TDDE01 Exam 2018-01-11

perli944

Assignment 1

Question 1

2p Before scaling: One variable explains 99.7% of the variation.
After scaling: Nine variables are required to explain >95% of the variation.

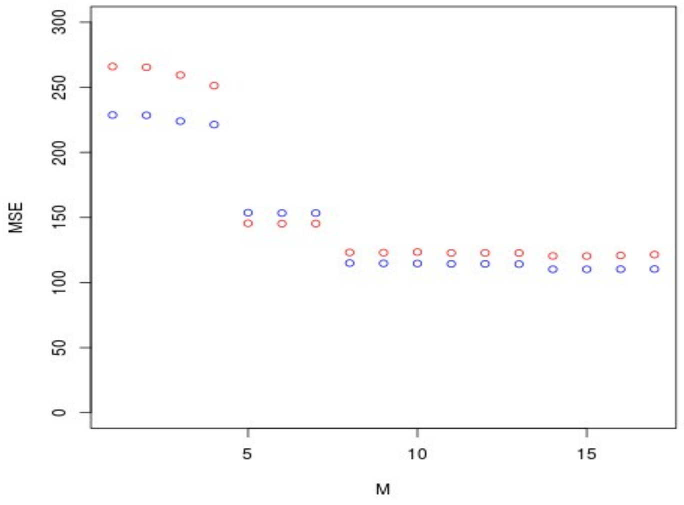
The variables in the given data set differs a lot in variance and some variables are considered more important than they should before they have been scaled.

Question 2

The plot below shows PCR iterations with M components. I tried with M values between 1 and 17.

I used mean squared error to compare the predicted utime-values with the values from the actual data. The blue points is the errors for the training data and the red is for the test data.

1.5p



The error when predicting on the training data is generally lower since the model has been trained on the same data. However, when more information is taken into account (more components are considered, which in turn account for more variation) then we get less affect by bias towards the training data in the model. This can be seen on the graph, since there is less difference between the MSE of training and test data which higher values of M.

Question 3

For PCR with M = 8 the model has the following dependency in the principal components.

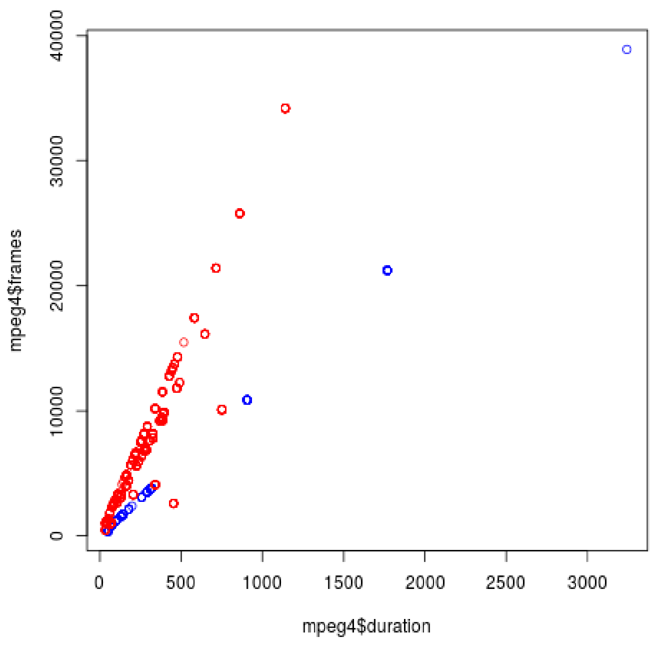
0p

```
       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X       99.701   99.927   99.987   100.00   100.00   100.00   100.00   100.00
utime    3.894    3.971    5.883     6.98    35.41    35.55    35.55    51.69
```

Question 4

The blue points in the plot belongs to the mpeg4 class, the red points are the others. It seems that in general, mpeg4 has fewer frames per duration than the other class except for a few points. The classes should be separable by a linear decision boundary.

1p



Question 5

My LDA model predicted the classes with "frames" and "duration" as features with a misclassification rate of 17.2%.

I believe that the reason of imperfection is that the "other"-class includes some video formats that has similar frame-rates to mpeg4 (the red dots we can see closer to or below the blue dotted line in the plot above) and that they are misclassified as mpeg4. Also close to origo, the points seems to be pretty close, at least from an graphical inspection, which may make them prone to misclassification.

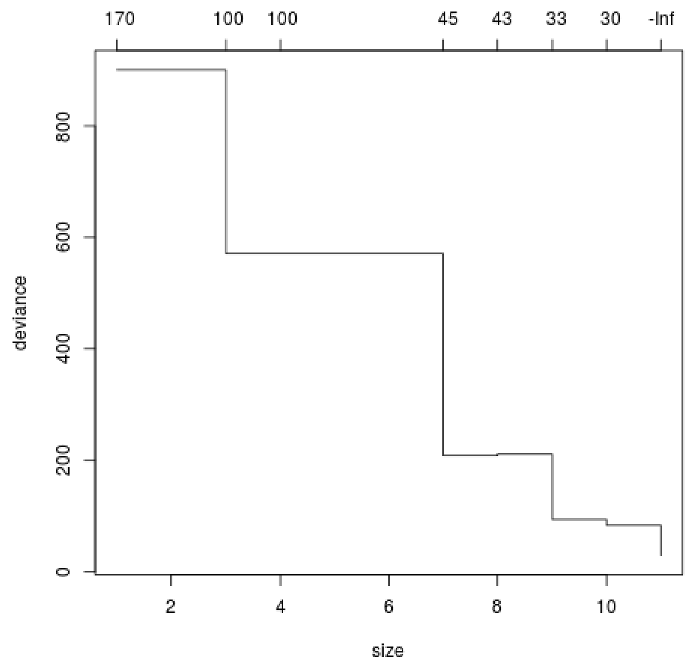
0.5p

i.e. the clusters of observation belonging to the different classes are not perfectly separated.

Question 6

I fitted a tree model for the same formula as in the last model, class as target and "frames" and "duration" as features, then ran cross-validation for that model. Below, I have included a plot of the deviance with regard to the tree size. It shows that larger trees are best in this case, using all 11 leaves gives the lowest deviance.

2p



The error values:

```
    deviance size
1  29.68039   11
2  83.36153   10
3  93.79438    9
4 210.78546    8
5 208.16330    7
6 570.76983    4
7 570.76983    3
8 900.39715    1
```

In order to minimize the error the tree needs many leaves because the decision boundary is diagonal when plotting frames versus duration. Each leaf only has conditions for one variable, and the decision boundary depends on a combination of the variables.

Assignment 2: Support vector machines

Question 1

I split the spam-data into two datasets, one for test and one for training.

I then created the three models, trained them, predicted classes for the test-data and calculated their misclassification rates.

Model	**Kernel**	**Width**	**C-value**	**Misclass-rate**
#1	Gaussian	0.05	0.5	9.1%
#2	Gaussian	0.05	1	7.7%
#3	Gaussian	0.05	5	8.3%

Their misclassification rates did not differ that much, but we can see that model #2 got the best result.

Question 3

My SVM of choice:

```
model2 = ksvm(type ~ ., data = spam_train, kernel = "rbfdot",
                  kpar = list(sigma = 0.05), C = 1)
```

Question 4

The C parameter is the chosen cost for constraint violation. A high C gives more importance to outliers, while a low C gives them less importance.

Assignment 2: Neural Networks

Question 5

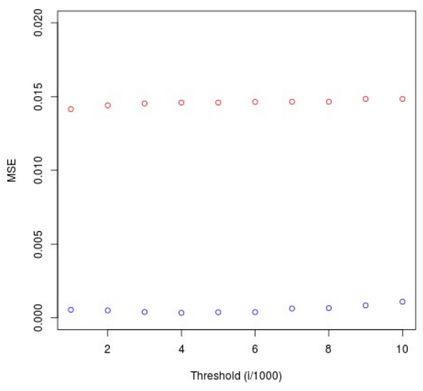
The first noticeable difference is that all of the single layer models have much lower errors than the multi layer ones:

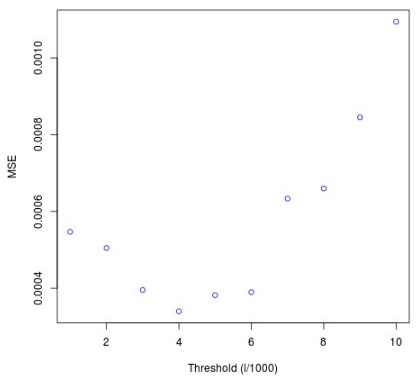
```
       Single.layer   Multi.layer
1  0.0005470766106 0.01414961878
2  0.0005049679830 0.01440991300
3  0.0003953675602 0.01453800736
4  0.0003400357697 0.01459536187
5  0.0003822182977 0.01459536451
6  0.0003897488396 0.01463929955
7  0.0006335707428 0.01465524654
8  0.0006597009662 0.01465524654
9  0.0008449961508 0.01484359238
10 0.0010943638214 0.01484359238
```

The multi layer models seem to have a fairly constant error independent of the threshold value.

Examining the results of the single layer models closer, we can see that a threshold value of 4/1000 gives the best result.

The left plot below shows the multi layer models in red and the single layer in blue. The right one is zoomed in a bit to show the differences between the single layer models.





Question 7

I would not say that more layers are better, at least not for this application. Here we can clearly see that they are greatly outperformed by the single layer models, but for each kind of application we should reconsider which is the best.

Appendix

Assignment 1

```
library(pls)
library(ggplot2)
library(MASS)
library(tree)

video_data = read.csv("~/video.csv")

# division of data
set.seed(12345)
id = sample(1:nrow(video_data), floor(nrow(video_data)*0.5))
video_train = video_data[id,]
video_test = video_data[-id,]

data1 = video_train # all numeric variables except utime
data1$codec = c()
data1$utime = c()

# PCA
res = prcomp(data1, scale. = FALSE)
sprintf("%2.3f", explvar(res))

# PCR
M_range = c(1:17)

data2 = video_train
data2$codec = c()

mse_train = M_range
mse_test = M_range
for( M in M_range ) {
  f = formula(paste("utime ~ ."))
  pcr.fit = pcr(f, data = data.frame(data2), ncomp = M)

  pred_train = predict(pcr.fit, newdata = video_train, ncomp = M)
  pred_test = predict(pcr.fit, newdata = video_test, ncomp = M)

  mse_train[M] = sum((pred_train - video_train$utime)**2)/nrow(video_train)
  mse_test[M] = sum((pred_test - video_test$utime)**2)/nrow(video_test)
}

tab = cbind(mse_test, mse_train)

png("pcr_errors.png")
plot(mse_train, col = "blue", ylim = c(0,300), ylab = "MSE", xlab = "M")
points(mse_test, col = "red", ylim = c(0,300))
dev.off()

# PCR with M = 8
data3 = video_train
data3$codec = c()

f = formula(paste("utime ~ ."))
pcr.fit = pcr(f, data = data.frame(data3), ncomp = 8)
summary(pcr.fit)
plot(pcr.fit)

# separation and plot of mpeg and others
```

```
class = c(1:nrow(video_data))
for( i in 1:nrow(video_data) ){
  if(video_data$codec[i] != "mpeg4") {
    class[i] = "other"
  } else {
    class[i] = "mpeg4"
  }
}
video_data$class = class

mpeg4 = subset(video_data, class == "mpeg4")
other = subset(video_data, class == "other")

png("mpeg4_vs_other.png")
plot(mpeg4$duration, mpeg4$frames, col = "blue")
points(other$duration, other$frames, col = "red")
dev.off()

# LDA
lda.data = data.frame(duration = scale(video_data$duration), frames =
scale(video_data$frames), class = video_data$class)
lda.fit = lda(class ~ ., data = lda.data)
lda.pred = predict(lda.fit, data = lda.data)
mean(lda.pred$class != lda.data$class)

# decision tree
tree.data = lda.data
tree.fit = tree(class ~ ., data = tree.data)

tree.cv = cv.tree(tree.fit)

png("tree_cv.png")
plot(tree.cv)
dev.off()
```

Assignment 2

```
library(kernlab)
library(neuralnet)

# support vector machine
data(spam)

set.seed(12345)
id = sample(1:nrow(spam), floor(nrow(spam)*0.5))
spam_train = spam[id,]
spam_test = spam[-id,]

model1 = ksvm(type ~ ., data = spam_train, kernel = "rbfdot", kpar = list(sigma
= 0.05), C = 0.5)
model2 = ksvm(type ~ ., data = spam_train, kernel = "rbfdot", kpar = list(sigma
= 0.05), C = 1)
model3 = ksvm(type ~ ., data = spam_train, kernel = "rbfdot", kpar = list(sigma
= 0.05), C = 5)

pred1 = predict(model1, spam_test[,-58])
pred2 = predict(model2, spam_test[,-58])
pred3 = predict(model3, spam_test[,-58])

mean(pred1 != spam_test$type)
mean(pred2 != spam_test$type)
mean(pred3 != spam_test$type)
```

```
# neural network
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
winit <- runif(31,-1,1)

# errors for single layer
MSE <- c(1:10)
for (i in 1 : 10) {
  nn <- neuralnet(Sin~Var, data=tr, hidden = 10, threshold = i/1000,
startweights = winit)
  pred <- compute(nn, va$Var)$net.result
  MSE[i] <- mean((pred - va$Sin)**2)
}
print(MSE)
print(which.min(MSE))

# errors for multi layer
MSE2 <- c(1:10)
for (i in 1 : 10) {
  nn <- neuralnet(Sin~Var, data=tr, hidden = c(3,3), threshold = i/1000,
startweights = winit)
  pred <- compute(nn, va$Var)$net.result
  MSE2[i] <- mean((pred - va$Sin)**2)
}
print(MSE2)
print(which.min(MSE2))
plot(MSE2, xlab="Threshold (i/1000)")

# results merged into a dataframe
MSEs = data.frame("Single layer" = MSE, "Multi-layer" = MSE2)

# plots
png("NN_error.png")
plot(MSE, xlab="Threshold (i/1000)", col = "blue", ylim = c(0,0.02))
points(MSE2, col = "red")
dev.off()

png("NN_error2.png")
plot(MSE, xlab="Threshold (i/1000)", col = "blue")
dev.off()
```