Exam help document TDDE01

1.0 Theory

1.1 Basic concepts

1.1.1 Bias-variance tradeoff

Bias: The inability of a model to capture the true relationship. High bias = underfittning
Variance: Using a complex model will fit the training data well but it is very sensitive to change. High variance = model was overfitted to training data.
Optimal model: Low bias – it can accurately represent the relationship. Low variance – makes consistently good predictions on different data sets.
In decision trees, the bias-variance tradeoff involves balancing simplicity and complexity. A tree with too few branches (high bias) may not capture all the patterns in the data, leading to underfitting. Conversely, a tree with many branches (low bias) may fit the training data too closely, including noise, resulting in overfitting (high variance). The goal is to create a tree that is complex enough to accurately model the data but simple enough to perform well on new, unseen data. Pruning methods help achieve this balance by reducing overfitting while maintaining predictive power.

1.1.2 Supervision Supervised training (lecture 1a)

Data set has a desired outcome, a right answer. Unsupervised does not.

1.1.3 Cross entropy loss (minus log likelihood)

The lower the better prediction to the true value, e.g. perfect model Cross-entropy loss = 0

```
cross.entropy <- function(p, phat){
  x <- 0
  for (i in 1:length(p)){
    x <- x + (p[i] * log(phat[i]))
  }
  return(-x)
}
```

1.1.4 Holdout method (lecture 1a)

Separate data into training and test - Train on training data, evaluate on testing data

1.1.5 Cross validation, K-fold Cross validation

1. Separate data set into k partitions 2. Train on k-1 partitions 3. Test on 1 partition 4. Choose: a. Model with smallest CV-score b. Parameter (e.g. lambda in Ridge) with smallest CV-score.

1.1.6 Holdout vs cross validation

CV is always better than holdout, but more computationally heavy - Holdout is easy and fast, CV requires many iterations - Holdout is good for large data sets, CV is good for small data sets

1.1.7 Confusion matrix

Displays the number of correct and incorrect predictions compared to the actual values, with one axis representing the predicted classes and the other the actual classes.

	PREDICTED			
TRUE		0	1	Total
	0	TN	FP	N
	1	FN	TP	P

1.1.8 Accuracy, TPR, Recall, FPR, Specificity, Precision, F1 score

The F1 score is a measure that combines precision and recall of a classification model into a single metric. It's useful because it provides a balance between the precision (how accurate the positive predictions are) and recall (how well the model identifies all relevant cases), making it particularly effective for evaluating models on imbalanced datasets. A high F1 score is good.

ACTUAL VALUES	POSITIVE	NEGATIVE
POSITIVE	TP	FN
NEGATIVE	FP	TN

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

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

$$TPR = TP/P$$

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

$$FPR = FP/N$$

- **Specificity**

$$Specificity = 1 - FPR$$

1.2 Regression, classification, regularization

1.2.1 Continuous - Ridge, Lasso

Ridge - Keep all predictors but make them small to make model less complex
regularization - Good when correlated variables are strongly correlated - Lin reg can't be used when p > n (# features > # observations) since XtX singular (p*p) □ Ridge is good here - Coefficients tend towards zero with growing lambda
Lasso - Variable selection and regularization (shrinking coefficients towards zero) - Good when p >> n - Coefficients become zero with growing lambda
Similarities: Both models are linear. Both the LASSO and Ridge-models penalize the complexity of the model using lambda. Degrees of freedom are a measure of the number of parameters that can vary in a statistic. Since a larger lambda decreases the number of active parameters, the degrees of freedom will also decrease. Features should be standardized, since only one lambda is used.

1.2.2 Discrete (classification) - KNN, Logistic regression (lecture 1a)

K-Nearest Neighbor - Let the K nearest neighbors "vote". Distance is calculated using a kernel (Euclidean distance is standard knn). Low K most complex = High variance low bias
Logistic regression – Discriminant model Fits a S-curve to data. If p(obese) > 0.5: classify as obese. How: Given a line (weights) calculate the likelihood of the observations. Do this for different lines and pick the best.

1.3 Decision trees

Decision trees are used for both classification and regression tasks. They work by splitting the data into subsets based on different criteria, with each node in the tree representing a test on an attribute and each branch representing the outcome of that test. Decision trees are intuitive and easy to visualize but can be prone to overfitting if not properly tuned. To select the optimal number of leaves for your decision tree according to the deviance criterion, you need to perform tree pruning. This process involves growing a large tree and then systematically cutting back the tree to find the subtree that leads to the lowest prediction error. In R, you can do this using the cv.tree function from the tree package, which performs cross-validation to estimate how well the tree will perform on unseen data. The cv.tree function provides a measure of deviance (or error) for each size of the tree. The size with the lowest deviance is usually the best choice. This process helps to balance the bias-variance tradeoff by finding a tree that is neither too complex (high variance) nor too simple (high bias).
How to prune?Use cv.tree to perform cross-validation on your tree. Plot the deviance against the size of the tree (number of leaves) to visualize the relationship. Use the size with the lowest deviance to prune your tree.

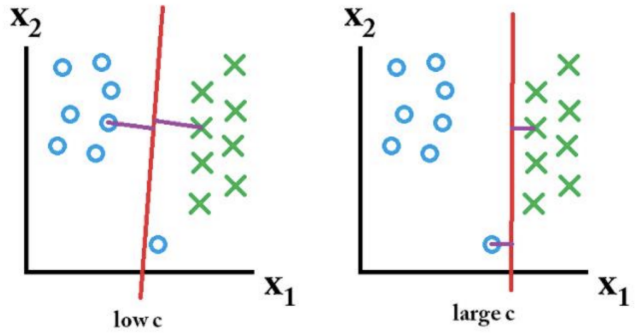
```
# Perform cross-validation
cv_tree <- cv.tree(default_tree, FUN=prune.tree)
# Plot deviance against tree size
plot(cv_tree$size, cv_tree$dev, type="b", xlab="Number of Leaves",
ylab="Deviance")
# Select the optimal size (number of leaves)
optimal_size <- which.min(cv_tree$dev)
# Prune the tree to the optimal size
optimal_tree <- prune.tree(default_tree, best = optimal_size)
# Print the summary of the optimal tree
summary(optimal_tree)
```

A tree with too few leaves (high bias) may not capture all the patterns in the data, leading to underfitting. A tree with too many leaves (high variance) may capture too much noise, leading to overfitting.

1.4 Dimensionality reduction (PCA)
Principle component analysis (PCA) PC1 is a linear combination of all features that explains the most of the total variance in the sample. Loadings = coeff of the linear combination of the initial values used to form the principal component. Features should be standardized! If not, variation in one feature could be much larger than that in another because of the absolute values.

1.5 Kernel Methods
h is the bandwidth of the kernel, determining how quickly the influence of a single training point decreases with distance. The Gaussian kernel maps input features into a higher-dimensional space, allowing linear classification or regression algorithms to find nonlinear relationships in the original feature space. This mapping is achieved implicitly through the kernel trick, avoiding the computational cost of a literal transformation to a higher-dimensional space. The choice of h is crucial, as it affects the flexibility of the model. A small value makes the decision boundary more sensitive to individual points (potentially leading to overfitting), while a large value results in a smoother decision boundary.

1.6 Support Vector Machines (SVM)
The parameter C decides how much we relax this constraint (avoid misclassifying training points).
- Large C-values o Smaller-margin hyperplane that is better at classifying training data
- Risk of overfitting When C is large, larger slacks penalize the objective function of SVM's more than when C is small. As C approaches infinity, this means that having any slack variable set to non-zero would have infinite penalty. Consequently, as C approaches infinity, all slack variables are set to 0 and we end up with a hard-margin SVM classifier. Hard margin SVM's are extremely sensitive to outliers and are more likely to overfit. And because of the nature of the constraints of the hard margin SVM, for training data that is not linearly separable, it will not be able to find a margin at all. - Small C-values Larger-margin hyperplane, even if it misclassifies some training data o Very small values will misclassify linearly separable data o Risk of underfitting When C is small and approaches 0, we essentially have the opposite problem. Our slack variables for all data points are free to be as large as possible to maximize the margin and it's easy for our model to underfit the training data. If C = 0, we are not really classifying anything as we are not enforcing the constraints seen above. We are only minimizing the sum of squares of weights used



1.7 Neural Networks

1.7.1 activation functions
Binary step function: (y = 0 if x < 0, 1 otherwise) - Cannot support multi-class classification. Does not have a derivative in one point
Linear Activation Function: Better than binary, has more than two outputs. But two big problems: a) Impossible to use backpropagation. Backpropagation depends on gradient descent to train. Since the derivative is constant, the algorithm stops working. b) The entire network becomes linear it doesn't matter how many layers you add: the last layer will always be a linear combination of the first layer.
Sigmoid: [0, 1] Good: - [0, 1] --> Normlized, can be interpreted as probability

ReLU: Rectifier e.g. y = max(0, 1) or y = max(0.1 * x, x) Good: - Computationally easy - Non-linear --> Backpropagation Bad: Regular ReLU saturates for x < 0. Leaky ReLU fixes this y = max(0.1 * x, x).

1.8 What is each lecture about?
Lecture 1a - Types of learning, **KKNN**
Lecture 1b - **Statistics**, Bayes, distribution, probability, fitting a model
Lecture 1c - **introduction to R**, vectors, sequence, matrix, factor, functions, plot
Lecture 1d - Linear regression, data scaling, degrees of freedom, Ridge, logistic reg, optimization
Lecture 2a - **model selection**, holdout, Cross validation, Bias variance tradeoff, ROC,
Lecture 2b - **Decision trees**, cross entropy, gini index, selecting tree, pruning, R code for trees
Lecture 2c - **PCA** - principle component analysis, PCA in R, ICA
Lecture 2d - **Parameter optimization**, cost minimization, loss function, glmnet, Lasso, ridge, earl stop
Lecture 3a - **Kernels methods**, histogram, moving window, kernel trick, kernel ridge regression
Lecture 3b - **Support vector machines**, Support vector regression, support vector classification
Lecture 3c - **Neural networks**, backpropagation,
Lecture 3d - **Neural networks** Convolutional Neural Networks (CNN), strengths and weakness of NN

2.0 General R code

2.1.1 Split training/test

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

2.1.2 Split training/test/validation

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

2.2.0 Models

2.2.1 Linear regression + CV

```
library(cvTools)
fit1=lm(V9~V3+V4+V5+V6+V7+V8, data=data)
fit2=lm(V9~V3+V4+V5+V6+V7, data=data)
f1=cvFit(fit1, y=data$V9, data=data,K=10, foldType="consecutive")
f2=cvFit(fit2, y=data$V9, data=data,K=10, foldType="consecutive")
res=cvSelect(f1,f2)
```

```
summary(fit1)
confint(fit1, level=0.95) # CIs for model parameters
residuals(fit1) # residuals
degrees of freedom = trace(P) [P = hat matrix]
fitted <- predict(fit1, interval = "confidence") # plot the data and the fitted
line
attach(mydata)
plot(Year, Price)
lines(Year, fitted[, "fit"]) # plot the confidence bands
lines(Year, fitted[, "lwr"], lty = "dotted", col="blue")
lines(Year, fitted[, "upr"], lty = "dotted", col="blue")
detach(mydata)
```

2.2.2 Lasso/Ridge + CV

```
library(glmnet)
lasso = glmnet(as.numeric(x), as.numeric(y), alpha = 1, family = "gaussian")
plot(lasso, xvar = "lambda", label = TRUE, main="lasso regression")

cross_validation = cv.glmnet(as.numeric(x), as.numeric(y), alpha=1,
family="gaussian")
opt_lambda = cross_validation$lambda.min
predict(cross_validation, newx = x, s = opt_lambda)
coef(cross_validation, s = "lambda.min")
```

2.2.3 Logistic regression

```
logistic_model <- glm(diabetes ~ pgc + age, data = dataframe, family =
"binomial")
predict(logistic_model , newdata = train, type="response")>0.5
#logistic_model ger probabilistic model: 1/(1+exp(z)), z = intercept + b*x1 ...
#Decision boundary is given by x = 0 We classify as 1 when x > 0 (sigm(0) =
0.5).
```

2.2.4 KNN

```
library(kknn)
train$V65 <- as.factor(train$V65)
kknn(V65 ~ ., train = train, test = test, k = 30, kernel = "rectangular")
predict(model_test)
```

2.2.5 Decision tree

```
tree_model <- tree(y ~ ., data = train_data, control =
tree.control(nrow(train_data), mindev = 0.0005))
predict(tree_model, newdata = validation_data, type = "class") | type = "tree"
prune.tree(tree_model, best = leaves)
cv.res=cv.tree(tree_model)
plot(cv.res$size, cv.res$dev, type="b", col="red")
print(finalTree0)
```

2.2.6 PCA (scaling)

```
library(caret) # scaling
```

```
scaler_train = preProcess(train_data) # preProccessa inte test. Predict med
scaler_train
scaled_train_data = predict(scaler_train, train_data)
cov_matrix = cov(train_scaled)
eigen_values = eigen(cov_matrix)
PCA = data.frame(PCA = 1:10, Variance = (eigen_values$values)[1:10])
PCA_res = princomp(train_scaled)
PC1_coordinates = PCA_res$scores[,1]
PC1_loadings = PCA_res$loadings[1, ]
```

2.2.7 Kernel

```
gaussianKernel = function(x_diff, h) {
  return(exp(-(x_diff / h)^2))   }
for (h in seq(0, 1, 0.1)){
  # Prediction using Gaussian kernel. All k(x*-xi/h) for the new_point (x*) to
training data (xi)
  predictions <- rep(0, nrow(x_train))
 for (i in 1:nrow(x_train)) {
       # Compute the difference between the new point and each data point in X
       x_diff <- new_point - x_train[i, ]
       # Apply the Gaussian kernel function to the difference
       kernel_value <- sum(gaussianKernel(x_diff, h))
       predictions[i] <- kernel_value
}
  # Weighted sum of target variable
  weighted_sum <- sum(predictions %*% y_train)
  # Normalizing the prediction
  normalized_prediction <- weighted_sum / sum(predictions)
  mse = c(mse, mean((y_test[1] - normalized_prediction)^2))
}
cauchyKernel <- function(x_diff, h) {
  return(1 / (1 + (x_diff / h)^2))
}
epanechnikovKernel <- function(x_diff, h) {
  norm_x_diff <- x_diff / h
  kernel_values <- ifelse(abs(norm_x_diff) <= 1, (1 - norm_x_diff^2), 0)
  return(kernel_values)
}
uniformKernel <- function(x_diff, h) {
  kernel_values <- ifelse(abs(x_diff/h) < 1, 1, 0)
  return(kernel_values)
}
```

2.2.8 SVM (partitioning included)

```
library(kernlab)
set.seed(1234567890)
data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
```

```
spam[,-58]<-scale(spam[,-58])
train <- spam[1:3000, ]
val <- spam[3001:3800, ]
trainva <- spam[1:3800, ]
test <- spam[3801:4601, ]
filter <-
ksvm(type~.,data=train,kernel="rbfdot",kpar=list(sigma=0.05),C=1,scaled=FALSE)
```

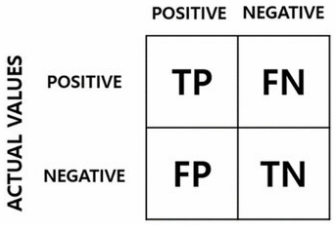
2.2.9 NN (scale, and CNN)

```
linear <- function(x) x
ReLU <- function(x) ifelse(x>0, x, 0)
softplus <- function(x) log(1 + exp(x))
winit <- runif(31, -1, 1)
nn_softplus <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights =
winit, act.fct = softplus)
```

2.3.1 ROC

```
 cm_tree <- table(pred_tree, test_data$y)
  #Sometimes model never predicts "yes", giving no tpr
  if (nrow(cm_tree) >= 2) {
       #TPR = TP / TP + FN
       tpr_tree <- c(tpr_tree, cm_tree[2, 2] / sum(cm_tree[, 2]))
       #FPR =  FP / FP + TN
       fpr_tree <- c(fpr_tree, cm_tree[2, 1] / sum(cm_tree[, 1]))
  }
plot(fpr_tree, tpr_tree, pch = 5, type = "b", col = "red", main = "ROC Curves",
xlab = "False Positive Rate (FPR)", ylab = "True Positive Rate (TPR)")
legend("bottomright", c("Optimal Tree", "Logistic Regression"), fill = c("red",
"blue"))
```

2.3.2 F1/Recall (Sensitivity)/Precision



$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

```
table(test_data$y, test_predictions)
precision <- confusion_matrix[2,2]/(confusion_matrix[2,2] +
confusion_matrix[1,2])
recall_rate <- confusion_matrix[2,2]/sum(confusion_matrix[2,])
f1_score <- (2 * precision * recall_rate) / (precision + recall_rate)
F1[i]=cm[2,2]/(cm[2,2]+0.5*(cm[1,2]+cm[2,1]))
```

2.3.3 Cost functions Classification

2.3.3.1 Hinge loss

```
hinge_loss <- function(y_true, y_pred) {
  loss <- 1 - y_true * y_pred
  loss[loss < 0] <- 0  # Set negative losses to zero
  return(mean(loss))      }
```

2.3.3.2 E-sensitive

```
e_sensitive_loss <- function(y_true, y_pred, epsilon = 0.1) {
  absolute_error <- abs(y_true - y_pred)
  loss <- ifelse(absolute_error <= epsilon, 0, absolute_error - epsilon)
  return(sum(loss))      }
```

2.3.3.3 Exponential loss

```
exponential_loss <- function(y_true, y_pred) {
  loss <- exp(-y_true * y_pred)
  return(mean(loss))         }
```

2.3.3.4 Loss matrix

```
# Decision tree classification with Loss matrix:
loss_matrix <- matrix(c(0, 1, 5, 0), byrow = TRUE, nrow = 2)
# Predictions using the pruned optimal tree
probabilities <- predict(pruned_optimal_tree, newdata = test_data, type =
"vector")
predictions_with_loss = ifelse(probabilities[,"yes"] * loss_matrix[2,1] <
probabilities[,"no"] * loss_matrix[1,2], "no", "yes")
```

2.3.4 Cost functions Classification

2.3.3.1 Cost functions Classification Mean squared error

mse_train = mean((target_train - predictions_train)^2)

2.3.3.2 Gradient decent (optim)

```
cost_function = function(theta) {
  predictions_train = x_train %*% theta
  predictions_test = x_test %*% theta
  mse_train = mean((target_train - predictions_train)^2)
  mse_test = mean((target_test - predictions_test)^2)
  #Saving the MSE values to the vector
  MSE_training <<- c(MSE_training, mse_train)
  MSE_testing <<- c(MSE_testing, mse_test)
  return(mse_train)
}
opt_result = optim(par = theta, fn = cost_function, method = "BFGS")
```

2.1 Syntax

```
if (x==1){}
#For loop ----------------------------------------------
for (pi in seq(0, 1, 0.9)){
  predict_tree_optimal <- ifelse(prob_tree_optimal[, 2]>pi, TRUE, FALSE)
  }
#function -----------------------------------------------
myfun <- function(x=20, y){
return(x)}
rep(value, repetitions)
#Vector -------------------------------------------------
length(a)
sum(a^2)
max(a) #Största värde
which.max(a) #Index av största värde
which(train$V65 == 8)
sort(prob_8)
order(prob_8, decreasing = TRUE)  #Vector av index
prob_8[-2] #exkludera index 2
#Filtrera data ------------------------------------------
st = st[st$date <= specified_date & st$time %in% times,]
train_data_without_target <- subset(train_data, select = -Class)
#Matrix apply -------------------------------------------
b_inverse=solve(b)
x_train = as.matrix(train_scaled[,-101])
x[2,-(1:5)] | #row 2 and all columns except 1:5
x[x>5] # alla värden större än 5
m3 = matrix(c(1,0,3), nrow = 2, ncol = 3, byrow = T)
dim(m3)
colMeans(m3), rowMeans(m3), colSums(m3),rowSums(m3)
cbind(m3, 1:2) lägger till en kolumn med värden 1:30
#level mapping ------------------------------------
numeric_f3 <- as.numeric(test_data$f3)
unique_values <- unique(numeric_f3)
data.frame(Level = levels(test_data$f3), NumericRepresentation = unique_values)
#List----------------------------------------------------
b <- list(first = 10, second = c(0, 1, 0), x = "mary")
#b$x → "mary" | b[[3],] → "mary"
coef(cross_validation, s = "lambda.min")


#SApply ------------------------------------------------------------
sapply(m3, log) # log av alla värden
function_train_data <- train_data
function_train_data[-1] <- lapply(2:ncol(train_data), function(i) {
  if (is.numeric(train_data[, i])) {
    function_train_data[, i] ^ i
  } else {
    train_data[, i]        }
})
library(dplyr)
```

```
function_train_data=train_data%>%mutate_if(is.numeric, list(x2=function(x) x^2))
#Conversion ------------------------------------------------------------
f4 = as.factor(c(1,4,2,2,1)) # [1] 1 4 2 2 1 \n Levels: 1 2 4
as.list(c("1", "0")) # [[1]] \n [1] "1 \n [[2]] \n [1] "0"
as.numeric(list(a=1, b=2) → 1 2
```

2.1.1 Back/forward propagation

```
set.seed(1234)
# produce the training data in dat
x <- runif(500,-4,4)
y <- sin(x)
dat <- cbind(x,y)
plot(dat)
gamma <- 0.01
h <- function(z){
  # activation function (sigmoid)
  return(1/(1+exp(-z)))                    }
hprime <- function(z){
  # derivative of the activation function (sigmoid)
  return(h(z) * (1 - h(z)))                    }
yhat <- function(x){
  # prediction for point x
q0 <- x
z1 <- w1 %*% q0 + b1
q1 <- as.matrix(apply(z1,1,h), nrow = 2, ncol = 1)
z2 <- w2 %*% q1 + b2
return(z2)
}
MSE <- function(){
  res <- NULL
  for(i in 1:nrow(dat)){
       res <- c(res,(dat[i,2] - yhat(dat[i,1])) ^ 2)
       }
  return(mean(res))
}
# initialize parameters
w1 <- matrix(runif(2,-.1,.1), nrow = 2, ncol = 1)
b1 <- matrix(runif(2,-.1,.1), nrow = 2, ncol = 1)
w2 <- matrix(runif(2,-.1,.1), nrow = 1, ncol = 2)
b2 <- matrix(runif(1,-.1,.1), nrow = 1, ncol = 1)
res <- NULL
for(i in 1:100000){
  if(i %% 1000 == 0){
       res <- c(res,MSE())
  }
   # forward propagation
  j <- sample(1:nrow(dat),1)
  q0 <- dat[j,1]
  z1 <- w1 %*% q0 + b1
  q1 <- as.matrix(apply(z1,1,h), nrow = 2, ncol = 1)
```

```
  z2 <- w2 %*% q1 + b2
   # backward propagation
  dz2 <- - 2 * (dat[j,2] - z2)
  dq1 <- t(w2) %*% dz2
  dz1 <- dq1 * hprime(z1)
  dw2 <- dz2 %*% t(q1)
  db2 <- dz2
  dw1 <- dz1 %*% t(q0)
  db1 <- dz1
  # parameter updating
  w2 <- w2 - gamma * dw2
  b2 <- b2 - gamma * db2
  w1 <- w1 - gamma * dw1
  b1 <- b1 - gamma * db1
}
plot(res, type = "l")
plot(dat)
points(dat[,1],lapply(dat[,1],yhat),col="red")
```

2.1.2 Plots

```
plot(1:8, train_error, ylab = "Misclassification Error", xlab = "Leaves", col =
"blue", type = "l", lty = 1, ylim = c(0, max(train_error, validation_error)),
main = "Missclass error")points(2:50, validScore[2:50], type = "b", col =
"green")
lines(1:8, validation_error, col = "red", type = "l", lty = 2)
legend("topright", legend = c("Training", "Validation"), col = c("blue", "red"),
lty = 1:2, cex = 0.8)
text(pruned_optimal_tree, pretty=1)

#hist(x,..) #plots a hisogram, persp(x,y,z,...) surface plots,
cloud(formula,data..) 3D scatter plot
#ggplot2-------------------------------------------------------------------
ggplot(data.frame(PC1 = PCA_res$scores[,1], PC2 = PCA_res$scores[,2]),
aes(x=PC1, y=PC2)) + geom_point(aes(color = Violent_Crimes)) +
scale_color_gradient(low = "blue", high = "red") + ggtitle("PC scores") +
xlab("PC1 scores") +  ylab("PC2 scores") +  theme_minimal()
#Heatmap ------------------------------------------------------------------
easiest_case_1_matrix <- matrix(as.numeric(easiest_cases_data[1, -65]), nrow =
8, ncol = 8)
heatmap(t(easiest_case_1_matrix), Colv = "Rowv", Rowv = NA)
```

2.1.3 Help commands

- Help browser – help.start()
- Search for something in help – help.search(“expression”)
- A quick reminder of function arguments: – args(function)
- Examples of how to use the function: – example(function)
- If some method is not installed on the computer: – RSiteSearch(”expression")

2.1.4 Random number generator

rnorm(n, mean = 0, sd = 1) returns *n* random numbers normally distributed.

dnorm(0) returns the density of the standard normal distribution at 0.
pnorm(1.96) gives the probability that a standard normally distributed variable is < or equal to 1.96.
qnorm(0.95) gives the value below which 95% of the data in a standard normal distribution fall.

2.1.5 Data manipulation - extercises coursepage

```
#E2 - Change row names in tecator1 to the values of Sample column plus 10
rownames(tecator1) = tecator1$Sample + 10
#E3 - Change column name in tecator1 from Sample to ID
colnames(tecator1)[1] ="ID"
#E4 - Extract rows in tecator1 such that Channel1 > 3 and Channel2 > 3 and columns
between number 5 and number 8
tecator1[tecator1$Channel1 > 3 & tecator1$Channel2 > 3, 5:8]
#E5 - Remove column ID in tecator1 (two options)
tecator1$ID=c()
tecator1 = tecator1[, -1]
#E6 - Update tecator1 by dividing its all Channel columns with their respective
means per column (two options)
for (i in 1:100) {
  # Assuming the channel columns are named as "Channel1", "Channel2", ...,
"Channel100"
  channel_col_name = paste("Channel", i, sep="")
  # Calculate the mean of the i-th Channel column
  divider = mean(tecator1[[channel_col_name]])
  # Divide the i-th Channel column by its mean
  tecator1[[channel_col_name]] = tecator1[[channel_col_name]] / divider
}
#Oleg
library(stringr)
index=str_which(colnames(tecator1), "Channel")
tecatorChannel=tecator1[,index]
means=colMeans(tecatorChannel)
tecator1[,index]=tecator1[,index]/matrix(means, nrow=nrow(tecatorChannel),
ncol=ncol(tecatorChannel), byrow=TRUE)
#E7 - Compute a sum of squares for each row between 1 and 5 in tecator1 without
writing loops and make it as a matrix with one column
m = matrix(c(rowSums(tecator1[1,]^2), rowSums(tecator1[2,]^2),
rowSums(tecator1[3,]^2), rowSums(tecator1[4,]^2), rowSums(tecator1[5,]^2)), nrow =
5, ncol = 1)
#Oleg
sumsq=apply(tecator1[1:5,], MARGIN = 1, FUN=function(x) return(sum(x^2)) )
tecator2=matrix(sumsq, ncol=1)
#E8 - Extract X as all columns except of columns 101-103 in tecator1, y as column
Fat and compute (XTX)−1XTy
X=as.matrix(tecator1[,1:100])
y=as.matrix(tecator1$Fat)
res=solve(t(X)%*%X)%*%t(X)%*%y
#Oleg
X=as.matrix(tecator1[,-c(101, 102, 103)]) #can be written more efficiently as
-(101:103)
y=as.matrix(tecator1[,"Fat", drop=F]) #keep it as a matrix, don't reduce
dimension.
result=solve(t(X)%*%X, t(X)%*%y)
#E9 - Use column Channel1 in tecator1 to compute new column ChannelX which is a
factor with the following levels: "high" if Channel1>1 and "low" otherwise
tecator1$ChannelX=as.factor(ifelse(tecator1$Channel1>1, "high", "low"))
#E10 - Write a for loop that computes regressions Fat as function of
Channeli,i=1,...100 and then stores the intercepts into vector Intercepts. Print
Intercepts.
channels = tecator1[,1:100]
```

```r
intercepts = numeric(length = ncol(channels))
for (i in 1:ncol(channels)) {
  dataset= data.frame(channels[,i])
  fit1 = lm(tecator1$Fat ~., data=dataset)
  intercepts[i] = coef(fit1)[1]
}
print(intercepts)
#Oleg
Intercepts=numeric(100)
for (i in 1:length(Intercepts)){
  regr=lm(formula=paste("Fat~Channel", i, sep=""), data=tecator1)
  Intercepts[i]=coef(regr)[1]
}
print(Intercepts)
#E11 - Given equation y=5x+1, plot this dependence for x between 1 and 3
x = c(1,3)
y=5*x+1
plot(x,y, type="1"
#Data manipulation: dplyr and tidyr
#E1 - Convert data set birth to a tibble birth1
library(dplyr)
library(tidyr)
birth1=tibble(birth)
#E2 - Select only columns X2002-X2020 from birth1 and save into birth2
birth2 = birth1%>%select(X2002:X2020)
#E3 - Create a new variable Status in birth1 that is equal to "Yes" if the record
says "born in Sweden with two parents born in Sweden" and "No" otherwise
birth1$Status = ifelse(birth1$foreign.Swedish.background == "born in Sweden with
two parents born in Sweden", "Yes", "No")
#with dplyr
birth1=birth1%>%mutate(Status=ifelse(foreign.Swedish.background=="born in Sweden
with two parents born in Sweden", "Yes", "No"))
#E4 - Count the amount of rows in birth 1 corresponding to various combinations of
sex and region
birth1%>%count(sex,region)
#E5 - Assuming that X2002-X2020 in birth1 show amounts of persons born respective
year, compute total amount of people born these years irrespective gender, given
Status and region. Save the result into birth3
birth3=birth1%>%select(-sex,- foreign.Swedish.background)%>%group_by(region,
Status)%>%summarise_all(sum)%>%ungroup()
birth3
#E6 - By using birth3, compute percentage of people in 2002 having Status=Yes in
different counties. Report a table with column region and Percentage sorted by
Percentage.
birth4=birth3%>% group_by(region)%>% mutate(Percentage=X2002/sum(X2002)*100)%>%
filter(Status=="Yes")%>% select(region, Percentage)%>% ungroup()%>%
arrange(Percentage)
birth4
#E7 - By using birth1, transform the table to a long format: make sure that years
are shown in column Year and values from the respective X2002-X2020 are stored in
column Born. Make sure also that Year values show years as numerical values and
store the table as birth5.
birth5 = birth1 %>% group_by(region, sex, foreign.Swedish.background, Status) %>%
pivot_longer(X2002:X2020, names_to = "Year", values_to = "Born") %>%
mutate(Year=as.numeric(stringr::str_remove(Year, "X")))
#E8 - By using birth5, transform the table to wide format: make sure that years
are shown as separate columns and their corresponding values are given by Born.
Columns should be named as "Y_2002" for example.
birth6 = birth5 %>% group_by(region, sex, foreign.Swedish.background, Status)%>%
```

```
pivot_wider(names_from = Year, values_from = Born, names_prefix = "Y_")
#E9 - By using blog data, filter out columns that have zeroes everywhere.
blogS=tibble(blog)%>%select_if(function(x) !all(x==0))
blogS
```

3.0 Computer labs

Computer lab 1 KNN, linear & ridge regression, logistic reg

Assignment 1 - Handwritten digit recognition with K-nearest neighbors.

Task 1.1 - Import the data into R and divide it into training, validation and test sets

The optdigits.csv data was split into train, test and validation sets of 50%/25%/25% randomly. This was done by setting an "id" list of integers, selecting rows of the imported optdigits.csv data frame (*id=sample(1:n, floor(n*0.5))*). Then these randomly generated rows were selected from the data and assigned to the training set (*train=data[id,]*). Similarly the rest of the data was split into set and validation. See appendix 1.1 for more detailed code.

Task 1.2 - Use train data to fit 30-NN classifier, estimate - CM table(), Misclassification errors

Then two models were created based on kknn. One with training as its test data set, and one with test as its test data set. *model_train* and *model_test* respectively.

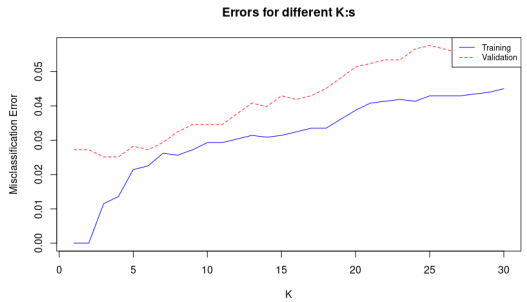
Predictions were made of the models on their respective test data and summarized into a confusion matrix. The overall prediction quality was quite similar for predictions on training data versus test data, around 95%. However, predictions made on training data were slightly better, about 1%. Simply put digits 1, 2, 7 and 9 were difficult to predict both on train and test data.

Task 1.3 - Find 2 cases of digit "8" in train data which were easiest to classify and 3 cases that were hardest to classify (i.e. having highest and lowest probabilities of the correct class). Reshape features as matrix 8x8 and visualize using heatmap() function with parameters Colv=NA and Rowv=NA)

The two cases where the model was most confident the digit was an eight and the three cases where it was the least difficult were extracted.

Task 1.4 - Fit a KNN classifiers to the train for different values of K =
1-30 and plot dependence of train and valid MissCE on the value of K. Optimal K according to this plot? estimate test error for the model having optimal K, compare with the train and valid

Misclassification error was plotted for different k values, shown by the graph below.



Complexity decreases as K is increased. In this case it also leads to higher error rates, indicating that the model is getting overfitted for higher K:s.

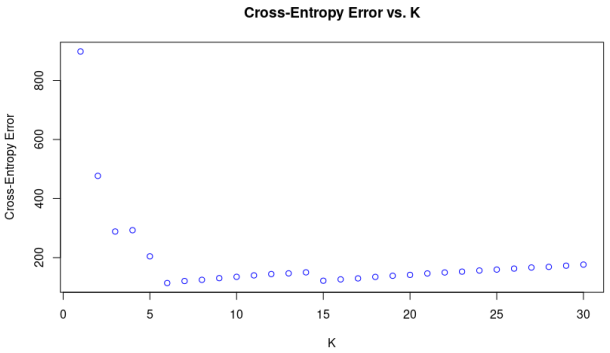
The optimal K for validation is 3. For K=3 the errors for the sets are:

Test Data:	0.02403344
Validation Data:	0.02513089
Training Data:	0.0115123

This shows that testing on the training data once again leads to a lower misclassification error, since that is what the model is trained on. However, tests and validation get a higher, about similar, error rate. This is reasonable as neither are separate for model training.

Task 1.5 - Fit KNN to train for diff K, compute error valid as cross entropy, plot dependence of error on K. Optimal K here? CE vs ME?

The cross-entropy was plotted for different K:s, similarly as in Task 4.



The optimal K here is 6. Cross-entropy might be a more suitable choice for this task of identifying digits, since it takes into account the model's confidence in its prediction, while misclassification rate regards a 99% incorrect classification the same as a 51% incorrect classification.

Assignment 2 - Linear regression and ridge regression

Task 2.1 - divide 60/40 and scale, motor_UPDRS is target

The first task was to import the "parkinsons.csv" file, divide it into training- and test data and scale it. The file was imported with the "read.csv()" function and scaled with the "preProcess()" function, which is a part of the external library "caret". The preProcess() function uses the method = c("center", "scale")) by default, which subtracts the mean and divides by the standard deviation for each column variable in the csv file. This way the variables contribute equally to the analysis which is necessary for the model to perform well.

```
######################## Task 1 ##########################################
#Divide the data into training and test (60/40) and scale it appropriately.

#Clean the data
ps_data = read.csv("parkinsons.csv", header = TRUE)
ps_clean_data = ps_data[,5:22] # Exclude subject, age, sex and test_time
ps_clean_data = ps_clean_data[,-2] #Exclude total_updirs => total of 17 vars

#Divide the data into training and test sets
set.seed(12345)  # For reproducibility
n = dim(ps_clean_data)[1] #numb of rows/obs
id = sample(1:n, floor(n * 0.6)) #take a random sample
train_data = ps_clean_data[id, ]
test_data = ps_clean_data[-id, ]

# Scale the data
scaler = preProcess(train_data) #subtracts mean and divides by std by using the default method: c("center", "scale")
train_scaled = predict(scaler, train_data) #scaling the training data by applying scaler transform
test_scaled = predict(scaler, test_data)
```

Task 2.2 - Linear regression + errors

The second task was to compute a linear regression model and estimate MSE, as well as comment on the most significant variables.

```
############################## Task 2 ####################################
#Building the linear regression model using the training data

ps_model = lm(motor_UPDRS ~ ., data = train_scaled)
summary(ps_model)

#Analysis of P-value to visualize the most significant variables
coefficients = summary(ps_model)$coefficients #Extracting coefficients and p-values from the summary

#Creating a data frame for the significant variables
significant_vars = data.frame(value = coefficients[coefficients[, "Pr(>|t|)"] < 0.05, "Pr(>|t|)"])
print(significant_vars)

#Predicting 'motor_UPDRS' using the linear model
predicted_train = predict(ps_model, train_scaled)
predicted_test = predict(ps_model,test_scaled)

#Calculate Mean Squared Error for training and test data
mse_train = mean((train_scaled$motor_UPDRS - predicted_train)^2)
mse_test = mean((test_scaled$motor_UPDRS - predicted_test)^2)

print(paste("Training MSE: ", mse_train))
print(paste("Test MSE: ", mse_test))
```

To achieve this a linear regression model was created with the “lm()” function. The model’s purpose was to predict the response variable “motor_UPDRS" based on the values of the predictors. The model summary provided coefficients and their P-vaules which were analyzed in order to comment on the significant variables. Small P-values (typically < 0.05) indicate strong evidence against the null hypothesis and in favor of significance. For these variables the null hypothesis can be rejected and it can be concluded that they contribute significantly to the model. To visualize this a separate table “significant_vars" was created where the variables with P-vaules < 0.05 were extracted. This can be seen below.

variable	value
Jitter.Abs.	3.316870e-05
Shimmer	4.054884e-03
Shimmer.APQ5	6.690868e-04
Shimmer.APQ11	6.365957e-07
NHR	4.849817e-05
HNR	6.450884e-11
DFA	6.566828e-43
PPE	6.750371e-12

To calculate the mean squared error we took the actual value of the response variable, which could be extracted from the “motor_UPDRS” column in the scaled training- and test data. We then subtracted the predicted value which was received from the model, squared it and took the mean. This measurement represents the average squared difference between the observed actual outcomes and the outcomes predicted by the model. In this case the MSE was ~0.879 for the training data and ~0.935 for the test data. The model has not seen the test data and since it is quite close to the training MSE it can be concluded that the model generalizes to new data quite well.

Training MSE: 0.878543102826276
Test MSE: 0.935447712156708

Task 2.3 - Implement Log-likelihood, ridge, ridigeOpt, DF

The third task was to implement four functions without any external packages. First, some global variables were defined that could be accessed by all the functions.

```r
#global variables for the functions
n = nrow(train_scaled) #Number of observations
y = as.matrix(train_scaled$motor_UPDRS) #The observed value in the motor_UPDRS column
x = as.matrix(train_scaled)[,2:17] #The predictor variables, columns Jitter... to PPE
```

The first function that was implemented was Log-likelihood.

```r
#Loglikelihood: how well does the model fit the data
loglikelihood <- function(theta, sigma) {
  return(-n / 2 * log(2 * pi * sigma^2) - (1 / (2 * sigma^2)) * sum((y - x %*% theta)^2))
}
```

After that the ridge function was implemented which call the loglikelihood function.

```r
#Ridge function
# Ridge penalty prevents overfitting by shrinking the coefficients
#=> less model complexity and improved generalization.
ridge_function = function(theta, lambda){
  sigma = theta[length(theta)]
  theta = theta[-17]
  # Calculate the Ridge penalty
  ridge_penalty = lambda * sum(theta^2)
  #call logLikelihood and add ridge_penalty to get ridge_value
  ridge_value = -loglikelihood(theta, sigma) + ridge_penalty
  return(ridge_value)
}
```

The third function optimized the theta and sigma variables. Optim() was fed a vector of 17 ones as an initial guess to start from.

```r
#Finds the parameter values that minimizes the loss function, starting from an initial guess.
RidgeOpt = function(lambda) {
  # Initial guesses for theta and sigma (17 ones)
  initial_guess = rep(1, 17)
  # Use optim to minimize the ridge regression loss function
  return(optim(initial_guess, fn = ridge_function, lambda = lambda, method = "BFGS"))
}
```

The last function that was implemented calculated the degrees of freedom.

```r
#Degree of freedom function
DF = function(lambda) {
  # Ensure the diagonal matrix I has the same dimensions as the number of predictors in x
  I = diag(ncol(x))
  # Compute the hat matrix H
  H = x %*% solve(t(x) %*% x + lambda * I) %*% t(x)
  # Calculate the degrees of freedom as the sum of the diagonal elements of H
  return(sum(diag(H)))
}
```

Task 2.4 - compute opt theta, report new MSE, DF conclusions?

In the last subtask the optimal values of theta were calculated using the ridgeOpt function for the given lambda values 1, 100 and 1000. The estimated parameters were then used to predict the response variable and calculate MSE for the training. and test data. This was implemented with a for loop that looped through a vector with the different lambda values.

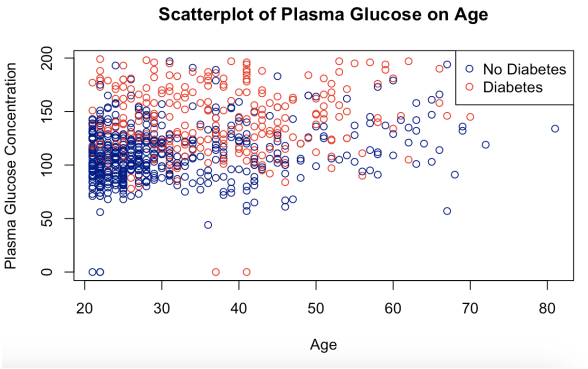
```r
# Defining lambda values
lambda_values <- c(1, 100, 1000)

#converting into matrices
x_train <- as.matrix(train_scaled %>% select(Jitter...:PPE))
x_test <- as.matrix(test_scaled %>% select(Jitter...:PPE))

#Loop through the lambda vector
for (l in lambda_values) {
  #computing optimal theta coefficients
  theta_opt = RidgeOpt(lambda = l)
  #extracting optimal coefficients
  opt_extracted = as.matrix(theta_opt$par[-17])
  pred_train_opt = x_train %*% opt_extracted
  pred_test_opt  = x_test %*% opt_extracted
  #MSE for training and test
  mse_train_opt = mean((train_scaled$motor_UPDRS - pred_train_opt)^2)
  mse_test_opt  = mean((test_scaled$motor_UPDRS  - pred_test_opt)^2)
  #degrees of freedom for respective lambda value
  df = DF(lambda = l)
  #prints
  print(paste("lambda =",l,"MSE training:", mse_train_opt))
  print(paste("lambda =",l,"MSE training:", mse_test_opt))
  print(paste("lambda =",l,"Degrees of freedom:", df))
}
```

The optimal penalty parameter lambda is 100, as it yields the lowest test MSE, indicating the best predictive performance. With a degree of freedom around 9.925, this model balances complexity and generalization well. The chosen lambda minimizes overfitting, suggesting a model that's complex enough to capture underlying trends but not so much that it fits irrelevant noise.

```
[1] "lambda = 1 MSE training: 0.878627189502917"
[1] "lambda = 1 MSE test: 0.934998015205606"
[1] "lambda = 1 Degrees of freedom: 13.8607362829965"
[1] "lambda = 100 MSE training: 0.884405556541252"
[1] "lambda = 100 MSE test: 0.932329601901113"
[1] "lambda = 100 Degrees of freedom: 9.92488712829542"
[1] "lambda = 1000 MSE training: 0.921117541108582"
[1] "lambda = 1000 MSE test: 0.953945694481502"
[1] "lambda = 1000 Degrees of freedom: 5.6439254878463"
```

Assignment 3 - logistic regression and basis function expansion

Task 3.1 - Make scatterplot of glucose on age, where obs are colored by diabetic level. Is diabetes easy to classify by logistic reg?



The “pima-indians-diabetes.csv” file consisted of data about medical details and the onset of diabetes in 5 years.

Firstly, the medical data “Age” and “Plasma glucose concentration” were shown in a scatter plot, as seen in the Figure to the right. In this figure, Red means diabetes and blue means no diabetes Based on this plot, classifying diabetes by a standard logistic regression model seems to be hard. There is no intuitive way to determine diabetes/no diabetes based on this, although a high plasma glucose concentration seems to be correlated to having diabetes.

Task 3.2 - see text

In the second task, a logistic regression model was trained using y = Diabetes as target, x1 = Plasma Glucose concentration, x2 = Age as features to make predictions for the observations using r = 0.5 as classifying threshold. The probabilistic equation for this is the following as a result of the estimated model is the following: $P\,(Diabetes\ =\ 1)\ =\ \frac{1}{1+e^{-(\beta_0+\beta_{pgc}*pgc\ +\beta_{age}*age)}}$

Where the values for β are derived from the following table, in the Estimate column:

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -5.912449   0.462620  -12.78  < 2e-16 ***
pgc          0.035644   0.003290   10.83  < 2e-16 ***
age          0.024778   0.007374    3.36 0.000778 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

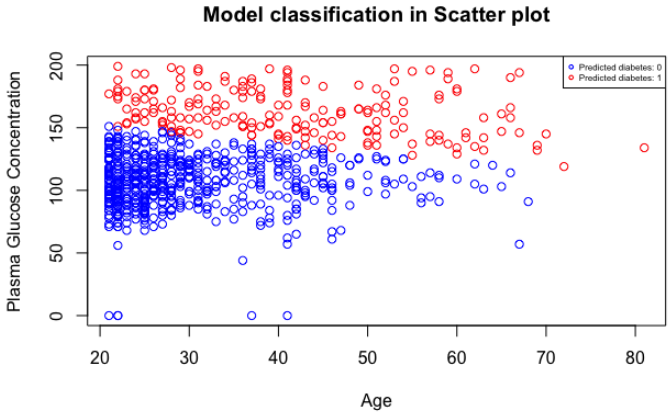
The table comes from the following code:

```r
logistic_model <- glm(diabetes ~ pgc + age, data = dataframe, family = "binomial")

summary(logistic_model)
```

The training misclassification error can be derived from the following code:

```r
prediction <- predict(logistic_model, dataframe, type = "response")
# r = 0.5
prediction1 <- ifelse(prediction > 0.5, 1, 0)

# Confusion matrix
confusion_matrix <- table(prediction1, diabetes)
print(confusion_matrix)

# Missclassification
misclass <- (1 - sum(diag(confusion_matrix)) / length(prediction1))
print(paste("Misclassification error:", misclass)) #0.2630208
```

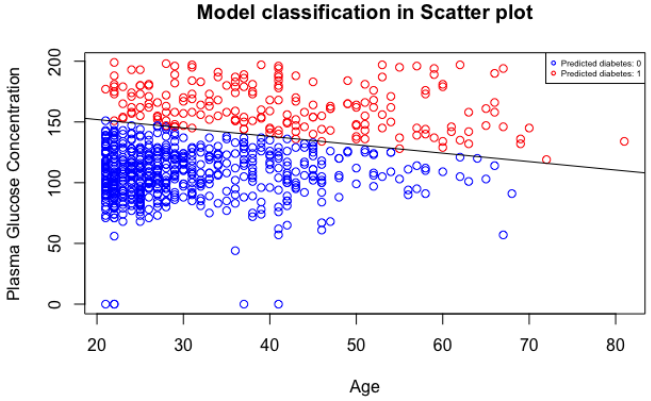
The misclassification error from this model is roughly 0.26, which is quite high. It seems as if the model cannot accurately classify diabetes based on these two factors alone. Furthermore, we can plot the classifications that the model makes with the following plot.



Task 3.3 - use model above, report equation of decision boundary, add curve for the boundary
Adding the curve showing this boundary can be done with the following code:

```r
# Decision boundry line
abline(a = coef(logistic_model)[["(Intercept)"]] / (-coef(logistic_model)[["pgc"]]),
       b = coef(logistic_model)[["age"]] / (-coef(logistic_model)[["pgc"]]),
       col = "black")
```

gives below:



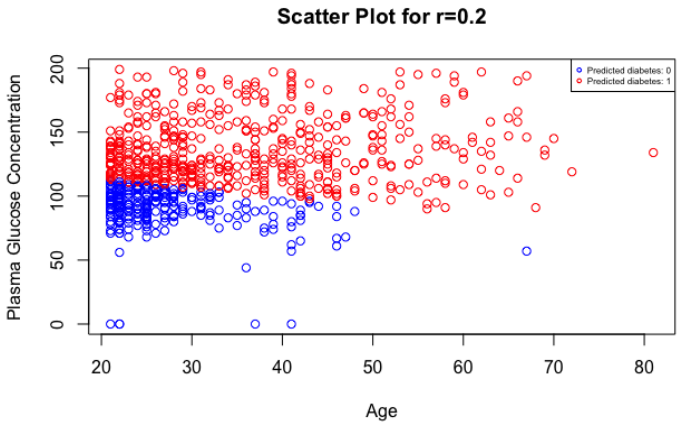
The decision boundary line can be observed. The equation for this is that coef(logistic_model)[["age"]] / (-coef(logistic_model)[["pgc"]]) is the slope and coef(logistic_model)[["(Intercept)"]] / (-coef(logistic_model)[["pgc"]] is the intercept.

Task 3.4 - same plot as above but with threshold r = 0.2 and 0.8, what happens when r changes

For the threshold 0.2:

```
# Prediction threshold r=0.2
prediction_2 <- ifelse(prediction > 0.2, 1, 0)

# r=0.2
plot(age, pgc, col = color_values[as.factor(prediction_2)],
     xlab = "Age", ylab = "Plasma Glucose Concentration",
     main = "Scatter Plot for r=0.2")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes: 1"),
       col = c("blue", "red"), pch = 1, cex = 0.5)


# Missclassification for r=0.2
confusion_matrix2 <- table(prediction_2, diabetes)
print(confusion_matrix2)
## Comment: Frequent prediction of diabates when this is not the case

misclass2 <- (1 - sum(diag(confusion_matrix2)) / sum(confusion_matrix2))
print(paste("Misclassification error:", misclass2)) #0.37239583
```

The same is done but with a new threshold, which results in a worse misclassification rate of roughly 0.37. Plotting this we get the following:

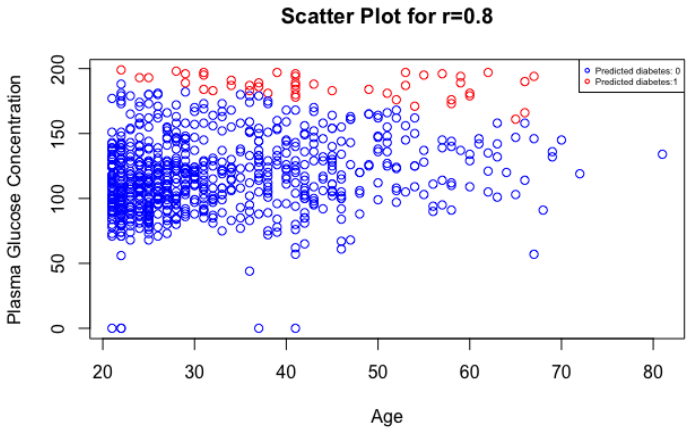


We can observe that the model now often predicts diabetes, compared to the model with the threshold of 0.5.

For the threshold 0.8:

```
# Prediction threshold r=0.8
prediction_3 <- ifelse(prediction > 0.8, 1, 0)

# r=0.8
plot(age, pgc, col = color_values[as.factor(prediction_3)],
     xlab = "Age", ylab = "Plasma Glucose Concentration",
     main = "Scatter Plot for r=0.8")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
       col = c("blue", "red"), pch = 1, cex=0.5)


# Misclassification for r=0.8
confusion_matrix3 <- table(prediction_3, diabetes)
print(confusion_matrix3)
## Comment: Frequent missed prediction of diabetes
misclass3 <- (1 - sum(diag(confusion_matrix3)) / sum(confusion_matrix3))
print(paste("Misclassification error:", misclass3)) #0.31510416
```

This still results in a worse misclassification rate than the model with r=0.5, but a better one than r=0.2. The misclassification rate for this one is roughly 0.31. If we plot the predictions:



We can then observe that the model seldom predicts diabetes, which is due to the high threshold. When the r value is 0.2, the model will predict mostly diabetes=true, while it is 0.8 the model will predict mostly diabetes=false. Overall, both values lead to a high misclassification rate than using r=0.5.

Task 3.5 - Perform basis function expansion trick by computing new features $z1$ = $x1$^4, $z2$ = $x1^{3}x2$, $z3$ = $x1^{2}x2$^2, $z4$ = $x1x2$^3, $z5$ = $x2$^4 adding them to the data set and then computing a logistic regression model with y as target and $x1$, $x2$, $z1$-$z5$ as features.

The basis function expansion trick is done by doing the following:

```
dataframe$z1 <- pgc^4
dataframe$z2 <- pgc^3 * age
dataframe$z3 <- pgc^2 * age^2
dataframe$z4 <- pgc * age^3
dataframe$z5 <- age^4
y <- diabetes

model <- glm(y ~ pgc + age + z1 + z2 + z3 + z4 + z5, data = dataframe, family = "binomial")
summary(model)

prediction_basis <- predict(model, dataframe, type = "response")
prediction_basis <- ifelse(prediction_basis > 0.5, 1, 0)

# Confusion Matrix and misclassification rate
cm_basis <- table(prediction_basis, y)
print(cm_basis)
misclass_basis <- (1 - sum(diag(cm_basis)) / sum(cm_basis))
print(paste("Misclassification error of new model:", misclass_basis))


plot(age, pgc, col = color_values[as.factor(prediction_basis)],
     xlab = "Age", ylab = "Plasma Glucose Concentration",
     main = "Scatter Plot on basis model")

legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
       col = c("blue", "red"), pch = 1, cex=0.5)
```

The misclassification rate for this model is roughly 0.24, which is an improvement from the first model but still quite high. So the prediction accuracy is a bit higher than the first model. Plotting this with the new boundary line:

It can be observed that the shape of the boundary line has changed, but it still cannot accurately predict diabetes. A conclusion from this could be that predicting diabetes solely from age and plasma glucose concentration seems to be difficult.

Computer lab 1 - code

```
Assignment 1
#####TASK1#####
rm(list = ls())
data = read.csv("optdigits.csv", header = FALSE)
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.5))
train = data[id, ]
rest = data[-id, ]
n2 = dim(rest)[1]
id2 = sample(1:n2, floor(n2 * 0.5))
test = rest[id2, ]
validation = rest[-id2, ]
rm(rest)
#####TASK2#####
library(kknn)
# Convert the outcome variable to a factor
train$V65 <- as.factor(train$V65)
test$V65 <- as.factor(test$V65)
# Fitting 30-nearest neighbor classifier on training data
model_train <- kknn(V65 ~ ., train = train, test = train, k = 30, kernel =
"rectangular")
# Fitting 30-nearest neighbor classifier on test data
model_test <- kknn(V65 ~ ., train = train, test = test, k = 30, kernel =
"rectangular")
# Making predictions on training and test data
prediction_train <- predict(model_train)
prediction_test <- predict(model_test)
# Confusion matrices
cm_train <- table(prediction_train, train$V65)
cm_test <- table(prediction_test, test$V65)
# Computing misclassification rates from confusion matrices
misclass_train <- 1 - sum(diag(cm_train)) / sum(cm_train)
misclass_test <- 1 - sum(diag(cm_test)) / sum(cm_test)
# Displaying results
print("Confusion Matrix for Training Data:")
print(cm_train)
print("Confusion Matrix for Test Data:")
print(cm_test)
print(paste("Misclassification Rate for Training Data:", misclass_train))
print(paste("Misclassification Rate for Test Data:", misclass_test))
##Comment on the quality of predictions for different digits and on the overall
##prediction quality
#Ans: Overall good. ~95% accuracy. 0.045 vs. 0.048 miss class.
```

```
#####TASK3#####
# Extract probabilities guessing 8 for each image in the train data.
prob_8 <- model_train$prob[, 9]# Easiest cases
easiest_cases <- order(prob_8, decreasing = TRUE)
easiest_cases <- easiest_cases[1:2]
# Hardest cases
hardest_cases <- order(prob_8)
hardest_cases <- hardest_cases[1:3]
# Extract the easiest and hardest features from the train data
easiest_cases_data <- train[easiest_cases, ]
hardest_cases_data <- train[hardest_cases, ]
# Remove last column and reshape into 8x8 numeric matrix
easiest_case_1_matrix <- matrix(as.numeric(easiest_cases_data[1, -65]), nrow =
8, ncol =8)
easiest_case_2_matrix <- matrix(as.numeric(easiest_cases_data[2, -65]), nrow =
8, ncol =8)
# Plot in heatmaps
heatmap(t(easiest_case_1_matrix), Colv = "Rowv", Rowv = NA)
heatmap(t(easiest_case_2_matrix), Colv = "Rowv", Rowv = NA)
# Remove last column and reshape into 8x8 numeric matrix
hardest_cases_1 <- matrix(as.numeric(hardest_cases_data[1, -65]), nrow = 8, ncol
= 8)
hardest_cases_2 <- matrix(as.numeric(hardest_cases_data[2, -65]), nrow = 8, ncol
= 8)
hardest_cases_3 <- matrix(as.numeric(hardest_cases_data[3, -65]), nrow = 8, ncol
= 8)
# Plot in heatmaps
heatmap(t(hardest_cases_1), Colv = "Rowv", Rowv = NA)
heatmap(t(hardest_cases_2), Colv = "Rowv", Rowv = NA)
heatmap(t(hardest_cases_3), Colv = "Rowv", Rowv = NA)
##Comment on whether these cases seem to be hard or easy to recognize visually.
##Ans: First one pretty difficult. Second one easy. All the three "hardest
cases" did indeed
seem difficult.
#####TASK4#####
# Create a sequence of K values from 1 to 30
key <- 1:30
# Initialize vectors to store misclassification errors for training and
validation
missclass_errortr <- numeric(30)
missclass_errorv <- numeric(30)
# Loop over different values of K
for (i in key) {
# Fit K-nearest neighbor classifier on training data
nearest1 <- kknn(as.factor(V65) ~ ., train = train, test = train, k = i, kernel
= "rectangular")
# Calculate misclassification error for training data
cm_nearest1 <- table(train$V65, predict(nearest1))
missclass_error_train <- 1 - sum(diag(cm_nearest1) / sum(cm_nearest1))
missclass_errortr[i] <- missclass_error_train
# Fit K-nearest neighbor classifier on validation data
```

```r
nearest2 <- kknn(as.factor(V65) ~ ., train = train, test = validation, k = i,
kernel ="rectangular")
# Calculate misclassification error for validation data
cm_nearest2 <- table(validation$V65, predict(nearest2))
missclass_error_valid <- 1 - sum(diag(cm_nearest2) / sum(cm_nearest2))
missclass_errorv[i] <- missclass_error_valid}
# Plot misclassification errors for training and validation
plot(key, missclass_errortr, ylab = "Misclassification Error", xlab = "K", col =
"blue", type = "l", lty = 1, ylim = c(0, max(missclass_errortr,
missclass_errorv)), main = "Errors for different K:s")
lines(key, missclass_errorv, col = "red", type = "l", lty = 2)
legend("topright", legend = c("Training", "Validation"), col = c("blue", "red"),
lty = 1:2, cex =0.8)
# Find the index where validation error is minimized
optimal_k_index <- which.min(missclass_errorv)
optimal_k <- key[optimal_k_index]
# Fit K-nearest neighbor classifier on training data with optimal K
optimal_nearest <- kknn(as.factor(V65) ~ ., train = train, test = test, k =
optimal_k, kernel ="rectangular")
# Calculate misclassification error for test data
cm_optimal <- table(test$V65, predict(optimal_nearest))
missclass_error_test <- 1 - sum(diag(cm_optimal) / sum(cm_optimal))
##How does the model complexity change when K increases and how does it affect
the training and validation errors?
##ANS: Complexity increases when K increases, as the number of parameters
increase.
## In this case it also leads to higher error rates, indicating overfitting.
##Report the optimal 𝐾 according to this plot.
cat("Optimal K:", optimal_k, "\n")
##ANS: 1
##Finally, estimate the test error for the model having the optimal K,
##compare it with the training and validation errors and make necessary
conclusions about the model quality.
cat("Misclassification Error for Training Data:", missclass_errortr[optimal_k],
"\n")
cat("Misclassification Error for Validation Data:", missclass_errorv[optimal_k],
"\n")
cat("Misclassification Error for Test Data:", missclass_error_test, "\n")
##ANS: Around 97% accuracy for validation and test, which is good.
## Testing on training data naturally gives 100% accuracy when k=1.
# TASK 5 #
key <- 1:30
entropy_error <- numeric(length = 30)
for (i in key) {
# Fit K-nearest neighbor classifier on validation data
nearest_valid <- kknn(as.factor(V65) ~ ., train = train, test = validation, k =
i, kernel = "rectangular")
# Initialize variables for cross-entropy calculation
cross_entropy_valid <- 0
# Loop over digits (0 to 9)
for (digit in 0:9) {
```

```
# Extract true labels for the digit in the validation set
true_valid <- validation$V65 == digit
# Extract probabilities for the digit in the validation set ("digit+1" because
vector is 1 indexed in $prob)
prob_valid <- nearest_valid$prob[true_valid, digit + 1] + 1e-15
# Calculate cross-entropy for validation data by += summary of all probabilities
for the digit.
cross_entropy_valid <- cross_entropy_valid + sum(-log(prob_valid))}
# Store the cross-entropy error for the given K
entropy_error[i] <- cross_entropy_valid}
# Plot the dependence of the validation error on the value of K
plot(key, entropy_error, ylab = "Cross-Entropy Error", xlab = "K", col = "blue",
main = "Cross-Entropy Error vs. K")
# Find the optimal K value
optimal_k_entropy <- which.min(entropy_error)
cat("Optimal K (based on cross-entropy):", optimal_k_entropy, "\n")
## Assuming that response has multinomial distribution,
## why might the cross-entropy be a more suitable choice
## of the error function than the misclassification error for this problem?
##ANS: Misclassification rate doesn't take into account the probabilities of the
model's predictions.
```

Assignment 2

```
#libraries
library(caret)
library(dplyr)
# Clear global environment
rm(list = ls())
############################ Task 1#############################################
#Divide the data into training and test (60/40) and scale it appropriately.
#Clean the data
ps_data = read.csv("parkinsons.csv", header = TRUE)
ps_clean_data = ps_data[,5:22] # Exclude subject, age, sex and test_time columns
ps_clean_data = ps_clean_data[,-2] #Exclude total_updirs => total of 17 vars
#Divide the data into training and test sets
set.seed(12345) # For reproducibility
n = dim(ps_clean_data)[1] #numb of rows/obs
id = sample(1:n, floor(n * 0.6)) #take a random sample
train_data = ps_clean_data[id, ]
test_data = ps_clean_data[-id, ]
#Scale the data
scaler = preProcess(train_data) #subtracts mean and divides by std by using the
default
method: c("center", "scale")
train_scaled = predict(scaler, train_data) #scaling the training data by
applying scaler transform
test_scaled = predict(scaler, test_data)
############################### Task 2#########################################
#Building the linear regression model using the training data
ps_model = lm(motor_UPDRS ~ . , data = train_scaled)
summary(ps_model)
```

```r
#Analysis of P-value to visualize the most significant variables
coefficients = summary(ps_model)$coefficients #Extracting coefficients and
p-values from
the summary
#Creating a data frame for the significant variables
significant_vars = data.frame(value = coefficients[coefficients[, "Pr(>|t|)"] <
0.05, "Pr(>|t|)"])
print(significant_vars)
#Predicting 'motor_UPDRS' using the linear model
predicted_train = predict(ps_model, train_scaled)
predicted_test = predict(ps_model,test_scaled)
#Calculate Mean Squared Error for training and test data
mse_train = mean((train_scaled$motor_UPDRS - predicted_train)^2)
mse_test = mean((test_scaled$motor_UPDRS - predicted_test)^2)
print(paste("Training MSE: ", mse_train))
print(paste("Test MSE: ", mse_test))
################################ Task 3#####################################
#global variables for the functions
n = nrow(train_scaled) #Number of observations
y = as.matrix(train_scaled$motor_UPDRS) #The observed value in the motor_UPDRS
column
x = as.matrix(train_scaled)[,2:17] #The predictor variables, columns Jitter...
to PPE
#Loglikelihood: how well the model fits the data
loglikelihood = function(theta, sigma) {
return(-n / 2 * log(2 * pi * sigma^2) - (1 / (2 * sigma^2)) * sum((y - x %*%
theta)^2))}
#Ridge function
# Ridge penalty prevents overfitting by shrinking the coefficients
#=> less model complexity and improved generalization.
ridge_function = function(theta, lambda){
sigma = theta[length(theta)]
theta = theta[-17]
# Calculate the Ridge penalty
ridge_penalty = lambda * sum(theta^2)
#call logLikelihood and add ridge_penalty to get ridge_value
ridge_value = -loglikelihood(theta, sigma) + ridge_penalty
return(ridge_value)}
#Finds the parameter values that minimizes the loss function, starting from an
initial guess.
RidgeOpt = function(lambda) {
# Initial guesses for theta and sigma (17 ones)
initial_guess = rep(1, 17)
# Use optim to minimize the ridge regression loss function
return(optim(initial_guess, fn = ridge_function, lambda = lambda, method =
"BFGS"))}
#Degree of freedom function
DF = function(lambda) {
# Ensure the diagonal matrix I has the same dimensions as the number of
predictors in x
I = diag(ncol(x))
```

```
# Compute the hat matrix H
H = x %*% solve(t(x) %*% x + lambda * I) %*% t(x)
# Calculate the degrees of freedom as the sum of the diagonal elements of H
return(sum(diag(H)))
}
#########################################Task4#####################################
# Defining lambda values
lambda_values <- c(1, 100, 1000)
#converting into matrices
x_train <- as.matrix(train_scaled %>% select(Jitter...:PPE))
x_test <- as.matrix(test_scaled %>% select(Jitter...:PPE))
#Loop through the lambda vector
for (l in lambda_values) {
#computing optimal theta coefficients
theta_opt = RidgeOpt(lambda = l)
#extracting optimal coefficients
opt_extracted = as.matrix(theta_opt$par[-17])
pred_train_opt = x_train %*% opt_extracted
pred_test_opt = x_test %*% opt_extracted
#MSE for training and test
mse_train_opt = mean((train_scaled$motor_UPDRS - pred_train_opt)^2)
mse_test_opt = mean((test_scaled$motor_UPDRS - pred_test_opt)^2)
#degrees of freedom for respective lambda value
df = DF(lambda = l)
#prints
print(paste("lambda =",l,"MSE training:", mse_train_opt))
print(paste("lambda =",l,"MSE training:", mse_test_opt))
print(paste("lambda =",l,"Degrees of freedom:", df))
```

Assignment 3

```
dataframe <- read.csv('pima-indians-diabetes.csv', header = FALSE)
color_values <- c("blue", "red")
set.seed(12345)
################################### TASK 1#####################################
# Independent variables
pgc <- dataframe$V2 # Plasma glucose concentration
age <- dataframe$V8
# Dependent variables
diabetes <- dataframe$V9
plot(age, pgc, col = ifelse(diabetes == 1, "red", "darkblue"),
xlab = "Age", ylab = "Plasma Glucose Concentration",
main = "Scatterplot of Plasma Glucose on Age")
# Legend
legend("topright", legend = c("No Diabetes", "Diabetes"),
col = c("darkblue", "red"), pch = 1)
#Do you think that Diabetes is easy to classify by a standard logistic
regression model
#that uses these two variables as features? Motivate your answer.
#Motivation: No, not easy to classify. Hard to determine based on only these
factors.
#if we look at the plot there is no intuitive way to determine diabetes/no
```

```r
diabetes
################################### TASK 2###################################
logistic_model <- glm(diabetes ~ pgc + age, data = dataframe, family =
"binomial")
summary(logistic_model)
prediction <- predict(logistic_model, dataframe, type = "response")
# r = 0.5
prediction1 <- ifelse(prediction > 0.5, 1, 0)
# Confusion matrix
confusion_matrix <- table(prediction1, diabetes)
print(confusion_matrix)
# Missclassification
misclass <- (1 - sum(diag(confusion_matrix)) / length(prediction1))
print(paste("Misclassification error:", misclass)) #0.2630208
plot(age, pgc, col = color_values[as.factor(prediction1)],
xlab = "Age", ylab = "Plasma Glucose Concentration",
main = "Model classification in Scatter plot")
# Legend
legend("topright", c("Predicted diabetes: 0", "Predicted diabetes: 1"), col =
c("blue", "red"),
pch = 1, cex = 0.5)
################################### TASK 3###################################
# Plot
plot(age, pgc, col = color_values[as.factor(prediction1)],
xlab = "Age", ylab = "Plasma Glucose Concentration",
main = "Scatter Plot")
# Decision boundry line
abline(a = coef(logistic_model)[["(Intercept)"]] /
(-coef(logistic_model)[["pgc"]]),
b = coef(logistic_model)[["age"]] / (-coef(logistic_model)[["pgc"]]),col =
"black")
################################### TASK 4###################################
# Prediction threshold r=0.2
prediction_2 <- ifelse(prediction > 0.2, 1, 0)
# r=0.2
plot(age, pgc, col = color_values[as.factor(prediction_2)],
xlab = "Age", ylab = "Plasma Glucose Concentration",
main = "Scatter Plot for r=0.2")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes: 1"),
col = c("blue", "red"), pch = 1, cex = 0.5)
# Missclassification for r=0.2
confusion_matrix2 <- table(prediction_2, diabetes)
print(confusion_matrix2)
## Comment: Frequent prediction of diabates when this is not the case
misclass2 <- (1 - sum(diag(confusion_matrix2)) / sum(confusion_matrix2))
print(paste("Misclassification error:", misclass2)) #0.37239583
# Prediction threshold r=0.8
prediction_3 <- ifelse(prediction > 0.8, 1, 0)
# r=0.8
plot(age, pgc, col = color_values[as.factor(prediction_3)],
xlab = "Age", ylab = "Plasma Glucose Concentration", main = "Scatter Plot for
```

```
r=0.8")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
col = c("blue", "red"), pch = 1, cex=0.5)
# Misclassification for r=0.8
confusion_matrix3 <- table(prediction_3, diabetes)
print(confusion_matrix3)
## Comment: Frequent missed prediction of diabetes
misclass3 <- (1 - sum(diag(confusion_matrix3)) / sum(confusion_matrix3))
print(paste("Misclassification error:", misclass3)) #0.31510416
## When the r value is 0.2, the model will predict mostly diabetes=true,
## while it is 0.8 the model will predict mostly diabetes=false.
## Overall, both values lead to a high misclassification rate.
##################################### TASK 5#####################################
dataframe$z1 <- pgc^4
dataframe$z2 <- pgc^3 * age
dataframe$z3 <- pgc^2 * age^2
dataframe$z4 <- pgc * age^3
dataframe$z5 <- age^4
y <- diabetes
model <- glm(y ~ pgc + age + z1 + z2 + z3 + z4 + z5, data = dataframe, family =
"binomial")
summary(model)
prediction_basis <- predict(model, dataframe, type = "response")
prediction_basis <- ifelse(prediction_basis > 0.5, 1, 0)
# Confusion Matrix and misclassification rate
cm_basis <- table(prediction_basis, y)
print(cm_basis)
misclass_basis <- (1 - sum(diag(cm_basis)) / sum(cm_basis))
print(paste("Misclassification error of new model:", misclass_basis))
plot(age, pgc, col = color_values[as.factor(prediction_basis)],
xlab = "Age", ylab = "Plasma Glucose Concentration",
main = "Scatter Plot on basis model")
legend("topright", legend = c("Predicted diabetes: 0", "Predicted diabetes:1"),
col = c("blue", "red"), pch = 1, cex=0.5)
# Add the decision boundary line
x_vals <- seq(min(age), max(age), length.out = 100)
y_vals <- seq(min(pgc), max(pgc), length.out = 100)
# Expanded grid with basis functions
new_data <- expand.grid(age = x_vals, pgc = y_vals)
new_data$z1 <- new_data$pgc^4
new_data$z2 <- new_data$pgc^3 * new_data$age
new_data$z3 <- new_data$pgc^2 * new_data$age^2
new_data$z4 <- new_data$pgc * new_data$age^3
new_data$z5 <- new_data$age^4
# Add the decision boundary
contour(x = x_vals, y = y_vals, z = matrix(predict(model, newdata = new_data,
type = "response"), ncol = length(y_vals)), levels = 0.5, add = TRUE, col =
"black")
```

Computer lab 2 - Explicit regularization, Decision trees and logistic regression, PCA

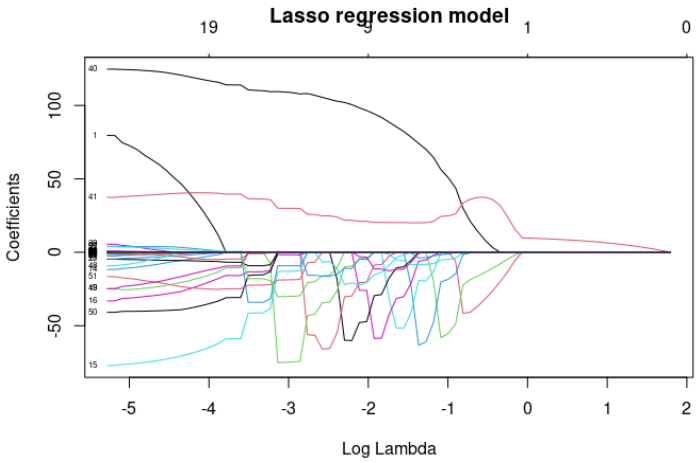
2.1 Explicit regularization

TASK 1 - Assume Fat can be modeled as linear regression where (Channels) are features.
Misclassification rate train: 0.00570911701090834
Misclassification rate test: 722.429419336971
An analysis based on this misclassification rate is that there seems to be overfitting of the training data, leading to a high misclassification rate for the test data. The overall quality of the model is quite low since it has a high misclassification rate for the testing data.
TASK 2 - Assume that Fat can be modeled as LASSO regression, report cost func optimized?
The cost function that should be optimized in this scenario is the following:

$$cost = argmin\{\frac{1}{n}\sum_{i=1}^{n}(Fat_i - \theta_0 - \theta_1 x_{1i} - \ldots - \theta_{100i} x_{100i})^2 + \lambda \sum_{j=1}^{p} abs|\theta_j|\}$$

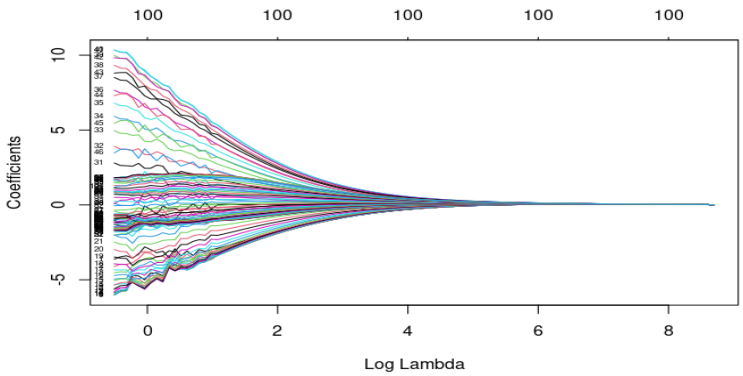
All of the channels are used in this function, so the channels 2-99 are also included, but not shown to simplify the visualization of the model.
TASK 3 - Fit LASSO regression to the training data. Present plot on how the regression coefficients depend on the log of penalty factor (log λ) and interpret this plot.

```
################################### TASK 3 ###################################

## LASSO regression

x = as.matrix(train_data %>% select(-Fat))
y = as.matrix(train_data %>% select(Fat))

lasso = glmnet(x, y, alpha = 1, family = "gaussian")
plot(lasso, xvar = "lambda", label = TRUE, main="Lasso regression model")
```



We can see from this plot that as log(λ) increases, the number of coefficients used decreases. This is due to the penalty becoming larger and therefore forcing coefficients to be zero.
What value of the penalty factor can be chosen if we want to select a model with only three features?
When interpreting the plot, we can observe that if we only want three features, we could implement a log lambda of about -0.3 (Ahead of when coefficient 40 is 0 but before when the red and green line turns to 0). Implementing this lambda would lead to only three features being used in the Lasso model.
TASK 4 - Repeat step 3 but fit Ridge instead of LASSO and compare plots from steps 3 and 4.

```
################################### TASK 4 ###################################

## Ridge regression

ridge = glmnet(x, y, alpha = 0, family="gaussian")
plot(ridge, xvar = "lambda", label = TRUE, main="Ridge regression")
```



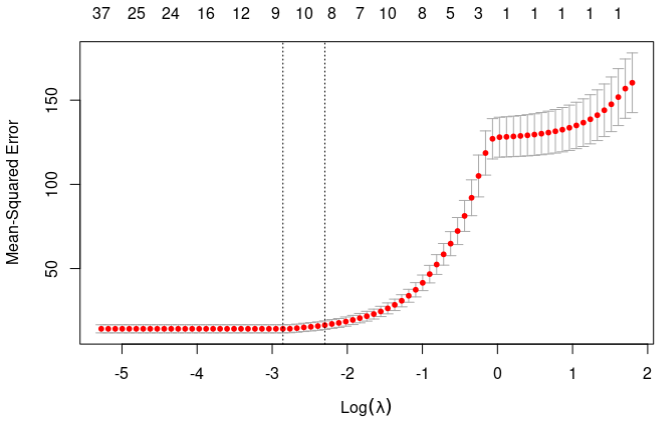
A conclusion that could be drawn from this is that Ridge Regression with L2 Regularization shrinks the coefficients towards zero when the log lambda increases but it is never exactly zero whereas the Lasso Regression with L1 Regularization forces the coefficients to exactly zero. Lasso effectively does feature selection, whereas Ridge just shrinks some parameters. Ridge regression is also harder to interpret, since it retains all of the features. Lasso on the other hand uses feature selection making it more interpretable by highlighting the important features as determined by the model.

2.1.5 TASK 5 - Use cross-validation with default folds to compute the optimal LASSO model

Compute optimal LASSO model

cross_validation = cv.glmnet(x, y, alpha=1, family="gaussian")
opt_lambda = cross_validation$lamda.min

Plot showing dependence of the CV score on log(λ)



The cross-validation score seems to increase as log(λ) increases. It increases heavily from between -2 to 1 and then seems to stagnate.

Optimal λ and variables chosen

The optimal lambda is: 0.05733535

coef(cross_validation, s="lambda.min")

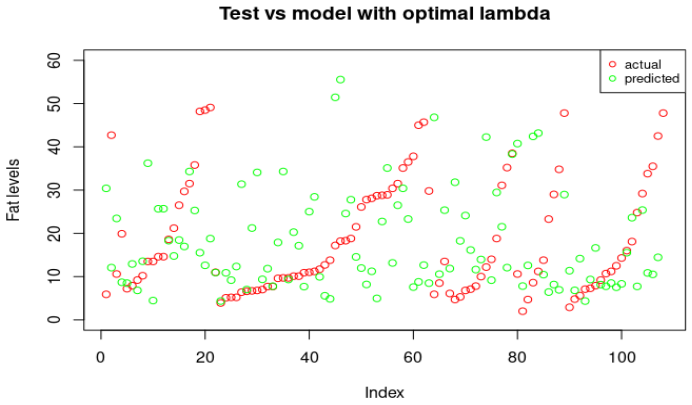
From this, we can see the channels used in the model, these are channels 52, 51, 41, 50, 16, 15, 14, 13 are chosen. 8 in total.

Does the information displayed in the plot suggest that the optimal λ value results in a statistically significantly better prediction than log(λ) = -4?

No, based on the graph it does not suggest that a value of -4 is significantly better since it seems that the mean-squared error for both is roughly the same. The change in mean-squared error can not be visually observed to be different.

Create a scatter plot of the original test versus predicted values for the model corresponding to optimal lambda and comment on whether the model predictions are

```
# Cross validation prediction
crossval_predict = predict(cross_validation, newx = x, s = opt_lambda)

plot(test_data$Fat, col="red", ylim = c(0,60), ylab="Fat levels", main = "Test vs model with optimal lambda")
points(crossval_predict, col = "green")
legend("topright", c("actual", "predicted"), col = c("red", "green"), pch = 1, cex = 0.8)
```



The predictions from the model seem to have quite a high degree of error. There is no clear pattern where the model successfully predicts the actual fat values and it seems to be quite random or sporadic. However, it is with mentioning that some of the values are predicted quite accurately, but there is no clear way of seeing a pattern of correct predictions based on this graph.

2.2 Decision trees and logistic regression for bank marketing

TASK 2

Report the misclassification rates for the training and validation data.

Training	**Validation**
default: 0.1048	default: 0.1092
minsize: 0.1048	minsize: 0.1092
mindev: 0.0936	mindev: 0.1118

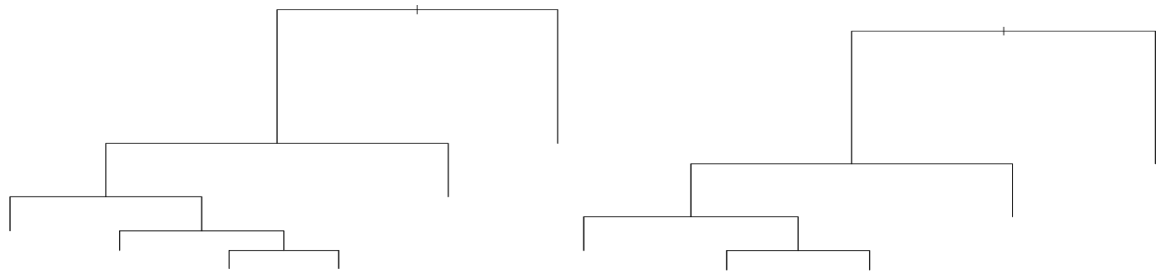
Which model is the best one among these three?

Default and node gets the same missclass on unseen data (valid), so equally good. mindev performs the worst because it is overfitted.

Report how changing the deviance and node size affected the size of the trees and explain

Restriction of each node containing a minimum amount of 7000 data points (default 10) made the tree smaller, since one branching was therefore not allowed to be done.

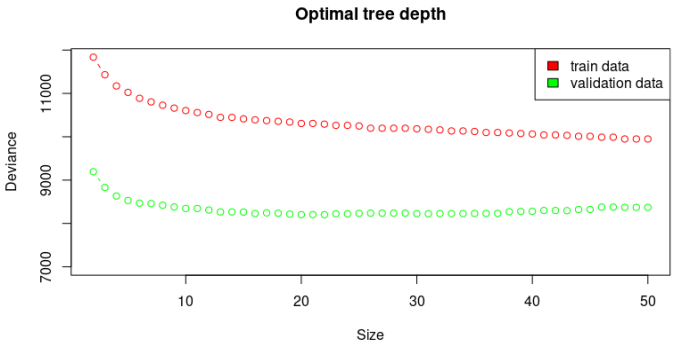
Restriction of each branching needing a mindev of 0.0005 (default 0.01) to occur led to a very large overfitted tree because lots of branching could be done.



Default minsize

2.2.3 TASK 3 - optimal tree? study up tro 50 leaves,

Present a graph of the dependence of deviances for the training and the validation data on the number of leaves

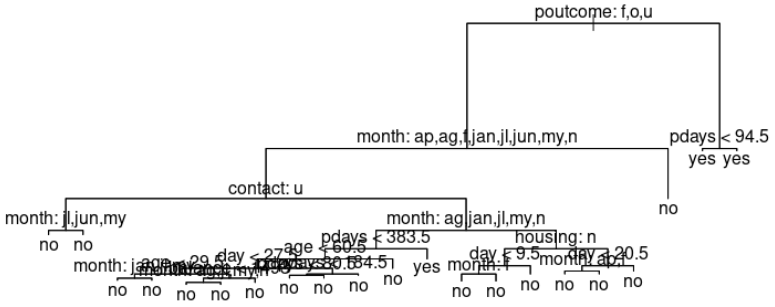


Interpret this graph in terms of bias-variance tradeoff.

It performs better on unseen data (validation lower deviance), indicating high bias and low variance. Train's deviance decreases with more leaves, indicating high bias and low variance. (more complex models, more leaves,)High Variance (Overfitting): The model is too complex, fitting the training data too closely and not generalizing well to new data.High Bias (Underfitting): The model is too simple and does not capture the underlying patterns in the data.

Report the optimal amount of leaves and which variables seem to be most important for decision making in this tree.

Optimal leaves are 47 for training data and 21 for validation data.



poutcome and *month* are variables most important, since they are branched on first. The other 2nd layer branching to the right covered in text is also poutcome.

Interpret the information provided by the tree structure (most important findings).

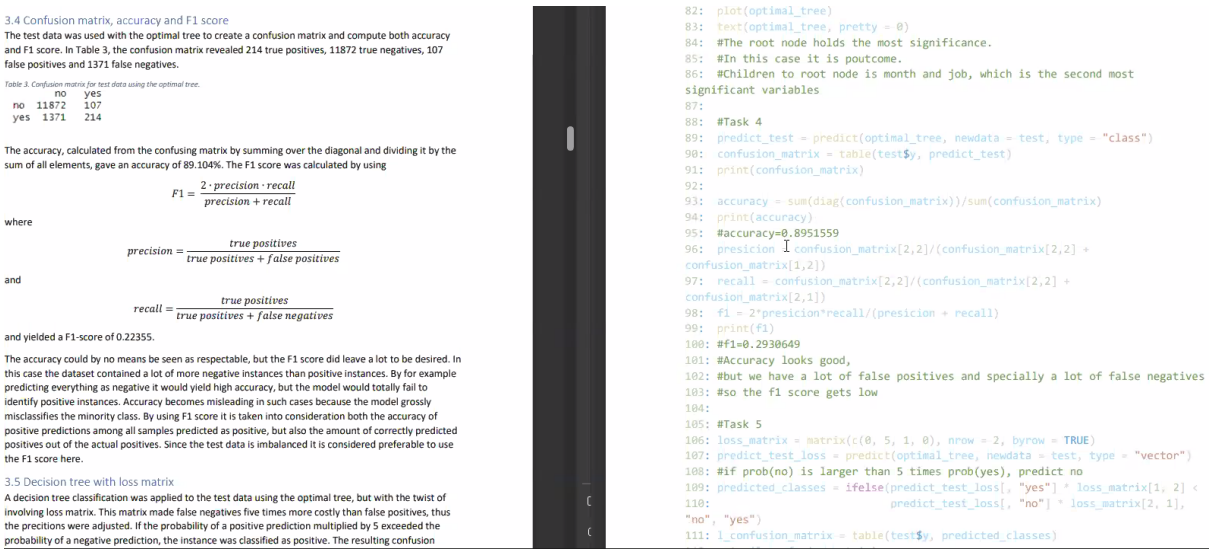
poutcome not being *f*, *o*, or *u* important to predict *y*. Same for *month* not being *april, august, february, january, july, june, may* or *november*.

2.2.4 TASK 4

Estimate the confusion matrix, accuracy and F1 score.

test_predictions		
	no	yes
no	11812	167
yes	1294	291

Accuracy:	0.8922884
F1:	0.2848752



3.4 Confusion matrix, accuracy and F1 score

The test data was used with the optimal tree to create a confusion matrix and compute both accuracy and F1 score. In Table 3, the confusion matrix revealed 214 true positives, 11872 true negatives, 107 false positives and 1371 false negatives.

Table 3. Confusion matrix for test data using the optimal tree.

	no	yes
no	11872	107
yes	1371	214

The accuracy, calculated from the confusing matrix by summing over the diagonal and dividing it by the sum of all elements, gave an accuracy of 89.104%. The F1 score was calculated by using

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

where

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

and

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

and yielded a F1-score of 0.22355.

The accuracy could by no means be seen as respectable, but the F1 score did leave a lot to be desired. In this case the dataset contained a lot of more negative instances than positive instances. By for example predicting everything as negative it would yield high accuracy, but the model would totally fail to identify positive instances. Accuracy becomes misleading in such cases because the model grossly misclassifies the minority class. By using F1 score it is taken into consideration both the accuracy of positive predictions among all samples predicted as positive, but also the amount of correctly predicted positives out of the actual positives. Since the test data is imbalanced it is considered preferable to use the F1 score here.

3.5 Decision tree with loss matrix

A decision tree classification was applied to the test data using the optimal tree, but with the twist of involving loss matrix. This matrix made false negatives five times more costly than false positives, thus the precitions were adjusted. If the probability of a positive prediction multiplied by 5 exceeded the probability of a negative prediction, the instance was classified as positive. The resulting confusion

```
82:  plot(optimal_tree)
83:  text(optimal_tree, pretty = 0)
84:  #The root node holds the most significance.
85:  #In this case it is poutcome.
86:  #Children to root node is month and job, which is the second most
significant variables
87:
88:  #Task 4
89:  predict_test = predict(optimal_tree, newdata = test, type = "class")
90:  confusion_matrix = table(test$y, predict_test)
91:  print(confusion_matrix)
92:
93:  accuracy = sum(diag(confusion_matrix))/sum(confusion_matrix)
94:  print(accuracy)
95:  #accuracy=0.8951559
96:  presicion = confusion_matrix[2,2]/(confusion_matrix[2,2] +
confusion_matrix[1,2])
97:  recall = confusion_matrix[2,2]/(confusion_matrix[2,2] +
confusion_matrix[2,1])
98:  f1 = 2*presicion*recall/(presicion + recall)
99:  print(f1)
100: #f1=0.2930649
101: #Accuracy looks good,
102: #but we have a lot of false positives and specially a lot of false negatives
103: #so the f1 score gets low
104:
105: #Task 5
106: loss_matrix = matrix(c(0, 5, 1, 0), nrow = 2, byrow = TRUE)
107: predict_test_loss = predict(optimal_tree, newdata = test, type = "vector")
108: #if prob(no) is larger than 5 times prob(yes), predict no
109: predicted_classes = ifelse(predict_test_loss[, "yes"] * loss_matrix[1, 2] <
110:                            predict_test_loss[, "no"] * loss_matrix[2, 1],
"no", "yes")
111: l_confusion_matrix = table(test$y, predicted_classes)
```

Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here.

Googling gives “an F1 score of **0.7 or higher** is often considered good”, so 0.304 indicates a bad predictive power. F1-score is preferred here, since the data is heavily imbalanced.The amount of “no” is a lot higher. Just guessing “no” gives good accuracy, since it's more frequently occurring than “yes”, which the model seems to do.

2.2.5 TASK 5 - Perform a decision tree classification of the test data with loss matrix

$$L = \begin{matrix} \\ Observed \\ \\ \end{matrix} \begin{matrix} Predicted \\ yes \\ no \end{matrix} \begin{pmatrix} 0 & 5 \\ 1 & 0 \end{pmatrix}$$

3.5 Decision tree with loss matrix

A decision tree classification was applied to the test data using the optimal tree, but with the twist of involving loss matrix. This matrix made false negatives five times more costly than false positives, thus the precitions were adjusted. If the probability of a positive prediction multiplied by 5 exceeded the probability of a negative prediction, the instance was classified as positive. The resulting confusion matrix displayed in Table 4 revealed 814 true positives, 11030 true negatives, 949 false positives and 771 false negatives.

```
105: #Task 5
106: loss_matrix = matrix(c(0, 5, 1, 0), nrow = 2, byrow = TRUE)
107: predict_test_loss = predict(optimal_tree, newdata = test, type = "vector")
108: #if prob(no) is larger than 5 times prob(yes), predict no
109: predicted_classes = ifelse(predict_test_loss[, "yes"] * loss_matrix[1, 2] <
110:                            predict_test_loss[, "no"] * loss_matrix[2, 1],
"no", "yes")
111: l_confusion_matrix = table(test$y, predicted_classes)
112: print(l_confusion_matrix)
```

Report the confusion matrix for the test data.

predictions_with_loss

	no	yes
no	11030	949
yes	771	814

Compare the results with the results from step 4 and discuss how the rates has changed

About 5 times as many yes predictions.

F1 Score: 0.4862605 (vs 0.2848752)

Accuracy: 0.8731937 (vs 0.8922884)

Lower accuracy, since predicting “no” is not as safe of a bet as earlier.
Higher F1 Score, because recall rate significantly increased (0.51 vs 0.18) while precision decreased by a lower amount (0.46 vs 0.64). *f1_score <- (2 * precision * recall_rate) / (precision + recall_rate)*

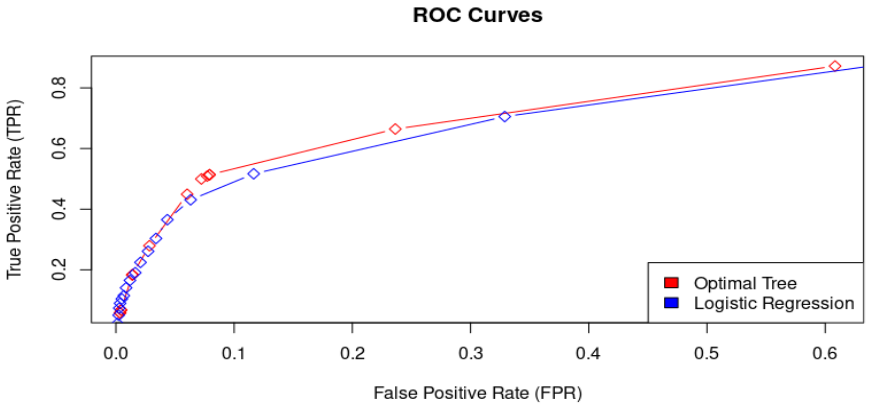
TASK 6

Use the optimal tree and a logistic regression model to classify the test data by using the following principle:

$$\hat{Y} = yes\ if\ p(Y = 'yes'|X) > \pi, otherwise\ \hat{Y} = no$$

where $\pi = 0.05,0.1,0.15, \ldots 0.9,0.95$. Compute the TPR and FPR values for the

Compute the TPR and FPR values for the two models and plot the corresponding ROC curves.



A Precision-Recall (PR) curve would be more benefitting to look at in this case because of the imbalance in the dataset. In imbalanced datasets where the negative class dominates, ROC curved might present an optimistic view by potentially overlooking how well the model performs on classifying a minority class. In contrast, a PR curve measures how well the model is able to classify positive instances accurately, making it more suitable for the dataset used in this assignment.

AUC for Optimal Tree is slightly larger, so therefore performing slightly better. TPR and FPR do not take into account imbalanced classes.

Why precision recall curve could be a better option here?

Precision recall curve would account for imbalance class, therefore still be an option.

2.3 Principal components and implicit regularization

TASK 1

```
############################## Task 1 ##########################################
#Scale the data and manually implement PCA

#Reading the dataset and excluding the target variable ViolentCrimesPerPop for PCA
communities_data = read.csv("communities.csv")
VCPP_excluded = communities_data[,1:100]

#Scaling the data
scaler = preProcess(VCPP_excluded)
data_scaled = predict(scaler, VCPP_excluded)

#Calculating the covariance matrix of the scaled data and computing the eigenvalues
cov_matrix = cov(data_scaled)
eigen_values = eigen(cov_matrix) #The values vector is sorted in decreasing order

#Calculating the cumulative proportion of variance explained by each PC
#Using the match() function to find how many components are needed to obtain 95% of the v
cumulative_variance = cumsum(eigen_values$values)
match(TRUE, cumulative_variance >= 95) #35 PCs
print(cumulative_variance) #printing the cumulative_variance vector to verify the step ab
print(eigen_values$values[1]) #25.01699
print(eigen_values$values[2]) #16.93597

#Plotting the variance of the 10 first componentents in a barplot for visualization purpo
PCA = data.frame(PCA = 1:10, Variance = (eigen_values$values)[1:10])
ggplot(PCA, aes(x = PCA, y = Variance)) + geom_bar(stat = "identity") + labs(title = "Var
```

The first task was to import the data from the “communities.csv” file and scale all variables except the target variable “ViolentCrimesPerPop”. This was done as in the lecture slides. After that we implemented PCA manually by calculating the covariance matrix and the eigenvalues. The eigenvalues express the amount of variance captured by each principal component and the eigenvector indicates the direction of each principal component since the elements in the vector represent the coefficients of the linear combination. In PCA, we often look for the largest eigenvalues and their corresponding eigenvectors because they represent the directions where the data has the most variance.

Report how many components are needed to obtain at least 95% of variance in the data.

To do this the eigenvalues of each principal component were cumulatively added and saved in the vector “cumulative_variance” using the cumsum() function. The match() function was then used with the logical condition in the code to find the index of the element where the sum was >= 95. This calculation showed that **35 components** were needed to obtain at least 95% of the variance in the data. To verify that the match() function worked correctly the cumulative_variance vector was also printed and the number of components were counted manually.

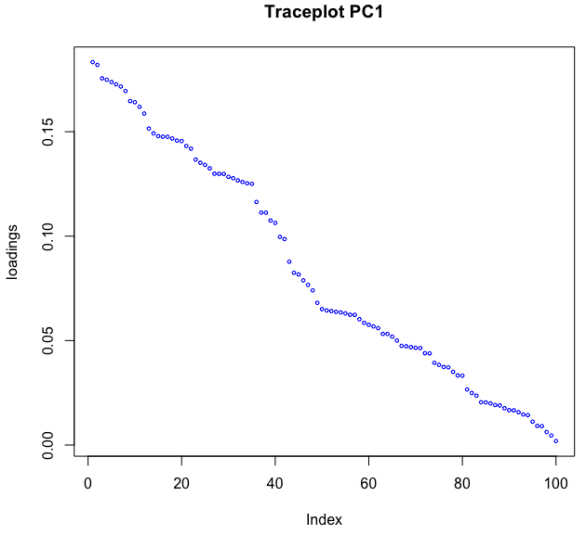
What is the proportion of variation explained by each of the first two principal components?

To answer this we simply had to print the first eigenvalues. This resulted in 25.01699 ~ **25%** for the first principal component and 16.93597 **~17%** for the second principal component. To visualize this in a nice way we also created a bar plot that shows the percentage of variance contained in the 10 first principal components which can be seen below.

TASK 2

```
############################## Task 2 ##########################################
#Repeat PCA using princomp(), make trace plot of the first principal component

PCA_res = princomp(data_scaled)
#Sorting the absolute values of the first PC's loadings in descending order
PC1_sorted = sort(abs(PCA_res$loadings[,1]), decreasing = TRUE)
plot(PC1_sorted, main="Traceplot PC1", col = "blue", cex = 0.5, ylab = "loadings")
print(head(PC1_sorted, 5)) #The 5 features that contribute mostly

#Plotting the PC scores
Violent_Crimes = communities_data$ViolentCrimesPerPop
ggplot(data.frame(PC1 = PCA_res$scores[,1], PC2 = PCA_res$scores[,2]), aes(x=PC1, y=PC2)) + geom_point(aes(co
```

In the second task we repeated the PCA analysis with the princomp() function as instructed. We then made a traceplot by first sorting the absolute values of the loadings for PC1 in decreasing order and then using the plot() function. The traceplot can be seen below.



Do many features have a notable contribution to this component?

The plot clearly shows that many variables have a notable contribution to PC1. This is because many have a high absolute value meaning that they strongly influence PC1.

Report which 5 features contribute mostly to the first principal component.

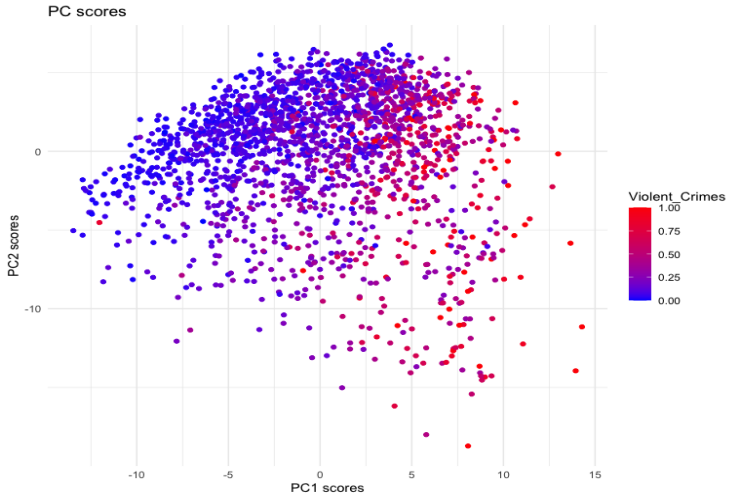
To see this we printed the head of the PC1_sorted vector with 5 elements, the result was the following:

medFamInc	medIncome	PctKids2Par	pctWInvInc	PctPopUnderPov
0.1833080	0.1819830	0.1755423	0.1748683	0.1737978

Comment whether these features have anything in common and whether they may have a logical relationship to the crime level.

These features are economic factors or have a relation to family economy. It is fair to assume that the economy and the cime level have a correlation and that the crime levels are often higher when the income is lower.

Finally we used ggplot2 to plot the PC scores in the coordinate system PC1 and PC2 with a color gradient defined in the ViolentCrimesPerPop variable, which can be seen below.



The plot shows more red dots for high values of PC1 scores and more blue dots for lower values of PC1 scores. This means that violentCrimesPerPop are more frequent for higher values, confirming the correlation mentioned above. When it comes to PC2 the scores seem to be somewhat evenly distributed around 0, meaning that the PC2 scores do not seem to affect violentCrimesPerPop in any significant way. In other words, PC1 has a logical relationship to crime level while PC2 does not.

TASK 3

```
############################## Task 3 ##################################################
#Splitting the data into training and test

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

#Scaling the data
scaler = preProcess(train)
train_scaled = predict(scaler, train)
test_scaled = predict(scaler, test)

#linear regression
fit1=lm(ViolentCrimesPerPop ~ ., data=train_scaled)
summary(fit1)

#Predicting ViolentCrimesPerPop using the linear model
predicted_train = predict(fit1, train_scaled)
predicted_test = predict(fit1,test_scaled)

#Calculate Mean Squared Error for training and test data
mse_train = mean((train_scaled$ViolentCrimesPerPop - predicted_train)^2)
mse_test = mean((test_scaled$ViolentCrimesPerPop - predicted_test)^2)

print(paste("Training MSE: ", mse_train)) #~0.275
print(paste("Test MSE: ", mse_test)) #~0.425
```

In the third task we divided the dataset into training and test (50/50) and scaled it again, this time with the target variable included as per the instruction. Then we created a linear regression model using the lm() function to predict the ViolentCrimesPerPop variable. To evaluate the quality of the model, training and test MSE was calculated which gave the following result.

Training MSE: 0.275207137480974

Test MSE: 0.424801137490899

The test MSE is higher than the training MSE, which is expected, as models tend to perform better on data they have seen compared to new data. The test MSE is notably higher than the training MSE, but not excessively so. This suggests that while the model has learned to predict the training data reasonably well, its predictions are less accurate when applied to the test data. This indicates a slightly overfitted model and while the model shows some generalization capability, there is room for improvement.

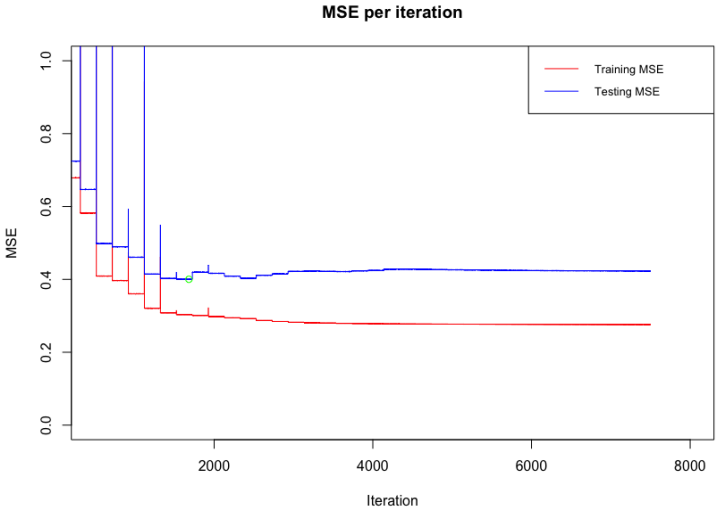
TASK 4

```
############################## Task 4 ########################################
#Some global variable used in calculations below
#Empty vectors for storing MSE values to be plotted
MSE_training = c()
MSE_testing = c()
#scaled training params, VCPP excluded
x_train = as.matrix(train_scaled[,-101])
x_test = as.matrix(test_scaled[,-101])
#Target variable
target_train = train_scaled$ViolentCrimesPerPop
target_test = test_scaled$ViolentCrimesPerPop

#Implementation of cost function
cost_function = function(theta) {
  predictions_train = x_train %*% theta
  predictions_test = x_test %*% theta

  mse_train = mean((target_train - predictions_train)^2)
  mse_test = mean((target_test - predictions_test)^2)
  #Saving the MSE values to the vector
  MSE_training <<- c(MSE_training, mse_train)
  MSE_testing <<- c(MSE_testing, mse_test)
  return(mse_train)
}

#starting at theta = 0
theta = rep(0, ncol(x_train))
#Using the optim() fucntion to optimize the cost
opt_result = optim(par = theta, fn = cost_function, method = "BFGS")
```

In the last task we implemented a cost function for linear regression. One hint was to only store the errors, which is why we created two empty vectors first. The cost function took theta as a parameter and computed the train and test MSE for different theta values. For each iteration these were saved in the empty vectors created in the beginning. Theta was initialized as a vector of 100 zeros (one for each parameter) as per the instruction. Then the optim() function was used to optimize the cost function.

```
#takes the minimum MSE testing value to see which iteration that is optimal
min_test = which.min(MSE_testing)
min_train = which.min(MSE_training)

#printing the optimal training and test MSE
print(MSE_testing[min_test])
print(MSE_training[min_train])
#the index for the iteration numner for optimal test MSE
print(min_test)

#defining start and endpoints for the graph
startpoint = 500
endpoint = 8000
#Plotting the training MSE
plot(MSE_training[startpoint:endpoint], type = "l", col = "red", lwd = 1, ylim = c(0, 1), ›
#Adding the testing MSE line
lines(MSE_testing[startpoint:endpoint], col = "blue", lwd = 1)
#Marking the min test MSE, adding startpoint as offset
points(min_test - startpoint, MSE_testing[min_test], col = "green")
#Adding a legend to the plot
legend("topright", legend = c("Training MSE", "Testing MSE"), col = c("red", "blue"), lwd =
```

The next step was to plot a graph showing the MSE errors for both test and train for every iteration. To do this we first used the which.min() function to get the index (iteration) of the smallest MSE for test and training. We then defined a startpoint at 500 based on the hint in the instruction. The endpoint was derived by plotting the full graph and then choosing a suitable value. After that we used the plot() function to plot both test and training as lines. We also added a point to mark the minimum test MSE and a legend to the top right corner for explanation. Below is the graph that the code resulted in.



We were also asked to comment which iteration number that is optimal according to the early stopping criterion. This was found by printing the “min_test” variable, which resulted in the iteration **2182**. The optimal iteration number is also marked in the graph with the green dot. In the graph it is easy to see that the performance on the test data slightly worsens and flattens out after iteration 2500 approximately. By the definition of the early stopping criterion the training process should be halted at this point which confirms that the iteration 2182 is the best according to the criterion.

Compute the training and test error in the optimal model, compare them with results in step 3 and make conclusions.

This was done by printing the MSE corresponding to the best iteration in the graph for both test and training and gave the following result:

Training MSE: 0.2752213

Test MSE: 0.4002329

From the linear regression model in 3.3 the following MSE:

Training MSE: 0.275207137480974

Test MSE: 0.424801137490899

Here we can see that the training errors are almost identical and that the test errors are close in 3.3 and 3.4. One conclusion that can be drawn from this is that using the optim() function to optimze theta has led to a modest improvement in the model's generalization, as seen by a slightly reduced test MSE from 0.4248 to 0.4002, while maintaining a comparable training MSE.

Computer lab 2 - Code

library(glmnet)

```
library(dplyr)
library(caret)
# Clear global env.
rm(list = ls())
############### Read data and divide it into test and train ##############
set.seed(12345)
df <- read.csv('tecator.csv', header=TRUE)
```

```r
data <- (df)[2:102] #Relevant columns
n = dim(data)[1]
id = sample(1:n, floor(n * 0.5)) # Take a random sample
train_data = data[id, ]
test_data = data[-id, ]
##################################### TASK 1 ####################################
# Model
fit = lm(Fat ~ . ,data = train_data)
summary(fit)
# Predictions
prediction_train = predict(fit, train_data)
prediction_test = predict(fit, test_data)
# Calculating errors, Mean Squared Error
mse_train = mean((train_data$Fat - prediction_train)^2)
mse_test = mean((test_data$Fat - prediction_test)^2)
print(paste("Train MSE: ", mse_train)) # 0.00570911701090834
print(paste("Test MSE: ", mse_test)) # 722.429419336971

##################################### TASK 3 ####################################
## LASSO regression
x = as.matrix(train_data %>% select(-Fat))
y = as.matrix(train_data %>% select(Fat))
lasso = glmnet(x, y, alpha = 1, family = "gaussian")
plot(lasso, xvar = "lambda", label = TRUE, main="Lasso regression model")
##################################### TASK 4 ####################################
## Ridge regression
ridge = glmnet(x, y, alpha = 0, family="gaussian")
plot(ridge, xvar = "lambda", label = TRUE, main="Ridge regression")
##################################### TASK 5 ####################################
cross_validation = cv.glmnet(x, y, alpha=1, family="gaussian")
plot(cross_validation)
opt_lambda = cross_validation$lambda.min # 0.05744535
coef(cross_validation, s = "lambda.min")
summary(cross_validation)
# Channels 52, 51, 41, 40, 16, 15, 14, 13. 8 total
# Cross validation prediction
crossval_predict = predict(cross_validation, newx = x, s = opt_lambda)
plot(test_data$Fat, col="red", ylim = c(0,60), ylab="Fat levels", main = "Test
vs model with optimal lambda")
points(crossval_predict, col = "green")
legend("topright", c("actual", "predicted"), col = c("red", "green"), pch = 1,
cex = 0.8)
-----------------------------------Assignment 2-------------------------------
library(tree)
#### TASK 1 ####
rm(list = ls())
data = read.csv2("bank-full.csv", stringsAsFactors = TRUE)
data = data[, -12]
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.4))
```

```r
train_data=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.3))
validation_data=data[id2,]
id3=setdiff(id1,id2)
test_data=data[id3,]
rm(data, id, id1, id2, id3, n)
#### TASK 2 ####
# Decision Tree with default settings
default_tree <- tree(y ~ ., data = train_data)
plot(default_tree)
# Decision Tree with the smallest allowed node size (7000)
node_size_tree <- tree(y ~ ., data = train_data, control =
tree.control(nrow(train_data), minsize = 7000))
plot(node_size_tree)
# Decision Tree with minimum deviance set to 0.0005
deviance_tree <- tree(y ~ ., data = train_data, control =
tree.control(nrow(train_data), mindev = 0.0005))
plot(deviance_tree)
# Extract misclassification rates
default_missclass = summary(default_tree)$misclass
node_size_missclass = summary(node_size_tree)$misclass
deviance_missclass = summary(deviance_tree)$misclass
# Misclassification rates for training data
train_missclass_rates <- list(
  default = default_missclass[1] / default_missclass[2],
  node_size = node_size_missclass[1] / node_size_missclass[2],
  deviance = deviance_missclass[1] / deviance_missclass[2])
# Predictions on validation set
pred_default <- predict(default_tree, newdata = validation_data, type = "class")
pred_node_size <- predict(node_size_tree, newdata = validation_data, type =
"class")
pred_deviance <- predict(deviance_tree, newdata = validation_data, type =
"class")
# Misclassification rates for validation data
validation_missclass_rates <- data.frame(
  default = mean(pred_default != validation_data$y),
  node_size = mean(pred_node_size != validation_data$y),
  deviance = mean(pred_deviance != validation_data$y)
)
#### TASK 3 ####
# Initialize vectors to store training and validation deviances
trainScore <- rep(0, 50)
validScore <- rep(0, 50)
# Find optimal number of leaves. 2:50 because just one node gives error for
prediciton
for (i in 2:50) {
  # Prune the tree to the specified number of leaves
  pruned_tree <- prune.tree(deviance_tree, best = i)
  # Check if the pruned tree is not a leaf node. i=1 gives error in predict
```

```
  pred_pruned_tree <- predict(pruned_tree, newdata = validation_data, type =
"tree")

  # Calculate deviances for training and validation data
  trainScore[i] <- deviance(pruned_tree)
  validScore[i] <- deviance(pred_pruned_tree)
}
# Visualize with plot
plot(2:50, trainScore[2:50], type = "b", col = "red", ylim = c(7000,
max(trainScore, validScore)),
       main = "Optimal tree depth", ylab = "Deviance", xlab = "Size")
points(2:50, validScore[2:50], type = "b", col = "green")
legend("topright", c("train data", "validation data"), fill = c("red", "green"))
# Optimal number of leaves.
optimal_leaves_train <- which.min(trainScore[-1])
optimal_leaves_valid <- which.min(validScore[-1])
optimal_leaves <- data.frame(train = optimal_leaves_train, valid =
optimal_leaves_valid)
# Optimal tree
optimal_tree <- tree(y ~ ., data = train_data, control =
tree.control(nrow(train_data), mindev = 0.0005, minsize = optimal_leaves_valid))
# Prune the tree
pruned_optimal_tree <- prune.tree(optimal_tree, best = optimal_leaves_valid)
# Display pruned tree structure
plot(pruned_optimal_tree)
text(pruned_optimal_tree, pretty=1)

#### TASK 4 ####
# Predictions on the test set using the optimal model
test_predictions <- predict(pruned_optimal_tree, newdata = test_data, type =
"class")
# Confusion Matrix
confusion_matrix <- table(test_data$y, test_predictions)
# Calculate Accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
# Calculate precision TP/(TP+FP)
precision <- confusion_matrix[2,2]/(confusion_matrix[2,2] +
confusion_matrix[1,2])
# Calculate Recall Rate TP/P
recall_rate <- confusion_matrix[2,2]/sum(confusion_matrix[2,])
# Calculate F1 Score
f1_score <- (2 * precision * recall_rate) / (precision + recall_rate)
# Print Results
cat("Confusion Matrix:\n", confusion_matrix)
cat("\nAccuracy for the optimal model:\n", accuracy)
cat("\nRecall Rate for the optimal model:\n", recall_rate)
cat("\nF1 Score for the optimal model:\n", f1_score)

#### TASK 5 ####
# Decision tree classification with Loss matrix:
loss_matrix <- matrix(c(0, 1, 5, 0), byrow = TRUE, nrow = 2)
```

```r
# Predictions using the pruned optimal tree
probabilities <- predict(pruned_optimal_tree, newdata = test_data)
# loss om den gissar yes när det är no är P(no|x) * 1 enligt matris
# loss om den gissar no när det är yes är P(yes|x) * 5 enligt matris
# Matrix multiplication (p(no|x)*1 p(yes|x)*5)
losses <- probabilities %*% loss_matrix
# Find column index with lowest value along each row (max of negative)
lowest_indices <- max.col(-losses)
# Convert to levels
predictions_with_loss <- levels(test_data$y)[lowest_indices]
# Confusion matrix with loss matrix
confusion_matrix_loss <- table(test_data$y, predictions_with_loss)
confusion_matrix_loss
# Calculate Accuracy
accuracy_loss <- sum(diag(confusion_matrix_loss)) / sum(confusion_matrix_loss)
# Calculate precision TP/(TP+FP)
precision_loss <- confusion_matrix_loss[2,2]/(confusion_matrix_loss[2,2] +
confusion_matrix_loss[1,2])
# Calculate Recall Rate TP/P
recall_rate_loss <- confusion_matrix_loss[2,2]/sum(confusion_matrix_loss[2,])
# Calculate F1 Score
f1_score_loss <- (2 * precision_loss * recall_rate_loss) / (precision_loss +
recall_rate_loss)
# Print Results
cat("Confusion Matrix with Loss Matrix:\n", confusion_matrix_loss)
cat("\nAccuracy for the model with Loss Matrix:\n", accuracy_loss)
cat("\nF1 Score for the model with Loss Matrix:\n", f1_score_loss)
#### TASK 6 ####
# Logistic regression model
logistic_model <- glm(y ~ ., data = train_data, family = "binomial")
# Initialize vectors for TPR and FPR
tpr_tree <- rep()
fpr_tree <- rep()
tpr_logistic <- rep()
fpr_logistic <- rep()
# Computing ROC curves for the optimal tree and logistic regression model
for (pi in seq(from = 0.05, to = 0.95, by = 0.05)) {
  # Optimal tree predictions
  pred_tree <- ifelse(predict(pruned_optimal_tree, newdata = test_data, type =
"vector")[, 2] > pi, "yes", "no")
  cm_tree <- table(pred_tree, test_data$y)
  #pred_tree no     yes
  #          no  TN FN
  #          yes FP TP
  #Sometimes model never predicts "yes", giving no tpr
  if (nrow(cm_tree) >= 2) {
       #TPR = TP / TP + FN
       tpr_tree <- c(tpr_tree, cm_tree[2, 2] / sum(cm_tree[, 2]))
       #FPR =  FP / FP + TN
       fpr_tree <- c(fpr_tree, cm_tree[2, 1] / sum(cm_tree[, 1]))
  }
```

```
  # Logistic regression predictions
  pred_logistic <- ifelse(predict(logistic_model, newdata = test_data, type =
"response") > pi, "yes", "no")
  cm_logistic <- table(pred_logistic, test_data$y)
  if (nrow(cm_logistic) >= 2) {
tpr_logistic <- c(tpr_logistic, cm_logistic[2, 2] / sum(cm_logistic[, 2]))
fpr_logistic <- c(fpr_logistic, cm_logistic[2, 1] / sum(cm_logistic[,1])) }
}
# Plotting ROC curves
plot(fpr_tree, tpr_tree, pch = 5, type = "b", col = "red", main = "ROC Curves",
xlab = "False Positive Rate (FPR)", ylab = "True Positive Rate (TPR)")
legend("bottomright", c("Optimal Tree", "Logistic Regression"), fill = c("red",
"blue"))
-----------------------------------Assignment3------------------------------
#libraries
library(caret)
library(ggplot2)
#Clear global environment
rm(list = ls())
############################## Task 1 ##########################################
#Scale the data and manually implement PCA
#Reading the dataset and excluding the target variable ViolentCrimesPerPop for
PCA
communities_data = read.csv("communities.csv")
VCPP_excluded = communities_data[,1:100]
#Scaling the data
scaler = preProcess(VCPP_excluded)
data_scaled = predict(scaler, VCPP_excluded)
#Calculating the covariance matrix of the scaled data and computing the
eigenvalues
cov_matrix = cov(data_scaled)
eigen_values = eigen(cov_matrix) #The values vector is sorted in decreasing
order
#Calculating the cumulative proportion of variance explained by each PC
#Using the match() function to find how many components are needed to obtain 95%
of the variance
cumulative_variance = cumsum(eigen_values$values)
match(TRUE, cumulative_variance >= 95) #35 PCs
print(cumulative_variance) #printing the cumulative_variance vector to verify
the step above
print(eigen_values$values[1]) #25.01699
print(eigen_values$values[2]) #16.93597
#Plotting the variance of the 10 first componentents in a barplot for
visualization purposes
PCA = data.frame(PCA = 1:10, Variance = (eigen_values$values)[1:10])
ggplot(PCA, aes(x = PCA, y = Variance)) + geom_bar(stat = "identity") +
labs(title = "Variance of Principal Components", x = "Principal Component", y =
"Variance Explained (%)") + theme_minimal()

############################## Task 2 ##########################################
#Repeat PCA using princomp(), make trace plot of the first principal component
```

```r
PCA_res = princomp(data_scaled)
#Sorting the absolute values of the first PC's loadings in descending order
PC1_sorted = sort(abs(PCA_res$loadings[,1]), decreasing = TRUE)
plot(PC1_sorted, main="Traceplot PC1", col = "blue", cex = 0.5, ylab =
"loadings")
print(head(PC1_sorted, 5)) #The 5 features that contribute mostly
#Plotting the PC scores
Violent_Crimes = communities_data$ViolentCrimesPerPop
ggplot(data.frame(PC1 = PCA_res$scores[,1], PC2 = PCA_res$scores[,2]),
aes(x=PC1, y=PC2)) + geom_point(aes(color = Violent_Crimes)) +
scale_color_gradient(low = "blue", high = "red") + ggtitle("PC scores") +
xlab("PC1 scores") +  ylab("PC2 scores") +  theme_minimal()
############################## Task 3 ##########################################
#Splitting the data into training and test
set.seed(12345)
n=nrow(communities_data)
id=sample(1:n, floor(n*0.5))
train=communities_data[id,]
test=communities_data[-id,]
#Scaling the data
scaler = preProcess(train)
train_scaled = predict(scaler, train)
test_scaled = predict(scaler, test)
#linear regression
fit1=lm(ViolentCrimesPerPop ~ ., data=train_scaled)
summary(fit1)
#Predicting ViolentCrimesPerPop using the linear model
predicted_train = predict(fit1, train_scaled)
predicted_test = predict(fit1,test_scaled)
#Calculate Mean Squared Error for training and test data
mse_train = mean((train_scaled$ViolentCrimesPerPop - predicted_train)^2)
mse_test = mean((test_scaled$ViolentCrimesPerPop - predicted_test)^2)
print(paste("Training MSE: ", mse_train)) #~0.275
print(paste("Test MSE: ", mse_test)) #~0.425
############################## Task 4 ##########################################
#Some global variable used in calculations below
#Empty vectors for storing MSE values to be plotted
MSE_training = c()
MSE_testing = c()
#scaled training params, VCPP excluded
x_train = as.matrix(train_scaled[,-101])
x_test = as.matrix(test_scaled[,-101])
#Target variable
target_train = train_scaled$ViolentCrimesPerPop
target_test = test_scaled$ViolentCrimesPerPop
#Implementation of cost function
cost_function = function(theta) {
  predictions_train = x_train %*% theta
  predictions_test = x_test %*% theta
  mse_train = mean((target_train - predictions_train)^2)
  mse_test = mean((target_test - predictions_test)^2)
```

```
  #Saving the MSE values to the vector
  MSE_training <<- c(MSE_training, mse_train)
  MSE_testing <<- c(MSE_testing, mse_test)
  return(mse_train)
}
#starting at theta = 0
theta = rep(0, ncol(x_train))
#Using the optim() fucntion to optimize the cost
opt_result = optim(par = theta, fn = cost_function, method = "BFGS")
#takes the minimum MSE testing value to see which iteration that is optimal
min_test = which.min(MSE_testing)
min_train = which.min(MSE_training)
#printing the optimal training and test MSE
print(MSE_testing[min_test])
print(MSE_training[min_train])
#the index for the iteration numner for optimal test MSE
print(min_test)
#defining start and endpoints for the graph
startpoint = 500
endpoint = 8000
#Plotting the training MSE
plot(MSE_training[startpoint:endpoint], type = "l", col = "red", lwd = 1, ylim =
c(0, 1), xlim = c(startpoint, endpoint), xlab = "Iteration", ylab = "MSE", main
= "MSE per iteration")
#Adding the testing MSE line
lines(MSE_testing[startpoint:endpoint], col = "blue", lwd = 1)
#Marking the min test MSE, adding startpoint as offset
points(min_test - startpoint, MSE_testing[min_test], col = "green")
#Adding a legend to the plot
legend("topright", legend = c("Training MSE", "Testing MSE"), col = c("red",
"blue"), lwd = 1, cex = 0.8)
```

Computer lab 3 - Kernel methods, Support vector machines and neural networks

Kernel Methods

This assignment was about implementing a kernel method to predict the hourly temperatures for a date and place in Sweden. To do this, we first used the provided template to load the data from the two files “stations.csv” and “temps50k.csv” and merge them by station number. Then, we chose an address in Vallastaden, Linköping, as the place for the temperature prediction and assigned “a" and “b” the coordinates of the address. Following that, we selected the date to predict to be 2010-12-24

```
#libraries
library(geosphere)

rm(list = ls())

set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

# The point to predict (adress in vallastaden, Linköping)
a <- 58.393379 #latitud
b <- 15.584957 #longitud

date <- "2010-12-24" # The date to predict
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")

#Filter out times and dates that should not be included
specified_date = as.POSIXct(date, format = "%Y-%m-%d")
st$date = as.POSIXct(st$date, format = "%Y-%m-%d")
st$time = format(strptime(st$time, format = "%H:%M:%S"), "%H:%M:%S")
st = st[st$date <= specified_date & st$time %in% times,]
```

(Christmas Eve) and specified a times vector with times between 04.00 and 24.00 in two-hour intervals, which was the different times to predict the temperature for. We then converted the dates and times in the 'st' dataset to be in a comparable format. Following that, we filtered out the dates after 2010-12-24 and the times that were not in our times vector. We did this in order to make a correct and unbiased prediction as per the instruction.

```
#kernel function
gaussianKernel = function(x_diff, h) {
  return(exp(-(x_diff / h)^2))
}

#function to calculate the distance between the point to predict and the station coordinates
calc_distance_diff = function() {
  stations_coordinates = matrix(c(st$longitude, st$latitude), ncol = 2)
  POI = matrix(c(b,a), ncol = 2)
  dist_diff = distHaversine(stations_coordinates, POI)
  return(dist_diff)
}

#Function to calculate the difference between the day to predict and the day of the measurement
calc_date_diff = function() {
  day_diff = as.numeric(difftime(date, st$date, units = "days")) %% 365
  return(day_diff)
}

#Function to calculate the difference between hours to predict and the hours the measurements were taken
calc_hour_diff = function(hour) {
  hour_diff = abs(as.numeric(difftime(strptime(times[hour], format = "%H:%M:%S"),strptime(cbind(st$time), forma
  return(hour_diff)
}
```

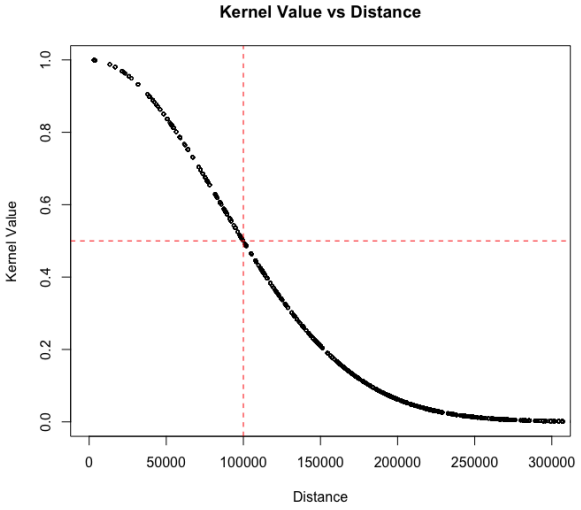
The next step was to implement the gaussian kernel formula as specified in the lecture slides. We then created three functions:

- **calc_distance_diff:** calculates the difference in distance between the point of interest and the stations using the distHaversine() function from the geosphere package.
- **calc_date_diff:** calculates the difference between the day to predict and the day of the measurement. It uses modulo 365 to account for the annual cyclical nature of the dates.
- **calc_hour_diff:** calculates the difference between hours to predict and the hours when the measurements were taken.

```
#function to plot kernel values against distance/days/hours
plot_kernel_vs_distance = function(h_value, against, xlim_value, v_value) {
  kernel_value = gaussianKernel(against, h_value)
  plot(against, kernel_value, type = "p", main = "Kernel Value vs Distance", xlab = "Distance", ylab = "Kernel Value", col = "bla
  abline(h = 0.5, col = 'red', lty = 2)
  abline(v = v_value, col = 'red', lty = 2)
}

#some h_values to test for distance
h_values = c(10000, 20000, 50000, 80000, 100000, 120000)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_distance_diff(), 300000, 100000)
}

#some h_values to test for days
h_values = c(2, 5, 7, 8, 9, 10)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_date_diff(), 20, 7)
}

#some h_values to test for hours
h_values = c(2, 4, 6, 8, 10, 12)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_hour_diff(), 15, 5)
}

#h_values final
h_distance = 120000
h_date = 8
h_time = 6
```

We were instructed to choose appropriate smoothing coefficients for each of the three kernels. To do this we created a function that plotted the kernel value against the distance/day/hour for different h_values. We then created vectors with some suitable h_vaules for each of the three kernels. We

looped through this vector to produce six plots for each kernel. This resulted in the final smoothing coefficients being 120 000 for distance, 8 for date and 6 for time. These smoothing coefficients ensured large kernel values for closer points and small values for distant points. Below is our reasoning for choosing the different smoothing coefficients for each of the three kernels.
Physical distance - When choosing a smoothing coefficient for the distance we reasoned that a measurement 100 km away from the point of interest should be 1/2 as important as a measurement on the exact location. As can be seen in the graph, the kernel value is 0.5 when the distance is 100 km for h_value 120 000, which is why we chose it.

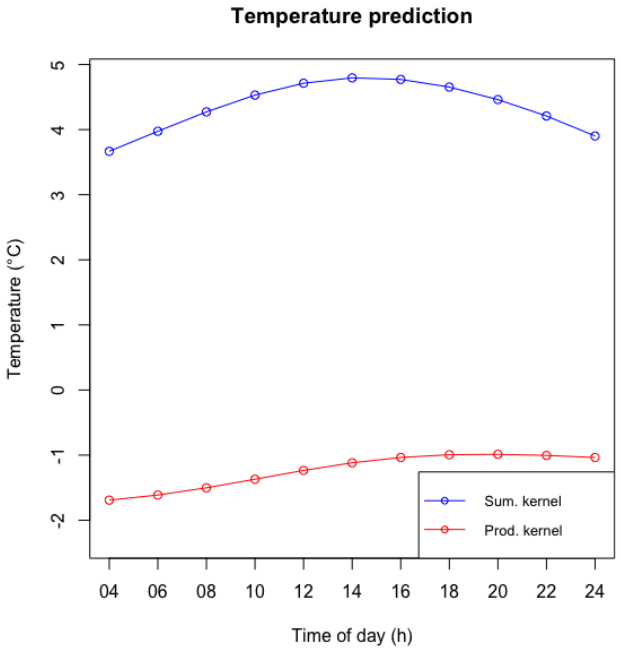


Days difference - When choosing the smoothing coefficient for the difference in days we reasoned that a measurement that differs a week (7 days) should be 1/2 as important as a measurement made the same day. This resulted in 8 as the h-value. As can be seen in the graph, the kernel value is 0.5 approximately when 7 days have passed. (graph removed for space)
Hours distance - When choosing a smoothing coefficient for the distance in hours we reasoned that a measurement that differs 5 hours should be 1/2 as important as a measurement made the same hour. This resulted in 6 as the h-value. This can be seen in the graph where the kernel value is 0.5 when 5 hours have passed.(graph removed for space)

```
#date & distance Kernels
date_kernel = gaussianKernel(calc_date_diff(), h_date)
dist_kernel = gaussianKernel(calc_distance_diff(), h_distance)

#temperature vectors
temp_sum = vector(length=length(times))
temp_prod = vector(length=length(times))

#looping through the hours in the times vector
for(hour in seq_along(times)) {
  hour_diff = calc_hour_diff(hour)
  hours_kernel = gaussianKernel(hour_diff, h_time)

  #Sum and product of the kernels
  kernel_sum = dist_kernel + date_kernel + hours_kernel
  kernel_prod = dist_kernel * date_kernel * hours_kernel

  #Add the temperatures to the temperature vectors
  temp_sum[hour] = sum(kernel_sum %*% st$air_temperature) / sum(kernel_sum)
  temp_prod[hour] = sum(kernel_prod %*% st$air_temperature) / sum(kernel_prod)
}

#Plotting temperatures for both sum and prod kernels
plot(temp_sum, type="o", main = "Temperature prediction", xlab = "Time of day (h)", ylab = "Temperatur
lines(temp_prod, type="o", col = "red")
axis(1, at = seq_along(times),  labels=substring(times, 1, 2))
legend("bottomright", c("Sum. kernel", "Prod. kernel"), col = c("blue", "red"),  pch=1, lty=1, box.lty
```

When we had chosen the h_values we could create the date- and distance kernels. After this we created two empty vectors to store the temperatures for the summation and multiplication of the kernels. Following that, we looped through the elements in the “times” vector and calculated an hour kernel for each hour in the vector. For each time we summarized and multiplied the kernels and then stored the predicted temperatures in the temp_sum and temp_prod vectors. Finally we plotted the summarized- and multiplied vectors in a graph showing how the predicted temperature varied from 04:00 to 24:00.

Analysis of the plot



The graph shows that the sum kernel method tends to forecast higher temperatures compared to the product kernel method. This difference can be explained by the way each method processes the input parameters. In the sum kernel method, a high value in one parameter can compensate for a lower value in another since the gaussian kernels are added. This can lead to higher overall weights and thus higher predicted temperatures.

In contrast, the product kernel method multiplies the values of the kernels together, which means that a low value in any single parameter can drastically reduce the overall kernel value, leading to lower weights and lower predicted temperatures. For example, a temperature reading taken at the correct location and time, but during a different season (large difference in days), still significantly influences the prediction in the sum method. However, in the product kernel method, the result is more cautious. If any single parameter (such as the season) is mismatched, it substantially reduces the influence of that reading on the final prediction. This makes the product kernel method more conservative since it only gives significant weight to temperature readings when all parameters closely align with the target conditions.

2.2 Support vector machines

2. Support Vector Machines

The code in the file `Lab3Block1_2021_SVMs_St.R` performs SVM model selection to classify the `spam` dataset. To do so, the code uses the function `ksvm` from the R package `kernlab`, which also includes the `spam` dataset. All the SVM models to select from use the radial basis function kernel (also known as Gaussian) with a width of 0.05. The C parameter varies between the models. Run the code in the file `Lab3Block1_2021_SVMs_St.R` and answer the following questions.

TASK 1 - Which filter do you return to the user ? filter0, filter1, filter2 or filter3?

The code trains four different support vector machine filters, each trained on different subsets of the dataset. For example, filter1 is trained on the training set. Regarding which filter should be recommended to the user, we should choose filter3. The reasoning for choosing this one is because it is trained on the entire dataset, allowing it to learn from a wider range of data. This could lead to better generalization. The other filters only use subset, and will therefore not be as efficient models. Since it uses the entire dataset, it could also be trained on any potential outlier data points in the dataset.

TASK 2 - What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why?

For this, we choose err2 as the generalization error returned to the user. This is because it has the largest coverage of the dataset while having separation between the predictions and training. For example, filter3 uses the entire dataset and then calculates the generalization error (err3) on the test data. Here, it has already been trained on this data, leading to a misleading answer for the error. The data for the error is not unseen and therefore misleading. Err2 is chosen since it has the largest coverage of training the model and separation, the error is calculated from test and the model is trained on trainva:

trainva <- spam[1:3800,]

test <- spam[3801:4601,]

TASK 3 - Instructions below

Once a SVM has been fitted to the training data, a new point is essentially classified according to the sign of a linear combination of the kernel function values between the support vectors and the new point. You are asked to implement this linear combination for `filter3`. You should make use of the functions `alphaindex`, `coef` and `b` that return the indexes of the support vectors, the linear coefficients for the support vectors, and the **negative** intercept of the linear combination. See the help file of the `kernlab` package for more information. You can check if your results are correct by comparing them with the output of the function `predict` where you set `type = "decision"`. Do so for the first 10 points in the `spam` dataset. Feel free to use the template provided in the `Lab3Block1_2021_SVMs_St.R` file.

The implementation of Task 3 is done in the following way:

```
sv <- alphaindex(filter3)[[1]]
support.vectors <- spam[sv, -58]
co <- coef(filter3)[[1]]
inte <- - b(filter3)

#  Smaller sigma value -> narrower Gaussian function
kernel.function <- rbfdot(0.05)

k <- NULL
dot.products <- NULL

# Predictions for the first 10 points
for (i in 1:10) {

  k2 <- NULL

  for (j in seq_along(sv)) {
    k2 <- unlist(support.vectors[j, ])
    sample <- unlist(spam[i, -58])
    # Add the dot product to the existing ones
    dot.products <- c(dot.products, kernel.function(sample, k2))
  }
  # Start and end index
  start <- 1 + length(sv) * (i - 1)
  end <- length(sv) * i

  # Calculate predictions and add them
  prediction <- co %*% dot.products[start:end] + inte
  k <- c(k, prediction)
}

k
predict(filter3, spam[1:10, -58], type = "decision")
```

2.3 Neural Networks

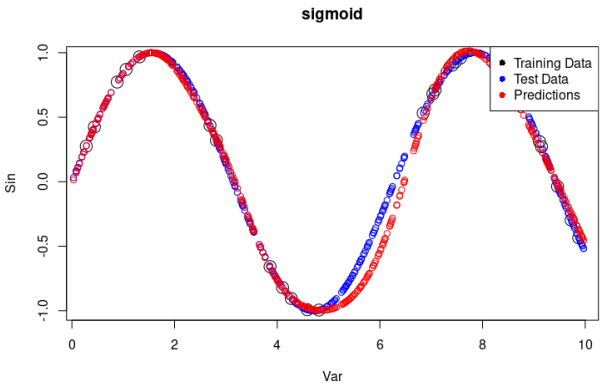
```
h2 <- function(x) max(0,x)
winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, startweights = wini

h3 <- function(x) log(1+exp(x))
winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, startweights = wini
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)

# Additional comments.

set.seed(1234)

Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test

set.seed(123)

h2 <- function(z) ifelse(z>=0,z,0)
nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, startweights = NULL
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)

# The plot above shows that we can get a good fit of the data in the lab using t
```

```
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test

set.seed(123)

h2 <- function(z) ifelse(z>=0,z,0)
nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, startweights = NULL, act.fct = h2)
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)

# The plot above shows that we can get a good fit of the data in the lab using the ReLU activation function.
# The rectifier linear unit (ReLU) is arguably the most common activation function today.
# It gives an error if implemented as h(z)=max(0,z) because it is not differentiable at 0.
# Implementing it as h(z)=ifelse(a>=0,a,0) seems to work fine with the neuralnet package.
# When the ReLU gives 0 value to a hidden unit, we say that the unit is inactive.
# Then, the input selects which hidden units are active, and the output is a linear function of the active units.
# Therefore, when using ReLU, a NN defines a piece-wise linear mapping from input to output.
# Of course, any function can be approximated by a piece-wise linear mapping arbitrarily well.
# For instance, the piece-wise linear mapping corresponding to the exercise above is the following.

plot(te[,1],predict(nn,te))
```

TASK 1 - instructions below

(1) Train a neural network to learn the trigonometric sine function. To do so, sample 500 points uniformly at random in the interval $[0, 10]$. Apply the sine function to each point. The resulting value pairs are the data points available to you. Use 25 of the 500 points for training and the rest for test. Use **one hidden layer with 10 hidden units**. You do not need to apply early stopping. Plot the training and test data, and the predictions of the learned NN on the test data. You should get good results. Comment your results.

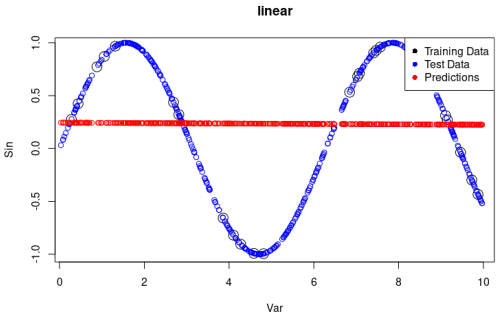
Plot the training and test data, and the predictions of the learned NN on the test data. Comment your results.

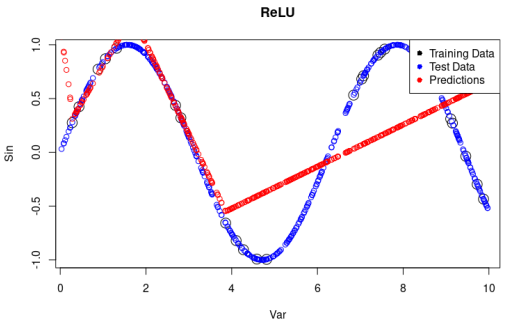


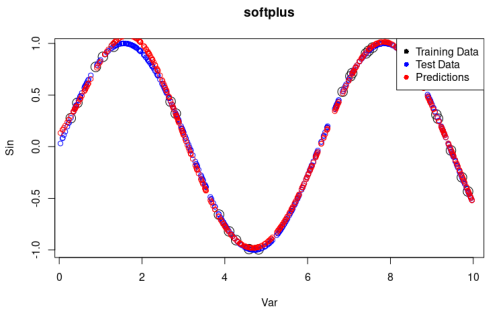
Struggles between where it lacks training data, otherwise looks good.

2.3.2 TASK 2 - instructions below

(2) In question (1), you used the default logistic (a.k.a. sigmoid) activation function, i.e. `act.fct = "logistic"`. Repeat question (1) with the following custom activation functions: $h_1(x) = x$, $h_2(x) = \max\{0, x\}$ and $h_3(x) = \ln(1 + \exp x)$ (a.k.a. linear, ReLU and softplus). See the help file of the `neuralnet` package to learn how to use custom activation functions. Plot and comment your results.

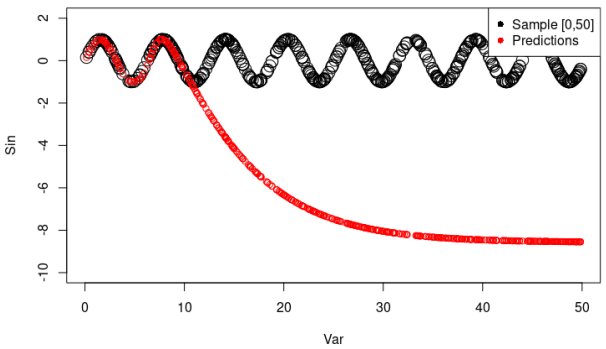






linear: Having a linear activation function essentially means that our neural network is a linear regression, which is unable to capture the non-linear pattern of the sinus function.
ReLU: ReLU is kind of a linear activation function (0 or x), so it would require a lot more lines than the 4 generated in this case to capture the sinus function.
softplus: performs very well where there is no training data when Var is around 5, but performs worse than the others when Var is around 2.

2.3.3 TASK 3 - sample 500 points uniformly at random in the interval [0,50]. apply sine function at each point. Use NN from Q1 to predict sine value of these 500 points. Plot and comment your results.



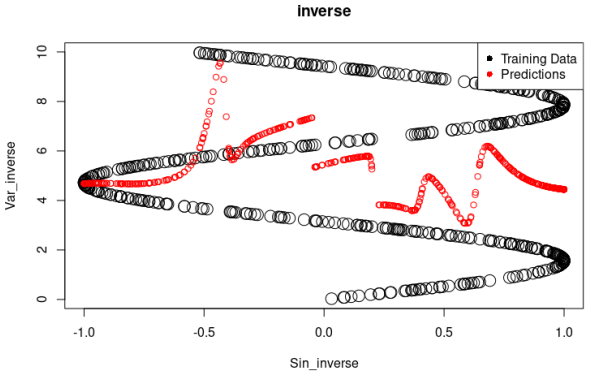
The neuralnet has never seen any values of var greater than 10 earlier, and thus has no clue what to output.

2.3.3 TASK 4 - In question (3), the predictions seem to converge to some value. Explain why this happens. To answer this question, you may need to get access to the weights of the NN learned. You can do it by running nn or nn$weights where nn is the NN learned.

It seems to converge to -8. For greater and greater inputs X, the hidden layer nodes will converge to either 0 or 1. The output of the hidden layer to the final node is therefore a constant, **C**. sigmoid(**C**) + 0.79935 = -8

2.3.3 TASK 5 - instructions below

(5) Sample 500 points uniformly at random in the interval $[0, 10]$, and apply the sine function to each point. Use all these points as training points for learning a NN that tries to predict x **from** $sin(x)$, i.e. unlike before when the goal was to predict $sin(x)$ from x. Use the learned NN to predict the **training data**. You should get bad results. Plot and comment your results. **Help**: Some people get a convergence error in this question. It can be solved by stopping the training before reaching convergence by setting `threshold = 0.1`.



One input may have multiple outputs. For example, the output can be either x=0 or x=pi, for input sin(x) = 0. It is therefore impossible for the model to know if the input sin(x) = 0 should output pi or 0.

Computer lab 3 - Code

```
—----------------------------Assignment 1—-----------------------------------
#libraries
library(geosphere)
rm(list = ls())
set.seed(1234567890)
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
# The point to predict (adress in vallastaden, Linköping)
a <- 58.393379 #latitud
b <- 15.584957 #longitud
date <- "2010-12-24" # The date to predict
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")
#Filter out times and dates that should not be included
specified_date = as.POSIXct(date, format = "%Y-%m-%d")
st$date = as.POSIXct(st$date, format = "%Y-%m-%d")
st$time = format(strptime(st$time, format = "%H:%M:%S"), "%H:%M:%S")
st = st[st$date <= specified_date & st$time %in% times,]
#kernel function
gaussianKernel = function(x_diff, h) {
```

```r
  return(exp(-(x_diff / h)^2))
}
#function to calculate the distance between the point to predict and the station
coordinates
calc_distance_diff = function() {
  stations_coordinates = matrix(c(st$longitude, st$latitude), ncol = 2)
  POI = matrix(c(b,a), ncol = 2)
  dist_diff = distHaversine(stations_coordinates, POI)
  return(dist_diff)
}
#Function to calculate the difference between the day to predict and the day of
the measurement
calc_date_diff = function() {
  day_diff = as.numeric(difftime(date, st$date, units = "days")) %% 365
  return(day_diff)
}
#Function to calculate the difference between hours to predict and the hours the
measurements were taken
calc_hour_diff = function(hour) {
  hour_diff = abs(as.numeric(difftime(strptime(times[hour], format =
"%H:%M:%S"),strptime(cbind(st$time), format = "%H:%M:%S"), units = "hours")))
  return(hour_diff)
}
#function to plot kernel values against distance/days/hours
plot_kernel_vs_distance = function(h_value, against, xlim_value, v_value) {
  kernel_value = gaussianKernel(against, h_value)
  plot(against, kernel_value, type = "p", main = "Kernel Value vs Distance",
xlab = "Distance", ylab = "Kernel Value", col = "black", cex = 0.5, xlim
=c(0,xlim_value))
  abline(h = 0.5, col = 'red', lty = 2)
  abline(v = v_value, col = 'red', lty = 2)
}
#some h_values to test for distance
h_values = c(10000, 20000, 50000, 80000, 100000, 120000)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_distance_diff(), 300000, 100000)
}
#some h_values to test for days
h_values = c(2, 5, 7, 8, 9, 10)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_date_diff(), 20, 7)
}
#some h_values to test for hours
h_values = c(2, 4, 6, 8, 10, 12)
for(h_value in h_values) {
  plot_kernel_vs_distance(h_value, calc_hour_diff(), 15, 5)
}
#h_values final
h_distance = 120000
h_date = 8
h_time = 6
```

```r
#date & distance Kernels
date_kernel = gaussianKernel(calc_date_diff(), h_date)
dist_kernel = gaussianKernel(calc_distance_diff(), h_distance)
#temperature vectors
temp_sum = vector(length=length(times))
temp_prod = vector(length=length(times))
#looping through the hours in the times vector
for(hour in seq_along(times)) {
  hour_diff = calc_hour_diff(hour)
  hours_kernel = gaussianKernel(hour_diff, h_time)

  #Sum and product of the kernels
  kernel_sum = dist_kernel + date_kernel + hours_kernel
  kernel_prod = dist_kernel * date_kernel * hours_kernel
  #Add the temperatures to the temperature vectors
  temp_sum[hour] = sum(kernel_sum %*% st$air_temperature) / sum(kernel_sum)
  temp_prod[hour] = sum(kernel_prod %*% st$air_temperature) / sum(kernel_prod)
}
#Plotting temperatures for both sum and prod kernels
plot(temp_sum, type="o", main = "Temperature prediction", xlab = "Time of day
(h)", ylab = "Temperature (°C)", col = "blue", ylim = c(-4, 6), xaxt="n")
lines(temp_prod, type="o", col = "red")
axis(1, at = seq_along(times),  labels=substring(times, 1, 2))
legend("bottomright", c("Sum. kernel", "Prod. kernel"), col = c("blue", "red"),
pch=1, lty=1, box.lty=1, cex=0.8)
```

----------Assignment 2 ----------

```r
library(kernlab)
set.seed(1234567890)
data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58])
train <- spam[1:3000, ]
val <- spam[3001:3800, ]
trainva <- spam[1:3800, ]
test <- spam[3801:4601, ]
by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){
  filter <-
ksvm(type~.,data=train,kernel="rbfdot",kpar=list(sigma=0.05),C=i,scaled=FALSE)
  mailtype <- predict(filter,val[,-58])
  t <- table(mailtype,val[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}
filter0 <-
ksvm(type~.,data=train,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)
*by,scaled=FALSE)
mailtype <- predict(filter0,val[,-58])
```

```
t <- table(mailtype,val[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)

filter1 <-
ksvm(type~.,data=train,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)
*by,scaled=FALSE)
mailtype <- predict(filter1,test[,-58])
t <- table(mailtype,test[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)

filter2 <-
ksvm(type~.,data=trainva,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_v
a)*by,scaled=FALSE)
mailtype <- predict(filter2,test[,-58])
t <- table(mailtype,test[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)

filter3 <-
ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*
by,scaled=FALSE)
mailtype <- predict(filter3,test[,-58])
t <- table(mailtype,test[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)

# Questions
# 1. Which filter do we return to the user ? filter0, filter1, filter2 or
filter3? Why?
# We return filter3 since it uses all of the data in the model.

# 2. What is the estimate of the generalization error of the filter returned to
the user? err0, err1, err2 or err3? Why?
# The error that should be returned to the user should be error2. This model
uses trainva as training
# and test for the predicitons. These two are separated. In filter3 for example
the data and prediction
# is interconnected, which is not optimal.

# 3. Implementation of SVM predictions.
sv <- alphaindex(filter3)[[1]]
support.vectors <- spam[sv, -58]
co <- coef(filter3)[[1]]
inte <- - b(filter3)
#  Smaller sigma value -> narrower Gaussian function
kernel.function <- rbfdot(0.05)
k <- NULL
dot.products <- NULL
# Predictions for the first 10 points
for (i in 1:10) {
  k2 <- NULL
  for (j in seq_along(sv)) {
    k2 <- unlist(support.vectors[j, ])
```

```
    sample <- unlist(spam[i, -58])
    # Add the dot product to the existing ones
    dot.products <- c(dot.products, kernel.function(sample, k2))
  }
  # Start and end index
  start <- 1 + length(sv) * (i - 1)
  end <- length(sv) * i
  # Calculate predictions and add them
  prediction <- co %*% dot.products[start:end] + inte
  k <- c(k, prediction)
}
predict(filter3, spam[1:10, -58], type = "decision")
```

——————————————————————————Assignment 3——————————————————————————

```
library(neuralnet)
### TASK 1 ###
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))

tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(10, -1, 1)
nn <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights = winit)
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, main="sigmoid", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)
############################## TASK 2 ######################################
# Custom activation function: h1(x) = x
linear <- function(x) x
nn_linear <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights = winit,
act.fct = linear)
# Plot same way as earlier
plot(tr, main = "linear", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_linear,te), col="red", cex=1)
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)
# Custom activation function: h2(x) = max{0, x}
ReLU <- function(x) ifelse(x>0, x, 0)
nn_ReLU <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights = winit,
act.fct = ReLU)
# Plot same way as earlier
plot(tr, main = "ReLU", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_ReLU,te), col="red", cex=1)
```

```
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)
# Custom activation function: h3(x) = ln(1 + exp x)
softplus <- function(x) log(1 + exp(x))
nn_softplus <- neuralnet(Sin ~ Var, data = tr, hidden = 10, startweights =
winit, act.fct = softplus)
# Plot same way as earlier
plot(tr, main = "softplus", cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn_softplus,te), col="red", cex=1)
legend("topright", legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"), pch = 16)
############################TASK 3 ###################################
set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata_50 <- data.frame(Var, Sin=sin(Var))
plot(mydata_50, cex = 2, ylim = c(-10, 2))
points(mydata_50[, 1], predict(nn, mydata_50), col = "red", cex = 1)
legend("topright", legend = c("Training", "Predictions"),
       col = c("black", "red"), pch = 16)
########################### TASK 4 ###################################
plot(nn)
nn$weights
########################### TASK 5 ###################################
set.seed(1234567890)
Var_inverse <- runif(500, 0, 10)
mydata_inverse <- data.frame(Sin_inverse=sin(Var_inverse), Var_inverse)
nn_inverse <- neuralnet(Var_inverse ~ Sin_inverse, data = mydata_inverse, hidden
= 10, threshold = 0.1)

plot(mydata_inverse, main="inverse", cex=2)
points(mydata_inverse[, 1],predict(nn_inverse, mydata_inverse), col="red",
cex=1)
legend("topright", legend = c("Training Data", "Predictions"),
       col = c("black", "red"), pch = 16)
```

Exam 2022-01-14

File **adult.csv** shown data collected in a population census in 1994. The following metrics are (**Removed by me**)

1. Read data into R with option stringsAsFactors=T and divide data randomly (60/20/20) into training, validation and test sets. Use the hold-out principle to first grow a decision tree with default settings and with Income level as target and all other variables except Native Country as inputs. Then use this tree to compute decision trees with various number of leaves and select the optimal tree according to the deviance criterion. Provide the plot of the dependence of the training and validation errors on the number of leaves and interpret this plot from the perspective of bias-variance tradeoff. Report the optimal tree and make some interpretations. Why training a logistic regression model with these features and target is not possible directly? **(4p)**
2. Use the optimal tree computed in step 1 to predict the target probabilities for the test data and estimate predictions for the decision thresholds $\pi = 0.1, 0.2, \ldots, 0.9$ in the rule $\hat{Y} = '>50K' \; if \; p(Y = '>50K'|X) > \pi, otherwise \; \hat{Y} = '\leq 50K'$. Use these predictions to report a table showing test accuracies and test F1 scores for various π. Which threshold is the best to use for these data and why? **(3p)**
3. Assume now that hours-per-week is a target variable and x_1=age, x_2=Capital gain, x_3=Capital loss are features. Use cross-validation to estimate a LASSO model with this target and these features from the training data. Report the cross-validation error plot, the optimal penalty factor value and how many features are selected by the model. Report also an equation of the estimated LASSO model. Does the cross-validation plot suggest that having 0 features is significantly better than having 3 features? **(3p)**

Assignment 2 (10p) KERNEL MODELS – 6 POINTS

In the course, you have learned about kernel models for classification and regression. Kernel models can also be used for density estimation, i.e. to model a probability distribution or density function p(x_*). In particular,

$$p(x_*) = \frac{1}{n}\sum_{i=1}^{n} k(\frac{x_* - x_i}{h})$$

where the kernel function k() must integrate to 1. To ensure this, you will hereinafter consider k() to be the density function of a Gaussian distribution with mean equal to 0 and standard deviation equal to 1.

You can get it by using the command `dnorm` in R.Run the code below to produce some learning data, which consist of 1000 samples from class 1 and 1000 samples from class 2. These points are stored in the variables data_class1 and data_class2. Implement the kernel model presented above to estimate the density function of the data sampled from class 1. Do the same for class 2. Use only 800 samples from class 1 and 800 samples from class 2. Once you have kernel models for the class conditional density functions p(x_*|class=1) and p (x_*|class=2), you can use them to produce posterior class probabilities p(class|x_*) via Bayes theorem. Specifically,

p(class=1|x_*) = p(x_*|class=1) p(class=1) / [p(x_*|class=1) p(class=1) + p(x_*|class=2) p(class=2)]

Use these probabilities to compute the correct classification rate on 200 samples that you did not use before, 100 from class 1 and 100 from class 2. Use this classification rate to select the kernel width h from among the values 0.1, 0.2, …, 4.9, 5. Finally, use the 200 samples that you have not used so far to estimate the generalization error of the kernel model selected.

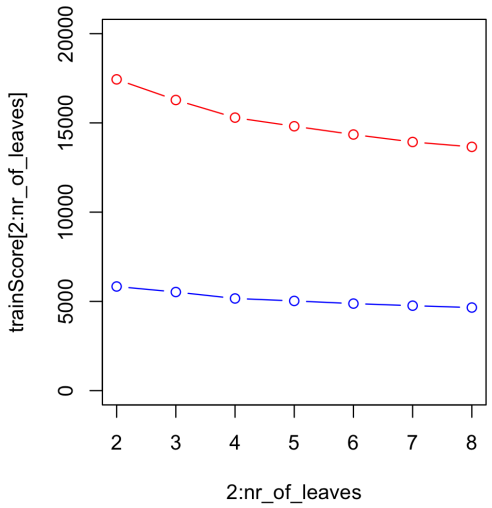
In summary, you should use 1600 samples to build kernel models of the class conditional density functions that you should convert into a probabilistic classifier via Bayes theorem. To select the kernel width, you should use 200 samples as validation set. Finally, you should use 200 samples to estimate the generalization error of the model selected.

NEURAL NETWORKS – 4 POINTS

Train a neural network to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting pairs are the data available to you. Use 25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent. That is, you should use the validation set to detect when to stop the gradient descent and so avoid overfitting (you can read more about early stopping in page 95 of the course textbook). Stop the gradient descent when the partial derivatives of the error function are

below a given threshold value. Check the argument `threshold` in the documentation of the R package `neuralnet`. Consider threshold values i/1000 with i = 1,…, 10. Initialize the weights of the neural network to random values in the interval [−1,1]. Use a neural network with a single hidden layer of 10 units. **Use the default values for the arguments not mentioned here**. Choose the most appropriate value for the threshold. **Motivate your choice.** Provide the final neural network learned with the chosen threshold. Feel free to use the following template.

Assignment 1
Task 1.1



The optimal number of nodes is 8 since the validation error continues to decrease as has not risen/flattend out. Therefore 8 leaves strikes a good balance between complexity and simplicity.

the following data was given from the just running opt_tree
1) root 19536 21610.00 <=50K (0.758344 0.241656)
2) C8: Not-in-family, Other-relative, Own-child, Unmarried 10666 5224.00 <=50K
4) C9 < 7073.5 10479 4190.00 <=50K (0.949518 0.050482)
8) C4: 10th, 11th, 12th, 1st-4th, 5th-6th, 7th-8th, 9th, Assoc-acdm, Assoc-voc, HS-grad, Preschool, Some-college 8431 2031.00 <=50K (0.974024 0.025976) *
9) C4: Bachelors, Doctorate, Masters, Prof-school 2048 1741.00 <=50K (0.848633 0.151367)
5) C9 > 7073.5 187 46.08 >50K (0.026738 0.973262) *
3) C8: Husband, Wife 8870 12210.00 <=50K (0.547914 0.452086)
6) C7: ?, Adm-clerical, Craft-repair, Farming-fishing, Handlers-cleaners, Machine-op-inspct, Other-service, Priv-house-serv, Protective-serv, Transport-moving 4947 5988.00 <=50K
12) C9 < 5095.5 4736 5458.00 <=50K (0.736909 0.263091) *
13) C9 > 5095.5 211 47.30 >50K (0.023697 0.976303) *
7) C7: Armed-Forces, Exec-managerial, Prof-specialty, Sales, Tech-support 3923 5070.00 >50K
14) C9 < 5095.5 3412 4592.00 >50K (0.399766 0.600234)
28) C4: 10th, 11th, 12th, 1st-4th, 5th-6th, 7th-8th, 9th, Assoc-acdm, Assoc-voc, HS-grad, Some-college 1614 2224.00 <=50K (0.545229 0.454771) *
29) C4: Bachelors, Doctorate, Masters, Prof-school 1798 2094.00 >50K (0.269188 0.730812)
15) C9 > 5095.5 511 14.47 >50K (0.001957 0.998043) *

Different interpretations can be done.

1) People who luckily sell their assets have high income
2) Married people with good education have high income
3) Married with Armed-Forces, Exec-managerial, Prof-specialty, Sales, Tech-support high income

Code:

```
library(tree)
rm(list = ls())
data = read.csv("adult.csv", stringsAsFactors = T)
n=dim(data)[1]
set.seed(12345)
```

```
id=sample(1:n, floor(n*0.6))
train=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.2))
valid=data[id2,]
id3=setdiff(id1,id2)
test=data[id3,]

default_tree = tree(C13~.-C12, data=train)
summary(default_tree)
plot(default_tree)
nr_of_leaves = 8
trainScore=rep(0,nr_of_leaves)
testScore=rep(0,nr_of_leaves)
for(i in 2:nr_of_leaves) {
  prunedTree=prune.tree(default_tree,best=i)
  pred=predict(prunedTree, newdata=valid,
               type="tree")
  trainScore[i]=deviance(prunedTree)
  testScore[i]=deviance(pred)
}
plot(2:nr_of_leaves, trainScore[2:nr_of_leaves], type="b", col="red",
     ylim=c(0,20000))
points(2:nr_of_leaves, testScore[2:nr_of_leaves], type="b", col="blue")
#first one is 0 so remove that column then add 1 to correct index
opt_leaves = which.min(testScore[-1])+1
opt_tree = prune.tree(default_tree, best = opt_leaves)
plot(opt_tree)
text(opt_tree)
opt_tree
```

Task 1.2

```
     [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
pis 0.1000000 0.2000000 0.3000000 0.4000000 0.5000000 0.6000000
acc 0.6549977 0.7299248 0.8410871 0.8410871 0.8481499 0.8481499
F1  0.5712650 0.6126404 0.6580773 0.6580773 0.6013704 0.6013704
     [,7]     [,8]     [,9]
pis 0.7000000 0.8000000 0.9000000
acc 0.8481499 0.8063872 0.8063872
F1  0.6013704 0.3288984 0.3288984
```

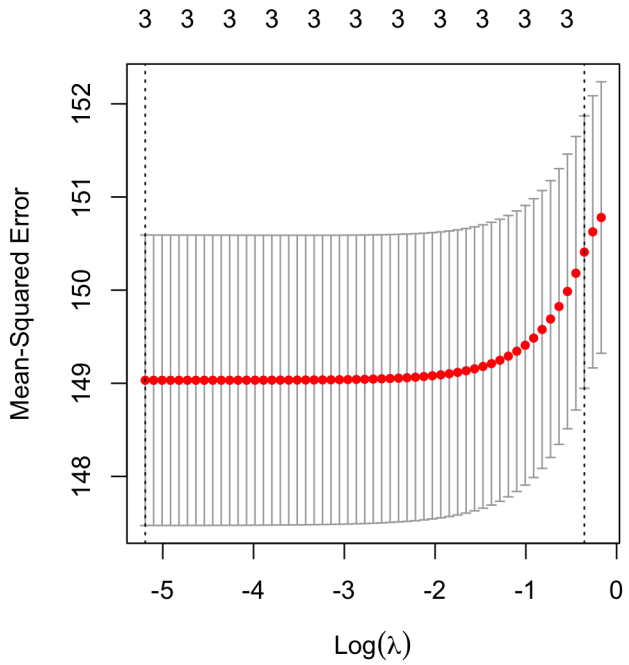
Optimal threshold is 0.3 or 0.4, F1 should be prioritized since classes are quite imbalanced.

Code:

```
#Task 1.2
pis = seq(0.1, 0.9, 0.1)
ps=length(pis)
F1=numeric(ps)
acc=numeric(ps)
test_pred = predict(opt_tree, newdata = test, type = "vector")
for (pi in pis) {
  pred_tree = ifelse(test_pred[,2]>pi, ">50K", "<=50K")
```

```
  tab=table(test$C13, pred_tree)
  TP = tab[2,2]
  FP = tab[1,2]
  FN = tab[2,1]
  TN = tab[1,1]
  P = TP + FN
  N = TN + FP
  precision = TP/(TP + FP)
  recall = TP/P
  F1[pi]=(2*precision*recall)/(precision + recall)
  acc[pi]=(TP+TN)/(P+N)
 }
 rbind(pis,acc,F1)
```

Task 1.3




```
 model$lambda.min
[1] 0.005552479
> coef(model, s="lambda.min")
4 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) 38.000806508
C1           0.052593047
C9           0.000107178
C10          0.001783425
```

3 variables selected, equation y=28+0.052C1+0.0001C9+0.0018C10+epsilon

The plot does not suggest that 0 features better than 3 (intervals overlap)

```
 #Task 1.3
 library(glmnet)
 library(dplyr)
```

```
selected_data = as.matrix(train %>% select(C1, C9, C10))
model=cv.glmnet(selected_data, train$C11, alpha=1,family="gaussian")
opt_lambda = model$lambda.min
print(opt_lambda)
plot(model)
coef(model, s="lambda.min")
```

Assignment 2

Task 2.1

Kernel Density Estimation (KDE) Functions: These functions use the Gaussian kernel to calculate KDE for a data point t in each class, averaging the densities of all samples scaled by bandwidth h.
Bandwidth Selection: The script tests various bandwidths (h) on a validation set, recording classification accuracy for each in the vector 'foo'.
Plotting and Optimal Bandwidth: A plot of accuracy against bandwidth helps identify the optimal h, found where foo reaches its maximum.
Generalization Error Estimation: Using the optimal h, KDE functions are updated with validation data. The model's generalization error is then estimated on a separate test set, reflecting the model's likely performance on new data. This demonstrates cross-validation for model tuning and performance assessment in a non-parametric context.

Code:

```
set.seed(123456789)
N_class1 <- 1000
N_class2 <- 1000
data_class1 <- NULL
for(i in 1:N_class1){
  a <- rbinom(n = 1, size = 1, prob = 0.3)
  b <- rnorm(n = 1, mean = 15, sd = 3) * a + (1-a) * rnorm(n = 1, mean = 4, sd = 2)
  data_class1 <- c(data_class1,b)
}
data_class2 <- NULL
for(i in 1:N_class2){
  a <- rbinom(n = 1, size = 1, prob = 0.4)
  b <- rnorm(n = 1, mean = 10, sd = 5) * a + (1-a) * rnorm(n = 1, mean = 15, sd = 2)
  data_class2 <- c(data_class2,b)
}
# Estimate the class conditional density functions: 2p.
conditional_class1 <- function(t, h){
  d <- 0
  for(i in 1:800)
    d <- d+dnorm((t-data_class1[i])/h)
  return (d/800)
}
conditional_class2 <- function(t, h){
  d <- 0
  for(i in 1:800)
    d <- d+dnorm((t-data_class2[i])/h)
  return (d/800)
}
# Estimate the class posterior probability distribution: 1p.
prob_class1 <- function(t, h){
  prob_class1 <- conditional_class1(t,h)*800/1600
```

```
  prob_class2 <-conditional_class2(t,h)*800/1600
  return (prob_class1/(prob_class1 + prob_class2))
}
# Select h value via validation: 1p.
foo <- NULL
for(h in seq(0.1,5,0.1)){
  foo <- c(foo, (sum(prob_class1(data_class1[801:900],
h)>0.5)+sum(prob_class1(data_class2[801:900], h)<0.5))/200)
}
plot(seq(0.1,5,0.1),foo)
max(foo)
which(foo==max(foo))*0.1
# Estimate the generalization error: 2p.
# To estimate the generalization error, we use the best h value found previously.
# Note that the training data is now the old training data union the validation data.
# Using just the old training data results results in an estimate that is a bit
# too pessimistic.
conditional_class1 <- function(t, h){
  d <- 0
  for(i in 1:900)
    d <- d+dnorm((t-data_class1[i])/h)
  return (d/900)
}
conditional_class2 <- function(t, h){
  d <- 0
  for(i in 1:900)
    d <- d+dnorm((t-data_class2[i])/h)
  return (d/900)
}
prob_class1 <- function(t, h){
  prob_class1 <- conditional_class1(t,h)*900/1800
  prob_class2 <-conditional_class2(t,h)*900/1800
  return (prob_class1/(prob_class1 + prob_class2))
}
h <- which(foo==max(foo))*0.1
(sum(prob_class1(data_class1[901:1000], h)>0.5)+sum(prob_class1(data_class2[901:1000],
h)<0.5))/200
```

Task 2.2

```
library(neuralnet)
set.seed(1234567890)
#Sampling 50 points randomly and uniformly in the interval [0,10]
Var <- runif(50, 0, 10)
#data frame with the variable and the sin value of the var
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(31, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10, startweights = winit,
```

```
                threshold = i/1000, lifesign = "full")
  aux <- predict(nn, tr) # Compute predictions for the trainig set and their squared error
  restr[i] <- sum((tr[,2] - aux)**2)/2
  aux <- predict(nn, va) # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
# The graphs show an example of overfitting, i.e. the threshold that achieves the lowest squared error
# in the training set is not the one that achieves the lowest error in the validation set. Therefore,
# early stopping is necessary, i.e. running gradient descent until convergence is not the best option,
# as the lowest threshold gives the best error in the training set but not in the validation set.
# Specifically, the validation set indicates that gradient descent should be stoped when
# threshold = 4/1000. So, the output should be a NN learnt with all (!) the data available and the
# threshold = 4/1000.
winit <- runif(31, -1, 1)
plot(nn <- neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit,
                     threshold = 4/1000, lifesign = "full"))
# Plot of the predictions (blue dots) and the data available (red dots)
plot(trva[,1],predict(nn,trva), col="blue", cex=3)
points(trva, col = "red", cex=3)
```

Exam 2023-01-14

```
##################Assignment 1################################
library(kknn)
library(dplyr)
#Task1
data = read.csv("tecator.csv")
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]
mse_test = c(length(30))
mse_train = c(length(30))
for (k in 1:30) {
  kknn_train = kknn(train$Fat ~. -Sample -Protein -Moisture, train=train,
test=train, kernel="rectangular", k=k)
  kknn_test = kknn(train$Fat ~. -Sample -Protein -Moisture, train=train,
test=test, kernel="rectangular", k=k)
  train_pred = predict(kknn_train)
  test_pred = predict((kknn_test))
  mse_train[k] = mean((train$Fat - train_pred)^2)
  mse_test[k] = mean((test$Fat - test_pred)^2)
}
plot(mse_test, type="l", x = seq(1,30,1), ylim=c(0,140))
lines(mse_train, col="blue")
print(which.min(mse_test))
mse_test[2]
```

```
mse_train[2]
#Test min value at X=2 gives mse of 75.49. This is the
#Optimal balance between a simple and a complex model, train mse = 21.77
#Task1.2
pca = princomp(data)
pca_scores = pca$scores
train_PC=data.frame(pca_scores[id,])
test_PC=data.frame(pca_scores[-id,])
train_PC$Fat = train$Fat
test_PC$Fat = test$Fat
train_kknn = kknn(Fat ~ + Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 +
Comp.7 + Comp.8 + Comp.9 + Comp.10, train=train_PC, test=train_PC,
kernel="rectangular", k=2)
test_kknn = kknn(Fat ~ + Comp.1 + Comp.2 + Comp.3 + Comp.4 + Comp.5 + Comp.6 +
Comp.7 + Comp.8 + Comp.9 + Comp.10, train=train_PC, test=test_PC,
kernel="rectangular", k=2)
mse_train_PC = mean((train_PC$Fat - predict(train_kknn))^2)
mse_test_PC = mean((test_PC$Fat - predict(test_kknn))^2)
#1.3
library(caret)
scaler = preProcess(train)
train_scaled = predict(scaler, train)
test_scaled = predict(scaler, test)
theta = rep(0,4)
mse_train_v = c()
mse_test_v = c()
cost_function = function(theta){
  predict_train = cbind(1, train_scaled$Fat, train_scaled$Fat^2,
train_scaled$Fat^3)%*%theta
  predict_test = cbind(1, test_scaled$Fat, test_scaled$Fat^2,
test_scaled$Fat^3)%*%theta
  mse_train = mean((train_scaled$Protein - predict_train)^2)
  mse_test = mean((test_scaled$Protein - predict_test)^2)
  mse_train_v <<- c(mse_train_v, log(mse_train))
  mse_test_v <<- c(mse_test_v, log(mse_test))
   return(mse_train)
}
opt = optim(par=theta, fn=cost_function, method = "CG")
mse_train_v = c()
mse_test_v = c()
opt20 = optim(par=theta, fn=cost_function, method = "CG", control =
list(maxit=20))
mse_train_v = c()
mse_test_v = c()
opt200 = optim(par=theta, fn=cost_function, method = "CG", control =
list(maxit=200))
predicted_protein20 = cbind(1, train_scaled$Fat, train_scaled$Fat^2,
train_scaled$Fat^3) %*% opt20$par
predicted_protein200 = cbind(1, train_scaled$Fat, train_scaled$Fat^2,
train_scaled$Fat^3) %*% opt200$par
x_axis = seq(1, length(mse_train_v), 1)
```

```
plot(x=x_axis,y=mse_train_v, type="l", xlim=c(0,200))
lines(mse_test_v, col="blue")
plot(train_scaled$Fat, train_scaled$Protein)
points(train_scaled$Fat, predicted_protein20, col="green")
plot(train_scaled$Fat, train_scaled$Protein)
points(train_scaled$Fat, predicted_protein200, col="green")

##############################Assignment 2################################
#Task 2.1
library(kernlab)
library(caret)
data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58])
tr <- spam[1:3000, ]
va <- spam[3001:3800, ]
te <- spam[3801:4601, ]
filter1 = ksvm(type~., data=tr, kenerl="rbfdot", kpar=list(sigma=0.05), C=0.5,
scaled=FALSE)
filter2 = ksvm(type~., data=tr, kenerl="rbfdot", kpar=list(sigma=0.05), C=1,
scaled=FALSE)
filter3 = ksvm(type~., data=tr, kenerl="rbfdot", kpar=list(sigma=0.05), C=5,
scaled=FALSE)
f1_val = predict(filter1, va[, -58])
t <- table(f1_val, va[,58])
err1 = (t[1,2]+t[2,1])/sum(t)

f2_val = predict(filter2, va[, -58])
t <- table(f2_val, va[,58])
err2 = (t[1,2]+t[2,1])/sum(t)

f3_val = predict(filter3, va[, -58])
t <- table(f3_val, va[,58])
err3 = (t[1,2]+t[2,1])/sum(t)

cat(err1, err2, err3)
# Filter 2 has lowest validation error, C=1
opt_error = predict(filter2, te[,-58])
t <- table(opt_error, te[,58])
err_opt = (t[1,2]+t[2,1])/sum(t)
cat("Generalization error is: ", err_opt)
# Returns filter trained on all available data to user
filter_user = ksvm(type~., data=spam, kenerl="rbfdot", kpar=list(sigma=0.05),
C=1, scaled=FALSE)
pred <- predict(filter_user, te[,-58])
t <- table(pred, te[,58])
err_opt = (t[1,2]+t[2,1])/sum(t)
cat("Generalization error for user is: ", err_opt)
# Purpose of parameter C is to account for the tradeoff between train and margin
```

Assignment 1 (10p)

File **tecator.csv** contains results of spectrographic analysis of various samples, including their characteristics such as Fat, Protein and Moisture content.

1. Split data into training and test sets (50/50) appropriately. Compute training and test MSE for nearest neighbor models having k=1,2,…,30 and Fat as target and all Channels as features. Plo dependences of the training and test errors as a function of k and interpret this plot in terms c bias-variance tradeoff. Report the training and test errors for the optimal model, and which k corresponds to the optimal model. **(4p)**
    a. **Note:** make sure to choose the right "kernel" parameter in kknn, that corresponds to the conventional K-nearest neighbor algorithm.
2. Perform PCA on all Channel columns of the original dataset (without scaling), obtain the PCA scores (coordinates of the data in the PC coordinate system), and then split the scores and Fa column into training and test data by using same observation indices as in step 1. Compute a k nearest neighbor model using Fat as target and first 10 PC columns as features and optimal k from step 1 and report training and test MSEs. Compare them to the MSEs obtained for the optimal model in step 1 and comment why there such a difference, especially for K-nearest neighbor models? **(2p)**
3. Scale the training and test sets from step 1 appropriately. Consider a third-degree polynomial model where Protein is target and Fat is feature and implement the code that uses the Conjugate Gradient (CG) optimizer to find optimal parameters of this model, and plots dependence of training and test log(MSE) on the optimization iteration number. Is early stopping needed? Make a plot of (Fat, Protein) training data overlayed by the (Fat, Protein_predicted) of the model corresponding to the number of iterations equal to 20, and same plot for the number of iterations equal to 200, compare these plots and make necessary conclusions. **(4p)**
    a. **Note:** you are supposed to use function optim() to conduct the optimization.
    b. **Note2:** as a vector of initial parameter values of the optimizer, use zero vector.

Assignment 2 (10p)

Support Vector Machines - 5 Points

You are asked to use the function **ksvm** from the R package **kernlab** to learn a support vector machine (SVM) for classifying the **spam** dataset that is included with the package. Consider th radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C paramete consider values 0.5, 1 and 5. This implies that you have to consider three models.

(2 p) Perform model selection, i.e. select the most promising of the three models (use any method of your choice).

(1 p) Estimate the generalization error of the SVM selected above (use any method of your choice).

(1 p) Produce the SVM that will be returned to the user, i.e. show the code.

(1 p) What is the purpose of the parameter C ?