Oblig 2

Håkon Kvernmoen

3/30/2021

Problem 1

a)

We begin by loading the data and saving it to the `spam` data-frame. We also include the `MASS` library to use GLM's

```
fil = "https://www.uio.no/studier/emner/matnat/math/STK2100/data/spam_data.txt"
spam = read.table(fil, header=T)

library(MASS)
```

We now preform a logistic regression on the training data, using all the explanatory variables. The `train` column only indicates if the data point should be used for training, so we do not include this variable in our fit, but rather use it as a subset to preform the fit.

```
fit = glm(y~.-train, data=spam, subset=spam$train, family = binomial)
```

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

```
pred = predict(fit, spam[!spam$train,], type="response")>0.5
```

The most natural measure to use for prediction performance is to compute the confusion matrix. We will also compute the percentage of false positives (FP) and false negatives (FN).

```
fit.confusion_mat = table(pred, spam$y[!spam$train])
fit.confusion_mat # Print confusion matrix
```

```
##
## pred    FALSE TRUE
##   FALSE  1733  148
##   TRUE    114 1070
```

```
fit.FP = fit.confusion_mat[2,1]/length(spam$y[!spam$train] == T) # Compute % of false positives
fit.FN = fit.confusion_mat[1,2]/length(spam$y[!spam$train] == F)# Compute % of false negatives
sprintf("FP: %.3f, FN: %.3f", fit.FP, fit.FN)
```

```
## [1] "FP: 0.037, FN: 0.048"
```

We see that the logistic regression using all the explanatory variables does a fairly good job. In about 4% of cases it labels the data as spam, even though it is not (FP) and in about 5% of cases it does not label the data as spam even though it is (FN).

b)

We have $p = 57$ explanatory variables. To reduce the dimensionality of the data while keeping as much of the origional relationships as possible, we compute the principle components.

```r
x.prcomp = prcomp(spam[,1:57], retx=T, scale=T)
d = data.frame(x.prcomp$x, y=spam$y, train=spam$train)
```

The parameter `scale` is set equal to true since we have not scaled the explanatory variables (design matrix) before computing the principle components. Scaling is important, since the PCA relies on finding the linear combinations of the columns from the design matrix that has maximum variance. If lets say the first explanatory variable (first column of design matrix) is 10 times bigger than all the other explanatory variables it would be over-represented in the PCA since the variance is higher (even tough the "spread" might be close to equal). When preforming a PCA we are only interested in the "spread" of each explanatory variable, independent of the data being on the interval $[-5, 5]$ or $[-5000, 5000]$. Thus scaling our explanatory variables such that the variance is a true measure of the actual "spread" of the data is crucial.

We then preform the logistic regression using the 2 first principal components.

```r
fit.pc = glm(y~.-train, data=d[, c(1:2, 58,59)], family = binomial, subset=d$train)
pred.pc = predict(fit.pc, d[!d$train, ], type="response")>0.5
fit.pc.confusion_mat = table(pred.pc, d$y[!d$train])
fit.pc.confusion_mat
```

```
##
## pred.pc FALSE TRUE
##   FALSE  1722  279
##   TRUE    125  939
```

```r
fit.pc.FP = fit.pc.confusion_mat[2,1]/length(d$y[!d$train] == T) # Compute % of false positives
fit.pc.FN = fit.pc.confusion_mat[1,2]/length(d$y[!d$train] == F)# Compute % of false negatives
sprintf("FP: %.3f, FN: %.3f", fit.pc.FP, fit.pc.FN)
```

```
## [1] "FP: 0.041, FN: 0.091"
```

We see that both FP and FN increases. FP increases slightly from 3.7% to 4.1% and FN increases quite a lot from 4.8% to 9.1%. This is somewhat excepted as we have only included the two first principal components. These are the most important components, but does not necessarily contain as much of the relationship between the explanatory variables and the target as all the 57 "raw" explanatory variables. This can be seen for instance by printing out the 5 largest principal components (square root of the eigenvalues of the covariance matrix)

```r
x.prcomp$sdev[1:5]
```

```
## [1] 2.567476 1.807602 1.415327 1.270102 1.243466
```

As we can see the to first principal components are largest (2.56... and 1.80...) but the rest are not dramatically smaller than these two. This is a pointer for us to include more than just two principal components.

c)

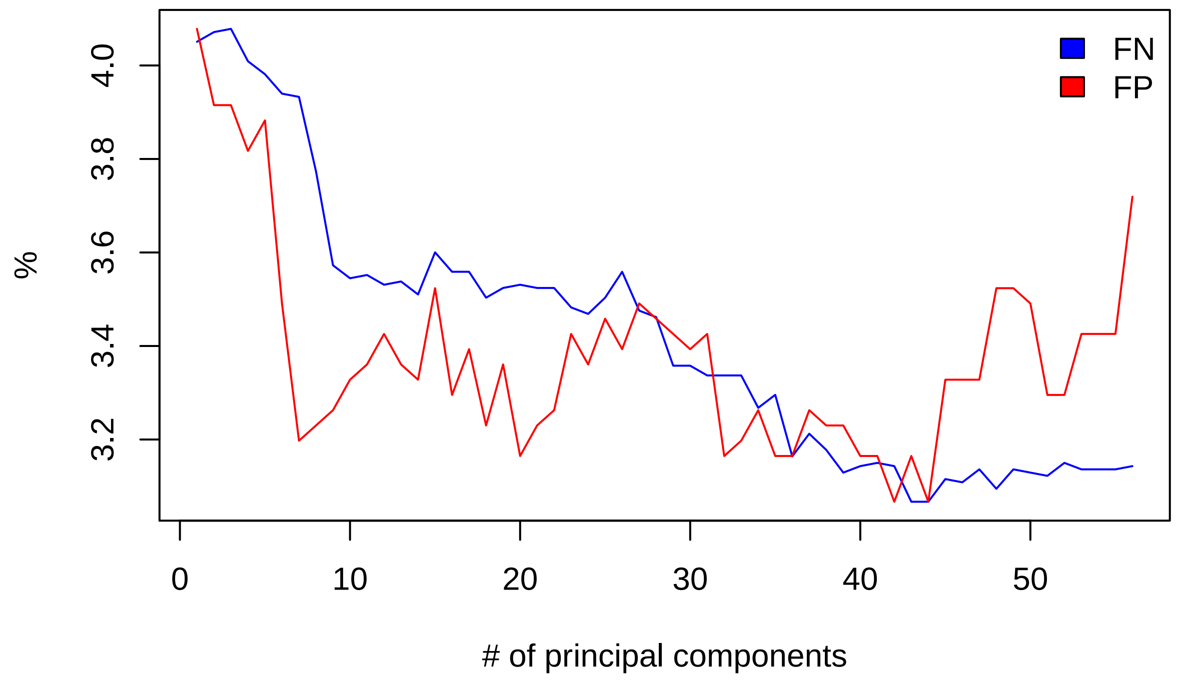
We will now try to increase the number of principal components in our logistic regression model. As a measure of prediction performance we will use the same FP/FN percentages as in **a** and **b**.

```
min_pc = 2
max_pc = 57
n_pc = max_pc-min_pc+1
FP_vec = seq(min_pc,max_pc, length.out = n_pc)
FN_vec = seq(min_pc,max_pc, length.out = n_pc)

for( i in 1:n_pc) {
  k = i+1
  fit.temp = glm(y~.-train, data=d[, c(1:k, 58,59)], family = binomial, subset=d$train)
  pred.temp = predict(fit.temp, d[!d$train, ], type="response")>0.5
  fit.temp.confusion_mat = table(pred.temp, d$y[!d$train])
  FP_vec[i] = fit.temp.confusion_mat[2,1]/length(d$y[!d$train] == F)
  FN_vec[i] = fit.temp.confusion_mat[1,2]/length(d$y[!d$train] == T)
}

cols = c("blue", "red")

plot.new()
plot(FN_vec*100, type="l", col=cols[1], axes=FALSE, xlab="# of principal components", ylab="%")
par(new=TRUE)
plot(FP_vec*100, type="l", col=cols[2], xlab="", ylab="")
legend("topright", c("FN","FP"), fill=cols, bty="n")
```



%
4.0
3.8
3.6
3.4
3.2
0
10
20
30
40
50
# of principal components
FN
FP