732A99 chetabook

Anubhav Dikshit(anudi287)

13 Januray 2019

Contents

Simple Tasks

Simple Tasks

Library

```r
library("ggplot2") # plots
library("tree") # decision tree
library("caret") # summary and confusion table
library("kknn") # kknn
library("xlsx") # reading excel
library("MASS") # Step AIC
library("jtools") # summ function
library("dplyr") # pipelining
library("glmnet") # lasso and ridge
library("mgcv") # spline
library("kernlab") # SVM
library("mboost") # ensemble ADA boost
library("randomForest") # randomforest
library("pamr") # Nearest shrunken
library("boot") # bootstrap
library("fastICA") # fastICA
library("MASS") # LDA
library("neuralnet") # Neural Network
library("e1071") # Naive Bayes

# colours (colour blind friendly)
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2",
               "#D55E00", "#CC79A7")
```

Reading Excel

```r
data <- xlsx::read.xlsx("spambase.xlsx", sheetName= "spambase_data")
data$Spam <- as.factor(data$Spam)
```

Spliting the Datasets

Divide into train/test

```r
data <- xlsx::read.xlsx("spambase.xlsx", sheetName= "spambase_data")
data$Spam <- as.factor(data$Spam)
# 50-50 split
n=nrow(data)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
```

Train/test/validation

```
data <- xlsx::read.xlsx("spambase.xlsx", sheetName= "spambase_data")
data$Spam <- as.factor(data$Spam)
# 50-25-25 split
n=nrow(data)
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,]
```

Custom code for Cross-Validation

```
#Randomly shuffle the data
new_data <- data[sample(nrow(data)),]

#Create N equally size folds
new_data$folds <- sample(rep(1:10, each = nrow(new_data)/10))

result <- NULL
#Perform N fold cross validation
for(i in 1:length(unique(new_data$folds))){
  testData <- new_data[new_data$folds != i,]
  trainData <- new_data[new_data$folds == i,]

  #Use the test and train data partitions however you desire, run model code here

  best_model <- glm(formula = Spam ~., family = binomial, data = trainData)
  predicted_value <- predict(best_model, testData, type = "response")
  pred_class <- ifelse(predicted_value > 0.50, 1, 0)
  temp <- 1 - (sum(ifelse(pred_class == testData$Spam,1,0))/nrow(testData))
  temp <- cbind(temp, i)
  colnames(temp) <- c("test_error", "fold")
  result <- rbind(result, temp)
  }
```

Misclassification error calculation

```
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
```

```
#missclass(data2$class, predict(m3, type="class")$class)
```

Regression

Logistic Regression

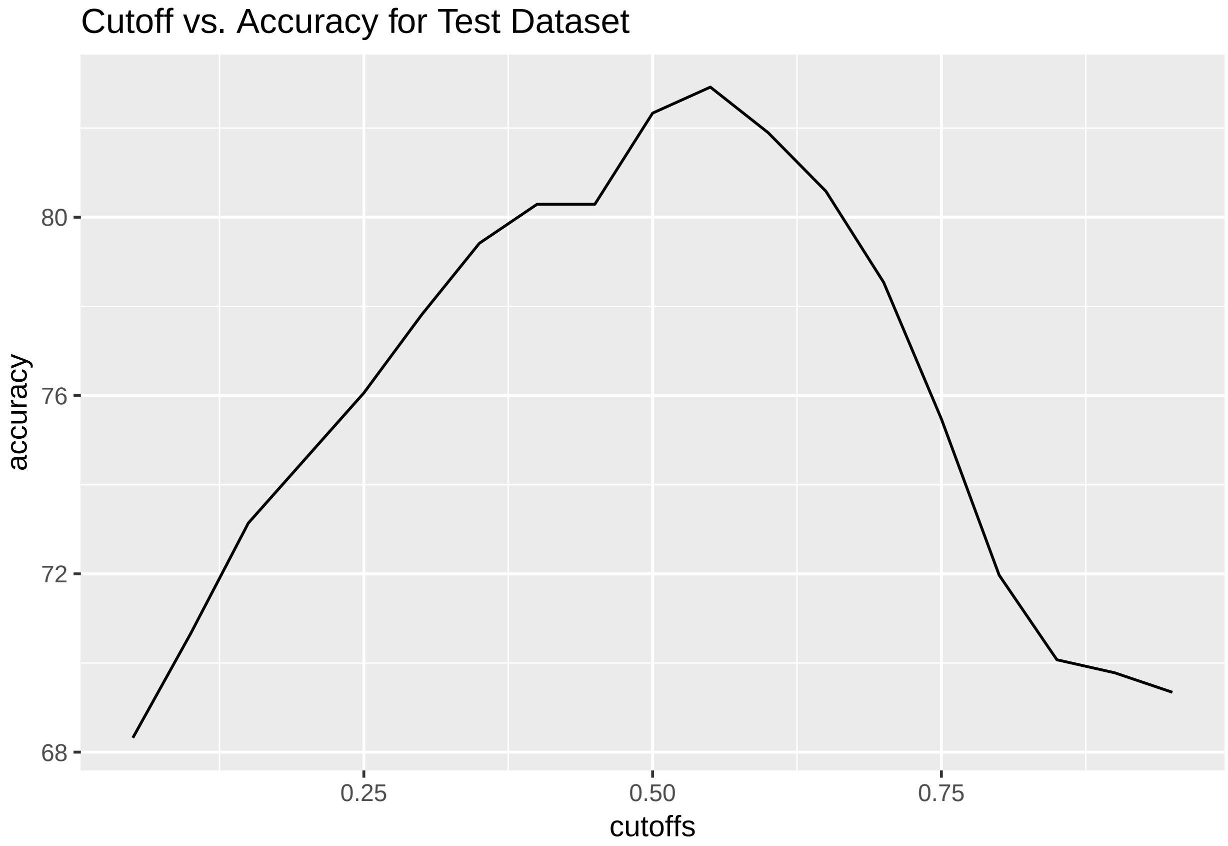
```
best_model <- glm(formula = Spam ~., family = binomial, data = train)
#summary(best_model)

train$prediction_prob <- predict(best_model, newdata = train, type = "response")
train$prediction_class_50 <- ifelse(train$prediction_prob > 0.50, 1, 0)

test$prediction_prob <- predict(best_model, newdata = test, type = "response")
test$prediction_class_50 <- ifelse(test$prediction_prob > 0.50, 1, 0)

conf_train <- table(train$Spam, train$prediction_class_50)
names(dimnames(conf_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_train)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Train
## Actual Train   0   1
##            0 803 142
##            1  81 344
##
##                Accuracy : 0.8372
##                  95% CI : (0.8166, 0.8564)
##     No Information Rate : 0.6453
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6341
##  Mcnemar's Test P-Value : 5.872e-05
##
##             Sensitivity : 0.9084
##             Specificity : 0.7078
##          Pos Pred Value : 0.8497
##          Neg Pred Value : 0.8094
##              Prevalence : 0.6453
##          Detection Rate : 0.5861
##    Detection Prevalence : 0.6898
##       Balanced Accuracy : 0.8081
##
##        'Positive' Class : 0
##
```

Choosing the best cutoff for test

```
cutoffs <- seq(from = 0.05, to = 0.95, by = 0.05)
accuracy <- NULL
```

```
for (i in seq_along(cutoffs)){
    prediction <- ifelse(test$prediction_prob >= cutoffs[i], 1, 0) #Predicting for cut-off

    accuracy <- c(accuracy,length(which(test$Spam == prediction))/length(prediction)*100)}

cutoff_data <- as.data.frame(cbind(cutoffs, accuracy))

ggplot(data = cutoff_data, aes(x = cutoffs, y = accuracy)) +
  geom_line() +
  ggtitle("Cutoff vs. Accuracy for Test Dataset")
```



KKNN

```
knn_model30 <- train.kknn(Spam ~., data = train, kmax = 30)

test$knn_prediction_class <- predict(knn_model30, test)

conf_test2 <- table(test$Spam, test$knn_prediction_class)
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)
```

```
## Confusion Matrix and Statistics
##
##              Predicted Test
## Actual Test   0   1
##           0 402  74
##           1  66 143
##
##                Accuracy : 0.7956
##                  95% CI : (0.7634, 0.8252)
##     No Information Rate : 0.6832
##     P-Value [Acc > NIR] : 3.278e-11
##
##                   Kappa : 0.5231
##  Mcnemar's Test P-Value : 0.5541
##
##             Sensitivity : 0.8590
##             Specificity : 0.6590
##          Pos Pred Value : 0.8445
##          Neg Pred Value : 0.6842
##              Prevalence : 0.6832
##          Detection Rate : 0.5869
##    Detection Prevalence : 0.6949
##       Balanced Accuracy : 0.7590
##
##        'Positive' Class : 0
##
```

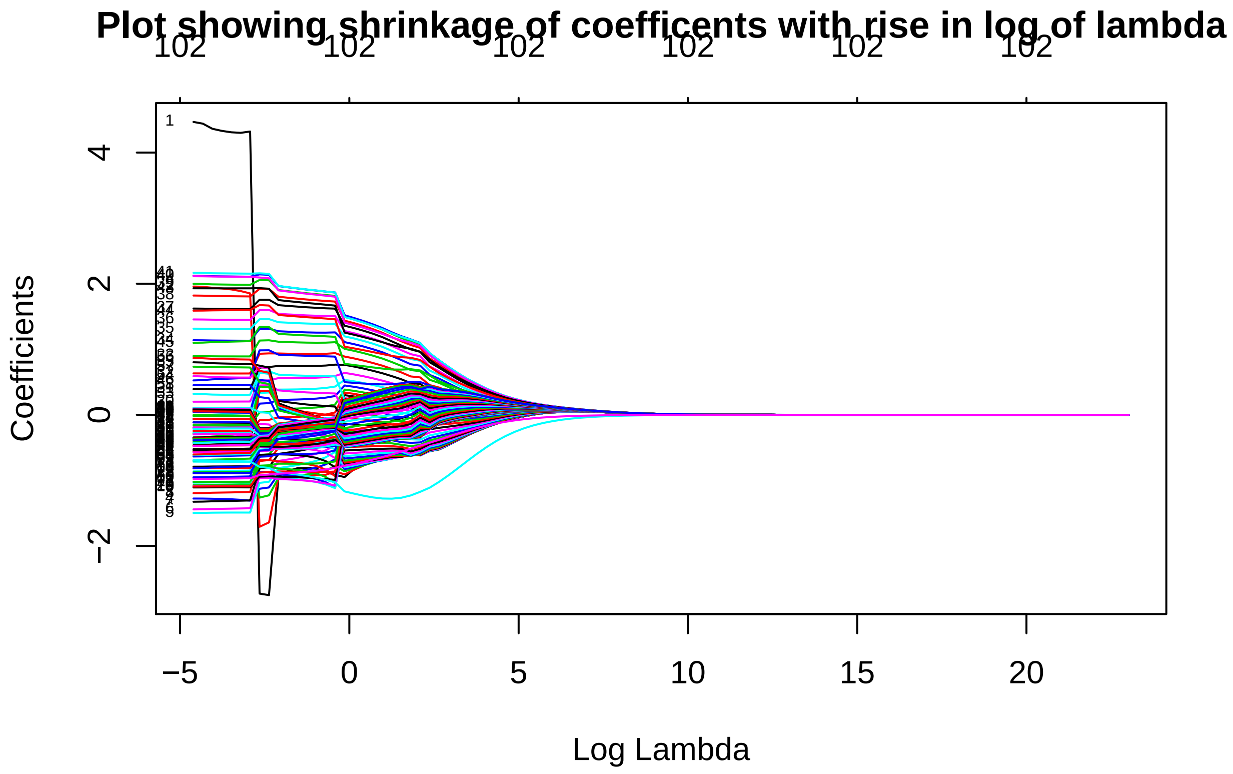
Step AIC

```r
tecator_data <- read.xlsx("tecator.xlsx", sheetName = "data")
tecator_data <- tecator_data[,2:NCOL(tecator_data)] # removing sample column


min.model1 = lm(Fat ~ 1, data=tecator_data[,-1])
biggest1 <- formula(lm(Fat ~.,  data=tecator_data[,-1]))

step.model1 <- stepAIC(min.model1, direction ='forward', scope=biggest1, trace = FALSE)
summ(step.model1)
```

```
## MODEL INFO:
## Observations: 215
## Dependent Variable: Fat
## Type: OLS linear regression
##
## MODEL FIT:
## F(29,185) = 4775.35, p = 0.00
## R² = 1.00
## Adj. R² = 1.00
##
## Standard errors: OLS
##                 Est.    S.E. t val.    p
## (Intercept)    93.46    1.59  58.86 0.00 ***
## Moisture       -1.03    0.02 -54.25 0.00 ***
## Protein        -0.64    0.06 -10.91 0.00 ***
## Channel100     66.56   48.18   1.38 0.17
## Channel41   -3268.11  826.92  -3.95 0.00 ***
## Channel7      -64.03   20.80  -3.08 0.00  **
## Channel48   -2022.46  254.46  -7.95 0.00 ***
## Channel42    4934.22 1124.96   4.39 0.00 ***
## Channel50    1239.52  236.09   5.25 0.00 ***
## Channel45    4796.22  783.38   6.12 0.00 ***
## Channel66    2435.79 1169.85   2.08 0.04   *
## Channel56    2373.00  540.06   4.39 0.00 ***
## Channel90    -258.27  247.22  -1.04 0.30
## Channel60    -264.27  708.11  -0.37 0.71
## Channel70      14.25  327.12   0.04 0.97
## Channel67   -2015.92  543.74  -3.71 0.00 ***
## Channel59     635.71  996.31   0.64 0.52
## Channel65    -941.61 1009.23  -0.93 0.35
## Channel58    1054.24  927.95   1.14 0.26
## Channel44   -5733.84 1079.19  -5.31 0.00 ***
## Channel18     299.80   88.43   3.39 0.00 ***
## Channel78    2371.11  361.25   6.56 0.00 ***
## Channel84    -428.99  338.35  -1.27 0.21
## Channel62    3062.97  769.59   3.98 0.00 ***
## Channel53    -804.39  203.44  -3.95 0.00 ***
## Channel75   -1461.42  402.26  -3.63 0.00 ***
## Channel57   -3266.79  876.71  -3.73 0.00 ***
## Channel63   -2844.66  906.40  -3.14 0.00  **
## Channel24    -308.71   97.87  -3.15 0.00  **
## Channel37     401.64  151.76   2.65 0.01  **
```

Ridge Regression

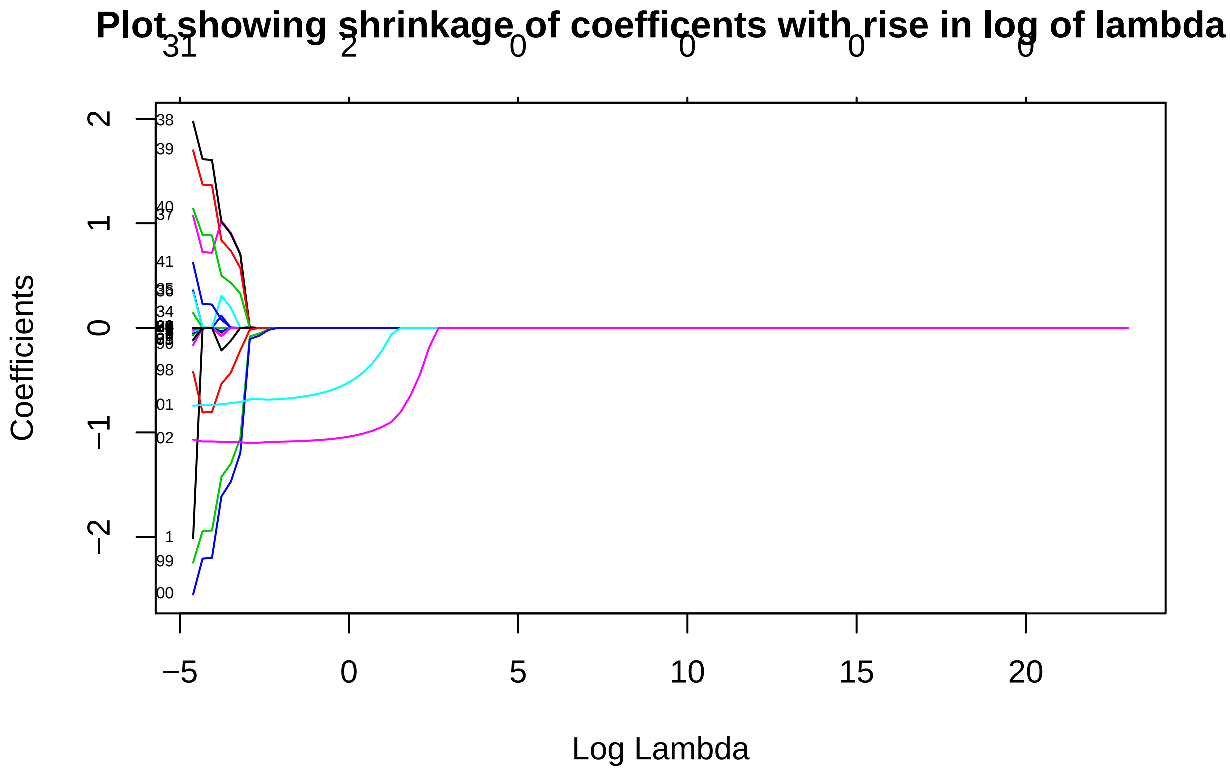
```
y <- tecator_data %>% select(Fat) %>% data.matrix()
x <- tecator_data %>% select(-c(Fat)) %>% data.matrix()

lambda <- 10^seq(10, -2, length = 100)

ridge_fit <- glmnet(x, y, alpha = 0, family = "gaussian", lambda = lambda)
plot(ridge_fit, xvar = "lambda", label = TRUE,
     main = "Plot showing shrinkage of coefficents with rise in log of lambda")
```



```
## Change of coefficent with respect to lambda
result <- NULL
for(i in lambda){
temp <- t(coef(ridge_fit, i)) %>% as.matrix()
temp <- cbind(temp, lambda = i)
result <- rbind(temp, result)
}
result <- result %>% as.data.frame() %>% arrange(lambda)
```

Lasso Regression

```
lambda <- 10^seq(10, -2, length = 100)

lasso_fit <- glmnet(x, y, alpha = 1, family = "gaussian", lambda = lambda)
plot(lasso_fit, xvar = "lambda", label = TRUE,
     main = "Plot showing shrinkage of coefficents with rise in log of lambda")
```



Lasso Regression using Cross Validation

```
#find the best lambda from our list via cross-validation

lambda_lasso <- 10^seq(10, -2, length = 100)
lambda_lasso[101] <- 0
lasso_cv <- cv.glmnet(x,y, alpha=1, lambda = lambda_lasso, type.measure="mse")

#coef(lasso_cv, lambda = lasso_cv$lambda.min)

lasso_cv$lambda.min
```

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

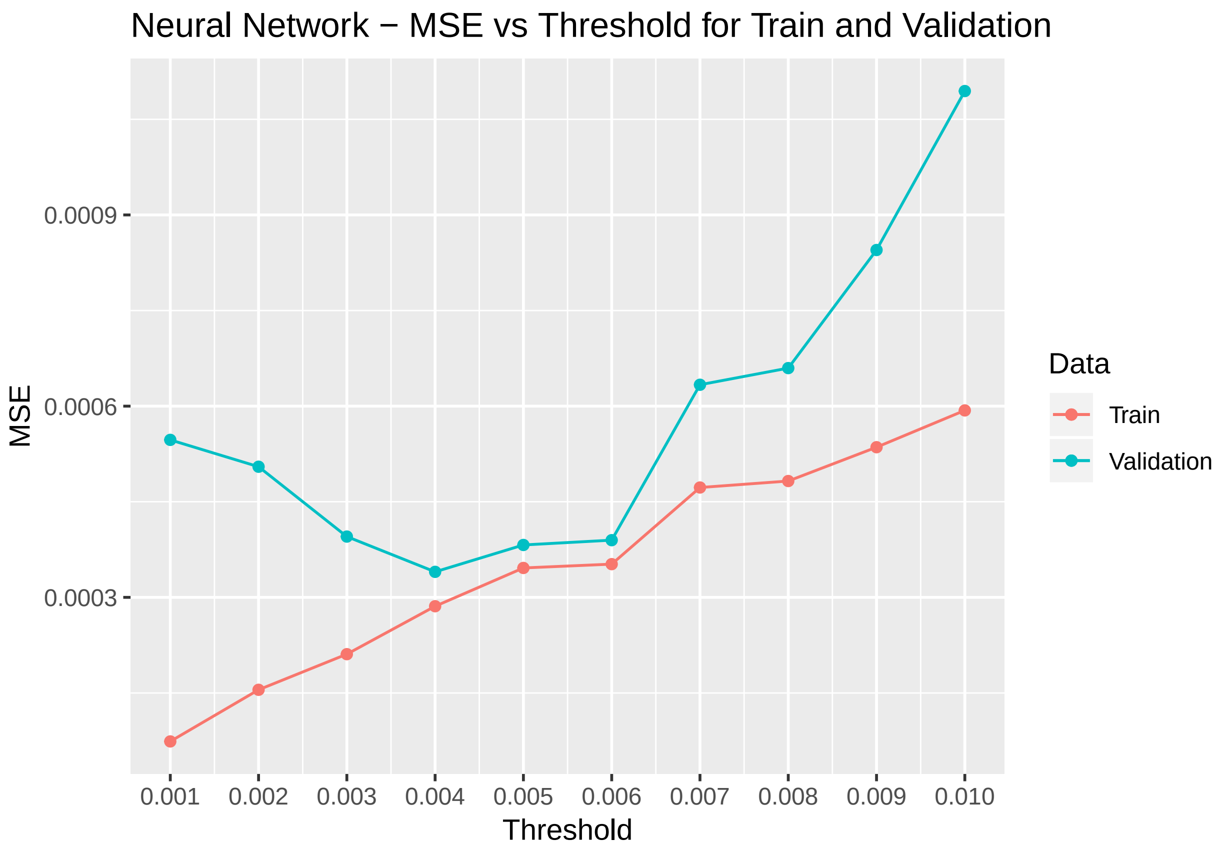
```
## Change of coefficent with respect to lambda
result_lasso <- NULL
for(i in 1:length(lambda_lasso)){
temp <- lasso_cv$cvm[i] %>% as.matrix()
temp <- cbind(CVM_error = temp, lambda = lasso_cv$lambda[i])
result_lasso <- rbind(temp, result_lasso)
}
```

Neural Network

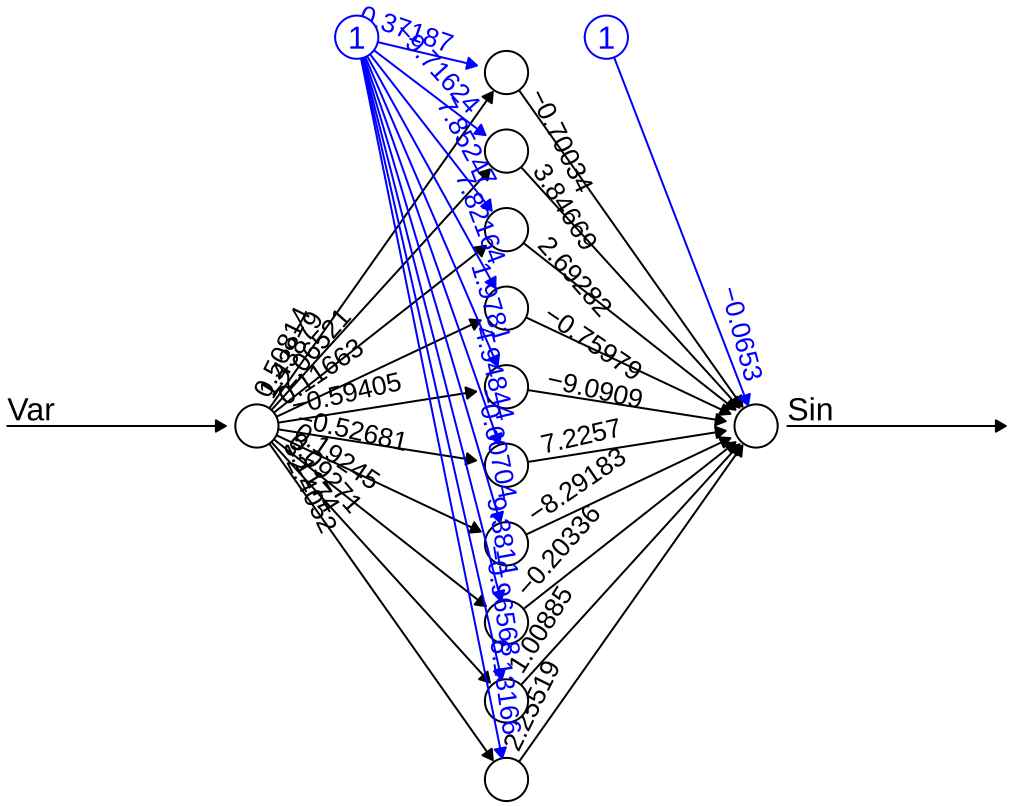
```r
#Generating data
set.seed(1234567890)
Var = runif(50, 0, 10)
trva = data.frame(Var, Sin = sin(Var))

# Training and validation split
tr = trva[1:25, ] # Training
va = trva[26:50, ] # Validation
nn_val_res_df = data.frame()

# Random initialization of the weights in the interval [-1, 1]
w_init = runif(31, -1, 1)



for(i in 1:10) {
print(paste("Running NN: ", i))
set.seed(1234567890)

# Training neural network
nn = neuralnet(Sin ~ Var, data = tr, hidden = 10,
startweights = w_init, threshold = i / 1000)

# Predicting values for train and validation
va_res = neuralnet::compute(nn, va$Var)$net.result
tr_res = neuralnet::compute(nn, tr$Var)$net.result

# Computing train and validation MSE
tr_mse = mean((tr_res - tr$Sin)^2)
va_mse = mean((va_res - va$Sin)^2)

# Storing data in data frame
nn_val_res_df = rbind(nn_val_res_df,
data.frame(thres_num = i, thres_val = i / 1000,
val_mse = va_mse, trn_mse = tr_mse))
}
```

```
## [1] "Running NN:  1"
## [1] "Running NN:  2"
## [1] "Running NN:  3"
## [1] "Running NN:  4"
## [1] "Running NN:  5"
## [1] "Running NN:  6"
## [1] "Running NN:  7"
## [1] "Running NN:  8"
## [1] "Running NN:  9"
## [1] "Running NN:  10"
```

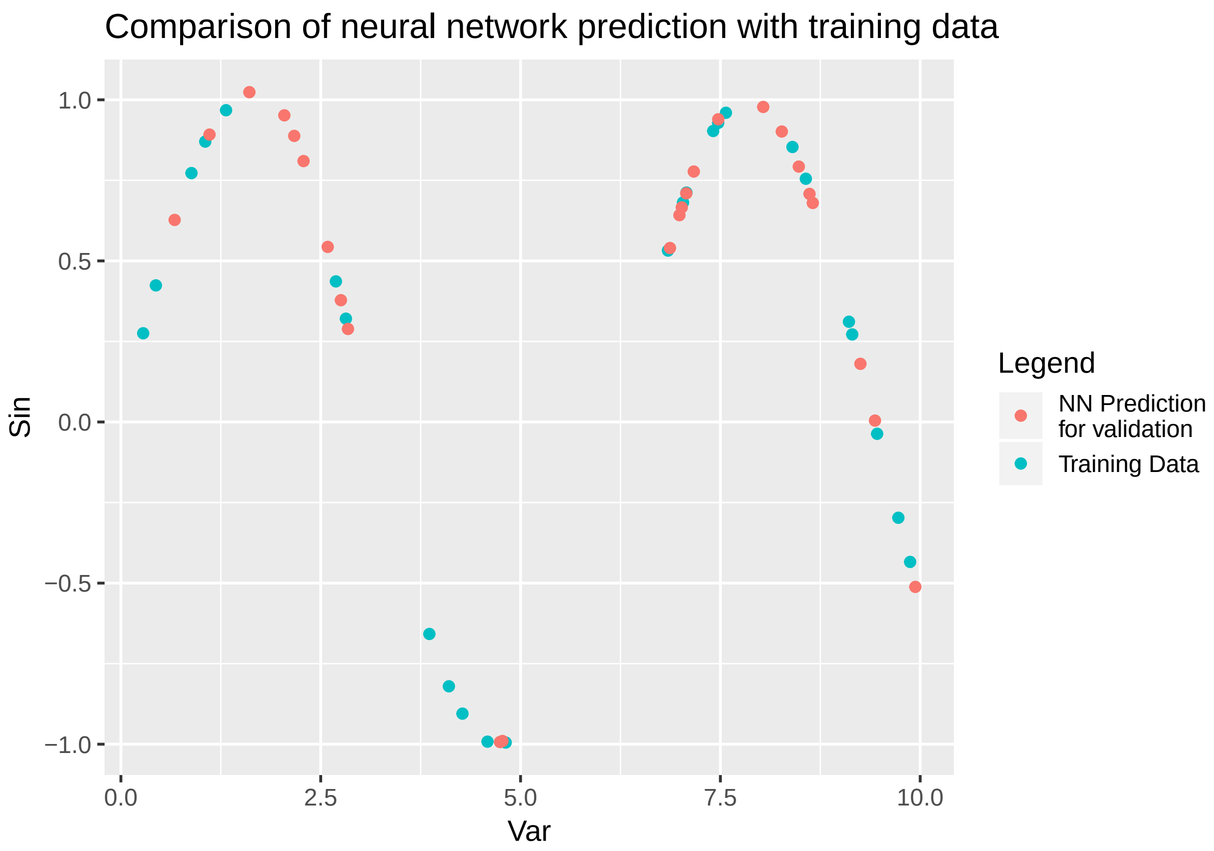
```r
# Plot of MSE vs threshold for train and validation
ggplot(nn_val_res_df) +
geom_point(aes(x = thres_val, y = val_mse, color = "Validation")) +
geom_line(aes(x = thres_val, y = val_mse, color = "Validation")) +
geom_point(aes(x = thres_val, y = trn_mse, color = "Train")) +
geom_line(aes(x = thres_val, y = trn_mse, color = "Train")) +
xlab("Threshold") + ylab("MSE") + labs(color = "Data") +
scale_x_continuous(breaks = (1:10)/1000) +
ggtitle("Neural Network - MSE vs Threshold for Train and Validation")
```



```
# Final neural network
# Best threshold = 0.001
opt_nn = neuralnet(Sin ~ Var, data = tr, hidden = 10,
startweights = w_init, threshold = 0.001)
plot(x = opt_nn, rep = "best", information = F)
```



```
# Plot of the predictions and the data
nn_pred_df = tr
nn_pred_df$Type = "Training Data"
nn_pred_df = rbind(nn_pred_df,
data.frame(Var = va$Var,
Sin = neuralnet::compute(opt_nn, va$Var)$net.result,
Type = "NN Prediction \nfor validation"))
ggplot(nn_pred_df, aes(x = Var, y = Sin, color = Type)) + geom_point() +
ggtitle("Comparison of neural network prediction with training data") +
labs(color = "Legend")
```

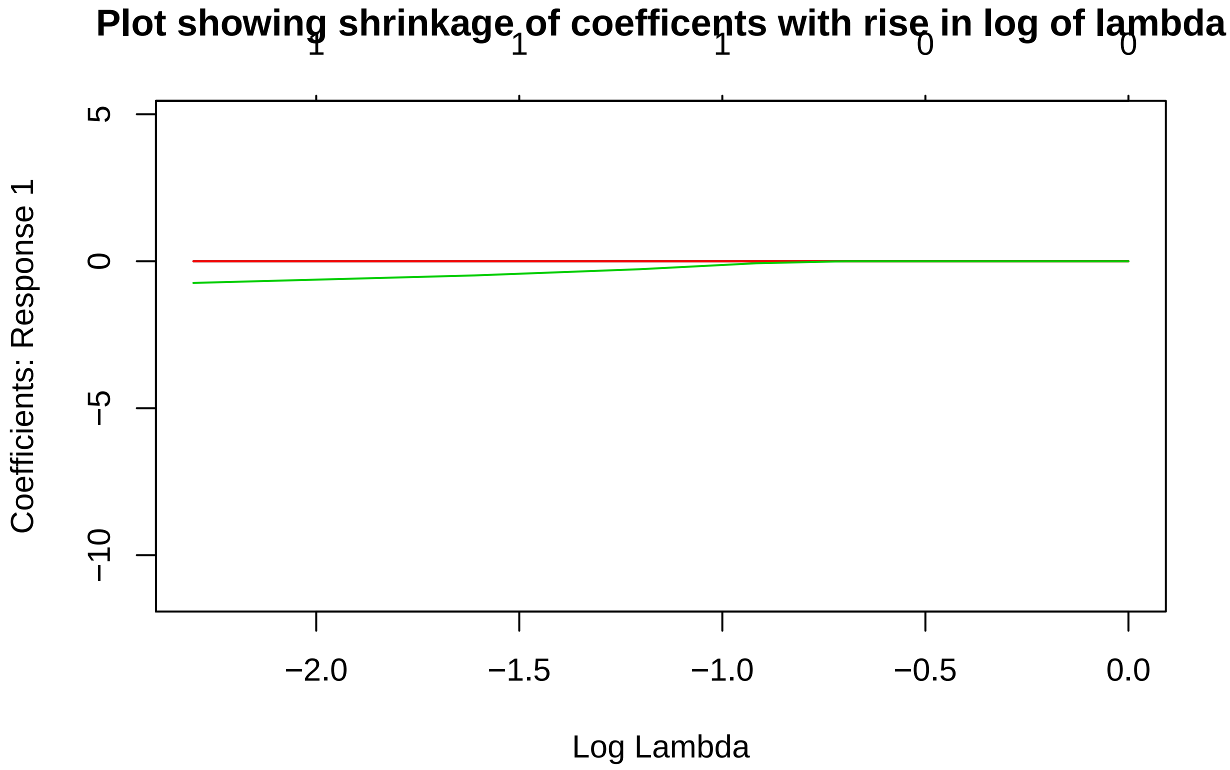


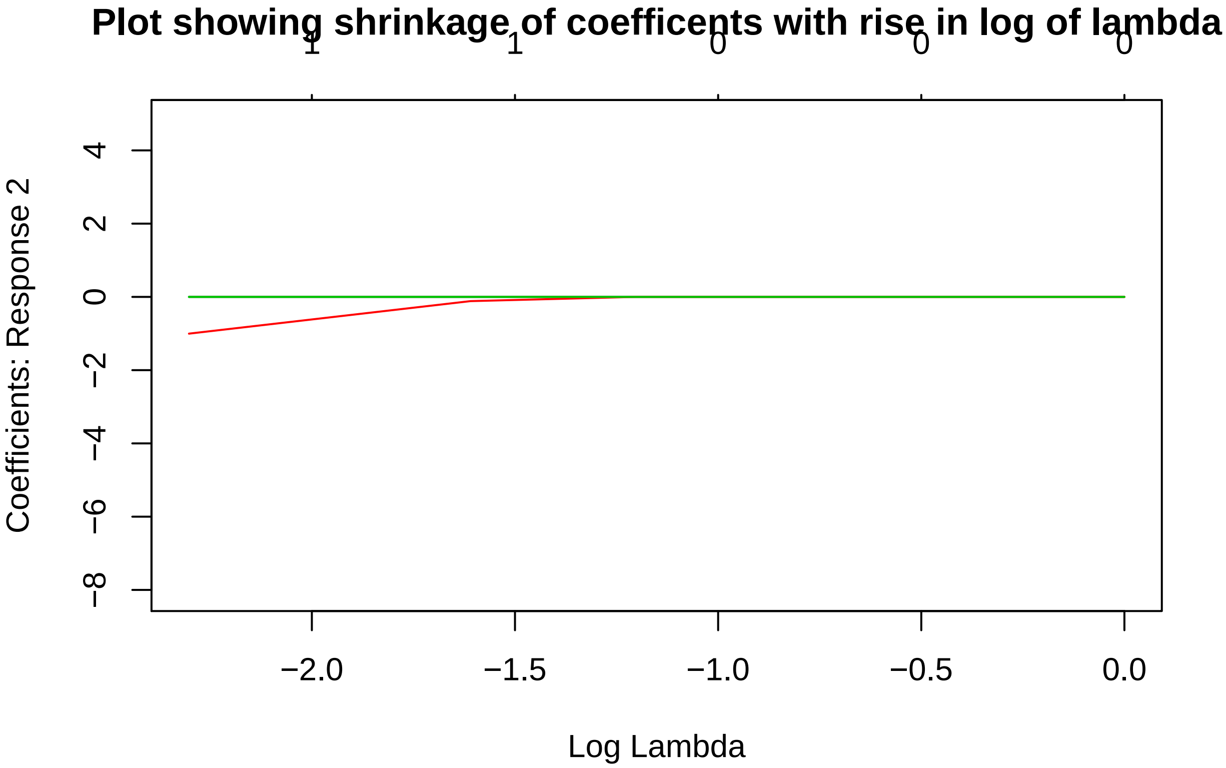
Classification

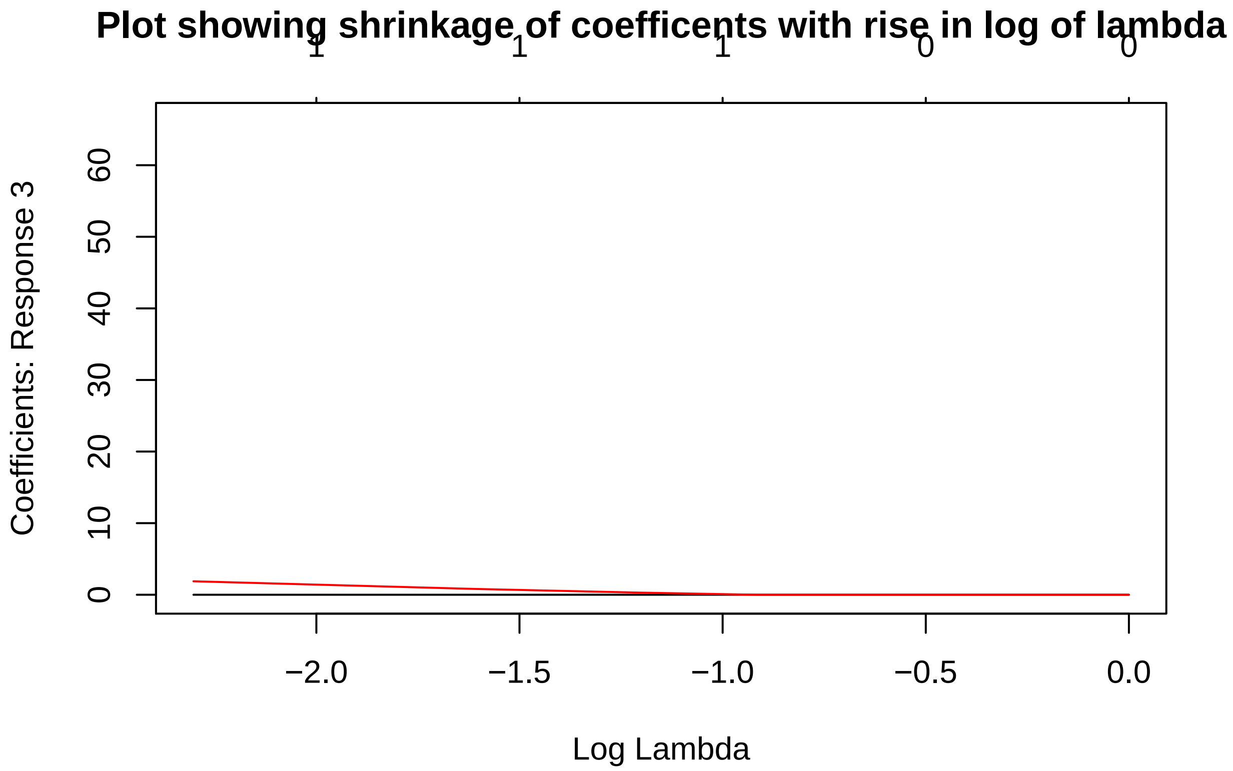
LASSO

```
n=dim(iris)[1]
set.seed(12345)
id=sample(1:n, floor(n*1/3))
train=iris[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*1/3))
valid=iris[id2,]
id3=setdiff(id1,id2)
test=iris[id3,]

y <- train %>% select(Species) %>% data.matrix()
x <- train %>% select(-c(Species)) %>% data.matrix()

y_valid <- valid %>% select(Species) %>% data.matrix()
x_valid <- valid %>% select(-c(Species)) %>% data.matrix()

lambda <- seq(from=0, to=1, by=0.1)
lasso_fit <- glmnet(x, y, alpha = 1, family = "multinomial", lambda = lambda)
```

```
plot(lasso_fit, xvar = "lambda", label = TRUE,
main = "Plot showing shrinkage of coefficents with rise in log of lambda")
```




Plot showing shrinkage of coefficents with rise in log of lambda
1
1
0
0
0
Coefficients: Response 2
4
2
0
−2
−4
−6
−8
−2.0
−1.5
−1.0
−0.5
0.0
Log Lambda



```
predicted <- predict(lasso_fit, newx = x_valid, type=c("class"))
new_predicted <- cbind(predicted, y_valid) %>% as.data.frame()

out <- NULL
for(i in 0:10){
  error <- ifelse(new_predicted[,i] == as.character(new_predicted$Species), 1, 0)
temp <- cbind(i, NROW(new_predicted) - sum(error))
out <- rbind(out, temp)
}
```

Naive Bayes, using default threshold

```
set.seed(12345)
credit_data <- xlsx::read.xlsx("creditscoring.xls", sheetName = "credit")
credit_data$good_bad <- as.factor(credit_data$good_bad)

n=NROW(credit_data)
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=credit_data[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=credit_data[id2,]
```

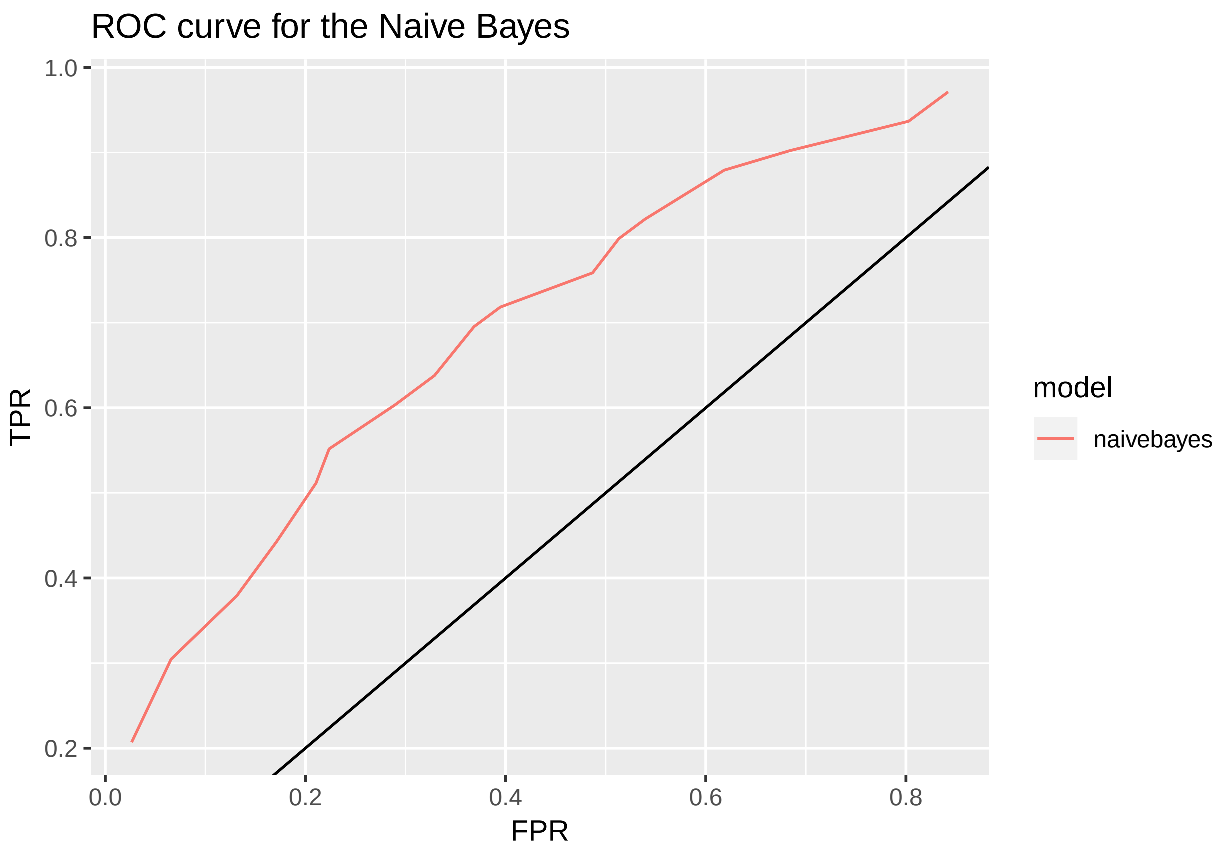
```r
id3=setdiff(id1,id2)
test=credit_data[id3,]

#Fitting the Naive Bayes model
credit_naive_model = e1071::naiveBayes(good_bad ~., data=train)

#Prediction on the dataset
predict_naive_train = predict(credit_naive_model, newdata=train, type = "class")
predict_naive_test = predict(credit_naive_model, newdata=test, type = "class")

conf_naive_train <- table(train$good_bad, predict_naive_train)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)
```

```
## Confusion Matrix and Statistics
##
##             Predicted Train
## Actual Train bad good
##         bad   95   52
##         good  98  255
##
##                Accuracy : 0.7
##                  95% CI : (0.6577337, 0.7398824)
##     No Information Rate : 0.614
##     P-Value [Acc > NIR] : 0.0000365481
##
##                   Kappa : 0.3377951
##  Mcnemar's Test P-Value : 0.0002385635
##
##             Sensitivity : 0.4922280
##             Specificity : 0.8306189
##          Pos Pred Value : 0.6462585
##          Neg Pred Value : 0.7223796
##              Prevalence : 0.3860000
##          Detection Rate : 0.1900000
##    Detection Prevalence : 0.2940000
##       Balanced Accuracy : 0.6614234
##
##        'Positive' Class : bad
##
```

Naive Bayes varying threshold and ROC curve

```r
# model
credit_naive_model = e1071::naiveBayes(good_bad ~., data=train)

# predicting class, getting probability
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")

# data mugging
probability_data_naive <- as.data.frame(cbind(predict_naive_test_prob,
                                              as.character(test$good_bad), "naivebayes"))
```

```
colnames(probability_data_naive) <- c("prob_bad", "prob_good",
                                         "actual_test_class", "model")

# final dataset
probability_data_naive$prob_good <- as.numeric(as.character(probability_data_naive$prob_good))



naive_list <- NULL
final <- NULL

for(threshold in seq(from = 0.05, to = 0.95, by = 0.05)){
 probability_data_naive$predicted_class <- ifelse(probability_data_naive$prob_good > threshold,
                                                  "good", "bad")

  df2 <- probability_data_naive[,c("model", "actual_test_class", "predicted_class")]
  df2$threshold <- threshold
  df2$match <- ifelse(df2$actual_test_class == df2$predicted_class, 1, 0)

  final <- rbind(df2, final)
}

# Creating the FRP and TRP for each model and threshold
final$temp <- 1

final_summary <- final %>%
group_by(model, threshold) %>%
summarise(total_positive = sum(temp[actual_test_class == "good"]),
          total_negative = sum(temp[actual_test_class == "bad"]),
          correct_positive = sum(temp[actual_test_class == "good" & predicted_class == "good"]),
          false_positive = sum(temp[actual_test_class == "bad" & predicted_class == "good"])) %>%
  mutate(TPR = correct_positive/total_positive, FPR = false_positive/total_negative)

ggplot(data = final_summary, aes(x = FPR, y=TPR)) + geom_line(aes(colour = model)) +
  geom_abline(intercept = 0.0, slope = 1) +
  ggtitle("ROC curve for the Naive Bayes")
```



Decision trees (tree lib)

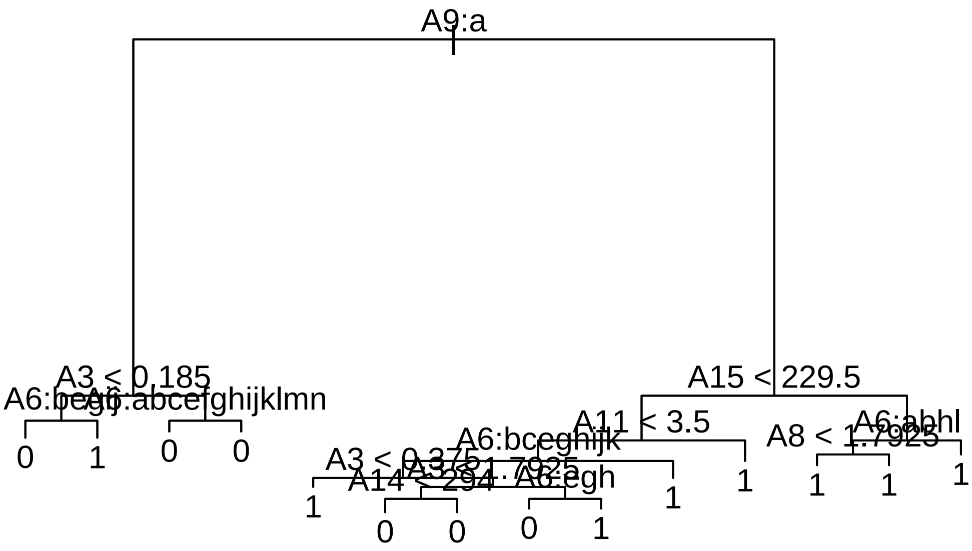
```
set.seed(12345)

data <- read.csv("crx.csv", header = TRUE)
data$Class <- as.factor(data$Class)

# 50-50 split
n=nrow(data)
id=sample(1:n, floor(n*0.8))
train=data[id,]
test=data[-id,]

tree_deviance <- tree::tree(Class~., data=train, split = c("deviance"))
tree_gini <- tree::tree(Class~., data=train, split = c("gini"))

# Visualize the decision tree with rpart.plot
summary(tree_deviance)
```

```
##
## Classification tree:
## tree::tree(formula = Class ~ ., data = train, split = c("deviance"))
## Variables actually used in tree construction:
## [1] "A9"  "A3"  "A6"  "A15" "A11" "A14" "A8"
## Number of terminal nodes:  14
## Residual mean deviance:  0.4751747 = 255.644 / 538
## Misclassification error rate: 0.09601449 = 53 / 552
```

```r
# predicting on the test dataset to get the misclassification rate.
predict_tree_deviance <- predict(tree_deviance, newdata = test, type = "class")
predict_tree_gini <- predict(tree_deviance, newdata = test, type = "class")

conf_tree_deviance <- table(test$Class, predict_tree_deviance)
names(dimnames(conf_tree_deviance)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_tree_deviance)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test  0  1
##           0 62 15
##           1  4 57
##
##                Accuracy : 0.8623188
##                  95% CI : (0.7933706, 0.9150265)
##     No Information Rate : 0.5217391
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.7260188
##  Mcnemar's Test P-Value : 0.02178146
##
##             Sensitivity : 0.9393939
##             Specificity : 0.7916667
##          Pos Pred Value : 0.8051948
##          Neg Pred Value : 0.9344262
##              Prevalence : 0.4782609
##          Detection Rate : 0.4492754
##    Detection Prevalence : 0.5579710
##       Balanced Accuracy : 0.8655303
##
##        'Positive' Class : 0
##
```

```r
# plot of the tree
plot(tree_deviance)
text(tree_deviance)
```

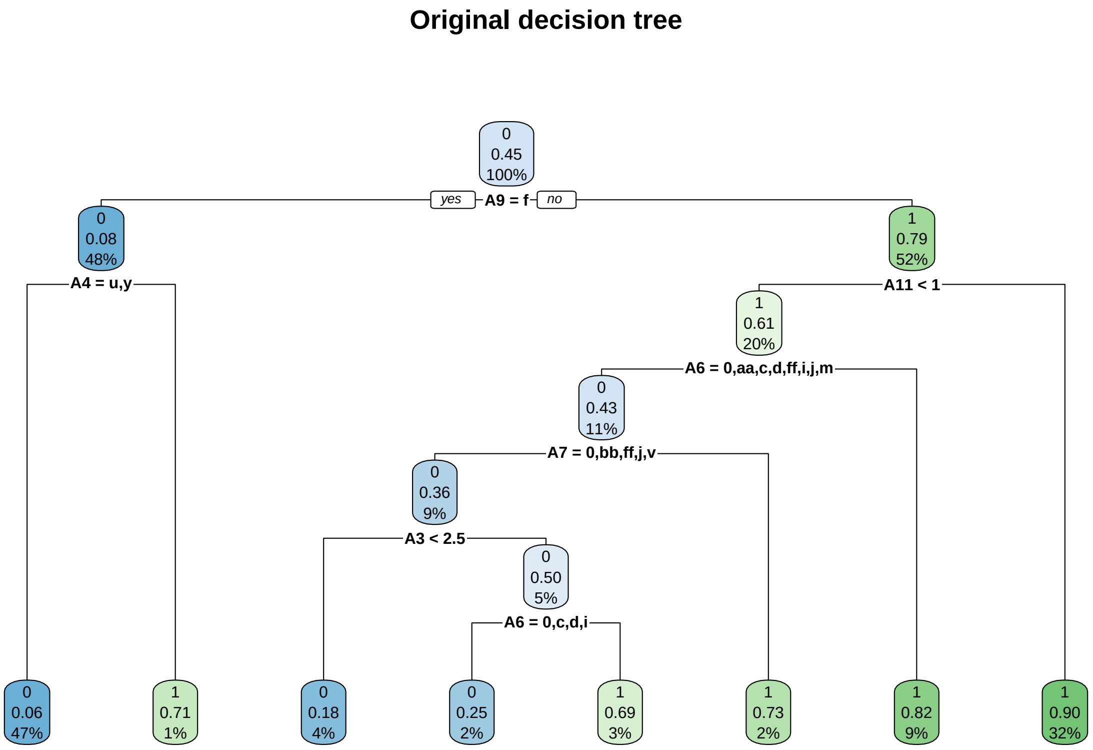


Trees using rpart

```
library(rpart.plot)
```

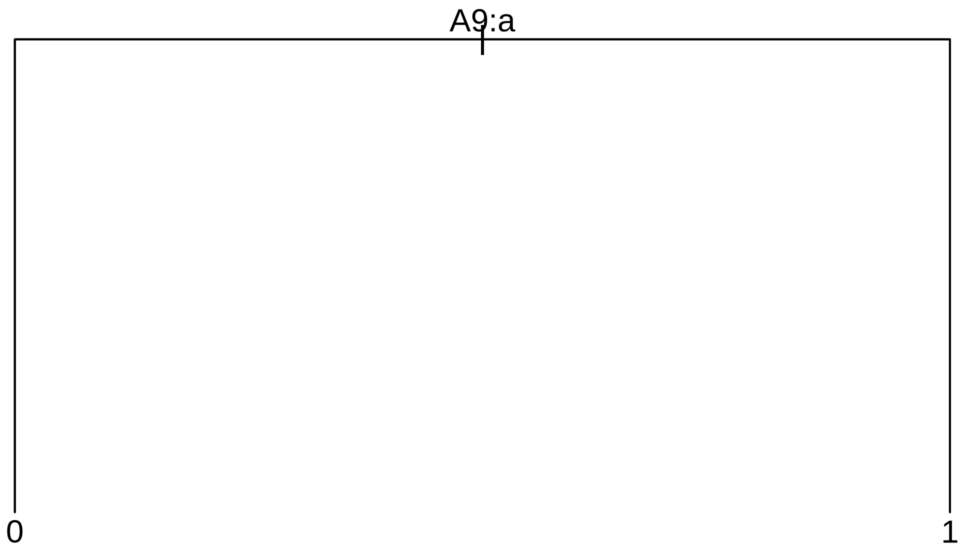
```
## Loading required package: rpart

##
## Attaching package: 'rpart'

## The following object is masked from 'package:survival':
##
##     solder
```

```
library(rpart)

set.seed(12345)


decision_tree_rpart <- rpart::rpart(data = train, formula = Class~., method = "class")
rpart.plot::rpart.plot(decision_tree_rpart, main= "Original decision tree")
```

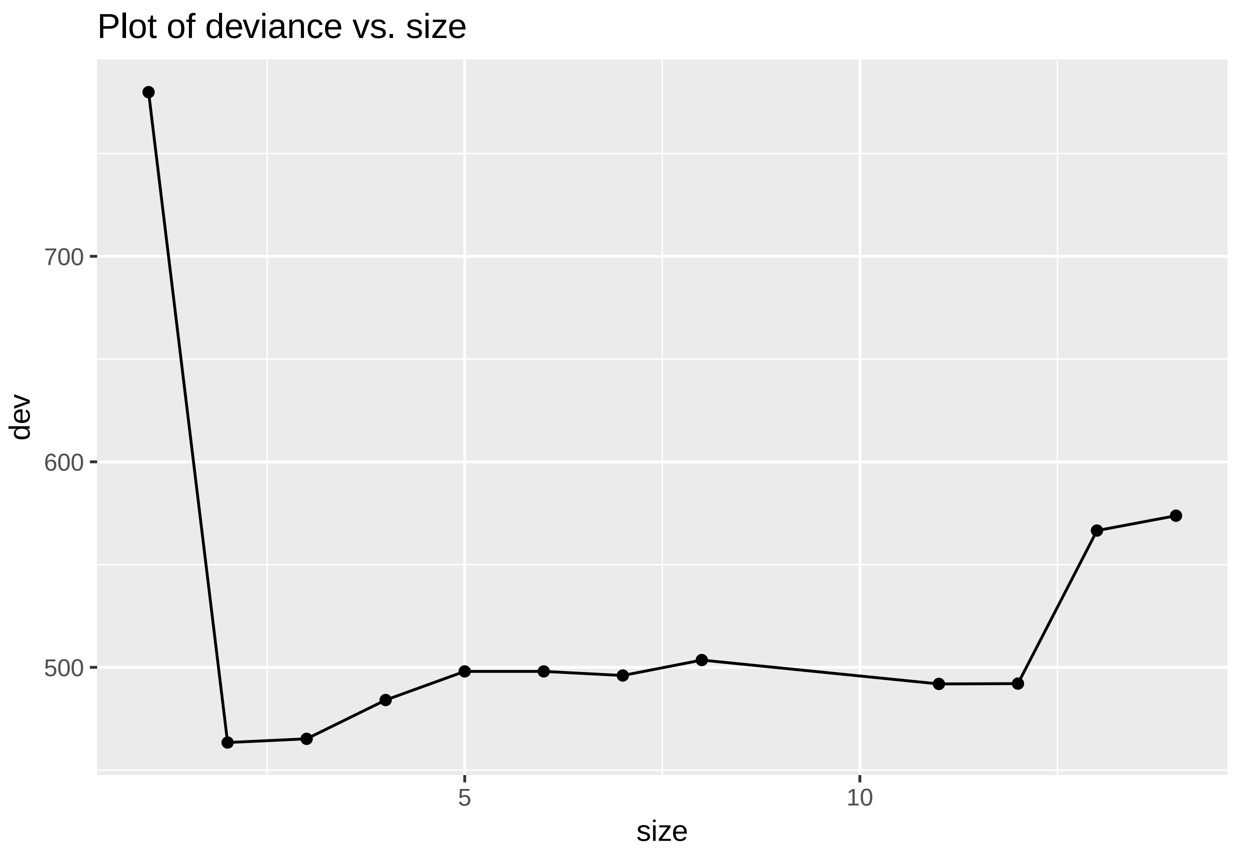


Pruning trees using cross validation

```
library(ggplot2)

set.seed(12345)
cv_tree <- cv.tree(tree_deviance, FUN = prune.tree, K = 10)
df_result <- as.data.frame(cbind(size = cv_tree$size, dev = cv_tree$dev))
# puring the tree for leaf size of 3
best_tree <- prune.tree(tree_deviance, best = 2)
plot(best_tree, main="Pruned Tree for the given dataset")
text(best_tree)
```

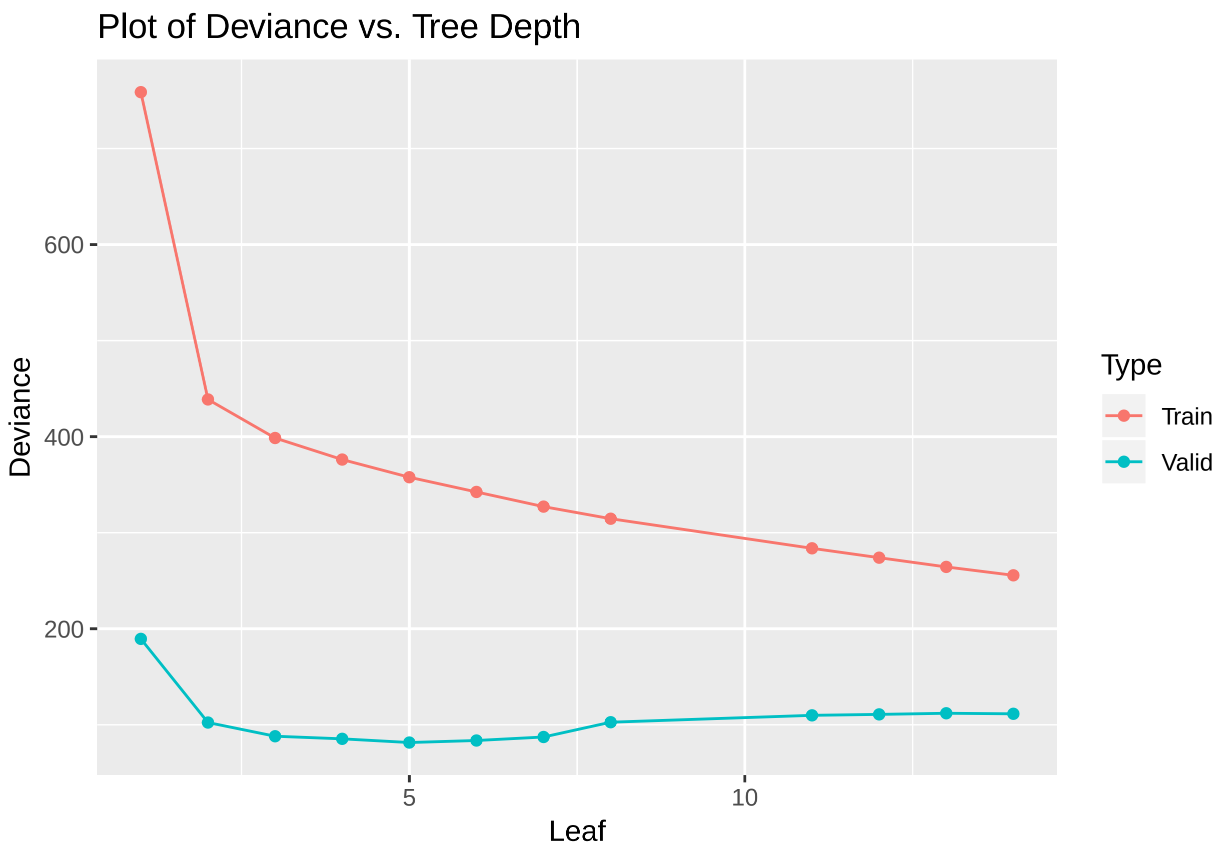


```
ggplot(df_result, aes(x = size, y = dev)) + geom_point() + geom_line() + ggtitle("Plot of deviance vs.
```



Prune the tree using error

```r
set.seed(12345)
tree_deviance <- tree::tree(Class~., data=train, split = c("deviance"))

tree_prune_train <- prune.tree(tree_deviance, method = c("deviance"))
tree_prune_valid <- prune.tree(tree_deviance, newdata = test ,method = c("deviance"))

result_train <- cbind(tree_prune_train$size,
tree_prune_train$dev, "Train")

result_valid <- cbind(tree_prune_valid$size,
tree_prune_valid$dev, "Valid")

result <- as.data.frame(rbind(result_valid, result_train))
colnames(result) <- c("Leaf", "Deviance", "Type")
result$Leaf <- as.numeric(as.character(result$Leaf))
result$Deviance <- as.numeric(as.character(result$Deviance))

# plot of deviance vs. number of leafs
ggplot(data = result, aes(x = Leaf, y = Deviance, colour = Type)) +
geom_point() + geom_line() +
ggtitle("Plot of Deviance vs. Tree Depth")
```

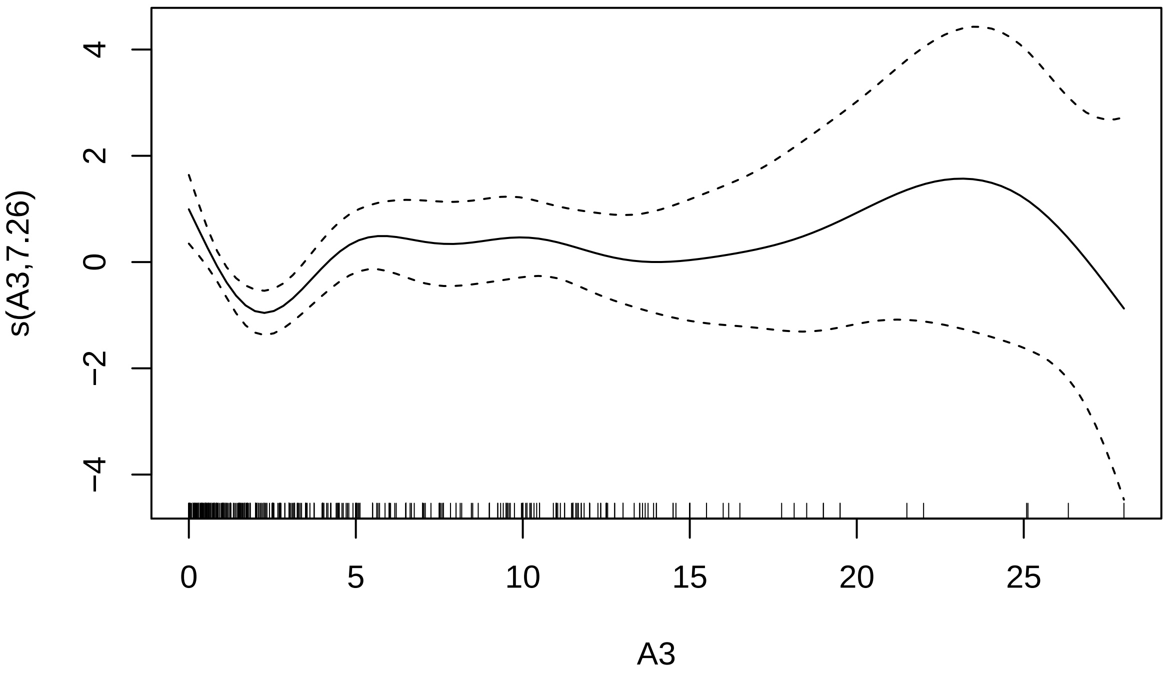


GAM Model or Spline

```
set.seed(12345)

# using family = binomial for classfication
gam_model <- mgcv::gam(data=train, formula = Class~s(A3)+A9, family=binomial)
summary(gam_model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## Class ~ s(A3) + A9
##
## Parametric coefficients:
##              Estimate Std. Error   z value              Pr(>|z|)
## (Intercept) -2.6202191  0.2479407 -10.56793 < 0.0000000000000000222 ***
## A9t          3.9741331  0.3003509  13.23163 < 0.0000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##            edf   Ref.df   Chi.sq   p-value
## s(A3) 7.263712 8.258808 22.28232 0.0056973 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.525   Deviance explained = 45.7%
## UBRE = -0.22005  Scale est. = 1         n = 552
```

```
plot(gam_model)
```



Support Vector Machine (SVM)

```
# width is the sigma here. kernel rbfdot is gaussian. vanilladot is linear
data(spam)
spam$type <- as.factor(spam$type)

## create test and training set
n=nrow(spam)
id=sample(1:n, floor(n*0.8))
spamtrain=spam[id,]
spamtest=spam[-id,]


model_0.05 <- kernlab::ksvm(type~., data=spamtrain, kernel="rbfdot", kpar=list(sigma=0.05), C=0.5)
#model_0.05

conf_model_0.05 <- table(spamtrain[,58], predict(model_0.05, spamtrain[,-58]))
```

```
names(dimnames(conf_model_0.05)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_model_0.05)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Test
## Actual Test nonspam spam
##     nonspam    2174   52
##     spam        112 1342
##
##                Accuracy : 0.9554348
##                  95% CI : (0.948261, 0.9618733)
##     No Information Rate : 0.6211957
##     P-Value [Acc > NIR] : < 0.00000000000000022204
##
##                   Kappa : 0.9060945
##  Mcnemar's Test P-Value : 0.000004082727
##
##             Sensitivity : 0.9510061
##             Specificity : 0.9626973
##          Pos Pred Value : 0.9766397
##          Neg Pred Value : 0.9229711
##              Prevalence : 0.6211957
##          Detection Rate : 0.5907609
##    Detection Prevalence : 0.6048913
##       Balanced Accuracy : 0.9568517
##
##        'Positive' Class : nonspam
##
```

ADA boost or Ensemble

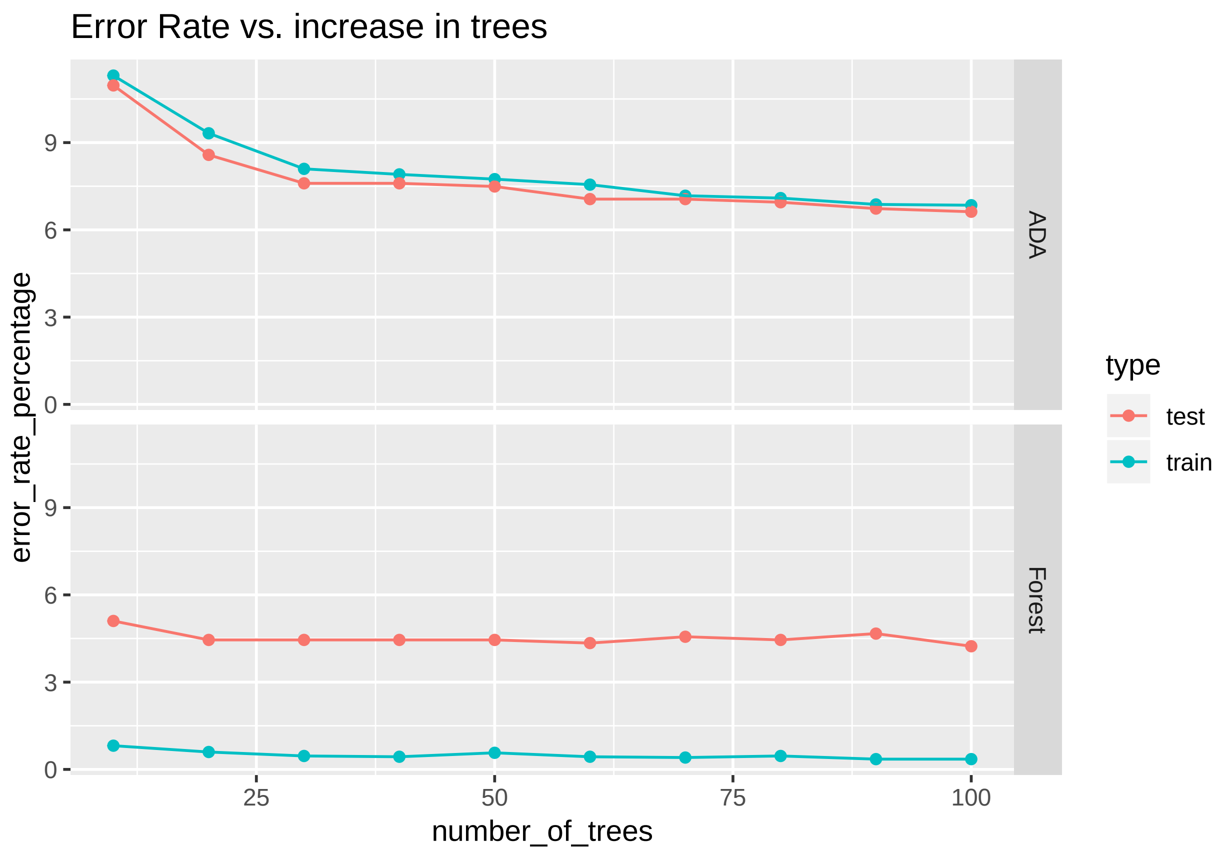
```
data(spam)
## create test and training set
n=nrow(spam)
id=sample(1:n, floor(n*0.8))
spamtrain=spam[id,]
spamtest=spam[-id,]

ada_model <- mboost::blackboost(type~., data = spamtrain, family = AdaExp(),
                                control=boost_control(mstop=15))
test_ada_model_predict <- predict(ada_model, newdata = spamtest, type = c("class"))
```

Random Forest

```
forest_model <- randomForest(type~., data = spamtrain, ntree = 15)
test_forest_model_predict <- predict(forest_model, newdata = spamtest, type = c("class"))
```

Comparing ADA boost and Randomforest

```
# using warnings = FALSE

final_result <- NULL

for(i in seq(from = 10, to = 100, by = 10)){
ada_model <- mboost::blackboost(type~.,
data = spamtrain,
family = AdaExp(),
control=boost_control(mstop=i))

forest_model <- randomForest(type~., data = spamtrain, ntree = i)

prediction_function <- function(model, data){
predicted <- predict(model, newdata = data, type = c("class"))
predict_correct <- ifelse(data$type == predicted, 1, 0)
score <- sum(predict_correct)/NROW(data)
return(score)
}

train_ada_model_predict <- predict(ada_model, newdata = spamtrain, type = c("class"))
test_ada_model_predict <- predict(ada_model, newdata = spamtest, type = c("class"))
train_forest_model_predict <- predict(forest_model, newdata = spamtrain, type = c("class"))
test_forest_model_predict <- predict(forest_model, newdata = spamtest, type = c("class"))

test_predict_correct <- ifelse(spamtest$type == test_forest_model_predict, 1, 0)
train_predict_correct <- ifelse(spamtest$type == train_forest_model_predict, 1, 0)

train_ada_score <- prediction_function(ada_model, spamtrain)
test_ada_score <- prediction_function(ada_model, spamtest)
train_forest_score <- prediction_function(forest_model, spamtrain)
test_forest_score <- prediction_function(forest_model, spamtest)

iteration_result <- data.frame(number_of_trees = i,
accuracy = c(train_ada_score,
test_ada_score,
train_forest_score,
test_forest_score),
type = c("train", "test", "train", "test"),
model = c("ADA", "ADA", "Forest", "Forest"))
final_result <- rbind(iteration_result, final_result)
}

final_result$error_rate_percentage <- 100*(1 - final_result$accuracy)

ggplot(data = final_result, aes(x = number_of_trees,
y = error_rate_percentage,
group = type, color = type)) +
geom_point() +
geom_line() +
ggtitle("Error Rate vs. increase in trees") +
facet_grid(rows = vars(model))
```



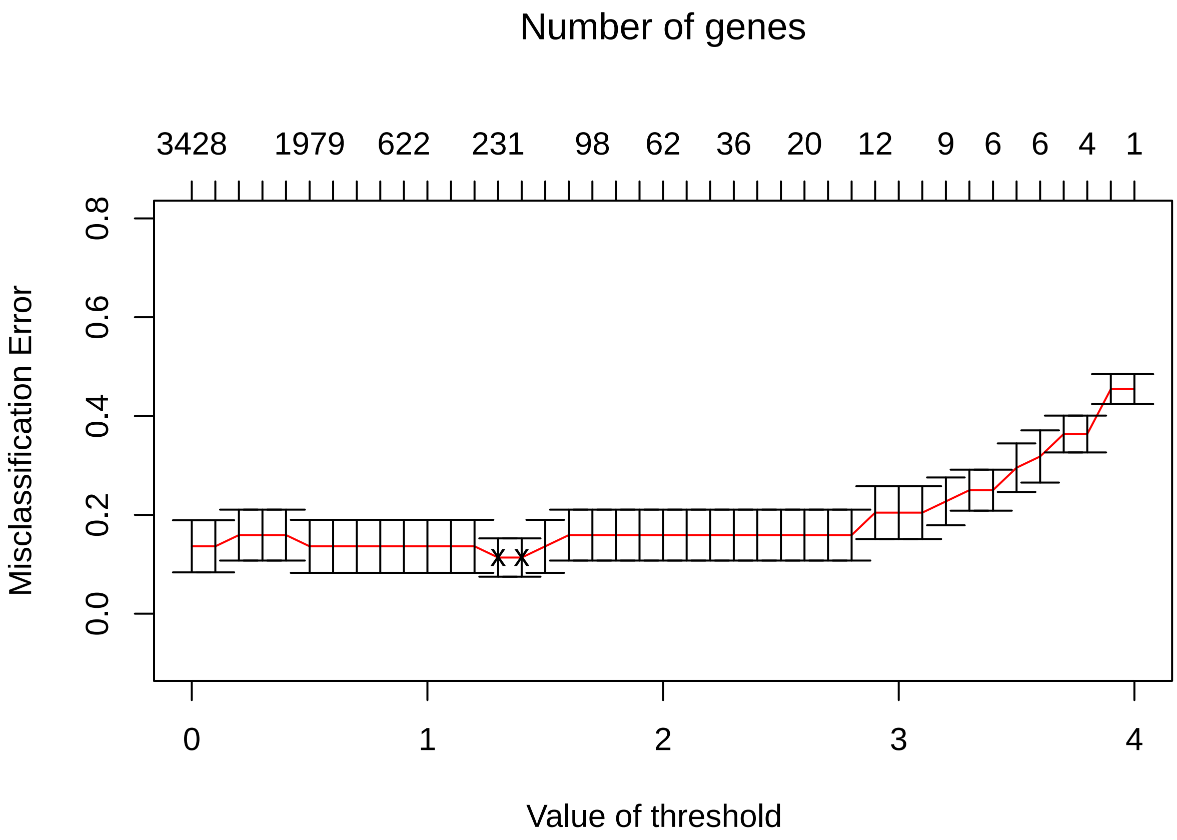
Nearest Shrunken Centroid (NSC)

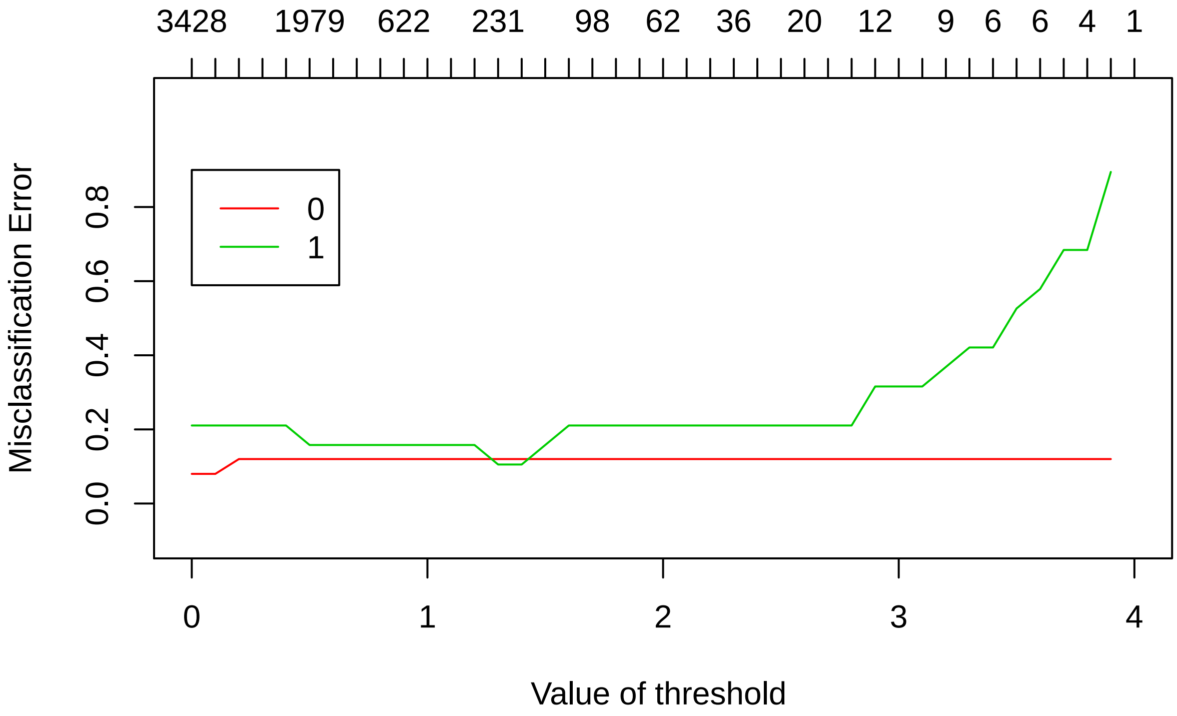
```
data <- read.csv(file = "data.csv", sep = ";", header = TRUE)
n=NROW(data)
data$Conference <- as.factor(data$Conference)


# Remember to scale the data, its cruical for this algorithm, like so scale_data = scale(data)

set.seed(12345)
id=sample(1:n, floor(n*0.7))
train=data[id,]
test = data[-id,]

rownames(train)=1:nrow(train)
x=t(train[,-4703])
y=train[[4703]]

rownames(test)=1:nrow(test)
x_test=t(test[,-4703])
y_test=test[[4703]]

mydata = list(x=x,y=as.factor(y),geneid=as.character(1:nrow(x)), genenames=rownames(x))
mydata_test = list(x=x_test,y=as.factor(y_test),geneid=as.character(1:nrow(x)), genenames=rownames(x))
```

```
model=pamr.train(mydata,threshold=seq(0, 4, 0.1))

cvmodel=pamr.cv(model, mydata)

# The value at which loglikehood is max, we can use this or use the threshold for which error is least
cvmodel$threshold[which.max(cvmodel$loglik)]


important_gen <- as.data.frame(pamr.listgenes(model, mydata, threshold = 1.3))
predicted_scc_test <- pamr.predict(model, newx = x_test, threshold = 1.3)
```

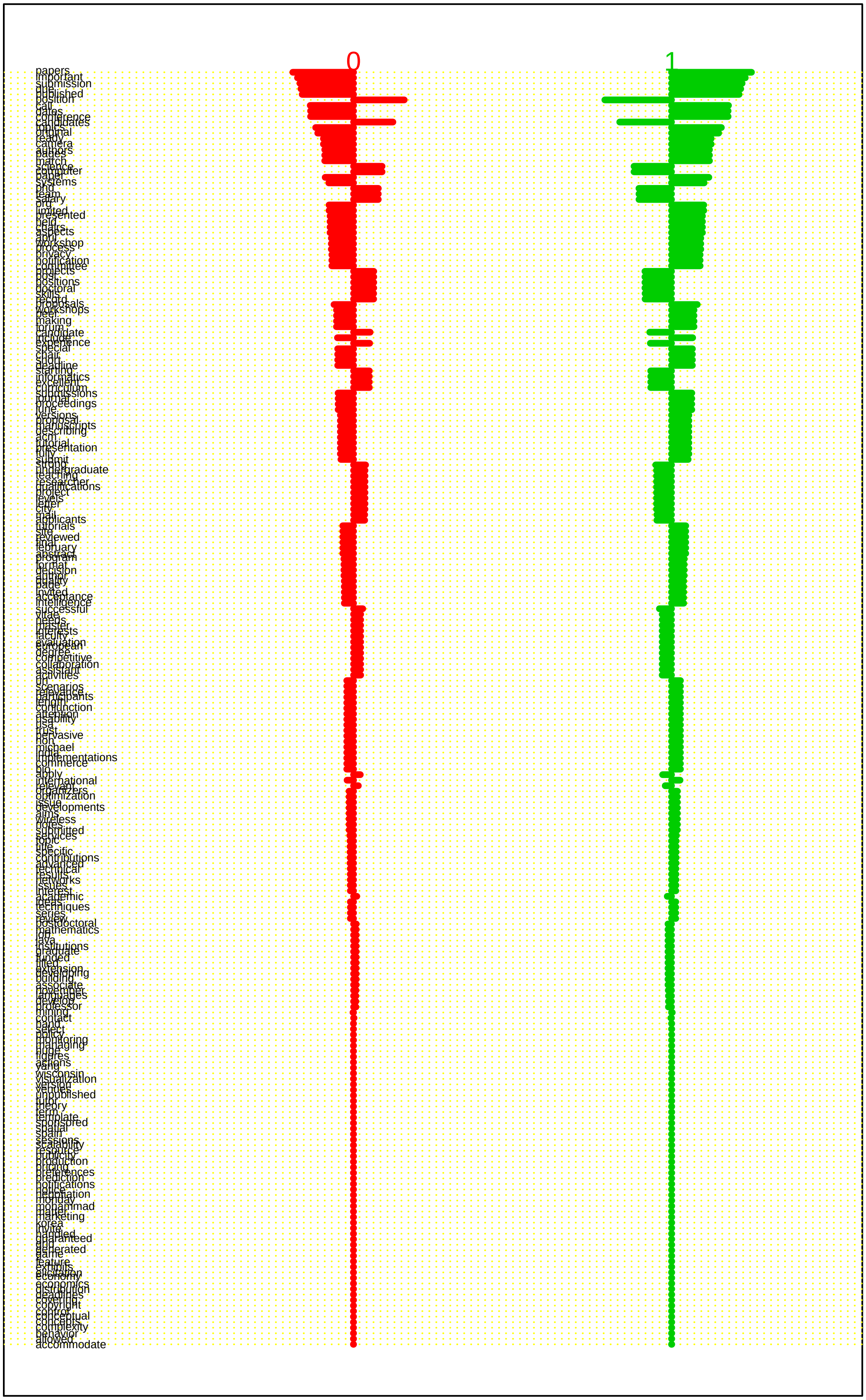
Plots

```
# use {r, fig.height=9} for better plots
pamr.plotcv(cvmodel)
```


Number of genes
3428
1979
622
231
98
62
36
20
12
9
6
6
4
1
Misclassification Error
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
Value of threshold


3428
1979
622
231
98
62
36
20
12
9
6
6
4
1
Misclassification Error
0.0
0.2
0.4
0.6
0.8
0
1
0
1
2
3
4
Value of threshold

```r
pamr.plotcen(model, mydata, threshold = 1.3)
```


0
1
papers
important
submission
due
published
position
call
dates
conference
candidates
topics
original
ready
camera
authors
pages
march
science
computer
paper
systems
phd
team
salary
org
limited
presented
held
chairs
aspects
april
workshop
process
privacy
notification
committee
projects
post
positions
doctoral
skills
record
proposals
workshops
peer
making
forum
candidate
include
experience
special
chair
short
deadline
starting
informatics
excellent
curriculum
submissions
journal
proceedings
june
versions
proposal
manuscripts
describing
acm
tutorial
presentation
fully
submit
strong
undergraduate
teaching
researcher
qualifications
project
levels
letter
city
mail
applicants
tutorials
site
reviewed
final
february
abstract
program
format
decision
author
quality
page
invited
acceptance
intelligence
successful
vitae
needs
master
interests
faculty
evaluation
european
degree
competitive
collaboration
assistant
activities
url
scenarios
relevance
participants
length
conjunction
attention
usability
usa
trust
pervasive
non
michael
india
implementations
commerce
bio
apply
international
relevant
organizers
optimization
issue
developments
aims
wireless
notes
submitted
services
topic
title
specific
contributions
advanced
technical
results
networks
issues
interest
academic
ideas
techniques
series
review
postdoctoral
mathematics
job
java
institutions
graduate
funded
filled
extension
developing
building
associate
november
languages
develop
professor
mining
contact
hand
select
policy
monitoring
managing
huge
figures
actions
yang
wisconsin
visualization
version
venues
unpublished
tutor
theory
term
template
sponsored
spatial
spain
sessions
scalability
resource
publicity
production
pricing
preferences
prediction
notifications
notice
negotiation
monday
mohammad
matter
marketing
korea
invite
handled
guaranteed
grid
generated
game
feature
exhibits
elicitation
economy
economics
distribution
deadlines
covering
copyright
control
conceptual
concepts
complexity
behavior
allowed
accommodate

Important features

```
## List the significant genes
NROW(important_gen)
```

```
## [1] 231
```

```
temp <- colnames(data) %>% as.data.frame()
colnames(temp) <- "col_name"
temp$index <- row.names(temp)

df <- merge(x = important_gen, y = temp, by.x = "id", by.y = "index", all.x = TRUE)
df <- df[order(df[,3], decreasing = TRUE ),]

knitr::kable(head(df[,4],10), caption = "Important feaures selected by Nearest Shrunken Centroids ")
```

Table 1: Important feaures selected by Nearest Shrunken Centroids

x
papers
important
submission
due
published
call
dates
conference
topics
original

Confusion table

```
conf_scc <- table(y_test, predicted_scc_test)
names(dimnames(conf_scc)) <- c("Actual Test", "Predicted Srunken Centroid Test")
result_scc <- caret::confusionMatrix(conf_scc)
caret::confusionMatrix(conf_scc)
```

```
## Confusion Matrix and Statistics
##
##            Predicted Srunken Centroid Test
## Actual Test  0  1
##           0 10  0
##           1  2  8
##
##                Accuracy : 0.9
##                  95% CI : (0.6830173, 0.9876515)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.003611472
##
##                   Kappa : 0.8
##  Mcnemar's Test P-Value : 0.479500122
##
```

```
##             Sensitivity : 0.8333333
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.8000000
##              Prevalence : 0.6000000
##          Detection Rate : 0.5000000
##    Detection Prevalence : 0.5000000
##       Balanced Accuracy : 0.9166667
##
##        'Positive' Class : 0
##
```

Elastic Net

```
x = train[,-4703] %>% as.matrix()
y = train[,4703]

x_test = test[,-4703] %>% as.matrix()
y_test = test[,4703]

cvfit = cv.glmnet(x=x, y=y, alpha = 0.5, family =   "binomial")
predicted_elastic_test <- predict.cv.glmnet(cvfit, newx = x_test, s = "lambda.min", type = "class")
tmp_coeffs <- coef(cvfit, s = "lambda.min")
elastic_variable <- data.frame(name = tmp_coeffs@Dimnames[[1]][tmp_coeffs@i + 1], coefficient = tmp_coe
knitr::kable(elastic_variable, caption = "Contributing features in the elastic model")
```

Table 2: Contributing features in the elastic model

name	coefficient
(Intercept)	-1.0189312955
abstracts	-0.3011264328
aspects	0.0736775805
bio	0.0228765136
call	0.3319900155
candidates	-0.1878310774
computer	-0.2832064906
conceptual	0.0380843574
conference	0.1965329661
dates	0.2416630036
due	0.5211724945
evaluation	-0.1796400822
exhibits	0.3782699866
important	0.3924275218
languages	-0.0258469943
making	0.1892393673
manuscripts	0.0325584417
original	0.0558204697
papers	0.3853809791
peer	0.0967211078
position	-0.3750829937
process	0.0016238373
projects	-0.1904079978

name	coefficient
proposals	0.0553553768
published	0.2818205886
queries	-0.3002458792
record	-0.1162514000
relevant	-0.1135564059
scenarios	0.0053469502
spatial	0.1925006829
submission	0.2803519351
team	-0.1291277610
versions	0.1545749085

```
conf_elastic_net <- table(y_test, predicted_elastic_test)
names(dimnames(conf_elastic_net)) <- c("Actual Test", "Predicted ElasticNet Test")
result_elastic_net <- caret::confusionMatrix(conf_elastic_net)
caret::confusionMatrix(conf_elastic_net)
```

```
## Confusion Matrix and Statistics
##
##            Predicted ElasticNet Test
## Actual Test  0  1
##           0 10  0
##           1  2  8
##
##                Accuracy : 0.9
##                  95% CI : (0.6830173, 0.9876515)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.003611472
##
##                   Kappa : 0.8
##  Mcnemar's Test P-Value : 0.479500122
##
##             Sensitivity : 0.8333333
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.8000000
##              Prevalence : 0.6000000
##          Detection Rate : 0.5000000
##    Detection Prevalence : 0.5000000
##       Balanced Accuracy : 0.9166667
##
##        'Positive' Class : 0
##
```

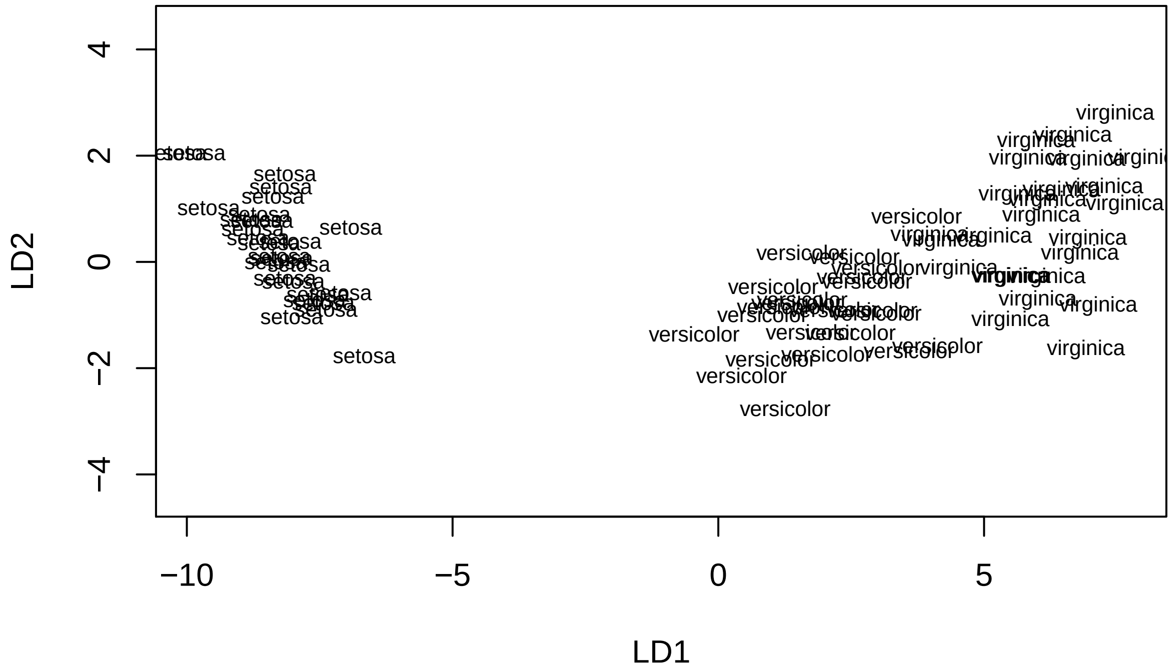
Linear discriminant analysis (LDA)

```
# Load the data
data <- iris
# Split the data into training (80%) and test set (20%)
n=NROW(data)
set.seed(12345)
```

```
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

model <- MASS::lda(Species~., data = train)
model
```

```
## Call:
## lda(Species ~ ., data = train)
##
## Prior probabilities of groups:
##       setosa   versicolor    virginica
## 0.3600000000 0.3066666667 0.3333333333
##
## Group means:
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa      5.007407407 3.425925926  1.481481481 0.237037037
## versicolor  5.986956522 2.752173913  4.317391304 1.330434783
## virginica   6.672000000 3.064000000  5.592000000 2.060000000
##
## Coefficients of linear discriminants:
##                        LD1           LD2
## Sepal.Length -0.4600006716  0.4122667884
## Sepal.Width  -1.4530508928  2.0465774457
## Petal.Length  2.5357940400 -1.4106420572
## Petal.Width   2.3130581207  3.4312098721
##
## Proportion of trace:
##    LD1    LD2
## 0.9882 0.0118
```

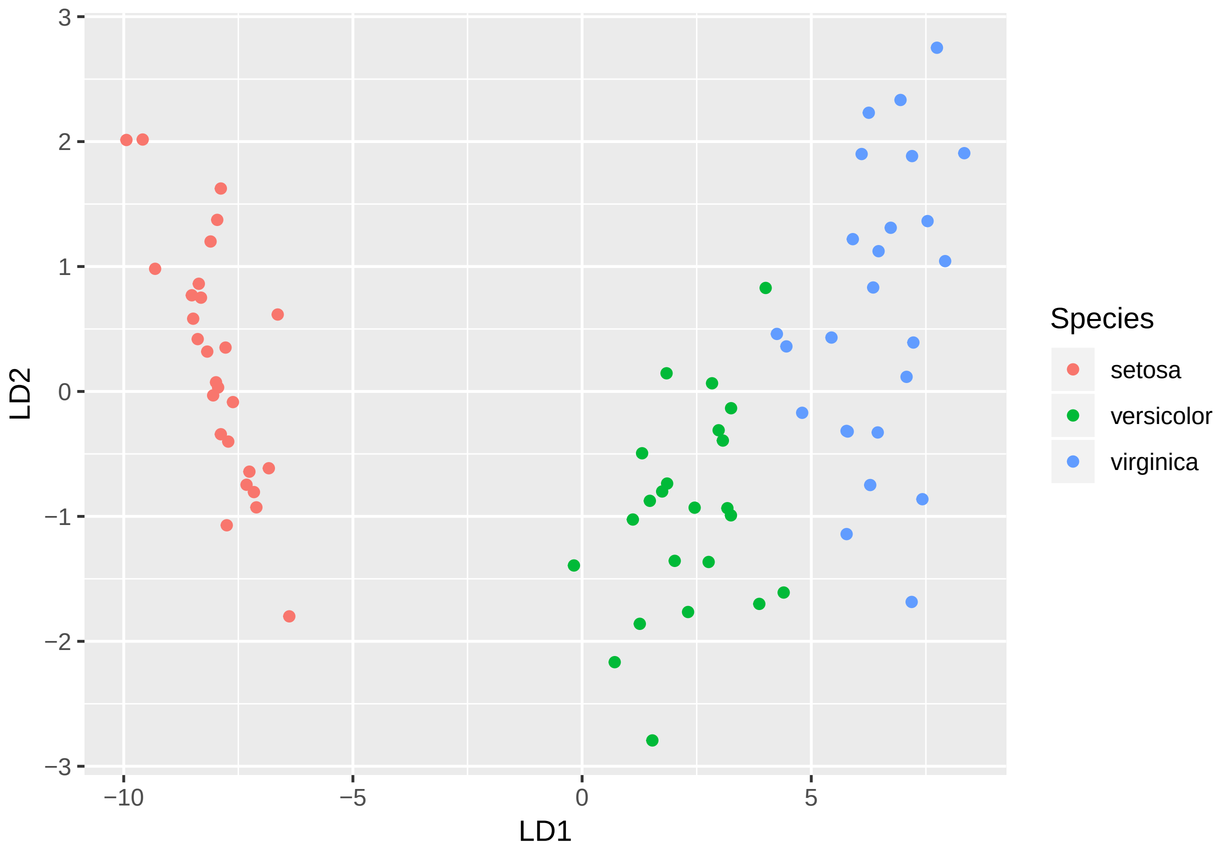
```
plot(model)
```



```r
predictions <- model %>% predict(test)
names(predictions)
```

```
## [1] "class"     "posterior" "x"
```

```r
lda.data <- cbind(train, predict(model)$x)
#plot(data2$frames,data2$duration, col=predict(m3)$class)

ggplot(lda.data, aes(LD1, LD2)) + geom_point(aes(color = Species))
```



Bootstrap and Big data

Principle Component Analysis

Components

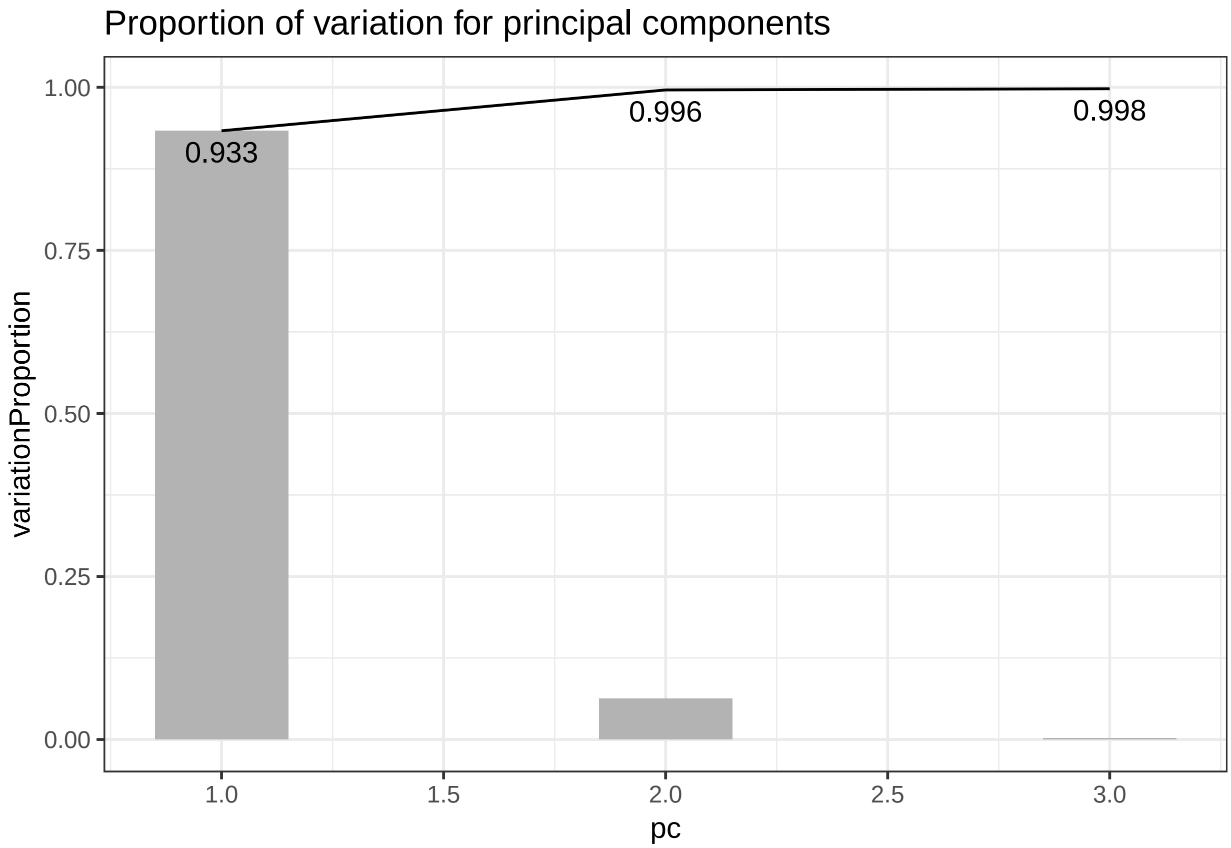
```
rm(list=ls())

set.seed(12345)
NIR_data <- read.csv2("NIRSpectra.csv")

## scaling is necessary else the column with high range will dominate, using prcomp(scale=TRUE)
pca_data =  select(NIR_data,-c(Viscosity))
pca_result = prcomp(pca_data)

contribution <- summary(pca_result)$importance
knitr::kable(contribution[,1:5],
             caption = "Contribution of PCA axis towards varience explaination")
```

Table 3: Contribution of PCA axis towards varience explaination

	PC1	PC2	PC3	PC4	PC5
Standard deviation	0.1220620179	0.0316204764	0.0054352509	0.0040106511	0.0033031459

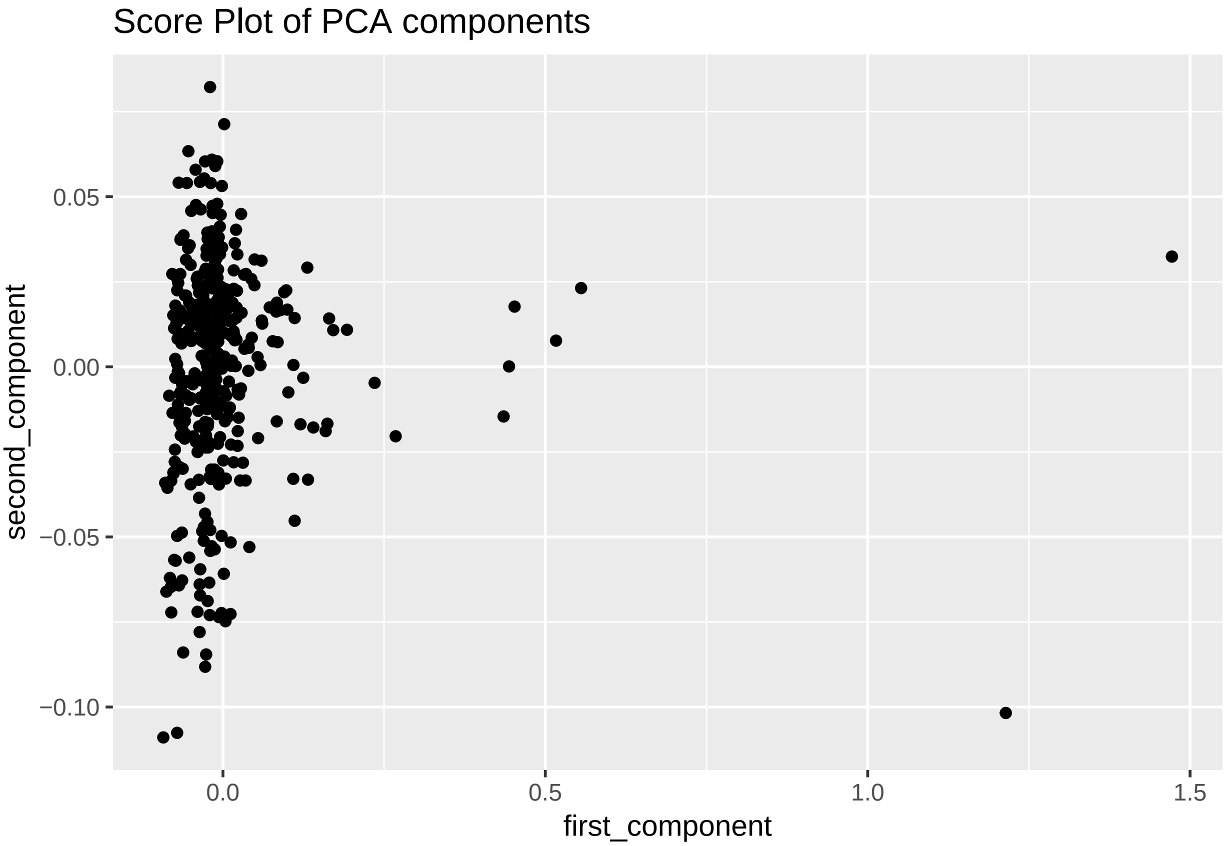
	PC1	PC2	PC3	PC4	PC5
Proportion of Variance	0.9333200000	0.0626300000	0.0018500000	0.0010100000	0.0006800000
Cumulative Proportion	0.9333200000	0.9959600000	0.9978100000	0.9988200000	0.9995000000

```
eigenvalues = pca_result$sdev^2

# plotting proportion of variation for principal components
plotData = as.data.frame(cbind(pc = 1:3,
variationProportion = eigenvalues[1:3]/sum(eigenvalues),
cummulative = cumsum(eigenvalues[1:3]/sum(eigenvalues))))

ggplot(data = plotData) +
geom_col(aes(x = pc, y = variationProportion), width = 0.3, fill = "grey70") +
geom_line(data = plotData,
aes(x = pc, y = cummulative)) +
geom_text(aes(x = pc, y = cummulative, label = round(cummulative, 3)), size = 4,
position = "identity", vjust = 1.5) +
theme_bw() +
ggtitle("Proportion of variation for principal components")
```

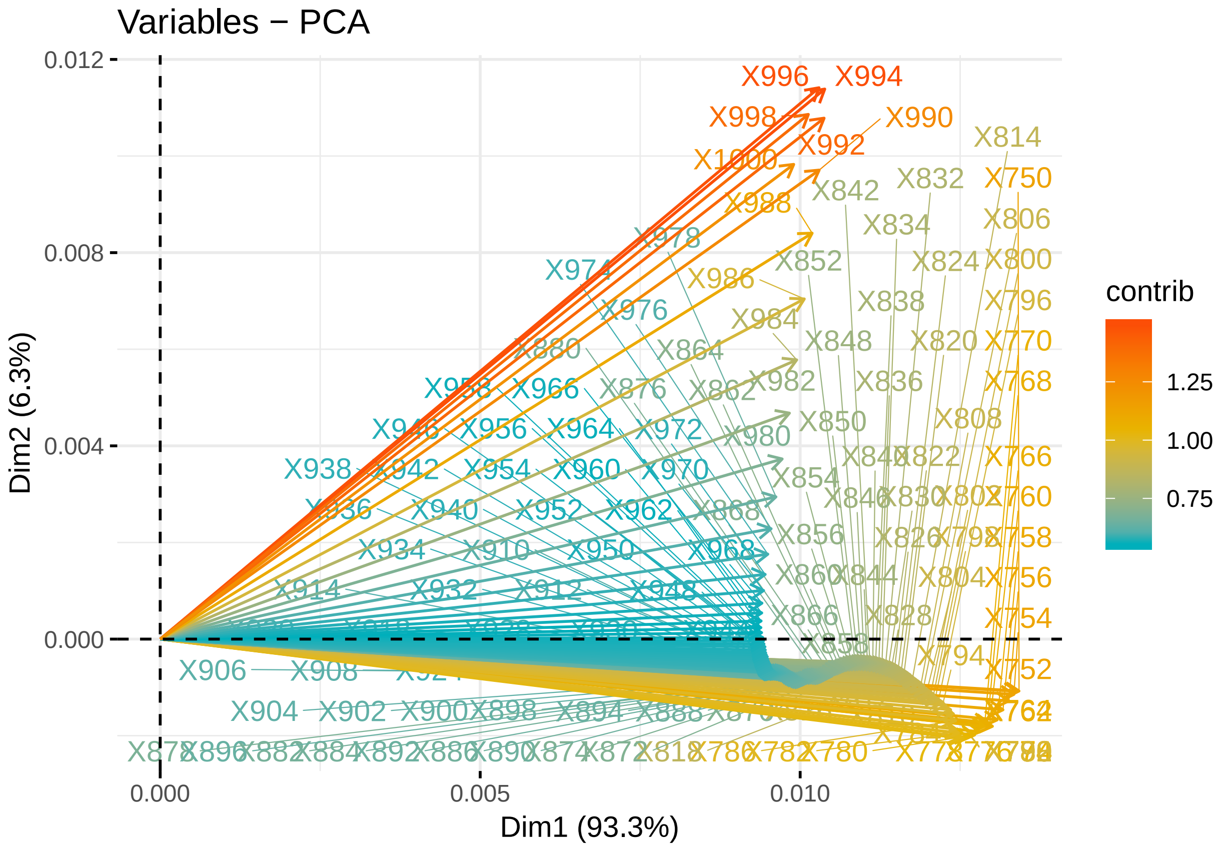


```
# pca components and the viscocity
pca_result_data = cbind(first_component = pca_result$x[,1],
                                second_component = pca_result$x[,2]) %>% as.data.frame()

# plotting the data variation and the viscocity
```

```r
ggplot(data = pca_result_data, aes(x = first_component, y = second_component)) +
  geom_point() + ggtitle("Score Plot of PCA components")
```

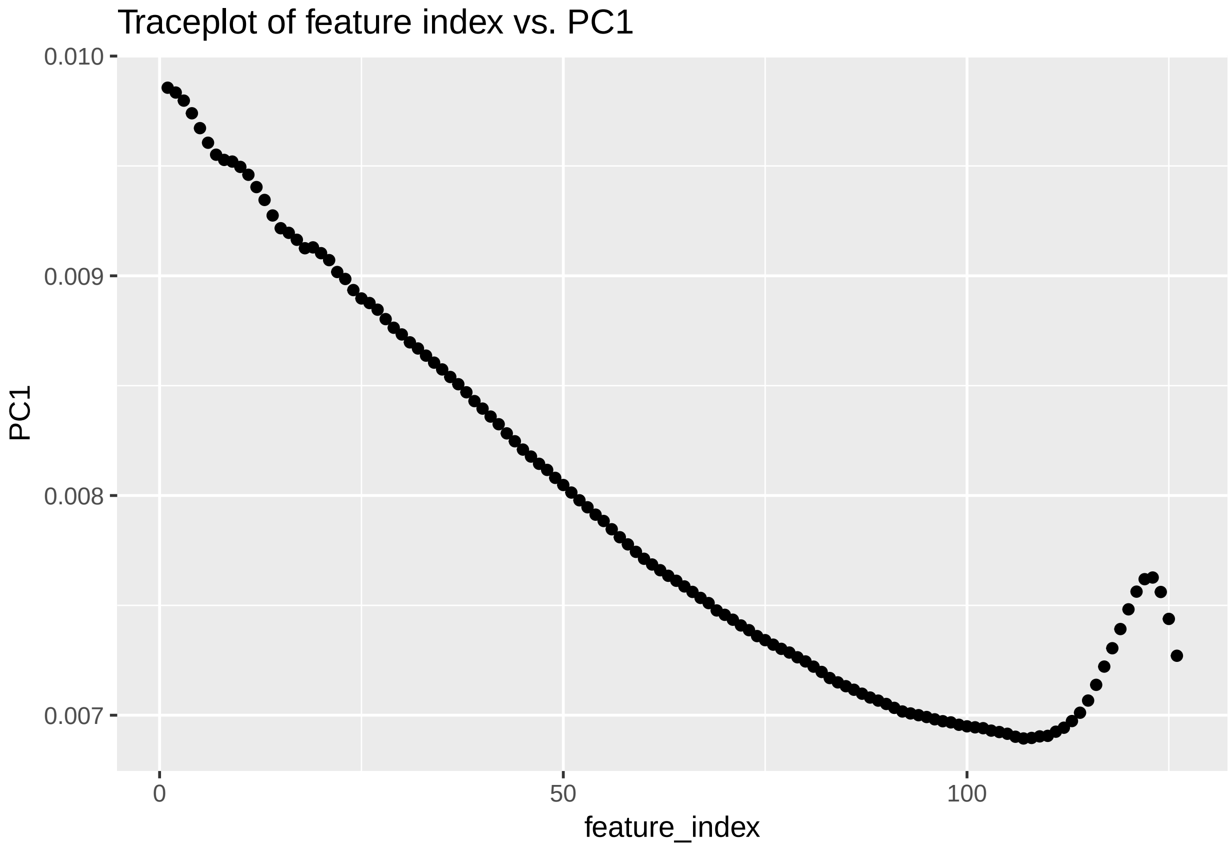


```r
# showing the score of PCA component
factoextra::fviz_pca_var(pca_result,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
             )
```

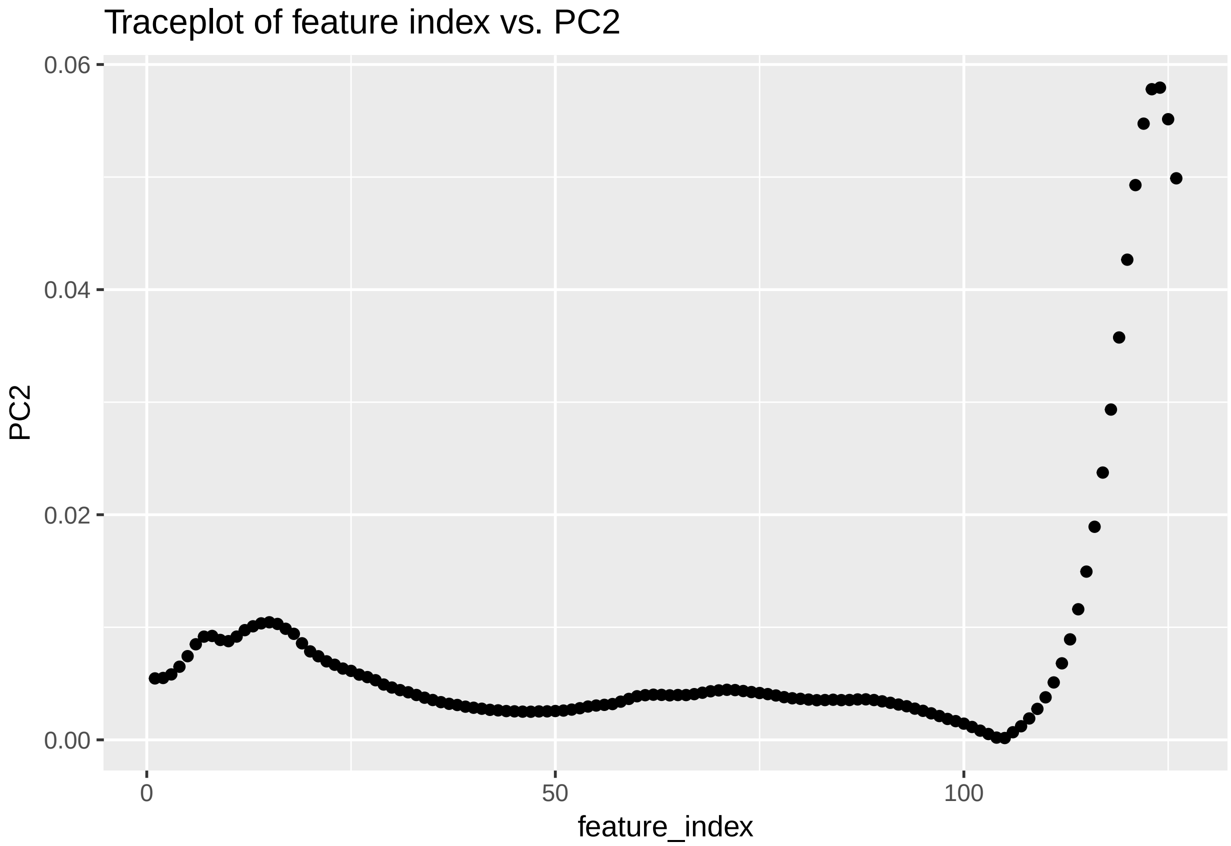


Trace plots of PCA

```r
set.seed(12345)

# creating extra columns
aload <- abs(pca_result$rotation[,1:2])
components <- sweep(aload, 2, colSums(aload), "/")
components <- as.data.frame(components)
components$feature_name <- rownames(components)
components$feature_index <- 1:nrow(components)

ggplot(data = components, aes(x = feature_index, y = PC1)) +
  geom_point() +
  ggtitle("Traceplot of feature index vs. PC1")
```



```r
ggplot(data = components, aes(x = feature_index, y = PC2)) +
  geom_point() +
  ggtitle("Traceplot of feature index vs. PC2")
```



```r
knitr::kable(components[1:10,],
             caption = "Contribution of Features towards the Principle Components")
```

Table 4: Contribution of Features towards the Principle Components

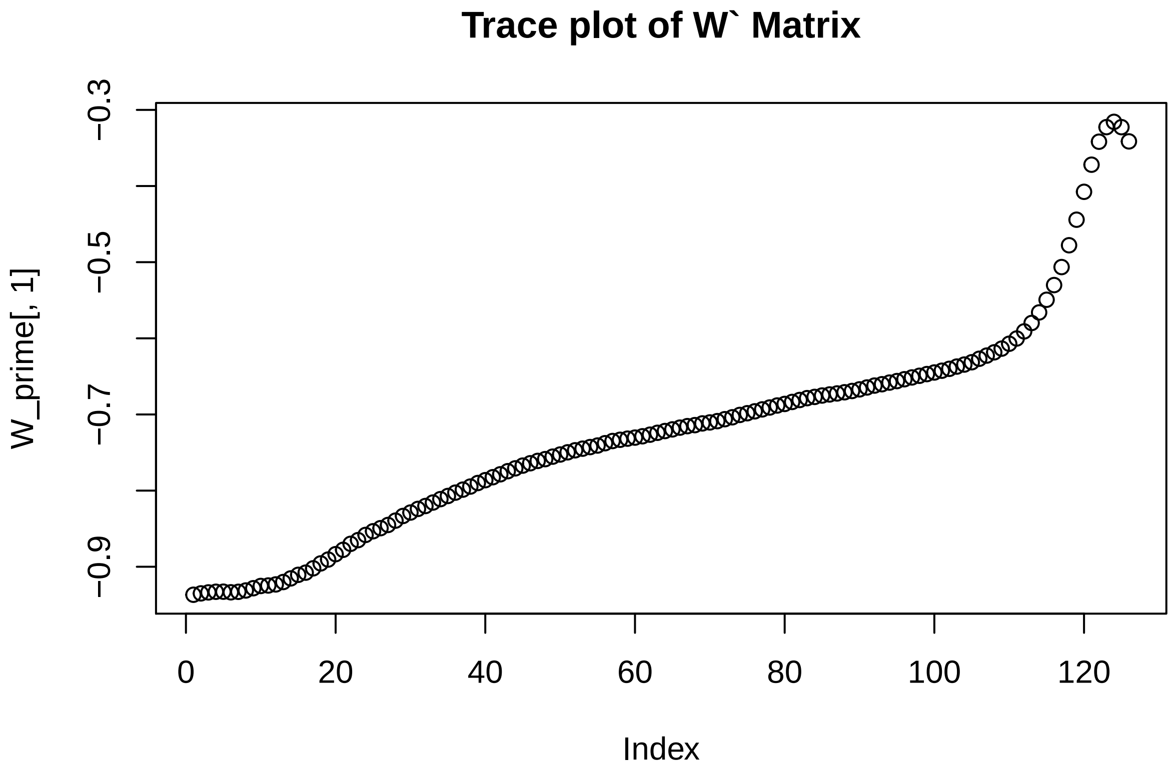
	PC1	PC2	feature_name	feature_index
X750	0.0098560469	0.0054568310	X750	1
X752	0.0098341197	0.0054945194	X752	2
X754	0.0097973028	0.0058118017	X754	3
X756	0.0097396794	0.0064942074	X756	4
X758	0.0096720361	0.0074337784	X758	5
X760	0.0096057136	0.0084895814	X760	6
X762	0.0095511383	0.0091662055	X762	7
X764	0.0095274543	0.0092284228	X764	8
X766	0.0095199165	0.0088732508	X766	9
X768	0.0094956043	0.0087673975	X768	10

FastICA

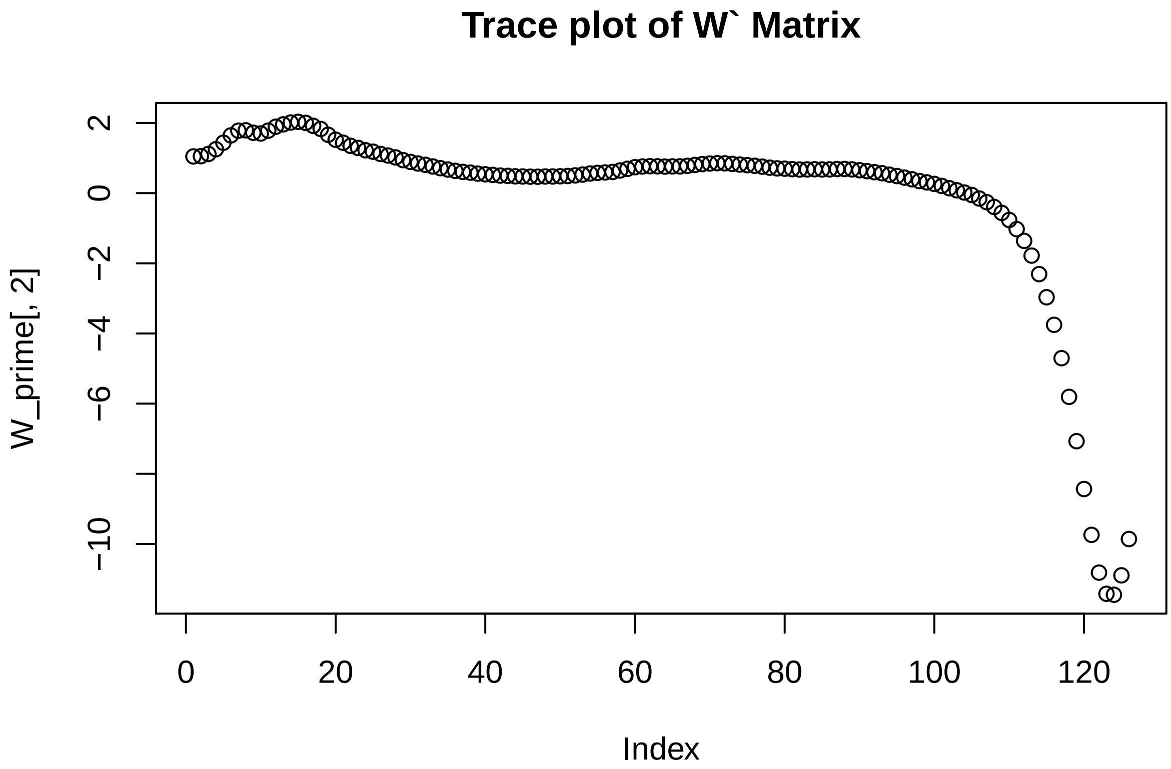
```r
library(fastICA)

set.seed(12345)

# X -> pre-processed data matrix
# K -> pre-whitening matrix that projects data onto the first n.compprincipal components.
# W -> estimated un-mixing matrix (see definition in details)
# A -> estimated mixing matrix
# S -> estimated source matrix

X <- as.matrix(pca_data)

ICA_extraction <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
method = "R", row.norm = FALSE, maxit = 200,
tol = 0.0001, verbose = TRUE)
```

```
## Centering

## Whitening

## Symmetric FastICA using logcosh approx. to neg-entropy function

## Iteration 1 tol = 0.01930238856

## Iteration 2 tol = 0.01303958688

## Iteration 3 tol = 0.002393581972

## Iteration 4 tol = 0.0006708453596

## Iteration 5 tol = 0.0001661601504

## Iteration 6 tol = 0.00003521604149
```

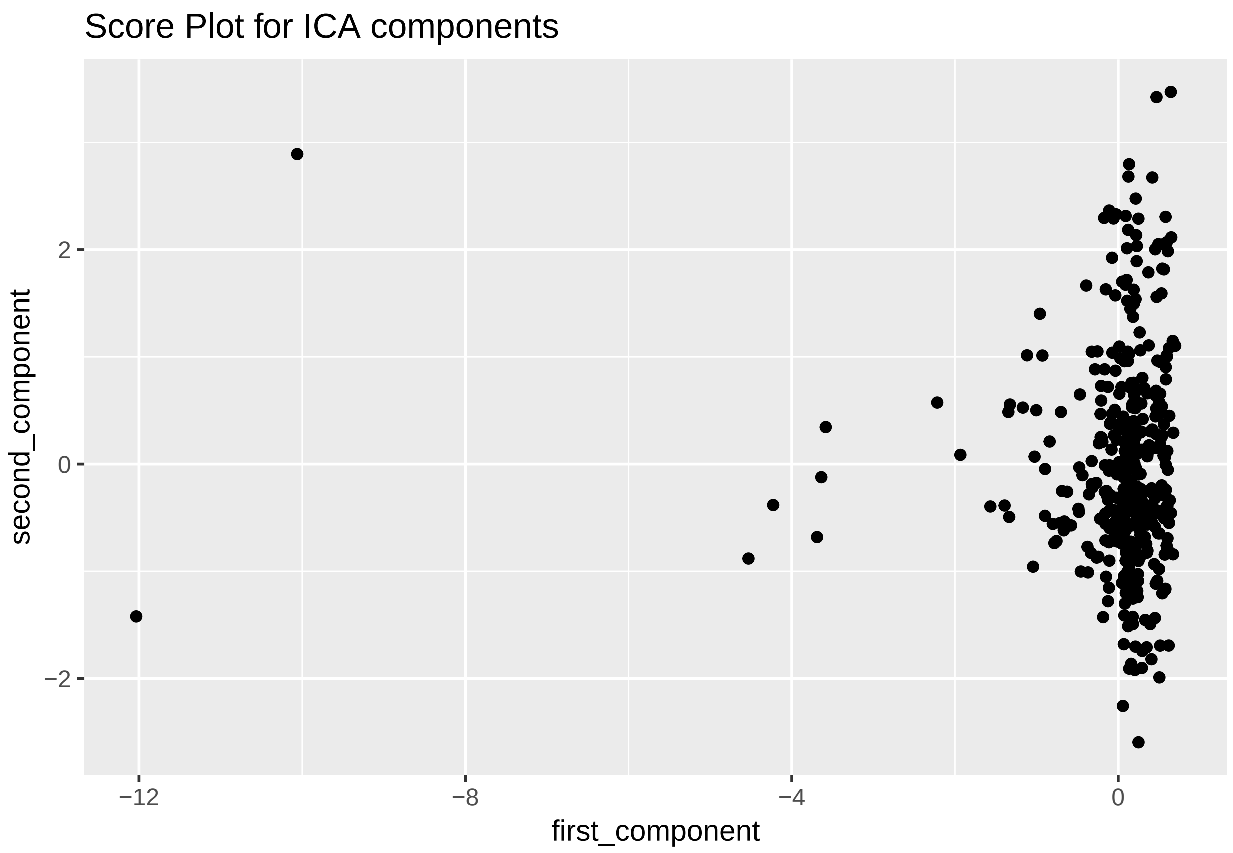
```r
W_prime <- ICA_extraction$K %*% ICA_extraction$W

#trace plots
plot(W_prime[,1], main = "Trace plot of W` Matrix")
```



```
#trace plots
plot(W_prime[,2], main = "Trace plot of W` Matrix")
```



```
# pca components and the viscocity
ICA_result_data = cbind(first_component = ICA_extraction$S[,1],
                        second_component = ICA_extraction$S[,2],
                        Viscosity = NIR_data$Viscosity) %>% as.data.frame()

# plotting the data variation
ggplot(data = ICA_result_data, aes(x = first_component, y = second_component)) +
  geom_point() + ggtitle("Score Plot for ICA components")
```



Implement Benjamini-Hochberg method

```
data <- read.csv(file = "data.csv", sep = ";", header = TRUE)
data$Conference <- as.factor(data$Conference)
set.seed(12345)
y <- as.factor(data[,4703])
x <- as.matrix(data[,-4703])
p_values <- data.frame(feature = '',P_value = 0,stringsAsFactors = FALSE)
for(i in 1:ncol(x)){
res = t.test(x[,i]~y, data = data,
alternative="two.sided"
,conf.level = 0.95)
p_values[i,] <- c(colnames(x)[i],res$p.value)
}
p_values$P_value <- as.numeric(p_values$P_value)
p <- p.adjust(p_values$P_value, method = 'BH')
length(p[which(p > 0.05)])
```

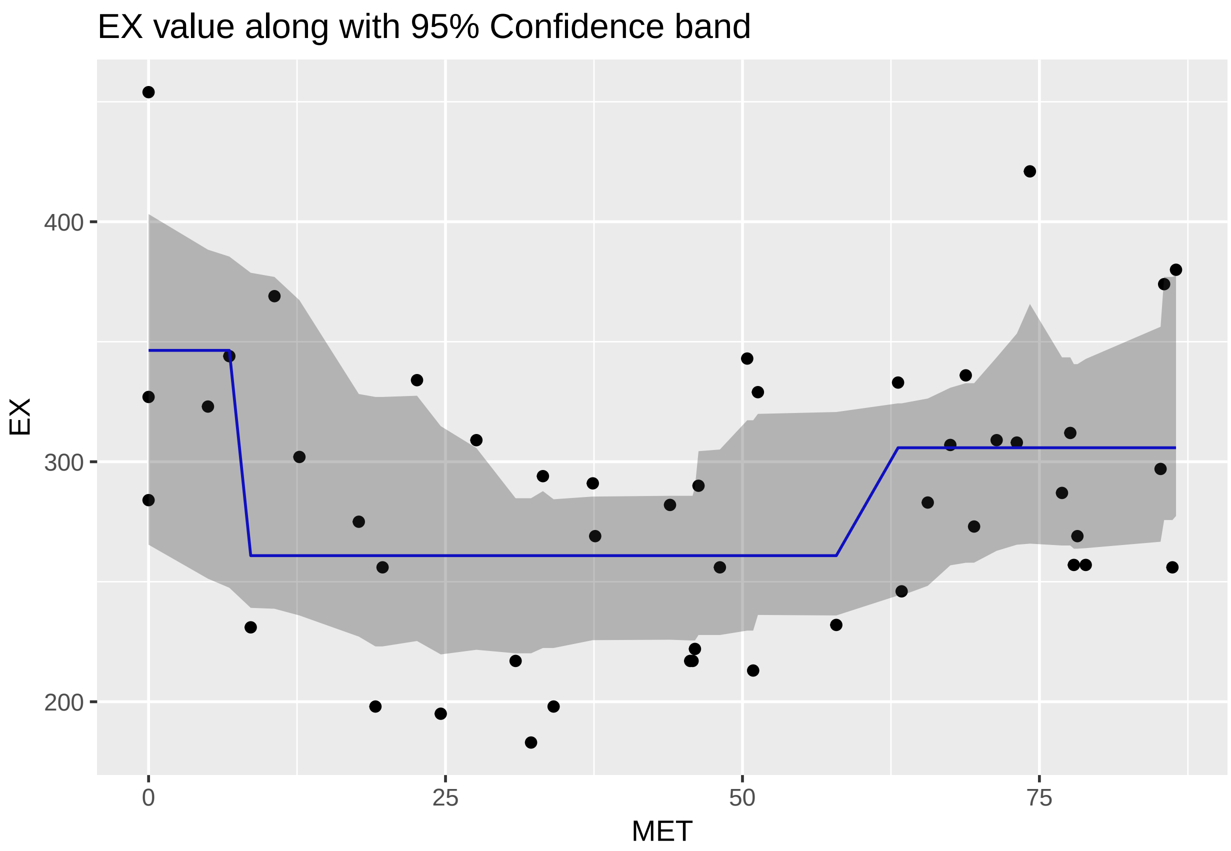
```
## [1] 4663
```

```
out <- p_values[which(p <= 0.05),]
out <- out[order(out$P_value),]
rownames(out) <- NULL
out
```

```
##           feature               P_value
## 1          papers 0.0000000001116909797
## 2      submission 0.0000000007949968929
## 3        position 0.0000000082193623640
## 4       published 0.0000001835157279467
## 5       important 0.0000003040833458119
## 6            call 0.0000003983539629261
## 7      conference 0.0000005091969773010
## 8      candidates 0.0000008612259484871
## 9           dates 0.0000013986185738606
## 10          paper 0.0000013986185738606
## 11         topics 0.0000050683729682040
## 12        limited 0.0000079079758951488
## 13      candidate 0.0000119060734289307
## 14         camera 0.0000209911877899371
## 15          ready 0.0000209911877899371
## 16        authors 0.0000215446089370647
## 17            phd 0.0000338267054292409
## 18       projects 0.0000349912277550768
## 19            org 0.0000374201040256446
## 20         chairs 0.0000586017469952769
## 21            due 0.0000648878090910497
## 22       original 0.0000648878090910497
## 23   notification 0.0000688221014991065
## 24         salary 0.0000797198143095279
## 25         record 0.0000909003772803830
## 26         skills 0.0000909003772803830
## 27           held 0.0001529174143198030
## 28           team 0.0001757570093013810
## 29          pages 0.0002007352997295650
## 30       workshop 0.0002007352997295650
## 31      committee 0.0002117019606737420
## 32    proceedings 0.0002117019606737420
## 33          apply 0.0002166413777846920
## 34         strong 0.0002246309035177580
## 35  international 0.0002295683995558020
## 36         degree 0.0003762328270248580
## 37      excellent 0.0003762328270248580
## 38           post 0.0003762328270248580
## 39      presented 0.0003765147311797180
```

Confidence band using Bootstrap

```r
library(boot)
set.seed(12345)
state_data <- read.csv2("state.csv")

# computing bootstrap samples
bootstrap <- function(data, indices){
  data <- state_data[indices,]

  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, best = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  return(final_fit_boot)
}

res <- boot(state_data, bootstrap, R=1000) #make bootstrap
e <- envelope(res, level = 0.95)
state_tree_regression <- tree(data = state_data, EX~MET,
                              control = tree.control(nobs=NROW(state_data),
                                                     minsize = 8))

# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, best = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)
data_for_ci <- cbind(upper_bound = e$point[1,],
                     lower_bound = e$point[2,],
                     EX = state_data$EX,
                     MET = state_data$MET,
                     predicted_value = predict_for_ci) %>% as.data.frame()

#plot cofidence bands

ggplot(data=data_for_ci, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
  ggtitle("EX value along with 95% Confidence band")
```

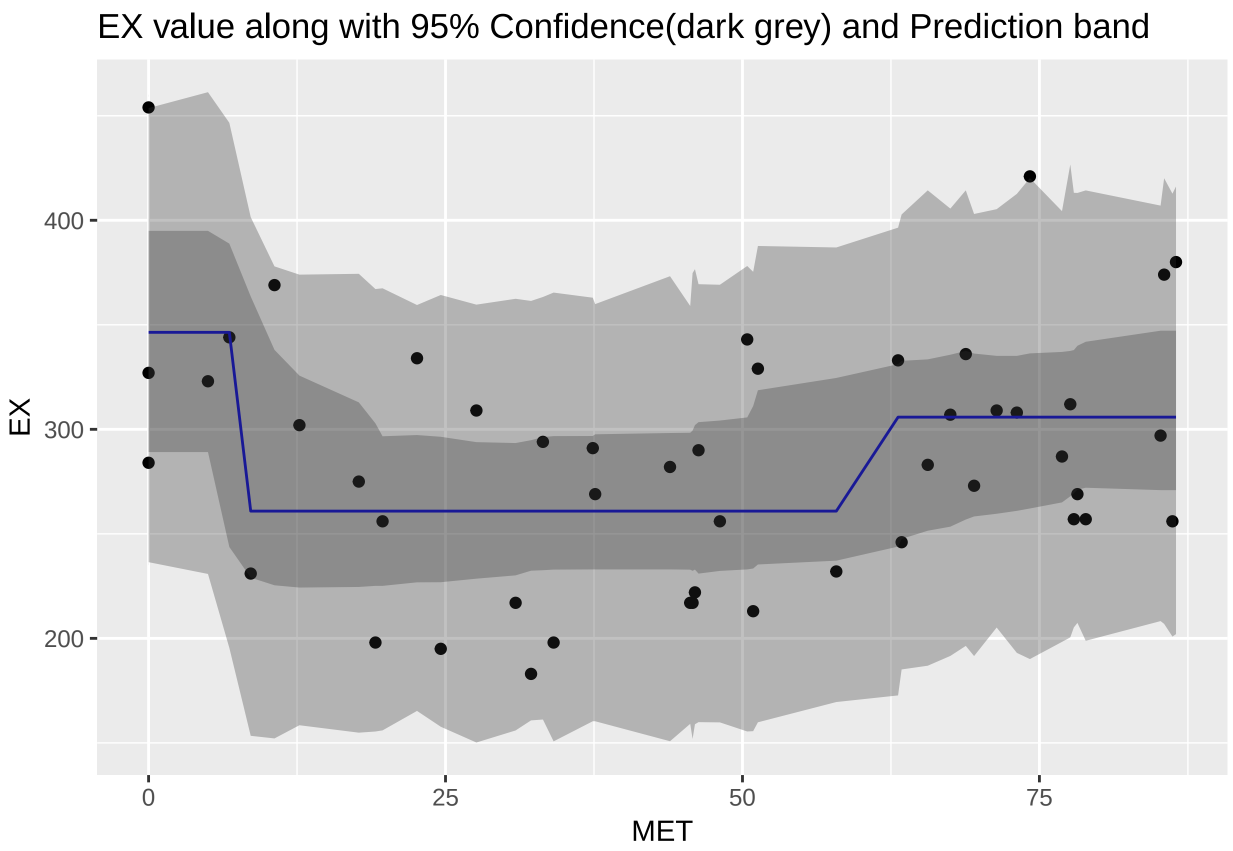


Prediction band using Bootstrap

```r
set.seed(12345)
state_tree_regression <- tree(data = state_data, EX~MET,
                              control = tree.control(nobs=NROW(state_data),
                                                     minsize = 8))


mle=prune.tree(state_tree_regression, best = 3)

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

# computing bootstrap samples
conf.fun <- function(data){
  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, best = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  return(final_fit_boot)
}


# computing bootstrap samples
pred.fun <- function(data){
  model <- tree(data = data,
       EX~MET,
       control = tree.control(nobs=NROW(data),
                              minsize = 8))

  model_purned <- prune.tree(model, best = 3)
  final_fit_boot <- predict(model_purned, newdata = state_data)
  final_fit <- rnorm(n = length(final_fit_boot),  mean = final_fit_boot, sd=sd(residuals(mle)))
  return(final_fit)
}


conf_para = boot(state_data, statistic=conf.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")
pred_para = boot(state_data, statistic=pred.fun, R=1000, mle=mle, ran.gen=rng, sim="parametric")

e1 <- envelope(conf_para, level = 0.95)
e2 <- envelope(pred_para, level = 0.95)
```

```
## Warning in envelope(pred_para, level = 0.95): unable to achieve requested overall error rate
```

```r
# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, best = 3)
predict_for_ci <- predict(state_cv_tree_purned, state_data)


data_for_ci_para <- cbind(upper_bound = e1$point[1,],
                     lower_bound = e1$point[2,],
                     upper_bound_pred = e2$point[1,],
                     lower_bound_pred = e2$point[2,],
                     EX = state_data$EX,
                     MET = state_data$MET,
                     predicted_value = predict_for_ci) %>% as.data.frame()

ggplot(data=data_for_ci_para, aes(x = MET, y = EX)) +
  geom_point(aes(x = MET,y=EX)) +
  geom_line(aes(x = MET, y=predicted_value), colour="blue") +
  geom_ribbon(aes(x = MET, ymin=lower_bound, ymax=upper_bound),alpha = 0.3) +
  geom_ribbon(aes(x = MET, ymin=lower_bound_pred, ymax=upper_bound_pred), alpha = 0.3) +
  ggtitle("EX value along with 95% Confidence(dark grey) and Prediction band")
```



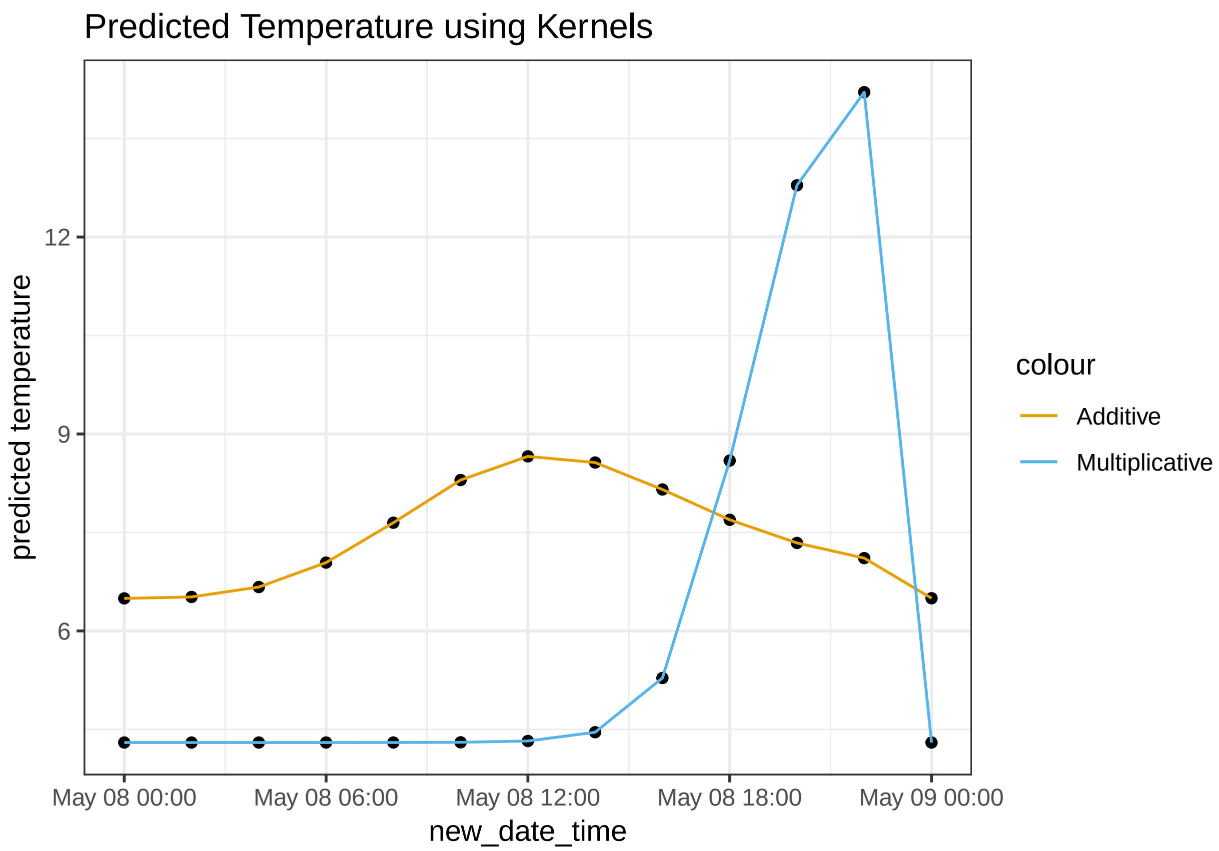
Kernal Estimation

Kernel Smoothing Regression

```
rm(list=ls())
set.seed(1234567890)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
rm(temps, stations)

st <- st[1:2000,]



kernel_method <- function(df, date, loc_long, loc_lat, h1, h2, h3) {

set.seed(1234567890)
start <- as.POSIXct(date)
interval <- 60
end <- start + as.difftime(1, units="days")
time_seq <- seq(from=start, by=interval*120, to=end)
time_seq <- as.data.frame(time_seq)
```

```
colnames(time_seq) <- "new_date_time"
time_seq$time_index <- rownames(time_seq)

df_new <- merge.data.frame(df,time_seq,all=TRUE)
rm(df)

df_new$new_date <- as.Date(df_new$new_date_time)
df_new$new_time <- format(df_new$new_date_time,"%H:%M:%S")
df_new$loc_long <- loc_long
df_new$loc_lat <-  loc_lat


df_new$h_distance <-  abs(geosphere::distHaversine(p1 = df_new[,c("loc_long", "loc_lat")],
                                        p2 = df_new[,c("longitude", "latitude")]))

df_new$h_date <- as.numeric(abs(difftime(df_new$new_date, df_new$date, units = c("days"))))

df_new$h_time <- as.numeric(abs(difftime(strptime(paste(df_new$new_date,
                                                      df_new$new_time),"%Y-%m-%d%H:%M:%S"),
                                         strptime(paste(df_new$new_date, df_new$time),
                                   "%Y-%m-%d %H:%M:%S"),
                          units = c("hour"))))


df_new$date_time <- paste(df_new$date, df_new$time)
df_new$hd_dist <- as.numeric(difftime(df_new$new_date_time,
                          df_new$date_time,
                          units = c("hour")))

## removing any negative dates and time
df_new$posterior_flag <- as.factor(ifelse(df_new$h_distance > 0 & df_new$hd_dist > 0, "retain", "drop")


## calculating kernel distance and choosing guassian kernel
df_new$h_distance_kernel <- exp(-(df_new$h_distance/h1)^2)
df_new$h_date_kernel <- exp(-(df_new$h_date/h2)^2)
df_new$h_time_kernel <- exp(-(df_new$h_time/h3)^2)
df_new$total_additive_dist <- (df_new$h_distance_kernel + df_new$h_date_kernel + df_new$h_time_kernel)
df_new$total_mul_dist <- (df_new$h_distance_kernel * df_new$h_date_kernel * df_new$h_time_kernel)

df_new$additive_num <- ifelse(df_new$posterior_flag == "retain",
                              df_new$h_distance_kernel*df_new$air_temperature +
                                df_new$h_date_kernel*df_new$air_temperature +
                                df_new$h_time_kernel*df_new$air_temperature,0)

df_new$mul_num <- ifelse(df_new$posterior_flag == "retain",
                         (df_new$h_distance_kernel) *
                                (df_new$h_date_kernel) *
                                (df_new$h_time_kernel*df_new$air_temperature),0)

df_new$additive_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_additive_dist, 0)
df_new$mul_den <- ifelse(df_new$posterior_flag == "retain", df_new$total_mul_dist, 0)
time = unique(time_seq$time_index)
result <- NULL

for(i in time){
temp <- df_new[df_new$time_index == i,]
additive_temp <- sum(temp$additive_num)/sum(temp$additive_den)
mult_temp <- sum(temp$mul_num)/sum(temp$mul_den)

temp <- cbind(additive_temp, mult_temp, i)
result <- rbind(temp,result)
}

result <- as.data.frame(result)
result <- merge(x =result, y = time_seq, by.x = "i", by.y = "time_index", all.x = TRUE)
result$additive_temp <- as.numeric(as.character(result$additive_temp))
result$mult_temp <- as.numeric(as.character(result$mult_temp))

p1 <- ggplot(data=result, aes(x=new_date_time)) +
  geom_point(aes(y = additive_temp)) +
  geom_point(aes(y = mult_temp)) +
  geom_line(aes(y = additive_temp, color = "Additive")) +
  geom_line(aes(y = mult_temp, color = "Multiplicative")) +
  scale_color_manual(values=c("#E69F00", "#56B4E9")) +
  ylab("predicted temperature") +
  theme_bw() +
  ggtitle("Predicted Temperature using Kernels")

final <- list(p1)
return(final)
}



kernel_method(df=st, date = "2000-05-08", loc_long = 17.6935,
loc_lat = 59.9953, h1 = 30000, h2 = 2, h3 = 5)
```

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



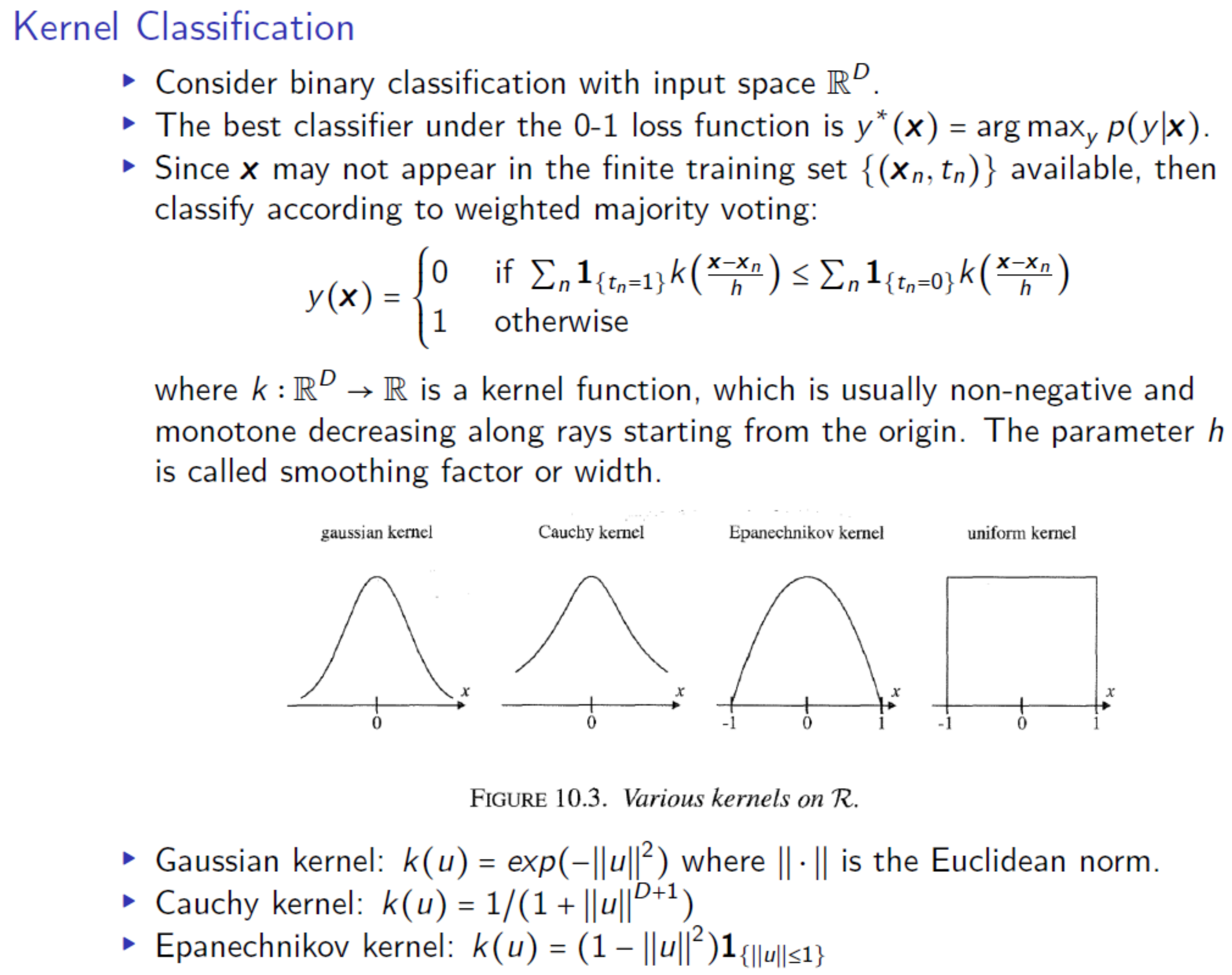
Onlearning SVM

```
set.seed(1234567890)
spam <- read.csv2("spambase.csv")
ind <- sample(1:nrow(spam))
spam <- spam[ind,c(1:48,58)]
h <- 1
beta <- 0
M <- 50
N <- 500 # number of training points

gaussian_k <- function(x, h) { # It is fine if students use exp(-x**2)/h instead
  return (exp(-(x**2)/(2*h*h)))
}

SVM <- function(sv,i) { #SVM on point i with support vectors sv
  yi <- 0
  for(m in 1:length(sv)) {
    xixm <- rbind(spam[i,-49],spam[sv[m],-49]) # do not use the true label when computing the distance
    tm <- 2 * spam[sv[m],49] - 1 # because the true labels must be -1/+1 and spambase has 0/1
    yi <- yi + tm * gaussian_k(dist(xixm, method="euclidean"), h)
  }
  return (yi)
}
```

```
errors <- 1
errorrate <- vector(length = N)
errorrate[1] <- 1
sv <- c(1)
for(i in 2:N) {
  yi <- SVM(sv,i)
  ti <- 2 * spam[i,49] - 1

  if(ti * yi < 0) {
    errors <- errors + 1
  }
  errorrate[i] <- errors/i

   cat(".") # iteration ", i, "error rate ", errorrate[i], ti * yi, "sv ", length(sv), "\n")
   flush.console()

  if(ti * yi <= beta) {
    sv <- c(sv, i)

    if (length(sv) > M) {
      for(m in 1:length(sv)) { # remove the support vector that gets classified best without itself
        sv2 <- sv[-m]
        ym2 <- SVM(sv2,sv[m])
        tm <- 2 * spam[sv[m],49] - 1

        if(m==1) {
          max <- tm * ym2
          ind <- 1
        }
        else {
          if(tm * ym2 > max) {
            max <- tm * ym2
            ind <- m
          }
        }
      }
      sv <- sv[-ind]

      # cat("removing ", ind, max, "\n")
      # flush.console()
    }
  }
}
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o")
M
beta
length(sv)
errorrate[N]
```

Kernel Notes

```r
knitr::include_graphics('./Kernel1.PNG')
```



Kernel Classification

- Consider binary classification with input space $\mathbb{R}^D$.
- The best classifier under the 0-1 loss function is $y^*(\boldsymbol{x}) = \arg\max_y p(y|\boldsymbol{x})$.
- Since $\boldsymbol{x}$ may not appear in the finite training set $\{(\boldsymbol{x}_n, t_n)\}$ available, then classify according to weighted majority voting:

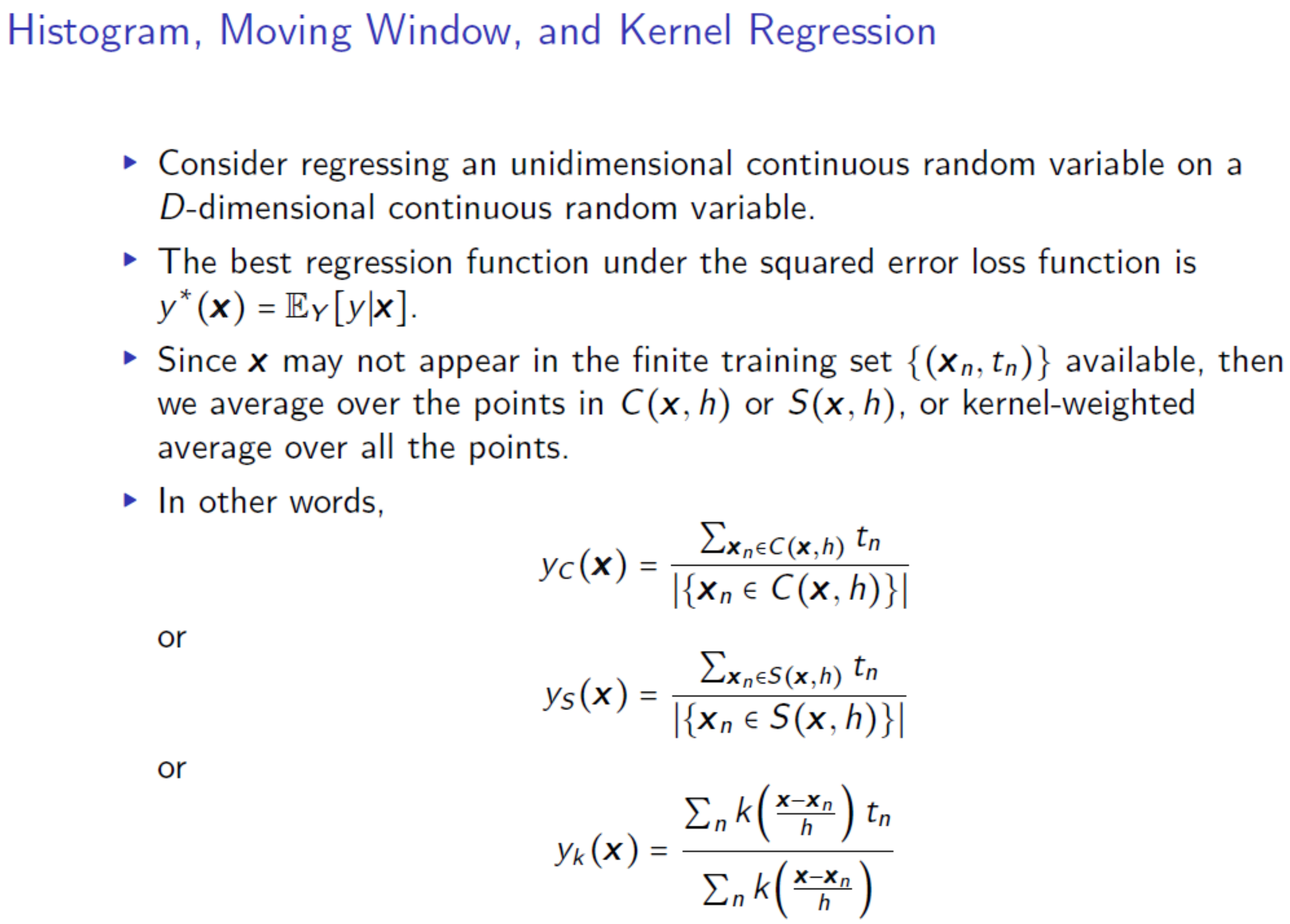
$$y(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \sum_n \mathbf{1}_{\{t_n=1\}} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) \leq \sum_n \mathbf{1}_{\{t_n=0\}} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

where $k : \mathbb{R}^D \to \mathbb{R}$ is a kernel function, which is usually non-negative and monotone decreasing along rays starting from the origin. The parameter h is called smoothing factor or width.

FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

- Gaussian kernel: $k(u) = exp(-||u||^2)$ where $||\cdot||$ is the Euclidean norm.
- Cauchy kernel: $k(u) = 1/(1 + ||u||^{D+1})$
- Epanechnikov kernel: $k(u) = (1 - ||u||^2)\mathbf{1}_{\{||u||\leq 1\}}$

```r
knitr::include_graphics('./Kernel2.PNG')
```



Histogram, Moving Window, and Kernel Regression

- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable.
- The best regression function under the squared error loss function is $y^*(\boldsymbol{x}) = \mathbb{E}_Y[y|\boldsymbol{x}]$.
- Since $\boldsymbol{x}$ may not appear in the finite training set $\{(\boldsymbol{x}_n, t_n)\}$ available, then we average over the points in $C(\boldsymbol{x}, h)$ or $S(\boldsymbol{x}, h)$, or kernel-weighted average over all the points.
- In other words,

$$y_C(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_n \in C(\boldsymbol{x},h)} t_n}{|\{\boldsymbol{x}_n \in C(\boldsymbol{x}, h)\}|}$$

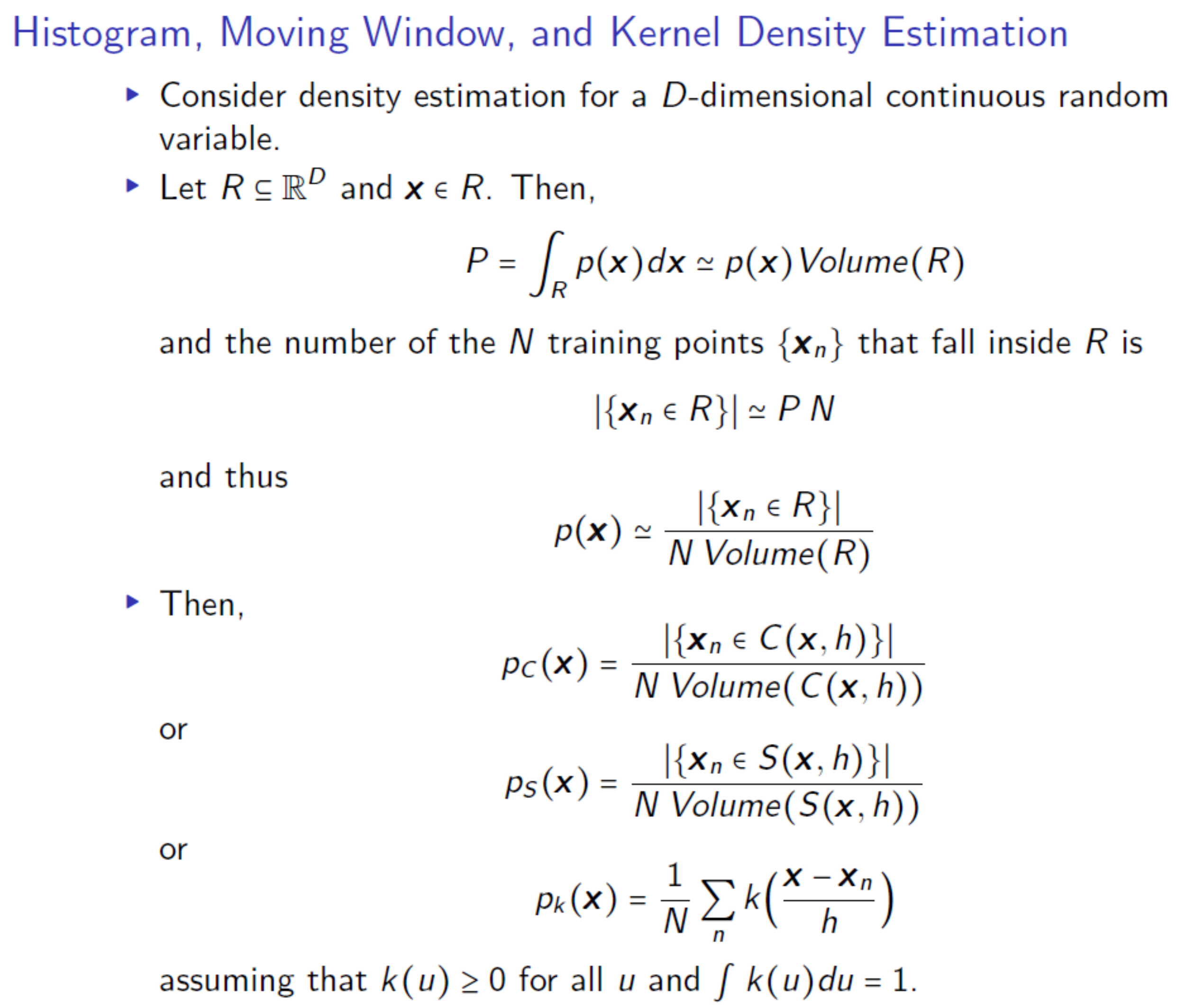
or

$$y_S(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_n \in S(\boldsymbol{x},h)} t_n}{|\{\boldsymbol{x}_n \in S(\boldsymbol{x}, h)\}|}$$

or

$$y_k(\boldsymbol{x}) = \frac{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) t_n}{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)}$$

```r
knitr::include_graphics('./Kernel3.PNG')
```



Histogram, Moving Window, and Kernel Density Estimation

- Consider density estimation for a D-dimensional continuous random variable.
- Let $R \subseteq \mathbb{R}^D$ and $\boldsymbol{x} \in R$. Then,

$$P = \int_R p(\boldsymbol{x}) d\boldsymbol{x} \simeq p(\boldsymbol{x}) Volume(R)$$

  and the number of the N training points $\{\boldsymbol{x}_n\}$ that fall inside R is

$$|\{\boldsymbol{x}_n \in R\}| \simeq P\,N$$

  and thus

$$p(\boldsymbol{x}) \simeq \frac{|\{\boldsymbol{x}_n \in R\}|}{N\, Volume(R)}$$

- Then,

$$p_C(\boldsymbol{x}) = \frac{|\{\boldsymbol{x}_n \in C(\boldsymbol{x}, h)\}|}{N\, Volume(C(\boldsymbol{x}, h))}$$

  or

$$p_S(\boldsymbol{x}) = \frac{|\{\boldsymbol{x}_n \in S(\boldsymbol{x}, h)\}|}{N\, Volume(S(\boldsymbol{x}, h))}$$

  or

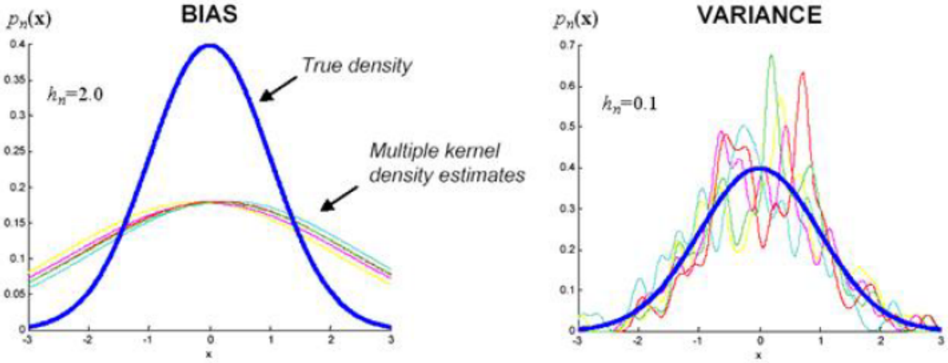
$$p_k(\boldsymbol{x}) = \frac{1}{N} \sum_n k\Big(\frac{\boldsymbol{x} - \boldsymbol{x}_n}{h}\Big)$$

  assuming that $k(u) \geq 0$ for all u and $\int k(u) du = 1$.

```
knitr::include_graphics('./Kernel4.PNG')
```

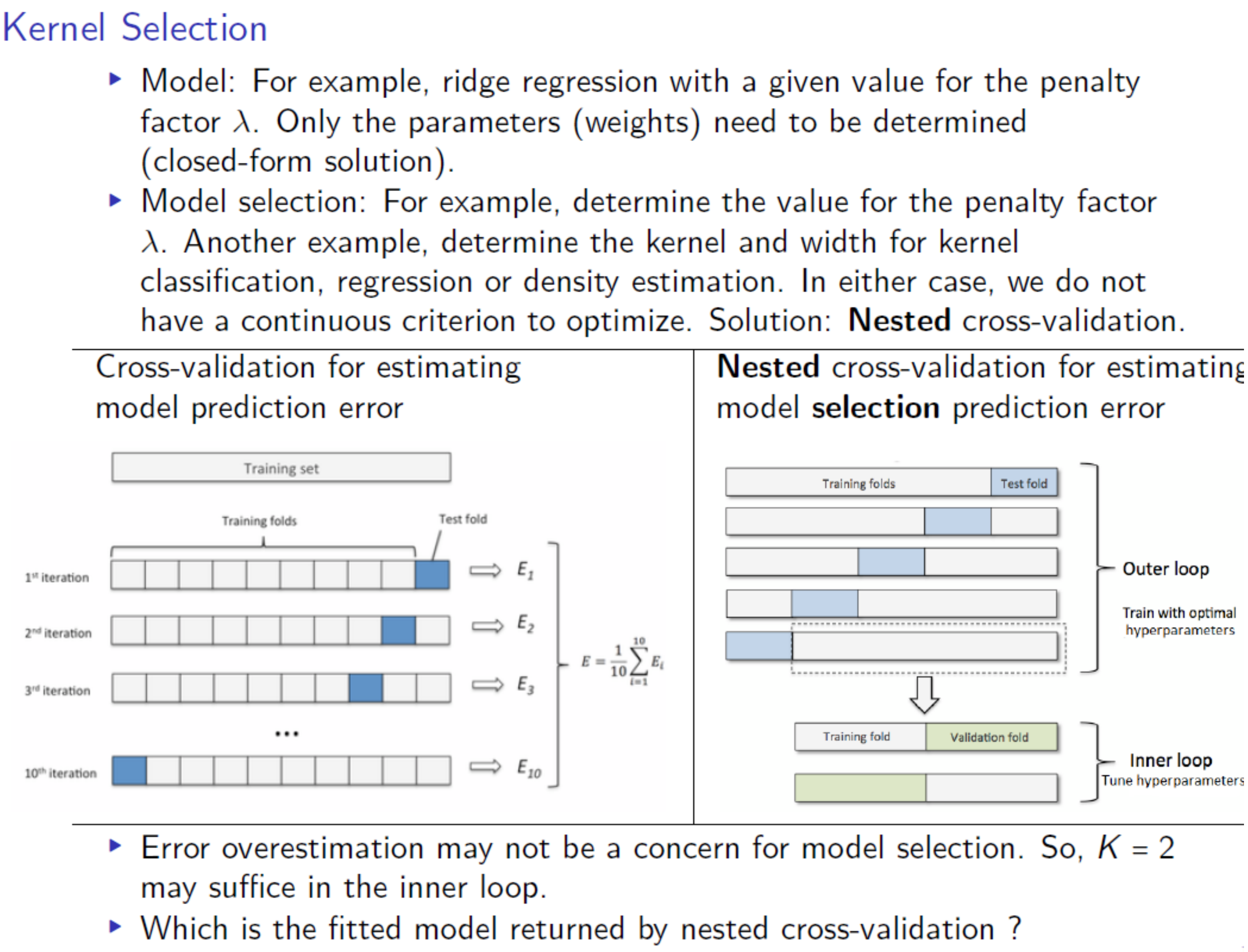
Kernel Selection

- How to choose the right kernel and width ? E.g., by cross-validation.
- What does "right" mean ? E.g., minimize loss function.
- Note that the width of the kernel corresponds to a bias-variance trade-off.



- Small width implies considering few points. So, the variance will be large (similar to the variance of a single point). The bias will be small since the points considered are close to $\boldsymbol{x}$.
- Large width implies considering many points. So, the variance will be small and the bias will be large.

```
knitr::include_graphics('./Kernel5.PNG')
```



Kernel Selection

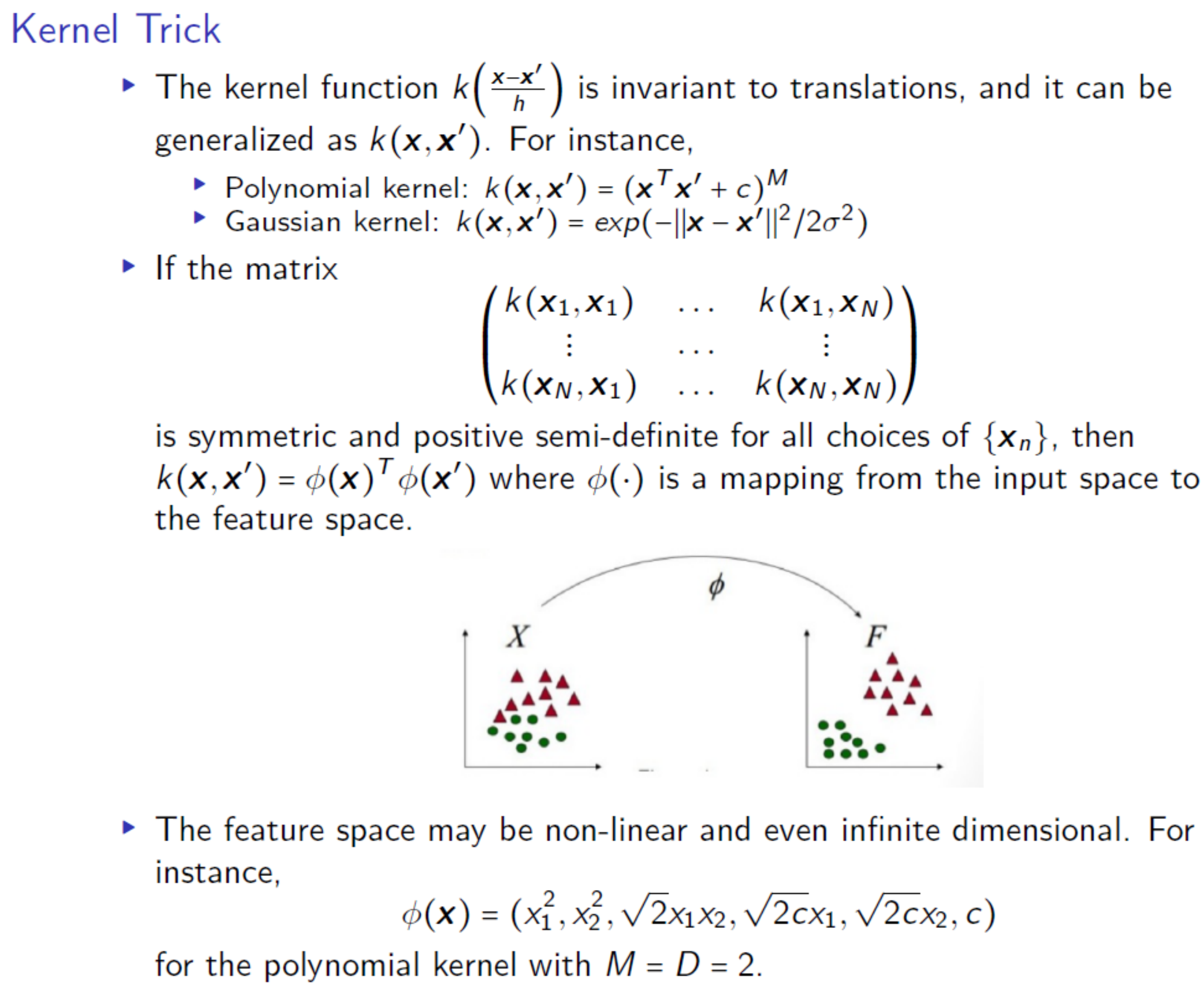
- Model: For example, ridge regression with a given value for the penalty factor λ. Only the parameters (weights) need to be determined (closed-form solution).
- Model selection: For example, determine the value for the penalty factor λ. Another example, determine the kernel and width for kernel classification, regression or density estimation. In either case, we do not have a continuous criterion to optimize. Solution: **Nested** cross-validation.

Cross-validation for estimating model prediction error

Nested cross-validation for estimating model **selection** prediction error

- Error overestimation may not be a concern for model selection. So, $K = 2$ may suffice in the inner loop.
- Which is the fitted model returned by nested cross-validation ?

```r
knitr::include_graphics('./Kernel6.PNG')
```

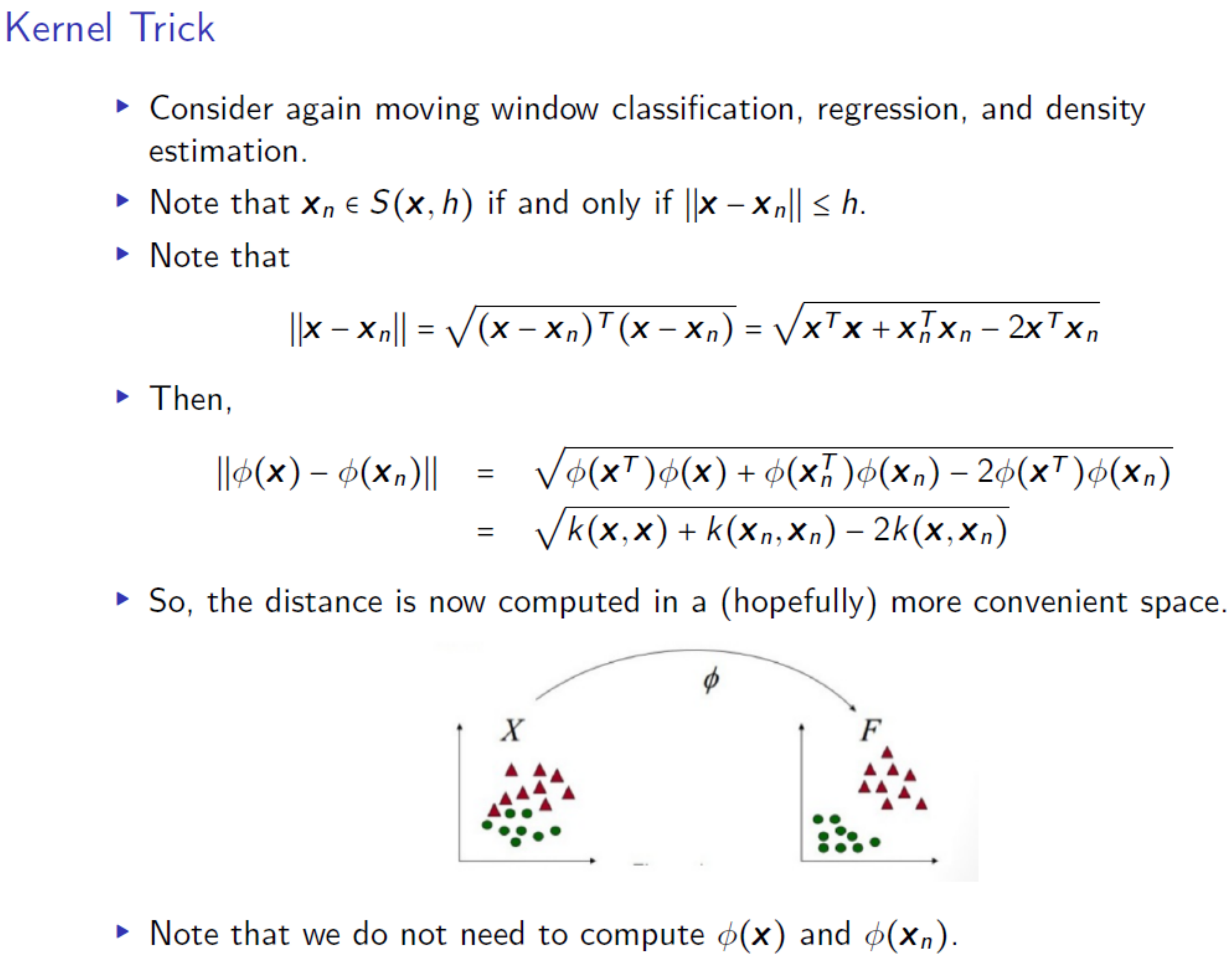


Kernel Trick

- The kernel function $k\left(\frac{x-x'}{h}\right)$ is invariant to translations, and it can be generalized as $k(\boldsymbol{x},\boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x},\boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^M$
  - Gaussian kernel: $k(\boldsymbol{x},\boldsymbol{x}') = exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2/2\sigma^2)$
- If the matrix
$$\begin{pmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1,\boldsymbol{x}_N) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_N,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N,\boldsymbol{x}_N) \end{pmatrix}$$
is symmetric and positive semi-definite for all choices of $\{\boldsymbol{x}_n\}$, then $k(\boldsymbol{x},\boldsymbol{x}') = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$ where $\phi(\cdot)$ is a mapping from the input space to the feature space.

- The feature space may be non-linear and even infinite dimensional. For instance,
$$\phi(\boldsymbol{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c)$$
for the polynomial kernel with $M = D = 2$.

```r
knitr::include_graphics('./Kernel7.PNG')
```



Kernel Trick

- Consider again moving window classification, regression, and density estimation.
- Note that $\mathbf{x}_n \in S(\mathbf{x}, h)$ if and only if $||\mathbf{x} - \mathbf{x}_n|| \leq h$.
- Note that

$$||\mathbf{x} - \mathbf{x}_n|| = \sqrt{(\mathbf{x} - \mathbf{x}_n)^T(\mathbf{x} - \mathbf{x}_n)} = \sqrt{\mathbf{x}^T\mathbf{x} + \mathbf{x}_n^T\mathbf{x}_n - 2\mathbf{x}^T\mathbf{x}_n}$$

- Then,

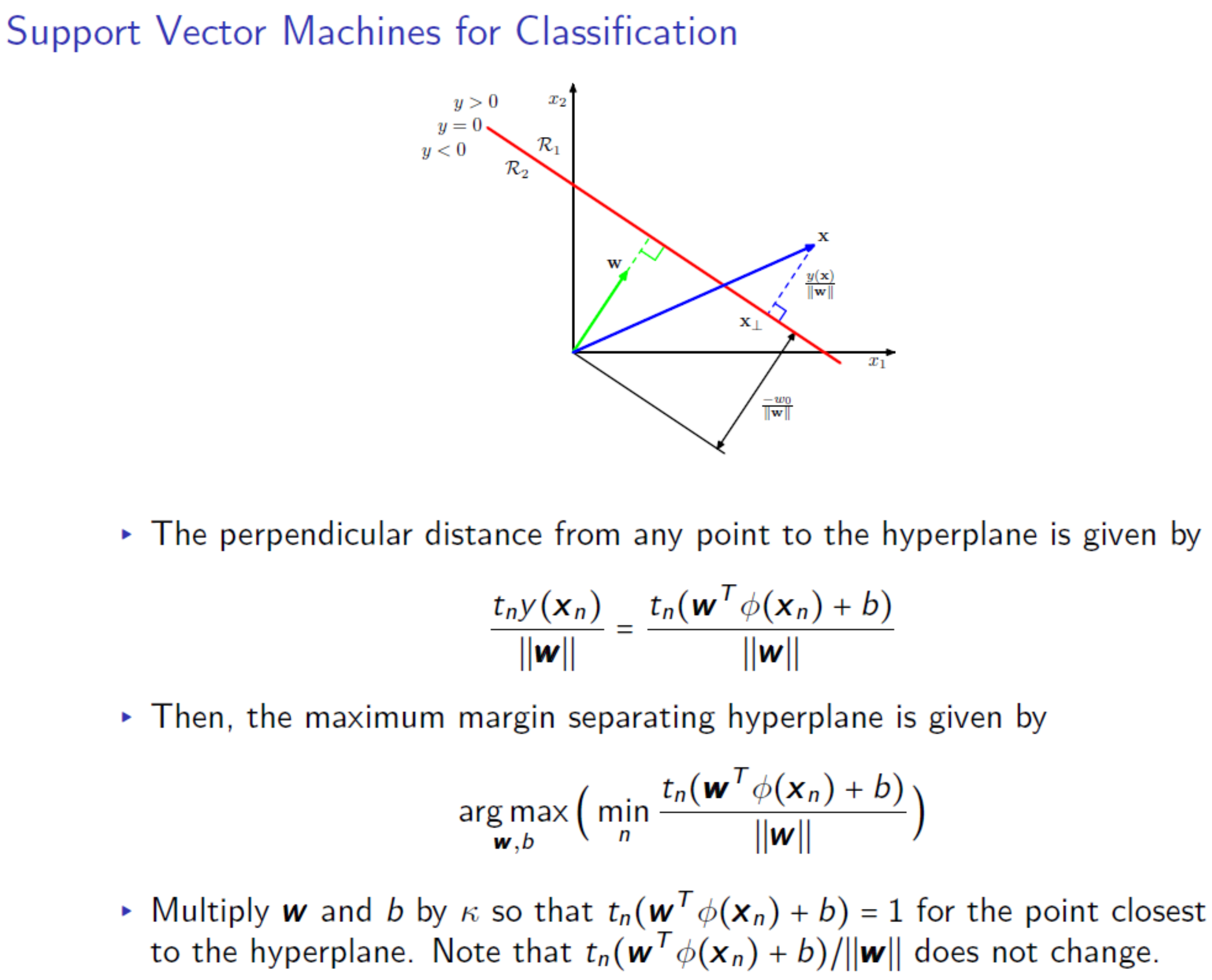
$$\begin{aligned} ||\phi(\mathbf{x}) - \phi(\mathbf{x}_n)|| &= \sqrt{\phi(\mathbf{x}^T)\phi(\mathbf{x}) + \phi(\mathbf{x}_n^T)\phi(\mathbf{x}_n) - 2\phi(\mathbf{x}^T)\phi(\mathbf{x}_n)} \\ &= \sqrt{k(\mathbf{x}, \mathbf{x}) + k(\mathbf{x}_n, \mathbf{x}_n) - 2k(\mathbf{x}, \mathbf{x}_n)} \end{aligned}$$

- So, the distance is now computed in a (hopefully) more convenient space.

- Note that we do not need to compute $\phi(\mathbf{x})$ and $\phi(\mathbf{x}_n)$.

## SVM Notes

```
knitr::include_graphics('./SVM1.PNG')
```



Support Vector Machines for Classification

$y > 0$
$y = 0$
$y < 0$
x_2
$\mathcal{R}_1$
$\mathcal{R}_2$
$\mathbf{w}$
$\mathbf{x}$
$\frac{y(\mathbf{x})}{\|\mathbf{w}\|}$
$\mathbf{x}_\perp$
x_1
$\frac{-w_0}{\|\mathbf{w}\|}$

- The perpendicular distance from any point to the hyperplane is given by

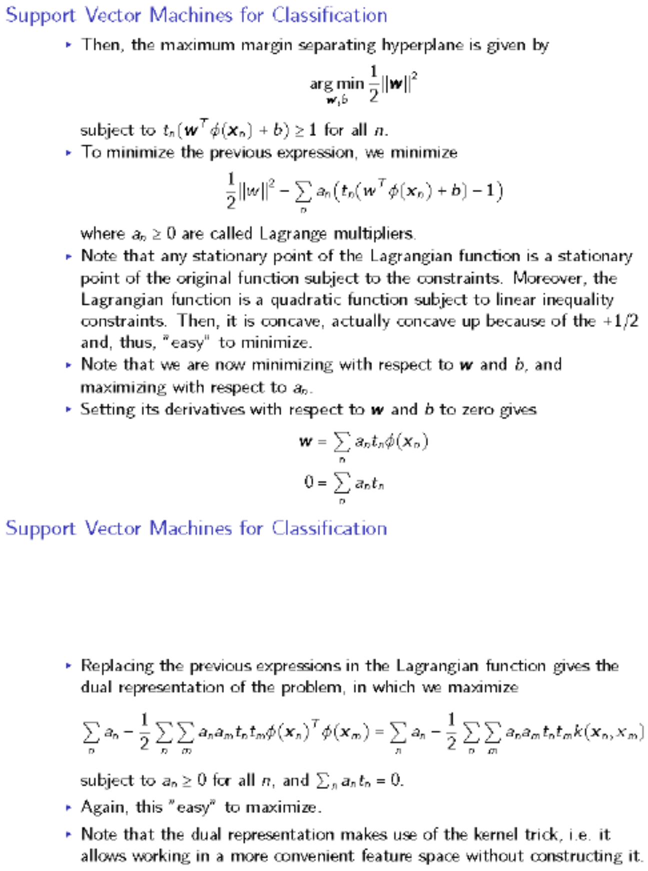
$$\frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||} = \frac{t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b)}{||\boldsymbol{w}||}$$

- Then, the maximum margin separating hyperplane is given by

$$\arg\max_{\boldsymbol{w},b}\left(\min_n \frac{t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b)}{||\boldsymbol{w}||}\right)$$

- Multiply $\boldsymbol{w}$ and b by κ so that $t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b) = 1$ for the point closest to the hyperplane. Note that $t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b)/||\boldsymbol{w}||$ does not change.

```
knitr::include_graphics('./SVM2.PNG')
```



Support Vector Machines for Classification

- Then, the maximum margin separating hyperplane is given by

$$\arg\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2$$

subject to $t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n)+b) \geq 1$ for all n.

- To minimize the previous expression, we minimize

$$\frac{1}{2}||w||^2 - \sum_n a_n\big(t_n(w^T\phi(\boldsymbol{x}_n)+b)-1\big)$$

where $a_n \geq 0$ are called Lagrange multipliers.

- Note that any stationary point of the Lagrangian function is a stationary point of the original function subject to the constraints. Moreover, the Lagrangian function is a quadratic function subject to linear inequality constraints. Then, it is concave, actually concave up because of the +1/2 and, thus, "easy" to minimize.
- Note that we are now minimizing with respect to $\boldsymbol{w}$ and b, and maximizing with respect to a_n.
- Setting its derivatives with respect to $\boldsymbol{w}$ and b to zero gives

$$\boldsymbol{w} = \sum_n a_n t_n \phi(\boldsymbol{x}_n)$$

$$0 = \sum_n a_n t_n$$

Support Vector Machines for Classification

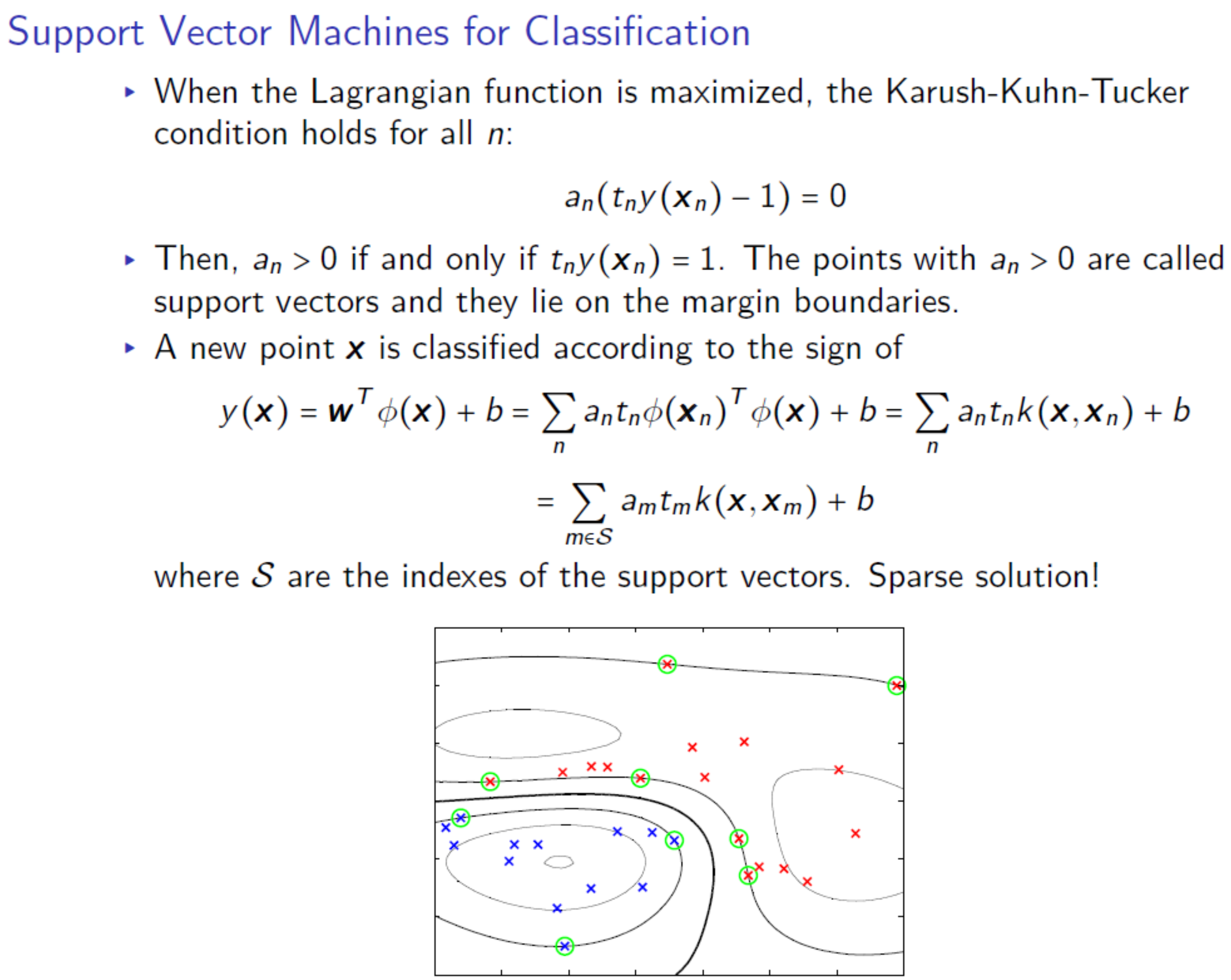
- Replacing the previous expressions in the Lagrangian function gives the dual representation of the problem, in which we maximize

$$\sum_n a_n - \frac{1}{2}\sum_n\sum_m a_n a_m t_n t_m \phi(\boldsymbol{x}_n)^T\phi(\boldsymbol{x}_m) = \sum_n a_n - \frac{1}{2}\sum_n\sum_m a_n a_m t_n t_m k(\boldsymbol{x}_n, x_m)$$

subject to $a_n \geq 0$ for all n, and $\sum_n a_n t_n = 0$.

- Again, this "easy" to maximize.
- Note that the dual representation makes use of the kernel trick, i.e. it allows working in a more convenient feature space without constructing it.

```r
knitr::include_graphics('./SVM3.PNG')
```



Support Vector Machines for Classification

- When the Lagrangian function is maximized, the Karush-Kuhn-Tucker condition holds for all n:

$$a_n(t_n y(\mathbf{x}_n) - 1) = 0$$

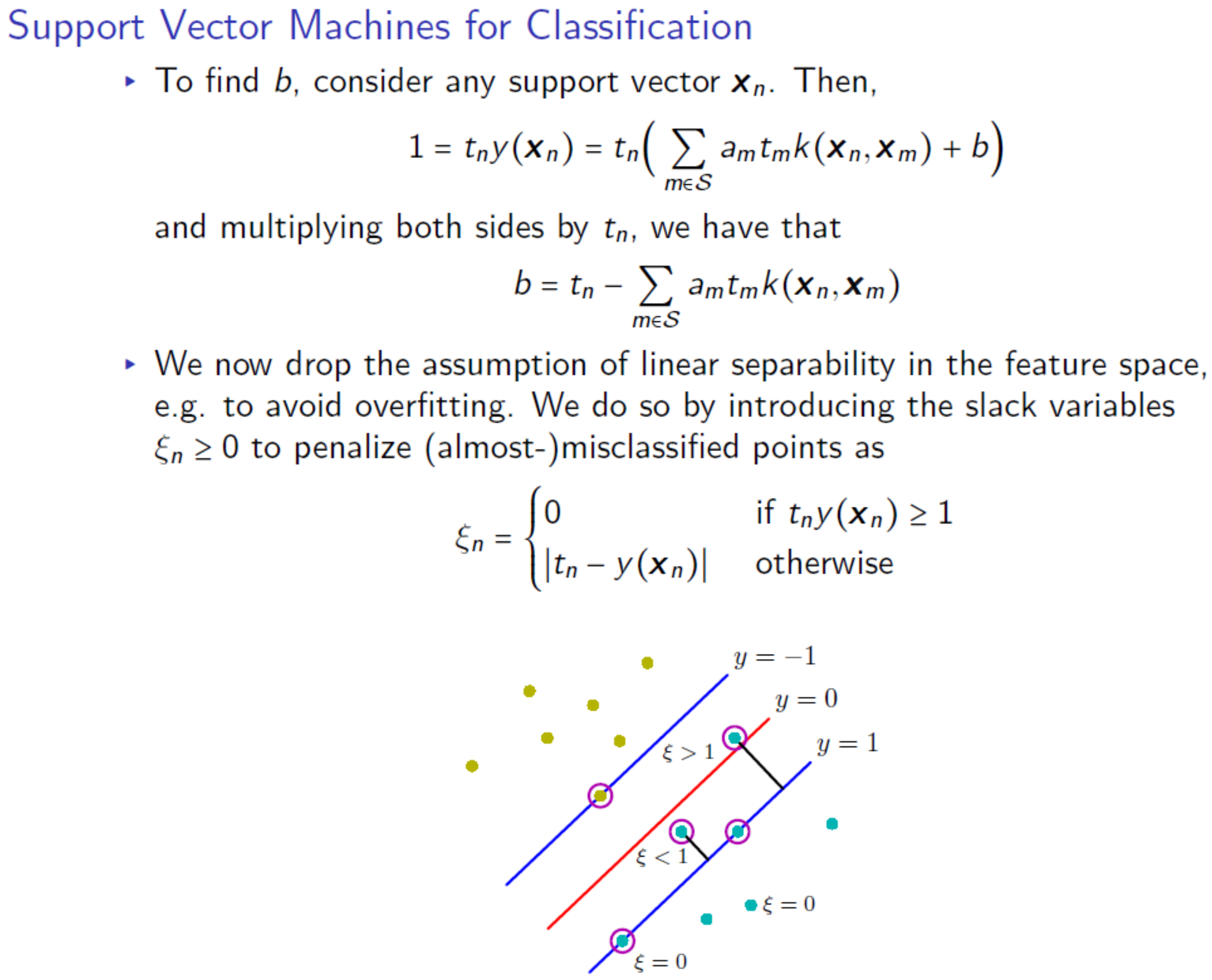
- Then, $a_n > 0$ if and only if $t_n y(\mathbf{x}_n) = 1$. The points with $a_n > 0$ are called support vectors and they lie on the margin boundaries.
- A new point $\mathbf{x}$ is classified according to the sign of

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_n a_n t_n \phi(\mathbf{x}_n)^T \phi(\mathbf{x}) + b = \sum_n a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b$$

$$= \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}, \mathbf{x}_m) + b$$

where $\mathcal{S}$ are the indexes of the support vectors. Sparse solution!

```r
knitr::include_graphics('./SVM4.PNG')
```



Support Vector Machines for Classification

- To find b, consider any support vector $\mathbf{x}_n$. Then,

$$1 = t_n y(\mathbf{x}_n) = t_n\Big(\sum_{m\in\mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b\Big)$$

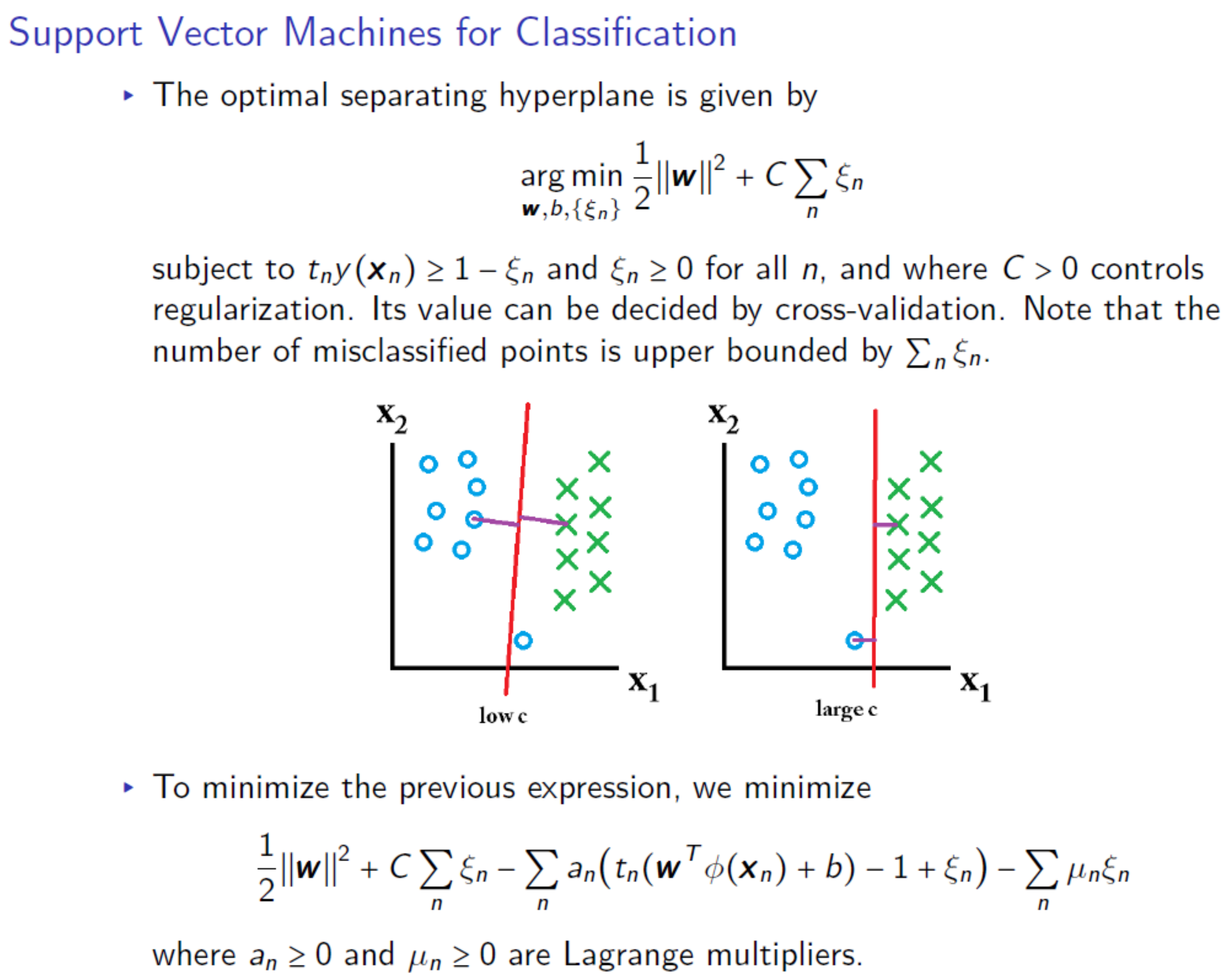
and multiplying both sides by t_n, we have that

$$b = t_n - \sum_{m\in\mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

- We now drop the assumption of linear separability in the feature space, e.g. to avoid overfitting. We do so by introducing the slack variables $\xi_n \geq 0$ to penalize (almost-)misclassified points as

$$\xi_n = \begin{cases} 0 & \text{if } t_n y(\mathbf{x}_n) \geq 1 \\ |t_n - y(\mathbf{x}_n)| & \text{otherwise} \end{cases}$$

```
knitr::include_graphics('./SVM5.PNG')
```



Support Vector Machines for Classification

- The optimal separating hyperplane is given by

$$\underset{\boldsymbol{w},b,\{\xi_n\}}{\arg\min}\ \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_n \xi_n$$

subject to $t_n y(\boldsymbol{x}_n) \geq 1 - \xi_n$ and $\xi_n \geq 0$ for all n, and where $C > 0$ controls regularization. Its value can be decided by cross-validation. Note that the number of misclassified points is upper bounded by $\sum_n \xi_n$.

- To minimize the previous expression, we minimize

$$\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_n \xi_n - \sum_n a_n\Big(t_n(\boldsymbol{w}^T\phi(\boldsymbol{x}_n) + b) - 1 + \xi_n\Big) - \sum_n \mu_n\xi_n$$

where $a_n \geq 0$ and $\mu_n \geq 0$ are Lagrange multipliers.

```
knitr::include_graphics('./SVM6.PNG')
```

Support Vector Machines for Classification

- Setting its derivatives with respect to $\boldsymbol{w}$, b and ξ_n to zero gives

$$\begin{aligned} \boldsymbol{w} &= \sum_n a_n t_n \phi(\boldsymbol{x}_n) \\ 0 &= \sum_n a_n t_n \\ a_n &= C - \mu_n \end{aligned}$$

- Replacing these in the Lagrangian function gives the dual representation of the problem, in which we maximize

$$\sum_n a_n - \frac{1}{2} \sum_n \sum_m a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

  subject to $a_n \geq 0$ and $a_n \leq C$ for all n, because $\mu_n \geq 0$.
- When the Lagrangian function is maximized, the Karush-Kuhn-Tucker conditions hold for all n:

$$\begin{aligned} a_n(t_n y(\boldsymbol{x}_n) - 1 + \xi_n) &= 0 \\ \mu_n \xi_n &= 0 \end{aligned}$$

- Then, $a_n > 0$ if and only if $t_n y(\boldsymbol{x}_n) = 1 - \xi_n$ for all n. The points with $a_n > 0$ are called support vectors and they lie
  - on the margin if $a_n < C$, because then $\mu_n > 0$ and thus $\xi_n = 0$, or
  - inside the margin (even on the wrong side of the decision boundary) if $a_n = C$, because then $\mu_n = 0$ and thus ξ_n is unconstrained.