TDDE01 - Lab 1

David Forslöf - davfo018, assignment 1
Martin Friberg - marfr370, assignment 2
Filip Frenning - filfr933, assignment 3

2020/11/18

1 Assignment 1. Handwritten digit recognition with K-means

The data was first split into training, testing and validation sets by the partitioning principle with a 50%/25%/25% split respectively.

The `kknn` function was used with `K=30` which produced the confusion matrices shown below. The overall quality of predictions for the training data, as well as test data, is pretty low for a digit recognition model. A missclassification error of about 5.3% and 4.5% respectively for digit recognition is by today standards not good enough. We can see that the digit 8 and digit 9 was hard to classify in both training and test data, while digit 0 and digit 6 were easy to classify.

	0	1	2	3	4	5	6	7	8	9
0	202	0	0	0	1	0	0	0	0	1
1	0	179	1	0	3	0	2	3	10	3
2	0	11	190	0	0	0	0	0	0	0
3	0	0	0	185	0	1	0	0	2	5
4	0	0	0	0	159	0	0	0	0	2
5	0	0	0	1	0	171	0	0	0	0
6	0	0	0	0	0	0	190	0	2	0
7	0	1	1	1	7	1	0	178	0	3
8	0	1	0	0	1	0	0	1	188	3
9	0	3	0	1	4	8	0	0	2	183

The confusion matrix for the test data is presented below.

	0	1	2	3	4	5	6	7	8	9
0	77	0	0	0	0	0	0	0	0	0
1	0	81	0	0	0	1	0	0	7	1
2	0	2	98	0	0	1	0	0	0	1
3	0	0	0	107	0	0	0	1	1	1
4	1	0	0	0	94	0	0	0	0	0
5	0	0	0	2	0	93	0	0	0	0
6	0	0	0	0	2	2	90	0	0	0
7	0	0	0	0	6	1	0	111	0	1
8	0	0	3	1	2	0	0	0	70	0
9	0	3	0	1	5	5	0	0	0	85

1.1 Easy and hard identfied eights

The image below highlights the 2 of the easiest eights to classify and 3 of the hardest. Visually the easy cases seems to be easy to recognize visually, while the hard cases seems to be hard to recognize visually. Concluded, the algorithm seems to struggle where humans also struggle.

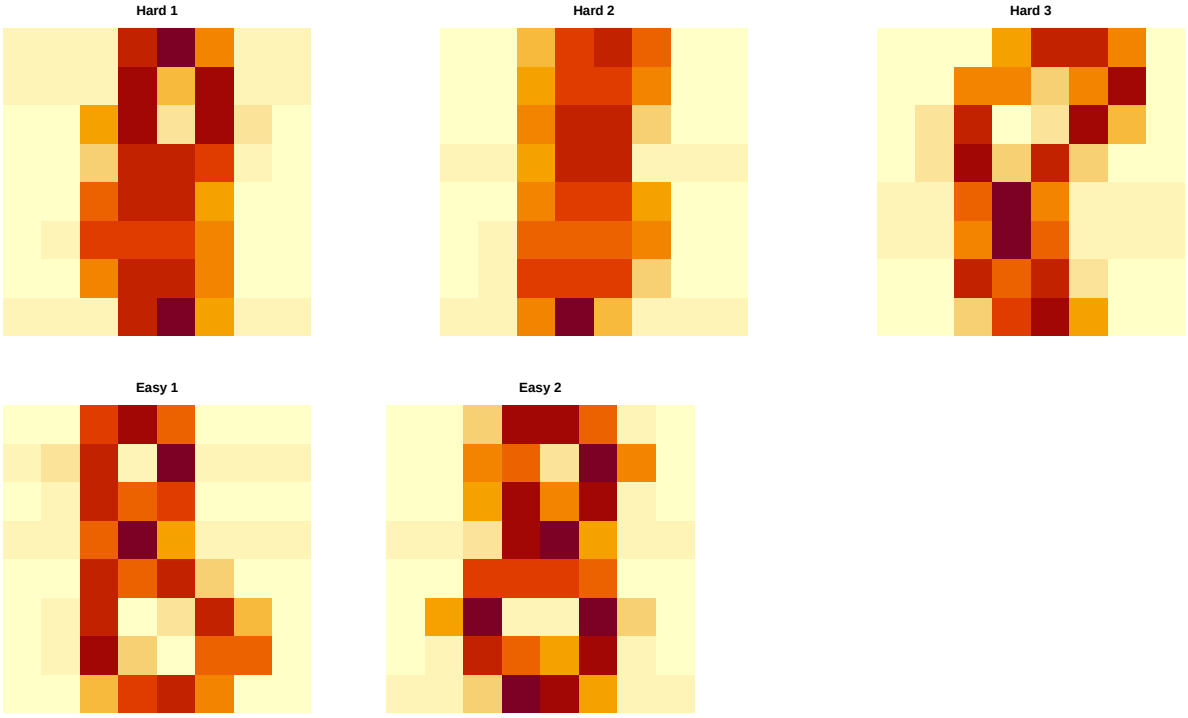

Figure 1: Easy and hard identified eights

1.2 Missclassification errors for different values of K

The model complexity decreases with increased values of K. We can also see in the figure below that the training and validation errors increases with higher values of K. The optimal K according to this plot alone is 6 since they have the lowest combined missclassification errors. Lower K's leads to bias being 0, but when new data should be predicted it has a higher change of being a missclassification. When K increases the training error increases (increases bias). The error for the optimal K seems to be about 2.3%.

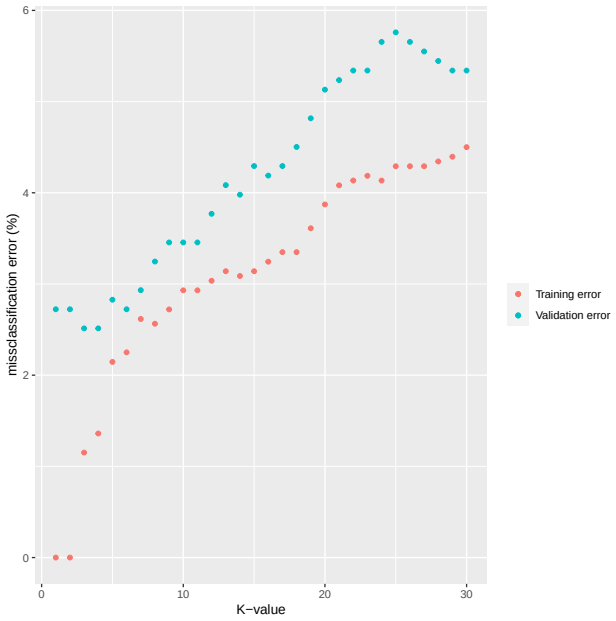

Figure 2: Task 4

1.3 Cross-entropy levels for different values of K

When evalutaing the cross-entropy risk it seems that `K=6` is the best value for the model. Cross-entropy seems to be a better empirical risk function since it takes the probability for each correct digit in consideration instead of just missclassifications.

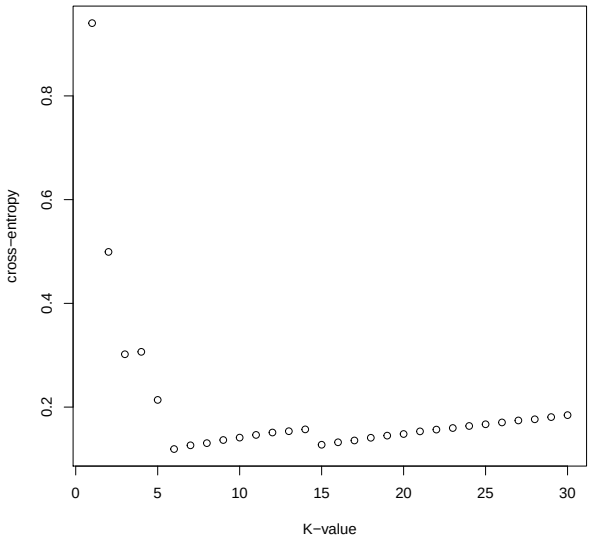

Figure 3: Task 5

2 Assignment 2. Ridge regression and model selection

The data file parkinson.csv is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring. The purpose is to predict Parkinson's disease symptom score (motor UPDRS) from the following voice characteristics:

- Jitter(%),Jitter(Abs),Jitter:RAP,Jitter:PPQ5,Jitter:DDP - Several measures of variation in fundamental frequency
- Shimmer,Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,Shimmer:APQ11,Shimmer:DDA - Several measures of variation in amplitude
- NHR,HNR - Two measures of ratio of noise to tonal components in the voice
- RPDE - A nonlinear dynamical complexity measure
- DFA - Signal fractal scaling exponent
- PPE - A nonlinear measure of fundamental frequency variation

2.1 Probabilistic model as a Bayesian model of the motor_UPDRS

Assuming that motor_UPDRS is normally distributed and can be modeled by Ridge regression of the voice characteristics.

$$motor\_UPDRS \sim N(y|w_0 + Xw, \sigma I)$$

$$w \sim N(0, \frac{\sigma}{\lambda} I)$$

2.2 Scaling the data

Data from the the file about voice characteristics and whether or not the person has parkinson's was scaled i.e calculating the mean and standard deviation of the entire vector, then scaling each element by those values by subtracting the mean and dividing by the standard deviation. The data was then divided into a training set which consisted of 60% of the data from the parkinson's data file as well as a training set which consisted of 40% of the data from the parkinson's data file.

2.3 Implementation of four different functions to calculate ridge regression and degrees of freedom

a. Loglikelihood function that for a given parameter vector w and dispersion σ computes the log-likelihood function log P(D|w,σ):

$$\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{(2\sigma^2)}(Y - XW)^T(Y - XW)$$

b. Ridge function that for given vector w, scalar σ and scalar λ uses function from 2a and adds up a Ridge penalty to the minus loglikelihood:

$$-\text{Loglikelihood} + \lambda \sum_{j=1}^{p} w_j^2$$

c. RidgeOpt function that depends on scalar λ, uses function from 2b and function optim() with method="BFGS" to find the optimal w and σ for the given λ, where rep(1,17) is a vector consisting of the weights and σ to be optimized:

```
RidgeOpt = function(lamda, train){
  optim(rep(1,17), fn=Ridge, lamda=lamda, train=train , method = "BFGS")
}
```

d. Calculating degrees of freedom of the regression model based on λ:

$$P = X(X^T X + \lambda I)^{-1} X^T$$

$$trace(P)$$

2.4 Computation of optimal w parameters for $\lambda = 1$, 100, 1000 in order to calculate the minimum mean squared error value.

MSE for the training data became:

- $\lambda = 1$: 0.8732...
- $\lambda = 100$: 0.8790...
- $\lambda = 1000$: 0.9156...

and for the test data:

- $\lambda = 1$: 0.9290...
- $\lambda = 100$: 0.9263...
- $\lambda = 1000$: 0.9479...

$\lambda = 100$ is a more appropriate penalty parameter among the selected ones of $\lambda = 1$, $\lambda = 100$, $\lambda = 1000$ since $\lambda = 100$ gives the lowest MSE value for the test data even though it is slightly worse for the training data.

The mean squared error is in this case a good measure of loss since we want to know how much the predicted values deviate from the actual values of the motor_UPDRS. It does not matter if the predicted value is above or below the actual one, we just want to estimate how far off the predicted value is on average. We also consider values that are far off from the actual motor_UPDRS value to be worse which is why MSE is a better measurement than the absolute error.

2.5 Calculation of the AIC (Akaike Information Criterion) scores for the Ridge models with values lamda = 1, 100, 1000 with their corresponding optimal variables

Values of the AIC criterion for the different lambdas:

- $\lambda = 1$: 6528...
- $\lambda = 100$: 6510...
- $\lambda = 1000$: 6554...

The optimal model according to the AIC criterion is when $\lambda = 100$ since the chosen lambda gives the lowest value of the AIC criterion. The theoretical advantage of using the AIC criterion compared to the holdout method is that the AIC criterion attempt to quantify both the model performance on the training dataset and the complexity of the model. Whereas the holdout method only tries to evaluate the model performance on the training dataset.

3 Assignment 3. Linear regression and LASSO

The data was first divided into training and test data by the partitioning principle 50/50.

3.1 Task 1 - Linear Regression

Since we in this task are going to model the level of Fat by linear regression, we get the probabilistic model as such:

$$y \sim N(w_0 + w_1 x, \sigma^2)$$

In this case, the 'y' from the model is represented by the Fat and symbolize the response from the fitting. We build the formula by using the probabilistic model, and symbolically specifying the model for the lm function by saying that "Fat ~ .". Here the dot represents all of the absorbance characteristics that are used as features. When we have fitted the linear regression to the training and test data we use these to make predictions on the training and test data. These predictions, along with the original values for Fat, were then inserted in the mean-squared-error formula below.

To be able to estimate the errors for the training and test data we first have to set up the formula for mean-squared-error (MSE):

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

The MSE for the fitted and predicted training data is very low at a value of about 0.00571 which indicate that the prediction was of high quality. On the other hand, our MSE for the fitted and predicted test data was very high at a value of about 722.42942 which indicate a prediction of a lower quality. This can be explained by the fact that we fit our linear regression to the training data.

3.2 Task 2 - LASSO Regression function

The objective function that should be optimized in this scenario is:

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

Which is subject to:

$$\sum_{j=1}^{p}|w_i| \leq s$$

3.3 Task 3 - LASSO Regression

The plot below illustrates how the regression coefficients on the y-axis depend on the log of the penalty factor. This essentially means that when the lambda values get smaller, the regularization of the model has less effect and the model therefore is unregularized. This type of model has a lot of non-zero coefficients and could be hard to interpret. Since the lasso penalizes the sum of the coefficients absolute values, when the lambda or penalty factor increase, the coefficients eventually become zero for high lambdas and the model becomes more regularized. Lasso can therefore perform some variable selection.

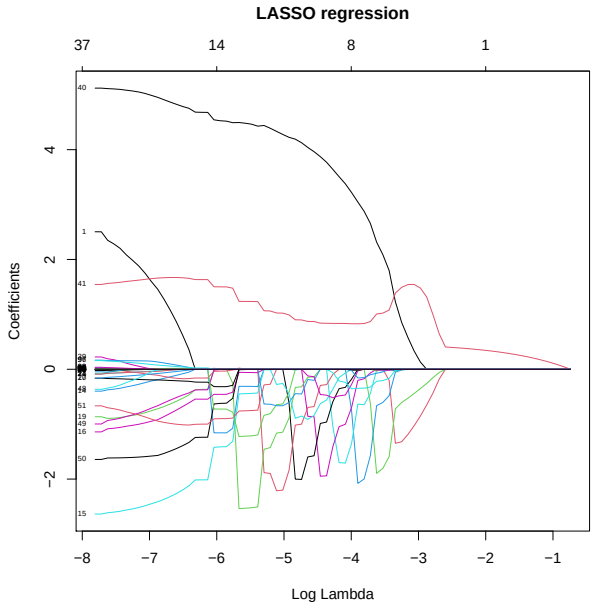

Figure 4: Lasso

As we can see in the graph, when log lambda has a value of about -2.8, three of the features are separate from a value of zero.

3.4 Task 4 - Degrees of Freedom

The plot below shows how the degrees of freedom of the lasso model depend on the log lambda, or the log of the penalty factor.

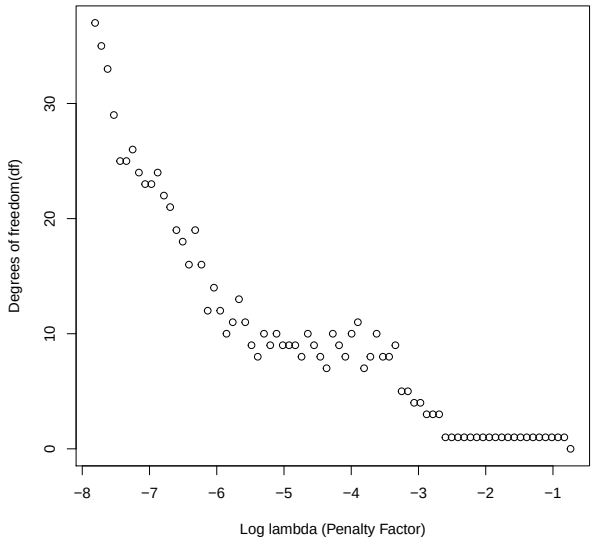

Figure 5: Degrees of Freedom

This trend is very much expected, since we know that the degrees of freedom should decrease and be approaching a value of zero when lambda increases.

3.5 Task 5 - Ridge Regression

As we see in the plot below, the coefficients close in on a value of zero when lambda increase, but in the case of the ridge regression the values never actually become equal to zero. In the same way as the lasso, when lambda is smaller the model becomes unregularized because of its many coefficients separate from zero. Also, in the same way as the lasso, the penalty factor penalize the variables when it has a higher value.

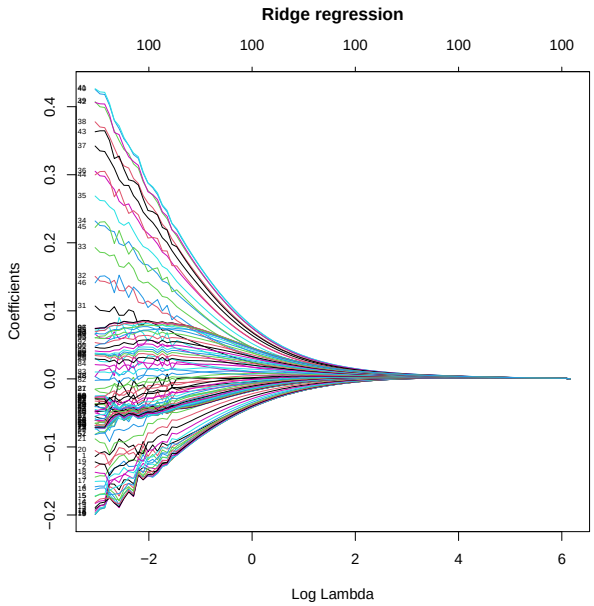

Figure 6: Ridge

3.6 Task 6 - Cross-Validation

To compute the optimal LASSO model we use cross-validation. Below we show a plot where we see how the CV score depend on the log of the penalty factor. We see in the graph that as the log of the penalty factor increase, the cross-validated MSE also increase and can help interpret the optimal lambda in our LASSO model.

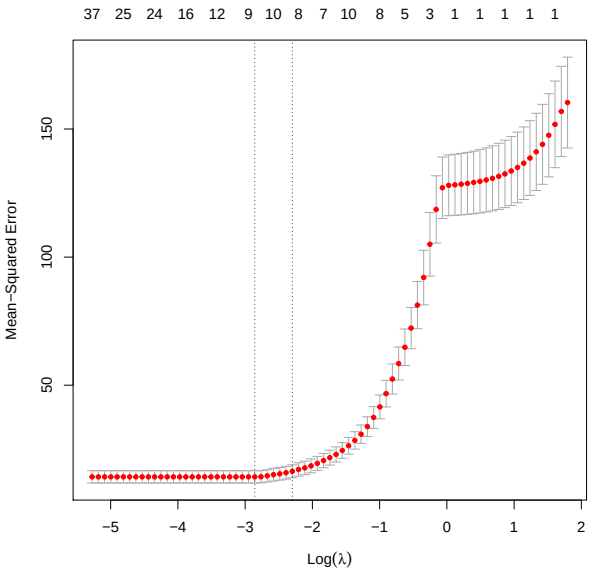

Figure 7: CV

From the model we can get the optimal lambda which has a value of about 0.057445. The amount of variables chosen in the model with this optimal lambda is 8.

The scatter plot of the original test versus the predicted test values is shown below, and as we can see the scatter plot indicate a quite good model since the plot resembles a linear relationship between the variables and some form of positive correlation.

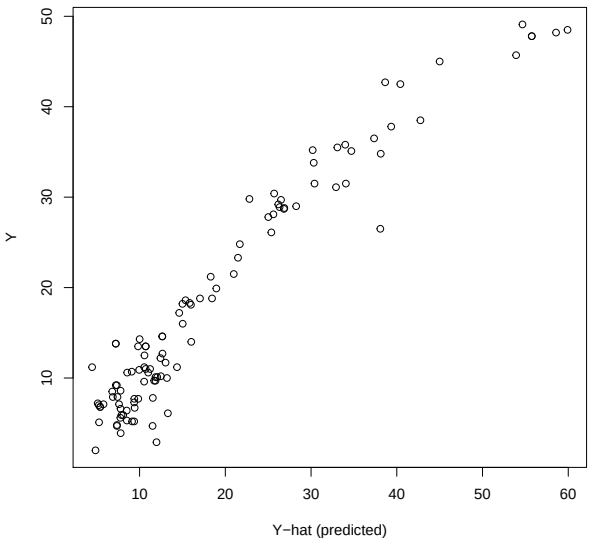

Figure 8: ScatterCV

3.7 Task 7 - Generating new values

By using the rnorm-function in R together with our optimal function from previous tasks, we can generate new target values. These are shown in relation to the original values for Fat in the test data in the scatter plot below.

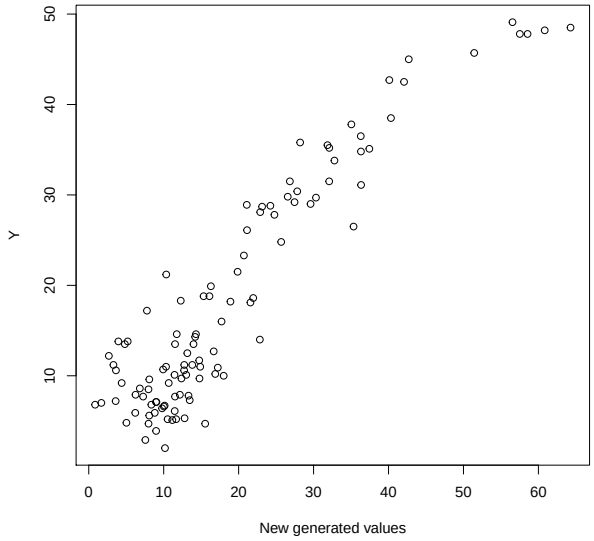

Figure 9: Newfatvalues

In this scatter plot we can also say that the newly generated values have a fairly good quality as the plot closely resembles a linear relationship between the variables and some form of positive correlation.

4 Appendix

4.1 Code for Assignment 1

```
data = read.csv('optdigits.csv', header=F)

# TASK 1
# Split data 50/25/25
n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.25))
id3 = setdiff(id1,id2)
train = data[id,]
valid = data[id2,]
test = data[id3,]

# TASK 2
# Use training data to fit 30-nearest neighbor classifier with function kknn() and
# kernel="rectangular" from package kknn and estimate

library(kknn)
library(gridExtra)
library(grid)

kknn_train = kknn(as.factor(V65) ~ ., train=train, test=train, k = 30,
                  kernel = 'rectangular')
kknn_fitted_train = fitted(kknn_train)

kknn_test = kknn(as.factor(V65) ~ ., train=train, test=test, k = 30,
                 kernel = 'rectangular')
kknn_fitted_test = fitted(kknn_test)

confusionM_train = table(kknn_fitted_train, train$V65)
misclassification_train = 1-sum(diag(confusionM_train)/sum(confusionM_train))

confusionM_test = table(kknn_fitted_test, test$V65)
misclassification_test = 1-sum(diag(confusionM_test)/sum(confusionM_test))

# TASK 3
# Find any 2 cases of digit "8" in the training data which were easiest to classify
# and 3 cases that were hardest to classify
library(dplyr)
make_heatmap = function(list, name) {
  x <- unlist(list)
  x <- matrix(x, nrow=8, byrow=TRUE)
  heatmap(x, Colv=NA, Rowv=NA, main=name)
}

eights_row_no = which(grepl(8, train$V65))
probs_eights = data.frame(row_no=eights_row_no,
                          probs=kknn_train[["prob"]][eights_row_no,9])
```

```
easy = (probs_eights %>% top_n(2))[1:2,1]
hard = (probs_eights %>% top_n(-3))[1:3,1]

make_heatmap(train[easy[1],][1:64], 'Easy 1')
make_heatmap(train[easy[2],][1:64], 'Easy 2')
make_heatmap(train[hard[1],][1:64], 'Hard 1')
make_heatmap(train[hard[2],][1:64], 'Hard 2')
make_heatmap(train[hard[3],][1:64], 'Hard 3')

# TASK 4
# Plot errors and dependecy
library(ggplot2)
x4 = vector()
y4_train = vector()
y4_valid = vector()
for (k in 1:30) {
  kknn_train = kknn(as.factor(V65) ~ ., train=train, test=train, k = k,
                    kernel = 'rectangular')
  kknn_valid = kknn(as.factor(V65) ~ ., train=train, test=valid, k = k,
                    kernel = 'rectangular')
  confusionM_train = table(fitted(kknn_train), train$V65)
  confusionM_valid = table(fitted(kknn_valid), valid$V65)
  x4[k] = k
  y4_train[k] = (1-sum(diag(confusionM_train)/sum(confusionM_train)))*100
  y4_valid[k] =  (1-sum(diag(confusionM_valid)/sum(confusionM_valid)))*100
}
data_frame = data.frame(y1=y4_train, y2=y4_valid, x=x4)
task4_plot = ggplot(data_frame, aes(x=x4)) + geom_point(aes(y=y1, color='Training error'))
+ geom_point(aes(y=y2, color='Validation error')) + labs(color="") + xlab('K-values') +
  ylab('Missclassification error')

# TASK 5
x5 = vector()
y5 = vector()
for (k in 1:30) {
  prob_kknn = kknn(as.factor(V65) ~ ., train=train, test=valid, k = k,
                   kernel = 'rectangular')
  pred_kknn = fitted(prob_kknn)
  confusionM_valid = table(pred_kknn, valid$V65)
  x5[k] = k
  crossent = 0
  for (i in 1:dim(valid)[1]) {
    q = prob_kknn[["prob"]][i, valid$V65[i]+1]
    crossent = crossent + (log(q+1e-15))
  }
  y5[k] = -crossent / dim(valid)[1]
}
plot(x5, y5, xlab='K-value', ylab='cross-entropy')
```

4.2 Code for Assignment 2

```r
#TASK 2
library(DMwR)
library(MLmetrics)
csvfile = read.csv("parkinsons.csv")
csvfile = scale(csvfile)
set.seed(12345)
n=dim(csvfile)[1]
id=sample(1:n, floor(n*0.6))
train=csvfile[id,]
test=csvfile[-id,]
y_train = train[,5]
y_test = test[,5]
x_train = train[,7:22]
x_test = test[,7:22]
# TASK 3
Loglikelihood = function(w, sigma, D) {
  n = dim(D)[1]
  Y = D[,5]
  X = as.matrix(D[,7:22])
  return(- n / 2 * log(2 * pi * sigma^2) - sum(t(Y - X%*%w)%*%(Y - X %*% w))/(2 * sigma^2))
}

Ridge = function(a, lamda, train){
  w = a[-1]
  sigma = a[1]
  return(- Loglikelihood(w, sigma, train) + lamda * sum(w^2))
}

RidgeOpt = function(lamda, train){
  optim(rep(1,17), fn=Ridge, lamda=lamda, train=train , method = "BFGS")
}

# The sum of the diagonals of a matrix is called the Trace of matrix and we have
# denoted that in the second line. df=trace(hat_matrix)
DF = function(lamda, train){
  X = as.matrix(train[,7:22])
  Y = train[,5]
  i_matrix = diag(16)
  ab = t(X)%*%X+lamda*i_matrix
  ba = solve(ab)
  hat_matrix = X%*%ba%*%t(X)
  return(sum(diag(hat_matrix)))
}


#TASK 4
# RidgeOpt for different lamdas
r_1 = RidgeOpt(1, train)

r_100 = RidgeOpt(100, train)

r_1000 = RidgeOpt(1000, train)
```

```r
# optimal weights and sigma for different lamdas
#lamda=1
parameters_1 = r_1$par
opt_weights_1 = parameters_1[-1]
opt_sigma_1 = parameters_1[1]
#lamda=100
parameters_100 = r_100$par
opt_weights_100 = parameters_100[-1]
opt_sigma_100 = parameters_100[1]
#lamda=1000
parameters_1000 = r_1000$par
opt_weights_1000 = parameters_1000[-1]
opt_sigma_1000 = parameters_1000[1]

prediction_train_1 = x_train %*% opt_weights_1
prediction_train_100 = x_train %*% opt_weights_100
prediction_train_1000 = x_train %*% opt_weights_1000

prediction_test_1 = x_test %*% opt_weights_1
prediction_test_100 = x_test %*% opt_weights_100
prediction_test_1000 = x_test %*% opt_weights_1000

MSE_1_train = MSE(y_train, prediction_train_1)
MSE_100_train = MSE(prediction_train_100, y_train)
MSE_1000_train = MSE(prediction_train_1000, y_train)

MSE_1_test = MSE(prediction_test_1, y_test)
MSE_100_test = MSE(prediction_test_100, y_test)
MSE_1000_test = MSE(prediction_test_1000, y_test)


#TASK 5

AIC = function(opt_weights, opt_sigma, lambda, test){
  return(2*DF(lambda, test)-2*Loglikelihood(opt_weights, opt_sigma, test))
}

AIC_1 = AIC(opt_weights_1, opt_sigma_1, 1, test)
AIC_100 = AIC(opt_weights_100, opt_sigma_100, 100, test)
AIC_1000 = AIC(opt_weights_1000, opt_sigma_1000, 1000, test)
print(sum(opt_weights_1))
print(sum(opt_weights_100))
print(sum(opt_weights_1000))
```

4.3 Code for Assignment 3

```
mydata = read.csv('tecator.csv')
library(glmnet)

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

# Assignment 3 Task 1
# Fit linear regression to the training data and
# estimate the training and test errors.

mean_squared_error = function(y, y_hat) {
  squared_error = (y-y_hat)^2
  sum_squared_error = sum(squared_error)
  n = length(squared_error)
  return(sum_squared_error/n)
}

fit = lm(formula = Fat ~., data = train[,-c(1, 103, 104)])

# Predictions
fitted_train = predict(fit, newdata = train)
fitted_test = predict(fit, newdata = test)

# MSE Train
MSE_train = mean_squared_error(train$Fat, fitted_train)

# MSE Test
MSE_test = mean_squared_error(test$Fat, fitted_test)

print(MSE_train)
print(MSE_test)

summary(fit)

# Assignment 3 Task 2,3
# Fit the Lasso regression model to the training data.
# Present a plot illustrating how the regression coefficients
# depend on the log of penalty factor (log lambda)
# What value of the penalty factor can be
# chosen if we want to select a model with only three features?

covariates=scale(train[,2:101])
response=scale(train$Fat)

model_Lasso=glmnet(as.matrix(covariates), response, alpha=1, family="gaussian")
plot(model_Lasso, xvar="lambda", label=TRUE, main="LASSO regression\n\n")

#Assignment 3 Task 4
#Present a plot of how degrees of freedom depend on the penalty parameter.
```

```r
plot(log(model_Lasso$lambda), model_Lasso$df, xlab = "Log lambda (Penalty Factor)",
     ylab = "Degrees of freedom(df)")

# Assignment 3 Task 5
# Fit Ridge instead of the LASSO regression and compare the
# plots from steps 3 and 5

model_Ridge=glmnet(as.matrix(covariates), response, alpha=0, family="gaussian")

plot(model_Ridge, xvar="lambda", label=TRUE, main="Ridge regression\n\n")

# Assignment 3 Task 6
# Use cross-validation to compute the optimal LASSO model. Present a plot
# showing the dependence of the CV score on log lambda and comment how the CV
# score changes with log lambda.
# Finally, create a scatter plot of the
# original test versus predicted test values for the model corresponding to
# optimal lambda

covariates_cv=train[,2:101]
response_cv=train$Fat

model_cv_Lasso=cv.glmnet(as.matrix(covariates_cv), response_cv, alpha=1,
                         family="gaussian")
lambda_best_cv = model_cv_Lasso$lambda.min

plot(model_cv_Lasso)
coef(model_cv_Lasso, s="lambda.min")

y = test[,102]
y_predict = predict(model_cv_Lasso, s = lambda_best_cv, newx = as.matrix(test[, 2:101]),
                    type="response")
plot(y_predict, y, xlab = "Y-hat (predicted)", ylab = "Y")

# Assignment 3 task 7
# Use the feature values from test data (the portion of test data with Channel
# columns) and the optimal LASSO model from step 6 to generate new target
# values.
# Make a scatter plot of original Fat in test data
# versus newly generated ones.

std_dev = sd(y_predict - y)
plot(rnorm(108, y_predict, std_dev), y, xlab = "New generated values", ylab = "Y")
```