Question 7

A)

$$\text{Laplace}(x|\mu,\alpha) = \frac{\alpha}{2} e^{-\alpha|x-\mu|}$$

negative log: $\left(-\log(x) = \log\left(\frac{1}{x}\right)\right)$

$$\log\left(\frac{1}{\frac{\alpha}{2} e^{-\alpha|x-\mu|}}\right)$$

Differentiate

$$\frac{d}{dx}\left[\log\left(\frac{1}{\frac{\alpha}{2} e^{-\alpha|x-\mu|}}\right)\right]$$

$$= \frac{\alpha}{2e^{\alpha|x-\mu}} \cdot \frac{d}{dx}\left[\frac{2e^{\alpha|x-\mu|}}{\alpha}\right]$$

$$= \frac{\frac{2}{\alpha} \cdot \frac{d}{dx}\left[e^{\alpha|x-\mu|}\right] \cdot \alpha e^{-\alpha|x-\mu|}}{2}$$

$$= e^{\alpha|x-\mu|} \cdot \frac{d}{dx}\left[\alpha|x-\mu|\right] \cdot e^{-\alpha|x-\mu|}$$

$$= \alpha \cdot \frac{d}{dx}\left[|x-\mu|\right]$$

$$= \alpha \cdot \frac{x-\mu}{|x-\mu|} \cdot \frac{d}{dx}[x-\mu]$$

$$= \frac{\alpha \cdot \left(\frac{d}{dx}[x] + \frac{d}{dx}[-\mu]\right)(x-\mu)}{|x-\mu|}$$

$$= \frac{\alpha \cdot (1+0)(x-\mu)}{|x-\mu|} = \frac{\alpha \cdot (x-\mu)}{|x-\mu|}$$

$|x-\mu|$ $|x-\mu|$

1 B)

$$\log\left[p(y|\vec{x};\vec{w})\,p(\vec{w})\right]$$

$$= \log\left[p(y|\vec{x};\vec{w})\right] + \log\left(p(\vec{w})\right)$$

$$= \log\left[p(y|\vec{x};\vec{w})\right] + \log\left[\prod_{n=1}^{R} p(w_n)\right]$$

$$= \log\left[p(y|\vec{x};\vec{w})\right] + \log\left[\prod_{n=1}^{R} \frac{\alpha}{2} e^{-\alpha|x|}\right]$$

$$= \log\left[p(y|\vec{x};\vec{w})\right] + \sum_{n=1}^{R} \log\left(\frac{\alpha}{2} e^{-\alpha(x)}\right)$$

7c) $\max \left[p(y \mid \vec{x}; \vec{w}) \, p(\vec{w}) \right]$

$= \max \left[\log \left(p(y \mid \vec{x}; \vec{w}) \right) + \log \left(p(\vec{w}) \right) \right]$

This corresponds to L1 regularization because $p(\vec{w}) = \frac{\alpha}{2} e^{-\alpha|x|}$ has an absolute value.

Assignment 5

Sebastian Doka

2023-03-30

Question 2

A)

```
coris = read.csv("coris.dat")
coris[["id"]] = NULL # remove "row.names" column
# Normalize the numeric columns:
normalise = function(x) {
return ((x - mean(x)) / sd(x)) }
normcoris <- as.data.frame(lapply(coris, normalise))
# use the unnormalized versions of the binary variables:
normcoris$famhist <- coris$famhist
normcoris$chd <- coris$chd # this is the response variable
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
## Loading required package: lattice
```

```
set.seed(1)
ind <- createDataPartition(normcoris$chd, p = .6, list=FALSE)
train = normcoris[ind,]
test = normcoris[-ind,]



model <- glm(chd ~ ., data=train, family=binomial)
summary(model)
```

```
##
## Call:
## glm(formula = chd ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -1.8537  -0.7747  -0.4293   0.8638   2.5509
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.15474    0.21026  -5.492 3.97e-08 ***
## sbp          0.03015    0.15495   0.195  0.84572
## tobacco      0.37981    0.15889   2.390  0.01683 *
## ldl          0.44514    0.16659   2.672  0.00754 **
## adiposity    0.15991    0.30610   0.522  0.60139
## famhist      0.75893    0.29746   2.551  0.01073 *
## typea        0.30349    0.15524   1.955  0.05058 .
## obesity     -0.30672    0.25298  -1.212  0.22535
## alcohol      0.13022    0.13808   0.943  0.34561
## age          0.72812    0.22941   3.174  0.00150 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 359.61  on 277  degrees of freedom
## Residual deviance: 281.52  on 268  degrees of freedom
## AIC: 301.52
##
## Number of Fisher Scoring iterations: 5
```

```
yprobs = predict(model, newdata=test, type='response')

ypred = round(yprobs)
table(ypred, test$chd)
```

```
##
## ypred  0  1
##     0 96 29
##     1 25 34
```

```
error_rate = mean(ypred != test$chd)
error_rate
```

```
## [1] 0.2934783
```

The variable with the most effect is the famhist because it has the highest estimated value(0.75893) meaning it has the most effect on the variable chd.

The miscalculation rate is: 0.2934783

```
library (randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.1.3
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed (1)
rf.coris <- randomForest (chd ~ ., data = train,mtry=6, importance = TRUE)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values.  Are you sure you want to do regression?
```

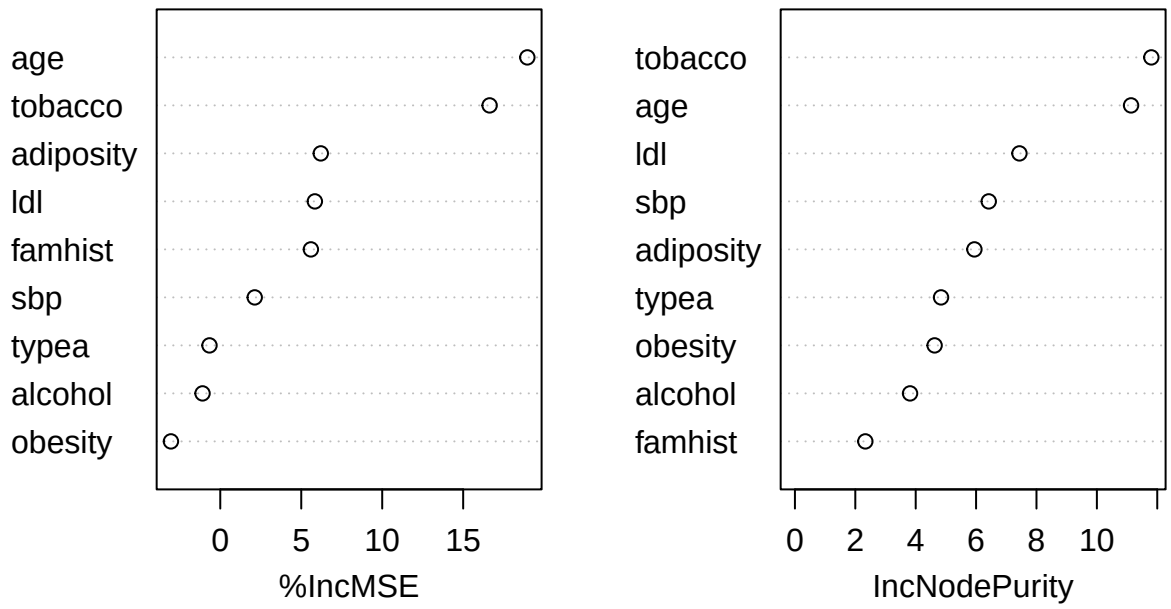
```
importance(rf.coris)
```

```
##               %IncMSE IncNodePurity
## sbp         2.1265043      6.418943
## tobacco    16.6329001     11.804149
## ldl         5.8404232      7.435982
## adiposity   6.2063156      5.943725
## famhist     5.5952444      2.331843
## typea      -0.6661421      4.838857
## obesity    -3.0476706      4.626176
## alcohol    -1.0952868      3.812399
## age        18.9628772     11.132201
```

```
yprobs = predict(rf.coris, newdata=test, type='response')
ypred = round(yprobs)
miscalulation_rate = mean(ypred != test$chd)
print(paste("The miscalculation rate is: ",miscalulation_rate))
```

```
## [1] "The miscalculation rate is:  0.33695652173913"
```

```
varImpPlot(rf.coris)
```



The variable with the greatest importance is tobacco with an importance of 16.6329001. The miscalculation rate is: 0.33695652173913.

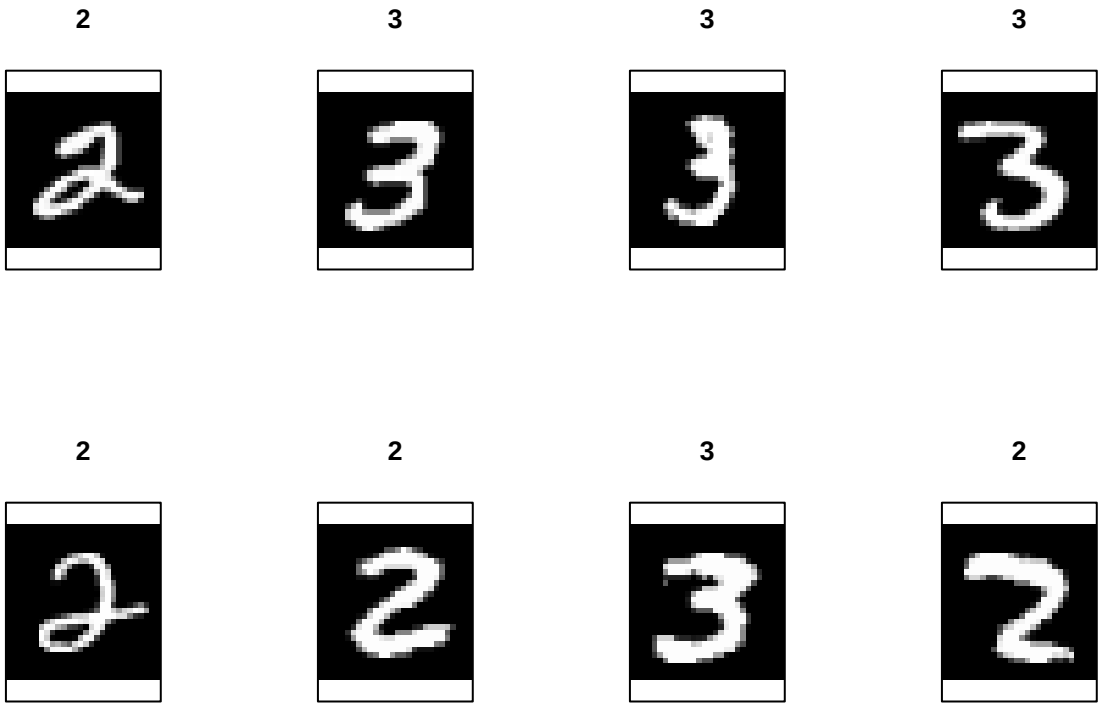
Question3)

```
load('mnist23small.Rdata') # 1000 training digits, 1000 test
# When first loaded, the y column in each data frame is of type integer
# and takes values 1 or 2 (the digit shown in the image).
# Convert these integer variables into nominal variables, i.e. factors.
# This is needed by some classification algorithms
test$y = as.factor(test$y)
train$y = as.factor(train$y)
# "Normalise" the intensity of each pixel
normalise <- function(x) {
return(x / 255)
}
train[,-1] = lapply(train[, -1], normalise)
test[,-1] = lapply(test[,-1], normalise)
# Note that "normalise" often means: centre (i.e. subtract the mean), and
# scale to variance 1, which is not what we've done here, but it's similar.



showdigit = function(imrow, label) {
im = matrix(imrow, nrow=28)[,28:1] #reverse
image(im, col=gray((0:255)/255),
xaxt='n', yaxt='n', main=label, asp = 1
```

```
)
}
par(mfrow=c(2,4))
for(i in 1:8){
showdigit(as.matrix(train[i, 2:785]), paste(train$y[i]))
}
```



A)KNN with K=30

```
library(class)
```

```
## Warning: package 'class' was built under R version 4.1.3
```

```
library(caret)

Xtrain = train[,2:785]
Xtrain_matrix = data.matrix(Xtrain)
Xtest = test[,2:785]
Xtest_matrix = data.matrix(Xtest)

Ntest = 1000
ypred = knn(Xtrain_matrix , Xtest_matrix ,train$y, k = 30, prob = TRUE)


error.rate = function(ypred, ytrue){
```

```
err.rate = mean(ypred != ytrue)
return(err.rate)
}
print(paste('Error rate = ', error.rate(ypred,test$y)))
```

```
## [1] "Error rate =  0.026"
```

B)

```
library(randomForest)
set.seed(1)
bag.mnist <- randomForest(train$y ~ ., data = train, mtry = ncol(train)-1, importance = TRUE)

ypred <- predict (bag.mnist , newdata = test)

print(paste('Error rate = ', error.rate(ypred,test$y)))
```

```
## [1] "Error rate =  0.053"
```

C)

```
set.seed(1)
rf.mnist <- randomForest(train$y ~ ., data = train, mtry = sqrt(ncol(train)-1), importance = TRUE)

ypred <- predict (rf.mnist , newdata = test)

print(paste('Error rate = ', error.rate(ypred,test$y)))
```

```
## [1] "Error rate =  0.021"
```

D)

```
library(nnet)
mnist.nn2 <- nnet(train$y ~ ., data = train, size = 10,rang = 0.01,
decay = 0.01, maxit = 50,MaxNWts = 7862)
```

```
## # weights:  7861
## initial  value 693.167695
## iter  10 value 50.936814
## iter  20 value 4.752770
## iter  30 value 2.954903
## iter  40 value 2.449441
## iter  50 value 2.335438
## final  value 2.335438
## stopped after 50 iterations
```

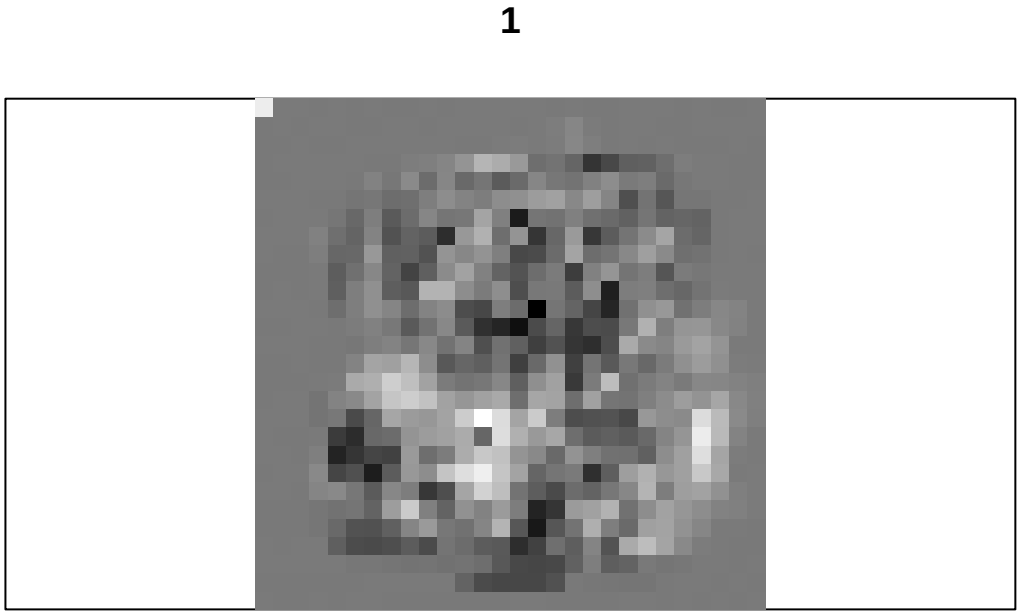
```
ypred <- predict (mnist.nn2 , newdata = test, type="class")

print(paste('Error rate = ', error.rate(ypred,test$y)))
```

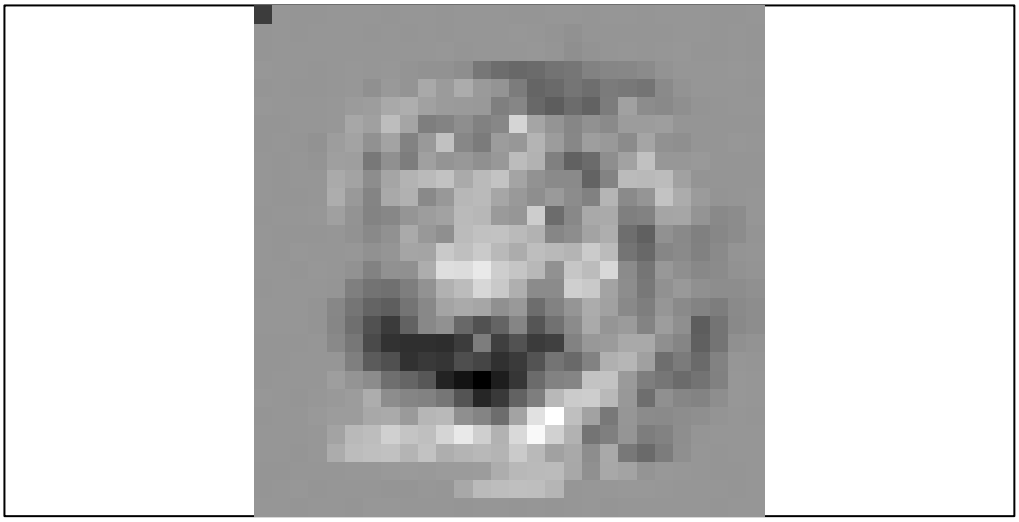
```
## [1] "Error rate =  0.036"
```

E)

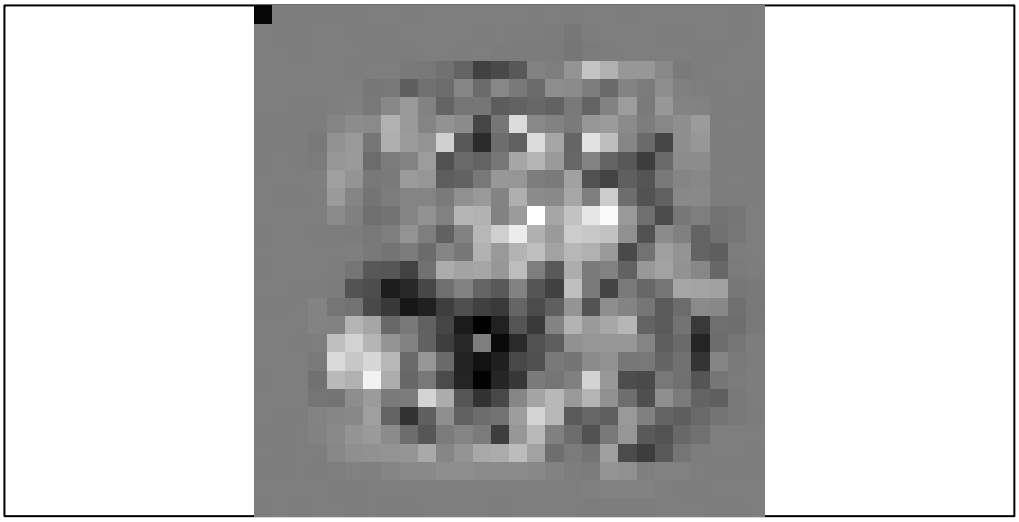
```
for (i in 1:10){
w=mnist.nn2$wts
recep = w[(785*(i-1)+1):(785*(i-1)+784)]
showdigit(as.matrix(recep), paste(i))}
```



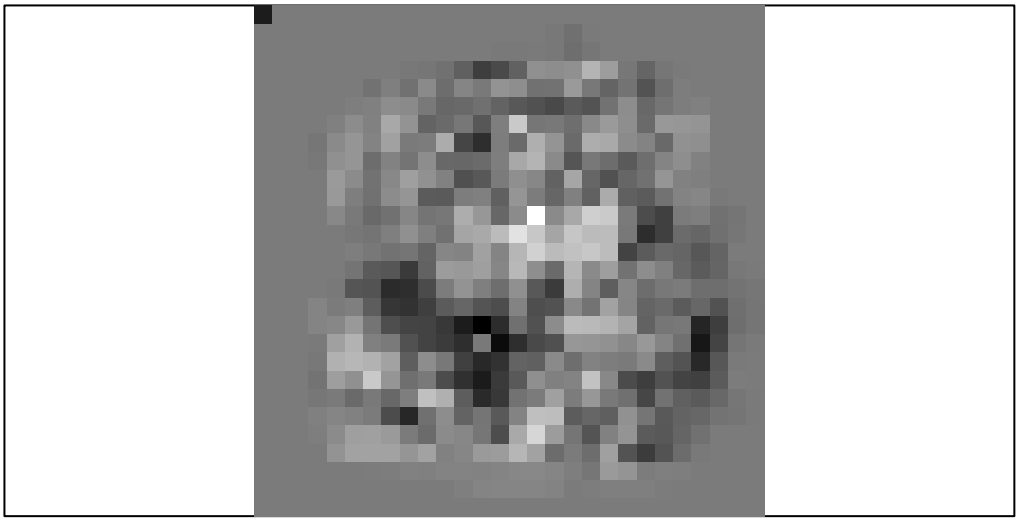
2



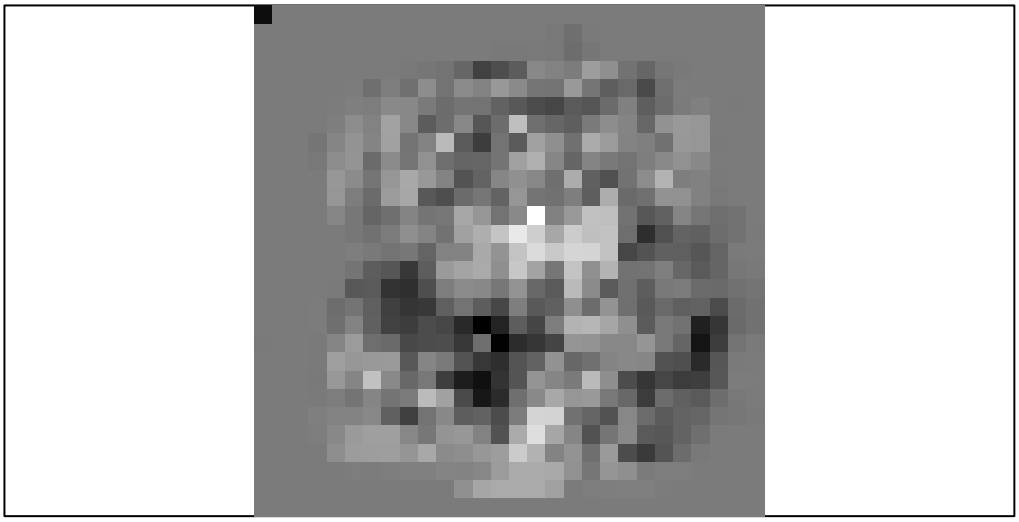
3



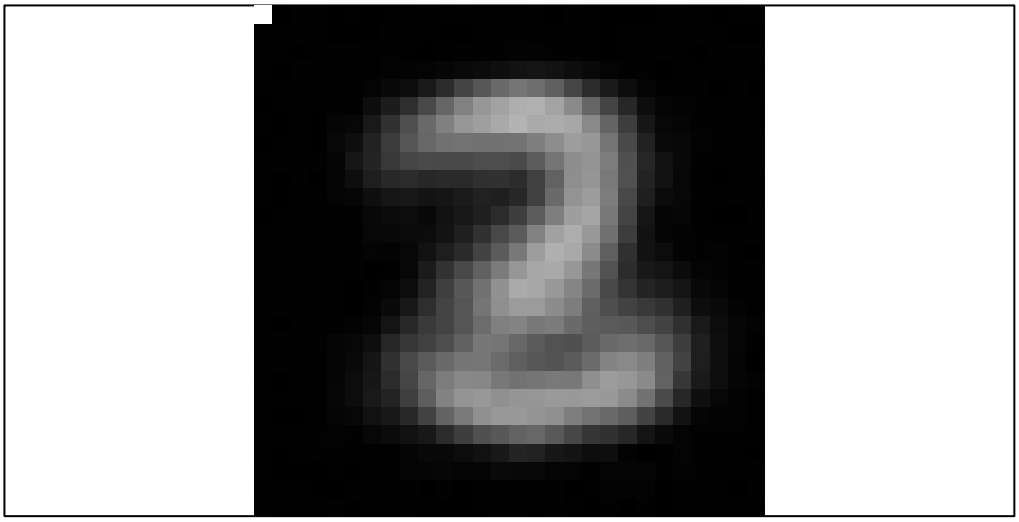
4



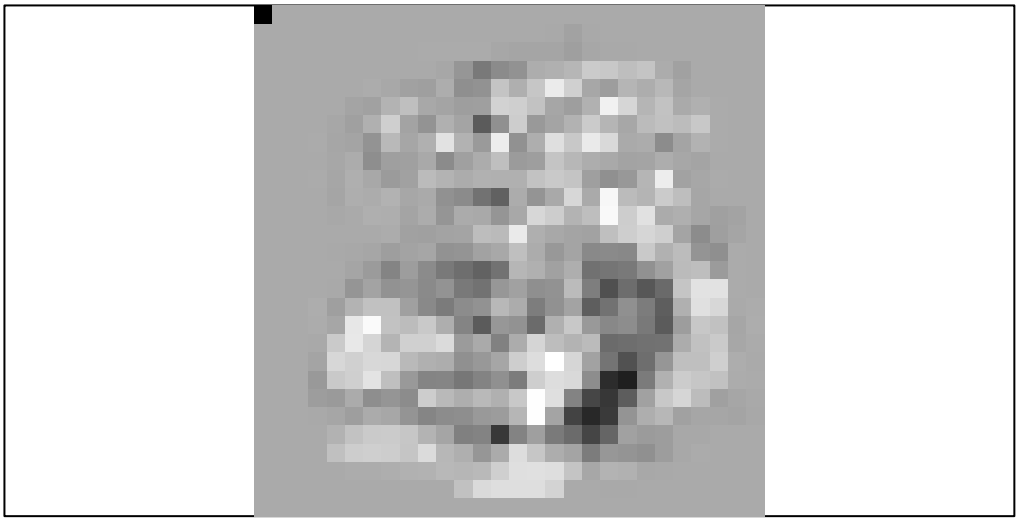
5



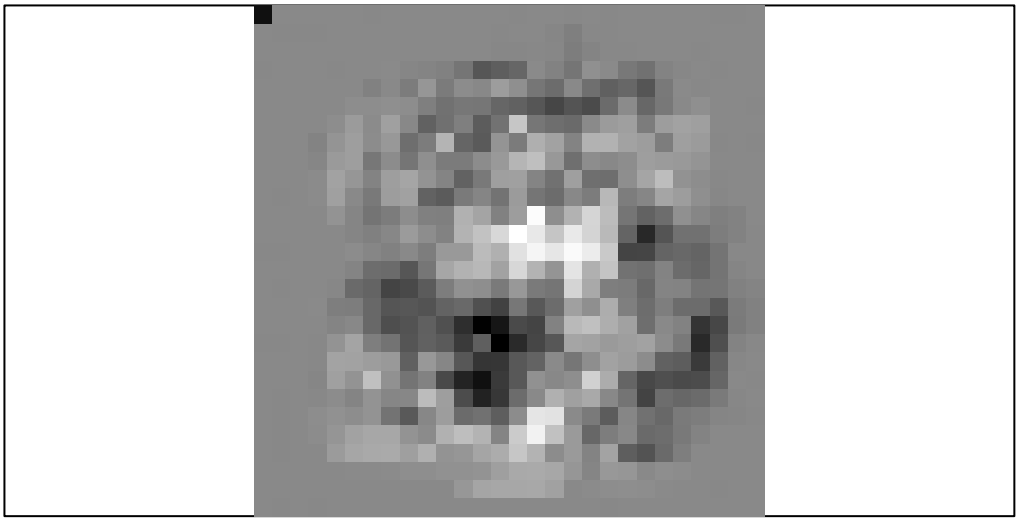
6



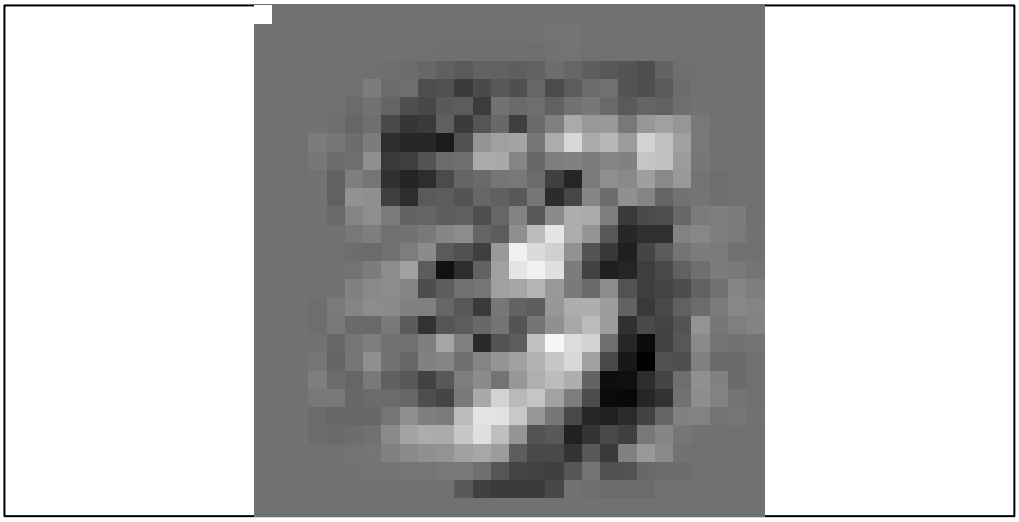
7



8



9



10