Lab 1 TDDE01

Daniel Kouznetsov (danko376), Liv Kåreborn (livka967), Erik Johansson (erijo073)

2022-11-10

Contributions

Assignment 1: Daniel Assignment 2: Liv Assignment 3: Erik

Assignment 1: Handwritten recognition with K-nearest neighbors

1.1)

Splitting data into training, validation and test sets where training consists of 50% of data and validation and test sets are 25% each.

```
library(kknn)
library(dplyr)

data = read.csv("optdigits.csv")
n = dim(data)[1]
set.seed(12345)
id.1 = sample(1:n, floor(n*0.5))

names(data)[ncol(data)] = "lastcol"
data$lastcol = as.factor(data$lastcol)

# 50% of data is training data
data.train = data[id.1,]

# Split remaining data into two
data.half = data[-id.1,]
n.half = dim(data.half)[1]
id.2 = sample(1:n.half, floor(n.half*0.5))
data.validation = data.half[id.2,]
data.test = data.half[-id.2,]
```

1.2)

Using function kknn() to fit a k-nearest neighbor classifier with k = 30.

```
fit.test = kknn(
  formula = lastcol~.,
  train = data.train,
  test = data.test,
  k = 30,
  kernel = "rectangular"
)
```

```
fit.train = kknn(
  formula = lastcol~.,
  train = data.train,
  test = data.train,
  k = 30,
  kernel = "rectangular"
)

# Confusion matrices for our predictions
cm.train = table(data.train$lastcol, fit.train$fitted.values)
cm.test = table(data.test$lastcol, fit.test$fitted.values)
cm.train
```

```
##
##       0   1   2   3   4   5   6   7   8   9
##   0 177   0   0   0   1   0   0   0   0   0
##   1   0 174   9   0   0   0   1   0   1   3
##   2   0   0 170   0   0   0   0   1   2   0
##   3   0   0   0 197   0   2   0   1   0   0
##   4   0   1   0   0 166   0   2   6   2   2
##   5   0   0   0   0   0 183   1   2   0  11
##   6   0   0   0   0   0   0 200   0   0   0
##   7   0   1   0   1   0   1   0 192   0   0
##   8   0  10   0   1   0   0   2   0 190   2
##   9   0   3   0   4   2   0   0   2   4 181
```

```
cm.test
```

```
##
##       0   1   2   3   4   5   6   7   8   9
##   0  88   0   0   0   0   0   0   0   0   0
##   1   0  85   3   0   0   0   0   0   0   1
##   2   0   0 105   0   0   0   0   1   3   1
##   3   0   0   0  88   0   1   1   3   2   0
##   4   1   0   0   0  99   0   0   4   1   2
##   5   0   0   0   3   0  80   1   0   1   6
##   6   0   2   0   0   0   0  79   0   0   0
##   7   0   1   0   0   0   1   0  88   0   1
##   8   0   4   0   1   1   0   0   0  93   1
##   9   0   1   0   2   0   0   0   6   0  95
```

```
# Function to calculate the number of incorrect predictions
mis = function(data, pred){
  incorrect.guesses = 0
  for (i in 1:length(pred)){
    if (data[i] != pred[i]){
      incorrect.guesses = incorrect.guesses + 1
    }
  }
  return (incorrect.guesses)
}

# Miscalculation errors
incorrect.guesses.train = mis(data.train$lastcol, fit.train$fitted.values)
mis.train = incorrect.guesses.train / length(fit.train$fitted.values)
```

```r
mis.train # 0.0423
```

```
## [1] 0.04238619
```

```r
incorrect.guesses.test = mis(data.test$lastcol, fit.test$fitted.values)
mis.test = incorrect.guesses.test / length(fit.test$fitted.values)
mis.test  # 0.0586
```

```
## [1] 0.05857741
```

The predictions for some numbers we're worse than others. For example it had a hard time classifying "4" which it guessed to be several other numbers multiple times. "5" when comparing with the training data guessed that it was "9" 11 times. Overall the quality of predictions I would say is not really that good. We have a misclassication error of 2-5% which would result in quite a lot of errors with a large set of predictions.

1.3)

Analyzing digits of "8", in this case two of the ones that we're easiest to classify and three that we're the hardest to classify.

From inspection of the vector for the probabilities we can quickly see that index 45 and 141 has a probability of 1, therefore we choose these indexes to work on for the ones that we're easy to classify.

For the numbers that we're hard to classify we need to write some code to figure it out.

```r
# Get the index for the lowest values for "8"
prob = fit.test$prob[,9]
prob.nozero = lapply(prob, function(x) if(x==0) return (NA) else return(x))

# lapply returns a list, unlists it to be able to use order
prob.nozero = unlist(prob.nozero)
ordered.prob = order(prob.nozero, decreasing=FALSE)

ordered.prob[1:3]
```

```
## [1] 30 48 65
```

The indexes with the lowest values for classifying "8" is 30, 48 and 65 with probabilities of 0.033.

```r
# Extract the values for the lowest probability classifications except for the last column
low = data.test %>% select(1:64)

# Heatmaps
lowest.prob = matrix(unlist(low[30,]), ncol = 8, nrow=8, byrow=TRUE)
heatmap(lowest.prob, Colv = NA, Rowv = NA)
```

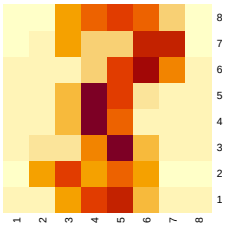
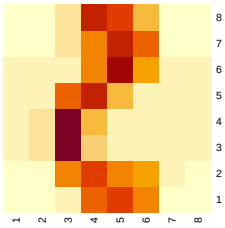
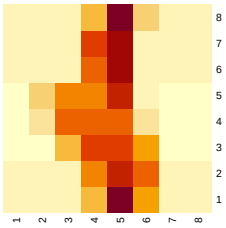
```r
lowest.prob = matrix(unlist(low[48,]), ncol = 8, nrow=8, byrow=TRUE)
heatmap(lowest.prob, Colv = NA, Rowv = NA)
```

```r
lowest.prob = matrix(unlist(low[65,]), ncol = 8, nrow=8, byrow=TRUE)
heatmap(lowest.prob, Colv = NA, Rowv = NA)
```

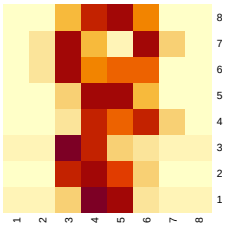
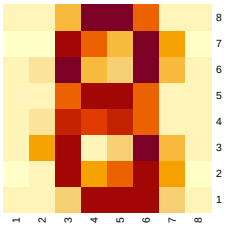
```r
# Same for high probability classifications
high = data.test %>% select(1:64)

# Heatmaps
highest.prob = matrix(unlist(high[45,]), ncol=8, nrow=8, byrow=TRUE)
heatmap(highest.prob, Colv = NA, Rowv = NA)
```

```
highest.prob = matrix(unlist(high[141,]), ncol=8, nrow=8, byrow=TRUE)
heatmap(highest.prob, Colv = NA, Rowv = NA)
```

Hard to classify

Easy to classify

The hard ones to classify are really hard to classify as 8's, from inspecting the plot I would guess that they are a 3, 2 and a 4. When checking the data for what it actually was we get that the actual numbers are 3, 5 and 1.

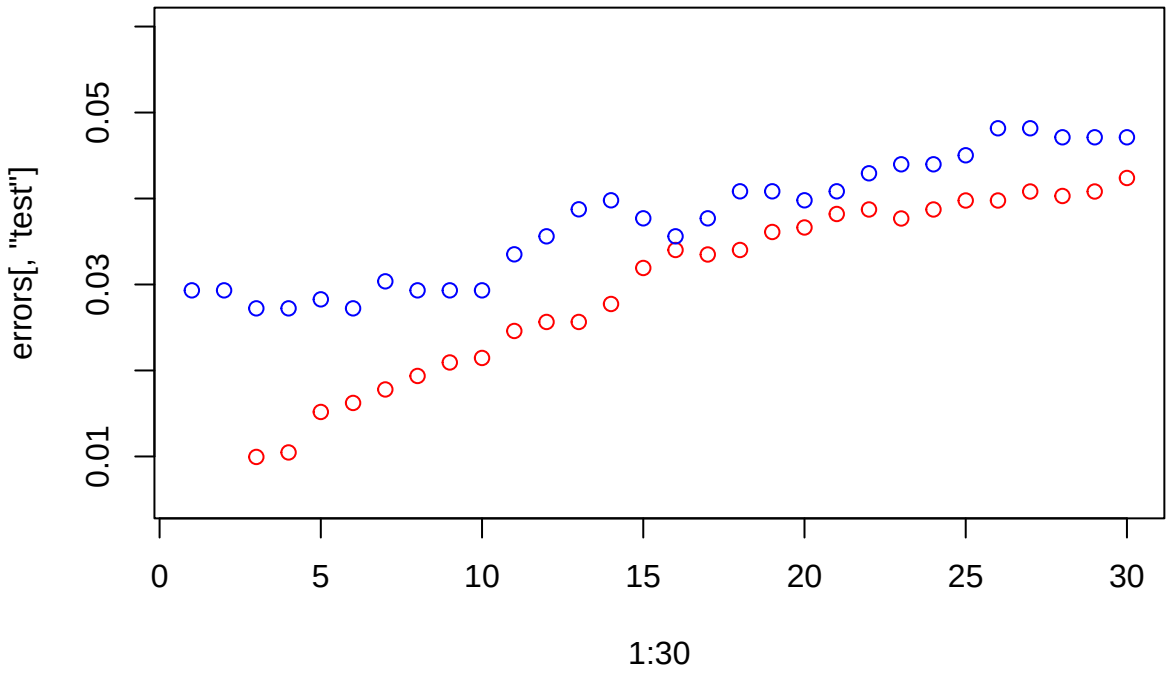
The ones that we're easy to classify are quite easy to visually analyze to come to the conclusion that they are eights, especially the last one.

1.4)

Fitting a K-nearest neighbors classifier to the data using K = 1, . . . , 30, plotting misclassification errors.

```
errors = matrix(data = NA, nrow = 30, ncol = 2, dimnames = list(c(), c("test", "validation")))

for (i in 1:30) {
  fit.test = kknn(
    formula = lastcol~.,
    train = data.train,
    test = data.train,
    k = i,
    kernel = "rectangular"
  )

  fit.validation = kknn(
    formula = lastcol~.,
    train = data.train,
    test = data.validation,
    k = i,
    kernel = "rectangular"
  )
```

```
  inc.test = mis(data.train$lastcol, fit.test$fitted.values)
  inc.validation = mis(data.validation$lastcol, fit.validation$fitted.values)

  mis.test = inc.test / length(fit.test$fitted.values)
  mis.validation = inc.validation / length(fit.validation$fitted.values)

  errors[i, "test"] = mis.test
  errors[i, "validation"] = mis.validation
}

plot(1:30, errors[,"test"], col="red", ylim=c(0.005, 0.06))
points(errors[, "validation"], col="blue")
```



From the plot it's hard to tell which value for K is the best when looking at the errors for the validation data, the error is nearly the same for K = 3, 4 and 6. We could say that they are the same and consider using the error for the training data to get the best K which is K = 3. Why I'm only using the values for validation errors in the first case is because these errors matter more because they were trained on some data and evaluated/tested on some other data, that is not the case for the training errors.

```
fit.test = kknn(
  formula = lastcol~.,
  train = data.train,
  test = data.test,
  k = 3,
  kernel = "rectangular"
)
```

```
fit.train = kknn(
  formula = lastcol~.,
  train = data.train,
  test = data.train,
  k = 3,
  kernel = "rectangular"
)

fit.validation = kknn(
  formula = lastcol~.,
  train = data.train,
  test = data.validation,
  k = 3,
  kernel = "rectangular"
)

inc.train = mis(data.train$lastcol, fit.train$fitted.values)
inc.validation = mis(data.validation$lastcol, fit.validation$fitted.values)
inc.test = mis(data.test$lastcol, fit.test$fitted.values)

mis.train = inc.train / length(fit.train$fitted.values)
mis.validation = inc.validation / length(fit.validation$fitted.values)
mis.test = inc.test / length(fit.test$fitted.values)

mis.train
```

```
## [1] 0.009942439
```

```
mis.validation
```

```
## [1] 0.02722513
```

```
mis.test
```

```
## [1] 0.03242678
```

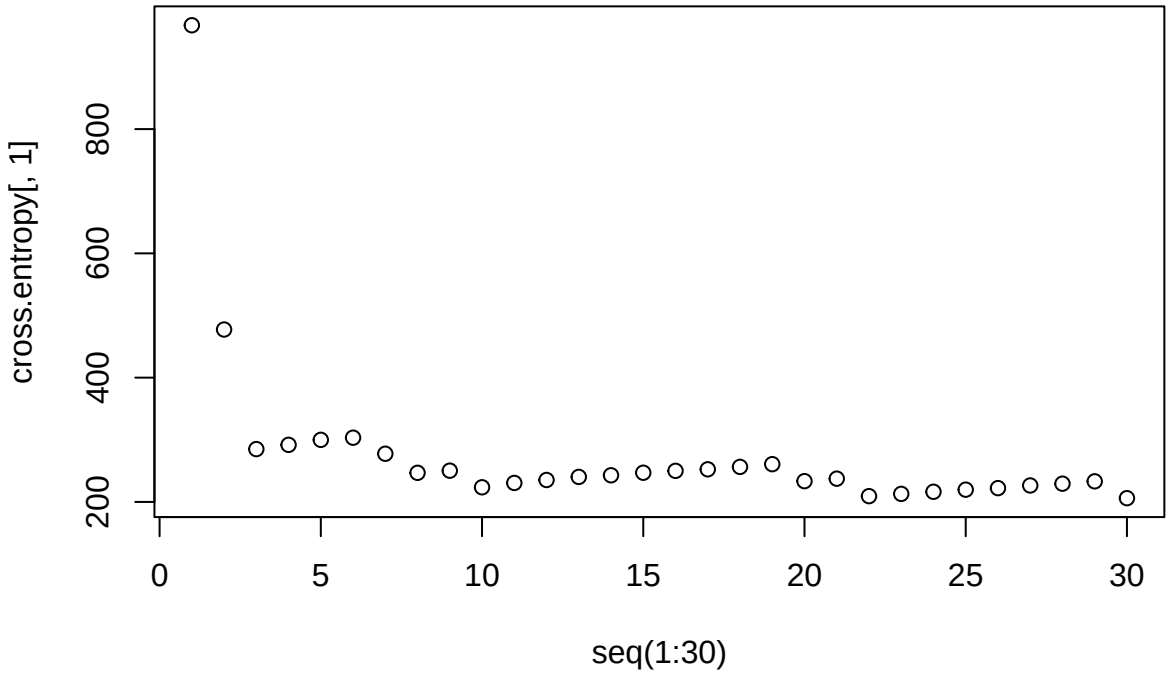
When comparing test data to predictions for our training data we have a much better accuracy for training data, this suggests that our model is over-fitted. Since we have a low K-value together with an overfit it means that the model complexity is high. A higher K-value would give us a less complex model and less of an overfit but this might also lead to an underfit if we go too high on the K-value.

1.5)

Computing the error as cross-entropy and plotting the dependence of the validation error on the value of K.

```
cross.entropy = matrix(data = 0, nrow = 30, ncol = 1, dimnames = list(c(), c("x.entropy")))

for (i in 1:30) {
  fit.validation = kknn(
    formula = lastcol~.,
    train = data.train,
    test = data.validation,
    k = i,
    kernel = "rectangular"
  )
```

```
  for (j in 1:length(data.validation$lastcol)){
    cross.entropy[i,1] = cross.entropy[i, 1]  - log(fit.validation$prob[j, data.validation$lastcol[j]] +
  }
}

plot(seq(1:30), cross.entropy[,1])
```




```
match(min(cross.entropy), cross.entropy) # best K
```

```
## [1] 30
```

When calculating the error using cross-entropy we get the best value for K to be 30. Cross-entropy might be a more suitable choice for the error function in this case because it also uses the probability for it choosing the correct class. When using classification error as function it does not take into account the probability for it being correct, i.e. having a 100% probability of classifying correctly should weigh more than having a less than 100% probability of being correct.

Assignment 2: Linear regression and ridge regression

```
library(caret)
```

```
## Loading required package: ggplot2
## Loading required package: lattice
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

```
library(ggplot2)
```

2.1

```
# read data and create data frame
parkinsons_data = read.csv("parkinsons.csv")
data = data.frame(parkinsons_data)
#remove irrelevant columns
data[, c('subject.','age', 'sex', 'test_time', 'total_UPDRS')] <- list(NULL)

#data = data.frame(x = c(1), y=c(2:ncol(data)))

# partition data into train/test
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=data[id,]
test=data[-id,]

# scale data
scaler=preProcess(train)
trainS=predict(scaler,train)
testS=predict(scaler,test)

# compute matrices for later calculations
Y = as.matrix(trainS$motor_UPDRS)
X_train = as.matrix(trainS[, c(2:ncol(trainS))])
X_test = as.matrix(testS[, c(2:ncol(testS))])
N = nrow(X_train)
```

2.2

```
# create lm object and get model
fit = lm(trainS$motor_UPDRS~.,data=trainS)
summary(fit)
```

```
##
## Call:
## lm(formula = trainS$motor_UPDRS ~ ., data = trainS)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0255 -0.7363 -0.1087  0.7333  2.1960
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    6.575e-15  1.583e-02   0.000 1.000000
## Jitter...      1.869e-01  1.496e-01   1.250 0.211496
## Jitter.Abs.   -1.696e-01  4.081e-02  -4.156 3.32e-05 ***
## Jitter.RAP    -5.270e+00  1.884e+01  -0.280 0.779688
```

```
## Jitter.PPQ5   -7.457e-02  8.778e-02  -0.850 0.395659
## Jitter.DDP     5.250e+00  1.884e+01   0.279 0.780541
## Shimmer        5.924e-01  2.060e-01   2.876 0.004055 **
## Shimmer.dB.   -1.727e-01  1.393e-01  -1.239 0.215380
## Shimmer.APQ3   3.207e+01  7.717e+01   0.416 0.677738
## Shimmer.APQ5  -3.875e-01  1.138e-01  -3.405 0.000669 ***
## Shimmer.APQ11  3.055e-01  6.124e-02   4.989 6.37e-07 ***
## Shimmer.DDA   -3.239e+01  7.717e+01  -0.420 0.674739
## NHR           -1.854e-01  4.557e-02  -4.068 4.85e-05 ***
## HNR           -2.385e-01  3.640e-02  -6.553 6.45e-11 ***
## RPDE           4.068e-03  2.267e-02   0.179 0.857576
## DFA           -2.803e-01  2.014e-02 -13.919  < 2e-16 ***
## PPE            2.265e-01  3.289e-02   6.886 6.75e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9396 on 3508 degrees of freedom
## Multiple R-squared:  0.1212, Adjusted R-squared:  0.1172
## F-statistic: 30.24 on 16 and 3508 DF,  p-value: < 2.2e-16
```

```r
# print(fit$coefficients)

# get train mse
y_hat_train = predict(fit, trainS)
train_MSE = mean((trainS$motor_UPDRS - y_hat_train)^2)
# print(train_MSE) # = 0.8785431

# get test mse
y_hat_test = predict(fit, testS)
test_MSE = mean((testS$motor_UPDRS - y_hat_test)^2)
# print(test_MSE) # = 0.9354477
```

2.3

```r
# log-likelihood
Loglikelihood <- function(theta, sigma) {
  e = X_train %*% theta - Y
  LL = -0.5*N*log(2*pi*sigma^2)-0.5*(1/sigma^2)*(t(e)%*%e)

  return(LL)
}


# function adding ridge to -LL
# Ridge <- function(theta, sigma, lambda) {
Ridge <- function(par, lambda) {
  theta = par[1:16]
  theta = as.matrix(theta)
  sigma = par[17]
  ridge = -Loglikelihood(theta, sigma) + lambda*(t(theta)%*%theta)
  return(ridge)
}
```

```
RidgeOpt = function(lambda) {
  # z = c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
  z = c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
  result = optim(z, fn=Ridge,lambda = lambda, method="BFGS")
  # result = optim(c(0,0), fn=Ridge(theta, sigma, lambda), method="BFGS")
  return(result)
}


DF = function(theta, lambda) {
  I = as.matrix(diag(ncol(X_train)))
  P = as.table(X_train%*%((t(X_train)%*%X_train + lambda*I)^-1)%*%t(X_train))
  df = sum(diag(P))
  # print(df)
  return(df)
}
```

2.4

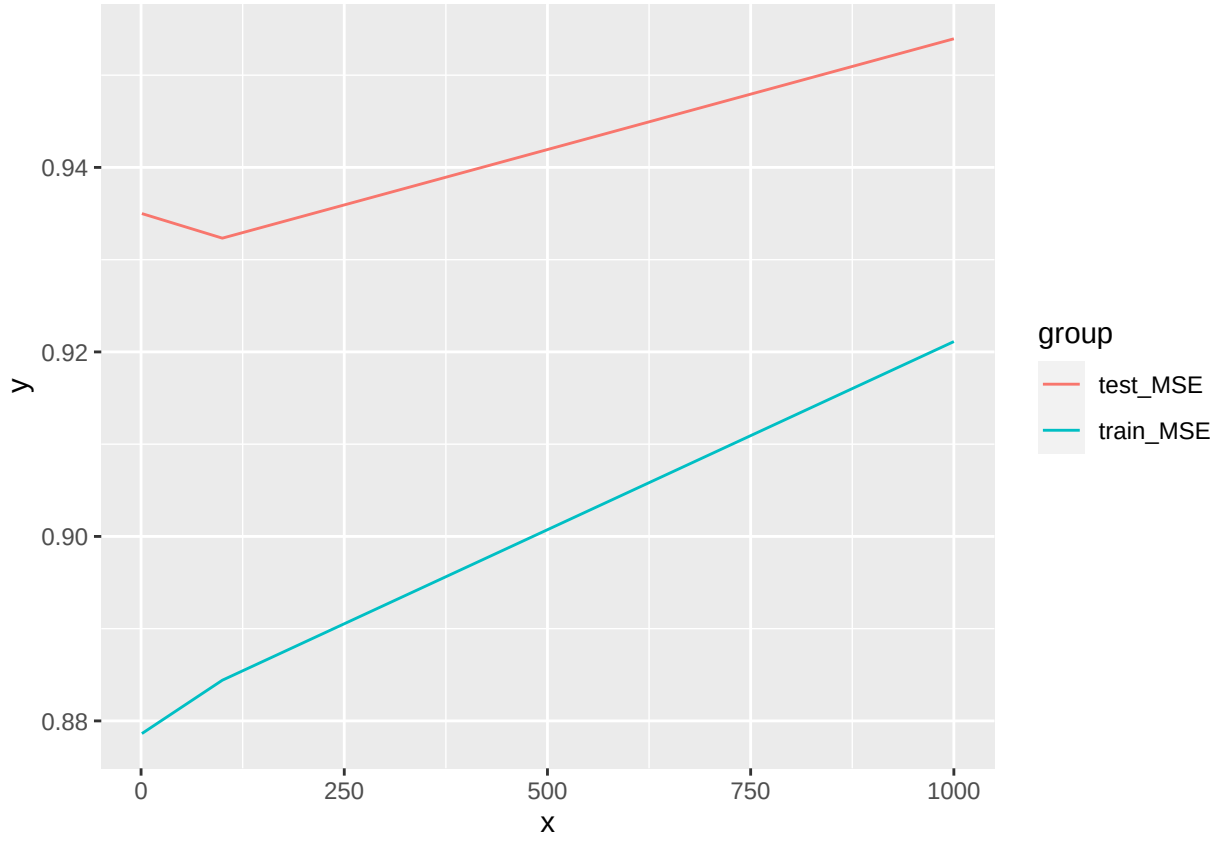
Solution

```
getMSE = function(theta) {
  # get train mse
  y_hat_train = X_train %*% theta
  train_MSE = mean((trainS$motor_UPDRS - y_hat_train)^2)
  # print(train_MSE)

  # get test mse
  y_hat_test = X_test %*% theta
  test_MSE = mean((testS$motor_UPDRS - y_hat_test)^2)
  # print(test_MSE)
  MSE = c(train_MSE, test_MSE)
  return(MSE)
}

# initialaze matrix and counter
M = matrix(data=NA, nrow=3, ncol=4)
i = 1
# loop through all lambda
for (lambda in c(1, 100, 1000))
{
  #get optimal parameters (theta, sigma)
  res = RidgeOpt(lambda)
  # create theta matrix
  par = (res$par)
  theta = par[1:16]
  theta = as.matrix(theta)
  # calculate MSE
  mse = getMSE(theta)
  # get degrees of freedom
  df = DF(theta, lambda)
  # add data to matix
  M[i,1] = lambda
```

```
  M[i,2] = mse[1]
  M[i,3] = mse[2]
  M[i,4] = df
  i = i+1
}

# create data frame
df_ <- as.data.frame(M)
names(df_) <- c("lambda","train_MSE","test_MSE","DF")
df_
```

```
##   lambda train_MSE  test_MSE       DF
## 1      1 0.8786272 0.9349974 255.9955
## 2    100 0.8844109 0.9323309 255.5585
## 3   1000 0.9211205 0.9539466 252.4633
```

```
# Plot data
df_reshaped <- data.frame(x = df_$lambda,
                          y = c(df_$train_MSE, df_$test_MSE),
                          group = c(rep("train_MSE", nrow(df_)),
                                    rep("test_MSE", nrow(df_))))

ggplot(df_reshaped, aes(x, y, col = group)) +  geom_line()
```



Choosing optimal penalty parameters

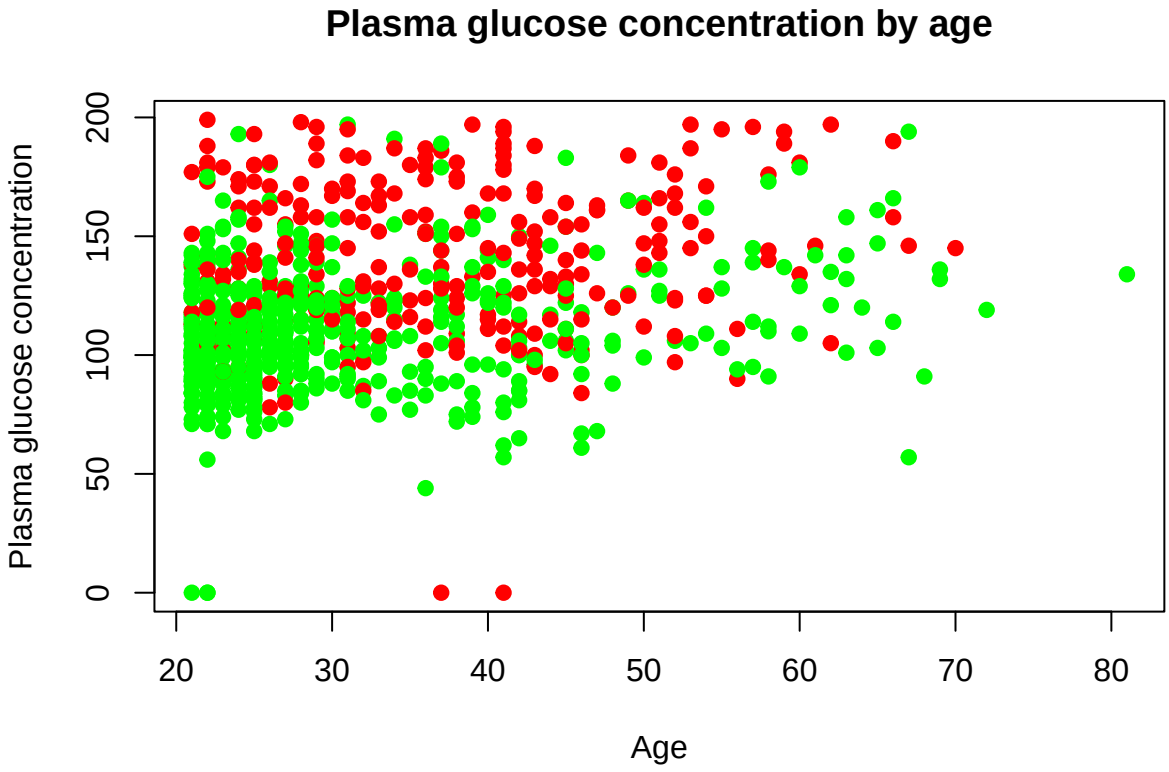
From the ridge function one can see that a quadratic penalty function, a Ridge parameter, is added to the negative maximum likelihood estimate.

When the ridge parameter, λ, is equal to zero the penalty term clearly has no effect and the outcome will be equivalent to the maximum likelihood estimate. When λ increases the Ridge parameter becomes more influential and the θ coefficient estimates approach zero. One advantage of this method is that the coefficients that influence the model the least will approach zero the fastest. This decreases the komplexity if the model and the degrees of freedom decreases.

One can also observe that the training MSE is smaller than for the test data which is reasonable. Also the training MSE increases for increased λ since that decreases the model complexity and the degrees of freedom. However it decreases for the test data up to $\lambda = 100$, and then increases. This is a result of overfitting and the optimal λ is therefor the λ that minimizes the MSE for the test data, $\lambda = 100$.

Assignment 3: Logistic regression and basis functions

3.1



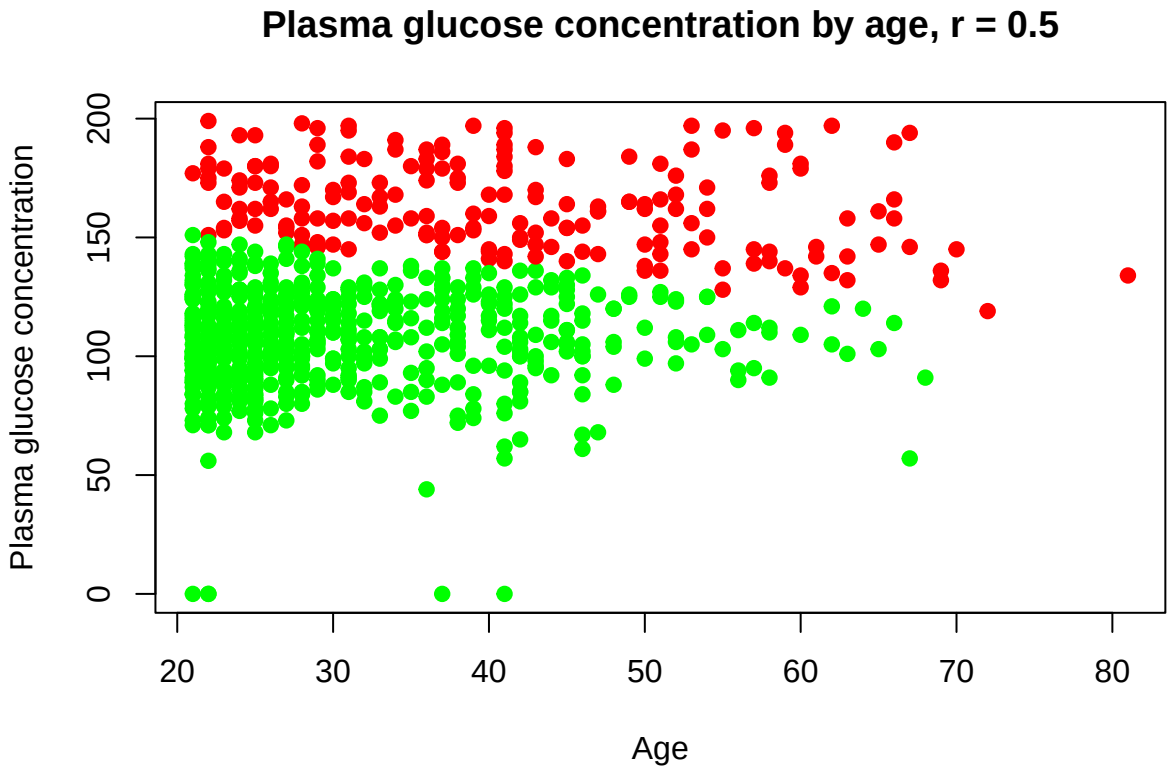
We believe that it is fairly easy to determine if someone has diabetes given their glucose levels using a standard logistic regression. It should be able to correctly classify most cases but there will also be a significant error. The motivation being that a persons glucose levels is a good indicator for diabetes but outliers will lower the amount of correct classifications. Only using the glucose levels will therefore give a good classification but i can probably be better with more information.

3.2

```
##                Estimate  Std. Error    z value     Pr(>|z|)
## (Intercept) -5.89785793 0.462449826 -12.753509 2.980232e-37
## pgc          0.03558250 0.003288130  10.821500 2.722834e-27
## age          0.02450157 0.007379078   3.320411 8.988507e-04
```

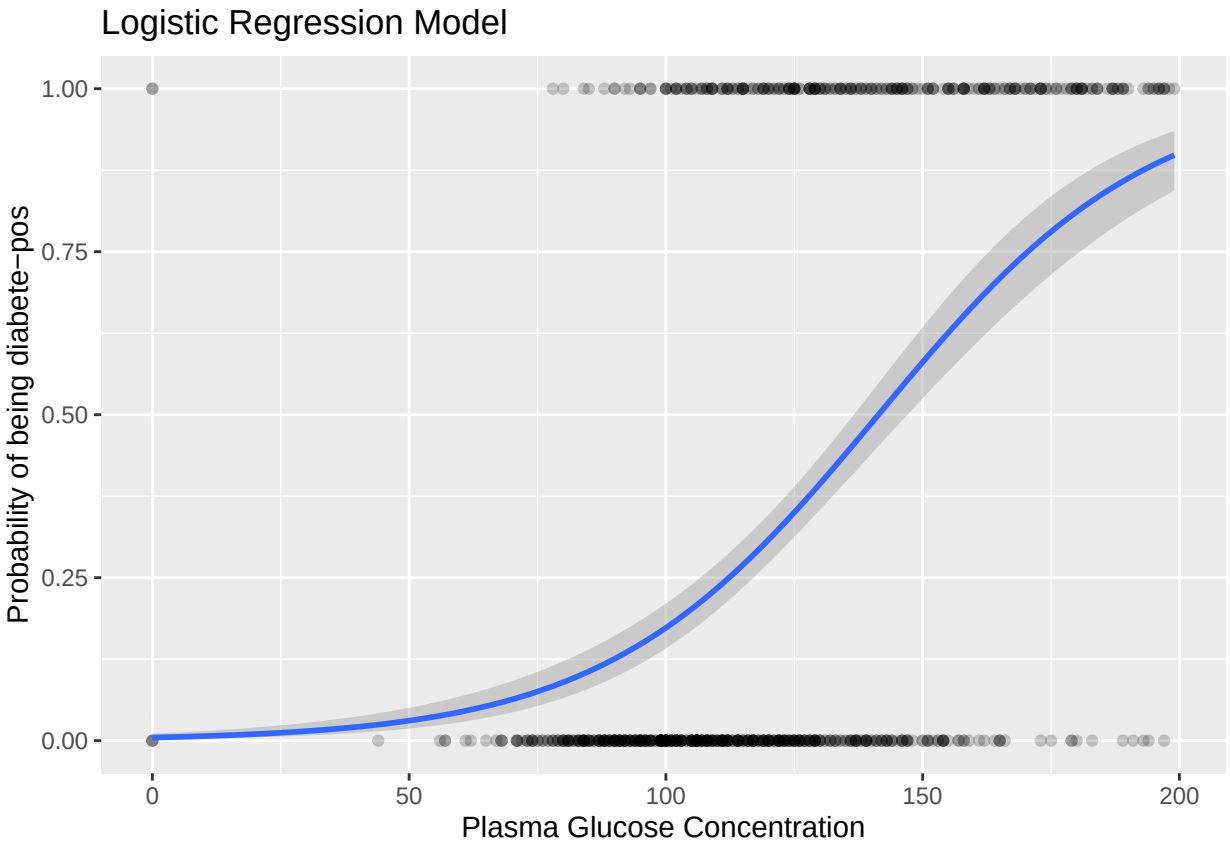
```
## [1] 0.2659713
```

The missclassification error of the model is shown above, 0.27.



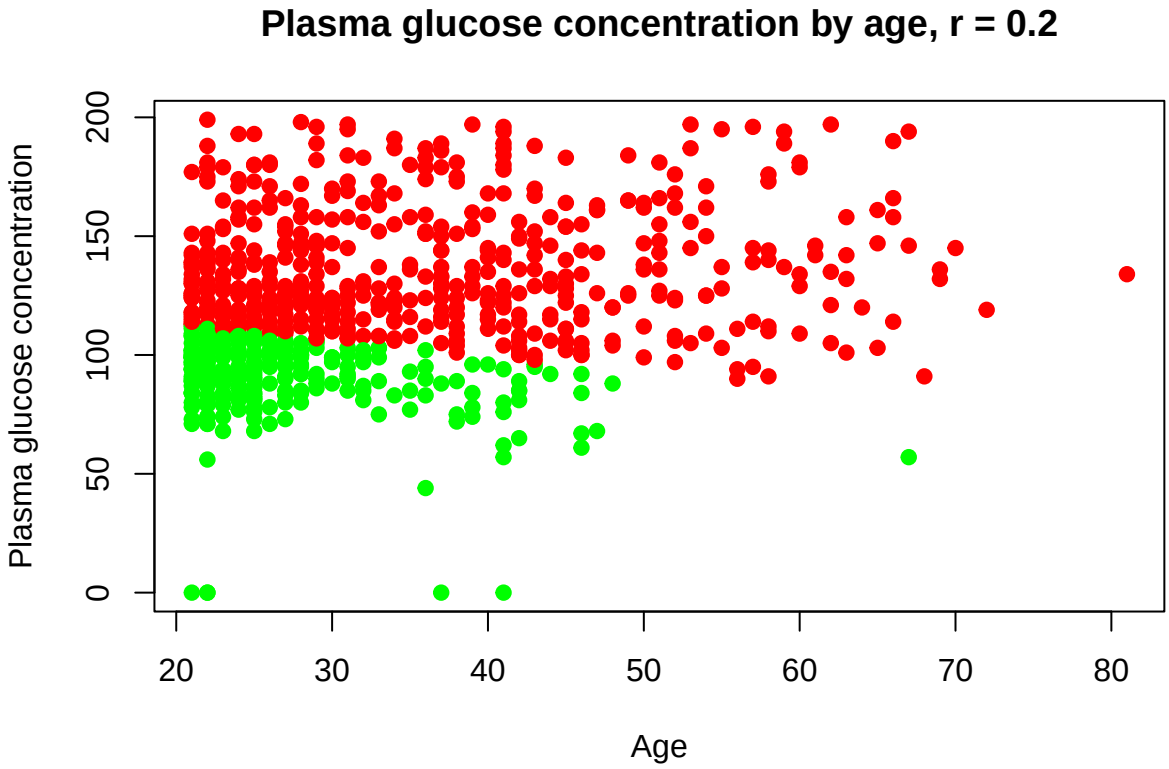
The result of the classification show that the model can classify diabetes at 73% accuracy with only two features which is good given the small amount of features given to the model.

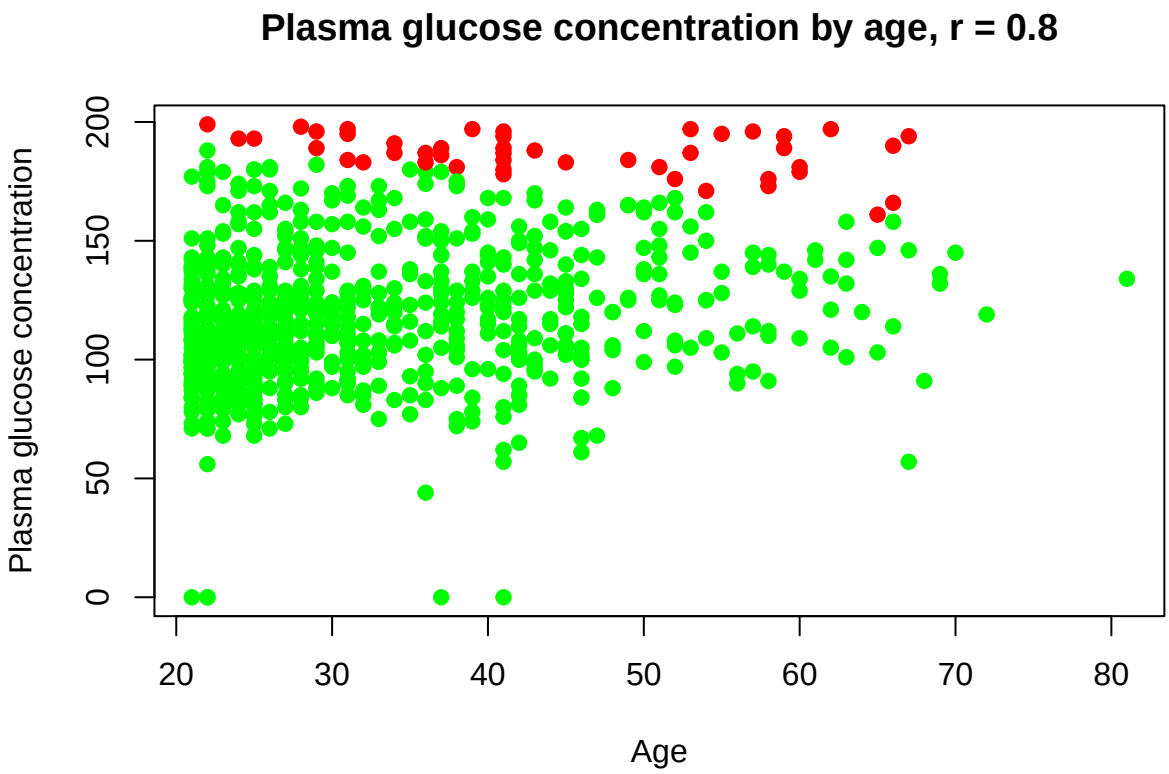
3.3



The model looks to catch the data distribution fairly well. A smaller r would allow more positive cases to be caught but would also make more false positives as the true negatives are denser at a lower r value.

3.4


Plasma glucose concentration by age, r = 0.2
Plasma glucose concentration
0
50
100
150
200
20
30
40
50
60
70
80
Age



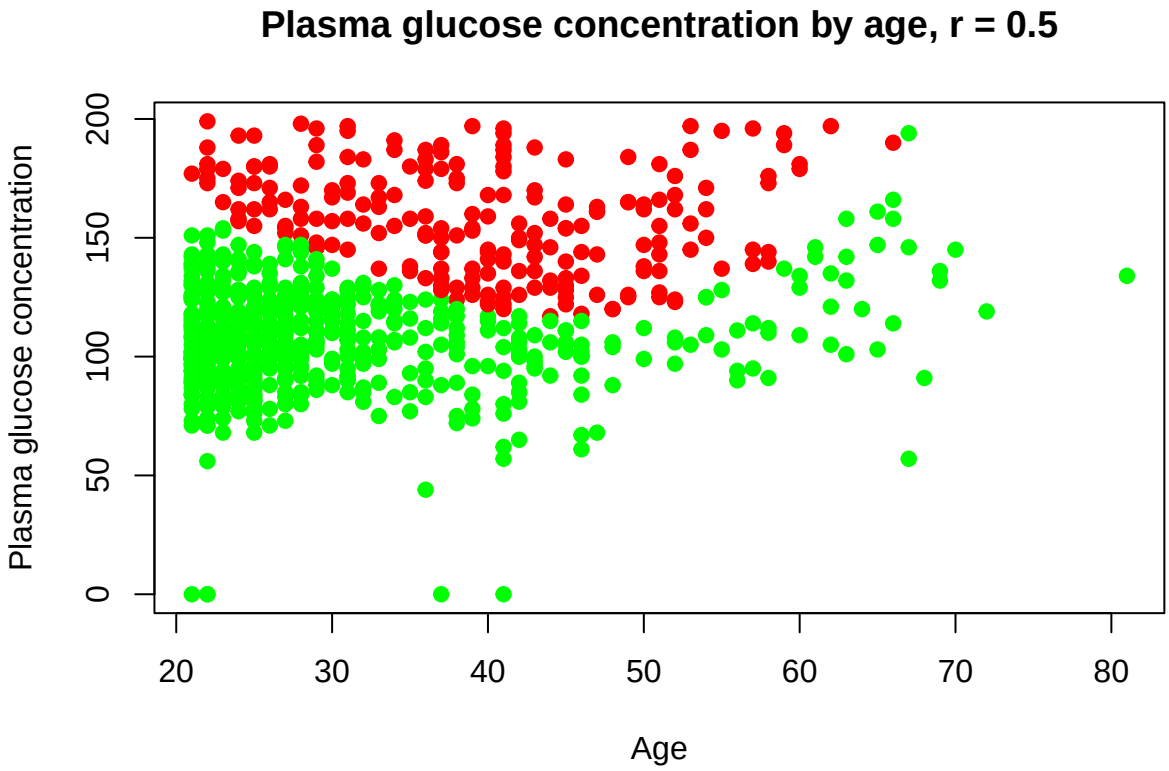
When r = 0.2 the model heavily overclassifies diabetes and there are alot of false positives Contrary when r = 0.8 the model heavily overclassifies false negatives. Depending on what false classification is the worst the value of r can help the model overclassifie to get more true classifications.

3.5

```
##                  Estimate   Std. Error   z value     Pr(>|z|)
## (Intercept) -8.817761e+00 1.110887e+00 -7.937587 2.061521e-15
## pgc          3.198114e-02 9.292847e-03  3.441479 5.785423e-04
## age          1.482157e-01 2.048535e-02  7.235205 4.648281e-13
## z1           5.508121e-09 4.098136e-09  1.344055 1.789305e-01
## z2          -6.268094e-08 4.751906e-08 -1.319070 1.871458e-01
## z3           2.741224e-07 1.712188e-07  1.601006 1.093755e-01
## z4          -5.295576e-07 2.010745e-07 -2.633638 8.447546e-03
```

```
## [1] 0.2411995
```

The error rate for the basis expanded model is show above to be 24%.



The model is slightly more accurate but does not seem to increase the accuracy of the model by much as it went from 27% to 24% in missclassification rate.

Comparing the two plots from the first model and the expanded model it seems the decision boundry was mostly a straight line in the first model but changed more into a curve where it classifies less positives at young and old ages but middle age get more positives.