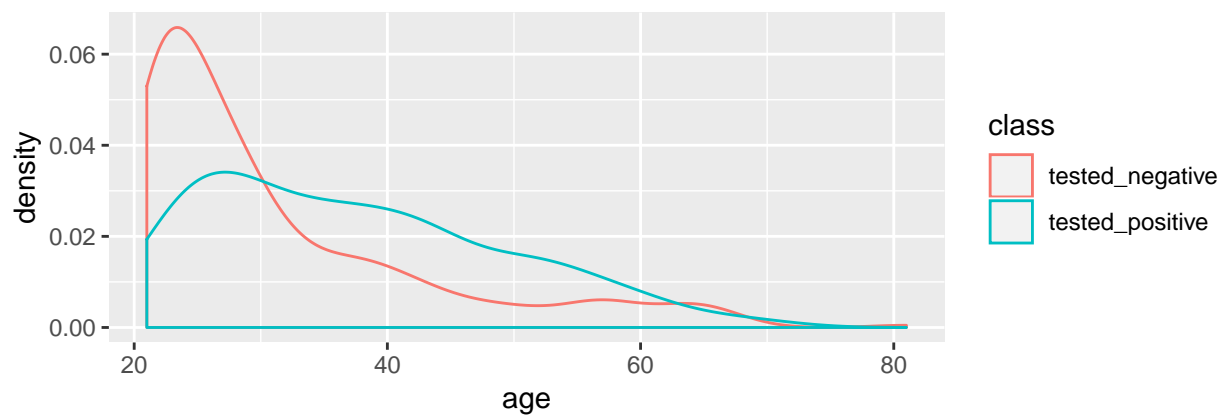
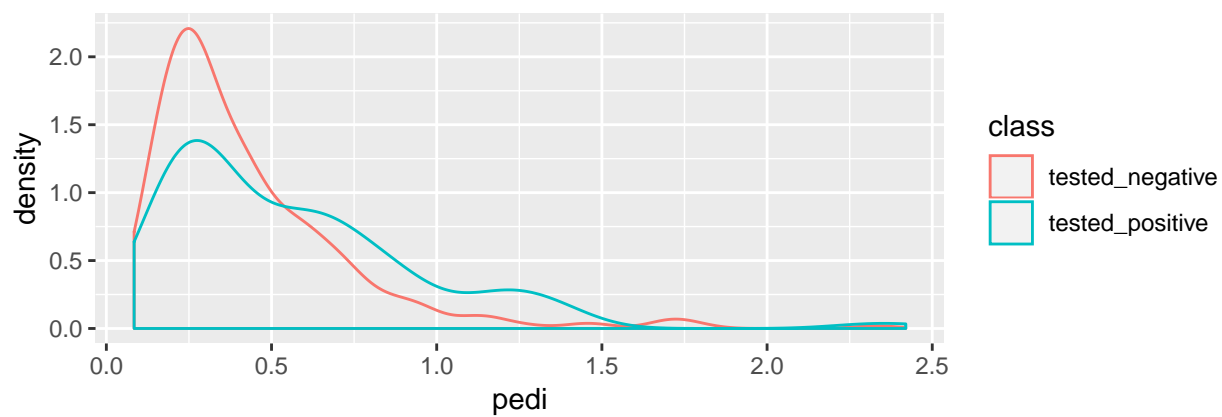


Assignment

Assignment 1

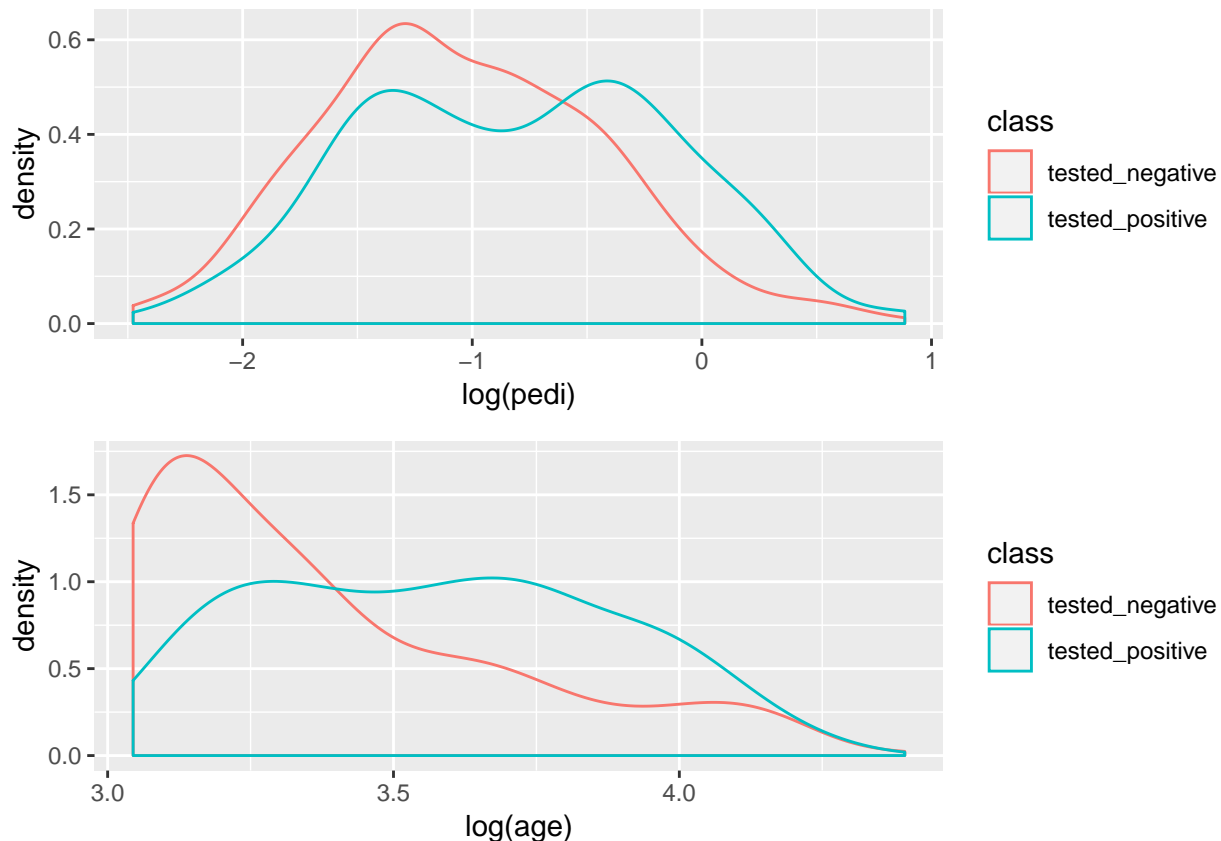
1

```
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-16
## Misclassification on training data: 0.328261
## Misclassification on test data: 0.373377
```



```
## Misclassification on training data: 0.334783
## Misclassification on test data: 0.357143
```

2.5p



The data was split into training and test sets and fit an LDA classifier. The misclassification error for training and test data were respectively 0.328261 and 0.373377. After that two density plots were created for pedi and age separately. The same procedure was made for a LDA classifier as $\log(\text{pedi})$ and $\log(\text{age})$ as features. This gave misclassification errors 0.334783 (training) and 0.357143 (test).

Tested negative and tested positive are more similar in the log-plot. From Misclassification, we can see that the log model was better, giving better result on test data.

2

3

```
## [1] "Variances: "
## 0.2540.2230.1310.1130.0890.0860.0560.048
## First 7 components are needed to exceed 95% variance, giving a total variance of: 95.191678%
##
## Loadings:
##      Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## preg                0.136          0.988
## plas          0.972 -0.130 -0.129 -0.102
## pres          0.127  0.896  0.311 -0.283
## skin                0.376 -0.854  0.281 -0.213
## insu -0.994
## mass                0.128 -0.188          0.973
## pedi                                1.000
```

2p

```
## age          0.162  0.145  0.340  0.901          -0.153
##
##              Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8
## SS loadings   1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
## Proportion Var 0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125
## Cumulative Var 0.125  0.250  0.375  0.500  0.625  0.750  0.875  1.000
```

As seen above, first 7 components are to be used to cover 95% of the variance. Scaling should be used, otherwise the result will be misleading. We are interested in how the data affects the result, not how great it is.

As can be seen above below *Loadings*:-output, The variable with gratest contribution to Component 1 is *insu*.

4

```
## Optimal penalty: 0.300000
```

```
## Number of coefficients: 4
```

0.5p

The output above shows that the optimal penalty was 0.3 and the number of coefficients used were 4. I Had trouble doing a matrix multiplication and couldn't calculate the rest of the task. But the result of the classifier will be pretty bad if compared to the original predictor values. This is because it is trained to output something different than with the original data, as the supposed aswers have been manipulated with normal distribution. A bit of my code for this task can be seen in the appendix.


Assignment 2


Part A

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##      alpha

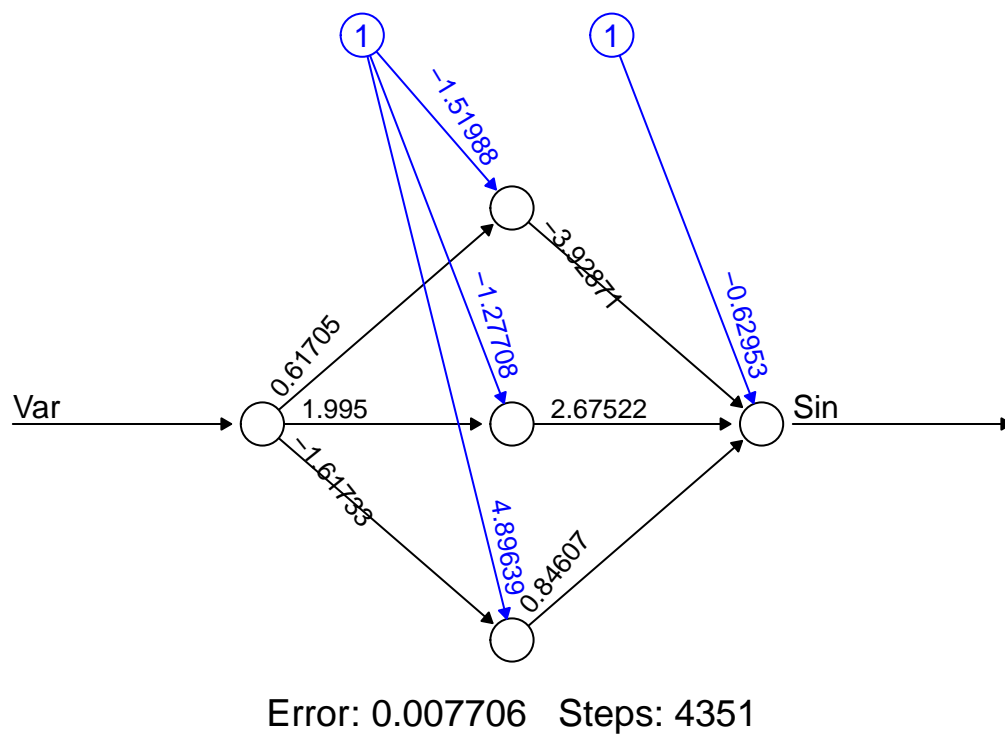
## Error for C=1 : 0.076667
## Error for C=5 : 0.080833
## Generalization error for C=1: 0.078111
```

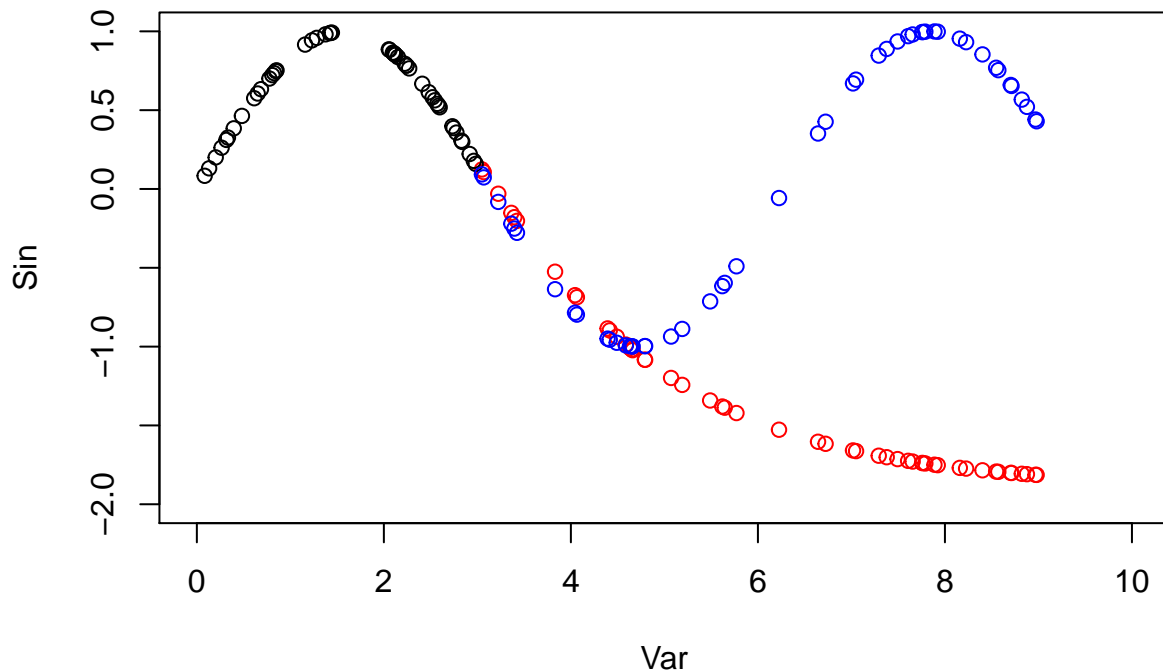
SVMs tries to separate the data with a hyperplane between the two classes. With this hyperplane and the support vectors, the border can be used to classify (separate) new data. 

The hyper parameters (kernel parameters, which is a list containing parameters to be used with the kernel function. These are listed in the R documentation with the *kpar* parameter) and the parameter *C*, which is the cost constraints violation. 

Part B

```
## NN, misclassification on test data: 0.136864
```





The model probably converges to -2 because from what is can see on the training data, the data is decreasing with a decreasing angle.

NNs work by getting an input and the correct answer, and corrects itself after every input so that it adapts it's weights to produce a better answer.

Backpropagation uses some measurement of loss, multiplies that loss with the chain derivative (how much each neuron affects the answer), and now we have information on how each neuron affects the loss.

Appendix

```
library("MASS")
library(ggplot2)
library("gridExtra")
library("stats")
library(glmnet)
library(methods)
setwd("/mnt/rolsi701/tenta")

set.seed(12345)

# Read and split data
diabetes = read.csv("diabetes.csv")
diabetes.pos = diabetes[which(diabetes$class=="tested_positive"),]
diabetes.neg = diabetes[which(diabetes$class=="tested_negative"),]
```

```

len = nrow(diabetes)
index = sample(1:len)
i1 = index[1:floor(len*0.6)]
i2 = index[(floor(len*0.6)+1):len]
tr = diabetes[i1,]
te = diabetes[i2,]

model = lda(formula=class~pedi+age, data=tr)
# Training
tr.pred = predict(model, tr)
tr.misclass = mean(tr$class != tr.pred$class)
message(sprintf("Misclassification on training data: %f", tr.misclass))
# Test
te.pred = predict(model, te)
te.misclass = mean(te$class != te.pred$class)
message(sprintf("Misclassification on test data: %f", te.misclass))
# Density plot
plot1 = ggplot(tr, aes(pedi, color=class))+geom_density()
plot2 = ggplot(tr, aes(age, color=class))+geom_density()
grid.arrange(plot1, plot2)

# With log
model = lda(formula=class~log(pedi)+log(age), data=tr)
# Training
tr.pred = predict(model, tr)
tr.misclass = mean(tr$class != tr.pred$class)
message(sprintf("Misclassification on training data: %f", tr.misclass))
# Test
te.pred = predict(model, te)
te.misclass = mean(te$class != te.pred$class)
message(sprintf("Misclassification on test data: %f", te.misclass))
# Density plot
plot1 = ggplot(tr, aes(log(pedi), color=class))+geom_density()
plot2 = ggplot(tr, aes(log(age), color=class))+geom_density()
grid.arrange(plot1, plot2)

# Answer: Tested negative and tested positive are more similar in the log-plot. From Misclassification,
# the log model was better, giving better result on test data, more general model (since training was w

# 2

# 3
pca.fit = prcomp(subset(tr, select = -class), scale.=F)
screeplot(pca.fit) # to get all eigen values
pca.fit = prcomp(subset(tr, select = -class), scale.=T)
screeplot(pca.fit) # to get all eigen values
# Get eigen values (standard deviation ^ 2)
lambda = pca.fit$sdev^2
message(sprintf("Variances: %2.3f", lambda/sum(lambda)))
message(sprintf("First 7 components are needed to exceed 95% variance, giving a total variance of: %f",
100*sum((lambda/sum(lambda))[1:7])))
PC1 = pca.fit$x[,1]
PC2 = pca.fit$x[,2]

```

```

pca.fit = princomp(subset(tr, select = -class))
print(loadings(pca.fit))

# Answer: Scaling should be used, otherwise the result will be misleading. We are interested in how the
# not how great it is.

# 4
model=cv.glmnet(as.matrix(subset(tr, select = c(-mass, -class))),
               tr$mass,
               lambda = seq(0, 2, 0.1),
               family="gaussian")
message(sprintf("Optimal penalty: %f", model$lambda.min))
#plot(model)
coeffs = coef(model, s=model$lambda.min)
message(sprintf("Number of coefficients: %i", sum (coeffs[2:8,] != 0)))

# Apply noise
tr.noise = tr
tr.noise$mass = dnorm(tr.noise$mass, mean = 0, sd=7)

for (l in seq(0, 2, 0.1)) {
  model=glmnet(as.matrix(subset(tr, select = c(-mass, -class))),
              tr$mass,
              lambda = 0,
              family="gaussian")

  t(model$beta)
  as.matrix(subset(te, select = c(-mass, -class)))
  # Matrix multiplication????
  pred = (model$beta)%*%as.matrix(subset(te, select = c(-mass, -class)))
}

library("kernlab")

# Part A

data(spam)
set.seed(12345)
len = dim(spam)[1]
index=sample(1:len)
tr = spam[index[1:2300], ]
va = spam[index[2301:3500], ]
te = spam[index[3501:len], ]

C = c(1, 5)
for (c in 1:2) {
  ksvm.fit = ksvm(as.factor(type)~., data = tr, kpar = list(sigma=0.05), C=C[c], kernel="rbfdot")
  pred = predict(ksvm.fit, va)
  t = table(pred, as.factor(va$type))
  message(sprintf("Error for C=%i : %f", C[c], (t[1,2]+t[2,1])/sum(t)))
}

ksvm.fit = ksvm(as.factor(type)~., data = tr, kpar = list(sigma=0.05), C=1, kernel="rbfdot")
pred = predict(ksvm.fit, te)
t = table(pred, as.factor(te$type))

```

```

message(sprintf("Generalization error for C=1: %f", (t[1,2]+t[2,1])/sum(t)))

# SVMs tries to separate the data with a hyperplane between the two classes. With this hyperplane and t
# can be used to classify new data.

# The hyper parameters (kernel parameters, which is a list containing parameters to be used with the
# kernel function. These are listed in the R documentation with the kpar parameter) and
# the parameter C, which is the cost constraints violation.

# Part B

library(neuralnet)
set.seed(1234567890)
Var = runif(50, 0, 3)
tr = data.frame(Var, Sin=sin(Var))
Var = runif(50, 3, 9)
te = data.frame(Var, Sin=sin(Var))

# Random initialization of the weights in the interval [-1, 1]
winit = runif(10, -1, 1)
# Create model
nn = neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3), startweights = winit)
comp = predict(nn, te)
mse = sum((comp - te$Sin)^2) / nrow(va)
message(sprintf("NN, misclassification on test data: %f", mse))

# Plot
plot(nn, ylab="Neural network", rep="best")
prediction(nn)$rep1
# Plot of the predictions (black dots) and the data (red dots)
pred = data.frame(Var=te$Var, Sin=comp)
plot(pred, col = "red", ylim = c(-2, 1), xlim = c(0, 10))
#points(prediction(nn)$rep1, col = "red")
points(tr, col = "black")
points(te, col = "blue")

# Answer:

# Answer: They get an input and the correct answer, and corrects itself after every input so that it ad
# a better answer.

# Answer: Use some measurement of loss, multiply that loss with the chain derivative (how much each neu
# now we have information on how each neuron affects the loss.

```

