Fahad Hameed-RMD Final

K-Nearest

```
library(ggplot2)  # K NEAREST NEIGHBOR
library(kknn)
library(readxl)
data <- read_excel("spambase.xlsx")
data <- as.data.frame(data) # convert into data frame
n=dim(data)[1]  # n = 2740  Length of data
set.seed(12345)
id=sample(1:n, floor(n*0.5))  # Divide data into 50% 2740/2 = 1370 Observations
train=data[id,] #train is data
test=data[-id,] #test data is newData
knearest=function(data,k,newdata) {
  n1=dim(data)[1]
  n2=dim(newdata)[1]
  p=dim(data)[2]
  Prob=numeric(n2)
  X = as.matrix(data[,-p]) # i Compute xhat
  Y = as.matrix(newdata[-p]) # change xn to Yn
  X_hat = X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)
  Y_hat = Y/matrix(sqrt(rowSums(Y^2)), nrow = n2 , ncol = p - 1)
  C <- X_hat %*% t(Y_hat) # iii Compute matrix C as abs(c(Xi,Ymj))
  D <- 1 - C # distacne matrix calculate
  for (i in 1:n2 ) {
    Ni <- order(D[i,])  # order will return the index after sorting from smaller to larger
    N_i <- data[Ni[1:k],"Spam"]  # get k values
    Prob[i] <- sum(N_i) / k   }
  return(Prob) #return proabilities
}
probalities <- knearest(train,5, test)
probalities <- ifelse(probalities > 0.5, 1,0) # if probability is > 0.5 set to 1 else to 0
conf_mat <- table(spam = test[,ncol(data)] , predicted_val = probalities) # Confusion Matrix
mc <- 1-sum(diag(conf_mat))/sum(conf_mat) # Misclassification Rate

ROC <- function(Y, Yfit, p) {
  m=length(p) # length of p
  TPR=numeric(m) # will create a vector of length p with values 0
  FPR=numeric(m)
  for(i in 1:m)  {
    t <- table(Y,Yfit>p[i])  # misclassification rate table
    TPR[i] <-  t[2,2]/sum(t[2,]) #True +ve Value / Sum of Positive Actual Values
    FPR[i] <-  t[1,2]/sum(t[1,]) #False +ve Value / Sum of Negative Actual Values
  }
  return (list(TPR=TPR,FPR=FPR))
}

pi_values <- seq(from = 0.05, to= 0.95 , by=0.05)
Y <- train[,ncol(data)]
knearest_p <- knearest(train, 5 , test) #knearst k = 5
```

```r
roc_curve_knearest <- ROC(Y, knearest_p , pi_values)
X<-  as.data.frame(roc_curve_knearest)
ggplot() + geom_line(data = X, aes(x = X$FPR, y = X$TPR), color = "red") +
           ggtitle("ROC curve Knearnest()")+xlab("FPR") +ylab("TPR")
sensitivity_kn <- 1 - roc_curve_knearest$FPR
```

Inference about lifetime of machines

The distrubution given is a exponential distrubution. The pdf for the exponetial distrubution is: $p(x|\theta) = \theta e^{-}x\theta$ The likelihood is the following: $\prod p(x|\theta) = \theta e^{-}\theta \sum x_i$ Finaly the log-likelihood was computed to: $log\, p(x|\theta) = n\, log(\theta) - \theta \sum_{n=1}^{N} \mathrm{x_i}$

```r
library(readxl) # Read the data and transform it in to a vector.
data <- read_excel("machines.xlsx") #Import the data to R
data$Length <- as.numeric(data$Length)
```

Plot the curve showing the dependence of log-likelihood on θ where the entire data is used for fitting.

```r
length_histogram <- hist(data$Length, plot=FALSE)
multiplier <- length_histogram$counts / length_histogram$density
multiplier <- max(multiplier[which(!is.nan(multiplier))])
length_density <- density(data$Length)
length_density$y <- length_density$y * multiplier
log_likelihood <- function(x, theta) {
  log(theta * exp(-theta * x))
}
thetas <- seq(0.1, 5, by=0.1)
log_likelihoods <- sapply(thetas, function(x) {
  sum(log_likelihood(x=data$Length, theta=x))
})
best_theta <- thetas[which.max(log_likelihoods)]
plot(length_histogram, col="orange", main="Machine Distribution",
     xlab="Lifetime", ylab="Frequency", xlim=c(0, 5))
lines(length_density, col="blue", lwd=2)
plot(thetas, log_likelihoods, main="Log-Likelihood", col="orange",
     xlab="Theta", ylab="Log-Likelihood", type="l", lwd=2)
log_likelihoods_6 <- sapply(thetas, function(x) {
  sum(log_likelihood(x=data$Length[1:6], theta=x))
})
ylim <- c(min(min(log_likelihoods), min(log_likelihoods_6)),
          max(max(log_likelihoods), max(log_likelihoods_6)))
plot(thetas, log_likelihoods, col="orange",
     main="Log-Likelihood", xlab="Theta", ylab="Log-Likelihood",
     type="l", ylim=ylim, lwd=2)
lines(thetas, log_likelihoods_6, col="blue", lwd=2)
plot(thetas, log_likelihoods_6, type="l", main="Log-Likelihood",
     xlab="Theta", ylab="Log-Likelihood", col="blue", lwd=2)
prior <- function(theta, lambda=10) {
  lambda * exp(-lambda * theta)
}
log_posteriors <- sapply(1:length(thetas), function(i) {
  log_likelihoods[i] + log(prior(thetas[i]))
})
```

```r
plot(thetas, log_posteriors, col="green",
     main="Log-Posterior", xlab="Theta", ylab="Log-Posterior",
     type="l", lwd=2)
set.seed(12345)
new_data <- rexp(50, best_theta)
par(mfrow=c(1, 2))
hist(new_data, breaks=14, main="Distrubtion of Generated Data",
     xlab="Lifetime", ylab="Frequency", xlim=c(0, 5), col="orange")
hist(data$Length, breaks=8, main="Distrubtion of Machine Data",
     xlab="Lifetime", ylab="Frequency", col="orange")
```

Feature selection by cross-validation in a linear model.

```r
mylin=function(X,Y, Xpred){
  Xpred1=cbind(1,Xpred)
  X= cbind(1,X)
  beta <- solve(t(X) %*% X) %*% (t(X) %*% Y) # Lecture 1d slide(7)
  Res=Xpred1 %*% beta
  return(Res)
}

myCV=function(X = as.matrix(swiss[,2:6]) , Y = swiss[[1]] , Nfolds = 5){
  n=length(Y)
  p=ncol(X)
  set.seed(12345)
  ind=sample(n,n)
  X1=X[ind,]
  Y1=Y[ind]
  sF=floor(n/Nfolds)
  MSE=numeric(2^p-1)
  Nfeat=numeric(2^p-1)
  Features=list()
  curr=0
  #we assume 5 features.
  for (f1 in 0:1)
    for (f2 in 0:1)
      for(f3 in 0:1)
        for(f4 in 0:1)
          for(f5 in 0:1){
            model = c(f1,f2,f3,f4,f5)
            if (sum(model)==0) next()
            SSE=0
            seq_1 <- seq(from=1, to=n, by = sF)
            seq_2 <- seq(0, n, sF)
            ind_ <- which(model == 1)
            for (k in 1:Nfolds){
              i <- seq_1[k]
              j <- seq_2[k+1]
              # Selecting n/kfold indices
              folds_ <- ind[i:j]
              Xtest <- X1[folds_,ind_]
```

```
            Xtrain <- X1[-folds_,ind_]
            Ytrain <- Y1[-folds_]
            Yp <- Y1[folds_]
            Ypred <- mylin(X = Xtrain,Y = Ytrain,Xpred = Xtest)
            SSE=SSE+sum((Ypred-Yp)^2)
          }
          curr=curr+1
          MSE[curr]=SSE/n
          Nfeat[curr]=sum(model)
          Features[[curr]]=model
        }
  plot(Nfeat,MSE, type = "p", xlab = "Number of features",
       ylab = "MSE", pch=19,cex=1)
  #MISSING: plot MSE against number of features
  i=which.min(MSE)
  return(list(CV=MSE[i], Features=Features[[i]]))
}
myCV(as.matrix(swiss[,2:6]), swiss[[1]], 5)
```

Linear regression and regularization ,lasso , ridge regression

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 Mi in which Moisture is normally distributed, and the expected Moisture is a polynomial function of Protein including the polynomial terms up to power i (i.e Mi is a linear model, M2 is a quadratic model and so on). Report a probabilistic model that describes Mi.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 Mi=1,2,3...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 λ and report how the coefficients change with λ. 6. Repeat step 6 but fit LASSO instead of the Ridge regression and compare the plots from steps 6 and 7. Conclusions? 7. Use cross-validation to find the optimal LASSO model (make sure that case λ =0 is also considered by the procedure) plot showing the dependence of the CV score and comment how the CV score changes with λ.

```
library(MASS)
library(readxl)
library(Matrix)
library(glmnet)

data = read_excel("tecator.xlsx")
plot(data$Protein,data$Moisture)

set.seed(12345)
n = nrow(data)
train_indexes = sample(1:n,floor(n*0.5))
train_data = data[train_indexes,]
test_data = data[-train_indexes,]
power = 6
```

```r
train_error = matrix(0,power,1)
test_error = matrix(0,power,1)
for(i in 1:power)
{
  model = lm(Moisture ~ poly(Protein,i), data=train_data)
  train_predictions = predict(model,train_data)
  test_predictions = predict(model,test_data)
  train_error[i,] = mean((train_data$Moisture - train_predictions)^2) # MSE Formula below
  test_error[i,] = mean((test_data$Moisture - test_predictions)^2)#mean((predicted-actual Data)^2)
}
ylim = c(min(rbind(train_error,test_error)),max(rbind(train_error,test_error)))#Limits of PLOT
plot(1:power,train_error, col="Green", ylim=ylim)
lines(1:power,train_error, col="Green")
points(1:power,test_error,col="Red")
lines(1:power,test_error, col="Red")

# High Bias -> High error on training set
# High Variance -> Fits training data well but high error in test set

model = lm(Fat ~ ., data=data)
steps = stepAIC(model,direction="both", trace=FALSE)
coeff_aics = steps$coefficients
n_coeff_aics = length(coeff_aics)
print(n_coeff_aics)

rl_data <- data.matrix(data)
y <- rl_data[,102] # Select Fat as the only response...
X <- rl_data[,2:101] # Select Channel1-Channel-100 features.
ridge <- glmnet(X, y, alpha = 0) # Fit with the Ridge regression. Alpha=0
plot(ridge, xvar="lambda", label=TRUE)
lasso <- glmnet(X, y, alpha = 1) # Fit with the Lasso regression. Alpha=1
plot(lasso, xvar="lambda", label=TRUE)

kfoldcv <- cv.glmnet(X, y,  type.measure="mse", nfolds=20) #cross validation on lasso
feature_selection <- coef(kfoldcv, s = "lambda.min")
plot(kfoldcv)
```

LDA and Logistic Regression

```r
library(ggplot2)
data<- read.csv2("australian-crabs.csv" ,sep = ",",dec=".")
p <- ggplot(data, aes(x=CL, y=RW)) + geom_point(aes(color=sex), size=2 ) +
  scale_color_manual (values = c('blue', 'red')) +
  labs(x="CL carspace length", y="RW rear Width", colour="Classes") +
  ggtitle("original data")

X<- data.frame(RW=data$RW , CL=data$CL )
Y <- data$sex
#1.2
library(MASS)
disc_fun=function(label, S)
```

```
{
  X1 = X[Y==label,]
  mean_v <- c(mean(X1$RW) ,mean(X1$CL))
  covaiance_mat_inverse <- solve(S)
  prior_prob <- nrow(X1) / nrow(X)
  w1 <- covaiance_mat_inverse %*% mean_v
  b1 <- ((-1/2) %*% t(mean_v) %*% covaiance_mat_inverse %*% mean_v) + log(prior_prob)
  w1<- as.vector(w1)
  return(c(w1[1], w1[2], b1[1,1]))
}
X1=X[Y=="Male",]
X2=X[Y=="Female",]
S=cov(X1)*dim(X1)[1]+cov(X2)*dim(X2)[1]
S=S/dim(X)[1]
#discriminant function coefficients
res1=disc_fun("Male",S)
res2=disc_fun("Female",S)
#1.2
#decision boundary coefficients 'res'
res <- c( -(res1[1]-res2[1]) , (res2[2]-res1[2]), (res2[3]-res1[3]))
# classification
d=res[1]*X[,1]+res[2]*X[,2]+res[3]
Yfit=(d>0)
plot(X[,1], X[,2], col=Yfit+1, xlab="CL", ylab="RW")
#slope and intercept
slope <- (res[2] / res[1] ) * -1
intercept <- res[3] /res[1] * -1
#1.3
#plot decision boundary
X<- cbind(X,sex=Y)
p <- ggplot(X, aes(x=CL, y=RW)) + geom_point(aes(color=sex), size=2 ) +
  scale_color_manual (values = c('blue', 'red')) +
  labs(x="CL carspace length", y="RW rear Width", colour="Classes") +
  geom_abline(slope = slope, intercept = intercept) +
  ggtitle("Descion Boundary LDA")
#1.4
glm1 <- glm(sex ~ CL + RW,family=binomial(link="logit"), data=data)
slope1 <- -(glm1$coefficients[2] / glm1$coefficients[3] )
intercept1 <- -(glm1$coefficients[1] /glm1$coefficients[3] )
print(qplot(
  x =data$CL,
  y = data$RW,
  data = data,
  color = data$sex ,
  main="CL vs RW",
  xlab="Carapace Length", ylab = "Rear Width")
  +geom_abline(slope = slope1, intercept = intercept1,colour='purple')+ggtitle("CL Vs RW in Logistic Reg
cat("Decision boundary with linear regression:",slope1, "+",intercept1, "* k\n")
```

Tree & Tree Prune Analysis of Credit Scoring

```
library(readxl)
library(tree)
library(e1071)
# Importing Data - 2.1
data <- read_excel("creditscoring.xls")
data <- as.data.frame(data)
data$good_bad <- as.factor(data$good_bad)
# Dividing Data into three Train(50%) Test(25%) Validation(25%)
n = nrow(data)
set.seed(12345)
n=dim(data)[1]
# 50% Training Data
id=sample(1:n, floor(n*0.5))
train=data[id,]
# 25% validation & testing Data
Sub_id = data[-id,]
m = dim(Sub_id)[1]
part1 = sample(1:m, floor(m*0.5))
validation = Sub_id[part1,]
testing = Sub_id[-part1,]
# Step 2
# Fitting data using Deviance and gini
tree_deviance = tree(as.factor(good_bad) ~ ., data = train, split = "deviance")
tree_gini = tree(as.factor(good_bad) ~ ., data = train, split = "gini")
summary(tree_gini) # to find number of nodes
# Prediction
## Misclassification for training data
devi_yfit = predict(tree_deviance, newdata = testing,type="class")
gini_yfit = predict(tree_gini, newdata = testing,type="class")
plot(tree_deviance)
plot(tree_gini)
devi_table = table(devi_yfit,testing$good_bad)
gini_table = table(gini_yfit,testing$good_bad)
devi_table

# Missclassification rate Deviance
missclass_devi <- 1-sum(diag(devi_table))/sum(devi_table)
missclass_devi
gini_table
# Missclassification rate Gini
missclass_gini <- 1-sum(diag(gini_table))/sum(gini_table)
missclass_gini
## Misclssification for test data:
devi_yfit = predict(tree_deviance, newdata = testing,type="class")
gini_yfit = predict(tree_gini, newdata = testing,type="class")
plot(tree_deviance)
plot(tree_gini)
devi_table = table(devi_yfit,testing$good_bad)
gini_table = table(gini_yfit,testing$good_bad)
devi_table
# Missclassification rate Deviance
```

```
missclass_devi <- 1-sum(diag(devi_table))/sum(devi_table)
missclass_devi
gini_table
# Missclassification rate Gini
missclass_gini <- 1-sum(diag(gini_table))/sum(gini_table)
missclass_gini
### Step 3
index = summary(tree_deviance)[4]$size
trainScore = rep(0,index)
testScore = rep(0,index)
# Graph training and validation
for(i in 2:index) {
  prunedTree=prune.tree(tree_deviance,best=i)
  pred=predict(prunedTree, newdata=validation,type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:index,trainScore[2:index], col="Red",type = "b", main = "Dependence of Deviance",
     ylim=c(min(testScore[2:index]),max(trainScore)), pch=19, cex=1, ylab="Deviance")
points(2:index,testScore[2:index],col="Blue",type="b", pch=19, cex=1)
# misclassification rate for test data
missclass_test_t = prune.tree(tree_deviance, best = 4)
summary(missclass_test_t)

yfit = predict(missclass_test_t, newdata = testing, type="class")
valid_ = table(testing$good_bad,yfit)
print("Confusion Matrix")
valid_
mc <- 1-sum(diag(valid_))/sum(valid_)
print("Misclassification rate")
mc
plot(missclass_test_t)
text(missclass_test_t)
### Step 4
# Naive Bayes 2.4
naye = naiveBayes(good_bad ~., data=train)
nav_test = predict(naye, newdata = testing[,-ncol(testing)], type = "class") # -ncol(testing) column
nav_train = predict(naye,newdata = train[,-ncol(train)])  # Removing good-bad
# Confusion Matrix Using Naive Bayes
nv_tbl_test = table(testing$good_bad,nav_test)
print(nv_tbl_test)
nv_tbl_train <- table(train$good_bad,nav_train)
print(nv_tbl_train)
# Missclassification train data value Using Naive Bayes
mc_nav_train <- 1-sum(diag(nv_tbl_train))/sum(nv_tbl_train)
cat("Misclassification train data value Using Naive Bayes is:",mc_nav_train)
# Missclassification test data value Using Naive Bayes
mc_nav_test <- 1-sum(diag(nv_tbl_test))/sum(nv_tbl_test)
cat("Misclassification test data value Using Naive Bayes is:",mc_nav_test)
### Step 5
# Naive Bayes With loss matrix 2.5
naye = naiveBayes(good_bad ~ ., data = train)
# Predicting using Naive
```

```
nav_test = predict(naye, testing[,-ncol(testing)] , type="raw")
nav_train = predict(naye, train[,-ncol(train)] , type="raw")
# applying loss matrix if greater then 10 True else False
nav_test = (nav_test[, 2] / nav_test[, 1]) > 10 # check with loss Matrix of 0,1,10,0 values
nav_train = (nav_train[, 2] / nav_train[, 1]) > 10
# confusion matrix for train & test
naive_table = table(testing$good_bad,nav_test)
naive_table_train = table(train$good_bad,nav_train)
# missclasification for train & test
naive_table_train
1-sum(diag(naive_table_train))/sum(naive_table_train)

naive_table
1-sum(diag(naive_table))/sum(naive_table)
```

Uncertainty estimation

1. Reorder data with respect to the increase of MET and plot EX versus MET.
2. Use package tree and **fit a regression tree model** with target EX and feature MET in which the **number of the leaves is selected by cross-validation**, use the entire data set and set minimum number of observations in a leaf equal to 8 (setting minsize in tree.control). Report the selected tree. Plot the original and the fitted data and **histogram of residuals**. Comment on the distribution of the residuals and the quality of the fit.
3. Compute and plot the **95% confidence bands for the regression tree model** from step 2 (fit a regression tree with the same settings and the same number of leaves as in step 2 to the resampled data) by using a **non-parametric bootstrap**. Comment whether the band is smooth or bumpy and try to explain why. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable.
4. Compute and plot the **95% confidence and prediction bands** the regression tree model from step 2 (fit a regression tree with the same settings and the same number of leaves as in step 2 to the resampled data) by using a parametric bootstrap, assume ????????~????????(????????????????,????????2) where ???????????????? are labels in the tree leaves and ????????2 is the residual variance. Consider the width of the confidence band and comment whether results of the regression model in step 2 seem to be reliable. Does it look like only 5% of data are outside the prediction band? Should it be?
5. Consider the histogram of residuals from step 2 and suggest what kind of bootstrap is actually more appropriate here.

```
# Assignment 3
library(tree)
# 3.1 Data import, reorder and Plot
set.seed(12345)
data = read.csv2("State.csv", header = TRUE)
data = data[order(data$MET),] # reordering data with increase of MET variable
plot(EX ~ MET, data = data, pch = 19, cex = 1,col="blue")
# 3.2
set.seed(12345)
control_parameter = tree.control(nobs = nrow(data),minsize = 8)
fit_tree = tree(formula = EX ~ MET,data = data,control = control_parameter)
leave_fit = cv.tree(fit_tree)
plot(leave_fit$size, leave_fit$dev, main = "Deviance Vs Size of Tree" ,
type="b",col="red", pch= 19,cex=1)
```

```r
op_tree = prune.tree(fit_tree,best = leave_fit$size[which.min(leave_fit$dev)])

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

fitted_val = predict(op_tree, newdata=data)

plot(data$MET, data$EX)
points(data$MET, fitted_val, col="red")

hist(residuals(op_tree))
# 3.3 Non-Paramatric Bootstrap
library(boot)
f_np = function(data,index){
  sample = data[index,]
  Ctrl = tree.control(nrow(sample), minsize = 8)
  fit = tree( EX ~ MET, data=sample, control = Ctrl)
  optimal_tree = prune.tree(fit, best= leave_fit$size[which.min(leave_fit$dev)])
  return(predict(optimal_tree, newdata=data))
}
np_bs = boot(data, statistic = f_np, R=1000)
conf_bound = envelope(np_bs,level=0.95) # For 95% Confidence interval

predictions = predict(op_tree,data)
plot(np_bs)
fig_data = data.frame(orig = data$EX, x=data$MET, pred=predictions,
                      upper=conf_bound$point[1,], lower=conf_bound$point[2,])
fig = ggplot(fig_data, aes(x,predictions,upper,lower))
p = fig + geom_point(aes(x, pred)) +
  geom_point(aes(x, orig),colour="blue") +
  geom_line(aes(x,upper),colour="red") +
  geom_line(aes(x,lower),colour="red")
p
# 3.4 Paramatric Bootstrap
set.seed(12345)
parama_conf = function(data){
  controll = tree.control(nrow(data), minsize = 8)
  fit = tree( EX ~ MET, data=data, control = controll)
  op_tree = prune.tree(fit, best=leave_fit$size[which.min(leave_fit$dev)])
  return(predict(op_tree, newdata=data))
}
param_predict = function(data){
  controll = tree.control(nrow(data), minsize = 8)
  fit = tree( EX ~ MET, data=data, control = controll)
  op_tree = prune.tree(fit, best=leave_fit$size[which.min(leave_fit$dev)])
  predictions = predict(op_tree, newdata=data)
  return(dbinom(nrow(data),predictions,sd(resid(fit))))
}
rnd = function(data, model){
  sample = data.frame(MET=data$MET, EX=data$EX)
  sample$EX = rnorm(nrow(data), predict(model,newdata=data),sd(resid(model)))
  return(sample)
}
```

```
set.seed(12345)
param_boot_conf = boot(data, statistic = parama_conf, R=1000, mle = op_tree,
                       ran.gen = rnd, sim = "parametric")
confidence_bound_param = envelope(param_boot_conf, level=0.95)
param_boot_predict = boot(data, statistic = param_predict, R=1000, mle = op_tree,
                          ran.gen = rnd,sim = "parametric")
prediction_bound_param = envelope(param_boot_predict, level=0.95)
plot(param_boot_conf)
plot(param_boot_predict)
predictions = predict(op_tree,data)
fig_data = data.frame(orig = data$EX, x=data$MET, pred=predictions,
                          upper_c=confidence_bound_param$point[1,],
                          lower_c=confidence_bound_param$point[2,],
                          upper_p=prediction_bound_param$point[1,],
                          lower_p=prediction_bound_param$point[2,])
para_plot = ggplot(fig_data, aes(orig,x,pred,upper_c,lower_c, upper_p, lower_p))
para_plot = para_plot + geom_point(aes(x, pred)) + geom_point(aes(x, orig),colour="blue") +
              geom_line(aes(x,upper_c),colour="red")+geom_line(aes(x,lower_c),colour="red")+
              geom_line(aes(x,upper_p),colour="green")+geom_line(aes(x,lower_p),colour="green")
para_plot
```

From the plot it can be observed that the data is scattered that is variance is high thus for the this type of data, decision trees would be the appropriate method

Principal components

1. Conduct a standard PCA by using the feature space and provide a plot explaining how much variation is explained by each feature. Does the plot show how many PC should be extracted? Select the minimal number of components explaining at least 99% of the total variance. Provide also a plot of the scores in the coordinates (PC1, PC2). Are there unusual diesel fuels according to this plot?
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?
3. Perform Independent Component Analysis with the number of components selected in step 1 (set seed 12345). Check the documentation for the fastICA method in R and do the following:
   a. Compute W'= K.W and present the columns of W' in form of the trace plots. Compare with the trace plots in step 2 and make conclusions. What kind of measure is represented by the matrix W'?
   b. Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.
4. Fit a PCR model in which number of components is selected by cross validation to the data, use seed 12345. Provide a plot showing the dependence of the mean-square predicted error on the number of the components in the model and comment how many components it is reasonable to select.

```
library(ggplot2)
library(fastICA)
library(pls)
library(reshape2)

data <- read.csv2("NIRSpectra.csv")

X <- scale(data[, -ncol(data)]) # removing column 127 viscosity
y <- data[, ncol(data)]
```

```r
pca <- prcomp(X) # can also do here scaling using scale=TRUE

lambda <- pca$sdev^2 # Eigenvalues
variances <- (lambda / sum(lambda))* 100 # proportion of variation

var99_comp_count <- which.max(cumsum(variances) > 99) # which show variance greater then 99%
components <- as.data.frame(pca$x[, 1:var99_comp_count]) # extracting first two components that show va

pc_comps <- 1:10
plot_data <- data.frame(x=pc_comps, Variance=variances[pc_comps])

ggplot(plot_data, aes(x=x, y=Variance)) +
  geom_bar(stat="identity") +
  scale_x_discrete(limits=pc_comps, labels=as.numeric(pc_comps)) +
  xlab("Principal Component")

ggplot(components) +
  geom_point(aes(x=PC1, y=PC2)) +
  scale_x_continuous(breaks=pretty(components$PC1, n=6)) +
  scale_y_continuous(breaks=pretty(components$PC2, n=6))

U<-pca$rotation#Loadings are the coordinates of the principal components in the original vector space

r1 <- pca$rotation[,1] ; r2 <- pca$rotation[,2]
ggplot() + geom_point(aes(x = 1:length(r1), y = r1, col = "PC1")) +
           geom_point(aes(x = 1:length(r2), y = r2, col = "PC2")) +
           labs(title = "Plot of loadings", x = "Index", y = "Loadings")

ica <- fastICA(X, var99_comp_count, alg.typ="parallel", fun="logcosh", alpha=1,
               method="R", row.norm=FALSE, maxit=200, tol=1e-06, verbose=FALSE)

Wp <- ica$K %*% ica$W # W`=K*W summary(ica)
d <- dim(Wp)[1]
Wp1 <- Wp[,1]
Wp2 <- Wp[,2]
ggplot() + geom_point(aes(x = 1:d, Wp1, col = "Wp1")) + geom_point(aes(x = 1:d, Wp2, col = "Wp2")) +
           labs(title = "Plot of columns of W'", x = "Index", y = "W'")

score1 <- ica$S[,1]
score2 <- ica$S[,2]
d1 <- dim(score1)[1]
ggplot() + geom_point(aes(x = score1, y = score2)) +
labs(title = "Plot of scores of principal components"
, x = "Index" , y = "W'")

predictors <- paste("X", seq(750, 1000, 2), sep="")
f <- formula(paste("Viscosity ~ ", paste(predictors , collapse = " + ")))

set.seed(12345)
pcr.fit <- pcr(f, data = data, validation = "CV")
validationplot(pcr.fit,val.type = "MSEP")
```

SPLINE, GAM, GLM

```
# library(readxl)
# library(ggplot2)
# library(reshape2)
#
# data <- read_excel("influenza.xlsx")
# ggplot(data)+geom_point(aes(x = Time, y = Mortality, color = "Mortality")) +
#   geom_point(aes(x = Time, y = Influenza, color = "Influenza"))
# # From plot it can be observed that influenza cases have the peaks at the same
# # points where the morltality plot has peaks,
# # it shows that when there is increasing rate of mortality, the influenza cases have increased
# # or it can be interpreted as increasing number of influenza cases have affected the mortality rate
#
# library(mgcv)
# k_week = length(unique(data$Week)) #Number of unique weeks in data
# gm_model<-gam(Mortality~Year+s(Week,k=k_week),data=data,family="gaussian",method="GCV.Cp") #GCV.Cp-Ge
# cat("Intercept =" , coef(gm_model)["(Intercept)"], "\n") # is W0
# cat("Year coefficent =" , coef(gm_model)["Year"])  # is W1
#
# #Probilistic model: y = wo + w1x1 + w2x1^2 + e  (where wo=intercept,  w1= est value of 1st var, w2= e
# #Probilistic model: $y = -680.589 + 1.233*x1 + s(Week) + {\epsilon}~N(0,{\sigma}^2)
# pred <- predict(gm_model)
# df_plot <- data.frame(time= data$Time, mortality = data$Mortality, pred = as.vector(pred))
#
# p1 <- ggplot(df_plot, aes(x= time, y = mortality)) +
#   geom_point(colour= "blue") +
#   geom_line(aes(time,pred),colour = "red") + coord_cartesian(xlim = c(1995,2003))
# p1
# plot(gm_model) # Spline component
# summary(gm_model) # to check significant values using p values
#
# m1<-gam(data$Mortality~data$Year+s(data$Week,k=52,sp=0.1),data = data,family = "gaussian")
# pred2<-predict(m1)
# m2<-gam(data$Mortality~data$Year+s(data$Week,k=52,sp=100),data = data,family = "gaussian")
# pred3<-predict(m2)
#
# df_plo3<-data.frame(pred2=as.vector(pred2),pred3=as.vector(pred3),
#                     mortality=data$Mortality,time=data$Time)
# p3<-ggplot(data=df_plo3,aes(x=time))+
#   geom_point(aes(y=mortality,colour="Actual Values"))+
#   geom_line(aes(y=pred2,colour="Predicted Values with very low sp"))+
#   geom_line(aes(y=pred3,colour="Predicted Values with very high sp"))+
#   scale_colour_manual("Legend",
#                       breaks = c("Predicted Values with very low sp",
#                                  "Predicted Values with very high sp","Actual Values"),
#                       values = c("red","blue","#99FF00"))+
#   ggtitle("Actual Mortality vs Predicted with different values of sp")
# p3
#
# summary(m1); summary(m2)
# # the very high annd very low values of penalty factor leads to underfitting of model,
# # as it is evident from the plot in which green line represents the predicted values when
```

```
# # penalty factor is too high while red line represents the mortality when penalty factor is too low
# # deviance and degrees of freedom decreases with increase in penalty factor
#
# df_plot <- data.frame(time= data$Time, influenza = data$Influenza, residual = as.vector(gm_model$resi
#
# p2 <- ggplot(df_plot, aes(x= time, y = influenza)) +
#   geom_point(colour= "blue") +
#   geom_line(aes(time,residual),colour = "red")
# p2
#
# # Yes, it is evident from the plot that the temporal pattern in residuals seems to be correlated
# # to the outbreak of influenza since the peaks in influenza occurs relative to the peaks in
# # residuals
#
# w <- unique(data$Week)
# y <- unique(data$Year)
# i <- unique(data$Influenza)
#
# fit_f <- gam(data$Mortality ~ s(data$Year,k=length(y)) + s(data$Week,k=length(w))
#          +s(data$Influenza,k=length(i)),data=data,family="gaussian",method="GCV.Cp")
# pred_f <- predict(fit_f)
#
# par(mfrow=c(2,2))
# plot(fit_f)
#
# # It can be illustrated from the plots of spline components that mortality does not depend much on
# # year and have little change annually that is with weeks, but the mortality shows a significant
# # relation with influenza that is with increasing cases of influenza, mortality increases
# par(mfrow=c(1,1))
# df_plot <- data.frame(time= data$Time, mortality = data$Mortality, predicted = as.vector(pred_f))
#
# p3 <- ggplot(df_plot, aes(x= time, y = mortality)) +
#   geom_point(colour= "blue") +
#   geom_line(aes(time,predicted),colour = "red")
# p3
# # The plot of original and fitted values implies that this model is better than the previous
# # models as it gives the predicted values closest to the original values. This also indicates that
# # including influenza in modelling has a significant impact on fitting.
```

High-dimensional methods

```
# Assignment 2
data <- read.csv2("data.csv", header = TRUE, sep = ";", quote = "\"",
                  dec = ",", fill = TRUE, check.names = FALSE)
data1 <- as.data.frame(data)
data_email <- as.data.frame(data)
data_email$Conference <- as.factor(data$Conference)
n=dim(data_email)[1]
set.seed(12345)
# 70% Training Data
id=sample(1:n, floor(n*0.7))
```

```
train=data_email[id,]
# 30% validation & testing Data
test = data_email[-id,]
library(pamr)
rownames(train) <- 1:nrow(train)
which(colnames(train)=="Conference")
x <- t(train[,-4703])
y <- train[[4703]]
test_x <- t(test[,-4703])
mydata <- list(x=x, y= as.factor(y),
               geneid=as.character(1:nrow(x)),genenames = rownames(x))
model_train <- pamr.train(mydata)
cv_model <- pamr.cv(model_train, data = mydata)
pamr.plotcv(cv_model)
print(cv_model)
model_fit <- pamr.train(mydata,threshold = cv_model$threshold[which.min(cv_model$error)])
par(mfrow=c(1,1),mar=c(2,2,2,2))
pamr.plotcen(model_train, mydata, threshold = model_fit$threshold)
features = pamr.listgenes(model_train,mydata, threshold = 1.306,genenames=TRUE)
cat( paste( colnames(train)[as.numeric(features[1:10,1])], collapse='\n' ) )
ypredict <- pamr.predict(model_train, newx = test_x,type = "class", threshold = 1.306)
conf_mat <- table(ypredict, test$Conference)
misclas_centroid <- 1 - (sum(diag(conf_mat))/sum(conf_mat))

## Part 2
library(glmnet)
set.seed(12345)
response <- train$Conference
predictors <- as.matrix(train[,-4703])
elastic_model <- glmnet(x=predictors,y=response,family = "binomial",alpha = 0.5)
cv.fit <- cv.glmnet(x=predictors,y=response,family="binomial",alpha = 0.5)
cv.fit$lambda.min
par(mar=c(2,2,2,2))
plot(cv.fit)
plot(elastic_model)
predictor_test <- as.matrix(test[,-4703])
ypredict <- predict(object = elastic_model,newx = predictor_test, s = cv.fit$lambda.min,
                    type = "class", exact = TRUE)
confusion_mat <- table(ypredict,test$Conference)
misclassification <- 1 - (sum(diag(confusion_mat))/sum(confusion_mat))

library(kernlab)
x <- as.matrix(train[,-4703])
y <- train[,4703]
svm_fit <- ksvm(data = train,Conference ~ . ,kernel="vanilladot",
                scaled = FALSE)
ypred <- predict(svm_fit, newdata = test, type="response")
confusion_mat <- table(ypred,test$Conference)
misclas_svm <- 1 - sum (diag(confusion_mat))/sum(confusion_mat)



benjamini_hochberg <- function(x, y, alpha) {
```

```
  pvalues <- apply(x, 2, function(feature) {
    t.test(feature ~ y, alternative="two.sided")$p.value
  })
  m <- length(pvalues)

  sorted <- sort(pvalues)
  values <- 1:m * alpha / m

  L <- which.min(sorted <= values) - 1
  mask <- sorted <= sorted[L]
  list(mask=mask, pvalues=sorted, features=colnames(x)[order(pvalues)][mask])
}

result <- benjamini_hochberg(x=data[,-ncol(data)], y=data[, ncol(data)], alpha=0.05)
rejected <- length(result$features)

cat("Top 10 features")
cat(paste(result$features[1:10], collapse='\n' ) )

ggplot() +
  ylab("P-Value") + xlab("Index") +
  geom_point(data=data.frame(x=1:length(result$features),
                             y=result$pvalues[result$mask]),
             aes(x=x, y=y), col="red") +
  geom_point(data=data.frame(x=((length(result$features) + 1):(ncol(data) -1)),
                             y=result$pvalues[!result$mask]),
             aes(x=x, y=y), col="blue")

ggplot() +
  ylab("P-Value") + xlab("Index") +
  geom_point(data=data.frame(x=1:length(result$features),
                             y=result$pvalues[result$mask]),
             aes(x=x, y=y), col="red") +
  geom_point(data=data.frame(x=((length(result$features) + 1):150),
                             y=result$pvalues[!result$mask][1:(150 - rejected)]),
             aes(x=x, y=y), col="blue")
```

ADABOOST

```
spambase <- read_excel("spambase.csv")
spambase <- as.data.frame(spambase)
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*2/3)) # 2/3 of training data
train=spambase[id,]
test=spambase[-id,]
number_of_trees <- seq(from = 10,to = 100, by = 10)

############################## ADABOOST ########################################
adaboost <- function(ntrees)
{
```

```
  fit <- blackboost(as.factor(Spam) ~., data = train,
              control = boost_control(mstop = ntrees, nu=0.1),family = AdaExp())
  ypredict <- predict(fit, newdata = test, type= "class") # misclassification test
  conf_mat <- table(ypredict,test$Spam) # confusion Matrix here test$spam is the last column
  error_test <- 1-sum(diag(conf_mat))/sum(conf_mat) #Misclassification rate
}
error_rates_ada <- sapply(number_of_trees, adaboost)
plot(error_rates_ada,type = "b",main="Adaboost Misclassification", xlab= "Number of Trees", ylab= "Erro
     col="blue", pch=19, cex=1)
# Loss Function = exp(-y * f)
```

Random Forest

```
random_forest <- function(ntrees)
{
  fit <- randomForest(as.factor(Spam) ~ ., data=train, importance=TRUE,
                      ntree = ntrees)
  # test misclassification
  ypredict <- predict(fit, test,type ="class")
  conf_mat <- table(ypredict,test$Spam)
  error_test <- 1-sum(diag(conf_mat))/sum(conf_mat)
}
error_rates_random <- sapply(number_of_trees, random_forest)
plot(error_rates_random,type = "b",main="Random Forest Misclassification", xlab= "Number of Trees", ylab
#comparsion random forest Vs adBoost
plot(y = error_rates_ada,x=number_of_trees, type = "l", col="red",
     main= "Performance Evaluation of Adaboost Vs Random Forest",
     xlab = "Number of Trees",ylab="Misclassification Rate", ylim = c(0,0.15))
points(y = error_rates_ada,x=number_of_trees,col="red", pch=19, cex=1)
lines(y = error_rates_random,x=number_of_trees, type= "l", col = "blue")
points(y = error_rates_random,x=number_of_trees,col="blue", pch=19, cex=1)
legend("topright",legend= c("adaboost","random forest"),
       col=c("red","blue"),lty=1,cex=0.8)
```

EM algorithm for Mixtures of Multivariate Benouilli Distributions

```
mixture_model <- function(my_k=2)
{
  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=my_k # 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(j in 1:my_k) {
  mu[j,] <- runif(D,0.49,0.51)
}
pi
mu
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 assignment
  # Bernoulli distribution
  for (n in 1:N)
  {
    prob_x=0
    for (k in 1:K)
    { #Multivariate Bernouilli distributions
      prob_x = prob_x+prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))) )*pi[k] #Documentation       Eq 1
```

```
    }
    for (k in 1:K)
    {
      z[n,k] = pi[k]*prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))) ) / prob_x  #Eq 4                E
    }
  }
  #Log likelihood computation.
  likelihood <-matrix(0,nrow =1000,ncol = K)
  llik[it] <-0
  for(n in 1:N)
  {
    for (k in 1:K)
    {
      likelihood[n,k] <- pi[k]*prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
    }
    llik[it]<- sum(log(rowSums(likelihood)))
  }
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1)
  {
    if (llik[it]-llik[it-1] < min_change)
    {
      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")
      }
      break
    }
  }
  #M-step: ML parameter estimation from the data and fractional component assignments
  mu<- (t(z) %*% x) /colSums(z)
  # N - Total no. of observations
  pi <- colSums(z)/N
}
cat("value of updated pi is " , pi )
cat("\n")
sprintf("value of updated mu is")
```

```
  print(mu)
  plot(llik[1:it], type="o")
}
mixture_model(2)
mixture_model(3)
mixture_model(4)
```

Kernal METHODS

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

stations <- read.csv("stations.csv",fileEncoding = "Latin1")
temps <- read.csv("temps50k.csv")

st <- merge(stations,temps,by="station_number")
  h_distance <- 1000000 # These three values are up to the students
  h_date <- 100
  h_time <- 200

a <- 58.4274 # The point to predict (up to the students)
b <- 14.826

date <- "2013-11-04" # 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","24:00:00")

temp <- vector(length=length(times))
temp_Multi <- vector(length=length(times))

point_intreset <- c(a,b)
gaussion_distance <- function(dist)
{
  return(exp(-dist^2))
}
d_stations <- function(db_point, distance_POI)
{
  dist <- distHaversine(db_point, distance_POI)
  return(gaussion_distance(dist / h_distance))
}
d_date <- function(db_point, date_POI)
{
  date_diff <- as.numeric(difftime(db_point,date_POI,unit = "days"))
  date_diff = abs(date_diff)
  date_diff[date_diff > 182] = 365 - date_diff[date_diff > 182]
  return(gaussion_distance(date_diff / h_date))
}
d_hour <- function(db_point, hour_POI)
{
```

```
  hour_diff <- as.numeric(difftime(db_point,hour_POI,unit = "hours"))
  hour_diff = abs(hour_diff)
  hour_diff[hour_diff > 12] = 24 - hour_diff[hour_diff > 12]
  return(gaussion_distance(hour_diff / h_time))
}
kernal_sum <- function(Original_data, POI,index)
{
  Original_data = fix_time(Original_data)
  POI = fix_time(POI)

  data_dist  = Original_data[,c("longitude", "latitude")]
  obs_dist   = c(POI$longitude, POI$latitude)

  data_date = Original_data$date
  obs_date  = POI$date

  data_time = Original_data$time
  obs_time  = POI$time

  # Calcualte kernels Sum
  kernal_stations = d_stations(data_dist,obs_dist)
  kernal_date = d_date(data_date,obs_date)
  kernal_hour = d_hour(data_time,obs_time)

  dist = kernal_stations + kernal_date + kernal_hour

  Original_data$distance = dist
  Original_data$data_dist = data_dist
  Original_data$data_date = data_date
  Original_data$data_time = data_time

  selection = Original_data

  return(sum(selection$distance * selection$air_temperature) / sum(selection$distance))
}
fix_time = function(data){
  data$time = as.POSIXct(data$time,format="%H:%M:%S")
  data$date = sub('\\d{4}(?=-)', '2016', data$date, perl=TRUE)
  return(data)
}

n = length(times)
data = data.frame(date=rep(date,n/length(date)), time=rep(times,n/length(times)), longitude=rep(a,n), la

for(i in 1:nrow(data))
{
  temp[i] <- kernal_sum(st, data[i,],i)
}
print(temp)
plot(temp, type="o")
```

Neural Networks

```
library(neuralnet)
library("grDevices")

set.seed(1234567890)
Var <- runif(50, 0, 10)#sample 50 points uniformly at random in intervals 0,10
trva <- data.frame(Var, Sin=sin(Var)) # apply Sin function to each point
tr <- trva[1:25,] # Training 1 to 25
va <- trva[26:50,] # Validation 26 to 50

# Random initializaiton of the weights in the interval [-1, 1]
results = rep(0,10) # numeric class 10 elements with 0 values
winit <- runif(250,-1,1) # 250 points between -1 and 1

for(i in 1:10) {
    nn <- neuralnet(formula = Sin ~ Var, data=tr, hidden = 10,
                    threshold = i/1000 ,startweights = winit)
  result = compute(nn, va$Var)$net.result
  results[i] = mean((result - va$Sin)^2)
}
best = which.min(results) # Most appropiate value of threshold select is i = 4
nn <- neuralnet(formula = Sin ~ Var , data=trva, hidden = 10, threshold = best/1000, startweights = win
plot(nn)
plot(prediction(nn)$rep1, col="Black")  # predictions (black dots)
points(trva, col = "red") # data (red dots)
```

Back Propogation

```
# JMP
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)
w_j <- runif(10, -1, 1);b_j <- runif(10, -1, 1);w_k <- runif(10, -1, 1);b_k <- runif(1, -1, 1)
l_rate <- 1/nrow(tr)^2
n_ite = 5000
error <- rep(0, n_ite)
error_va <- rep(0, n_ite)
for(i in 1:n_ite) {   # SGD
  for(n in 1:nrow(tr)) {
    z_j <- tanh(w_j * tr[n,]$Var + b_j)
    y_k <- sum(w_k * z_j) + b_k
    error[i] <- error[i] + (y_k - tr[n,]$Sin)^2
  }
  for(n in 1:nrow(va)) {
    z_j <- tanh(w_j * va[n,]$Var + b_j)
    y_k <- sum(w_k * z_j) + b_k
    error_va[i] <- error_va[i] + (y_k - va[n,]$Sin)^2
```

```
  }
  cat("i: ", i, ", error: ", error[i]/2, ", error_va: ", error_va[i]/2, "\n")
  flush.console()
  for(n in 1:nrow(tr)) {
    # forward propagation
    z_j <- tanh(w_j * tr[n,]$Var + b_j)
    y_k <- sum(w_k * z_j) + b_k
    # backward propagation
    d_k <- y_k - tr[n,]$Sin
    d_j <- (1 - z_j^2) * w_k * d_k
    partial_w_k <- d_k * z_j
    partial_b_k <- d_k
    partial_w_j <- d_j * tr[n,]$Var
    partial_b_j <- d_j
    w_k <- w_k - l_rate * partial_w_k
    b_k <- b_k - l_rate * partial_b_k
    w_j <- w_j - l_rate * partial_w_j
    b_j <- b_j - l_rate * partial_b_j
  }
}
w_j
b_j
w_k
b_k
plot(error/2, ylim=c(0, 5))
points(error_va/2, col = "red")
# prediction on training data
pred <- matrix(nrow=nrow(tr), ncol=2)
for(n in 1:nrow(tr)) {
  z_j <- tanh(w_j * tr[n,]$Var + b_j)
  y_k <- sum(w_k * z_j) + b_k
  pred[n,] <- c(tr[n,]$Var, y_k)
}
plot(pred)
points(tr, col = "red")
# prediction on validation data
pred <- matrix(nrow=nrow(tr), ncol=2)
for(n in 1:nrow(va)) {
  z_j <- tanh(w_j * va[n,]$Var + b_j)
  y_k <- sum(w_k * z_j) + b_k
  pred[n,] <- c(va[n,]$Var, y_k)
}
plot(pred)
points(va, col = "red")
```

Exam PLS Regression Model Cross Validation ,Tree

```
library(tree)
library(ggplot2)
library(pls)
```

```r
data <- read.csv2("../data/glass.csv")

set.seed(12345)
n <- nrow(data)

train_size <- floor(n * 0.5)
validation_size <- floor(n * 0.25)
test_size <- n - train_size - validation_size

idx <- 1:n
train_idx <- sample(x=idx, size=train_size)
validation_idx <- sample(x=idx[-train_idx], size=validation_size)
test_idx <- idx[-c(train_idx, validation_idx)]

train <- data[train_idx,]
validation <- data[validation_idx,]
test <- data[test_idx,]

## 1
sizes <- 2:8
validation_errors <- rep(0, length(sizes))
train_errors <- rep(0, length(sizes))
fit <- tree(Al ~ ., data=train)

for (size in sizes) {
    fit_pruned <- prune.tree(fit, best=size)
    validation_errors[size-1] <- mean((predict(fit_pruned, newdata=validation) - validation$Al)^2)
    train_errors[size-1] <- mean((predict(fit_pruned, newdata=train) - train$Al)^2)
}

plot_data <- data.frame(x=sizes, y1=validation_errors, y2=train_errors)

ggplot() +
    xlab("# of terminal nodes") + ylab("Mean Squarred Error") +
geom_line(data=plot_data, aes(x=x, y=y2), color="blue") +
    geom_line(data=plot_data, aes(x=x, y=y1), color="red")

## 2
optimal_size <- which.min(validation_errors) + 1
optimal_tree <- prune.tree(fit, best=optimal_size)
test_error <- mean((predict(optimal_tree, newdata=test) - test$Al)^2)
test_error

plot(optimal_tree)
text(optimal_tree, pretty=TRUE)

## 3
set.seed(12345)
fit <- plsr(Al ~ ., data=train, validation="CV")

summary(fit)
fit$validation
fit$scores
```

```
fit$loadings

optimal_fit <- plsr(Al ~ ., data=train, ncomp=6)

## a) 3 variables
## b) 6 variables
## c) According to CV the model with 6 components is best
## d) Na Mg Si Ca Ba
## e) Y_score = z1 + z2 + z3 + z4 + z5 + z6
rowSums(optimal_fit$scores)

## f)
test_error <- mean((predict(optimal_fit, newdata=test) - test$Al)^2)
test_error
```

Exam LDA , kernel epanechnikov

```
library(ggplot2)
library(MASS)

data <- mtcars
data$shp <- scale(data$hp)
data$sqsec <- scale(data$qsec)
data$am <- as.factor(data$am)

ggplot() +
    geom_point(data=data, aes(x=shp, y=sqsec, color=am))

## No, the data is not linearly separable
prior <- c(1, 1) / 2
fit_eq <- lda(am ~ shp + sqsec, data=data, prior=prior)
fit_eq

prior <- as.numeric(table(data$am) / sum(table(data$am)))
fit_neq <- lda(am ~ shp + sqsec, data=data, prior=prior)
fit_neq

prediction_eq <- predict(fit_eq, data)$class
prediction_neq <- predict(fit_neq, data)$class

plot_data1 <- data.frame(x=data$shp, y=data$sqsec, color=prediction_eq, type="eq")
plot_data2 <- data.frame(x=data$shp, y=data$sqsec, color=prediction_neq, type="neq")

plot_data <- rbind(plot_data1, plot_data2)

ggplot() +
    geom_point(data=plot_data, aes(x=x, y=y, color=color)) +
    facet_grid(type ~ .)

euclidean <- function(u) {
    sqrt(sum(u^2))
```

```
}

kernel.epan <- function(u) {
    (1 - euclidean(u)^2) * as.numeric((euclidean(u) <= 1))
}

kernel.density <- function(X, Xtest, lambda) {
    apply(Xtest, 1, function(x){
        s <- 0

        for (i in 1:nrow(X)) {
            s <- s + kernel.epan((X[i, ] - x) / lambda)
        }

        s / nrow(X)
    })
}

lambda <- 0.2

idx1 <- which(data$am == 0)
X1 <- as.matrix(data.frame(data$qsec[idx1], data$hp[idx1]))
Xtest1 <- as.matrix(data.frame(data$qsec, data$hp))
density1 <- kernel.density(X1, Xtest1, lambda)

idx2 <- which(data$am == 1)
X2 <- data.frame(data$qsec[idx2], data$hp[idx2])
Xtest2 <- data.frame(data$qsec, data$hp)
density2 <- kernel.density(X2, Xtest2, lambda)

densities <- data.frame(density1, density2)
prediction <- apply(densities, 1, function(x) which.max(x))

prediction_error <- mean(as.numeric(data$am) != prediction)
prediction_error

plot_data <- data.frame(x=data$hp, y=data$qsec, color=as.factor(prediction - 1))

ggplot() +
    geom_point(data=plot_data, aes(x=x, y=y, color=color))
```

Exam Neuralnet

```
library(neuralnet)

data <- read.csv("../data/wine.csv")
data$class[which(data$class == 2)] <- -1

set.seed(12345)
train_idx <- sample(1:nrow(data), size=floor(nrow(data) * 0.7))
train <- data[train_idx,]
```

```
test <- data[-train_idx,]

## 3
set.seed(12345)
formula <- paste("class ~ ", paste(names(data)[-1], collapse=" + "))
fit <- neuralnet(formula=formula, data=train, hidden=0, act.fct="tanh", linear.output=FALSE)
plot(fit)

weights <- fit$weights[[1]][[1]][-1,]
weights
variables <- fit$model.list$variables[order(abs(weights), decreasing=TRUE)]
variables

## 4
train_error <- mean(sign(compute(fit, train[, -1])$net.result) != train$class)
train_error

test_error <- mean(sign(compute(fit, test[, -1])$net.result) != test$class)
test_error

## 5
set.seed(12345)
formula <- paste("class ~ ", paste(names(data)[-1], collapse=" + "))
fit <- neuralnet(formula=formula, data=train, hidden=1, act.fct="tanh", linear.output=TRUE)
plot(fit)

train_error <- mean(sign(compute(fit, train[, -1])$net.result) != train$class)
train_error

test_error <- mean(sign(compute(fit, test[, -1])$net.result) != test$class)
test_error

## 6
## 1. A tanh function
## 2. A translated tanh function
## 3. Parabola
```

RANDOM

Matrix formulation of OLS regression

Optimality condition:

$X^T (y - Xw) = 0$

Parameter estimates and predictions:

Least squares estimates of the parameters $W\_hat = (X^T X)^{-1} X^T y$

Predicted values $Y\_hat = X\ w\_hat = X(X^T X)^{-1} X^T y = Py$ Linear regression belongs to the class of linear smoothers

Overfitting: solutions

Observed: Maximum likelihood can lead to overfitting.

Solutions Selecting proper parameter values Regularized risk minimization Selecting proper model type, for ex. number of parameters Houldout method Cross-validation

Cross-validation vs Holdout

Holdout is easy to do (a few model fits to each data) Cross validation is computationally demanding (many model fits) Holdout is applicable for large data Otherwise, model selection performs poorly Cross validation is more suitable for smaller data

LDA versus Logistic regression

Generative classifiers are easier to fit, discriminative involve numeric optimization LDA and Logistic have same model form but are fit differently LDA has stronger assumptions than Logistic, some other generative classifiers lead also to logistic expression New class in the data? Logistic: fit model again LDA: estimate new parameters from the new data Logistic and LDA: complex data fits badly unless interactions are included

LDA versus Logistic regression

LDA (and other generative classifiers) handle missing data easier Standardization and generated inputs: Not a problem for Logistic May affect the performance of the LDA in a complex way Outliers affect ?? ??? LDA is not robust to gross outliers LDA is often a good classification method even if the assumption of normality and common covariance matrix are not satisfied.

Principal components analysis

Idea: Introduce a new coordinate system (PC1, PC2, .) where The first principal component (PC1) is the direction that maximizes the variance of the projected data The second principal component (PC2) is the direction that maximizes the variance of the projected data after the variation along PC1 has been removed The third principal component (PC3) is the direction that maximizes the variance of the projected data after the variation along PC1 and PC2 has been removed In the new coordinate system, coordinates corresponding to the last principal components are very small ??? can take away these columns

```
# 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 component loadings (U)
# U=res$rotation
# head(U)
######### Trace plots
# U=loadings(res)
```

```
# plot(U[,1], main="Traceplot, PC1")
# plot(U[,2],main="Traceplot, PC2")
```

7 Methods to Calculate Principal Components,Loadings, Proportion of Total Variance and Correlation

```
# data<-mlb11
# x<-as.data.frame(data[,-c(1,2)])
# standardize<-function(x){ #standardize
# for (i in 1:dim(x)[2])
# {
#   x[,i] = (x[,i] - mean(x[,i]))/sd(x[,i])
#   }
# return(x)
#   }
# X<-as.data.frame(standardize(x))
#  # x variables
#
# #method 1: princomp
# fit <- princomp(X)
# #plot(fit,type="lines") # scree plot
# PC1<-fit$scores
#  # the principal components#biplot(fit)#method 2: by hand
# Sx= var(X)
# EP= eigen(Sx)
# V= EP$vectors
# PC2= as.matrix(X) %*% as.matrix(V)
# #method 3: prcomp
# pca <- prcomp(X,center = TRUE,scale. = TRUE)
# PC3<-predict(pca)
# #biplot(pca)#method 4: preProcess
# require(caret)
# trans = preProcess(X, method=c("BoxCox", "center", "scale", "pca"))
# PC4 = predict(trans,X)
# #method 5: PCA
# library(FactoMineR)
# PC5 = PCA(X, graph = FALSE)
# #method 6: dudi.pca
# library(ade4)
# PC6= dudi.pca(X,nf=5,scannf = FALSE)
# # nf = 5, choosing 5 axises #method 7:
# library(amap)
# PC7 = acp(X)
#
# summary(fit) #PC1
# cumsum(EP$values)/sum(EP$values) #PC2
# summary(pca)#PC3
# trans$thresh;  trans$numComp #PC5
# PC5$eig #PC5
# cumsum(PC6$eig)/sum(PC6$eig) #PC6
# cumsum((PC7$sdev)^2)/sum((PC7$sdev)^2)#PC7
```

Advantages of probabilistic PCA

More settings to specify??? more flexible Can be faster when M<

Rnorm() ==> Runif()

boot(data, statistic, R, sim = "ordinary", ran.gen = function(d, p) d, mle = NULL,.)

Nonparametric bootstrap:

Write a function statistic that depends on dataframe and index and returns the 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

Parametric bootstrap:

Compute value mle that estimates model parameters from the data Write function ran.gen that depends on data and mle and which generates new data Write function statistic that depend on data which will be generated by ran.gen and should return the estimator

mle=lm(Price~Area, data=data2)

rng=function(data, mle) { data1=data.frame(Price=data$Price, Area = data$Area) n=length(data$Price) −− > generatenewPricedata1Price=rnorm(n,predict(mle, newdata=data1),sd(mle$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")

Bootstrap cofidence bands for linear model

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")

Example: parametric bootstrap

mle=lm(Price~Area, data=data2)

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) 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")

NSC: example

Package pamr 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.charac 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)

Computational shortcuts p>>n

SVD decomposition X= UDVˆT = RVˆT If model is linear in parameters and has quadratic penalties: Transform data observations from X into R Minimize loss (minus log likelihood) with R instead of X and get ???? Original parameters ????=???????? Can be applied to many methods

Example: ridge regression

Support Vector Machine

```
# svmfit= svm(y~., data=dat , kernel ="radial", cost =10, gamma =1)
# plot(svmfit , dat)
#
# set.seed(1)
# tune.out=tune(svm , y~., data=dat, kernel ="radial",
# ranges = list(cost=c(0.1 ,1 ,10 ,100 ,1000),gamma=c(0.5,1,2,3,4) ))
# tune.out$best.model
#
# svmfit=svm(y~., data=dat , kernel ="radial", cost =10, gamma =2)
# plot(svmfit , dat)
#
# dat=data.frame(x=Khan$xtrain , y=as.factor(Khan$ytrain))
# out=svm(y~., data=dat , kernel ="linear",cost =10)
#
# tune.out=tune(svm , y~., data=dat, kernel ="linear",
#               ranges =list(cost=c(0.1 ,1 ,10)))
# tune.out$best.model
```

Decision Trees

```
# library(tree)
# library(ISLR)
# attach(Carseats)
# High=ifelse(Sales<=8,"No","Yes")
# Carseats=data.frame(Carseats,High)
# tree.carseats=tree(High~.-Sales,Carseats)
# summary(tree.carseats)
#
# plot(tree.carseats)
```

```
#
# train=sample(1:nrow(Carseats),200)
# Carseats.test=Carseats[-train,]
# High.test=High[-train]
# tree.carseats = tree(High~.-Sales ,Carseats ,subset =train )
# tree.pred=predict(tree.carseats ,Carseats.test ,type ="class")
# table(tree.pred,High.test)
#
# #Tree Pruning
# cv.carseats =cv.tree(tree.carseats,FUN=prune.misclass)
# plot(cv.carseats$size ,cv.carseats$dev ,type="b") # Cross-Validation Error Rate
#
# plot(cv.carseats$k ,cv.carseats$dev ,type="b")  # alpha
#
# prune.carseats =prune.misclass(tree.carseats,best =9)
# plot(prune.carseats)
# text(prune.carseats,pretty =0)
#
# tree.pred=predict(prune.carseats,Carseats.test,type="class")
# table(tree.pred ,High.test)
```

Variable and Model Selection

```
# vif(creditm1)
# par( mfrow=c(2,2))
# plot(creditm1)
```

Arti???cial Neural Network

```
# x<-matrix(c(3,5,5,1,10,2),ncol=2,byrow=T)
# y<-matrix(c(75,82,93))
# colmax<- apply(x,2,max)
# X<-t(t(x)/colmax)  #scaling the data
# Y<-y/100input_layer_size <- 2output_layer_size <- 1hidden_layer_size <- 3
# set.seed(10)
# W_1 <- matrix(runif(6),nrow = input_layer_size,ncol = hidden_layer_size)
#
# sigmoid <- function(Z) 1/(1 + exp(-Z))
# W_2 <- matrix(runif(3),nrow = hidden_layer_size,ncol = output_layer_size)
# cost_hist <- rep(NA, 10000)
# cost<-function(y,y_hat) 0.5 * sum((y-y_hat)^2)
# sigmoidprime <- function(z) exp(-z) / ((1+exp(-z))^2)
# scalar <- 5
# for(i in 1:10000){
#   Z_2 <- X %*% W_1
#   A_2 <- sigmoid(Z_2)
#   Z_3 <- A_2 %*% W_2
#   Y_hat <-sigmoid(Z_3)
#   cost_hist[i] <- cost(Y,Y_hat)
#   delta_3 <- (-(Y-Y_hat) * sigmoidprime(Z_3))
#   djdw2 <- t(A_2) %*% delta_3
```

```
#   delta_2 <- delta_3 %*% t(W_2) * sigmoidprime(Z_2)
#   djdw1 <- t(X) %*% delta_2
#   W_1 <- W_1 - scalar * djdw1
#   W_2 <- W_2 - scalar * djdw2}
# W_1
```

k-Fold Cross-Validation (cv.glm with K=i)

```
# set.seed(17)
# cv.error.10= rep(0 ,10)
# for (i in 1:10){
#   glm.fit= glm(mpg~poly(horsepower ,i),data=Auto)
#   cv.error.10[i]=cv.glm(Auto ,glm.fit ,K=10) $delta [1]
#   }
# cv.error.10
```

Performing the Bootstrap

```
# library(boot)
# set.seed(1)
# alpha.fn(Portfolio ,sample(100 ,100 , replace =T))
# boot(Portfolio ,alpha.fn,R=1000)
```

Bootstrap on Linear Regression

```
# boot.fn=function (data ,index ){
# return(coef(lm(mpg~horsepower ,data=data ,subset =index)))
#   }
# boot.fn2=function (data ,index ){
# coefficients(lm(mpg~horsepower +I( horsepower ^2) ,data=data,subset =index))  }
# boot.fn(Auto ,sample(392 ,392 , replace =T))
# summary(lm(mpg~horsepower +I(horsepower^2) ,data=Auto))$coef
```

Logistic regression

Data $Y_i \in \{Spam, Not\ Spam\}, X_i = \#of\ a\ word$

Model: $p(Y = Spam|w, x) = \frac{1}{1+e^{-w_0-w_1X}}$

Fitting: maximum likelihood

Prediction : $p(spam) = p(Y = spam|x)$

K-nearest neighbor density estimation

Data: Fish length $X_1, \dots X_N$

Model $p(x|\Delta) = \frac{K}{N \cdot \Delta}$

K: #neighbors in training data

Δ: length of the interval containing K neighbors

Learning: Fix some K or find an appropriate K

Prediction: predict $p(x|K)$

K-nearest neighbor classification

Given N observations $(\boldsymbol{X}_j, Y_j)$

$Y_j = C_i$, where $C_1, \dots C_m$ are possible class values

Model assumptions

Apply K-NN density estimation:

$$p(X = x|Y = C_i) = \frac{K_i}{N_i V}, p(C_i) = \frac{N_i}{N}$$

V: volume of the sphere

K_i: #obs from training data of $Y = C_i$ in the sphere

N_i: #obs from training data of $Y = C_i$

Bayesian classification

Prediction $\hat{Y}(\boldsymbol{x}) = C_l$

$$l = \arg \max_{i \in \{1,\dots,m\}} p(C_i|\boldsymbol{x})$$

Bayes theorem

$$p(C_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_i)p(C_i)}{p(\boldsymbol{x})}$$

We get

$$p(C_i|x) \propto \frac{K_i}{K}$$

K-nearest neighbor classification

Algorithm

1. Given training set D,number K, and test set T
2. For each $x \in T$
   1. For each $i = 1, \dots M$
      1. $p'(C_i|x) = \frac{K_i}{K}$
   2. Compute $l = \arg \max_{i \in \{1,\dots,m\}} p'(C_i|\boldsymbol{x})$
   3. Predict $\hat{Y}(x) = C_l$

Majority voting: prediction for x is defined by majority voting of K neighbors

Black swan paradox

In the coin example, $p(x^{new} = 1) = \frac{k}{n}$ if MLE used

If we made 3 attempts, no successes→ k=0

Does this mean $p(x^{new} = 1) = 0$??

Problem does not appear in Bayesian setting (posterior mean)

Probabilistic models

A distribution $p(x|w)$ or $p(y|x, w)$

Example:

$$x \sim Bin(n, \theta)$$

$$p(x = k|n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k}$$

$$y \sim N(\alpha_0 + \alpha_1 x, \sigma^2)$$

Fitting a model

Bayesian principle

Compute $p(w|D)$and then decide yourself what to do with this (for ex. MAP, mean, median)

Use bayes theorem

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \propto p(D|w)p(w)$$

$p(D)$ is **marginal likelihood**

$p(D) = \int p(D|w)p(w)dw$ or

$p(D) = \sum_i p(D|w_i)p(w_i)$

Types of supervised models

Generative models: model $p(X|Y,w)$and $p(Y|w)$

Example: k-NN classification

$$p(X=x|Y=C_i,K)=\frac{K_i}{N_iV}, p(C_i|K)=\frac{N_i}{N}$$

From Bayes Theorem,

$$p(Y=C_i|x,K)=\frac{K_i}{K}$$

Discriminative models: model $p(Y|X,w)$, X constant

Example: logistic regression

$$p(Y=1|\boldsymbol{w},\boldsymbol{x})=\frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

Loss matrix

Costs of classifying $Y=C_k$ to C_j:

Rows: true, columns: predicted

$$L=\|L_{ij}\|, i=1,\dots,n, j=1,\dots,n$$

Example 1: 0/1-loss

$$L=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$$

Example 2: Spam

$$L=\begin{pmatrix}0 & 100\\ 1 & 0\end{pmatrix}$$

Loss and decision

Expected loss minimization

R_j : classify to C_j

$$EL=\sum_k\sum_j\int_{R_j}L_{kj}p(\boldsymbol{x},C_k)d\boldsymbol{x}$$

Choose such R_j that EL is minimized

Two classes $EL=\int_{R_1}L_{21}p(x,C_2)dx+\int_{R_2}L_{12}p(x,C_1)\,dx$

Loss and decision

- How to minimize *EL*?
  - We are free to assign x to either R_1 or R_2
  - Assigning x to region with smallest $L_{ij}p(x,C_i)$ will make *EL* smaller
- →Rule:
  - $L_{12}p(x,C_1)>L_{21}p(x,C_2)$ →predict y as C_1

$$\frac{p(C_1|x)}{p(C_2|x)}>\frac{L_{21}}{L_{12}}\rightarrow predict\ y\ as\ C_1$$

- 0/1 Loss: classify to the class which is more probable!

Loss and decision

- Continuous targets: squared loss
  - Given a model $p(x,y)$, minimize

$$EL=\int L\left(y,\hat{Y}(x)\right)p(x,y)dxdy$$

- Using **square loss**, the optimal is posterior mean

$$\hat{Y}(x)=\int yp(y|x)dy$$

ROC curves

- Binary classification
- The choice of the thershold $\hat{x} = \frac{L_{21}}{L_{12}}$ affects prediction→ what if we don't know the loss? Which classifier is better?
- **Confusion matrix**

		PREDICTED		
TRUE		1	0	Total
	1	TP	FN	N_+
	0	FP	TN	N_-

ROC curves

- **True Positive Rates** (TPR) = **sensitivity** = **recall**
  - Probability of detection of positives: TPR=1 positives are correctly detected

$$TPR = TP/N_+$$

- **False Positive Rates** (FPR)
  - Probability of false alarm: system alarms (1) when nothing happens (true=0)

$$FPR = FP/N_-$$

- **Specificity**

$$Specificity = 1 - FPR$$

- **Precision**

$$Precision = \frac{TP}{TP + FP}$$

Ridge regression

Problem: linear regression can overfit:
Take $Y := Y, X_1 = X, X_2 = X^2, \dots, X_p = X^p$→ polinomial model, fit by linear regression
High degree of polynomial leads to overfitting.

Equivalent form

$$\hat{w}^{ridge} = \text{argmin} \sum_{i=1}^{N} (y_i - w_0 - w_1 x_{1j} - \dots - w_p x_{pj})^2$$

$$\textbf{subject to} \sum_{j=1}^{p} w_j^2 \leq s$$

Solution

$$\hat{\boldsymbol{w}}^{\boldsymbol{ridge}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$$

$$\hat{y} = X\hat{w} = X(X^T X + \lambda I)^{-1} X^T y = Py$$

Properties of Ridge regression

- Extreme cases:
  - $\lambda = 0$ usual linear regression (no shrinkage)
  - $\lambda = +\infty$ fitting a constant ($w = 0$ except of w_0)
- When input variables are ortogonal (not realistic), $X^T X = I \rightarrow$

  $\widehat{w}^{\text{ridge}} = \frac{1}{1+\lambda} \mathbf{w}^{\mathbf{linreg}}$ → coefficients are equally shrunk
- **Ridge regression is particularly useful if the explanatory variables are strongly correlated to each other.**
  - Correlated variables often correspond large w→shrunk
- Degrees of freedom decrease when λ increases
  - $\lambda = 0 \rightarrow d.f. = p$
- Shrinking enables estimation of regression coefficients even if the number of parameters exceeds the number of cases! ($X^T X + \lambda I$ is always nonsingular)
  - Compare with linear regression
- How to estimate λ?
  - cross-validation

- Bayesian view
  - Ridge regression is just a special form of Bayesian Linear Regression with constant σ^2:

$$\boldsymbol{y} \sim N\left(\boldsymbol{y} \middle| \boldsymbol{w_o} + \boldsymbol{X}\boldsymbol{w}, \boldsymbol{\sigma^2 I}\right)$$

$$\boldsymbol{w} \sim \boldsymbol{N}\left(\mathbf{0}, \frac{\sigma^2}{\lambda} \boldsymbol{I}\right)$$

Theorem MAP estimate to the Bayesian Ridge is equal to solution in frequenist Ridge

$$\widehat{\boldsymbol{w}}^{\boldsymbol{ridge}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-\mathbf{1}} \boldsymbol{X}^{\boldsymbol{T}} \boldsymbol{y}$$

- In Bayesian version, we can also make inference about λ

R code: use package **glmnet** with alpha=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)
```

Choosing the best model by cross-validation:

```
model=cv.glmnet(as.matrix(covariates),
response, alpha=0,family="gaussian")
model$lambda.min
plot(model)
coef(model, s="lambda.min")
```

How good is this model in 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)
```

```
> sum((ynew-mean(y))^2)/sum((y-mean(y))^2)
[1] 0.5438148
> sum((ynew-y)^2)
[1] 18.04988
>
```

LASSO

- **Idea**: Similar idea to Ridge
- Minimize minus loglikelihood plus **linear** penalty factor→ **l_1 regularization**
  - Given that model is Gaussian, we get **LASSO** (least absolute shrinkage and selection operator):

$$\hat{w}^{lasso} = \text{argmin} \left\{ \sum_{i=1}^{N} (y_i - w_0 - w_1 x_{1j} - \ldots - w_p x_{pj})^2 + \lambda \sum_{j=1}^{p} |w_i| \right\}$$

- $\lambda > 0$ is **penalty factor**

LASSO Coding in R: use glmnet() with alpha=1

Variable selection

Alternative 1: Variable subset selection

- Best subset selection:
  - Consider different subsets of the full set of features, fit models and evaluate their quality
    - Problem: computationally difficult for p around 30 or more
    - How to choose the best model size? Some measure of predictive performance normally used (ex. AIC).
- Forward and Backward stepwise selection
  - Starts with 0 features (or full set) and then adds a feature (removes feature) that most improves the measure selected.
    - Can handle large *p* quickly
    - Does not examine all possible subsets (not the "best")

Use stepAIC() in MASS

```
library(MASS)
fit <- lm(V9~.,data=data.frame(data1))
step <- stepAIC(fit, direction="both")
step$anova
summary(step)
```

An estimator

- $\widehat{\boldsymbol{w}} = \delta(D)$ (some function of your data) – an **estimator**
- Optimal parameter values?→there can be many ways to compute them (MLE, shrinkage…)
  - Compare Bayesian: given estimators $\boldsymbol{w^1}$ and $\boldsymbol{w^2}$, we **can** compare them! $p(\boldsymbol{w^1}|\boldsymbol{D}) > \boldsymbol{p}(\boldsymbol{w^2}|\boldsymbol{D})$
  - There is no easy way to compare estimators in frequentist tradition

Example: Linear regression

- Estimator 1: $\boldsymbol{w} = (X^T X)^{-1} X^T Y$ (maximum likelihood)
- Estimator 2: $\boldsymbol{w} = (0, \dots, 0,1)$
- Which one is better?
  - A comparison strategy is needed!

Loss functions

- How to define loss function?
  - No unique choice, often defined by application
  - **Normal practice: Choose the loss related to minus loglikelihood**

Example: Predicting the amount of the product at the storage:

$$L(Y, \hat{y}) = \begin{cases} 10, +\hat{y}/Y \ \hat{y} \geq Y \\ 1000, \hat{y} < Y \end{cases}$$

Example: Compute loss function related to

- Normal distribution

Loss functions

Classification problems

- Common loss function $L(\mathrm{Y}, \hat{y}) = \begin{cases} 1, Y = \hat{y} \\ 0, Y \neq \hat{y} \end{cases}$
- When minimizing the loss, equivalent to misclassification rate

Model selection

- **Problem**: true model & true $\boldsymbol{w}$ unknown→can not compute expected loss!
- How to find an optimal model?
  - Consider what expected loss (**risk**) depends on $R(Y, \hat{y}) = E[L(Y, \hat{y}(X, D))]$
- Random factors:
  - D – **training set**
  - Y, X – data to be predicted **(validation set)**

Holdout method

- Simplify the risk estimation:
  - Fix D as a particular training set T
  - Fix Y,X as a particular validation set V
- Risk becomes (**empirical risk**)

$$\hat{R}(y, \hat{y}) = \frac{1}{|V|} \sum_{(X,Y) \in V} L(Y, \hat{y}(X, T))$$

  - Estimator is fit by Maximum Likelihood using training set
  - Risk estimated by using validation set
  - Model with minimum empirical risk is selected

Cross-validation

Compared to holdout method:

- Why do we use only some portion of data for training- can we use more (increase accuracy)?

Cross-validation (Estimates Err)

K-fold cross-validation (rough scheme, show picture):

1. Permute the observations randomly
2. Divide data-set in K roughly equally-sized subsets
3. Remove subset #i and fit the model using remaining data.
4. Predict the function values for subset #i using the fitted model.
5. Repeat steps 3-4 for different i
6. CV= squared difference between observed values and predicted values (another function is possible)

Note: if K=N then method is *leave-one-out* cross-validation.

K-fold cross-validation: $CV = \frac{1}{N} \sum_{i=1}^{N} L(Y_i, \hat{y}^{-k(i)}(x_i))$

Analytical methods

- Analytical expressions to select models
  - AIC (Akaike's information criterion)

Idea: Instead of $R(Y,\hat{y}) = E[L(Y,\hat{y}(X,D))\,]$ consider **in-sample** risk (only *Y* in *D* is random):

$$R_{in}(Y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N} E_{Y_i}\left[L\left(Y_i,\hat{y}(X,D)\right)|D, X \in D\right]$$

- One can show that

$$R_{in}(Y,\hat{y}) \approx R_{train} + \frac{2}{N}\sum_{i} cov(\hat{y}_i, Y_i)$$

  where $R_{train} = \sum_{X_i,Y_i \in T} L(Y_i,\hat{y_i})$

- Recall, **degrees of freedom** $\mathrm{df(model)} = \frac{1}{\sigma^2}\sum_i cov(\hat{y}_i, Y_i)$
  - When model is linear, *df* is the number of parameters.
- If loss is defined by minus two loglikelihood,

$$AIC \equiv -2loglik(D) + 2df(model)$$

Logistic regression

- Discriminative model
- Model for binary output
  - $C = \{C_1 = 1, C_2 = 0\}$

$$p(Y = C_1|X) = sigm(\boldsymbol{w}^T\boldsymbol{x})$$

$$sigm(a) = \frac{1}{1+e^{-a}}$$

- Alternatively

$$Y \sim Bernoulli(sigm(a)), a = \boldsymbol{w^T x}$$

$$sigm(a) = \frac{1}{1+e^{-a}}$$

- When Y is categorical,

$$p(Y = C_i|x) = \frac{e^{\boldsymbol{w_i^T x}}}{\sum_{j=1}^{K} e^{\boldsymbol{w_j^T x}}} = softmax(\boldsymbol{w_i^T x})$$

- Alternatively

$$Y \sim Multinoulli\left(softmax(\boldsymbol{w}_1^T\boldsymbol{x}), \ldots softmax(\boldsymbol{w}_K^T\boldsymbol{x})\right)$$

Logistic model- yet another form

$$ln\frac{p(Y=1|X=x)}{P(Y=0|X=x)} = ln\frac{p(Y=1|X=x)}{1-P(Y=1|X=x)}$$
$$= logit(p(Y=1|X=x)) = \boldsymbol{w^T x}$$

Here $logit(t) = \ln\left(\frac{t}{1-t}\right)$

Note $p(Y|X)$ is connected to $\boldsymbol{w^T x}$ via logit link

Fitting logistic regression

- In binary case,

$$\log P(D|w) = \sum_{i=1}^{N} y_i \log(\, sigm(w^T x_i)) \; + (1 - y_i) \log(1 - sigm(w^T x_i))$$

  - Can not be maximized analytically, but unique maximizer exists
- To maximize loglikelihood, optimization used
  - Newton's method traditionally used (Iterative Reweighted Least Squares)
  - Steepest descent,Quasi-newton methods…

Estimation:

For new x, estimate $p(y) = [p_1, \dots p_C]$ and classify as $\arg \max_i p_i$

Decision boundaries of logistic regression are linear

- use glm() with family="binomial"
  - Predicted probabilities: predict(fit,newdata, type="response")

Exponential family

More advanced error distributions are sometimes needed!

Many distributions belong to **exponential** family:

Normal, Exponential, Gamma, Beta, Chi-squared..

Bernoulli, Multinoulli, Poisson…

$$p(\boldsymbol{x}|\boldsymbol{\eta}) = h(\boldsymbol{x}) g(\boldsymbol{\eta}) e^{(\boldsymbol{\eta}^T u(\boldsymbol{x}))}$$

Easy to find MLE and MAP

Non-exponential family distributions: uniform, Student t

Generalized linear models

Canonical links are normally used

MLE computations simplify

MLE $\widehat{w} = F(X^T Y)$ → computations do not depend on all data but rather a summary (sufficient statistics)→ computations speed up **glm**(formula, family, data)

Generalized linear models

Assume Y from the exponential family

Model is $Y \sim EF(\mu, \dots),\ f(\mu) = \boldsymbol{w^T x}$

Alt $\mu = f^{-1}(\boldsymbol{w^T x})$

f^{-1} is activation function

f is link function (in principle, arbitrary)

Arbitrary f will lead to (s – dispersion parameter)

$$p(y|w,s) = h(y,s) g(\boldsymbol{w}, \boldsymbol{x}) e^{\frac{b(\boldsymbol{w},\boldsymbol{x})y}{s}}$$

If f is a canonical link, then

$$p(y|w,s) = h(y,s) g(\boldsymbol{w}, \boldsymbol{x}) e^{\frac{(\boldsymbol{w}^T \boldsymbol{x})y}{s}}$$

Generalized linear model

$$High \sim Gamma(1, \frac{1}{w_0 + w_1 Open})$$

```
Call:
glm(formula = high ~ open, family = Gamma(link = "inverse"),
    data = data)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-0.0052879  -0.0028896  -0.0006678   0.0016598   0.0148083

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.355e-02  1.962e-04   69.06   <2e-16 ***
open        -4.604e-05  1.379e-06  -33.39   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Quadratic discriminant analysis

Generative classifier

Main assumptions:

$\boldsymbol{x}$ is now **random** as well as y

$$p(\boldsymbol{x}|y = C_i, \theta) = N(\boldsymbol{x}|\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$$

Unknown parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}\}$

If parameters are estimated, classify:

$$\hat{y}(\boldsymbol{x}) = \arg\max_c p(y = c|\boldsymbol{x}, \theta)$$

Linear discriminant analysis (LDA)

Assumtion $\Sigma_i = \Sigma, i = 1, \dots K$

Then $p(y = c_i|x) = softmax(w_i^T x + w_{0i})$ →exactly the same form as the logistic regression

$$w_{0i} = -\frac{1}{2}\boldsymbol{\mu}_i^T \Sigma^{-1} \boldsymbol{\mu}_i + \log \pi_i$$

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

$^{\cdot 1}\boldsymbol{\mu}_i$

Decisio

Discriminant function:

Linear discriminant analysis (LDA)

- Difference LDA vs logistic regression??
  - Coefficients will be estimated differently! (models are different)
- How to estimate coefficients
  - find MLE.

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{i:y_i=c} \mathbf{x}_i, \quad \hat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c} \sum_{i:y_i=c} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)^T$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{c=1}^{k} N_c \hat{\Sigma}_c$$

  - Sample mean and sample covariance are MLE!
  - If class priors are parameters (**proportional priors**),

$$\hat{\pi}_c = \frac{N_c}{N}$$

LDA and QDA: code library **MASS**

lda(formula, data, ..., subset, na.action)

- Prior – class probabiliies
- Subset – indices, if training data should be used

qda(formula, data, ..., subset, na.action)

predict(..)

Naive Bayes classifiers - discrete inputs

- Given $D = \{(X_{m1}, \dots X_{mp}, Y_m),\ m = 1, \dots n\}$
- Assume $X_i \in \{x_1, \dots x_J\}, i = 1, \dots p, Y \in \{y_1 \dots y_K\}$
- Denote $\theta_{ijk} = p(X_i = x_j | Y = y_k)$
  - How many parameters?
- Denote $\pi_k = p(Y = y_k)$

- Maximum likelihood: assume θ_{ijk} and π_k are constants
  - $\hat{\theta}_{ijk} = \frac{\#\{X_i = x_j \& Y = y_k\}}{\#\{Y = y_k\}}$
  - $\hat{\pi}_k = \frac{\#\{Y = y_k\}}{n}$
  - Classification using 0-1 loss: $\hat{Y} = \arg\max_y p(Y = y | X)$

Decision trees

- A tree $T = < r_i, s_{r_i}, R_j, i = 1 \dots S, j = 1 \dots L >$
  - $x_{r_i} \le s_{r_i}$ splitting rules (conditions), S- their amount
  - R_j-terminal nodes, L- their amount
  - labels μ_j in each terminal node

Model:

- $Y|T$ for R_j comes from exponential family with mean μ_j
- Fitting by MLE:
  - Step 1: Finding optimal tree
  - Step 2: Finding optimal labels in terminal nodes

Classification trees

- Impurity measure Q(
  - R_m is a tree node (reg
  - Node can be split unl

  Misclassification er

  Gini index:

  Cross-entropy or d

- Note: In many source

Example: Cross –entropy is M

Fitting regression trees: CART

Step 1: Finding optimal tree: grow the tree in order to minimize global objective

1. Let C_0 be a hypercube containing all observations
2. Let queue C={C_0}
3. Pick up some C_i from C and find a variable X_j and value s that split C_j into two hypercubes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\}$$

and solve

$$\min_{j,s}[N_1 Q(R_1) + N_2 Q(R_2)]$$

4. Remove C_j from C and add R_1 and R_2
5. Repeat 3-4 as many times as needed (or until each cube has only 1 observation)

Probabilistic PCA

z_i-latent variables, x_i- observed variables

$$\boldsymbol{z} \sim \boldsymbol{N}(0, \boldsymbol{I})$$
$$\boldsymbol{x}|\boldsymbol{z} \sim \boldsymbol{N}(\boldsymbol{x}|\boldsymbol{Wz} + \boldsymbol{\mu}, \sigma^2\boldsymbol{I})$$

Alternatively

$$\boldsymbol{z} \sim \boldsymbol{N}(0, \boldsymbol{I}), \boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{Wz} + \boldsymbol{\epsilon}, \boldsymbol{\epsilon} \sim \boldsymbol{N}(\boldsymbol{0}, \boldsymbol{\sigma^2 I})$$

Interpretation: Observed data (X) is obtained by rotation, scaling and translation of standard normal distribution (Z) and adding some noise.

Aim: extract Z from X

Distribution of x:

$$\boldsymbol{x} \sim \boldsymbol{N}(\boldsymbol{\mu}, \boldsymbol{C})$$
$$\boldsymbol{C} = \boldsymbol{WW^T} + \boldsymbol{\sigma^2 I}$$

Rotation invariance

Assume that x was generated from $z' = Rz, RR^T = I$, $p(x)$ does not change!

$$\boldsymbol{x}|\boldsymbol{z'} \sim \boldsymbol{N}(\boldsymbol{x}|\boldsymbol{Wz'} + \boldsymbol{\mu}, \sigma^2\boldsymbol{I})$$

Model will not be able find latent factors uniquely!

It does not distinguish z from z'

Estimation of parameters: ML

Theorem. ML estimates are given by

$$\mu_{ML} = \bar{x}$$
$$W_{ML} = U_M(L_M - \sigma_{ML}^2 I)^{\frac{1}{2}}R$$
$$\sigma_{ML}^2 = \frac{1}{p-M}\sum_{i=M+1}^{p}\lambda_i$$

U_M matrix of M eigenvectors
L_M diagonal matrix of M eigenvalues
R any orthogonal matrix

Estimation of Z

Use mean of posterior

$$\hat{z} = (W_{ML}^T W_{ML} + \sigma_{ML}^2 I)^{-1} W_{ML}^T (x - \mu)$$

Connection to standard PCA

Assume $R = I, \sigma^2 = 0$→get standard PCA components scaled by inverse root of eigenvalues

$$Z = XUL^{-\frac{1}{2}}$$

Independent component analysis (ICA)

- Probabilistic PCA does not capture latent factors
  - Rotation invariance
- Let's choose distribution which is not rotation invariant→will get unique latent factors
- Choose non-Gaussian $p_i(z) = p(z)$
- Assuming latent features are **independent**

$$p(z) = \prod_{i=1}^{M} p_i(z_i)$$

- Model

$$x = \mu + Wz + \epsilon, \qquad \epsilon \sim \boldsymbol{N}(0, \Sigma)$$

- **Estimation A: Maximum likelihood** ($V = W^{-1}$)
  - Assuming noise-free x

$$\max_V \sum_{i=1}^{n} \sum_{j=1}^{p} \log\left(p_j(v_j^T x_i)\right)$$

Subject to $||v_i|| = 1$

- Setting $G_j(z) = -\log\left(p_j(z)\right), z_j = v_i^T \boldsymbol{x}$ and assuming large sample

$$\min_V \sum_{j=1}^{p} E\left(G_j(z_j)\right)$$

Subject to $||v_i|| = 1$

- **Prewhitening**
  - Use PCA: $X' = XU$
  - Computing z_is for given V: $Z = X'V$
- **Estimation B**: maximize negentropy
  - ICA looks for model which is as much non-Gaussian as possible
- **Entropy** $H(z) = -\int p(z) \log p(z)\, dz = E\left(-\log p\,(z)\right)$
- **Negentropy** $J(z_i) = H(z_i') - H(z_i)$
  - $z_i' \sim N(Ez_i, var(z_i))$
- **Negentropy maximization**

$$\max_V \sum_{j=1}^{p} J(z_j) = \min_V \sum_{j=1}^{p} H(z_j) = \min_V \sum_{j=1}^{p} E\left(-\log p\,(z_j)\right)$$

Natural cubic spline

- A cubic spline f is called **natural cubic spline** if its 2nd and 3rd derivatives are zero at a and b

Note that f is linear on extreme intervals

Basis functions of natural cubic splines

$$N_1(X)=1,\ N_2(X)=X,\ N_{k+2}=d_k(X)-d_{K-1}(X),\quad k=1,\ldots,K-2$$

$$\text{where } d_k(X)=\frac{(X-\xi_k)_+^3-(X-\xi_K)_+^3}{\xi_K-\xi_k}$$

Fitting smooth functions to data

Minimize $$RSS(f,\lambda)=\sum_{i=1}^{N}(y_i-f(x_i))^2+\lambda\int\{f''(t)\}^2dt$$

where λ is **smoothing parameter**.

$\lambda=0$: any function interpolating data

$\lambda=+\infty$: least squares line fit

Optimality of smoothing splines

The function f minimizing RSS for a given λ is a natural cubic spline with knots at all unique values of x_i (NOTE: N knots!)

Minimizing sum of squares -->

$$f(x)=\sum_{j=1}^{N}N_j(x)\theta_j=N(x)^T\Theta$$

$$RSS(\Theta,\lambda)=(\mathbf{y}-\mathbf{N}\Theta)^T(\mathbf{y}-\mathbf{N}\Theta)+\lambda\Theta^T\Omega_N\Theta$$

$$\{\mathbf{N}\}_{ij}=N_j(x_i)\quad\{\Omega_N\}_{ij}=\int N_i''(t)N_j''(t)dt$$

$$\hat{\Theta}=(\mathbf{N}^T\mathbf{N}+\lambda\Omega_N)^{-1}\mathbf{N}^T\mathbf{y}$$

A smoothing spline is a linear smoother

Smoothing spline is a **linear smoother**. $\hat{f}=\mathbf{N}(\mathbf{N}^T\mathbf{N}+\lambda\Omega_N)^{-1}\mathbf{N}^T\mathbf{y}=\mathbf{S}_\lambda\mathbf{y}$

Compare with other smoothers, such as linear regression.

Smoothing splines and shrinkage

$$\mathbf{S}_\lambda\mathbf{y}=\sum_{k=1}^{N}\mathbf{u}_k\rho_k(\lambda)\mathbf{u}_k^T\mathbf{y}$$

Degrees of freedom

It can be shown that $\mathbf{S}_\lambda=(\mathbf{I}+\lambda\mathbf{K})^{-1}$

where $\mathbf{K}$ is **penalty matrix**

Eigenvalue decomposition of $\mathbf{K}$:

$$\mathbf{S}_\lambda=\sum_{k=1}^{N}\rho_k(\lambda)\mathbf{u}_k\mathbf{u}_k^T$$

$$\rho_k(\lambda)=\frac{1}{1+\lambda d_k}$$

d_k and $\mathbf{u}_k$ are eigenvalues and eigenvectors

Penalty and degrees of freedom

$df_\lambda = trace(\mathbf{S}_\lambda)$ →

$$df_\lambda = \sum_{k=1}^{N} \frac{1}{1+\lambda d_k}$$

λ increase → df_λ decrease

higher λ → higher penalization.

Smoother matrix is has banded nature →
local fitting method

Splines

Smoothing splines : smooth.spline()
Natural clubic splines: ns() in **splines**
Thin plate splines: Tps() in **fields**

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

Automated selection of smoothing parameters

$$df_\lambda = trace(\mathbf{S}_\lambda) = \sum_{k=1}^{N} \frac{1}{1+\lambda d_k}$$

Use either df_λ or λ
Given df_λ →solve equation→find λ

Use holdout principle or cross validation for parameter tuning

Generalized additive models

- Model

$$Y \sim EF(\mu, \dots)$$

where

- $g(\mu) = \alpha + s_1(X_1) + s_2(X_2) + s_p(X_p)$
- $s_i(X)$ - smoothers, normally splines
- EF – distribution from exponential family
- g – Link function

- Often linear terms are often included separately

$$EY = \alpha + s_1(X_1) + \dots + s_p(X_p) + \sum_{j=1}^{q} \beta_j X_{p+j}$$

Example: EF= normal, EF=Bernoulli (logistic)

Sometimes even higher orders are included (thin-plate splines)

$$g(\mu) = \alpha + s_1(X_1) + \dots + s_p(X_p) + \sum_{j=1}^{q} \beta_j X_{p+j} + s_{12}(X_1, X_2)$$

Method is reasonable to apply when additivity is observed or admissble

Estimation by MLE $g(\mu) = \alpha + f_1(x_1) + \ldots + f_p(x_p)$ Generalized additive models

The backfitting algorithm for Normal model

1.Initiali ze : $\hat{\alpha} = \frac{1}{N}\sum_{i=1}^{N} y_i, \quad \hat{f}_j \equiv 0,\ j = 1,\ldots,p$

2.Cycle : $j = 1,\ldots,p,1,\ldots,p,\ldots,1,\ldots,p$

$$\hat{f}_j \leftarrow s_j\left[\left\{(y_i - \hat{\alpha} - \sum_{k\neq j}\hat{f}_k(x_{ik})\right\}\right]$$

$$\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N}\sum_{i=1}^{N}\hat{f}_j(x_{ij})$$

R: package **mgcv** (also package **gam**)

gam(formula, family,data,select, method)

Select allows for term (variable) selection

predict(), plot(), summary()...

s(k, sp)

k should be the same as the amount of **unique values** of this variable in **smoothing splines**

sp - smoothing penalty.

```
river=read.csv2("Rhine.csv")
res=gam(TotN_conc~Year+Month+s(Year)
          +s(Month), data=river)
library(rgl)
library(akima)
s=interp(river$Year,river$Month,
fitted(res))
persp3d(s$x, s$y, s$z, col="red")
```

Classification: LDA

- Standard LDA $\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k$

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c}\sum_{i:y_i=c}\mathbf{x}_i, \quad \hat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c}\sum_{i:y_i=c}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)^T$$

$$\hat{\Sigma} = \frac{1}{N}\sum_{c=1}^{k} N_c\,\hat{\Sigma}_c$$

- → Σ^{-1} does not exist...

Classification: NSC **Nearest Shrunken Centroids**

- Idea: Shrink classwise means towards overall mean

1. Compute $d_{kj} = \frac{\bar{x}_{kj} - \bar{x}_j}{m_k(s_j + s_0)}$
2. Shrink $d'_{kj} = sign(d_{kj})(|d_{kj}| - \Delta)_+$
3. Set $x'_{kj} = \bar{x}_j + m_k(s_j + s_0)d'_{kj}$

Regularized discriminant analysis RDA

Another way of solving singularity of Σ

γ is some constant

$$\hat{\Sigma}(\gamma) = \gamma\hat{\Sigma} + (1-\gamma)diag(\hat{\Sigma})$$

$\gamma = 0$→diagonal-covariance LDA

γ is chosen by CV

R: rda() in **klaR**

Regularized logistic regression

Usual logistic regression

$$p(Y = C_i|x) = \frac{e^{w_{i0}+\boldsymbol{w}_i^T\boldsymbol{x}}}{\sum_{j=1}^{K} e^{w_{j0}+\boldsymbol{w}_j^T\boldsymbol{x}}} = softmax(w_{i0} + \boldsymbol{w}_i^T\boldsymbol{x})$$

Lp -Regularization:

$$\max_{w} \sum_{i=1}^{n} \log p(Y_i|x_i) - \frac{\lambda}{2}\sum_{k=1}^{K} \|\boldsymbol{w}_i\|^p$$

Parameter redunancy is solved

L1 regularization: some w are shrunk to 0

Numerical optimization is used to solve

R: LiblineaR() in package **LiblineaR**

Elastic net

L1 regularization

$$\min_{w} -\log p(D|\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_1$$

$$\|w\|_1 = \sum_i |w_i|$$

For p>n, LASSO can extract at most n nonzero components

Severe regularization if $p \gg n$

L1 regularization→selects some feature among the correlated ones

L2 regularization→ w's of the correlated variables are shrunk towards each other are nonzero

Elastic net

Combine L1 and L2 to diminish effect of L1 regularization.

Elastic net regularization:

$$\min_{w} -\log p(D|\boldsymbol{w}) + \lambda(\alpha\|\boldsymbol{w}\|_1 + (1-\alpha)\|w\|_2)$$

α is set ad hoc or chosen by CV

Elastic net may select more than n features

R: glmnet() in **glmnet** package

Specify "family" for classification or regression

Kernel PCA

Usual PCA

Center $\boldsymbol{X}$

Find $\boldsymbol{S}\boldsymbol{u}_i = \lambda_i\boldsymbol{u}_i, \boldsymbol{S} = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X}, \boldsymbol{S} = [p \times p]$

$\boldsymbol{u}_i$ has dimension p

Project data on PCs: $Z = X\,U$

Problems: X is unknown, and it can be p can be very large

Kernel PCA: Equivalent formulation

1. Solve $\boldsymbol{K}'\boldsymbol{a}_i = \lambda_i'\boldsymbol{a}_i, i = 1,..M$

   $\boldsymbol{K} = \|K(\boldsymbol{x}_i, \boldsymbol{x}_j), i, j = 1,..n\|$

   Centering $\boldsymbol{K}' = \boldsymbol{K} - \boldsymbol{1_n K} - \boldsymbol{K 1_n} + \boldsymbol{1_n K 1_n}$

   $\lambda_i = \lambda_i'/n$
2. Scores for PC_i: $z_i(\boldsymbol{x}) = \sum_{i=1}^{n} a_{in}K(\boldsymbol{x}, \boldsymbol{x}_n)$

There are at most n eigenvectors even if $p>>n$

```
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

Hypothesis testing Voice Rehabilitation

Feature "MFCC_2nd.coef"

```
res=t.test(MFCC_2nd.coef~Quality,data=data,
alternative="two.sided")
res$p.value
```

```
res=oneway_test(MFCC_2nd.coef~as.factor(Quality),
data=data,paired=FALSE)
pvalue(res)
```

Benjamini-Hochberg method (BH method)

Shown that $FDR(BH) < \alpha$ for independent hypotheses

→we can control FDR!

Algorithm 18.2 *Benjamini–Hochberg (BH) Method.*

1. Fix the false discovery rate α and let $p_{(1)} \leq p_{(2)} \leq \cdots \leq p_{(M)}$ denote the ordered p-values
2. Define
$$L = \max\left\{j : p_{(j)} < \alpha \cdot \frac{j}{M}\right\}.$$
3. Reject all hypotheses H_{0j} for which $p_j \leq p_{(L)}$, the BH rejection threshold.

Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

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

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

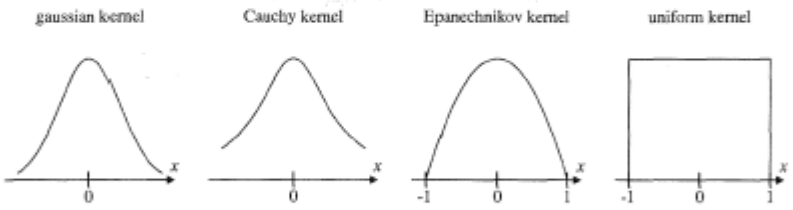


FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

- Gaussian kernel: $k(u) = exp(-||u||^2)$ where $||\cdot||$ is the Euclidean norm.
- Cauchy kernel: $k(u) = 1/(1 + ||u||^{D+1})$
- Epanechnikov kernel: $k(u) = (1 - ||u||^2)\mathbf{1}_{\{||u||\leq 1\}}$
- Moving window kernel: $k(u) = \mathbf{1}_{\{u \in S(0,1)\}}$

Histogram, Moving Window, and Kernel Regression

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

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

or

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

or

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

Histogram, Moving Window, and Kernel Density Estimation

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

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

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

$$|\{x_n \in R\}| \simeq P \, N$$

and thus

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

- Then,

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

or

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

or

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

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

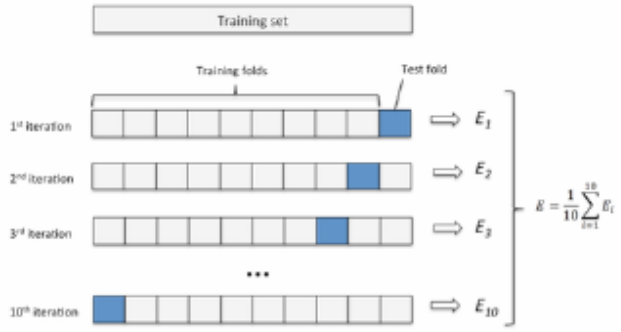
Kernel Selection

- Recall the fol
- Cross-validati
- If the training the prediction
- Note that the overestimates
- This seems to typically impl N/K test poi
- Typically, $K =$

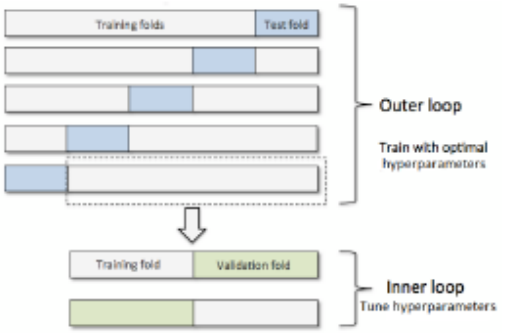
Kernel Selection

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

Cross-validation for estimating model prediction error



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

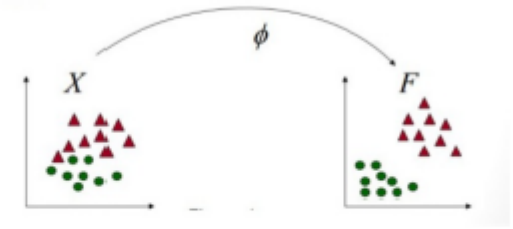


- Error overestimation may not be a concern for model selection. So, $K = 2$ may suffice in the inner loop.
- Which is the fitted model returned by nested cross-validation ?

15/19

Kernel Trick

- The kernel function $k\left(\frac{x-x'}{h}\right)$ is invariant to translations, and it can be generalized as $k(x,x')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x},\boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^M$
  - Gaussian kernel: $k(\boldsymbol{x},\boldsymbol{x}') = exp(-\|\boldsymbol{x}-\boldsymbol{x}'\|^2/2\sigma^2)$
- If the matrix
$$\begin{pmatrix} k(\boldsymbol{x}_1,x_1) & \dots & k(\boldsymbol{x}_1,x_N) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_N,\boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N,\boldsymbol{x}_N) \end{pmatrix}$$
is symmetric and positive semi-definite for all choices of $\{\boldsymbol{x}_n\}$, then $k(\boldsymbol{x},\boldsymbol{x}') = \phi(x)^T\phi(\boldsymbol{x}')$ where $\phi(\cdot)$ is a mapping from the input space to the feature space.



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

Kernel Trick

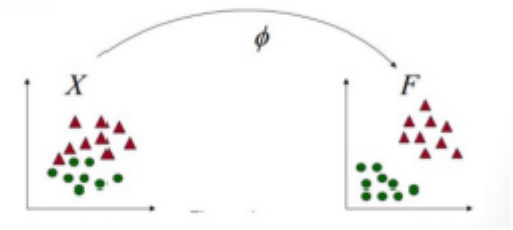
- Consider again moving window classification, regression, and density estimation.
- Note that $\boldsymbol{x}_n \in S(x, h)$ if and only if $||x - x_n|| \leq h$.
- Note that

$$||\boldsymbol{x} - \boldsymbol{x}_n|| = (\boldsymbol{x} - \boldsymbol{x}_n)^T (\boldsymbol{x} - \boldsymbol{x}_n) = \boldsymbol{x}^T \boldsymbol{x} + \boldsymbol{x}_n^T x_n - 2\boldsymbol{x}^T x_n$$

- Then,

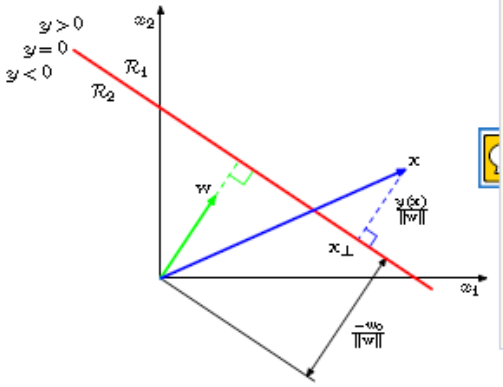
$$\begin{aligned} ||\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_n)|| &= \phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}) + \phi(\boldsymbol{x}_n^T)\phi(\boldsymbol{x}_n) - 2\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}_n) \\ &= k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{x}_n, x_n) - 2k(\boldsymbol{x}, \boldsymbol{x}_n)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

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



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

Support Vector Machines for Classification



Fahad Hameed

functions
svm
tune

26-Dec-2017 2:39 AM

Fahad Hameed

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

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

- Then, the maximum margin separating hyperplane is given by

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

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

Support Vector Machines for Classification

- Then, the maximum margin separating hyperplane is given by

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

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

- To minimize the previous expression, we minimize

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

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

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

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

$$0 = \sum_n a_n t_n$$

Support Vector Machines for Classification

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

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

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

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

Support Vector Machines for Classification

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

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

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

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

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

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

Support Vector Machines for Classification

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

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

and multiplying both sides by t_n, we have that

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

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

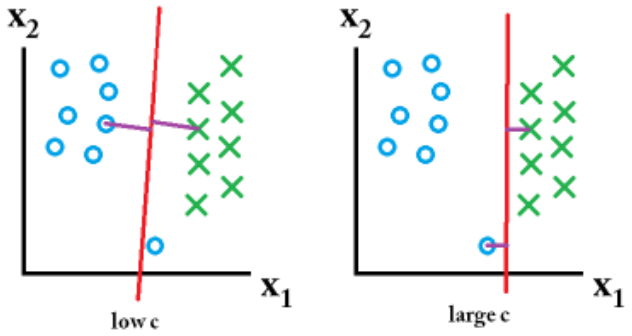
$$\xi_n = \begin{cases} 0 & \text{if } t_n y(\boldsymbol{x}_n) \geq 1 \\ |t_n - y(x_n)| & \text{otherwise} \end{cases}$$

Support Vector Machines for Classification

- The optimal separating hyperplane is given by

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

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



- To minimize the previous expression, we minimize

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

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

Support Vector Machines for Classification

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

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

$$0 = \sum_n a_n t_n$$

$$a_n = C - \mu_n$$

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

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

subject to $a_n \geq 0$ and $a_n \leq C$ for all n, because $\mu_n \geq 0$.

- When the Lagrangian function is maximized, the Karush-Kuhn-Tucker conditions hold for all n:

$$a_n(t_n y(\boldsymbol{x}_n) - 1 + \xi_n) = 0$$

$$\mu_n \xi_n = 0$$

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

Neural Networks

- Consider binary classification with input space $\mathbb{R}^D$. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$.
- SVMs classify a new point $\boldsymbol{x}$ according to

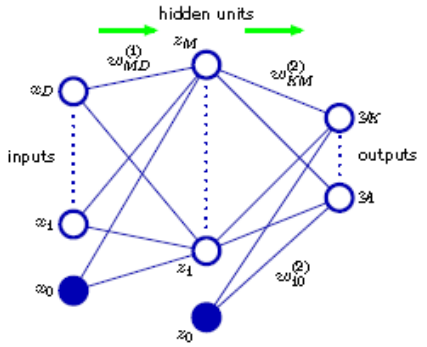
$$y(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}, \boldsymbol{x}_m) + b\right)$$

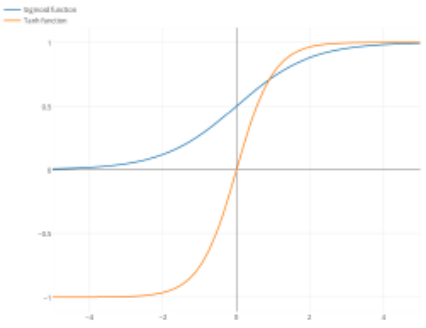
- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$
- For a new point x, SVMs predict

$$y(\boldsymbol{x}) = \sum_{m \in \mathcal{S}} (a_n - \widehat{a}_n) k(\boldsymbol{x}, \boldsymbol{x}_m) + b$$

- SVMs imply data-selected user-defined basis functions.
- NNs imply a user-defined number of data-selected basis functions.

Neural Networks





- Activations: $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- Output activation function for regression: $y_k(\boldsymbol{x}) = a_k$
- Output activation function for classification: $y_k(\boldsymbol{x}) = \sigma(a_k)$
- Sigmoid function: $\sigma(a) = \frac{1}{1+\exp(-a)}$
- Two-layer NN:

$$y_k(\boldsymbol{x}) = \sigma\left(\sum_j w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.
- All the previous is, of course, generalizable to more layers.

Backpropagation Algorithm

- Consider regressing an K-dimensional continuous random variable on a D-dimensional continuous random variable.
- Consider a training set $\{(\boldsymbol{x}_n, \boldsymbol{t}_n)\}$. Consider minimizing the error function

$$E(\boldsymbol{w}^t) = \sum_n E_n(\boldsymbol{w}^t) = \sum_n \frac{1}{2}\|\boldsymbol{y}(\boldsymbol{x}_n) - \boldsymbol{t}_n\|^2 = \sum_n \sum_k \frac{1}{2}(y_k(\boldsymbol{x}_n) - t_{nk})^2$$

- The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- Batch gradient descent

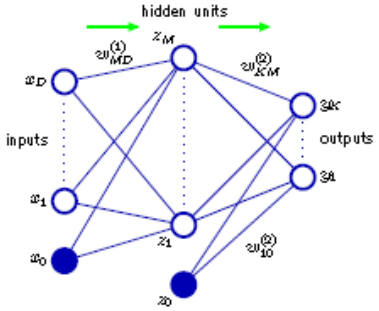
$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_n \nabla E(\boldsymbol{w}^t)$$

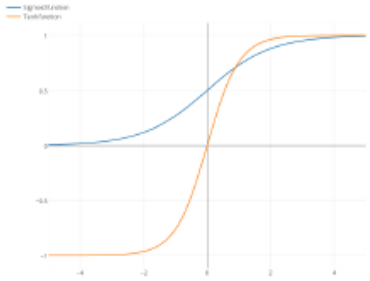
  where $\eta_n > 0$ is the learning rate ($\sum_n \eta_n = \infty$ and $\sum_n \eta_n^2 < \infty$ to ensure convergence, e.g. $\eta_n = 1/n$).
- Sequential, stochastic or online gradient descent

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_n \nabla E_n(\boldsymbol{w}^t)$$

  where n is chosen randomly or sequentially.
- Sequential gradient descent is less affected by the multimodality problem, as a local minimum of the whole data will not be generally a local minimum of each individual point.

Backpropagation Algorithm





- Example: $y_k = a_k$, and $z_j = h(a_j) = \tanh(a_j)$ where $\tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$.
- Note that $h'(a) = 1 - h(a)^2$.
- Backpropagation:
  1. Forward propagation, i.e. compute
  $$a_j = \sum_i w_{ji} x_i \text{ and } z_j = h(a_j) \text{ and } y_k = \sum_j w_{kj} z_j$$
  2. Compute
  $$\delta_k = y_k - t_k$$
  3. Backpropagate, i.e. compute
  $$\delta_j = (1 - z_j^2) \sum_k w_{kj} \delta_k$$
  4. Compute
  $$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

R MARKDOWN

echo=FALSE To Hide Code eval=FALSE To Hide Output