oblig2

2025-04-23

STK2100 Oblig 2 Bror

Task 2

a.

Load the data

```
wage <- read.csv("https://www.uio.no/studier/emner/matnat/math/STK2100/v25/oblig/wage.csv",header=TRUE)
#head(wage)
#summary(wage) #We can see that the qualitative variables are not properly understood by R

# Make R treat all qualitative factors as qualitative
wage$maritl <- as.factor(wage$maritl)
wage$race <- as.factor(wage$race)
wage$education <- as.factor(wage$education)
wage$jobclass <- as.factor(wage$jobclass)
wage$health <- as.factor(wage$health)
wage$health_ins <- as.factor(wage$health_ins)

summary(wage)
```

```
##       year           age              maritl        race
##  Min.   :2003   Min.   :18.00   Married  :2074   Black: 293
##  1st Qu.:2004   1st Qu.:33.75   Unmarried: 926   Other: 227
##  Median :2006   Median :42.00                    White:2480
##  Mean   :2006   Mean   :42.41
##  3rd Qu.:2008   3rd Qu.:51.00
##  Max.   :2009   Max.   :80.00
##                education             jobclass               health
##  1. < HS Grad      :268   1. Industrial :1544   1. <=Good     : 858
##  2. HS Grad        :971   2. Information:1456   2. >=Very Good:2142
##  3. Some College   :650
##  4. College Grad   :685
##  5. Advanced Degree:426
##
##   health_ins        wage
##  1. Yes:2083   Min.   : 20.09
##  2. No : 917   1st Qu.: 85.38
##                Median :104.92
##                Mean   :111.70
##                3rd Qu.:128.68
##                Max.   :318.34
```

i.

I was a bit unsure on how to do statified sampling on a data set with so many qualitative varibales, so i instead do a random sample, and then afterward check if all all the qualitative variables are present and repeat this until they are.

Got some help from chatgpt from this part especially using function(v)

```
print(length(wage$maritl)) #length of dataset is 3000
```

```
## [1] 3000
```

```
set.seed(420)

valid_split <- FALSE
while(!valid_split) {
    train_index <- sample(1:nrow(wage),size = floor(0.7 * nrow(wage)))
    train <- wage[train_index, ]
    test <- wage[-train_index, ]

    qual_var <- c("maritl","race","education","jobclass","health","health_ins")

    check_all <- sapply(qual_var, function(v) {
        table_train <- table(train[[v]])
        table_test <- table(test[[v]])

        all(table_train >= 154) && all(table_test >= 66) #returs true if there are more t
han 7 in train and 3 in test
        # that specific qualiatative varibale
    })
    #print("iteration")
    #print(check_all) # We can look on how many of the qualitative predictors where distr
ibuted properly
    valid_split <- all(check_all) # returns true if check all only consits of true values
ie, we have enough of the different qualitative variables
}
```

Tried tweaking the values to maximize the spliting of qualitative values to be distributed as equally as possbible following a 70/30 split. I now realize this program was overkill but i thought there was a higher percentage of the qualitative variables not being present # (ii)

ii.

```
fit.lm <- lm(wage ~ .,data = train)
summary(fit.lm)
```

```
##
## Call:
## lm(formula = wage ~ ., data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -97.01 -19.57  -2.80  14.07 211.67
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -2.510e+03  7.444e+02  -3.371 0.000761 ***
## year                         1.290e+00  3.711e-01   3.475 0.000522 ***
## age                          2.754e-01  7.119e-02   3.869 0.000113 ***
## maritlUnmarried             -1.407e+01  1.713e+00  -8.212 3.77e-16 ***
## raceOther                    1.147e+00  3.663e+00   0.313 0.754306
## raceWhite                    5.289e+00  2.546e+00   2.078 0.037853 *
## education2. HS Grad          7.706e+00  2.790e+00   2.762 0.005787 **
## education3. Some College     1.850e+01  2.981e+00   6.208 6.44e-10 ***
## education4. College Grad     3.282e+01  3.003e+00  10.927  < 2e-16 ***
## education5. Advanced Degree  5.488e+01  3.312e+00  16.569  < 2e-16 ***
## jobclass2. Information       4.240e+00  1.583e+00   2.679 0.007440 **
## health2. >=Very Good         6.513e+00  1.717e+00   3.794 0.000152 ***
## health_ins2. No             -1.784e+01  1.690e+00 -10.555  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34.09 on 2087 degrees of freedom
## Multiple R-squared:  0.3409, Adjusted R-squared:  0.3371
## F-statistic: 89.94 on 12 and 2087 DF,  p-value: < 2.2e-16
```

It seems that every covariate other than raceOther(0.754) is significant to a varying degree. Where raceWhite(0.04) follows as the next least significant predictor. While being umarried(3.8e-16) and not having health isurance(2e-16) seems to be the most statistically significant on predicting wage

iii.

```
pred.lm <- predict(fit.lm, newdata=test)
mse.lm <- mean((test$wage - pred.lm)^2)
print(mse.lm)
```

```
## [1] 1149.31
```

2b) (i)

```
library(gam)
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.22-5
```

```
fit.gam <- gam(wage ~ s(year, df=4) + s(age, df=4) + maritl + race + education + jobclass
+ health + health_ins , data = train)
summary(fit.gam)
```

```
##
## Call: gam(formula = wage ~ s(year, df = 4) + s(age, df = 4) + maritl +
##     race + education + jobclass + health + health_ins, data = train)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -97.243 -19.306  -2.669  14.180 212.122
##
## (Dispersion Parameter for gaussian family taken to be 1136.747)
##
##     Null Deviance: 3680381 on 2099 degrees of freedom
## Residual Deviance: 2365571 on 2081 degrees of freedom
## AIC: 20755.9
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                   Df  Sum Sq Mean Sq  F value    Pr(>F)
## s(year, df = 4)    1   19231   19231  16.9174 4.056e-05 ***
## s(age, df = 4)     1  109185  109185  96.0506 < 2.2e-16 ***
## maritl             1  105657  105657  92.9470 < 2.2e-16 ***
## race               2   18908    9454   8.3166 0.0002526 ***
## education          4  763259  190815 167.8604 < 2.2e-16 ***
## jobclass           1   11521   11521  10.1349 0.0014764 **
## health             1   22240   22240  19.5643 1.023e-05 ***
## health_ins         1  111153  111153  97.7819 < 2.2e-16 ***
## Residuals       2081 2365571    1137
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                 Npar Df  Npar F     Pr(F)
## (Intercept)
## s(year, df = 4)       3  0.9165    0.4321
## s(age, df = 4)        3 16.7902 8.877e-11 ***
## maritl
## race
## education
## jobclass
## health
## health_ins
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

ii.

Fitting splines only work for continious predictors as the natural cubic splines consists of multiple polynomials which would not make much sense to fit to non continious predictors.

iii. Well we can look at this question from 2 perspectives Firstly we know that both predictors in their nature that they both increase the same from year to year. This however does not mean that they have a linear relationship with wage. For example we less both before coming of working age, and after retiring. So this can not be modelled linearly. The same can be said for the year, where it does not necessarly follow a linear realtionship. For example during an economic recession where wages dicrease.

Secondly we can see that using a splin on the predictor year is not very significant(0.432 F statistic) so it does not matter if we model it as a spline. And it is therefore usually beneficial to keep the simpler model

In conclusion we can model year linearly as it does not matter much, despite it not necessarily having a linear relationship with wage.

iv. From the summary of fit.gam it seems that all the covariates are significant and where jobless seems to be the least significant with a p value of 0.00147. It therefore seems beneficial to keep all the predictors.

```
pred.gam <- predict(fit.gam, newdata=test)
mse.gam <- mean((test$wage-pred.gam)^2)
print("The mse prediciton error for the gam is")
```

```
## [1] "The mse prediciton error for the gam is"
```

```
print(mse.gam)
```

```
## [1] 1146.427
```

```
print("The mse prediction error for the linear model is")
```

```
## [1] "The mse prediction error for the linear model is"
```

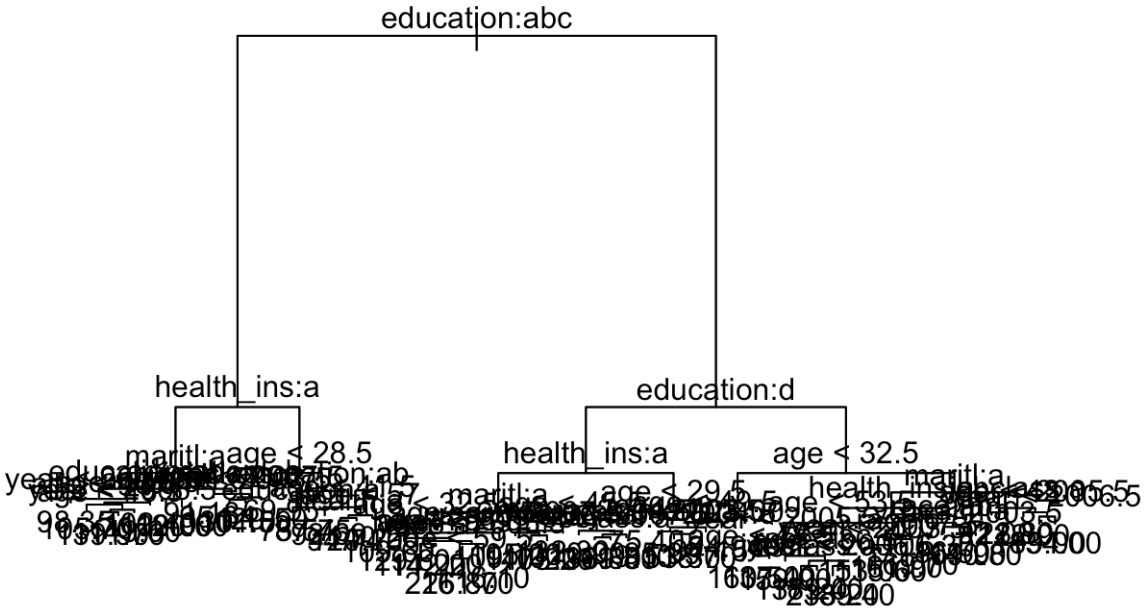
```
print(mse.lm)
```

```
## [1] 1149.31
```

The MSE for the two models are about the same with the MSE for the GAM being slighly better. Since the difference is small i would prefer using the simpler model. It would be beneficial to do a more robust test of the model to get a better estimate of the different. One way we could do this is using cross validation

2c) (i)

```
library(tree)
fit.tree <- tree(wage ~ ., data=train,control=tree.control(nobs=nrow(train),mindev=0.00
1)) #Fit a large tree
plot(fit.tree)
text(fit.tree)
```

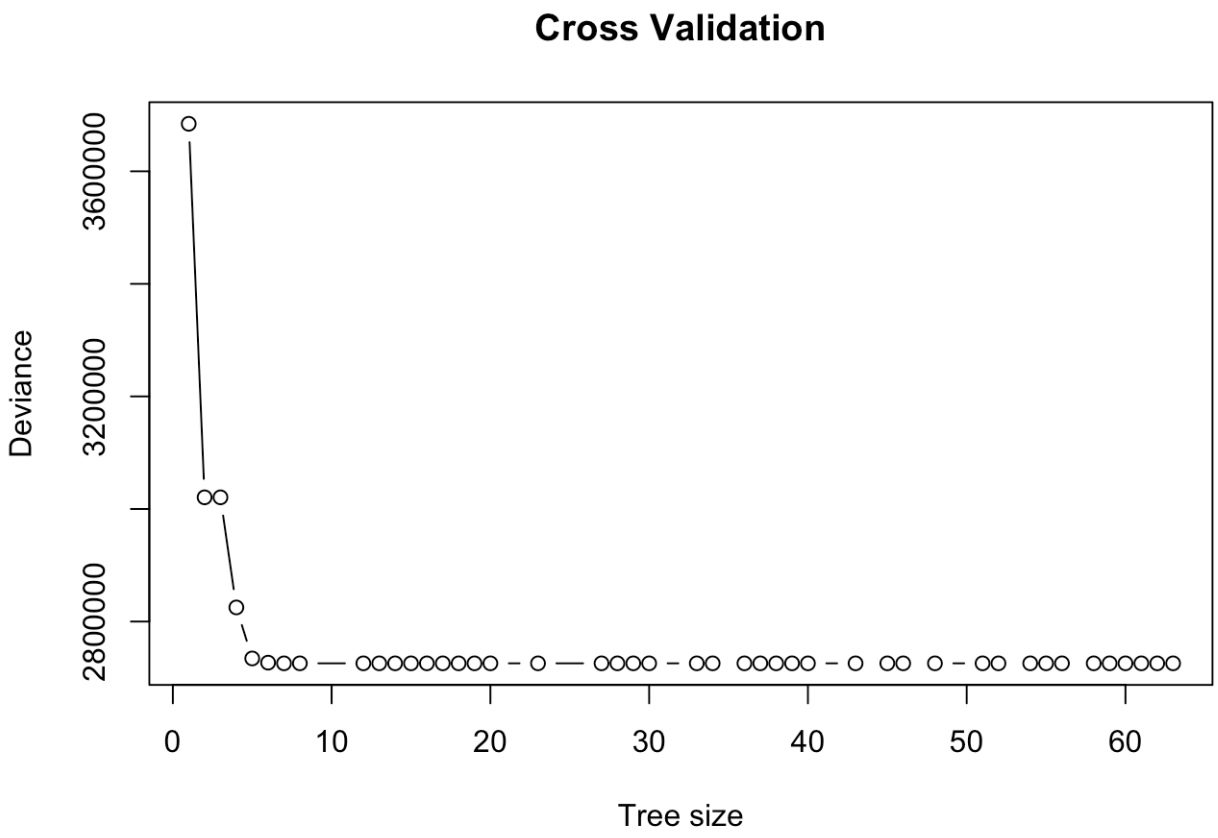


We can see that the splits are mainly in education. The first split is in Some college or less, and collage grad and more. The left tree then splits in whether they have health insurance or not. And the right tree splits on college grad and advanced degree. Which in turn has a left tree that splits further in whether they have health insurance or not. We can see from this tree that people with higher degrees and health insurance are more likely to have a higher wage. This is similar to our previous models where health insurance and education were among the most significant.

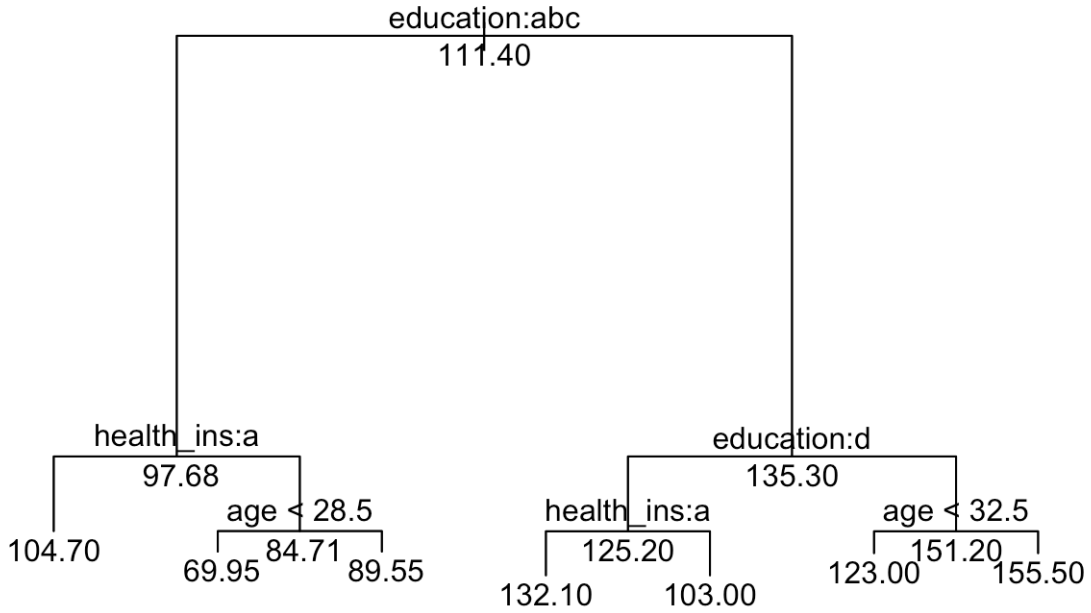
ii.

```
cv.tree.result <- cv.tree(fit.tree, K= 10)

plot(cv.tree.result$size,cv.tree.result$dev, type="b",xlab="Tree size",ylab="Deviance",ma
in="Cross Validation")
```




```
#Its a bit hard to see where the deviance flattens, but it seems to do so around 7.
pruned.tree <- prune.tree(fit.tree,best=7) #If we look closer at the data used for the pl
ot the deviance flattens around 5
plot(pruned.tree)
text(pruned.tree,all=TRUE)
```



#

TODO comment results from tree since change

iii.

```
pred.tree <- predict(pruned.tree,newdata=test)
mse.tree <- mean((test$wage - pred.tree)^2)

print("The mse prediciton error for the gam is")
```

```
## [1] "The mse prediciton error for the gam is"
```

```
print(mse.gam)
```

```
## [1] 1146.427
```

```
print("The mse prediction error for the linear model is")
```

```
## [1] "The mse prediction error for the linear model is"
```

```
print(mse.lm)
```

```
## [1] 1149.31
```

```
print("The mse prediction error for the regression tree is")
```

```
## [1] "The mse prediction error for the regression tree is"
```

```
print(mse.tree)
```

```
## [1] 1278.088
```

```
# We can see that the prediction error for the regression tree is the highest this far. I
would therefore choose the simplest model
# the linear model even though it has a bit more error than our additive model since it i
s small error difference and the linear model
# is by far the simplest
```

We can see that the prediction error for the regression tree is the highest this far. I would therefore choose the simplest model the linear model even though it has a bit more error than our additive model since it is small error difference and the linear model is by far the simplest

Task 3

Load the data

```
vert <- read.csv("https://www.uio.no/studier/emner/matnat/math/STK2100/v25/oblig/vertebra
l-column.csv",header=TRUE)
#head(vert)

vert$class <- as.factor(vert$class)
```

a. i. We can make sure to keep the same proportions of the two classes if we sample from them individually

```
index_0 <- which(vert$class == 0)
index_1 <- which(vert$class == 1)

train_index_0 <- sample(index_0, floor(2/3 * length(index_0)))
train_index_1 <- sample(index_1, floor(2/3 * length(index_1)))
train_index <- c(train_index_0, train_index_1)

train <- vert[train_index,]
test <- vert[-train_index,]
```

ii.

```
fit.logit <- glm(class ~., data=train,family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(fit.logit)
```

```
##
## Call:
## glm(formula = class ~ ., family = binomial, data = train)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  14.692526   4.362108   3.368 0.000757 ***
## pelvInc      30.343962  45.741470   0.663 0.507087
## pelvTilt    -30.245657  45.745118  -0.661 0.508498
## lumbLord      0.003024   0.030750   0.098 0.921659
## SacrSl      -30.476616  45.749167  -0.666 0.505304
## pelvRad      -0.103079   0.029845  -3.454 0.000553 ***
## degrS         0.182972   0.031418   5.824 5.75e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 258.39  on 205  degrees of freedom
## Residual deviance: 107.48  on 199  degrees of freedom
## AIC: 121.48
##
## Number of Fisher Scoring iterations: 8
```

It seems that only degrS and pelVRad along with the intercept are significant for classifying whether the person is healthy or not. While all other predictors have p values over 0.05

iii.

```
pred.prob <- predict(fit.logit, newdata=test,type="response")
pred.logit <- ifelse(pred.prob > 0.5,1,0)
logit.error <- mean(pred.logit != test$class)

print(logit.error)
```

```
## [1] 0.1538462
```

b. i.

```
library("MASS")

fit.lda <- lda(class ~., data=train)
```

ii.

```
pred.lda <- predict(fit.lda, newdata=test)
lda.error <- mean(pred.lda$class != test$class)

print(lda.error)
```

```
## [1] 0.1826923
```

The LDA model assumes a gaussian distribution with linear decision boundries. Our logistic model seems to do slighlty better than lda. This might imply that the gaussian assumption does not fit our data. The simple model therefore makes a better approximation.

#c) (i)

```
fit.qda <- qda(class ~., data=train)
```

ii.

```
pred.qda <- predict(fit.qda, newdata=test)
qda.error <- mean(pred.qda$class != test$class)

print(qda.error)
```

```
## [1] 0.2211538
```

QDA seems to do a even worse job than lda, this may be because the decision border are not nonlinear. Therefore having quadratic boundries may overfit the model to our training data.

d. (i/ii) Normalizing predictors

Ive decided to normalize the data slightly different to the lecture in week 12, the reason is because it is to my understanding bad practice to use data from the test set to train our model, as it can affect our accuracy on showing how good our actually is, since allowing the test data to affect the mean and standard deviation can bias our model.

I instead therefore use the mean and standard deviation of the training data to standardize the test set

```
library("nnet")

train_scaled <- scale(train[, -which(names(train) == "class")]) #We dont normalize the qu
altiative variable since it is not continious
test_scaled <- scale(test[, -which(names(test)=="class")],
    center = attr(train_scaled,"scaled:center"),
    scale = attr(train_scaled,"scaled:scale"))

train.standard <- cbind(class = train$class, as.data.frame(train_scaled)) # Adding the qu
alitative variables back
test.standard <- cbind(class = test$class, as.data.frame(test_scaled))
```

Ive decided to combine (i) and (ii) slightly i want to evaluate right after finishing fitting each model, this way i dont have to do another iteration through every model again to predict and calculate misclassification error. I find this way more clean.

I also dont want to iterate throughn decays and then find the best, and then afterwards iterate through sizes. As i dont think this will necessarily find the best decay/size combination. I therefore iterate through every single combination. This should not be computationally heavy as these are very simple neural networks.

```
sizes <- c(1, 2, 3, 4, 5, 6, 7, 8, 10)
decays <- c(0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1)

results <- expand.grid(size=sizes,decay=decays)
results$error <- NA

for (i in 1:nrow(results)) {
    fit <- nnet(class ~.,data = train.standard, size=results$size[i], decay=results$decay
[i], maxit=500,trace=FALSE)

    pred <- predict(fit, newdata = test.standard)
    pred.class <- ifelse(pred > 0.5,1,0)

    results$error[i] <- mean(pred.class != test.standard$class)
}
ordered.results <- results[order(results$error),]
print(ordered.results[0:7,])
```

```
##    size decay     error
## 21    3 0.010 0.1346154
## 13    4 0.001 0.1442308
## 25    7 0.010 0.1442308
## 32    5 0.050 0.1442308
## 40    4 0.100 0.1442308
## 42    6 0.100 0.1442308
## 43    7 0.100 0.1442308
```

There seem to really be no clear pattern on what decay values und3er 0.1 are the best and there also seem to be no apparent benefit to having a larger network

```
print("The error from the logistic regression is:")
```

```
## [1] "The error from the logistic regression is:"
```

```
print(logit.error)
```

```
## [1] 0.1538462
```

```
print("The error from the LDA is:")
```

```
## [1] "The error from the LDA is:"
```

```
print(lda.error)
```

```
## [1] 0.1826923
```

```
print("The error from the QDA is:")
```

```
## [1] "The error from the QDA is:"
```

```
print(qda.error)
```

```
## [1] 0.2211538
```

```
print("The smallest error for the single layer neural network is:")
```

```
## [1] "The smallest error for the single layer neural network is:"
```

```
print(ordered.results[1,])
```

```
##    size decay     error
## 21    3  0.01 0.1346154
```

The neural network seems to do the best followed by the logistic regression. It seems like making linear assumptions regarding the splitting of the data does well, hence why the lda with linear decision boundries does better than qda. This also applies for logistic regression which is also linear in its boundries. Im a bit confused on how a signle layer neural network does better. As it is also only linear when it doesnt have multiple layers. But there seems to be a high variance in terms of which parameters gets the best test error. As they seem to change with every iteration of the script.

In conclusion the models which makes more linear assumptions does better. Which means non linear fitting by qda, overfits nonlinearly to the training data. Neural networks seems to do the bast, but this may come from us fitting many different models rather than a single layer neural network being superior.