrix`
##        Predicted
## Actual Bad Good
##   Bad   71    5
##   Good 122   52
##
## $`Missclassification Rate`
## [1] 0.508
```

- When applying the loss matrix,the FPR has dramatically decreased, however the overall missclassification rate has deteriorated.

Assignment 3. Uncertainty estimation

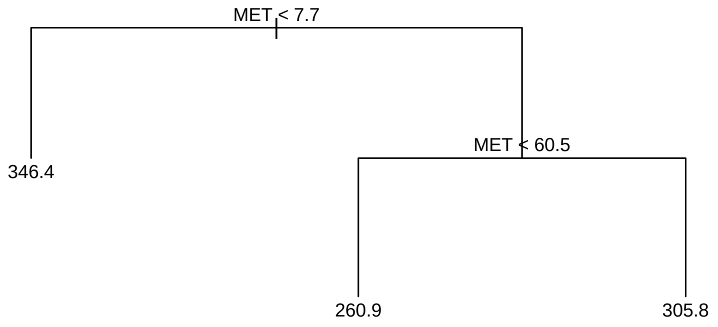
3.1 Appropriate Model

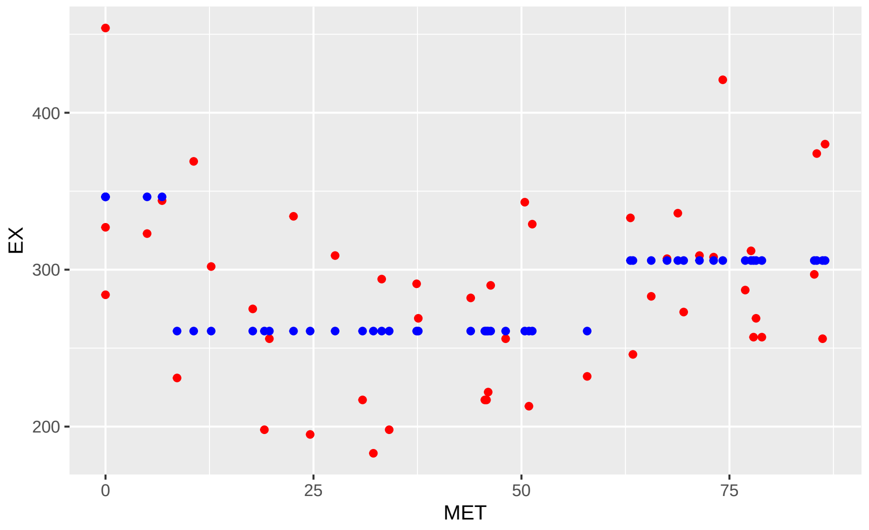


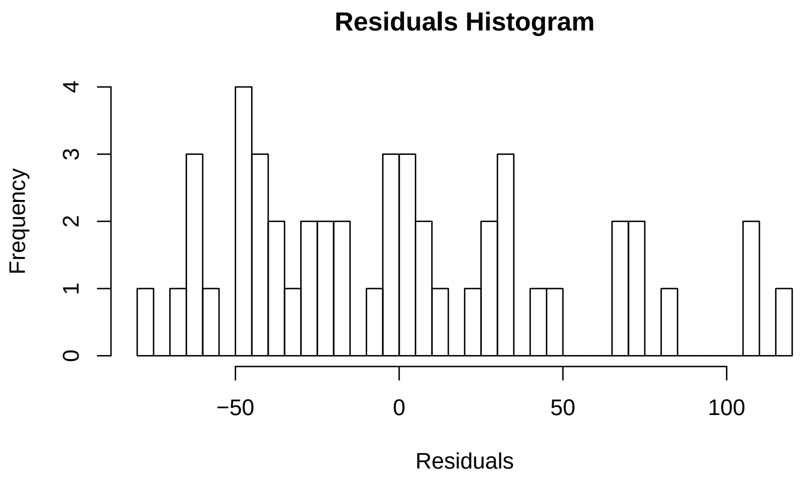
- A 3 degree polynomial can make a good fit for this data.
- A cupic polynomial spline also can fit the data, however because it is a low degree polynomial the spline is not really needed, since there is not much advantage using the spline given that there is not much oscillation (the line go far up or down)

3.2 Tree Model

```
##
## Regression tree:
## snip.tree(tree = tree_model, nodes = 7:6)
## Number of terminal nodes:  3
## Residual mean deviance:  2698 = 121400 / 45
## Distribution of residuals:
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -77.88  -43.88   -4.88    0.00   30.13  115.20
```

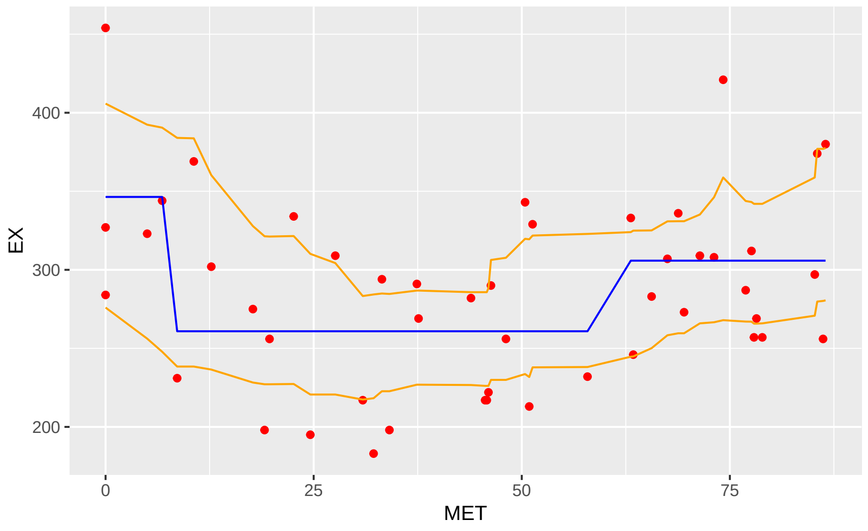






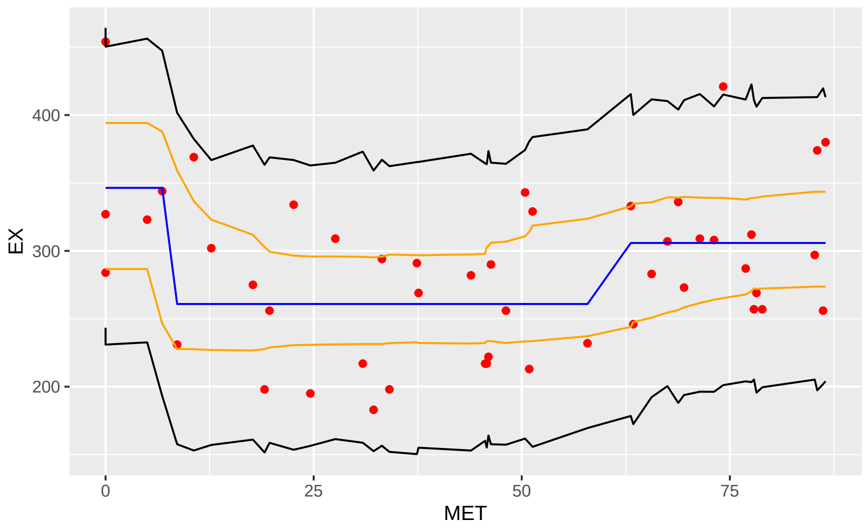
- The residuals does not follow any known distribution (Normal, Binomial, ... etc.), and while the Decision Trees do not require any distributional assumptions they mostly work with highly discrete expectation variables (they do not provide smooth prediction)
- Because its hierarchical greedy approach the tree is inefficient in using the predicting variables and cause over–sensitivity to the training set and to irrelevant variabls, that's why the fit is not very good.

3.3 Nonparametric Bootstrap



- The confidence interval is bumby because of the variance of point estimates as we are getting large and varied residuals every time take a bootstrap sample.
- Because the confidence interval is rather large it confirms the belief that the tree model is a poor fit for this data set and not reliable.

3.4 Parametric Bootstrab



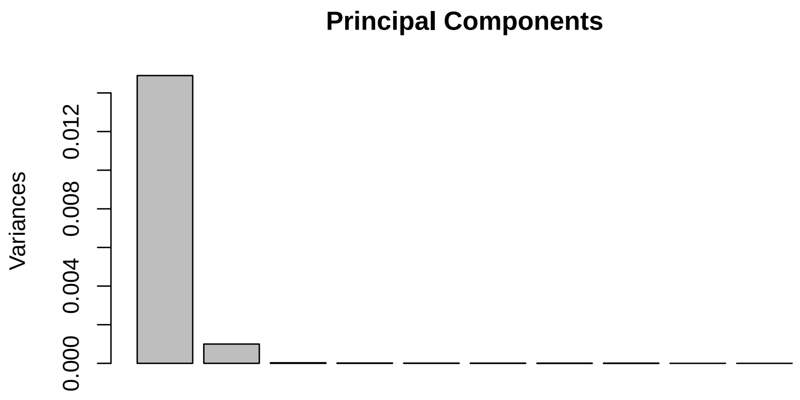
- The confidence interval is smoother and narrower with parametric boostrap which make the tree model a slightly more reliable, however the confidence interval still wide enough allowing wide range of the prediction means to fit in.
- Only 2 points out of 48 are outside the prediction interval which is around 5%, and that's how it should be because we also calculated the error in the prediction interval, meaning 95% the future sampled EX values should be within this interval.

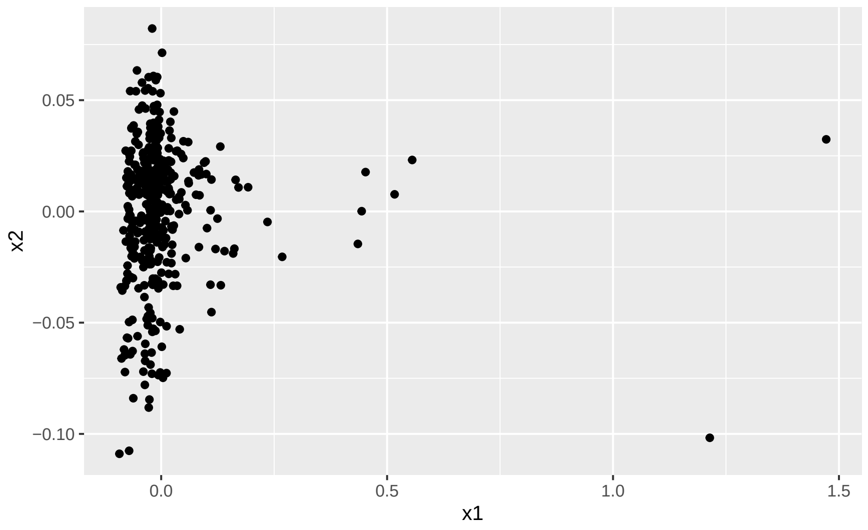
3.5 Optimal Bootstrap

- The assumption of normality in 3.4 does not fit the observed distribution of the residuals, therefore simulating new data from a normally distributed residuals will not provide model that estimates the original data, the narrower and smoother confidence bands only means the model fits the simulated data more.
- Therefore, the non-parametric bootstrap is the more appropriate here because it does not assume anything about the data.

Assignment 4. Principal components

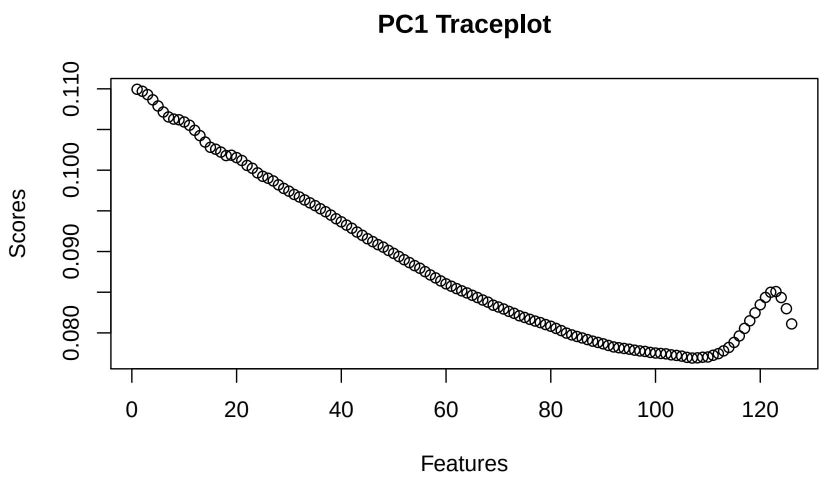
4.1 Standard PCA

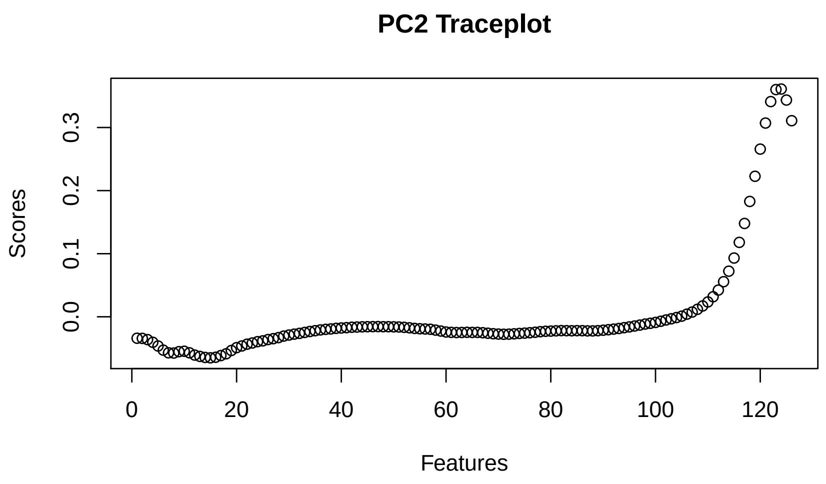




- As can be seen from the plot, the first two components explain more than 99% of the variation, and yes there are couple of oulier diesel fuels.

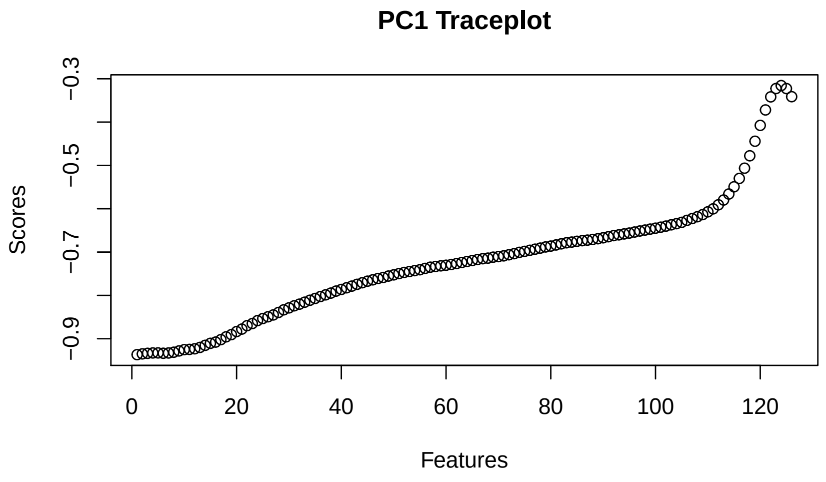
4.2 Score Plots

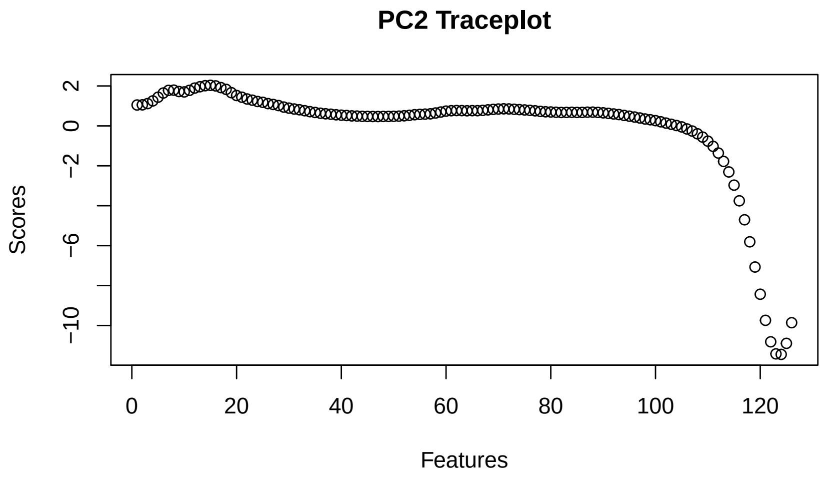


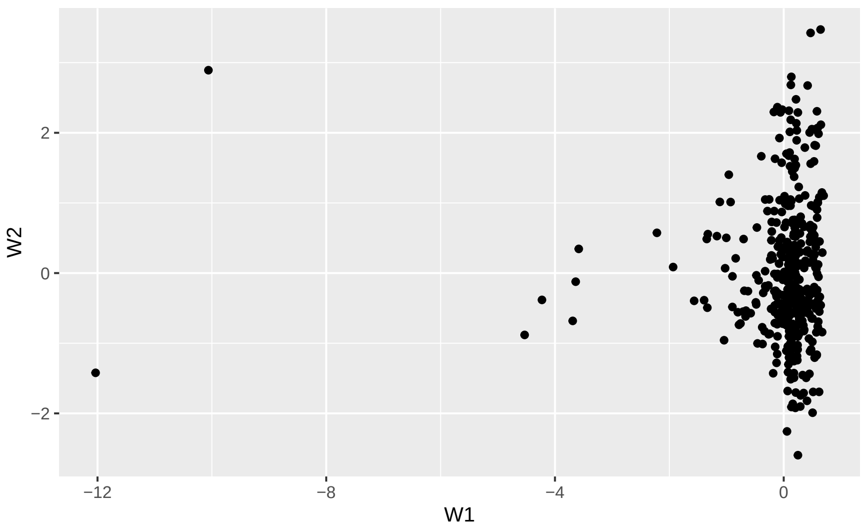


- As can be seen from the plots, PC2 is only explained by few features (features around #110 to #126)

4.3 Independent Component Analysis







- W is the un-mixing matrix that maximizes the non-gaussianity of the components so we can extract the independent components.
- The ICA plot basically takes the standarized PCA factors after scaling and whitening “projecting the data onto its principal component directions” and rotate them to maximmize the non-gaussianity of the two components, thats why we see the ICA plot as a rotated PCA plot.

Appendix

```
RNGversion('3.5.1')
setwd("D:/Machine Learning/Workshop/Machine Learning/Block 1/Lab 2")

library(tree)
library(ggplot2)
library(e1071)
library(SDMTools)
library(ggrepel)
library(boot)
library(fastICA)

#Step 1
data <- read.csv("Data/creditscoring.csv", header = TRUE)

n     <- dim(data)[1]
set.seed(12345)
id    <- sample(1:n, floor(n*0.5))
train <- data[id,]

id1   <- setdiff(1:n, id)
set.seed(12345)
id2   <- sample(id1, floor(n*0.25))
valid <- data[id2,]
id3   <- setdiff(id1,id2)
test  <- data[id3,]
```

```r
#Step 2

D_tree <- function(data, measure){
  model     <- tree(as.factor(good_bad) ~ ., data = train, split = measure)
  fit       <- predict(model, newdata = data, type="class")
  con_mat   <- table( "Actual" = data$good_bad, "Predicted" = fit)
  miss_rate <- 1-sum(diag(con_mat))/sum(con_mat)

  result    <- list("Confusion Matrix" = con_mat, "Missclassification Rate" = miss_rate)
  return(result)
}

print("Training data")
D_tree(train, "deviance")

print("Testing data")
D_tree(test, "deviance")

print("Training data")
D_tree(train, "gini")

print("Testing data")
D_tree(test, "gini")

#Step 3
my_tree <- tree(as.factor(good_bad) ~ ., data = train, split = "deviance")

index      <- summary(my_tree)[4]$size
trainScore <- rep(0,index)
testScore  <- rep(0,index)

for(i in 2:index) {
  prunedTree    <- prune.tree(my_tree,best=i)
  pred          <- predict(prunedTree, newdata=valid,  type="tree")
  trainScore[i] <- deviance(prunedTree)
  testScore[i]  <- deviance(pred)
}

plot(2:index,
     trainScore[2:index],
     col  = "Red",
     type = "b",
     main = "Dependence of Deviance",
     ylim = c(250,600),
     pch  = 19,
     cex  = 1,
     xlab = "Number of Leaves",
     ylab = "Deviance")
points(2:index,
       testScore[2:index],
       col  = "Blue",
       type = "b",
       pch  = 19,
       cex  = 1)


final_tree <- prune.tree(my_tree, best = 4)
final_fit  <- predict(final_tree, newdata = test, type="class")
final_mat  <- table("Actual" = test$good_bad, "Predeicted" = final_fit)
final_rate <- 1-sum(diag(final_mat))/sum(final_mat)

plot(final_tree)
text(final_tree, pretty = 0)
summary(final_tree)
list("Confusion Matrix" = final_mat, "Misclassification Rate" = final_rate)

#Step 4
bayes <- function(data){
  model     <- naiveBayes(as.factor(good_bad) ~ ., data = train)
  fit       <- predict(model, newdata = data)
  con_mat   <- table("Actual" = data$good_bad, "Predicted" = fit)
  miss_rate <- 1-sum(diag(con_mat))/sum(con_mat)

  result    <- list("Confusion Matrix" = con_mat, "Missclassification Rate" = miss_rate)
  return(result)
}
print("Training data")
bayes(train)
print("Testing data")
bayes(test)


#Step 5

pi          <- seq(0.05, 0.95, 0.05)

tree_fit    <- predict(final_tree, newdata = test)
tree_good   <- tree_fit[,2]
true_assign <- ifelse(test$good_bad == "good", 1, 0)

tree_TPR_FPR   <- matrix(nrow = 2, ncol = length(pi))
rownames(tree_TPR_FPR) <- c("TPR", "FPR")

for (i in 1:length(pi)){
  tree_assign <- ifelse(tree_good > pi[i], 1, 0)
  tree_mat    <- confusion.matrix(tree_assign, true_assign)

  tpr1 <- tree_mat[2,2]/(tree_mat[2,1] + tree_mat[2,2])
  fpr1 <- tree_mat[1,2]/(tree_mat[1,1] + tree_mat[1,2])

  tree_TPR_FPR[,i] <- c(tpr1,fpr1)
}

knitr::kable(round(tree_TPR_FPR,2))

#options(scipen = 999)
```

```r
bayes      <- naiveBayes(good_bad ~ ., data = train)
bayes_fit  <- predict(bayes, newdata = test, type = "raw")
bayes_good <- bayes_fit[,2]

bayes_TPR_FPR <- matrix(nrow = 2, ncol = length(pi))
rownames(bayes_TPR_FPR) <- c("TPR", "FPR")


for (i in 1:length(pi)) {
  bayes_assign <- ifelse(bayes_good > pi[i], 1, 0)
  bayes_mat    <- confusion.matrix(bayes_assign, true_assign)

  tpr2 <- bayes_mat[2,2]/(bayes_mat[2,1] + bayes_mat[2,2])
  fpr2 <- bayes_mat[1,2]/(bayes_mat[1,1] + bayes_mat[1,2])

  bayes_TPR_FPR[,i] <- c(tpr2,fpr2)
}

knitr::kable(round(bayes_TPR_FPR,2))

# ROC Optimal Tree & Naive Bayes
ggplot() +
  geom_line(aes(x = tree_TPR_FPR[2,], y = tree_TPR_FPR[1,], col = "Optimal Tree")) +
  geom_line(aes(x = bayes_TPR_FPR[2,], y = bayes_TPR_FPR[1,], col = "Naive Bayes")) +
  xlab("False-Positive Rate") +
  ylab("True-Positive Rate") +
  ggtitle("ROC")

loss_mat <- matrix(c(0,10,1,0), nrow = 2)

loss_fun <- function(data,loss_mat){
  prob        <- ifelse(data$good_bad == "good",1,0)

  bayes_model <- naiveBayes(as.factor(good_bad) ~ ., data = train)
  bayes_fit   <- predict(bayes_model, newdata = data, type = "raw")

  #To penalize the FPR, the probability of the predicted as good need to be
  #10 times the probability of the predicted as bad to be classified as good
  bayes_fit   <- ifelse(loss_mat[1,2] * bayes_fit[,2] > loss_mat[2,1] * bayes_fit[,1],1,0)

  con_mat     <- table("Actual" = prob, "Predicted" = bayes_fit)
  miss_rate   <- 1-sum(diag(con_mat))/sum(con_mat)
  rownames(con_mat) <- c("Bad", "Good")
  colnames(con_mat) <- c("Bad", "Good")

  result    <- list("Confusion Matrix" = con_mat, "Missclassification Rate" = miss_rate)
  return(result)
  }

print("Training data")
loss_fun(train,loss_mat)
print("Testing data")
loss_fun(test,loss_mat)
```

```r
### Assignment 3. Uncertainty estimation

state <- read.csv2("Data/State.csv", header = TRUE)
state <- state[order(state$MET),]

ggplot(data = as.data.frame(state), aes(y = state[,1], x = state[,3]))+
  xlab("MET") + ylab("EX")+
  geom_text_repel(label = state[,8], size = 2)+
  geom_point(color = 'red')

tree_model <- tree(EX ~ MET,
                   data = state,
                   control = tree.control(nobs = nrow(state),
                                          minsize = 8))
set.seed(12345)
best_tree1 <- cv.tree(tree_model)
best_tree2 <- prune.tree(tree_model, best = 3)
summary(best_tree2)

plot(best_tree2)
text(best_tree2, pretty=1,
     cex = 0.8,
     xpd = TRUE)

tree_pred <- predict(best_tree2, newdata = state)

ggplot(data = as.data.frame(state),
       aes(y = state[,1], x = state[,3])) +
  xlab("MET") +
  ylab("EX") +
  geom_point(col = "red") +
  geom_point(x = state$MET, y = tree_pred, col = "blue")

hist(residuals(best_tree2),
     main = "Residuals Histogram",
     xlab = "Residuals")

f <- function(data, ind){
  set.seed(12345)
  sample  <- state[ind,]
  my_tree <- tree(EX ~ MET,
                data = sample,
                control = tree.control(nobs = nrow(sample), minsize = 8))

  pruned_tree <- prune.tree(my_tree, best = 3)

  pred    <- predict(pruned_tree, newdata = state)
  return(pred)
}

res  <- boot(state, f, R=1000)
```

```
conf <- envelope(res, level=0.95)

ggplot(data = as.data.frame(state),
       aes(y = state[,1], x = state[,3])) +
  xlab("MET") +
  ylab("EX") +
  geom_point(col = "red") +
  geom_line(aes(x = state$MET, y = tree_pred), col = "blue") +
  geom_line(aes(x = state$MET, y = conf$point[1,]), col = "orange") +
  geom_line(aes(x = state$MET, y = conf$point[2,]), col = "orange")

mle <- best_tree2

rng <- function(data, mle){
  data1    <- data.frame(EX = data$EX, MET = data$MET)
  n        <- length(data1$EX)
  pred     <- predict(mle, newdata = state)
  residual <- data1$EX - pred
  data1$EX <- rnorm(n, pred, sd(residual))
  return(data1)
}

f1 <- function(data){
  res      <- tree(EX ~ MET,
                   data = data,
                   control = tree.control(nobs=nrow(state),minsize = 8))
  opt_res  <- prune.tree(res, best = 3)
  return(predict(opt_res, newdata = data))
}

f2 <- function(data){
  res      <- tree(EX ~ MET,
                   data = data,
                   control = tree.control(nobs=nrow(state),minsize = 8))
  opt_res  <- prune.tree(res, best = 3)
  n        <- length(state$EX)
  opt_pred <- predict(opt_res, newdata = state)
  pred     <- rnorm(n,opt_pred, sd(residuals(mle)))
  return(pred)
}
set.seed(12345)
par_boot_conf <- boot(state, statistic = f1, R = 1000, mle = mle, ran.gen = rng, sim = "parametric")
conf_interval <- envelope(par_boot_conf, level=0.95)

set.seed(12345)
par_boot_pred <- boot(state, statistic = f2, R = 1000, mle = mle, ran.gen = rng, sim = "parametric")
pred_interval <- envelope(par_boot_pred, level=0.95)


ggplot(data = as.data.frame(state),
       aes(y = state[,1], x = state[,3])) +
  xlab("MET") +
  ylab("EX") +
  geom_point(col = "red") +
  geom_line(aes(x = state$MET, y = tree_pred), col = "blue") +
  geom_line(aes(x = state$MET, y = conf_interval$point[1,]), col = "orange") +
  geom_line(aes(x = state$MET, y = conf_interval$point[2,]), col = "orange") +
  geom_line(aes(x = state$MET, y = pred_interval$point[1,]), col = "black") +
  geom_line(aes(x = state$MET, y = pred_interval$point[2,]), col = "black")



#Assignment 4. Principal components

data    <- read.csv2("Data/NIRspectra.csv", header = TRUE)
spectra <- data

spectra$Viscosity <- c()
comp              <- prcomp(spectra)
lambda            <- comp$sdev^2

var               <- sprintf("%2.3f", lambda/sum(lambda)*100)

screeplot(comp, main = "Principal Components")

ggplot() +
  geom_point(aes(comp$x[,1],comp$x[,2])) +
  xlab("x1") + ylab("x2")

plot(comp$rotation[,1],
     main="PC1 Traceplot",
     xlab = "Features",
     ylab = "Scores")
plot(comp$rotation[,2],
     main="PC2 Traceplot",
     xlab = "Features",
     ylab = "Scores")


a   <- as.matrix(spectra)
set.seed(12345)
ica <- fastICA(a,
               2,
               fun = "logcosh",
               alpha = 1,
               row.norm = FALSE,
               maxit = 200,
               tol = 0.0001,
               verbose = TRUE)

posterior = ica$K %*% ica$W

plot(posterior[,1],
     main="PC1 Traceplot",
     xlab = "Features",
     ylab = "Scores")
```

```r
plot(posterior[,2],
     main="PC2 Traceplot",
     xlab = "Features",
     ylab = "Scores")

ggplot() +
  geom_point(aes(ica$S[,1],ica$S[,2])) +
  labs(x = "W1", y = "W2")
```