lab1block1P

Prudhvi Peddmallu

23 April 2019

Assignment 1. Spam classification with nearest neighbors

kknn(nearest neighbour),logisticregression,Feature selection by cross-validation in a linear model

The data file spambase.xlsx contains information about the frequency of various words, characters etc for a total of 2740 e-mails. Furthermore, these e-mails have been manually classified as spams (spam = 1) or regular e-mails (spam = 0). 1. Import the data into R and divide it into training and test sets (50%/50%) by using the following code: n=dim(data)[1] set.seed(12345) id=sample(1:n, floor(n*0.5)) train=data[id,] test=data[-id,] 2. Use logistic regression (functions glm(), predict()) to classify the training and test data by the classification principle$\hat{Y} = 1 if p(Y = 1/X) > 0.5$,otherwise$\hat{Y} = 0$ and report the confusion matrices (use table()) and the misclassification rates for training and test data. Analyse the obtained results. 3. Use logistic regression to classify the test data by the classification principle $\hat{Y} = 1 if p(Y = 1/X) > 0.9$,otherwise$\hat{Y} = 0$ and report the confusion matrices (use table()) and the misclassification rates for training and test data. Compare the results. What effect did the new rule have? 4. Use standard classifier kknn() with K=30 from package kknn, report the the misclassification rates for the training and test data and compare the results with step 2. 5. Repeat step 4 for K=1 and compare the results with step 4. What effect does the decrease of K lead to and why? ## libraries of kknn & glm

```
library(readxl)
library(kknn)
library(ggplot2)
library(glmnet)
library(MASS)
```

xlsx file

1.1-divideing the data

```
library(readxl)
spambase <-read_xlsx('spambase.xlsx')
### diveding the data
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=spambase[id,]
test=spambase[-id,]
```

1.2&1.3logistic regression

```
model <- glm(Spam~.,family=binomial(link='logit'),data=train)
predection1_test <- predict(model,newdata=test,type='response')
predection5_test <-ifelse(predection1_test > 0.5,1,0)

predection1_train <- predict(model,newdata=train,type='response')
predection5_train <-ifelse(predection1_train > 0.5,1,0)
confusionmatrix5_test<-data.frame("Spam" = test$Spam,
              "predection5_test"= as.numeric(predection5_test))
confusionmatrix5_test = table(confusionmatrix5_test)
confusionmatrix5_train<-data.frame("Spam" = train$Spam,"predection5_train"=
                                     as.numeric(predection5_train))
confusionmatrix5_train = table(confusionmatrix5_train)
diagonal_train<-sum(diag(confusionmatrix5_train))
misclassificationrate5_train <- 1-(sum(diagonal_train)/sum(confusionmatrix5_train))
diagonal_test<-diag(confusionmatrix5_test)
misclassificationrate5_test<-1-(sum(diagonal_test)/sum(confusionmatrix5_test))
```

```
cat("Using glm classifier dividing data at 0.5 ")

cat("Confusion matrix for train data")

confusionmatrix5_train

cat("Confusion matrix for test data")

confusionmatrix5_test

cat("misclassification rate for train data",misclassificationrate5_train)

cat("\nmisclassification rate for test data",misclassificationrate5_test)
```

1.2&1.3-2method

```
spam_data <- read.xlsx("spambase.xlsx", sheetName = "spambase_data")
spam_data$Spam <- as.factor(spam_data$Spam)
tecator_data <- read.xlsx("tecator.xlsx", sheetName = "data")
tecator_data <- tecator_data[,2:NCOL(tecator_data)] # removing sample column
set.seed(12345)
n = NROW(spam_data)
id = sample(1:n, floor(n*0.5))
train = spam_data[id,]
test = spam_data[-id,]
best_model <- glm(formula = Spam ~., family = binomial, data = train)
summary(best_model)
# prediction
train$prediction_prob <- predict(best_model, newdata = train, type = "response")
test$prediction_prob <- predict(best_model, newdata = test , type = "response")
train$prediction_class_50 <- ifelse(train$prediction_prob > 0.50, 1, 0)
test$prediction_class_50 <- ifelse(test$prediction_prob > 0.50, 1, 0)
train$prediction_class_90 <- ifelse(train$prediction_prob > 0.90, 1, 0)
test$prediction_class_90 <- ifelse(test$prediction_prob > 0.90, 1, 0)
```

```
# plots
ggplot(train, aes(prediction_prob, color = Spam)) +
geom_density(size = 1) + ggtitle("Training Set's Predicted Score for 50% cutoff") +
scale_color_economist(name = "data", labels = c("negative", "positive")) +
theme_economist()
ggplot(test, aes(prediction_prob, color = Spam)) +
geom_density(size = 1) + ggtitle("Test Set's Predicted Score for 50% cutoff") +
scale_color_economist(name = "data", labels = c("negative", "positive")) +
theme_economist()
#confusion table
conf_train <- table(train$Spam, train$prediction_class_50)
```

```
names(dimnames(conf_train)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train)
conf_test <- table(test$Spam, test$prediction_class_50)
names(dimnames(conf_test)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test)
#confusion table
conf_train1 <- table(train$Spam, train$prediction_class_90)
names(dimnames(conf_train1)) <- c("Actual Train", "Predicted Train")
conf_train1
conf_test1 <- table(test$Spam, test$prediction_class_90)
names(dimnames(conf_test1)) <- c("Actual Test", "Predicted Test")
conf_test1
#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")
```

Analysis:-When the algorithm was modeled using classification principle which divided at 0.5, the observed misclassification rate was 0.177 for test and 0.162 for train. As variance is not high, the issue of overfitting is not happening and the misclassification rate is comparitively smaller value which means the model is not underfitting as well. When the algorithm classified data at 0.9 , the misclassification rate increased to 0.312 for test data and 0.306 for training data. So the misclassification rate is increasing using this classification principle. On analysing the misclassification matrix we see an increase in observations which are actually spam but not considered as spam because of increment in the conditioning value. So the efficiency of the model in predicting the spams has decreased.

1.4-kknn-knearest neighbour

```
#train data
knearest_30_1 <- kknn(formula = factor(Spam) ~., train, train,k = 30, kernel = "optimal")
fit_kknn30_1 <- fitted(knearest_30_1)
tablekknn_30_1 <- table(train$Spam,fit_kknn30_1)
misclassificationkknn_30_1 <- 1- (sum(diag(tablekknn_30_1))/sum(tablekknn_30_1))
```

```
#test data
knearest_30_2 <- kknn(formula = factor(Spam) ~., train, test,k = 30, kernel = "optimal")
fit_kknn30_2 <- fitted(knearest_30_2)
tablekknn_30_2 <- table(test$Spam,fit_kknn30_2)
misclassificationkknn_30_2 <- 1- (sum(diag(tablekknn_30_2))/sum(tablekknn_30_2))
```

```
cat("kknn algorithm with k = 30")

cat("Confusion matrix for train data")

tablekknn_30_1

cat("Confusion matrix for test data")

tablekknn_30_2

cat("misclassification rate for train data = ", misclassificationkknn_30_1)

cat("misclassification rate for test data = ", misclassificationkknn_30_2)
```

1.4 kknn-2method

```
#k=30
knn_model30 <- train.kknn(Spam ~., data = train, kmax = 30)
train$knn_prediction_class <- predict(knn_model30, train)
test$knn_prediction_class <- predict(knn_model30, test)
conf_train2 <- table(train$Spam, train$knn_prediction_class)
names(dimnames(conf_train2)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train2)
conf_test2 <- table(test$Spam, test$knn_prediction_class)
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)
#k=1
knn_model1 <- train.kknn(Spam ~., data = train, kmax = 1)
train$knn_prediction_class <- predict(knn_model1, train)
test$knn_prediction_class <- predict(knn_model1, test)
conf_train2 <- table(train$Spam, train$knn_prediction_class)
names(dimnames(conf_train2)) <- c("Actual Train", "Predicted Train")
confusionMatrix(conf_train2)
conf_test2 <- table(test$Spam, test$knn_prediction_class)
names(dimnames(conf_test2)) <- c("Actual Test", "Predicted Test")
confusionMatrix(conf_test2)
```

The classifier kknn() was used to do the classification model. This function is used to do classification using K nearest neighbour algorithm. Here we take K=30 (considering 30 nearest neighbours). A misclassification rate of 0.329 is obtained for test data and 0.172 is obtained for train data.

On comparing this results with step 2, there is no much change in train error but the test error shows a significant increase where here it is 0.329 whereas in step 2 it was 0.17.

So here we can say that the model in step 2 is better compared to the model using k nearest neighbout algorithm with number of neighbours is 30.

k=1

Misclassification for test data increases from 0.32 to 0.34 whereas training error decreased to zero on selecting k as 1. Here none of the training data is misclassified. So here we can say that when we take k as 1, the model is highly biased to train data and is not a generalised model

So as value of K increases, the model is getting generalised which inturn decreases the test error.

The accuracy of k nearest neighbour algorithm depends on selecting the appropriate features. If not the model will be affected by the noisy data.

Assignment 3. Feature selection by cross-validation in a linear model.

1. Implement an R function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation without using any specialized function like lm() (use only basic R functions). Your function should depend on: . X: matrix containing X measurements. . Y: vector containing Y measurements . Nfolds: number of folds in the cross-validation. You may assume in your code that matrix X has 5 columns. The function should plot the CV scores computed for various feature subsets against the number of features, and it should also return the optimal subset of features and the corresponding cross-validation (CV) score. Before splitting into folds, the data should be permuted, and the seed 12345 should be used for that purpose.
2. Test your function on data set swiss available in the standard R repository: . Fertility should be Y . All other variables should be X . Nfolds should be 5 Report the resulting plot and interpret it. Report the optimal subset of features and comment whether it is reasonable that these specific features have largest impact on the target

3.163.2-k-fold cross-validation

```
feature_selection <-  function(X,Y,Nfolds){


  Intercept <- rep(1,length(Y))

  Number_Of_Features <- c()
  cv_rate <- c()
  best_model <- 0
  cv_rate_bestmodel <- 0

  total_features <- ncol(X)

  for(k in 1: total_features){
    M = combn(total_features , k)

    for(j in 1:ncol(M)){
      Z = X[ ,M[ , j]]

      #Adding intercept
      Z = as.matrix(cbind(Intercept , Z))

      data <- cbind(Y ,  Z)
```

```
      #Randomly shuffle the data
      set.seed(12345)

      data_s<-data[sample(nrow(data)), ]

      #Create N equally size folds
      folds <- cut(seq(1,nrow(data_s)),breaks=Nfolds,labels=FALSE)
      #folds <- rep_len(1:Nfolds,nrow(data_s))

      cv = 0
      #Nfolds cross validation to find the best model

      for(i in 1:Nfolds){

        testIndexes <- which(folds==i,arr.ind=TRUE)

        test_fold <- data_s[testIndexes, ]
        training_fold <- data_s[-testIndexes, ]

        X_train = as.matrix(training_fold[,-1])
        Y_train = as.matrix(training_fold[,1])

        beta_hat <- solve(t(X_train)%*%X_train)%*%t(X_train)%*% Y_train
        y_hat <- as.vector(t(beta_hat) %*% t(as.matrix(test_fold[,-1])))

        residual_vector = as.vector(test_fold[,1]) - y_hat
        rss <- sum(residual_vector^2)/(length(y_hat))

        cv = cv+rss

      }

      cv = cv/Nfolds
      cv_rate <- c(cv, cv_rate)

      if(min(cv_rate)==cv){
        best_model = colnames(Z[,-1])
        cv_rate_bestmodel = cv
      }

      Number_Of_Features <- c((ncol(Z)-1), Number_Of_Features)
    }
  }

  output <- list("Minimum CV rate" = cv_rate_bestmodel,
                 "Features selected" = as.vector(best_model))
  plot(x=Number_Of_Features,cv_rate,
       xlab="Number of features",
          ylab="Cross validation rate",
                main = "MSE vs Cross validation rate")
  return(output)
}
data(swiss)
```

```
Y = swiss$Fertility
X = swiss[,-c(1)]
Nfolds = 5

feature_selection(X,Y,Nfolds)
```

By using package k fold of cv

```
#By using package k fold of cv
#this is to perform one model
model<-glm(swiss~Fertility,data=swiss)
mse_cv<-cv.glm(data=swiss,model)#this gives the crossvalidation values
mse_cv<-cv.glm(data = swiss,model)$delta[1]#gives MSE value for the model
#k folds like 10folds and 5 folds we need to use K = 5 or 10
library(boot)
Msevalue= Null()
for (i in 1:10) {
  model<-glm(swiss~ploy(Fertility),i)#ploy gives fertility power,i will take power values
  Msevalue[i]=cv.glm(data = swiss,model,k= 10)$delta[1]

}
Msevalue
```

3.1&3.2-2method-k-fold cross validation

```
subset_function <- function(X,Y,N){
# X = swiss[,2:6]
# Y = swiss[,1:1]
# N = 5
df <- cbind(X,Y)
temp <- NULL
final <- NULL
for(i in 1:NCOL(X)){
combs <- as.data.frame(gtools::combinations(NCOL(X), r=i, v=colnames(X), repeats.allowed=FALSE))
combs <- tidyr::unite(combs, "formula", sep = ",")
temp <- rbind(combs, temp)
}
set.seed(12345)
df2 <- df[sample(nrow(df)),]
df2$k_fold <- sample(N, size = nrow(df), replace = TRUE)
result <- NULL
for (j in 1:NROW(temp))
{
for(i in 1:N){
train = df2[df2$k_fold != i,]
test = df2[df2$k_fold == i,]
vec <- temp[j,]
train_trimmed = lapply(strsplit(as.character(vec), ","), function(x) train[x])[[1]]
test_trimmed = lapply(strsplit(as.character(vec), ","), function(x) test[x])[[1]]
y_train = train[,c("Y"), drop = FALSE]
```

```
y_test = test[,c("Y"), drop = FALSE]
train_trimmed = as.matrix(train_trimmed)
test_trimmed = as.matrix(test_trimmed)
y_test = as.matrix(y_test)
y_train = as.matrix(y_train)
t_train = as.matrix(t(train_trimmed))
t_test = as.matrix(t(test_trimmed))
betas = solve(t_train %*% train_trimmed) %*% (t_train %*% y_train)
train_trimmed = as.data.frame(train_trimmed)
test_trimmed = as.data.frame(test_trimmed)
train_trimmed$type = "train"
test_trimmed$type = "test"
final <- rbind(train_trimmed, test_trimmed)
y_hat_val = as.matrix(final[,1:(ncol(final)-1)]) %*% betas
mse = (Y - y_hat_val)^ 2
data <- cbind(i, vec, mse, type = final$type)
result <- rbind(data, result)
}}
result <- as.data.frame(result)
colnames(result) <- c("kfold", "variables", "mse", "type")
result$mse <- as.numeric(result$mse)
result$no_variables <- nchar(as.character(result$variables)) - nchar(gsub('\\,', "", result$variables))
variable_performance <- result_test %>% group_by(kfold, no_variables) %>%
summarise(MSE = mean(mse, na.rm = TRUE))
myplot <- ggplot(data = variable_performance, aes(x = no_variables, y = MSE, color=kfold)) +
geom_line() + ggtitle("Plot of MSE(test) vs. Number of variables")
myplot2 <- ggplot(data = result_test, aes(x = variables, y = mse, color=kfold)) +
geom_bar(stat="identity") + ggtitle("Plot of MSE(test) vs. Features by folds") + coord_flip()
best_variables <- result_test %>% group_by(variables) %>%
summarise(MSE = mean(mse, na.rm = TRUE)) %>% arrange(MSE) %>%
select(variables) %>% slice(1) %>% as.vector()
return(list(myplot, myplot2, best_variables))
}
subset_function(X = swiss[,2:6], Y = swiss[,1:1], N = 5)
```

From the plot, it is clear that as number of features increases the variance in cross validation rate is decreasing and also the cross validation rate reduces to smaller values which means the error rates are decreasing and model efficiency is increasing.

We used the swiss data which gives information on fertility rates in switzerland.

The features selected for the best model are :

Agriculture, Education, Catholic and Infant mortality along with the intercept. This feature selection can be justified as

1. Agriculture: The income from agriculre can affect fertility rate and higer the income, the socio economic conditions becomes better which in turn supports higher fertility and vice versa
2. Education: As more people are educated, the knowledge on factors affecting fertility impact fertility
3. Catholic: Catholic religion also affects fertility positively due to the preaching given by the religion
4. Infant mortality: Infant mortality rate directly affects fertility. As mortality rate increases fertility decreases

Also from feature selection, the feature examination is not selected and it can be justified as a person writing or clearing an examination does not affect fertility.

Assignment 4. Linear regression and regularization

The Excel file tecator.xlsx contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry

1. Import data to R and create a plot of Moisture versus Protein. Do you think that these data are described well by a linear model?
2. Consider model ???????????????? in which Moisture is normally distributed, and the expected Moisture is a polynomial function of Protein including the polynomial terms up to power ???????? (i.e M1 is a linear model, M2 is a quadratic model and so on). Report a probabilistic model that describes ????????????????. Why is it appropriate to use MSE criterion when fitting this model to a training data?
3. Divide the data into training and validation sets(50%/50%) and fit models M????????????????,????????=1.6. For each model, record the training and the validation MSE and present a plot showing how training and validation MSE depend on i (write some R code to make this plot). Which model is best according to the plot? How do the MSE values change and why? Interpret this picture in terms of bias-variance tradeoff. Use the entire data set in the following computations:
4. Perform variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors by using stepAIC. Comment on how many variables were selected.
5. Fit a Ridge regression model with the same predictor and response variables. Present a plot showing how model coefficients depend on the log of the penalty factor ???????? and report how the coefficients change with ????????.
6. Repeat step 5 but fit LASSO instead of the Ridge regression and compare the plots from steps 5 and 6. Conclusions?
7. Use cross-validation to find the optimal LASSO model (make sure that case ????????=0 is also considered by the procedure) , report the optimal ???????? and how many variables were chosen by the model and make conclusions. Present also a plot showing the dependence of the CV score and comment how the CV score changes with ????????.
8. Compare the results from steps 4 and 7.

4.1-linear model

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

```
#linearmodel
p<-ggplot(tecator, aes(x= Protein, y=Moisture))+
                      geom_point()+
                  xlab("Protein")+ylab("Moisture")+
                  ggtitle("Moisture vs Protein")
p
```

The plot looks linear by seeing the distribution of scatter points. A linear relationship is observed where as the protein increases, moisture is also increasing.

4.2-proof-The probabilistic model

The data can be fit using a polynomial function below:

$y(x, w) = \sum_{j=0}^{6} w_j x^j$ where x is the protein variable and $w = (w_0.....w_6)$ are the coefficients

Here, it is appropriate to use MSE criterion while fitting the model to training data because, the value of the coefficients in the above equation can be determined by minimising the error function and mean squared error function is one of the most simple and easily calculated error function.

Assuming the observed output values as $t = (t_1.....t_6)^T$

We can minimize the error function and select values for coefficients that have the minimum value using :

$E(w) = \frac{1}{2} \sum_{n=1}^{6} y((x_n, w) - t_n)^2$

The probabilistic model

The goal here is to make predictions for the Moisture values based on a set of protein values as inputs. The probability distribution of the target variables can be expressed as below

$p(t|x, w, \beta) = \mathcal{N}(t|y(x, w), \beta^{-1})$

The above distribution explains that the output values has a gaussian distribution with mean $y(x, w)$ and variance β^{-1}

To find the value of the coefficients can be derived using maximum likelihood. The likelihood function can be defined as below

$p(t|x, w, \beta) = \prod_{n=1}^{6} \mathcal{N}(t_n|y(x_n, w), \beta^{-1}$

maximizing the logarithm of above likelihood function gives us the coefficients

$lnp(t|x, w, \beta) = -\beta/2 \sum_{j=1}^{6} (y(x_n, w) - t_n)^2$

Maximising the above function gives the polynomial coeffieients to be used with various polynomial degrees of protein to obtain the result.

4.3-Each model validation and MSE-mycode &groupcode

```
library(gally)
#mycode
protein<-tecator_data %>%#Dividingthe data50/50
select(Moisture,Protein) %>%
mutate(
Protein2 = Protein^2,
Protein3 = Protein^3,
Protein4 = Protein^4,
Protein5 = Protein^5,
Protein6 = Protein^6
)
#divide the data
n=dim(protein)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
trainp=protein[id,]
testp=protein[-id,]
#Train & Test-MSE for
#protein model 0
```

```r
protein_model_0<-lm(Moisture~ Protein , data = trainp)
proteintrain_0MSE <- mean(protein_model_0$residuals^2)
protein_model0predict <- predict(protein_model_0,testp)
protein_test_0MSE<-sum(((protein_model0predict - testp$Moisture)^2))/nrow(testp)

#protein model 1
protein_model_1<-lm(Moisture~ Protein + Protein2, data = trainp)
proteintrain_1MSE <- mean(protein_model_1$residuals^2)
protein_model1predict <- predict(protein_model_1,testp)
protein_test_1MSE<- sum(((protein_model1predict - testp$Moisture)^2))/nrow(testp)

#protein model 2
protein_model_2<-lm(Moisture~ Protein + Protein2 + Protein3, data = trainp)
proteintrain_2MSE <- mean(protein_model_2$residuals^2)
protein_model2predict <- predict(protein_model_2,testp)
protein_test_2MSE<- sum(((protein_model1predict - testp$Moisture)^2))/nrow(testp)

#protein model 3
protein_model_3<-lm(Moisture~ Protein + Protein2 + Protein3 + Protein4, data = trainp)
proteintrain_3MSE <- mean(protein_model_3$residuals^2)
protein_model3predict <- predict(protein_model_3,testp)
protein_test_3MSE<- sum(((protein_model3predict - testp$Moisture)^2))/nrow(testp)

#protein model 4
protein_model_4<-lm(Moisture~ Protein + Protein2 + Protein3 + Protein4 + Protein5,data = trainp)
proteintrain_4MSE <- mean(protein_model_4$residuals^2)
protein_model4predict <- predict(protein_model_4,testp)
protein_test_4MSE<- sum(((protein_model4predict - testp$Moisture)^2))/nrow(testp)

#protein model 5
protein_model_5<-lm(Moisture~ Protein+Protein2 +Protein3+Protein4 +Protein5+Protein6,data = trainp)
proteintrain_5MSE <- mean(protein_model_5$residuals^2)
protein_model5predict <- predict(protein_model_5,testp)
protein_test_5MSE<- sum(((protein_model5predict - testp$Moisture)^2))/nrow(testp)
#plotMSE & model complexity
a <- c(protein_test_0MSE,protein_test_1MSE,protein_test_2MSE,protein_test_3MSE,
       protein_test_4MSE,protein_test_5MSE)
b <- c(proteintrain_0MSE,proteintrain_1MSE,proteintrain_2MSE,proteintrain_3MSE,
       proteintrain_4MSE,proteintrain_5MSE)
protein_plot<-data.frame(a,b)

ggplot(protein_plot) + geom_line(aes(y=a,x=1:6),colour='red') + geom_line(aes(y=b,x=1:6),colour='blue')

#groupcode
tecator$Protein2 <- tecator$Protein^2
tecator$Protein3 <- tecator$Protein^3
tecator$Protein4 <- tecator$Protein^4
tecator$Protein5 <- tecator$Protein^5
tecator$Protein6 <- tecator$Protein^6

n=dim(tecator)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
```

```
tecator_train=tecator[id,]
tecator_test=tecator[-id,]


formula_list <- c(Moisture~Protein,Moisture~Protein+Protein2,
                  Moisture~Protein+Protein2+Protein3,
                  Moisture~Protein+Protein2+Protein3+Protein4,
                  Moisture~Protein+Protein2+Protein3+Protein4+Protein5,
                  Moisture~Protein+Protein2+Protein3+Protein4+Protein5+Protein6)



fitlist <- lapply(formula_list, function(x){

lm(x, data = tecator_train)
})


MSE_train <- sapply(fitlist , function(x){

  sum((x$fitted.values - tecator_train$Moisture)^2)/nrow(tecator_train)
})


MSE_test <- sapply(fitlist , function(x){

  y_hat_test = predict(x, tecator_test)
  sum((y_hat_test - tecator_test$Moisture)^2)/nrow(tecator_test)
})

MSE_data <- data.frame("order"= 1:length(MSE_train),"train" = MSE_train,
                       "test" = MSE_test)
ggplot(data = MSE_data)+geom_line(aes(x = order, y = MSE_train), colour = "Blue")+
  geom_line(aes(x = order, y=MSE_test), color = "Red")+
  xlab("Order of Polynomial") + ylab("Mear squared error")
```

4.3-2method

```
final_data <- tecator_data
magic_function <- function(df, N)
{
df2 <- df
for(i in 2:N)
{
df2[paste("Protein_",i,"_power", sep="")] <- (df2$Protein)^i
}
df2 <- df2[c("Protein_2_power", "Protein_3_power",
"Protein_4_power", "Protein_5_power",
"Protein_6_power")]
df <- cbind(df,df2)
return(df)
}
```

```
final_data <- magic_function(final_data, 6)
set.seed(12345)
n = NROW(final_data)
id = sample(1:n, floor(n*0.5))
train = final_data[id,]
test = final_data[-id,]
# model building
M_1 <- lm(data = train, Moisture~Protein)
M_2 <- lm(data = train, Moisture~Protein+Protein_2_power)
M_3 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power)
M_4 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
Protein_4_power)
M_5 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
Protein_4_power+Protein_5_power)
M_6 <- lm(data = train, Moisture~Protein+Protein_2_power+Protein_3_power+
Protein_4_power+Protein_5_power+Protein_6_power)
train$type <- "train"
test$type <- "test"
final_data <- rbind(test, train)
# predicting new values
M_1_predicted <- predict(M_1, newdata = final_data)
M_2_predicted <- predict(M_2, newdata = final_data)
M_3_predicted <- predict(M_3, newdata = final_data)
M_4_predicted <- predict(M_4, newdata = final_data)
M_5_predicted <- predict(M_5, newdata = final_data)
M_6_predicted <- predict(M_6, newdata = final_data)
# calculating the MSE
final_data$M_1_error <- (final_data$Moisture - M_1_predicted)^2
final_data$M_2_error <- (final_data$Moisture - M_2_predicted)^2
final_data$M_3_error <- (final_data$Moisture - M_3_predicted)^2
final_data$M_4_error <- (final_data$Moisture - M_4_predicted)^2
final_data$M_5_error <- (final_data$Moisture - M_5_predicted)^2
final_data$M_6_error <- (final_data$Moisture - M_6_predicted)^2
# Chainning like Chainsaw
final_error_data <- final_data %>% select(type, M_1_error, M_2_error, M_3_error,
M_4_error, M_5_error, M_6_error) %>%
gather(variable, value, -type) %>%
separate(variable, c("model", "power", "error"), "_") %>%
group_by(type, power) %>%
summarise(MSE = mean(value, na.rm=TRUE))
ggplot(final_error_data, aes(x = power, y = MSE, color=type)) + geom_point() +
ggtitle("Mean squared error vs. model complexitiy by dataset type")
```

According to the plot, when the polynomial order is 1 and 2, the bias is high but the variance is less. So from this we can say that the model here is simple and will not be able to capture underlying patterns as described by data. When polynomial order increses further beyond 2, the variance is high but is stabilised at a particular value until the order of polynomial increases upto 5. Beyond 5, both bias and varience increases. So from this we can conclude that, when the order of the polynomial is beyond 5 overfitting happens.

When we take bias - variance tradeoff into consideration , model with order of polynomial from 3 to 5 gives the better model

4.4step AIC-linera model by using AIc

```
library(MASS)
formula_t = as.formula(paste("Fat~", paste(names(tecator[2:101]),
collapse = "+")))
fit <- lm(formula = formula_t,data=tecator)
step <- stepAIC(fit, direction="backward")
selcted_variables = step$anova
summary(step$anova)
mycode by using this i got 63 features
library(MASS)
library(dplyr)
#Filtering  the data
tecator_data1<-tecator_data[,2:102]
#perform variable selection of a linear model
Fat_lm <- lm(Fat~.,data = tecator_data1)
#predictors by using stepAIC
AIC_fat <-stepAIC(Fat_lm,direction = "both" ,trace = FALSE)
# anova will display final variables
AIC_fat$anova
```

```
#2method
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)
```

38 variables are selected. So if we select the 38 features which are obtained using the above algorithm and model a regression algorithm, the model will be as good in predicting the output as the model which uses all 100 variables. Here on trade of between goodness of fit and simplicity of model, a model with 38 features was giving the best result

4.5-Ridge-regression-alpha=0

```
#mycode
#libraries
library(glmnet)
#Ridge-regression plot
Channel_ridge=scale(tecator_data[,2:101])
Fat_ridge=scale(tecator_data[,102])
Ridge_model=glmnet(as.matrix(Channel_ridge),
Fat_ridge, alpha=0,family="gaussian")
plot(Ridge_model, xvar="lambda", label=TRUE)
#groupcode
reg_data = tecator[,2:102]#tectator is the data
reg_data=scale(reg_data)
covariates=reg_data[,1:100]
response=reg_data[, 101]
model_ridge= glmnet(as.matrix(covariates), response,
                    alpha=0,family="gaussian")
plot(model_ridge, xvar="lambda", label=TRUE)
#2-method
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)
table_cofe <- head(result, 10) %>% select(Channel1, Channel2, Channel84, Channel62,
Channel53, Channel75, Channel57,Protein,
lambda)
knitr::kable(table_cofe, caption = "Coefficent of Ridge Regression vs. Lambda")
```

Analysis: The idea of lasso and ridge regression is to introduce bias to variables in order to reduce/account for multicollinearity. Introducing bias (lambda) to covarience matrix is done by multiplying the diagonal elements by lambda(often 1+lambda), this inflates the covarience of predictors compared to correleations of predictors. The idea is to test the stability of betas that is how likely are the betas/coefficents of regressions to be stable if we keep introducing bias. We can clearly see that 'Channel1' and 'Channel2' betas go from positive to negative with very little bias introduced while terms like 'Channel75' dont change the beta signs. Thus the practice is exclude the terms whose beta/coefficent dont change drastically much within say first 10 introduction of lambda. We see that many of the terms/coefficent tend to zero at around log(lambda) that is ~5. In the above plot, as the lambda value increases, the coefficients are heavily penalized and the effect of all the coeeficients other than intercept reduces to zero (not absolute sero) when Log Lambda reaches 4. So when lamda reached 4, all the coefficients shrink to abosilute zero.

4.6-Lasso regression model alpha=1

```
#groupcode
model_lasso= glmnet(as.matrix(covariates), response,
alpha=1,family="gaussian")
plot(model_lasso, xvar="lambda", label=TRUE)

#mycode
#Lasso-regression plot
Channel_lasso=scale(tecator_data[,2:101])
Fat_lasso=scale(tecator_data[,102])
Lasso_model=glmnet(as.matrix(Channel_lasso),Fat_lasso, alpha=1,family="gaussian")
plot(Lasso_model, xvar="lambda", label=TRUE)
#2method
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")
```

When we compare the plot of LASSO with the plot of Ridge regression, as labda increases the coefficients of less relevent features are set to 0 and are not considered and select the other features. So this method of shrinkage can be used for feature selection which is absent in ridge regression.

Analysis: We quickly see that very little introduction of penalisation/bias is all it takes to make many terms/coefficent to zero. This implies for the full dataset lasso is much better suited for regularisation compared to ridge. At lambda around 1 (log lambda is 0) we get only two or three non zero terms.

4.7-Crossvalidation to find optimal LASSO

```
#group code
model_lasso1=cv.glmnet(as.matrix(covariates), response, alpha=1,
                       family="gaussian",lambda=seq(0,1,0.001))

lambda_min = model_lasso1$lambda.min
plot(model_lasso1)
#my code
#Useing-cross-validation to find the optimal LASSO model
Channel_lasso_cross=scale(tecator_data[,2:101])
Fat_lasso_cross=scale(tecator_data[,102])
Lasso_crossmodel=cv.glmnet(as.matrix(Channel_lasso_cross),
Fat_lasso_cross, alpha=1,family="gaussian")
plot(Lasso_crossmodel)
coef(Lasso_crossmodel, s="lambda.min")
#2method
#find the best lambda from our list via cross-validation
lambda_lasso <- 10^seq(10, -2, length = 100)
lambda_lasso[101] <- 0
lasso_cv <- cv.glmnet(x,y, alpha=1, lambda = lambda_lasso, type.measure="mse")
coef(lasso_cv, lambda = lasso_cv$lambda.min)
lasso_cv$lambda.min
result_lasso <- NULL
for(i in 1:length(lambda_lasso)){
temp <- lasso_cv$cvm[i] %>% as.matrix()
temp <- cbind(CVM_error = temp, lambda = lasso_cv$lambda[i])
result_lasso <- rbind(temp, result_lasso)
}
result_lasso <- result_lasso %>% as.data.frame() %>% arrange(lambda)
colnames(result_lasso) <- c("Cross_Mean_Error", "Lambda")
ggplot(result_lasso, aes(x = log(Lambda), y = Cross_Mean_Error)) + geom_point() +
ggtitle("Cross Validation Error vs. Lambda")
```

Analysis: The minimum value of lambda was 0, implies zero penalisation. The variables selected are: Channel98, Channel99, Channel100, Protein, Moisture, Channel37, Channel38, Channel39, Channel40 along with intercept. We see that Cross validation error is lowest at lambda= 0 and remains low till lambda~1 (log lambda 0) after which the error drastically increases at log(lambda) ~ 2.5, the error maxes out and remains about the same for higher values of lambda. This implies that more bias introduction will lead to worse performance.

4.7comparing step AIC and lasso

In step 4 we are using the function stepAIC which selectes the best model using Akaike information criterion. Here first a model with lowest AIC value is selected and the features or predictors are removed one by one and reached a final model with a set of features which has impact on the model.

In step 7, we use cross validation in LASSO regression. Here we consider a tradeoff between goodness of fit and the error rate and we choose a lambda value which gives us the best model taking the error rate as well

as complexity of model into consideration. So here we can see from the plot that the maximum number of features that the cross validation on LASSO gives us 22 which is lesser than 38 obtained in step 7(Using AIC). In AIC all the variables impacting the model are taken where as in LASSO a trade-off is made where we compromise on few features to minimise the error.

loglikelihood

Assignment 2. Inference about lifetime of machines

The data file machines.xlsx contains information about the lifetime of certain machines, and the company is interested to know more about the underlying process in order to determine the warranty time. The variable is following: . Length: shows lifetime of a machine 1. Import the data to R.

2. Assume the probability model $p(x/\theta) = \theta e^{-\theta x}$ for x=Length in which observations are independent and identically distributed. What is the distribution type of x? Write a function that computes the log-likelihood log $p(x/\theta)$ for a given ???????? and a given data vector x. Plot the curve showing the dependence of log-likelihood on θ where the entire data is used for fitting. What is the maximum likelihood value of θ according to the plot?

3. Repeat step 2 but use only 6 first observations from the data, and put the two log-likelihood curves (from step 2 and 3) in the same plot. What can you say about reliability of the maximum likelihood solution in each case?

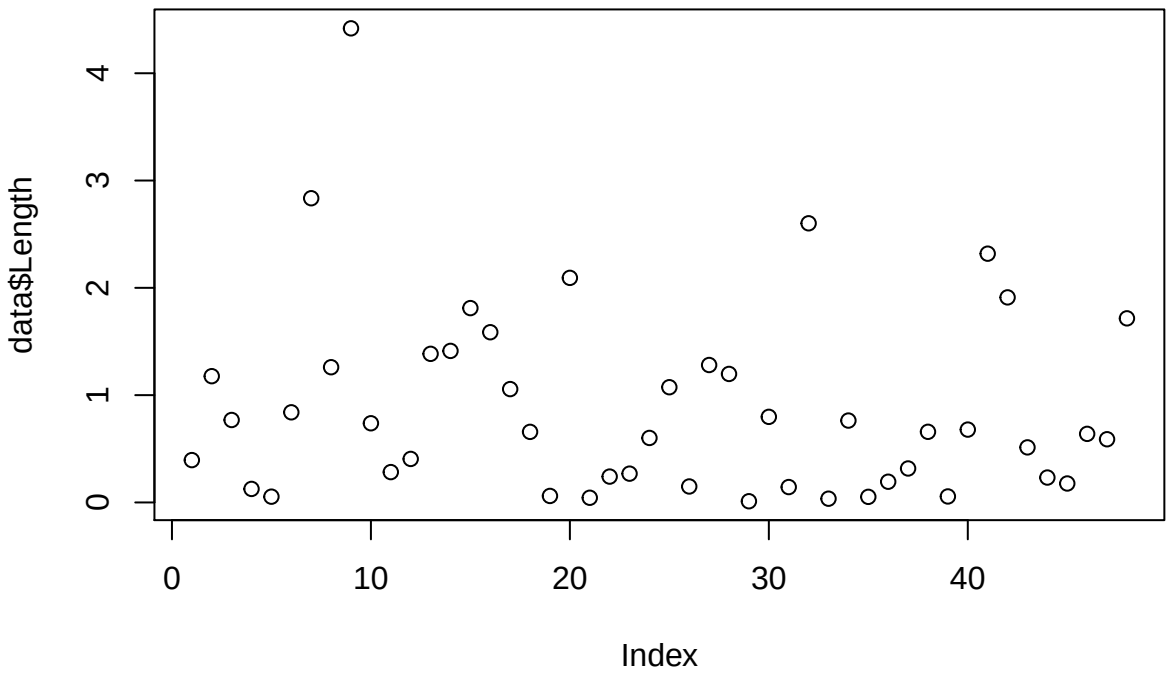
4. Assume now a Bayesian model with $p(x/\theta) = \theta e^{-\theta x}$ and prior = $(p(\theta) = \lambda e^{-\lambda\theta}, \lambda = 10$ Write a function computing $I(\theta) = log(p(x/\theta).p(\theta))$ What kind of measure is actually computed by this function? Plot the curve showing the dependence of $I(\theta)$ on θ???????? computed using the entire data and overlay it with a plot from step 2. Find an optimal θ and compare your result with the previous findings.

5. Use θ ???????? value found in step 2 and generate 50 new observations from $p(x/\theta) = \theta e^{-\theta x}$ (use standard random number generators). Create the histograms of the original and the new data and make conclusions.

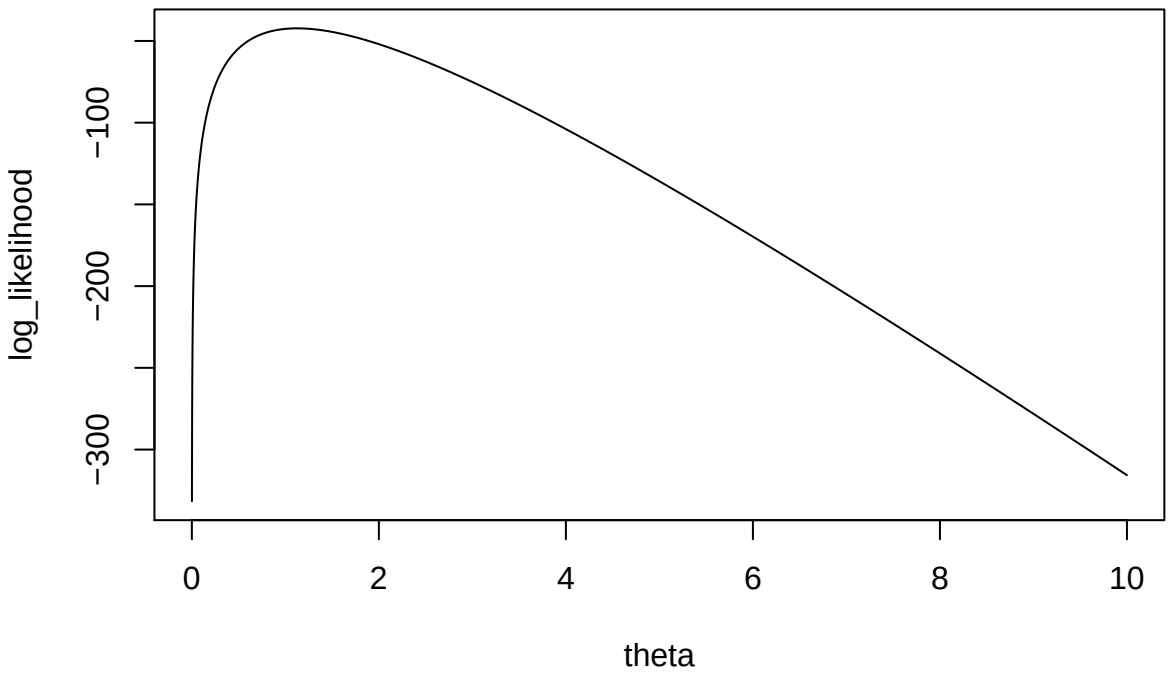
```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.5.3
```

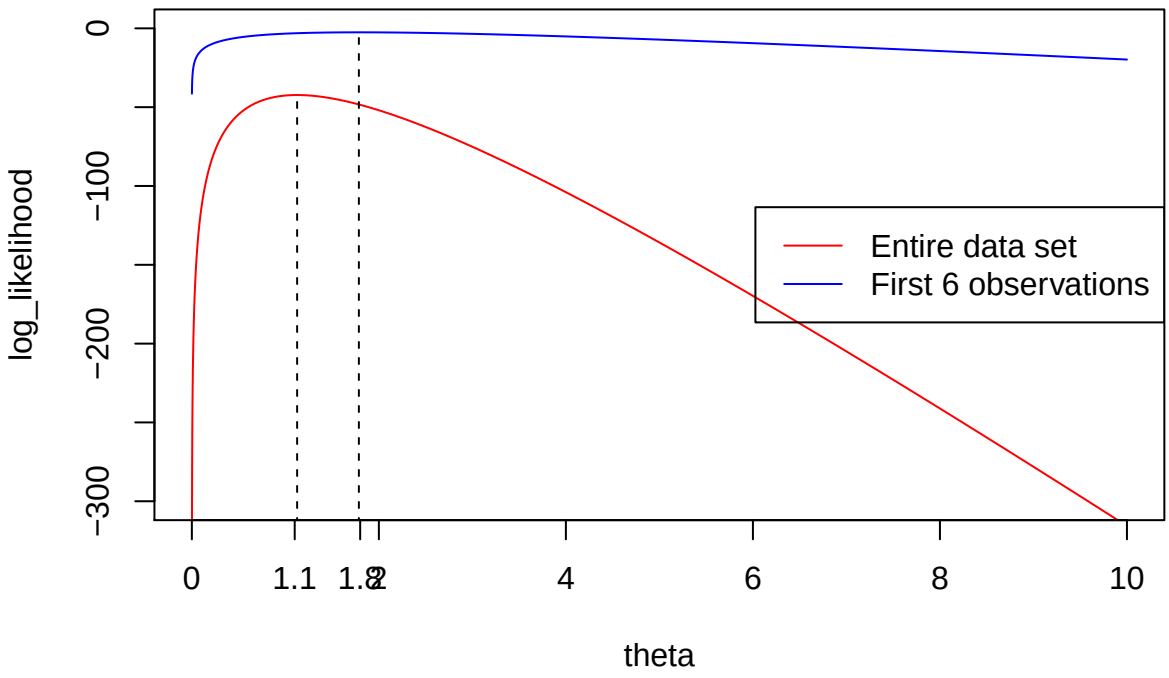
```
data <- read_excel("machines.xlsx")
plot(data$Length)
```



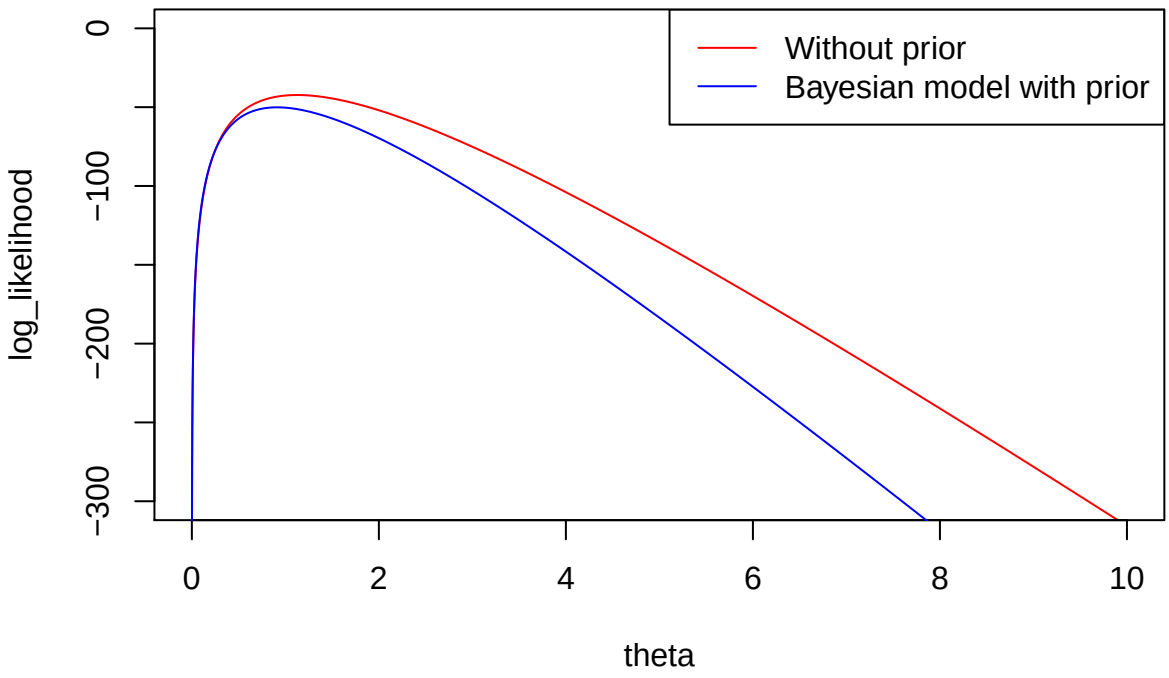

```
length = dim(data)[1]
x = c(data$Length)
theta <- seq(0, 10, 0.001)
log_likelihood <- length*log(theta) - theta*sum(x)
plot(theta, log_likelihood, type = 'l')
```




```
first_6 <- x[1:6]
log_likelihood_2 <- 6*log(theta) - theta*sum(first_6)
plot(theta, log_likelihood, type = 'l', ylim = c(-300,0), col = "red")
lines(theta, log_likelihood_2, type = 'l', col = "blue")
best_theta = theta[which.max(log_likelihood)]
best_theta_2 = theta[which.max(log_likelihood_2)]
lines(c(best_theta, best_theta), c(-1000, max(log_likelihood)), lty=2)
lines(c(best_theta_2, best_theta_2), c(-1000, max(log_likelihood_2)), lty=2)
axis(1, at=round(best_theta_2, 1))
axis(1, at=round(best_theta, 1))
legend("right", legend=c("Entire data set", "First 6 observations"),
col=c("red", "blue"), lty=c(1,1), cex)
```




```
l <- length*log(theta) - theta * sum(x) + log(10) - 10 * theta
plot(theta, log_likelihood, type = 'l', ylim = c(-300,0), col = "red")
lines(theta, l, type = 'l', col="blue")
legend("topright", legend=c("Without prior", "Bayesian model with prior"),
col=c("red", "blue"), lty=c(1,1), cex)
```




```
best_theta2 = theta[which.max(l)]
set.seed(12345)
random <- rexp(50, best_theta)
#hist(random)
```

```
#hist(data$Length)
```

lab1block2

Prudhvi Peddmallu

23 April 2019

1. ENSEMBLE METHODS

The file spambase.csv contains information about the frequency of various words, characters, etc. for a total of 4601 e-mails. Furthermore, these e-mails have been classified as spams (spam = 1) or regular e-mails (spam = 0). You can find more information about these data at. Your task is to evaluate the performance of Adaboost classification trees and random forests on the spam data. Specifically, provide a plot showing the error rates when the number of trees considered are 10; 20; : : : ; 100. To estimate the error rates, use 2/3 of the data for training and 1/3 as hold-out test data.

To learn Adaboost classification trees, use the function blackboost() of the R package mboost. Specify the loss function corresponding to Adaboost with the parameter family. To learn random forests, use the function randomForest of the R package randomForest. To load the data, you may want to use the following code: sp <- read.csv2("spambase.csv") sp$Spam < $-as.factor(sp$Spam)

Adaboost classification trees & randomForest.

```
library(mboost)
library(randomForest)
library(ggplot2)
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)
n=dim(sp)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
sp_train=sp[id,]
sp_test=sp[-id,]
n_trees <- seq(10, 100, by = 10)
```

```
residualerror_ab <- sapply(n_trees, function(x){
model = blackboost(Spam~., sp_train,
family = AdaExp(),control = boost_control(mstop = x))
yhat <- predict(model, sp_test, type = "class")
c_matrix <- table(yhat,sp_test$Spam)
1-(sum(diag(c_matrix))/sum(c_matrix))
})
ab_errors <- data.frame("ntrees" = n_trees, "error" = residualerror_ab)
ggplot(data = ab_errors, aes(x = n_trees, y = error))+geom_point()+geom_line() +
xlab("Number of Trees") + ylab("Error") + ggtitle("Number of Trees vs Error")
```

```
residualerror_rf <- sapply(n_trees, function(x){
model <- randomForest(Spam~., sp_train, importance=TRUE,
proximity=TRUE,ntree = x)
yhat <- predict(model, sp_test, type = "class")
c_matrix <- table(yhat,sp_test$Spam)
1-(sum(diag(c_matrix))/sum(c_matrix))
})
```

```
rf_errors <- data.frame("ntrees" = n_trees, "error" = residualerror_rf)
ggplot(data = rf_errors, aes(x = n_trees, y = error))+geom_point()+geom_line() +
xlab("Number of Trees") + ylab("Error") + ggtitle("Number of Trees vs Error")
```

2-method

Ada boost classification

```
#libraries
library(mboost)
```

```
#Divideing the data into train(2/3=70%) and test(1/3=30%)
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train=spambase[id,]
test=spambase[-id,]
```

```
#Ada boost classification
#loop for sequence to get order 10, 20, 30...100
storeing<- c()
sequences <- c()
count<-0
  for(i in seq(from=10, to=100 ,by =10))
  {
     sequences<-c(i,sequences)
#fitting the model adaboost
spambase_EM_fit <- blackboost(Spam~., data = train, family = AdaExp(),control = boost_control(mstop =i))
#prediction we are giving type = class because the adaboost gives the tree contain the classes
predectionEM_train <- predict(spambase_EM_fit,newdata=train,type='class')
#confusion matrix
confusionmatrix_trainEM = table(train$Spam,predectionEM_train)
# misclassification errors according to i(10,20,30...)
diagonal_trainEM<-diag(confusionmatrix_trainEM)
summ_trainEM<-sum(confusionmatrix_trainEM)
misclassificationrate_trainEM<-1-(sum(diagonal_trainEM)/sum(summ_trainEM))
storeing<-c(misclassificationrate_trainEM,storeing)

}
#plot the misclassification vs i
{plot(x=sequences,y=storeing,
xlab="sequence models",
ylab="misclassification errors"
)
lines(sequences,storeing,col="blue")}
```

```
library(randomForest)
```

```
#Divideing the data into train(2/3=70%) and test(1/3=30%)
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train_RF=spambase[id,]
```

```
test_RF=spambase[-id,]
```

Random forest

```
#Random forest
storeing_RF<-c()
sequences_RF<-c()
count<-0
for(i in seq(from=10,to=100,by=10)){
 sequences_RF<-c(i,sequences_RF)
#fitting the model adaboost
spambase_RF_fit <- randomForest(Spam~., data = train_RF,control = boost_control(mstop =i))
#prediction we are giving type = class because the adaboost gives the tree contain the classes
predectionRF_train <- predict(spambase_RF_fit,newdata=train_RF,type='class')
#confusion matrix
confusionmatrix_trainRF = table(train_RF$Spam,predectionRF_train)
# misclassification errors according to i(10,20,30...)
diagonal_trainRF<-diag(confusionmatrix_trainRF)
summ_trainRF<-sum(confusionmatrix_trainRF)
misclassificationrate_trainRF<-1-(sum(diagonal_trainRF)/sum(summ_trainRF))
storeing_RF<-c(misclassificationrate_trainRF,storeing_RF)
}
#plot the misclassification vs i
{plot(x=sequences_RF,y=storeing_RF,
xlab="sequence models-RF",
ylab="misclassification errors"
)
lines(sequences_RF,storeing_RF,col="purple")  }
```

2-method-Adaboost classification trees & randomForest.

Loading Input files

```
spam_data <- read.csv(file = "spambase.data", header = FALSE)
colnames(spam_data)[58] <- "Spam"
spam_data$Spam <- factor(spam_data$Spam, levels = c(0,1), labels = c("0", "1"))
```

Splitting into Train and Test with 66% and 33% ratio.

```
set.seed(12345)
n =  NROW(spam_data)
id = sample(1:n, floor(n*(2/3)))
train = spam_data[id,]
test = spam_data[-id,]
```

Trainning the Model

Adaboost with varying depth

```
final_result <- NULL
for(i in seq(from = 10, to = 100, by = 10)){
```

```
ada_model <- mboost::blackboost(Spam~.,
                                data = train,
                                family = AdaExp(),
                               control=boost_control(mstop=i))
forest_model <- randomForest(Spam~., data = train, ntree = i)
prediction_function <- function(model, data){
  predicted <- predict(model, newdata = data, type = c("class"))
  predict_correct <- ifelse(data$Spam == predicted, 1, 0)
  score <- sum(predict_correct)/NROW(data)
  return(score)
}
train_ada_model_predict <- predict(ada_model, newdata = train, type = c("class"))
test_ada_model_predict <- predict(ada_model, newdata = test, type = c("class"))
train_forest_model_predict <- predict(forest_model, newdata = train, type = c("class"))
test_forest_model_predict <- predict(forest_model, newdata = test, type = c("class"))
test_predict_correct <- ifelse(test$Spam == test_forest_model_predict, 1, 0)
train_predict_correct <- ifelse(train$Spam == train_forest_model_predict, 1, 0)
train_ada_score <-  prediction_function(ada_model, train)
test_ada_score <-  prediction_function(ada_model, test)
train_forest_score <-  prediction_function(forest_model, train)
test_forest_score <-  prediction_function(forest_model, test)
iteration_result <- data.frame(number_of_trees = i,
                               accuracy = c(train_ada_score,
                                            test_ada_score,
                                            train_forest_score,
                                            test_forest_score),
                               type  = c("train", "test", "train", "test"),
                               model = c("ADA", "ADA",  "Forest", "Forest"))
final_result <- rbind(iteration_result, final_result)
}
final_result$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))
```

2. MIXTURE MODELS-EM

Your task is to implement the EM algorithm for mixtures of multivariate Benoulli distributions. Please use the template in the next page to solve the assignment. Then, use your implementation to show what happens when your mixture models has too few and too many components, i.e. set K = 2; 3; 4 and compare results. Please provide a short explanation as well. ## Description of the EM algorithm EM is an iterative expectation maximumation technique. The way this works is for a given mixed distribution we guess the components of the data. This is done by first guessing the number of components and then randomly initializing the parameters of the said distribution (Mean, Varience).

Sometimes the data do not follow any known probability distribution but a mixture of known distributions such as:

$$p(x) = \sum_{k=1}^{K} p(k).p(x|k)$$

where p(x|k) are called mixture components and p(k) are called mixing coefficients: where p(k) is denoted by

$$\pi_k$$

With the following conditions

$$0 \leq \pi_k \leq 1$$

and

$$\sum_k \pi_k = 1$$

We are also given that the mixture model follows a Bernoulli distribution, for bernoulli we know that

$$Bern(x|\mu_k) = \prod_i \mu_{ki}^{x_i}.(1-\mu_{ki})^{(1-x_i)}$$

The EM algorithm for an Bernoulli mixed model is:

Set pi and mu to some initial values Repeat until pi and mu do not change E-step: Compute p(z|x) for all k and n M-step: Set piˆk to piˆk(ML) from likehood estimate, do the same to mu

M step:

$$p(z_{nk}|x_n, \mu, \pi) = Z = \frac{\pi_k p(x_n|\mu_k)}{\sum_k p(x_n|\mu_k)}$$

E step:

$$\pi_k^{ML} = \frac{\sum_N p(z_{nk}|x_n, \mu, \pi)}{N}$$

$$\mu_{ki}^{ML} = \frac{\sum_n x_{ni} p(z_{nk}|x_n, \mu, \pi)}{\sum_n p(z_{nk}|x_n, \mu, \pi)}$$

The maximum likehood of E step is:

$$\log_e p(X|\mu, \pi) = \sum_{n=1}^{N} \log_e \sum_{k=1}^{K} .\pi_k.p(x_n|\mu_k)$$

Given question code

```
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
```

```
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
# Producing the training data
for(n in 1:N) {
k <- sample(1:3,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[k,d])
}
}
K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
mu[k,] <- runif(D,0.49,0.51)
}
pi
mu
for(it in 1:max_it) {
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
#points(mu[4,], type="o", col="yellow")
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignments
# Your code here
#Log likelihood computation.
# Your code here
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the lok likelihood has not changed significantly
# Your code here
#M-step: ML parameter estimation from the data and fractional component assignments
# Your code here
}
pi
mu
plot(llik[1:it], type="o")
```

Code

To compare the results for K = 2,3,4, the em_loop-function provides a graphical analysis for every iteration. The function includes comments which explain what I did at which step to create the EM algorithm. The function will be finally run with K = 2,3,4.

```
em_loop = function(K) {
# Initializing data
```

```
set.seed(1234567890)
max_it = 100 # max number of EM iterations
min_change = 0.1 # min change in log likelihood between two consecutive EM iterations
N = 1000 # number of training points
D = 10 # number of dimensions
x = matrix(nrow=N, ncol = D) # training data
true_pi = vector(length = K) # true mixing coefficients
true_mu = matrix(nrow = K, ncol = D) # true conditional distributions
true_pi = c(rep(1/K, K))
if (K == 2) {
true_mu[1,] = c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,] = c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
plot(true_mu[1,], type = "o", xlab = "dimension", col = "blue",
ylim = c(0,1), main = "True")
points(true_mu[2,], type="o", xlab = "dimension", col = "red",
main = "True")
} else if (K == 3) {
true_mu[1,] = c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,] = c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,] = c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
plot(true_mu[1,], type = "o", xlab = "dimension", col = "blue", ylim=c(0,1),
main = "True")
points(true_mu[2,], type = "o", xlab = "dimension", col = "red",
main = "True")
points(true_mu[3,], type = "o", xlab = "dimension", col = "green",
main = "True")
} else {
true_mu[1,] = c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,] = c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,] = c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
true_mu[4,] = c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)
plot(true_mu[1,], type = "o", xlab = "dimension", col = "blue",
ylim = c(0,1), main = "True")
points(true_mu[2,], type = "o", xlab = "dimension", col = "red",
main = "True")
points(true_mu[3,], type = "o", xlab = "dimension", col = "green",
main = "True")
points(true_mu[4,], type = "o", xlab = "dimension", col = "yellow",
main = "True")
}
z = matrix(nrow = N, ncol = K) # fractional component assignments
pi = vector(length = K) # mixing coefficients
mu = matrix(nrow = K, ncol = D) # conditional distributions
llik = vector(length = max_it) # log likelihood of the EM iterations
# Producing the training data
for(n in 1:N) {
k = sample(1:K, 1, prob=true_pi)
for(d in 1:D) {
x[n,d] = rbinom(1, 1, true_mu[k,d])
}
}
# Random initialization of the paramters
pi = runif(K, 0.49, 0.51)
```

```
pi = pi / sum(pi)
for(k in 1:K) {
mu[k,] = runif(D, 0.49, 0.51)
}
#EM algorithm
for(it in 1:max_it) {
# Plotting mu
# Defining plot title
title = paste0("Iteration", it)
if (K == 2) {
plot(mu[1,], type = "o", xlab = "dimension", col = "blue", ylim = c(0,1), main = title)
points(mu[2,], type = "o", xlab = "dimension", col = "red", main = title)
} else if (K == 3) {
plot(mu[1,], type = "o", xlab = "dimension", col = "blue", ylim = c(0,1), main = title)
points(mu[2,], type = "o", xlab = "dimension", col = "red", main = title)
points(mu[3,], type = "o", xlab = "dimension", col = "green", main = title)
} else {
plot(mu[1,], type = "o", xlab = "dimension", col = "blue", ylim = c(0,1), main = title)
points(mu[2,], type = "o", xlab = "dimension", col = "red", main = title)
points(mu[3,], type = "o", xlab = "dimension", col = "green", main = title)
points(mu[4,], type = "o", xlab = "dimension", col = "yellow", main = title)
}
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignments
for (n in 1:N) {
# Creating empty matrix (column 1:K = p_x_given_k; column K+1 = p(x|all k)
p_x = matrix(data = c(rep(1,K), 0), nrow = 1, ncol = K+1)
Calculating p(x|k) and p(x|all k)
for (k in 1:K) {
# Calculating p(x|k)
for (d in 1:D) {
p_x[1,k] = p_x[1,k] * (mu[k,d]^x[n,d]) * (1-mu[k,d])^(1-x[n,d])
}
p_x[1,k] = p_x[1,k] * pi[k] # weighting with pi[k]
# Calculating p(x|all k) (denominator)
p_x[1,K+1] = p_x[1,K+1] + p_x[1,k]
}
# Calculating z for n and all k
for (k in 1:K) {
z[n,k] = p_x[1,k] / p_x[1,K+1]
}
}
# Log likelihood computation
for (n in 1:N) {
for (k in 1:K) {
log_term = 0
for (d in 1:D) {
log_term = log_term + x[n,d] * log(mu[k,d]) + (1-x[n,d]) * log(1-mu[k,d])
}
llik[it] = llik[it] + z[n,k] * (log(pi[k]) + log_term)
}
}
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
```

```
flush.console()
# Stop if the log likelihood has not changed significantly
if (it != 1) {
if (abs(llik[it] - llik[it-1]) < min_change) {
break
}
}
# M-step: ML parameter estimation from the data and fractional component assignments
# Updating pi
for (k in 1:K) {
pi[k] = sum(z[,k])/N
}
# Updating mu
for (k in 1:K) {
mu[k,] = 0
for (n in 1:N) {
    mu[k,] = mu[k,] + x[n,] * z[n,k]
}
mu[k,] = mu[k,] / sum(z[,k])
}
}
# Printing pi, mu and development of log likelihood at the end
return(list(
pi = pi,
mu = mu,
logLikelihoodDevelopment = plot(llik[1:it],
type = "o",
main = "Development of the log likelihood",
xlab = "iteration",
ylab = "log likelihood")
))
}
```

K=2

```
em_loop(2)
```

K=3

```
## K=3
em_loop(3)
```

K=4

```
em_loop(4)
```

Analysis

Comparing the final plots for each of the cases, it becomes clear that when the mixture model has more components (K = 4), the EM algorithm does not perform as accurate as for fewer components (K = 2 or K = 3). The segregation between each component gets diluted as the components get higher.

EM algo with matrix

```
em_mat <- function(k){
set.seed(1234567890)
# max number of EM iterations
max_it <- 100
# min change in log likelihood between two consecutive EM iterations
min_change <- 0.1
#------------- Producing Training data and Initialization ----------------#
# number of training points
N <- 1000
# number of dimensions
D <- 10
# training data
x <- matrix(nrow=N, ncol=D)
# true mixing coefficients
true_pi <- vector(length = k)
true_pi <- rep(1/k, k)
# true conditional distributions
true_mu <- matrix(nrow = k, ncol = D)
if(k == 2){
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
}else if(k == 3){
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
}else {
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
points(true_mu[4,], type="o", col="yellow")
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
true_mu[4,]=c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)}
# Producing the training data
for(n in 1:N) {
l <- sample(1:k,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[l,d])
}
}
# fractional component assignments
z <- matrix(nrow = N, ncol = k)
# mixing coefficients
pi <- vector(length = k)
# conditional distributions
```

```
mu <- matrix(nrow = k, ncol = D)
# log likelihood of the EM iterations
llik <- vector(length = max_it)
# Random initialization of the paramters
pi <- runif(k,0.49,0.51)
pi <- pi / sum(pi)
for(i in 1:k) {
mu[i,] <- runif(D,0.49,0.51)
}
#----------------------- Iteration stage -----------------------#
for(it in 1:max_it) {
if(k == 2){
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
}else if(k == 3){
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
}else{
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
points(mu[4,], type="o", col="yellow")}
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignments
# Updating z matrix
p_Xn_MUn <- exp(x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu)))
numerator <- matrix(rep(pi,N), ncol = k, byrow = TRUE) * p_Xn_MUn
denominator <- rowSums(numerator)
Z_nk <- numerator/denominator
# Updating pi
pi <- colSums(Z_nk)/N
# Updating mu
mu <- (t(Z_nk) %*% x)/colSums(Z_nk)
#Log likelihood computation.
llik[it] <- sum(Z_nk * ((x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu))
) + matrix(rep(pi,N), ncol = k, byrow = TRUE)))
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the log likelihood has not changed significantly
if(it >= 2){
if((llik[it] - llik[it-1]) < min_change){break()}
}
#M-step: ML parameter estimation from the data and fractional component assignments
# pi_ML
pi_ML <- pi
#mu_ML
mu_ML <- mu
}
#----------------------- output stage -----------------------#
df <- data.frame(Iteration = 1:length(llik[which(llik != 0.000)])
, log_likelihood = llik[which(llik != 0.000)])
plot <- ggplot(data = df) +
```

```
geom_point(mapping = aes(x = Iteration, y = log_likelihood),
color = 'black') +
geom_line(mapping = aes(x = Iteration, y = log_likelihood),
color = 'black', size = 1) +
ggtitle('Maximum likelihood vs Number of iterations') +
theme(plot.title = element_text(hjust = 0.5)) +
theme_light()
output <- list(pi_ML = pi_ML,
mu_ML = mu_ML,
plot = plot
)
output
}
EM_2 <- em_mat(2)
EM_2$plot
EM_2$pi_ML
EM_2$mu_ML
EM_3 <- em_mat(3)
EM_3$plot
EM_3$pi_ML
EM_3$mu_ML
EM_4 <- em_mat(4)
EM_4$plot
EM_4$pi_ML
EM_4$mu_ML
```

EM-2 method

```
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
# Producing the training data
for(n in 1:N) {
k <- sample(1:3,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[k,d])
}
}
K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
```

```
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
eachValue <- matrix(nrow=N, ncol=K)
total <- vector(length = N)
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {

mu[k,] <- runif(D,0.49,0.51)
}
pi
mu
for(it in 1:max_it) {
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
points(mu[4,], type="o", col="yellow")
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignments
for (i in 1:N) {
for (j in 1:k) {
p <- prod(dbinom(x[i, ], 1, mu[j, ]))
z[i,j] <- pi[j] * p
eachValue[i,j] <- z[i,j]
}
# Using Baye's Theorem
z[i, ] <- z[i, ] /sum(z[i, ])
# for calculating log likelihood
total[i] <- log(sum(eachValue[i,]))
}
#Log likelihood computation.
llik[it] <- sum(total)
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the lok likelihood has not changed significantly
if(it > 1 && (llik[it]-llik[it-1]<=min_change)){
break
}
#M-step: ML parameter estimation from the data and fractional component assignments
for(j in 1:K)
{
NK <- sum(z[,j])
mu[j,] <- (t(z[,j])%*%x)/sum(z[,j])
pi[j] <- NK/N
}
}
pi
mu
plot(llik[1:it], type="o")
set.seed(1234567890)
# max_it <- 100 # max number of EM iterations
```

```
# min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
# N=1000 # number of training points
# D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
# Producing the training data
for(n in 1:N) {
k <- sample(1:3,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[k,d])
}
}
K=2 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
eachValue <- matrix(nrow=N, ncol=K)
total <- vector(length = N)
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
mu[k,] <- runif(D,0.49,0.51)
}
pi
mu
for(it in 1:max_it) {
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
#points(mu[3,], type="o", col="green")
#points(mu[4,], type="o", col="yellow")
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignments
for (i in 1:N) {
for (j in 1:k) {
p <- prod(dbinom(x[i, ], 1, mu[j, ]))
z[i,j] <- pi[j] * p
eachValue[i,j] <- z[i,j]
}
# Using Baye's Theorem
z[i, ] <- z[i, ] /sum(z[i, ])
# for calculating log likelihood
total[i] <- log(sum(eachValue[i,]))
}
```

```
#Log likelihood computation.
llik[it] <- sum(total)
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the lok likelihood has not changed significantly
if(it > 1 && (llik[it]-llik[it-1]<=min_change)){
break
}
#M-step: ML parameter estimation from the data and fractional component assignments
for(j in 1:K)
{
NK <- sum(z[,j])
mu[j,] <- (t(z[,j])%*%x)/sum(z[,j])
pi[j] <- NK/N
}
}
pi
mu
plot(llik[1:it], type="o")
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
# Producing the training data
for(n in 1:N) {
k <- sample(1:3,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[k,d])
}
}
K=4 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
eachValue <- matrix(nrow=N, ncol=K)
total <- vector(length = N)
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)

pi <- pi / sum(pi)
for(k in 1:K) {
```

```
mu[k,] <- runif(D,0.49,0.51)
}
pi
mu
for(it in 1:max_it) {
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
points(mu[4,], type="o", col="yellow")
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignments
for (i in 1:N) {
for (j in 1:k) {
p <- prod(dbinom(x[i, ], 1, mu[j, ]))
z[i,j] <- pi[j] * p
eachValue[i,j] <- z[i,j]
}
# Using Baye's Theorem
z[i, ] <- z[i, ] /sum(z[i, ])
# for calculating log likelihood
total[i] <- log(sum(eachValue[i,]))
}
#Log likelihood computation.
llik[it] <- sum(total)
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the lok likelihood has not changed significantly
if(it > 1 && (llik[it]-llik[it-1]<=min_change)){
break
}
#M-step: ML parameter estimation from the data and fractional component assignments
for(j in 1:K)
{
NK <- sum(z[,j])
mu[j,] <- (t(z[,j])%*%x)/sum(z[,j])
pi[j] <- NK/N
}
}
pi
mu
plot(llik[1:it], type="o")
```

lab2block1

Prudhvi Peddmallu

23 April 2019

deviance and gene and optimal tree depth and naive bayes classification of roc curves, TPR,FPR,NOn-parametric bootstrap and parametric bootstrap,lossmatric,Principle components.,decesion tree

Assignment 2. Analysis of credit scoring

1. Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.
2. Fit a decision tree to the training data by using the following measures of impurity

a. Deviance
b. Gini index and report the misclassification rates for the training and test data. Choose the measure providing the better results for the following steps.

3. Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data.
4. Use training data to perform classification using Naïve Bayes and report the confusion matrices and misclassification rates for the training and for the test data. Compare the results with those from step 3.
5. Use the optimal tree and the Naïve Bayes model to classify the test data by using the following principle $\hat{Y} = 1 if p(Y = good/X > \pi$, otherwise $\hat{Y} = 0$ π=0.05,0.1,0.15,.0.9,0.95. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?
6. Repeat Naïve Bayes classification as it was in step 4 but use the following loss matrix: predicted observed good 0 1 bad 10 0 and report the confusion matrix for the training and test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
library(readxl)
library(tree)
library(ggplot2)
library(MASS)
library(e1071)
library(boot)
```

2.1divideing the data into test train and validation

Import the data to R and divide into training/validation/test as 50/25/25:

```
creditscore <- readxl::read_xls ("creditscoring.xls")

creditscore1<-read_excel("creditscoring.xls")
```

```
creditscore$good_bad <- as.factor(creditscore$good_bad)


n=dim(creditscore)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
cs_train=creditscore[id,]

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


id3=setdiff(id1,id2)
cs_test=creditscore[id3,]
```

The data was loaded and was divided into training, test and validation samples.

2.2Fit a decision tree-deviance & geni

```
fit_train=tree(good_bad ~., data=cs_train, split = "deviance")

#plot(fit_train)
#text(fit_train, pretty = 0)
#summary(fit_train)
Yfit=predict(fit_train, newdata=cs_test, type="class")
mc_table <- table(cs_test$good_bad,Yfit)
misclass_rate <- 1-sum(diag(mc_table))/sum(mc_table)
fit_train_gini=tree(good_bad ~., data=cs_train, split = "gini")
#summary(fit_train_gini)
Yfit_gini=predict(fit_train_gini, newdata=cs_test, type="class")
mc_table_gini <- table(cs_test$good_bad,Yfit_gini)
misclass_rate_gini <- 1-sum(diag(mc_table_gini))/sum(mc_table_gini)
```

2.3optimal tree depth

Q-Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data.

```
fit_tree=tree(good_bad ~., data=cs_train, split = "deviance")

trainScore=rep(0,15)# because we have 15 leaves intial tree
validscore=rep(0,15)
for(i in 2:15) {
  prunedTree=prune.tree(fit_tree,best=i)
  pred=predict(prunedTree, newdata=cs_valid, type="tree")
  trainScore[i]=deviance(prunedTree)
```

```
  validscore[i]=deviance(pred)
}
df <- data.frame("Tree" = seq(2,15), "Test Score" = validscore[2:15], "Train Score" = trainScore[2:15])
ggplot(df, aes(x=Tree))+geom_point(aes(y=df$Test.Score))+geom_line(aes(y=df$Test.Score), colour = "Red")
  geom_point(aes(y=df$Train.Score))+geom_line(aes(y=df$Train.Score), colour = "Blue")+
  ylab("Train / Validation Score")+xlab("Number of Trees")+ggtitle("Deviance vs Number of Leaves")


optimalTree=prune.tree(fit_tree, best=4)

plot(optimalTree)
title(main = "Optimal Tree")
text(optimalTree)

Yfit_test=predict(optimalTree, newdata=cs_test, type="class")
mc_table_test <- table(cs_test$good_bad,Yfit_test)

misclass_rate <- 1-sum(diag(mc_table_test))/sum(mc_table_test)

cat("Misclassification table\n\n")
mc_table_test
cat("\nMisclassification rate for test data = ", misclass_rate )
```

2.4naivebayes-misclassificate and confusion matrix

```
fit_nb=naiveBayes(good_bad~., data=cs_train, split = "deviance")

Yfit_nb_train = predict(fit_nb, newdata=cs_train)
Yfit_nb_test=predict(fit_nb, newdata=cs_test)

cat("Confusion matrix for train data\n\n")
 table(cs_train$good_bad,Yfit_nb_train)

cat("\n\n Confusion matrix for test data\n\n")
table(cs_test$good_bad,Yfit_nb_test)

m_rate_train <- mean(cs_train$good_bad!=Yfit_nb_train)
m_rate_test <- mean(cs_test$good_bad!=Yfit_nb_test)

cat("\n Misclassification rate for train data = ",m_rate_train)
cat("\n\n Misclassification rate for test data = ",m_rate_test)
```

2.5-tpr-fpr

```
Yfit_nb=predict(fit_nb, newdata=cs_test, type = "raw")
Yfit_tree=predict(optimalTree, newdata=cs_test)


TPR_tree = c()
```

```
FPR_tree = c()
TPR_nb = c()
FPR_nb = c()

pi <- seq(from = 0.05,to = 0.95,by = 0.05)

for(i in 1:length(pi)){
  #optimal tree
  pred_tree <- ifelse(Yfit_tree[,2] > pi[i],"good","bad")
  mc_table_tree <- table(cs_test$good_bad,pred_tree)
  if(ncol(mc_table_tree)==1){
    if(colnames(mc_table_tree)=="good"){
      mc_table_tree <- cbind("bad"=c(0,0), mc_table_tree)
    }else{
      mc_table_tree <- cbind(mc_table_tree,"good"=c(0,0))
    }
  }
  TPR_tree <- c(TPR_tree,mc_table_tree["good","good"]/sum(mc_table_tree["good",]))
  FPR_tree <- c(FPR_tree,mc_table_tree["bad","good"]/sum(mc_table_tree["bad",]))


  #naivebayes
  pred_nb <- ifelse(Yfit_nb[,2] > pi[i],"good","bad")
  mc_table_nb <- table(cs_test$good_bad,pred_nb)
  if(ncol(mc_table_nb)==1){
    if(colnames(mc_table_nb)=="good"){
      mc_table_nb <- cbind("bad"=c(0,0), mc_table_nb)
    }else{
      mc_table_nb <- cbind(mc_table_nb,"good"=c(0,0))
    }
  }
  TPR_nb <- c(TPR_nb,mc_table_nb["good",2]/sum(mc_table_nb["good",]))
  FPR_nb <- c(FPR_nb,mc_table_nb["bad",2]/sum(mc_table_nb["bad",]))

}


plot(FPR_tree,TPR_tree,xlab = "FPR", ylab = "TPR")
title(main = "Optimal Tree")
lines(FPR_tree,TPR_tree)


plot(FPR_nb,TPR_nb, xlab = "FPR", ylab = "TPR")
title(main = "Naive Bayes")
lines(FPR_nb,TPR_nb)
```

2.6-naive bayes for-loss matrix

Naïve Bayes classification as it was in step 4 but use the following loss matrix: predicted observed good 0 1 bad 10 0

```r
fit_nb=naiveBayes(good_bad~., data=cs_train, type = "prob")
Yfit_nb_train_1 <- predict(fit_nb, newdata=cs_train)
mtable_train_1 <- table(Yfit_nb_train_1,cs_train$good_bad)

Yfit_nb_train = predict(fit_nb, newdata=cs_train, type = "raw")
Yfit_nb_train <- as.data.frame(Yfit_nb_train)
output <- rep(NA, nrow(Yfit_nb_train))
Yfit_nb_train <- cbind(output,Yfit_nb_train)

for(i in 1:nrow(Yfit_nb_train)){
  if((Yfit_nb_train[i,"good"]*1) > (Yfit_nb_train[i,"bad"]*10))
  {
    Yfit_nb_train[i,"output"] <- "good"
  }else{
    Yfit_nb_train[i,"output"] <- "bad"
  }
}


Yfit_nb_test=predict(fit_nb, newdata=cs_test, type = "raw")
Yfit_nb_test_1 <- as.data.frame(Yfit_nb_test)
output <- rep(NA, nrow(Yfit_nb_test_1))
Yfit_nb_test_1 <- cbind(output,Yfit_nb_test_1)

for(i in 1:nrow(Yfit_nb_test_1)){
  if((Yfit_nb_test_1[i,"good"]*1) > (Yfit_nb_test_1[i,"bad"]*10))
  {
    Yfit_nb_test_1[i,"output"] <- "good"
  }else{
    Yfit_nb_test_1[i,"output"] <- "bad"
  }
}

mc_table_train <- table(cs_train$good_bad,Yfit_nb_train$output)
mc_table_test <- table(cs_test$good_bad,Yfit_nb_test_1$output)
m_rate_train <- 1-sum(diag(mc_table_train))/sum(mc_table_train)
m_rate_test <- 1-sum(diag(mc_table_test))/sum(mc_table_test)
```

2-ass-2group code

```r
#library
library(readxl)
library(tree)
```

```r
#Reading the data
private_enterprise <-read_excel("creditscoring.xls")
private_enterprise<-na.omit(private_enterprise)
private_enterprise$good_bad = as.factor(private_enterprise$good_bad)
```

deviance train decesion tree

```
fit_model_deviance_train=tree(formula = good_bad~., data=train,split = "deviance")
predection_trainD <-predict(fit_model_deviance_train,newdata=train,
                            type="class")
confusionmatrix_trainD<-table( train$good_bad , predection_trainD)
diagonal_trainD<-diag(confusionmatrix_trainD)
summ_trainD<-sum(confusionmatrix_trainD)
misclassificationrate_trainD<-1-(sum(diagonal_trainD)/sum(summ_trainD))
misclassificationrate_trainD
plot(fit_model_deviance_train)
text(fit_model_deviance_train, pretty=0)
fit_model_deviance_train
summary(fit_model_deviance_train)
```

gini train decesion tree do for test

```
fit_model_gini_train=tree(formula = good_bad~., data=train,split = "gini")
predection_trainG <-predict(fit_model_gini_train,newdata=train,
                            type="class")
confusionmatrix_trainG<-table( train$good_bad , predection_trainG)
diagonal_trainG<-diag(confusionmatrix_trainG)
summ_trainG<-sum(confusionmatrix_trainG)
misclassificationrate_trainG<-1-(sum(diagonal_trainG)/sum(summ_trainG))
misclassificationrate_trainG
plot(fit_model_gini_train)
text(fit_model_gini_train, pretty=0)
fit_model_gini_train
summary(fit_model_gini_train)
```

2.2-gini test decesion tree

```
fit_model_gini_test=tree(formula = good_bad~., data=test,split = "gini")
predection_testG <-predict(fit_model_gini_test,newdata=test,
                            type="class")
confusionmatrix_testG<-table( test$good_bad , predection_testG)
diagonal_testG<-diag(confusionmatrix_testG)
summ_testG<-sum(confusionmatrix_testG)
misclassificationrate_testG<-1-(sum(diagonal_testG)/sum(summ_testG))
misclassificationrate_testG
plot(fit_model_gini_test)
text(fit_model_gini_test, pretty=0)
fit_model_gini_test
summary(fit_model_gini_test)
```

2.3tree depth train

```
#fit the model on train
fit_tree_depth_train=tree(good_bad~., data=train)
trainScore_dp=rep(2,15)
testScore_dp=rep(2,15)
for(i in 2:15) {
  #pruneing on test
  prunedTree=prune.tree(fit_tree_depth_train,best=i)
  pred=predict(prunedTree, newdata=test, type="tree")
  trainScore_dp[i]=deviance(prunedTree)
  testScore_dp[i]=deviance(pred)
}
plot(2:15, trainScore_dp[2:15], type="b", col="red", ylim=c(0,700))
points(2:15, testScore_dp[2:15], type="b", col="blue")
#finding the tree depth on validation
finalTree_leaf=prune.tree(fit_tree_depth_train, best=12)
Yfit_leaf=predict(finalTree_leaf, newdata=valid, type="class")
#table of the tree
table(valid$good_bad,Yfit_leaf)
#depth of tree
nodes_traindp_opt<-as.numeric(rownames(finalTree_leaf$frame))
depth_tree_train<<-max(tree:::tree.depth(nodes_traindp_opt))
depth_tree_train
```

2.4naive's bayes

```
#libraries for navies
library(MASS)
library(e1071)
```

naive bayes train

```
#navi bayes train
fitmodel_navitrain=naiveBayes(good_bad~., data=train)
predection_trainN <-predict(fitmodel_navitrain,newdata=train,
                            type="class")
confusionmatrix_trainN<-table( train$good_bad , predection_trainN)
diagonal_trainN<-diag(confusionmatrix_trainN)
summ_trainN<-sum(confusionmatrix_trainN)
misclassificationrate_trainN<-1-(sum(diagonal_trainN)/sum(summ_trainN))
misclassificationrate_trainN
```

naive bayes for test

```
#navi bayes test
fitmodel_navitest=naiveBayes(good_bad~., data=test)
predection_testN <-predict(fitmodel_navitrain,newdata=test,
                           type="class")
confusionmatrix_testN<-table( test$good_bad , predection_testN)
diagonal_testN<-diag(confusionmatrix_testN)
summ_testN<-sum(confusionmatrix_testN)
misclassificationrate_testN<-1-(sum(diagonal_testN)/sum(summ_testN))
misclassificationrate_testN
```

2.6-loss matrix naive bayes train

```
naive_train_loss=naiveBayes(good_bad~., data=train,type="prob")
predict_naivetrain_loss<-predict(naive_train_loss, newdata=train)
mtable_naivetrain_loss<- table(predict_naivetrain_loss,train$good_bad)
Ynaive_nb_train = predict(naive_train_loss, newdata=train, type = "raw")

Ynaive_nb_train <- as.data.frame(Ynaive_nb_train)
output <- rep(NA, nrow(Ynaive_nb_train))
Ynaive_nb_train <- cbind(output,Ynaive_nb_train)


for(i in 1:nrow(Ynaive_nb_train)){
if((Ynaive_nb_train[i,"good"]*1) > (Ynaive_nb_train[i,"bad"]*10))
  {
    Ynaive_nb_train[i,"output"] <- "good"
  }else{
    Ynaive_nb_train[i,"output"] <- "bad"
  }}

Ynaivetest=predict(naive_train_loss, newdata=test, type = "raw")
Ynaivetest1 <- as.data.frame(Ynaivetest)
output <- rep(NA, nrow(Ynaivetest1))
Ynaivetest1 <- cbind(output,Ynaivetest1)


for(i in 1:nrow(Ynaivetest1)){
  if((Ynaivetest1[i,"good"]*1) > (Ynaivetest1[i,"bad"]*10))
  {
    Ynaivetest1[i,"output"] <- "good"
  }else{
    Ynaivetest1[i,"output"] <- "bad"
  }
}


missclasification_tabletrainls <- table(train$good_bad,Ynaive_nb_train$output)
mcisclassification_tabletestls <- table(test$good_bad,Ynaivetest1$output)
misrate_trainls <- 1-sum(diag(missclasification_tabletrainls))/sum(missclasification_tabletrainls)
misrate_testls <- 1-sum(diag(mcisclassification_tabletestls))/sum(mcisclassification_tabletestls)
```

```
cat("Confusion matrix for train data \n")
missclasification_tabletrainls
cat("\n\nConfusion matrix for test data \n")
mcisclassification_tabletestls

cat("\n Misclassification rate for train data = ",misrate_trainls)
cat("\n\n Misclassification rate for test data = ",misrate_testls)
```

2-ass-2-method-AN

2.1

```
set.seed(12345)
credit_data <- 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,]
id3=setdiff(id1,id2)
test=credit_data[id3,]
```

2.2

```
set.seed(12345)
# Create a decision tree model
credit_tree_deviance <- tree(good_bad~., data=train, split = c("deviance"))
credit_tree_gini <- tree(good_bad~., data=train, split = c("gini"))
# Visualize the decision tree with rpart.plot
summary(credit_tree_deviance)
summary(credit_tree_gini)
# predicting on the test dataset to get the misclassification rate.
predict_tree_deviance <- predict(credit_tree_deviance, newdata = test, type = "class")
predict_tree_gini <- predict(credit_tree_gini, newdata = test, type = "class")
conf_tree_deviance <- table(test$good_bad, predict_tree_deviance)
names(dimnames(conf_tree_deviance)) <- c("Actual Test", "P2redicted Test")
caret::confusionMatrix(conf_tree_deviance)
conf_tree_gini <- table(test$good_bad, predict_tree_gini)
names(dimnames(conf_tree_gini)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_tree_gini)
```

Analysis: On the Training dataset model with 'deviance' had a misclassfication rate of 21.05% while the model with 'gini' split had the misclassification rate of 23.68%. For the test dataset we see that the model with 'deviance' type of split has a accuracy of 73.2% or misclassifiaction rate of 26.8%, we see that to predict

'good' the accuracy is 89.08% but for predicting bad its just 36.84%. Thus our our model is heavily baised towards predicting cases as 'good'. For the test dataset we see that the model with 'gini' type of split has a accuracy of 63.6% or misclassifiaction rate of 36.4%, we also see that to predict 'good' the accuracy is 81.03% but for predicting bad its just 18.97%. Thus our model is heavily baised towards predicting cases as 'good' even more than the model which uses 'deviance' to split variable. Both our models would lead to many bad loan applicants to be given loans which is never a good thing, however among the model the one using 'deviance' mode for split is better by 9.6%. Thus we will select model using 'deviance' for further model building.

2.3

```
set.seed(12345)
credit_tree <- tree(good_bad~., data=train, split = c("deviance"))
credit_tree_purned_train <- prune.tree(credit_tree, method = c("deviance"))
credit_tree_purned_valid <- prune.tree(credit_tree, newdata = valid ,method = c("deviance"))
result_train <- cbind(credit_tree_purned_train$size,
credit_tree_purned_train$dev, "Train")
result_valid <- cbind(credit_tree_purned_valid$size,
credit_tree_purned_valid$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 (Shows Deviance least at 4)")
# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=4)
plot(credit_tree_sniped)
text(credit_tree_sniped)
# misclassification rate for best pruned tree
result_prune_test <- predict(credit_tree_sniped, newdata = test, type = "class")
conf_prune_tree_test <- table(test$good_bad, result_prune_test)
names(dimnames(conf_prune_tree_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_prune_tree_test)
```

Analysis: Choosing optimal depth tree we get that '4' as the best depth. The variables used in the best tree are- duration, history, savings. From the tree structure we can see that the following variables are best to split on, 'savings' < 2.5 then 'duration' < 43.5 and 'history' < 1.5. The accuracy on the model trained on 'train' dataset is accuracy 73.2% and misclassifiaction 26.8% while on the 'test' dataset accuracy is 74.4%, thus the misclassification rate is 25.6%. We see that model predicts 'good' applicants very well (accuracy of 96.55%) while it classifies 'bad' applicant way badly (accuracy is 23.68%). Thus this model would be very bad for the business and would likely to run the business bankrupt.

2.4

```
#Fitting the Naive Bayes model
credit_naive_model = 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$good_bad, predict_naive_train)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)
conf_naive_test <- table(test$good_bad, predict_naive_test)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
```

Analysis: For the train dataset using NaiveBayes method we get accuracy 70% or misclassification of 30%, here we alsonotice that the accuracy of class 'bad' is 64.63% while for class 'good' is 72.24%, thus the model is morebalanced in predicting, thus its still baised in predict one class over the other. For the test dataset using NaiveBayes method we get accuracy 68.4% or misclassification of 31.6%, herewe also notice that the accuracy of class 'bad' is 60.53% while for class 'good' is 71.84%, thus the model isalmost the same compared to train.Compared to step3, we see that for the 'train' dataset the optimal tree has accuracy of 73.2% while it is74.4% on the 'test' dataset. For the NaiveBayes model, accuracy on the 'train' dataset is 70%and while it is68.4% on the 'test' datatset.Accuracy is only part of the story what we see is better here is that this model classifies 'bad' customersbetter better for both train and test dataset than decision tree (60+% for both train and test for naive compared 36.84% train, 23.68% test for decision tree).

2.5

```
set.seed(12345)
credit_tree <- tree(good_bad~., data=train, split = c("deviance"))
credit_naive_model = naiveBayes(good_bad ~., data=train)
# prune the tree to the required depth
credit_tree_sniped <- prune.tree(credit_tree, best=7)
# predicting class, getting probability
predict_prune_test_prob <- predict(credit_tree_sniped, newdata = test)
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$good_bad), "naivebayes"))
probability_data_tree <- as.data.frame(cbind(predict_prune_test_prob,
                                             as.character(test$good_bad), "tree"))
probability_data_combined <- rbind(probability_data_tree, probability_data_naive)
colnames(probability_data_combined) <- c("prob_bad", "prob_good",
                                         "actual_test_class", "model")
# final dataset
probability_data_combined$prob_good <- as.numeric(as.character(probability_data_combined$prob_good))
# changing the threshold and printing the probability
tree_list <- NULL
naive_list <- NULL
final <- NULL
for(threshold in seq(from = 0.05, to = 0.95, by = 0.05)){
 probability_data_combined$predicted_class <- ifelse(probability_data_combined$prob_good > threshold, "

  df2 <- probability_data_combined[,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) %>%
  select(model, threshold, TPR, FPR)
ggplot(data = final_summary, aes(x = FPR, y=TPR)) + geom_line(aes(colour = model)) +
  geom_abline(intercept = 0, slope = 1) +
  ggtitle("ROC curve for the Naive Bayes vs. Tree Model")
```

2.6

```
set.seed(12345)
credit_naive_model = naiveBayes(good_bad ~., data=train)
# predicting class, getting probability
predict_naive_train_prob <- predict(credit_naive_model, newdata=train, type = "raw")
predict_naive_test_prob <- predict(credit_naive_model, newdata=test, type = "raw")
train <- cbind(predict_naive_train_prob, train)
test <- cbind(predict_naive_test_prob, test)
# class based on the loss matrix
train$predicted_class <- ifelse(train$good > 10*train$bad, "good", "bad")
test$predicted_class <- ifelse(test$good > 10*test$bad, "good", "bad")
# confusion matrix
conf_naive_train <- table(train$good_bad, train$predicted_class)
names(dimnames(conf_naive_train)) <- c("Actual Train", "Predicted Train")
caret::confusionMatrix(conf_naive_train)
conf_naive_test <- table(test$good_bad, test$predicted_class)
names(dimnames(conf_naive_test)) <- c("Actual Test", "Predicted Test")
caret::confusionMatrix(conf_naive_test)
```

Analysis: We see a major drop in accuracy for both train and test cases for using the new loss matrix, theaccuracy for train is 45.4% and for test cases its 49.2%, this was 70% and 68.4% respectively.The biggest difference is in recognizing the bad customers, where the accuracy is 93.20% and 93.42% for thetrain and test cases respectively. This implies our model identifies 'bad' customers with a very high accuracywhile it suffers to detect 'good' customers (accuracy of 25% and 29%). This is actually good for businessbecause its many times more important to identify the bad customers instead of good customer(minimizerisk).The change in the accuracy is due to loss matrix where we have ensured that a customer is considered 'good'customer only if the probability of them being a 'good'customer is 10 times greater than them being a 'bad'customer.

Assignment 3. Uncertainty estimation

Uncertainty estimation

ASS3-method

3.1

```
rm(list=ls())
set.seed(12345)
state_data <- read.csv2("state.csv")
state_data <- state_data %>% arrange(MET)
ggplot(data = state_data, aes(x=MET, y = EX)) +
geom_point() +
geom_smooth(method = 'loess') +
ggtitle("Plot of MET vs. EX")
```

Analysis: As evident from the graph the best model, linear regression will not be a good fit, even the trend is non linear. Piece wise linear model(spline) might be a good one, thus regression per group/cluster will be a good approach.

3.2

```
set.seed(12345)
state_tree_regression <- tree(data = state_data, EX~MET,
control = tree.control(nobs=NROW(state_data),
minsize = 8))
state_cv_tree <- cv.tree(state_tree_regression, FUN = prune.tree)
temp <- cbind(size = state_cv_tree$size, deviance = state_cv_tree$dev) %>% as.data.frame()
ggplot(data = temp, aes(x = size, y = deviance)) +
geom_point() +theme_bw() +
ggtitle("Deviance vs. number of leaves")
# The best size is either 3 or 4
# puring the tree for leaf size of 3
state_cv_tree_purned <- prune.tree(state_tree_regression, best = 3)
plot(state_cv_tree_purned, main="Pruned Tree for the given dataset")
text(state_cv_tree_purned)
compare_data <- predict(state_cv_tree_purned, newdata = state_data)
compare_data <- cbind(compare_data, state_data$EX, state_data$MET)
compare_data <- as.data.frame(compare_data)
colnames(compare_data) <- c("predicted_value", "EX", "MET")
compare_data$residual <- compare_data$EX - compare_data$predicted_value
# plots
ggplot(data=compare_data, aes(x = MET, y = EX)) +
geom_point(aes(x = MET,y=EX)) +
geom_line(aes(x = MET, y=predicted_value), colour="blue") +
ggtitle("EX value along with Predicted Value")
ggplot(compare_data, aes(x = EX, y = predicted_value)) +
geom_point() +
```

```
ggtitle("Plot of Actual vs. Predicted Value")
ggplot(compare_data, aes(x = predicted_value, y = residual)) +
geom_point() + geom_abline(slope=0, intercept=0) +
ggtitle("Plot of Predicted Value vs. Residual")
# Residuals
ggplot(data = compare_data, aes(residual)) +
geom_histogram(col="black",aes(fill=..count..),bins = 30) +
scale_fill_gradient("Count", low = "grey", high = "black") +
theme_bw() +
ggtitle("Histogram of residuals") +
labs(x="Residual", y="Count")
```

Analysis: The predicted vs. Actual provides us the insight that for lower values of variable, our model is over predicting(actual value ~200) while the predicted value is ~250. While for larger values (~400) our model is underpredicting (~330). At around the mean value (~300) the predicted values are both under and over predictedthus no bias. Thus for values that are away from the mean our model is baised towards over/under predictedwhile values close to mean our model is not biased. From the plot of Predicted vs. Residual values we can see that error appears random, neither is largebias/concentration of the error towards any value expect at lower/higher values(more points on one side ofthe line)From the histogram we can see that the histogram has higher values on the left of zero, and a longer tailon the right. Thus from the above three points we can see that there is scope of improvement in the model especially in the extreme values of the predicted values.

3.3

```
set.seed(12345)
# 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)
}
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")
res <- boot(state_data, bootstrap, R=1000) #make bootstrap
```

Analysis: The confidence bands certainly appear to be bumpy and not smooth, the confidence bands are bumpy becausethe predicted values shows large flucations. From the width of the confidence band we can assume that ourmodel is not a good one. Ideally we want the confidence band to be narrow thus a wider band suggests weneed further tuning to the model

3.4

```
set.seed(12345)
mle=prune.tree(state_tree_regression, best = 3)
#data2 = state_data
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)

# 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")
```

Analysis: From the plot we see that the width of the 'confidence band'(darker grey) is narrower compared to nonparametric.We also notice that most of the data is at in the 'confidence band', thus I believe that ourregression model does make sense.We can defintely see that most of the data is under the 'prediction band' more than 95%, thus we canreasonably expect that less than 5% of the data is outside of the 'prediction band'. Based on our assumptionof data is normally distributed it does makes sense for prediction band to contain 95% of the data.

3.5

```
set.seed(12345)
# Residuals
geom_histogram(col="black",aes(fill=..count.., y = ..density..),bins = 30) +
scale_fill_gradient("Count", low = "grey", high = "black") +
geom_density(colour = "red") +
theme_bw() +
ggtitle("Histogram of residuals")
```

Analysis:Judging from the density plot we can say that the density resembles that of a 'Beta Distribution' thus a known distribution is fit better using parametric bootstrap.

Assignment 4. Principal components

Question 1

```
library(fastICA)
library(lattice)
library(plotly)
```

```
nirspectra <- read.csv2("NIRspectra.csv")
```

```
ncolum <- ncol(nirspectra)

NIRdata=scale(nirspectra[,-c(ncolum)])

nir_pca=prcomp(NIRdata, center = T)
```

```r
lambda=nir_pca$sdev^2
#proportion of variation
sprintf("%2.3f",lambda[c(1,2,3,4)]/sum(lambda)*100)

# plot percentage of variance explained for each principal component
screeplot(nir_pca, main = "Scree plot", xlab = "Principal Components")
```

```r
NIRdata2 <- cbind(NIRdata, nir_pca$x[,1:2])

NIRdata2 <- as.data.frame(NIRdata2)

#scores
 ggplot(data = NIRdata2, aes(x=PC1, y=PC2) )+
  geom_point()+
  ggtitle("Score plot - PC2 vs PC1")
```

pc-trace plots

2. Make trace plots of the loadings of the components selected in step 1. Is there any principle component that is explained by mainly a few original features?

```r
#the rotation component gives the loadings
U=nir_pca$rotation

#nrow(U)
#sorted.loadings <- U[order(U[, 1]), 1]

myXlab  <- "Variables"
myYlab <- "Variable Loadings"

plot(U[,1], main="Loadings Plot for PC1", xlab=myXlab, ylab=myYlab, cex=1.5, col="red")
plot(U[,2], main="Loadings Plot for PC2", xlab=myXlab, ylab=myYlab, cex=1.5, col="red")
```

From the trace plots above, it is clearly visible that the Principal Component 2 has few features explining it. Among 127 features which explains a diesel fuel, only around 20 features are correlated to PC2 as shown in the Loadings plot for PC2. PC1 is correlated to a higher number of features.

Question 3a

```r
set.seed(12345)
ica<-fastICA(NIRdata, n.comp=2)
W_p <- ica$K %*% ica$W

xvar <- "Features"
yvar <- "Correlation"
plot(W_p[,1], main="Traceplot", cex = 1.5, col = "red", xlab = xvar, ylab = yvar)

plot(W_p[,2],main="Traceplot", cex = 1.5, col = "red", xlab = xvar, ylab = yvar)
```

Question 3b

```
NIRdata3 <- cbind(ica$S[,1:2],NIRdata)

NIRdata3 <- as.data.frame(NIRdata3)
#scores
ggplot(data = NIRdata3, aes(x=V1, y=V2))+geom_point()+
     ggtitle("Score plot")
```

my code PCA

```
 nirspectra<-read.csv2("NIRSpectra.csv",header = TRUE)
```

4.1-PCA

```
#plot on viscosity
data_NIR=nirspectra
data_NIR$Viscosity=c()
res=prcomp(data_NIR)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
```

```
#rotation
U_rotation=res$rotation
head(U_rotation)
#plot on rotation
plot(res$x[,1], res$x[,2])
```

4ass-2method-an

4.1

```
rm(list=ls())
set.seed(12345)

NIR_data <- read.csv2("NIRSpectra.csv")
pca_data = select(NIR_data,-c(Viscosity))
pca_result = prcomp(pca_data)
contribution <- summary(pca_result)$importance
knitr::kable(contribution[,1:5],
caption = "Contribution of PCA axis towards varience explaination")
eigenvalues = pca_result$sdev^2
```

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

Analysis: From the plot of PCA component vs. Viscoity, we can see that 2 components are needed (second component of PCA is needed), this is due to the fact the most of the data is vertically spread, removing this dimension would make it impossible to differenitate the types of diesel. The minimum number of components that account for 99% of the total varience is 2, mainly PC1 and PC2. From the plot we can also see that there are evidently some diesel that are outliers (diesel with first_component> 0.5 and second component~4).

4.2

```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")
ggplot(data = components, aes(x = feature_index, y = PC2)) +
geom_point() +
ggtitle("Traceplot of feature index vs. PC2")
knitr::kable(components[1:10,],
caption = "Contribution of Features towards the Principle Components")
```

Analysis: While for PC1, the loadings of the variables do not differ too much (maximum loading ~ 0.11, minimum loading ~ 0.075), for PC the loadings differ extremely. This principal component is explained mainly by the variables with a high index (on the right of the plot). Here, the maximum loading (~ 0.35)

is much higher than for most of the other variables. From these two plots is evident that there are no few features(<5) which form the core essence of the PCA components, thus the PCA components is made up of many (20+) features atleast and they cannot be limited

4.3

```
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)
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")
```

Analysis:Comparing with the trace plots from step 2, we see that ICA trace plots are almost a (flipped using horizontalaxis) flipped image of the PCA trace plots.The W'(multipled matrix) represents the loadings of the ICA components, which are very similar to the PCA loadings.Comparing the Score plots here we find that the plot is again a flipped version of the score plot from PCA(flipped along vertical axis).

Lab2Block2

Prudhvi Peddmallu

22 April 2019

Assignment 1. Using GAM and GLM to examine the mortality rates

The Excel document influenza.xlsx contains weekly data on the mortality and the number of laboratory-confirmed cases of influenza in Sweden. In addition, there is information about population-weighted temperature anomalies (temperature deficits).

1. Use time series plots to visually inspect how the mortality and influenza number vary with time (use Time as X axis). By using this plot, comment how the amounts of influenza cases are related to mortality rates.
2. Use gam() function from mgcv package to fit a GAM model in which Mortality is normally distributed and modelled as a linear function of Year and spline function of Week, and make sure that the model parameters are selected by the generalized cross-validation. Report the underlying probabilistic model.
3. Plot predicted and observed mortality against time for the fitted model and comment on the quality of the fit. Investigate the output of the GAM model and report which terms appear to be significant in the model. Is there a trend in mortality change from one year to another? Plot the spline component and interpret the plot.
4. Examine how the penalty factor of the spline function in the GAM model from step 2 influences the estimated deviance of the model. Make plots of the predicted and observed mortality against time for cases of very high and very low penalty factors. What is the relation of the penalty factor to the degrees of freedom? Do your results confirm this relationship?
5. Use the model obtained in step 2 and plot the residuals and the influenza values against time (in one plot). Is the temporal pattern in the residuals correlated to the outbreaks of influenza? 6 Fit a GAM model in R in which mortality is be modelled as an additive function of the spline functions of year, week, and the number of confirmed cases of influenza. Use the output of this GAM function to conclude whether or not the mortality is influenced by the outbreaks of influenza. Provide the plot of the original and fitted Mortality against Time and comment whether the model seems to be better than the previous GAM models.

```
library(pamr)
library(glmnet)
library(dplyr)
library(kernlab)
library(ggplot2)
library(akima)
library(mgcv)
library(readxl)
library(grid)
library(plotly)
library(dplyr)
```

```
#data-file
library(xlsx)
flu_data = read.xlsx("influenza.xlsx", sheetName = "Raw data")
library(readxl)
influenza<-read_xlsx("influenza.xlsx")
```

Question1.1-Time series plots to visually inspect how the mortality and influenza

```
library(ggplot2)
#plots
p<-ggplot(data = influenz, aes(x = Time, y = Influenza)) + geom_line(aes(color = "#00AFBB"))
p
q<-ggplot(data = influenz, aes(x = Time, y = Mortality)) + geom_line(aes(color = "#00AFBB"))
q
```

There is increase in influenza cases there is increase in mortality till 2000. After that influenza seems to be having lesser impact on mortality.

Question1.2-Fit a GAM model

```
gam_model=gam(Mortality~Year+s(Week,k=length(unique(influenz$Week))),data=influenz,method="GCV.Cp")
gam_model
(or)
res = gam(Mortality ~ Year +s(Week, k=length(unique(influenza$Week))),method="GCV.Cp", data=influenza)
gam.check(res)
(or)
gam_model <- mgcv::gam(data = flu_data, Mortality~Year+s(Week), method = "GCV.Cp")
summary(gam_model)
```

Analysis: Using the default parameter settings within the *gam*-function implies that *Mortality* is normally distributed (*family=gaussian()*). Also, since *method = "GCV.Cp"*, this leads to the usage of GCV (*Generalized Cross Validation score*) related to the smoothing parameter estimation. The underlying probabilistic model can be written as:

$$Mortality = N(\mu, \sigma^2)$$

$$\hat{Mortality} = Intercept + \beta_1 Year + s(Week) + \epsilon$$

where

$$\epsilon = N(0, \sigma^2).$$

Question1.3:-Plot predicted and observed mortality against time for the fitted model and comment on the quality of the fit.

```
predicted <- predict(res, newdata = influenza, type='response')
ggplot(data =influenza) +
geom_point(aes(Time,Mortality), colour = "Blue") +
geom_line(aes(Time,predicted), colour = "Red")
summary(res)
plot(res)#Plot the spline component
```

Analysis:Significant terms in model - According to the p values in the summary, Spline function of weekis the most sognificant term.Trend in mortality change from one year to another - It seems to be following a cyclic trendand seems to have decreased with the years.

Question1.4:-the penalty factor of the spline function in the GAM model

```
res2 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=0),
method="GCV.Cp", data=influenza)
predicted2 <- predict(res2, newdata = influenza, type='response')
p1 <- ggplot(data =influenza) +
geom_point(aes(Time,Mortality),colour = "Blue") +
geom_line(aes(Time,predicted2),colour = "Red") +
ggtitle("sp = 0")
res3 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=0.01),
method="GCV.Cp", data=influenza)
predicted3 <- predict(res3, newdata = influenza, type='response')
p2 <- ggplot(data =influenza) +
geom_point(aes(Time,Mortality),colour = "Blue") +
geom_line(aes(Time,predicted3),colour = "Red") +
ggtitle("sp = 0.01")
res4 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=1),
method="GCV.Cp", data=influenza)
predicted4 <- predict(res4, newdata = influenza, type='response')
p3 <- ggplot(data =influenza) +
geom_point(aes(Time,Mortality),colour = "Blue") +
geom_line(aes(Time,predicted4),colour = "Red") +
ggtitle("sp = 1")
grid.newpage()
grid.draw(cbind(ggplotGrob(p1), ggplotGrob(p2), ggplotGrob(p3), size = "last"))
#anova(res2)
#some more models with different sp values
res5 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=1.5),
method="GCV.Cp", data=influenza)
res6 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=2),method="GCV.Cp", data=influenza)
res7 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=3),
method="GCV.Cp", data=influenza)
res8 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=6),
method="GCV.Cp", data=influenza)
res9 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=9),
method="GCV.Cp", data=influenza)
res10 = gam(Mortality ~ Year +
s(Week, k=length(unique(influenza$Week)), sp=100),
method="GCV.Cp", data=influenza)
2-method
model_deviance <- NULL
for(sp in c(0.001, 0.01, 0.1, 1, 10))
{
k=length(unique(flu_data$Week))

gam_model <- mgcv::gam(data = flu_data, Mortality~Year+s(Week, k=k, sp=sp), method = "GCV.Cp")
temp <- cbind(gam_model$deviance, gam_model$fitted.values, gam_model$y, flu_data$Time_fixed,
```

```
              sp, sum(influence(gam_model)))
model_deviance <- rbind(temp, model_deviance)
}
model_deviance <- as.data.frame(model_deviance)
colnames(model_deviance) <- c("Deviance", "Predicted_Mortality", "Mortality", "Time",
                              "penalty_factor", "degree_of_freedom")
model_deviance$Time <- as.Date(model_deviance$Time, origin = '1970-01-01')
# plot of deviance
p6 <- ggplot(data=model_deviance, aes(x = penalty_factor, y = Deviance)) +
geom_point() +
  geom_line() +
      theme_light() +
ggtitle("Plot of Deviance of Model vs. Penalty Factor")
p6
# plot of degree of freedom
p7 <- ggplot(data=model_deviance, aes(x = penalty_factor, y = degree_of_freedom)) +
geom_point() +
  geom_line() +
      theme_light() +
ggtitle("Plot of degree_of_freedom of Model vs. Penalty Factor")
p7
model_deviance_wide <- melt(model_deviance[,c("Time", "penalty_factor",
                                              "Mortality", "Predicted_Mortality")],
                            id.vars = c("Time", "penalty_factor"))
# plot of predicted vs. observed mortality
p8 <- ggplot(data=model_deviance_wide[model_deviance_wide$penalty_factor == 0.001,],
             aes(x= Time, y = value)) +
  geom_point(aes(color = variable), size=0.7) +
  geom_line(aes(color = variable), size=0.7) +
  scale_color_manual(values=c("#E69F00", "#009E73")) +
  theme_light() +
  ggtitle("Plot of Mortality vs. Time(Penalty 0.001)")
p9 <- ggplot(data=model_deviance_wide[model_deviance_wide$penalty_factor == 10,],
             aes(x= Time, y = value)) +
  geom_point(aes(color = variable), size=0.7) +
    geom_line(aes(color = variable), size=0.7) +
  scale_color_manual(values=c("#E69F00", "#009E73")) +
    theme_light() +
  ggtitle("Plot of Mortality vs. Time(Penalty 10)")
p8
p9
```

Question1.5:-plot the residuals and the influenza values against time (in one plot).

```
edf is estimated degree of freedom if you see res2 u can get edf
edf vs sp
x <- c(0, 0.01, 1, 1.5, 2, 3, 6, 9, 100)
y <- c(sum(res2$edf), sum(res3$edf), sum(res4$edf), sum(res5$edf), sum(res6$edf),
sum(res7$edf), sum(res8$edf), sum(res9$edf), sum(res10$edf))
data.frame(cbind(sp=x,edf=y) )
ggplot(data = influenza, aes(x = Time)) +
```

```
geom_line(aes(y = Influenza,colour = "Influenza")) +
geom_line(aes(y = res$residuals,colour = "Residuals")) +
scale_colour_manual("", breaks = c("Influenza", "Residuals"),
values = c("Red", "Blue"))
result
# sp edf
# 1 0.00 29.000000
# 2 0.01 6.689771
# 3 1.00 3.491752
# 4 1.50 3.365887
# 5 2.00 3.291922
# 6 3.00 3.208210
# 7 6.00 3.112130
# 8 9.00 3.076759
# 9 100.00 3.007268
#1.5
k=length(unique(flu_data$Week))
gam_model <- mgcv::gam(data = flu_data, Mortality~Year+s(Week, k=k), method = "GCV.Cp")
temp <- flu_data
temp <- cbind(temp, residuals = gam_model$residuals)
p10 <- ggplot(data = temp, aes(x = Time_fixed)) +
  geom_line(aes( y = Influenza, color = "Influenza")) +
  geom_line(aes(y = residuals, color = "residuals")) +
      theme_light() +
  scale_color_manual(values=c(Influenza = "#009E73", residuals = "#E69F00")) +
  labs(y = "Influenza / Residual") +
  ggtitle("Plot of Influenza Residual vs. Time")
p10
```

Analysis:With increase in Penalty factor the degrees of freedom should decrease. Our results also confirm this relationship. Initially the decline is steep. Some of the peaks in Influenza outbreaks correspond to peaks in the residuals of the fitted model. Still,however, a lot of variance in the residuals is not correlated to Influenza outbreaks. Therefore, I would say that the Influenza outbreaks are not correlated to the residuals.

Question1.6:-6. Fit a GAM model in R in which mortality is be modelled as an additive function of the spline functions of year, week, and the number of confirmed cases of influenza.

```
new_res=gam(Mortality~s(Year,k=length(unique(influenza$Year)))
+s(Week, k=length(unique(influenza$Year))))this is continued up
summary(new_res)
plot(new_res, residuals=TRUE, page=1)
new_predicted <- predict(new_res, newdata = influenza, type='response')
ggplot(data = influenza, aes(x = Time)) +
geom_point(aes(y = Mortality, colour = "Mortality")) +
geom_line(aes(y = new_predicted,colour = "New Model")) +
geom_line(aes(y = predicted,colour = "Previous Model")) +
scale_colour_manual("", breaks = c("Mortality","New Model", "Previous Model"),
values = c("Blue", "Red", "Green"))
#(or)
gam_model_additive <- mgcv::gam(data = flu_data, Mortality~s(Year)+s(Week), method = "GCV.Cp")
k1 = length(unique(flu_data$Year))
```

```
k2 = length(unique(flu_data$Week))
k3 = length(unique(flu_data$Influenza))
gam_model_additive <- gam(Mortality ~ s(Year, k=k1) +
                                      s(Week, k=k2) +
                                     s(Influenza, k=k3),
                          data = flu_data)
summary(gam_model_additive)
flu_data$fitted.values = gam_model_additive$fitted.values

p11 <- ggplot(data = flu_data, aes(x = Time_fixed)) +
  geom_line(aes( y = Mortality, color = "Mortality")) +
  geom_line(aes(y = fitted.values, color = "fitted.values")) +
      theme_light() +
  scale_color_manual(values=c(Mortality = "#009E73", fitted.values = "#E69F00")) +
  labs(y = "Mortality / fitted.values") +
  ggtitle("Plot of Mortality and Fitted vs. Time")
p11
```

Analysis:-From the p values, we can conclude that Influenza cases have high impact on mortality The additive GAM model clearly has the best fit. Much of the variance of the data (81.9% is the adjusted R^2) is captured by the model. Given that the GAM models in step 2 and step 4 do not include the influenza variable from the dataset, and the the model above does, one can say that most likely mortality is influenced by the outbreaks of influenza.

Assignment 2. High-dimensional methods

The data file data.csv contains information about 64 e-mails which were manually collected from DBWorld mailing list. They were classified as: 'announces of conferences' (1) and 'everything else' (0) (variable Conference) 1. Divide data into training and test sets (70/30) without scaling. Perform nearest shrunken centroid classification of training data in which the threshold is chosen by cross-validation. Provide a centroid plot and interpret it. How many features were selected by the method? List the names of the 10 most contributing features and comment whether it is reasonable that they have strong effect on the discrimination between the conference mails and other mails? Report the test error. 2. Compute the test error and the number of the contributing features for the following methods fitted to the training data: a. Elastic net with the binomial response and α ????????=0.5 in which penalty is selected by the cross-validation b. Support vector machine with "vanilladot" kernel. Compare the results of these models with the results of the nearest shrunken centroids (make a comparative table). Which model would you prefer and why? 3. Implement Benjamini-Hochberg method for the original data, and use t.test() for computing p-values. Which features correspond to the rejected hypotheses? Interpret the result

Question2.1:- Perform nearest shrunken centroid classification

```
library(pamr)
library(glmnet)
library(dplyr)
library(kernlab)
# dividing data into train and test set
data <- read.csv(file = "data.csv", sep = ";", header = TRUE, fileEncoding = "Latin1")
data$Conference=as.factor(data$Conference)
rownames(data)=1:nrow(data)
n=dim(data)[1]
```

```r
set.seed(12345)
id=sample(1:n, floor(n*0.7))#0.7 is 70%
train=data[id,]
test=data[-id,]
#Perform of training data in which the threshold is chosen
# by cross-validation
x = t(train[,-4703])#4703 are the variables in the data
y = train[[4703]]
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))
model = pamr.train(mydata,threshold=seq(0,4, 0.1))
cvmodel=pamr.cv(model,mydata)
pamr.plotcv(cvmodel)
# from cv model,when the threshold is 1.3 and 1.4, the error is least. Hence selecting the threshold as
pamr.plotcen(model, mydata, threshold=1.4)
# The method selected 231 features
contri_genes=pamr.listgenes(model,mydata,threshold=1.4)
# 10 most contributing features
imp = as.numeric(contri_genes[1:10,1])
cat( paste( colnames(data[imp]), collapse='\n' ) )
#test error
pred_test <- pamr.predict(model, newx = x_test, threshold=1.4)
misclass_table <- table(y_test,pred_test)
test_error <- 1 - sum(diag(misclass_table))/sum(misclass_table)
#Elastic net
x = train[,-4703]%>% as.matrix()
y = train[[4703]]
x_test = test[,-4703]%>% as.matrix()
y_test = test[[4703]]
cvfit <- cv.glmnet(x, y, family = "binomial", alpha = 0.5)
plot(cvfit)
predict_elastic <- predict.cv.glmnet(cvfit, newx = x_test, s="lambda.min",type = "class")
coeffs <- coef(cvfit,s="lambda.min")
e_variables = as.data.frame(coeffs[which(coeffs[,1]!=0),])
cmatrix_elastic <- table(y_test, predict_elastic)
testerror_elastic <- 1 -sum(diag(cmatrix_elastic))/sum(cmatrix_elastic)
```

From plot of cross validation model, which shows the depandance of misclassification error vs the threshold value, we can see that the misclassification error is least when the threshold is 1.3 and 1.4. So for this analysis, threshold value of 1.4 is chosen. Nearest centroid method computes a standardized centroid for each class. The nearest shrunken centroid classification "shrinks" the class centroids towards zero by threshold, setting it equal to zero if it hits zero. After the shrinkage, the new sample is classifies using the centroid rule. So this will make the classifier more accurate by reducing the effecte of unimportant features so feature selection happens here. Here the centroid plot shows the shrunken centroids for each of the variables which has non zero differance to the shrunken centroids of two classes. The method selected 231 features. The 10 most contributing features are: paper, important, submissions, due, published, position, call, conferance, dates, candidates All the names and features mentioned above are significant in classification of emails as conferance emails or not. Test Error : 0.1 - 10% test error and 90% accuracy.

Question2.2:-SVM

```
#SVM
svm_model <- ksvm(x, y, type = 'C-svc', kernel = "vanilladot", scale = F)
predict_svm <- predict(svm_model,newdata = x_test, type = "response")
mtable_svm <- table(y_test,predict_svm)
testerror_svm <- 1 -sum(diag(mtable_svm))/sum(mtable_svm)
# table
table_result <- data.frame("Model" = c("Nearest Shrunken Centroid Model",
"ElasticNet Model", "SVM Model"), "Features" = c(231,40,41),
"Error" = c(test_error, testerror_elastic,testerror_svm ))
knitr::kable(table_result, caption = "Model Comparison")
```

On comparing all three models, Nearest Shrunken Centeroid model and Elastic model has same rate of test error which is 0.1 (10%) where centeroid model takes 231 parameters as contributing parameters whereas Elastic net model makes use of less number of features to do the prediction as the contributing features. In case of SVM model, it uses 41 features as contributing features and the error rate is 0.05 or 5% which is less than the above two models and it uses 41 features as contributing features which is not significantly differant compared to the elastic model. So based on the above two factors, SVM model is better for classification of the data because of the accuracy that the model provides with less number of features

Question:-2.3-Benjamini-Hochberg method

```
pvalues <- c()
x <- data[,-4703]
#y <- as.vector(data[,4703])
for(i in 1:(ncol(data)-1)){
t_res <- t.test(x[,i]~Conference, data = data)
14
pvalues[i] <- t_res$p.value
}
p_adj <- p.adjust(pvalues, method = "BH", n = length(pvalues))
p_adj_df <- data.frame("feature" = colnames(data[,-4703]), "pvals" = p_adj)
p_adj_df <- p_adj_df[which(p_adj_df[,2] <= 0.05), ]
num_features <- nrow(p_adj_df)
p_adj_df[c(1:39),1]
```

We consideres null hypothesis as the feature is not significant for analysing if the email is a conferance email and alternate hopothesis: The feature is significant in classifying coferance email. From the above analysis, we can see that the model selected 39 features as significant features by rejecting the null hypothesis(the features with p values less than 0.05). The features corresponding to rejected hypothesis are: [1] apply authors call camera candidate candidates chairs ## [36] submission team topics workshop 4702 Levels: a4soa aaai aachen aalborg aamas aarhus aaron aau abbadi abdalla abdallah abductive abilities The above features are the significant features as selected by the model. From the above feature list, the words team, important, topics, presented, proceedings, helg, org, international, call, papers, phd, published etc seems very relevant in classification of a conferance email

Assignment 2-2method

2.1.

```
rm(list=ls())
gc()
data <- read.csv(file = "data.csv", sep = ";", header = TRUE)
```

```
n=NROW(data)
data$Conference <- as.factor(data$Conference)
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)
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
pamr.plotcv(cvmodel)
pamr.plotcen(model, mydata, threshold = 1.3)
```

```
### important features
## List the significant genes
NROW(important_gen)
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 ")
```

```
### confusion table
library(caret)
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)
```

2.2

```
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")
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)
# svm
svm_fit <- kernlab::ksvm(x, y, kernel="vanilladot", scale = FALSE, type = "C-svc")
predicted_svm_test <- predict(svm_fit, x_test, type="response")
conf_svm_tree <- table(y_test, predicted_svm_test)
names(dimnames(conf_svm_tree)) <- c("Actual Test", "Predicted SVM Test")
result_svm <- caret::confusionMatrix(conf_svm_tree)
#caret::confusionMatrix(conf_svm_tree)
# creating table
final_result <- cbind(result_scc$overall[[1]]*100,
                      result_elastic_net$overall[[1]]*100,
                      result_svm$overall[[1]] *100) %>% as.data.frame()
features_count <- cbind(NROW(important_gen), NROW(elastic_variable), NCOL(data))
final_result <- rbind(final_result, features_count)
colnames(final_result) <- c("Nearest Shrunken Centroid Model",
                            "ElasticNet Model", "SVM Model")
rownames(final_result) <- c("Accuracy", "Number of Features")
#knitr::kable(final_result, caption = "Comparsion of Models on Test dataset")
```

2.3. Implement Benjamini-Hochberg method for the original data, and use t.test() for computing p-values. Which features correspond to the rejected hypotheses? Interpret the result.

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

previous code

Prudhvi Peddmallu

23 April 2019

neuralnet-package

```
library(neuralnet)

set.seed(1234567890)
Var <- runif(50, 0, 3)
tr <- data.frame(Var, Sin=sin(Var))
Var <- runif(50, 3, 9)
te <- data.frame(Var, Sin=sin(Var))
```

```
#setwd("your directory")
mydata <- read.csv("dividendinfo-1.csv")
#attach(mydata)
##Data Normalization-This involves adjusting the data to a common scale so as to accurately compare pre
scaleddata<-scale(mydata)
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
maxmindf <- as.data.frame(lapply(mydata, normalize))
#We base our training data (trainset) on 80% of the observations. The test data (testset) is based on t
# Training and Test Data we have 459 observations in data so 80% is 367 are train and remaining are tes
trainset <- maxmindf[1:160, ]
testset <- maxmindf[161:200, ]
#Neural Network
#Neural Network
library(neuralnet)
nn <- neuralnet(dividend ~ fcfps + earnings_growth + de + mcap + current_ratio, data=trainset, hidden=c
nn$result.matrix
plot(nn)
#Testing The Accuracy Of The Model
#Test the resulting output
temp_test <- subset(testset, select = c("fcfps","earnings_growth", "de", "mcap", "current_ratio"))
head(temp_test)
nn.results <- compute(nn, temp_test)
results <- data.frame(actual = testset$dividend, prediction = nn.results$net.result)
```

MMs

```
# JMP

install.packages("mvtnorm")
library(mvtnorm)

set.seed(1234567890)
```

```
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=300 # number of training points
D=2 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data

# Producing the training data
mu1<-c(0,0)
Sigma1 <- matrix(c(5,3,3,5),D,D)
dat1<-rmvnorm(n = 100, mu1, Sigma1)
mu2<-c(5,7)
Sigma2 <- matrix(c(5,-3,-3,5),D,D)
dat2<-rmvnorm(n = 100, mu2, Sigma2)
mu3<-c(8,3)
Sigma3 <- matrix(c(3,2,2,3),D,D)
dat3<-rmvnorm(n = 100, mu3, Sigma3)
plot(dat1,xlim=c(-10,15),ylim=c(-10,15))
points(dat2,col="red")
points(dat3,col="blue")
x[1:100,]<-dat1
x[101:200,]<-dat2
x[201:300,]<-dat3
plot(x,xlim=c(-10,15),ylim=c(-10,15))

K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional means
Sigma <- array(dim=c(D,D,K)) # conditional covariances
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0,1)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0,5)
  Sigma[,,k]<-c(1,0,0,1)
}
pi
mu
Sigma

for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  llik[it] <- 0
  for(n in 1:N) {
    for(k in 1:K) {
      z[n,k] <- pi[k]*dmvnorm(x[n,],mu[k,],Sigma[,,k])
    }

    #Log likelihood computation.
    llik[it] <- llik[it] + log(sum(z[n,]))
```

```
    z[n,] <- z[n,]/sum(z[n,])
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1) {
    if(abs(llik[it] - llik[it-1]) < min_change) {
      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  for(k in 1:K) {
    pi[k] <- sum(z[,k]) / N
    for(d in 1:D) {
        mu[k, d] <- sum(x[, d] * z[, k]) / sum(z[,k])
    }
    for(d in 1:D) {
      for(d2 in 1:D) {
        Sigma[d,d2,k]<-sum((x[, d]-mu[k,d]) * (x[, d2]-mu[k,d2]) * z[, k]) / sum(z[,k])
      }
    }
  }
}
pi
mu
Sigma
```

NNs

```
# JMP

library(neuralnet)
set.seed(1234567890)

Var <- runif(50, 0, 3)
tr <- data.frame(Var, Sin=sin(Var))
Var <- runif(50, 3, 9)
te <- data.frame(Var, Sin=sin(Var))
winit <- runif(10, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 3, startweights = winit, lifesign = "full")
plot(tr,xlim=c(0,9),ylim=c(-2,2))
points(te,col="blue")
points(te$Var,compute(nn,te$Var)$net.result,col="red")
plot(nn)

# The predictions converge to -2 because the sigmoid functions saturate for large Var values.
```

svm

```
# JMP

library(kernlab)
set.seed(1234567890)

data(spam)

index <- sample(1:4601)
tr <- spam[index[1:2500], ]
va <- spam[index[2501:3500], ]
te <- spam[index[3501:4601], ]

# My first guess was that you may get an error ("No sums found") when C=0 because your data may not be
# separable. However, one gets the error even when the data is linearly separable, e.g.

ksvm(type~.,data=tr[1:2,57:58],kernel="rbfdot",kpar=list(sigma=0.05),C=0)

# It is true that theory requires C>0 but, then, I would have expected an information message telling m
# a positive C, not an error. Bottom line: I am not sure why you get an error but I do not think it is
# C must be positive. Issuing an error would be a weird way to instruct the user to raise the value of
# expected that C=0 instruct the system to try to find a linearly separable SVM.

# Model selection.

er<-NULL
myStep<-0.1
for(myC in seq(0.1,10,myStep)){
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=myC)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  er<-c(er,(t[1,2]+t[2,1])/sum(t))
}
plot(er)

# Final model.

min(er)
which.min(er)
filter<-ksvm(type~.,data=spam[index[1:3500],],kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(er)*myS
mailtype <- predict(filter,te[,-58])
t <- table(mailtype,te[,58])
(t[1,2]+t[2,1])/sum(t)
filter<-ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(er)*myStep)

# That the validation and test error are similar suggests no overfitting.

# Pseudocode.

rbfkernel <- rbfdot(sigma = 0.05)
sv<-alphaindex(filter)[[1]]
co<-coef(filter)[[1]]
```

```
k<-NULL
for(i in 1:1000){
  k2<-NULL
  for(j in 1:length(sv)){
    k2<-c(k2,co[j]*rbfkernel(unlist(tr[sv[j],-58]),unlist(va[i,-58])))
    }
  k<-c(k,sum(k2)-b(filter))
}
k
```

loss matrix 0 2 1

```
       6 0 100
       6 2 0
```

```
library("e1071")
library("dplyr")
options(scipen=999)

data <- iris

set.seed(12345)
model = naiveBayes(Species ~., data=data)

prob <- predict(model, newdata=data, type = c("raw"))
prob <- prob %>%  as.data.frame()
colnames(prob) <- c("prob_setosa", "prob_versicolor", "prob_virginica")

data2 <- cbind(data, prob)

loss_matrix <- matrix(data = c(0,2,1,6,0,100,6,2,0), byrow = TRUE, ncol=3, nrow=3)
loss_matrix

## logic consider the first row, predicted class = 1, prob_class_1 > 2 * prob_of_class_2 & prob_class_1


data2$prediced_class <- ifelse(data2$prob_setosa > 2 * data2$prob_versicolor & data2$prob_setosa > 1 *
                               ifelse(data2$prob_versicolor > 6 * data2$prob_setosa & data2$prob_versic
                                      ifelse(data2$prob_virginica > 2 * data2$prob_versicolor & data2$p

## confirmation of results
table(data2$Species, data2$prediced_class)
```

ordinary least squares regression-predection

```
mydata=read.csv2("Bilexempel.csv")
fit1=lm(Price~Year, data=mydata)
summary(fit1)
fit2=lm(Price~Year+Mileage+Equipment,
data=mydata)
```

```
summary(fit2)
#Prediction
fitted <- predict(fit1, interval =
"confidence")
# plot the data and the fitted line
attach(mydata)
plot(Year, Price)
lines(Year, fitted[, "fit"])
# plot the confidence bands
lines(Year, fitted[, "lwr"], lty = "dotted",
col="blue")
lines(Year, fitted[, "upr"], lty = "dotted",
col="blue")
detach(mydata)
```

Ridge regression-crossvalidation

```
#usepackageglmnetwithalpha=0 (Ridge regression)
data=read.csv("machine.csv", header=F)
covariates=scale(data[,3:8])
response=scale(data[, 9])
model0=glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
plot(model0, xvar="lambda", label=TRUE)
#Choosingthe best modelby cross-validation
model=cv.glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
model$lambda.min
plot(model)
coef(model, s="lambda.min")
#Howgoodis thismodelin prediction?
ind=sample(209, floor(209*0.5))
data1=scale(data[,3:9])
train=data1[ind,]
test=data1[-ind,]
covariates=train[,1:6]
response=train[, 7]
model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian",
lambda=seq(0,1,0.001))
y=test[,7]
ynew=predict(model, newx=as.matrix(test[, 1:6]), type="response")
#Coefficient of determination
sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
sum((ynew-y)^2)
```

Variableselection-stepAIC

```
library(MASS)
fit <- lm(V9~.,data=data.frame(data1))
```

```
step <- stepAIC(fit, direction="both")
step$anova
summary(step)
```

Manipulate for mixture models

```
install.packages("manipulate")
library(manipulate)

data(faithful)
hist(faithful$waiting,freq = FALSE)

NormalPlot <- function(mu1,sigma1,mu2,sigma2,pi1){
  xGrid <- seq(40, 100, by=0.001)
  pdf = dnorm(xGrid, mean=mu1, sd=sigma1) * pi1 + (1-pi1) * dnorm(xGrid, mean=mu2, sd=sigma2)
  hist(faithful$waiting,freq = FALSE)
  lines(xGrid, pdf, type = 'l', lwd = 3, col = "blue")
}

manipulate(
  NormalPlot(mu1,sigma1,mu2,sigma2,pi1),
  mu1 = slider(40, 100, step=.1, initial = 70, label = "mu1"),
  sigma1 = slider(0, 10, step=.1, initial = 1, label = "sigma1"),
  mu2 = slider(40, 100, step=.1, initial = 70, label = "mu2"),
  sigma2 = slider(0, 10, step=.1, initial = 1, label = "sigma2"),
  pi1 = slider(0, 1, step=.01, initial = .5, label = "pi1")
)
```

Holdout-diveding the data

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train=data[id,]
test=data[-id,]
#Howtopartition intotrain/valid/test?
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
train=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
valid=data[id2,]
id3=setdiff(id1,id2)
test=data[id3,]
```

Cross-validatation-Try modelswithdifferent predictorsets

```
data=read.csv("machine.csv", header=F)
library(cvTools)
fit1=lm(V9~V3+V4+V5+V6+V7+V8, data=data)
fit2=lm(V9~V3+V4+V5+V6+V7, data=data)
fit3=lm(V9~V3+V4+V5+V6, data=data)
f1=cvFit(fit1, y=data$V9, data=data,K=10,
foldType="consecutive")
f2=cvFit(fit2, y=data$V9, data=data,K=10,
foldType="consecutive")
f3=cvFit(fit3, y=data$V9, data=data,K=10,
foldType="consecutive")
res=cvSelect(f1,f2,f3)
plot(res)
```

LDA

```
resLDA=lda(Equipment~Mileage+Year, data=mydata)
print(resLDA)
#plot
plot(mydata$Year, mydata$Mileage,
col=as.numeric(Pred$class)+1, pch=21,
bg=as.numeric(Pred$class)+1,
main="Prediction")
#Misclassified items
```

Splines

.Smoothingsplines: smooth.spline() .Naturalcubicsplines: ns() in splines .Thinplatesplines: Tps() in fields

```
res1=smooth.spline(data$Time,data$RSS_anchor2,df=10)
predict(res1,x=data$Time)$y
```

Generalizedadditivemodels

```
library(mgcv)
library(akima)
library(plotly)
river=read.csv2("Rhine.csv")
res=gam(TotN_conc~Year+Month+s(Year,
k=length(unique(river$Year)))+
s(Month, k=length(unique(river$Month))),
data=river)
s=interp(river$Year,river$Month, fitted(res))
print(res)
summary(res)
```

```
res$sp
plot(res)
plot_ly(x=~s$x, y=~s$y, z=~s$z, type="surface")
```

Naive Bayes

.naiveBayesin packagee1071

```
library(MASS)
library(e1071)
n=dim(housing)[1]
ind=rep(1:n, housing[,5])
housing1=housing[ind,-5]
fit=naiveBayes(Sat~., data=housing1)
fit
Yfit=predict(fit, newdata=housing1)
table(Yfit,housing1$Sat)
```

Decision trees

```
#treepackage
#Alternative: rpart
#tree(formula, data, weights, control, split = c("deviance", "gini"), .)
#print(), summary(), plot(), text()
library(tree)
n=dim(biopsy)[1]
fit=tree(class~., data=biopsy)
plot(fit)
text(fit, pretty=0)
fit
summary(fit)
#.Misclassificationresults-tree
Yfit=predict(fit, newdata=biopsy, type="class")
table(biopsy$class,Yfit)
#.Selectingoptimal treeby penalizing
#Cv.tree()
set.seed(12345)
ind=sample(1:n, floor(0.5*n))
train=biopsy[ind,]
valid=biopsy[-ind,]
fit=tree(class~., data=train)
set.seed(12345)
cv.res=cv.tree(fit)
plot(cv.res$size, cv.res$dev, type="b",
col="red")
plot(log(cv.res$k), cv.res$dev,
type="b", col="red")
#.Selectingoptimal treeby train/validation
fit=tree(class~., data=train)
trainScore=rep(0,9)
```

```
testScore=rep(0,9)
for(i in 2:9) {
prunedTree=prune.tree(fit,best=i)
pred=predict(prunedTree, newdata=valid,
type="tree")
trainScore[i]=deviance(prunedTree)
testScore[i]=deviance(pred)
}
plot(2:9, trainScore[2:9], type="b", col="red",
ylim=c(0,250))
points(2:9, testScore[2:9], type="b", col="blue")
#.Final tree: 5 leaves
finalTree=prune.tree(fit, best=5)
Yfit=predict(finalTree, newdata=valid,
type="class")
table(valid$class,Yfit)
```

Nearest ShrunkenCentroids-NSC

```
#Packagepamr
#pamr.train()
#pamr.cv
data0=read.csv2("voice.csv")
data=data0
data=as.data.frame(scale(data))
data$Quality=as.factor(data0$Quality)
library(pamr)
rownames(data)=1:nrow(data)
x=t(data[,-311])
y=data[[311]]
mydata=list(x=x,y=as.factor(y),geneid=as.character(1:nrow(x)), genenames=rownames(x))
model=pamr.train(mydata,threshold=seq(0,4, 0.1))
pamr.plotcen(model, mydata, threshold=1)
pamr.plotcen(model, mydata, threshold=2.5)
a=pamr.listgenes(model,mydata,threshold=2.5)
cat( paste( colnames(data)[as.numeric(a[,1])], collapse='\n' ) )
cvmodel=pamr.cv(model,mydata)
print(cvmodel)
pamr.plotcv(cvmodel)
```

Kernel PCA

```
library(kernlab)
K <- as.kernelMatrix(crossprod(t(x)))
res=kpca(K)
barplot(res@eig)
plot(res@rotated[,1], res@rotated[,2], xlab="PC1",
ylab="PC2")
```

Feature assessment

```
res=t.test(MFCC_2nd.coef~Quality,data=data,
alternative="two.sided")
res$p.value
res=oneway_test(MFCC_2nd.coef~as.factor(Qu
ality), data=data,paired=FALSE)
pvalue(res)
```

Nonparametricbootstrap:

```
#Writea functionstatisticthatdependson dataframeand index and returnsthe estimator
library(boot)
data2=data[order(data$Area),]#reordering data according to Area
# computing bootstrap samples
f=function(data, ind){
data1=data[ind,]# extract bootstrap sample
res=lm(Price~Area, data=data1) #fit linear model
#predict values for all Area values from the original data
priceP=predict(res,newdata=data2)
return(priceP)
}
res=boot(data2, f, R=1000) #make bootstrap
```

Parametricbootstrap:

.Computevaluemlethatestimatesmodelparameters from the data .Writefunctionran.genthatdependson dataand mleand whichgenerates new data .Writefunctionstatisticthatdependon datawhichwillbe generatedby ran.genand shouldreturnthe estimator

```
mle=lm(Price~Area, data=data2)
rng=function(data, mle) {
data1=data.frame(Price=data$Price, Area=data$Area)
n=length(data$Price)
#generate new Price
data1$Price=rnorm(n,predict(mle, newdata=data1),sd(mle$residuals))
return(data1)
}
f1=function(data1){
res=lm(Price~Area, data=data1) #fit linear model
#predict values for all Area values from the original data
priceP=predict(res,newdata=data2)
return(priceP)
}
res=boot(data2, statistic=f1, R=1000, mle=mle,ran.gen=rng, sim="parametric")
```

Uncertaintyestimation

```
#Bootstrapcofidencebands for linearmodel
e=envelope(res) #compute confidence bands
fit=lm(Price~Area, data=data2)
priceP=predict(fit)
plot(Area, Price, pch=21, bg="orange")
points(data2$Area,priceP,type="l") #plot fitted line
#plot cofidence bands
points(data2$Area,e$point[2,], type="l", col="blue")
points(data2$Area,e$point[1,], type="l", col="blue")
```

Estimationofthe modelquality-parametricbootstrap

```
mle=lm(Price~Area, data=data2)
f1=function(data1){
res=lm(Price~Area, data=data1) #fitlinear model
#predict values for all Area valuesfrom the original data
priceP=predict(res,newdata=data2)
n=length(data2$Price)
predictedP=rnorm(n,priceP,
sd(mle$residuals))
return(predictedP)
}
res=boot(data2, statistic=f1, R=10000,
mle=mle,ran.gen=rng, sim="parametric")
```

PCA-principle

```
mydata=read.csv2("tecator.csv")
data1=mydata
data1$Fat=c()
res=prcomp(data1)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
#Principal componentloadings(U)
U=res$rotation
head(U)
#Data in (PC1, PC2) -scores (Z)
plot(res$x[,1], res$x[,2], ylim=c(-5,15))
#Traceplots
U=loadings(res)
plot(U[,1], main="Traceplot, PC1")
plot(U[,2],main="Traceplot, PC2")
```

Independent component analysis-ICA

```
S <- cbind(sin((1:1000)/20), rep((((1:200)-100)/100), 5)) #Generate data
A <- matrix(c(0.291, 0.6557, -0.5439, 0.5572), 2, 2)
X <- S %*% A
a <- fastICA(X, 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001, verbose = TRUE) #ICA
```

Untitled

Prudhvi Peddmallu

24 April 2019

pca-pcr

The data file video.csv contains characteristics of a sample of Youtube videos. Import data to R and divide it randomly (50/50) into training and test sets. 1. Perform principal component analysis using the numeric variables in the training data except of "utime" variable. Do this analysis with and without scaling of the features. How many components are necessary to explain more than 95% variation of the data in both cases? Explain why so few components are needed when scaling is not done. (2p) 2. Write a code that fits a principle component regression ("utime" as response and all scaled numerical variables as features) with ???????? components to the training data and estimates the training and test errors, do this for all feasible ???????? values. Plot dependence of the training and test errors on ???????? and explain this plot in terms of bias-variance tradeoff. (Hint: prediction function for principal component regression has some peculiarities, see predict.mvr) (2p) 3. Use PCR model with ????????=8 and report a fitted probabilistic model that shows the connection between the target and the principal components. (1p)

4. Use original data to create variable "class" that shows "mpeg" if variable "codec" is equal to "mpeg4", and "other" for all other values of "codec". Create a plot of "duration" versus "frames" where cases are colored by "class". Do you think that the classes are easily separable by a linear decision boundary? (1p)
5. Fit a Linear Discriminant Analysis model with "class" as target and "frames" and "duration" as features to the entire dataset (scale features first). Produce the plot showing the classified data and report the training error. Explain why LDA was unable to achieve perfect (or nearly perfect) classification in this case. (2p)
6. Fit a decision tree model with "class" as target and "frames" and "duration" as features to the entire dataset, choose an appropriate tree size by cross-validation. Report the training error. How many leaves are there in the final tree? Explain why such a complicated tree is needed to describe such a simple decision boundary. (2p)

```
data0=read.csv("video.csv")

data1=data0
data1$codec=c()

n=dim(data1)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data1[id,]
test=data1[-id,]


data11=data1
data11$utime=c()
res=prcomp(data11)
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)

res=prcomp(scale(data11))
lambda=res$sdev^2
```

```
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)


data11=t(apply(as.matrix(data1[1:100,1:18]), 1, combn, 3, prod))
library(pamr)
mydata=as.data.frame(scale(data11))
rownames(mydata)=1:nrow(mydata)
x=t(mydata)
y=data0$codec[1:100]
mydata1=list(x=x,y=as.factor(y),geneid=as.character(1:nrow(x)), genenames=rownames(x))
model=pamr.train(mydata1,threshold=seq(0,4, 0.1))
set.seed(12345)
cvmodel=pamr.cv(model,mydata1)
pamr.plotcv(cvmodel)

cvmodel$threshold[which.max(cvmodel$loglik)]

data2=data0
data2$class=ifelse(data2$codec=="mpeg4", "mpeg4", "other")
data2$codec=c()
data2$frames=scale(data2$frames)
data2$duration=scale(data2$duration)

plot(data2$frames,data2$duration, col=as.factor(data2$class), cex=0.5)
m3=lda(as.factor(class)~frames+duration, data=data2)

plot(data2$frames,data2$duration, col=predict(m3)$class)

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

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

library(tree)
m4=tree(as.factor(class)~frames+duration, data=data2)
set.seed(12345)
cv.res=cv.tree(m4)
plot(cv.res$size, cv.res$dev, type="b",
     col="red")

print(m4)
plot(m4)
missclass(data2$class, predict(m4, type="class"))
```

neuralnet-twolayers

NEURAL NETWORKS (3p) Train a neural network (NN) to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting pairs are the data available to you. Use 25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent. Consider threshold values i/1000 with i = 1,

. . . , 10. Initialize the weights of the neural network to random values in the interval [???1, 1]. Consider two NN architectures: A single hidden layer of 10 units, and two hidden layers with 3 units each. Choose the most appropriate NN architecture and threshold value. Motivate your choice. Feel free to reuse the code of the corresponding lab. (1p) Estimate the generalization error of the NN selected above (use any method of your choice). (1p) In the light of the results above, would you say that the more layers the better ? Motivate your answer.

```r
# JMP

library(neuralnet)

# two layers

set.seed(1234567890)

Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# plot(trva)
# plot(tr)
# plot(va)

restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(22, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit,
                  threshold = i/1000, lifesign = "full")

  # nn$result.matrix

  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared err
  restr[i] <- sum((tr[,2] - aux)**2)/2

  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

# one layer

set.seed(1234567890)

Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# plot(trva)
```

```r
# plot(tr)
# plot(va)

restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(41, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(10), startweights = winit,
                  threshold = i/1000, lifesign = "full")

  # nn$result.matrix

  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared err
  restr[i] <- sum((tr[,2] - aux)**2)/2

  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

# estimate generalization error for the best run above (one layer with threshold 4/1000)

Var <- runif(50, 0, 10)
te <- data.frame(Var, Sin=sin(Var))

winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit,
                    threshold = 4/1000, lifesign = "full")
sum((te[,2] - compute(nn, te[,1])$net.result)**2)/2
```

SUPPORT VECTOR MACHINES-gaussian

(3p) Implement the budget online support vector machine (SVM). Check the course slides for the pseudo-code. Feel free to use the template below. Note that you are not using all the attributes and points in the file. Run your code on the spambase.csv file for the (M, ??) values (500,0) and (500,-0.05). Plot the error rate as a function of the number of training point. (2p) Analyze the results obtained. In particular, ??? explain why (500,0) gives better results than (500,-0.05), and ??? explain why the setting (50,0) is the slowest (you do not need to run your code until completion for this setting). #GIVEN CODE IN QUESTION

```r
set.seed(1234567890)
spam <- read.csv2("spambase.csv")
ind <- sample(1:nrow(spam))
spam <- spam[ind,c(1:48,58)]
h <- 1
beta <- # Your value here
M <- # Your value here
N <- 500 # number of training points
gaussian_k <- function(x, h) { # Gaussian kernel
# Your code here
```

```
}
SVM <- function(sv,i) { # SVM on point i with support vectors sv

# Your code here
# Note that the labels in spambase.csv are 0/1 and SVMs need -1/+1
# Then, use 2*label-1 to convert from 0/1 to -1/+1
# Do not include the labels when computing the Euclidean distance between
# the point i and each of the support vectors. This is the distance to use
# in the kernel function. You can use dist() to compute the Euclidean distance
}
errors <- 1
errorrate <- vector(length = N)
errorrate[1] <- 1
sv <- c(1)
for(i in 2:N) {
# Your code here
}
plot(errorrate[seq(from=1, to=N, by=10)], ylim=c(0.2,0.4), type="o")
length(sv)
errorrate[N]
```

CODE

```
set.seed(1234567890)
spam <- read.csv2("../Lab2aBlock2/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]
```

SVM

You are asked to use the function ksvm from the R package kernlab to learn a support vector machine (SVM) for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5. This implies that you have to consider three models. (2p) Perform model selection, i.e. select the most promising of the three models (use any method of your choice except cross-validation or nested cross-validation). (1p) Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or

nested cross-validation). (1p) Produce the SVM that will be returned to the user, i.e. show the code. (1p) What is the purpose of the parameter C ?

```
library(kernlab)
set.seed(1234567890)

data(spam)

# Model selection

index <- sample(1:4601)
tr <- spam[index[1:2500], ]
va <- spam[index[2501:3501], ]
te <- spam[index[3502:4601], ]

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=0.5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)

# Error estimation

filter <- ksvm(type~.,data=spam[index[1:3501], ],kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,te[,-58])
t <- table(mailtype,te[,58])
(t[1,2]+t[2,1])/sum(t)

# Final model

filter <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
```

GLM

```
library(glmnet)
library(dplyr)
library(ggplot2)
crabs<-read.csv("australian-crabs.csv")
crabss<-crabs[,c(1,7,8)]
model<-glm(data=crabss, formula=species ~ CW + BD, family=binomial(link='logit'))

ggplot
```

```
slope <- coef(crabs_logit)[2]/(-coef(crabs_logit)[3])
intercept <- coef(crabs_logit)[1]/(-coef(crabs_logit)[3])

# Fit the decision boundary
plot_x <- c(min(crabss$CW)-2, max(crabss$CW)+2)
plot_y <- (-1 /coef(model)[3]) * (coef(model)[1] * plot_x + coef(model)[2])
db.data1 <- data.frame(rbind(model, crabss))
colnames(db.data) <- c('x','y')

ggplot()+geom_line(data=db.data, aes(x=x, y=y))


set.seed(1234)
crabs<- rnorm(6,7)
x2 <- rnorm(20)

y <- sign(-1 - 2 * x1 + 4 * x2 )

y[ y == -1] <- 0

df <- cbind.data.frame( y, x1, x2)

mdl <- glm( y ~ . , data = df , family=binomial)

slope <- coef(mdl)[2]/(-coef(mdl)[3])
intercept <- coef(mdl)[1]/(-coef(mdl)[3])

library(lattice)
xyplot( x2 ~ x1 , data = df, groups = y,
   panel=function(...){
       panel.xyplot(...)
       panel.abline(intercept , slope)
       panel.grid(...)
       })
model <- glm(Species~.,family=binomial(link='logit'),data=crabss)
```

2NAIVE BAYES

```
#navies bayes
library(MASS)
library(e1071)
n=dim(crabs)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=crabs[id,]
test=crabs[-id,]
fitmodel_navitrain=naiveBayes(species~ CW+BD, data=train)
predection_trainN <-predict(fitmodel_navitrain,newdata=test,
                            type="class")
confusionmatrix_trainN<-table( test$species , predection_trainN)
confusionmatrix_trainN
```

```
diagonal_trainN<-diag(confusionmatrix_trainN)
summ_trainN<-sum(confusionmatrix_trainN)
misclassificationrate_trainN<-1-(sum(diagonal_trainN)/sum(summ_trainN))
misclassificationrate_trainN
```

PC

```
#PC
library(dplyr)
crabsP<-read.csv("australian-crabs.csv")
b1<-crabsP[,c(7,8)]
b2<-scale(b1)
#b3<-as.vector(b2)
#pca
#plot on viscosity
data_NIR=b2
res=prcomp(data_NIR)
lambda=res$sdev^2
#eigenvalues
lambda
#proportion of variation
sprintf("%2.3f",lambda/sum(lambda)*100)
screeplot(res)
```

BOOTSTRAP

```
dataB = read.csv('bank.csv', sep=';', dec=',')
data_bank<-read.csv2('bank.csv')
model_glm = glm('Visitors ~ .', data=data, family=poisson(link='log'))

library(ggplot2)
library(boot)
# Creating a copy of the original data.
original_data = data
# Data to be predicted.
data_seq = list(Time=seq(12, 13, 0.05))
# Creating the sampling function.
rng = function(data, model)
{
# Getting the parameters for the poisson distribution.
# In this case y_hat is the mean of the poisson distribution
# aka lambda.
lambda_hat = predict(model, newdata=data, type='response')
nobs = nrow(data)
# Sampling from a poisson distribution.
sample = rpois(nobs, lambda_hat)
data$Visitors = sample
return(data)
}
```

```r
# Function for the prediction interval.
pi_glm = function(data)
{
# Fitting a new model given the sample generated.
new_model = glm('Visitors ~ .',
data=data,
family=poisson(link='log'))
# Estimating E[Y|X] given the bootsrap model

# on the original data.
lambda_hat = predict(new_model, newdata=data_seq, type='response')
# Generating a sample from the bootstraped model.
nobs = length(data_seq$Time)
sample = rpois(nobs, lambda_hat)
return(sample)
}
# Running the bootstrap for the prediction interval.
results = boot(data,
statistic=pi_glm,
R=1000,
mle=model_glm,
ran.gen=rng,
sim='parametric')
p_results = envelope(results)
lambda_hat = predict(model_glm, newdata=data_seq, type='response')
p = ggplot() +
geom_line(aes(x=data_seq$Time, y=p_results$point[1, ], color="Prediction Interval")) +
geom_line(aes(x=data_seq$Time, y=p_results$point[2, ], color="Prediction Interval")) +
geom_point(aes(x=data_seq$Time, y=lambda_hat)) +
scale_colour_manual(values=c("#604dc5", "#020c0b"))
print(p)
```

```r
# plot dependence of CW vs BD where the points are colored by species.
library(dplyr)
library(ggplot2)
crabs %>%
ggplot(aes(x = BD, y = CW))+
geom_point(aes(color = species))+
theme_classic()
# split data into train and test
set.seed(12345)
n <- nrow(crabs)
id <- sample(1:n, floor(0.5*n))
crabs_train <- crabs[id,]
crabs_test <- crabs[-id,]
# Naive bayes classifier
library(e1071)
crabs_model <- naiveBayes(
species ~ BD + CW,
data = crabs_train
)
# prediction
crabs_pred <- predict(crabs_model, newdata = crabs_test,
type = "class"
```

```
)
# confusion matrix
table(crabs_test$species, crabs_pred)
# misclassification rate
mean(crabs_test$species != crabs_pred)
# test for independence between features.
# note it is not required only to prove a point.
cor.test(crabs$CW, crabs$BD)
```

```
# fit a logistic model
crabs_logit <- glm(species ~ BD + CW,
family = binomial(link = "logit"),
data = crabs_train
)
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
# prediction
crabs_pred2 <- predict(crabs_logit, newdata = crabs_test,
type = "response"
)
# threshold 0.5
crabs_pred2 <- ifelse(crabs_pred2 <0.5, 0,1)
table(crabs_test$species, crabs_pred2)
crabs_class <- ifelse(crabs_pred2 == 0, "Blue", "Orange")
# misclassification
mean(crabs_test$species != crabs_class)
# slope and intercept to get the decision boundary
# note this coefficients come from the trained model.
# [1] is the intercept coefficient in the crabs_logit
# [2] is coefficient for BD
# [3] is coefficient for CW
slope <- coef(crabs_logit)[2]/(-coef(crabs_logit)[3])
intercept <- coef(crabs_logit)[1]/(-coef(crabs_logit)[3])
#Plot the classification with the decision boundary
# plot
crabs_test %>%
ggplot(aes(x = BD, y = CW))+
geom_point(aes(color = species))+
geom_abline(slope = slope, intercept = intercept)+
theme_classic()
```

```
library(geosphere)
# A general gaussian kernel that takes one value
k.gaussian <- function(u) {
return(exp(-abs(u)^2))
}
# A kernel that preprocess two longitude, latitude argumnets
# for two position and find the distance between them
# with the help of distHaversine from the package geosphere.
# The calculated value is then passed throught to the general
# gaussian kernel
k.dist.station.poi <- function(s.longitude, s.latitude,
poi.longitude, poi.latitude) {
return(k.gaussian(distHaversine(c(s.longitude, s.latitude),
```

```
c(poi.longitude, poi.latitude))/h_distance))
}
# A kernel that calculates how many days is between two dates.
# But ignores the year difference between them, this is so that we get
# a periodic kernel. But it has to use 365.25 because of each 4 years we
# have a leap year.
# When the distance is calculated it is passed throught to the
# general gaussian kernel.
k.dist.day.poi <- function(d, poi) {
daydiff <- min(abs(as.numeric(difftime(d, poi))%%365.25),
abs(as.numeric(difftime(poi,d))%%365.25))
return(k.gaussian(daydiff/h_date))
}
# A kernel that fins the time difference between two times.
# We have to check both t2-t1 and t2-t1-24 beacuse of the
# reason if we have the times t2 = 22 and t1 = 1 we get
# 21 when we want to have the time difference of 3, which we get
# from the second case.
# This is so that the maximun time difference we can have
# is 12 hours and not 24
k.dist.hour.poi <- function(h, poi) {
t <- as.difftime(c(toString(h),
toString(poi)), units = "hours")
t <- min(abs(as.numeric(t[2]-t[1])), abs(as.numeric(t[2]-t[1]-24)))
return(k.gaussian(t/h_time))
}
set.seed(1234567890)
# Read in the data
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
# Is somewhere near Vadstena
a <- 58.4274 # The point to predict
b <- 14.826
#date <- "1970-04-04" # The date to predict (up to the students)
h_distance <- 100000 # How long distance away to consider. The unit is meters
h_date <- 15 # How many days away to consider. The unit is days
h_time <- 3 # How many hours away to consider. The unit is hours
#a <- 67.827
#b <- 20.3387
date <- "2000-07-04"
# Set up a the times that we shall predict
times <- paste(seq(4, 24, 2), ":00:00", sep = '')
# Create the vector where we store the predicted times
temp.sum <- vector(length=length(times))
temp.prod <- temp.sum
# Remove all dates after the predicted date, including the prediction date
st <- st[which(difftime(st[,"date"], date, units = "days") <= 0),]

library(geosphere)
set.seed(1234567890)
# Create the data frame that we shall pass to our kernels
poi <- data.frame(b, a, date, times)
```

```r
colnames(poi) <- c("longitude", "latitude", "date", "time")
vec.kern.dist <- 1:nrow(st)
vec.kern.day <- vec.kern.dist
vec.kern.hour <- vec.kern.dist
# Precalculate the distance and day kernel
# We do this to avoid calculating this for every time
# that we want to predict. This is beacuse the distance
# and day difference does not change when we change
# the time.
vec.kern.dist <- mapply(k.dist.station.poi, st$longitude, st$latitude,
poi[1,]$longitude, poi[1,]$latitude)
vec.kern.day <- mapply(k.dist.day.poi, st$date, poi[1,]$date)
# Calculate the time difference kernel for every data point and
# time. Then for each time apply sum or multiple the kernels
# with each other to then multiple with the correct temperature
# for the day and divide by the sum&multiplication.
# Now we have predicted the temperature for the given
# place, day and time.
for (i in 1:nrow(poi)) {
vec.kern.hour <- mapply(k.dist.hour.poi, st$time, poi[i,]$time)
vec.kern.sum <- vec.kern.dist+vec.kern.day+vec.kern.hour
temp.sum[i] <- sum(vec.kern.sum*st$air_temperature)/sum(vec.kern.sum)
vec.kern.prod <- vec.kern.dist*vec.kern.day*vec.kern.hour
temp.prod[i] <- sum(vec.kern.prod*st$air_temperature)/sum(vec.kern.prod)
}
# Plot the difference predicted temperature
plot(temp.sum, type="o")
plot(temp.prod, type="o")
x <- seq(0, 500000, 10000)
y <- k.gaussian(x/h_distance)
plot(x=x, y=y)
# ==============
# Assignment 2
# ==============
library(neuralnet)
set.seed(1234567890)
var <- runif(50, 0, 10)
trva <- data.frame(var, sin=sin(var))
tr <- trva[1:25,]
va <- trva[26:50,]
mse <- rep(0, 10)
# Random initial weights
winit <- runif(31, -1, 1)
for(i in 1:10){
# Train neural net
nn <- neuralnet(sin~var, data=tr, hidden=c(10), threshold = i/1000, startweights = winit)
# Predict for test data
pred <- compute(nn, va$var)$net.result
# Compute error (mean squared)
mse[i] <- mean((va$sin - pred)^2)
}
# Threshold which produced minimal error
threshold <- which.min(mse)/1000
```

```r
# Plot net corresponding to minimal error
plot(nn <- neuralnet(sin~var, data=tr, hidden=c(10), threshold=threshold, startweights=winit),
rep="best", main="The Obtained Network")
# Plot prediction and true values
invisible(capture.output(res <- prediction(nn)$rep1))
library(ggplot2)
vis_res <- data.frame(x=res[,1], y=res[,2], Set="Prediction")
vis_orig <- data.frame(x=trva$var, y=trva$sin, Set="Original")
visuals <- rbind(vis_res, vis_orig)
p <- ggplot(visuals, aes(x=x, y=y, color=Set)) + geom_point(shape = 21, size=3, stroke=1) +
labs(x="x", y="Sin(x)") +
ggtitle("Orignal and predicted data")
print(p)
```

Lab 3 Block 1

Maria , Jasleen, Prudhvi

December 18, 2018

```
set.seed(1234567890)
library(geosphere)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

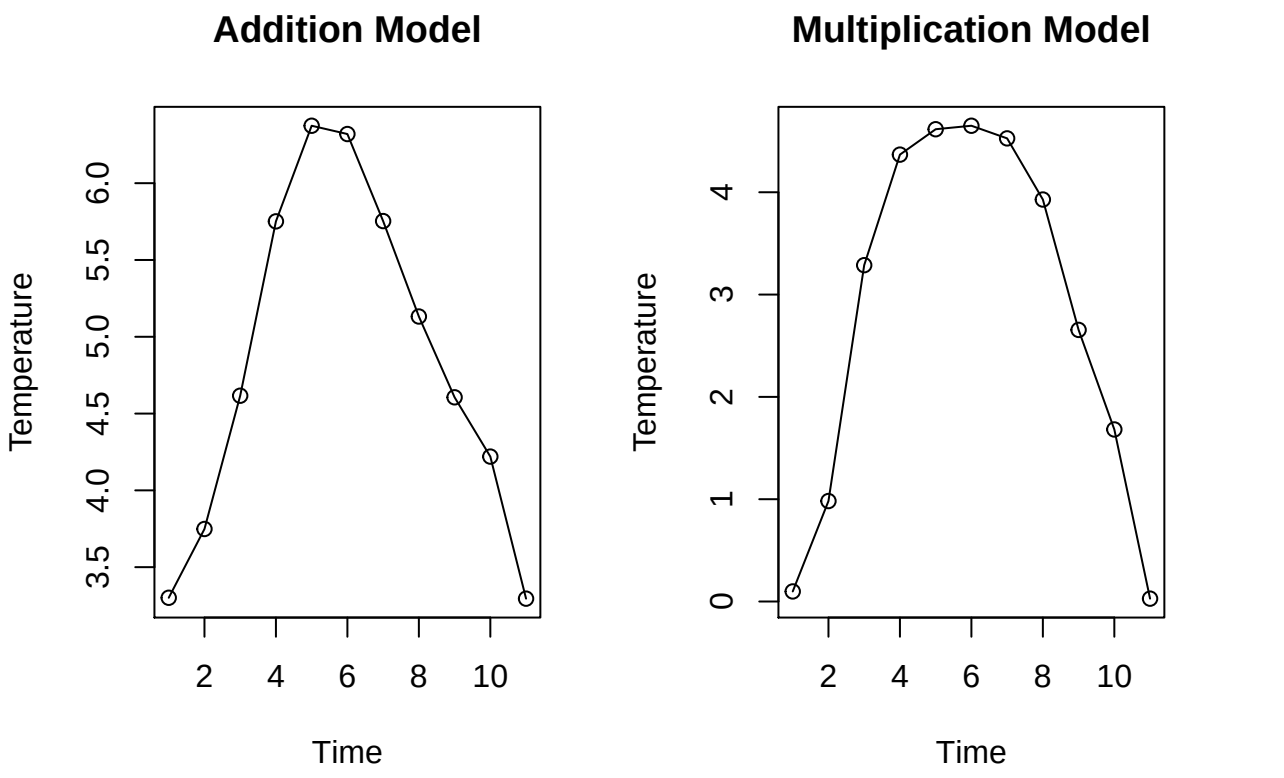
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
stations <- read.csv("stations.csv")
stations <- stations[,-c(2,3,6,7,8)]
temps <- read.csv("temps50k.csv")
temps <- temps[,-c(5)]

pred_temp <- function(date){
st <- merge(stations,temps,by="station_number")
h_distance <- 50000
h_date <- 4
h_time <- 3
a <- 58.4274
b <- 14.826
date <- date
#date <- "2013-01-04" # The date to predict (up to the students)
times <- c("04:00:00", "06:00:00", "08:00:00","10:00:00","12:00:00", "14:00:00",
           "16:00:00","18:00:00","20:00:00","22:00:00","00:00:00")

st$datetime <- as.POSIXct(paste(st$date,st$time), format = "%Y-%m-%d %H:%M:%S")
st_new <- st
temp_add <- vector(length=length(times))
temp_mult <- vector(length=length(times))

cureentdatetime <- as.POSIXct(paste(date,"04:00:00"), format = "%Y-%m-%d %H:%M:%S")

st_new <- st_new%>%filter(datetime < cureentdatetime)

for(i in 1:length(times)){
  st_new$dist_diff <- as.numeric(distHaversine(p1=st_new[,c(3,2)], p2 = c(b,a), r=6378137))
  st_new$date_diff <- abs(as.numeric(difftime(date, st_new$date)))
  st_new$time_diff <-  abs(as.numeric(difftime(strptime(paste(date,times[i]),"%Y-%m-%d %H:%M:%S"),
                                               strptime(paste(date,st_new$time),
                                                        "%Y-%m-%d %H:%M:%S")))/3600)
```

```
  k_dist <- exp(-(st_new$dist_diff/h_distance)^2)
  k_date <- exp(-(st_new$date_diff/h_date)^2)

  k_time <- exp(-(st_new$time_diff/h_time)^2)

  k_sum  <- k_dist+k_date+k_time

  k_temp <- k_sum*st_new$air_temperature
  temp_add[i] <- sum(k_temp)/sum(k_sum)

  k_prd <-  k_dist * k_date * k_time
  k_temp <- k_prd *st_new$air_temperature

  temp_mult[i] <- sum(k_temp)/sum(k_prd)
}


par(mfrow = c(1,2))
plot(temp_add, type="o",ylab = "Temperature",xlab = "Time", main = "Addition Model")
plot(temp_mult, type="o",ylab = "Temperature",xlab = "Time", main = "Multiplication Model" )

}
```

```
pred_temp("2013-01-04")
```



In an area of 50km the temperature might not change much. Also in the range of 3-4 days the differance

in temperature will be very less. A range of 3 hours was taken in a day as that seems reasonable for a temperature change in one day.

Assignment 2

```
library(kernlab)
data(spam)
set.seed(12345)

index <- sample(4601,floor(4601*0.7))

spamtrain <- spam[index,]
spamtest <- spam[-index,]

model1 <- ksvm(type~.,data=spamtrain,kernel="rbfdot",
               kpar=list(sigma=0.05),C=5)

model2 <- ksvm(type~.,data=spamtrain,kernel="rbfdot",
               kpar=list(sigma=0.05),C=1)

model3 <- ksvm(type~.,data=spamtrain,kernel="rbfdot",
               kpar=list(sigma=0.05),C=0.5)

## predict mail type on the test set
mailtype_model1 <- predict(model1,spamtest[,-58])
mailtype_model2 <- predict(model2,spamtest[,-58])
mailtype_model3 <- predict(model3,spamtest[,-58])

## Check results
cm_model1 <- table(mailtype_model1,spamtest[,58])
cm_model2 <- table(mailtype_model2,spamtest[,58])
cm_model3 <- table(mailtype_model3,spamtest[,58])

##Accuracy

acc_model1 <- sum(diag(cm_model1))/sum(cm_model1)
acc_model2 <- sum(diag(cm_model2))/sum(cm_model2)
acc_model3 <- sum(diag(cm_model3))/sum(cm_model3)

cat("Accuracy of Model 1",acc_model1)
```

```
## Accuracy of Model 1 0.9225199
```

```
cat("\nAccuracy of Model 2",acc_model2)
```

```
##
## Accuracy of Model 2 0.9254164
```

```
cat("\nAccuracy of Model 3",acc_model3)
```

```
##
## Accuracy of Model 3 0.916727
```

```
#Error
#te_model1 <- 1-acc_model1
```

```r
te_model2 <- 1-acc_model2
#te_model3 <- 1-acc_model3

cat("\nError of model",te_model2)
```

```
##
## Error of model 0.07458364
```

The model with value 1 for parameter C is the best model as the error rate is less for that model.

The parameter C defines the cost of constraints violation this is the 'C'-constant of the regularization term in the Lagrange formulation.

The cost parameter penalizes large residuals. As cost increases, the model becomes more flexible with less misclassification. Cost parameter also helps to adjust the bias/variance trade-off. The greater the cost parameter, the more variance in the model and the less bias and fewer misclassifications are allowed.

In 3 models above the model with c value 0.5 will have more bias where as the model with c as 5 will have higher variance. The model with c as 1 is the best model of the 3 considering bias variance tradeoff.

ml-1

Prudhvi Peddmallu

24 April 2019

```
# Required libraries
library(readxl) # reading excel,library(dplyr),library(stringr) # string manipulation,library(kknn) # K
# library(fastICA) # ICA,library(kernlab) # SVM,library(neuralnet) # Neural networks,library(mvtnorm),l
-------------------------------------------------------------
inf = read.csv("given_files/Influenza.csv")
lambdas = seq(10, 2910, 100)
log_poisson = function(lambda, y){
lp = -lambda + y*log(lambda) - sum(log(1:y))
}
calc_llik = function(lambda){
m_llik = 0
for(i in 1:nrow(inf)){
m_llik = m_llik - log_poisson(lambda, inf$Mortality[i])
}
return(m_llik)
}
llik_df = data.frame()
for(lamb in lambdas){
llik_lb = calc_llik(lamb)
llik_df = rbind(llik_df, data.frame(lambda = lamb, minus_log_likelihood = llik_lb))
}
# Plot that shows the dependence of minus log-likelihood on lambda
ggplot(llik_df) + geom_line(aes(x=lambda, y=minus_log_likelihood)) +
ylab("Minus Log likelihood") + xlab("Lambda") + ggtitle("Minus Log likelihood vs Lambda")
inf2 = as.data.frame(scale(inf[ , -3]))
inf2$Mortality = inf$Mortality
n = dim(inf2)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
inf2_train = inf2[id,]
inf2_test = inf2[-id,]
# Perform k-fold CV to select lambda for a LASSO model
inf2_cv_model = cv.glmnet(as.matrix(inf2_train[ , -9]), matrix(inf2_train$Mortality),
alpha = 1, family = "poisson", lambda = seq(0, 30, 0.01))
plot(inf2_cv_model)
# Find optimal lambda and model
optimal_lambda = inf2_cv_model$lambda.min
print(paste("Optimal lambda: ", optimal_lambda))
optimal_lasso_model = glmnet(as.matrix(inf2_train[ , -9]), matrix(inf2_train$Mortality),
alpha = 1, family = "poisson", lambda = optimal_lambda)
pred_trn = predict(optimal_lasso_model, newx = as.matrix(inf2_train[ , -9]), type = "response")
pred_tst = predict(optimal_lasso_model, newx = as.matrix(inf2_test[ , -9]), type = "response")
train_mse = sum((pred_trn - inf2_train$Mortality)^2)/nrow(inf2_train)
test_mse = sum((pred_tst - inf2_test$Mortality)^2)/nrow(inf2_test)
print(paste("Train MSE: ", train_mse))
print(paste("Test MSE: ", test_mse))
print(coef(optimal_lasso_model))
```

```r
print(paste("alpha = ", coef(optimal_lasso_model)[1]))
print(paste("exp(alpha) = ", exp(coef(optimal_lasso_model)[1])))
inf_9596 = inf[inf$Year == 1995 | inf$Year == 1996, ]
# Benjamini Hochberg method
# Function to compute p-value for given column with Conference
get_p_value = function(col){
form = as.formula(paste(col, "~", "Year"))
res = t.test(form, data = inf_9596, alternative = "two.sided", paired = TRUE)
res$p.value
}
# Compute p-values
p_values = sapply(colnames(inf_9596[ , -1]), get_p_value)
p_value_df = data.frame(feature = colnames(inf_9596[, -1]),
p_value = p_values)
p_value_df = p_value_df[order(p_value_df$p_value), ]
p_value_df$feature_num = 1:nrow(p_value_df)
# Removing Week because the p.value is NaN
p_value_df = p_value_df[p_value_df$feature!="Week", ]
alpha = 0.05
M = nrow(p_value_df)
L = max(which(p_value_df$p_value < (alpha * p_value_df$feature_num / M)))
p_L = p_value_df[L, "p_value"]
# Set hypotheses results
p_value_df$hypo_res = ifelse(p_value_df$p_value <= p_L, "Rejected", "Confirmed")
ggplot(p_value_df) +
geom_point(aes(x = feature_num, y = p_value, color = "Confirmed")) +
geom_abline(slope = alpha/M, intercept = 0) +
labs(color = "Hypothesis Result") +
geom_vline(xintercept = L, linetype = "dashed") +
xlab("Feature number") + ylab("p-value") +
ggtitle("Benjamini-Hochberg plot for hypotheses analysis")
rejected_features = p_value_df$feature[p_value_df$hypo_res == "Rejected"]
pca_inf = prcomp(inf2_train[,-9])
lambda = pca_inf$sdev^2
var_prop = lambda / sum(lambda)
cum_var_prop = cumsum(var_prop)
var_prop_df = data.frame(var_prop = var_prop, cum_var_prop = cum_var_prop,
pc_num = 1:length(var_prop))
kable(var_prop_df)
inf_pc_x = pca_inf$x
infpc_cv_model = cv.glmnet(inf_pc_x, matrix(inf2_train$Mortality),
alpha = 1, family = "poisson", lambda = seq(0, 50, 0.1))
plot(infpc_cv_model)
# Find optimal lambda and model
optimal_pc_lambda = infpc_cv_model$lambda.min
print(paste("Optimal lambda: ", optimal_pc_lambda))
optimal_pc_lasso_model = glmnet(inf_pc_x, matrix(inf2_train$Mortality),
alpha = 1, family = "poisson", lambda = optimal_pc_lambda)
print(coef(optimal_pc_lasso_model))
```