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. This is a 2×2 matrix with TN (true negative) and TP (true positive) on the diagonal. The element below the diagonal is FN (false negative) and above the diagonal FP (false positive). To make our measure more concise and direct we will calculate the TPR (true positive rate) and TNR (true negative rate). These are defined as

$$\text{Confusion matrix: } \begin{pmatrix} TN & FN \\ FP & TP \end{pmatrix} \quad TPR = \frac{TP}{TP+FN} \quad \text{and} \quad TNR = \frac{TN}{TN+FP}$$

We aim to have both TPR and TNR close to one. If TPR = 1 we have no false negatives and if TNR = 1 we have no false positives. If FN = TP and TN = FP we have TPR = 0.5 and TNR = 0.5 and the models preforms no better than guessing. We compute the confusion matrix and make a function to calculate TPR/TNR as we will use them a lot.

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

set_dec = function(x, k) trimws(format(round(x, k), nsmall=k))
true_PN_rate = function (mat, dec=4, display=F) {
```

```
  TPR = mat[2,2]/(mat[2,2]+mat[1,2])
  TNR = mat[1,1]/(mat[1,1]+mat[2,1])

  if(display) {
    paste("TPR = ", set_dec(TPR, dec), "TNR = ", set_dec(TNR, dec))
  }
  else {
    c(TPR, TNR)
  }
}

fit.confusion_mat # Print confusion matrix
```

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

```
print(true_PN_rate(fit.confusion_mat, display=T))
```

```
## [1] "TPR =  0.8785 TNR =  0.9383"
```

Thus $TPR \approx 0.88$ and $TNR \approx 0.94$. How accurate it needs to be is dependent on the application, but we can probably obtain better results using a more complex model. The goal of a spam filter is to hide unwanted emails from the users inbox. Two or three spam emails slipping trough is not the end of the world, but we should definitely try to avoid marking an actual email as spam (FP) since it might contain important information such as a change in flight times or your power bill. Thus the most important measure for our purposes is to maximize the TNR.

b)

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

```
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.

```
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
```

```
print(true_PN_rate(fit.pc.confusion_mat, display=T))
```

```
## [1] "TPR =  0.7709 TNR =  0.9323"
```

We observe that FN increases quite a lot an thus we end up with $TPR \approx 0.77$. FP (the one we are most worried about) increases just slightly resulting in $TNR \approx 0.93$. 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)

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

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

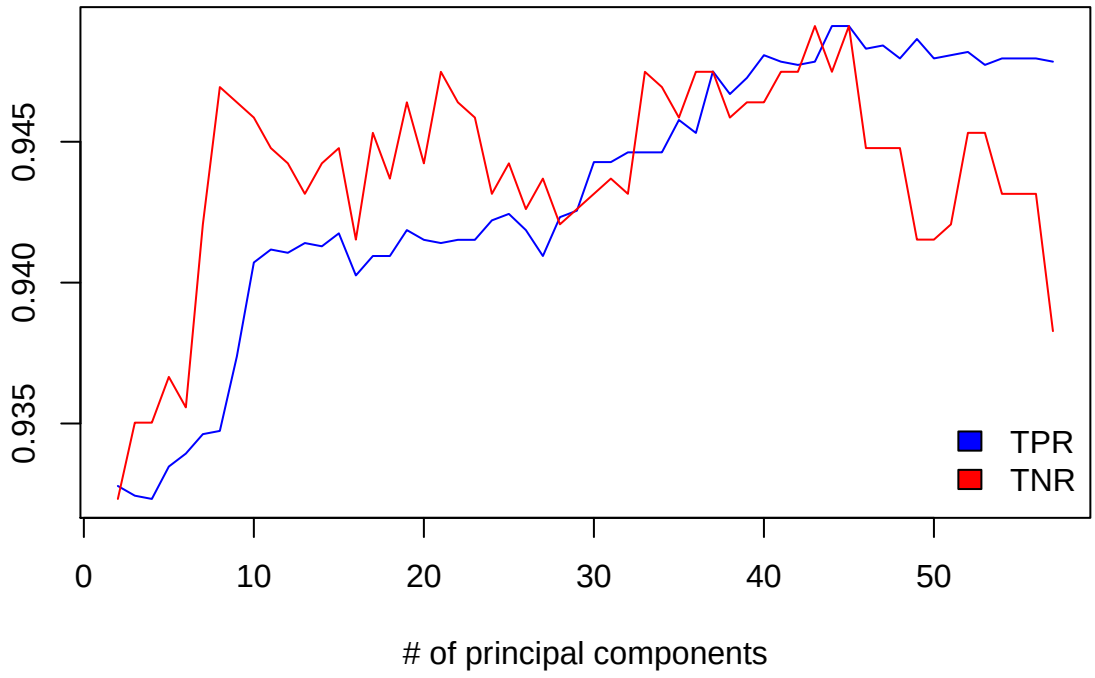
As we can see the to first principal components are largest (2.56... and 1.80...) but the rest are not dramatically smaller than these two. This is a pointer for us to include more than just two principal components.

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
n_pc_vec = seq(min_pc, max_pc, length.out = n_pc)
TPR_vec = seq(0, 0, length.out = n_pc)
TNR_vec = seq(0, 0, length.out = n_pc)
var_sum_vec = seq(0, 0, 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])
  TR = true_PN_rate(fit.temp.confusion_mat)
  TPR_vec[i] = TR[1]
```

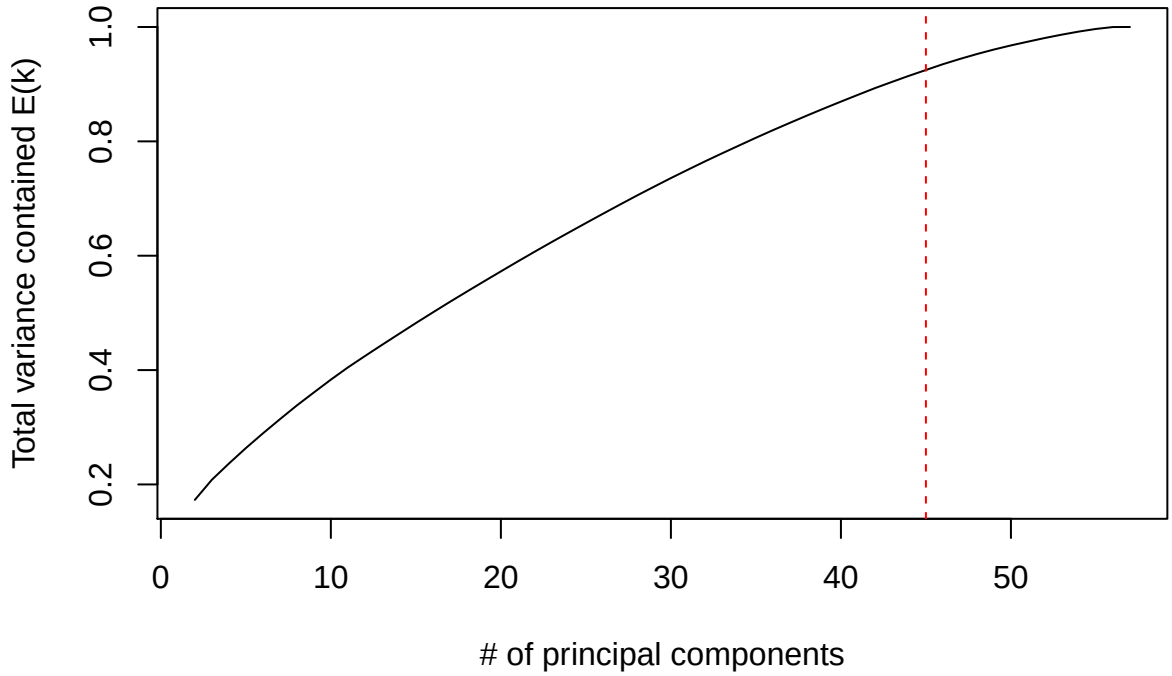
```
  TNR_vec[i] = TR[2]
  var_sum_vec[i] = sum(x.prcomp$sdev[1:k]^2)
}

var_sum_vec = var_sum_vec/sum(x.prcomp$sdev^2)
cols = c("blue", "red")
plot(n_pc_vec, TPR_vec, type="l", col=cols[1], axes=FALSE, xlab="# of principal components", ylab="")
par(new=TRUE)
plot(n_pc_vec, TNR_vec, type="l", col=cols[2], xlab="", ylab="")
legend("bottomright", c("TPR","TNR"), fill=cols, bty="n")
```



We see that about $k = 45$ gives the best performance for both TPR and TNR, both ending up a little under 0.95. This is a fairly good result and is an increase compared with only using the "raw" 57 explanatory variables. However on of the goals using principal components is to reduce the dimensionality. Of course $45 < 57$, but there might be some other modifications we can do to our model with the goal of using fewer explanatory variables. The sum of all the eigenvalues of the covariance matrix correspond to all the variance in our data. We can thus create a cumulative sum over all the kth first eigenvalues divided by the sum of all the eigenvalues to get a measure of how much variance the first k principal components contained. We note this as the normalized "energy" $E(k)$ as a function of principal components with N as the total number of principal components.

$$E(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{N} \lambda_i}$$

```r
plot(n_pc_vec, var_sum_vec, type="l", xlab="# of principal components", ylab="Total variance contained E
abline(v=45, lty=2, col="red")
```




```r
sprintf("E(45) = %.4f", var_sum_vec[45])
```

```
## [1] "E(45) = 0.9349"
```

We observe that $E(k)$ is decreasing with k as expected (the principal components are sorted by eigenvalue). The form looks somewhat like a variant of $\ln(k)$. A horizontal line is drawn at $k = 45$ where we observe that the amount of new variance given after $k = 45$ is small.

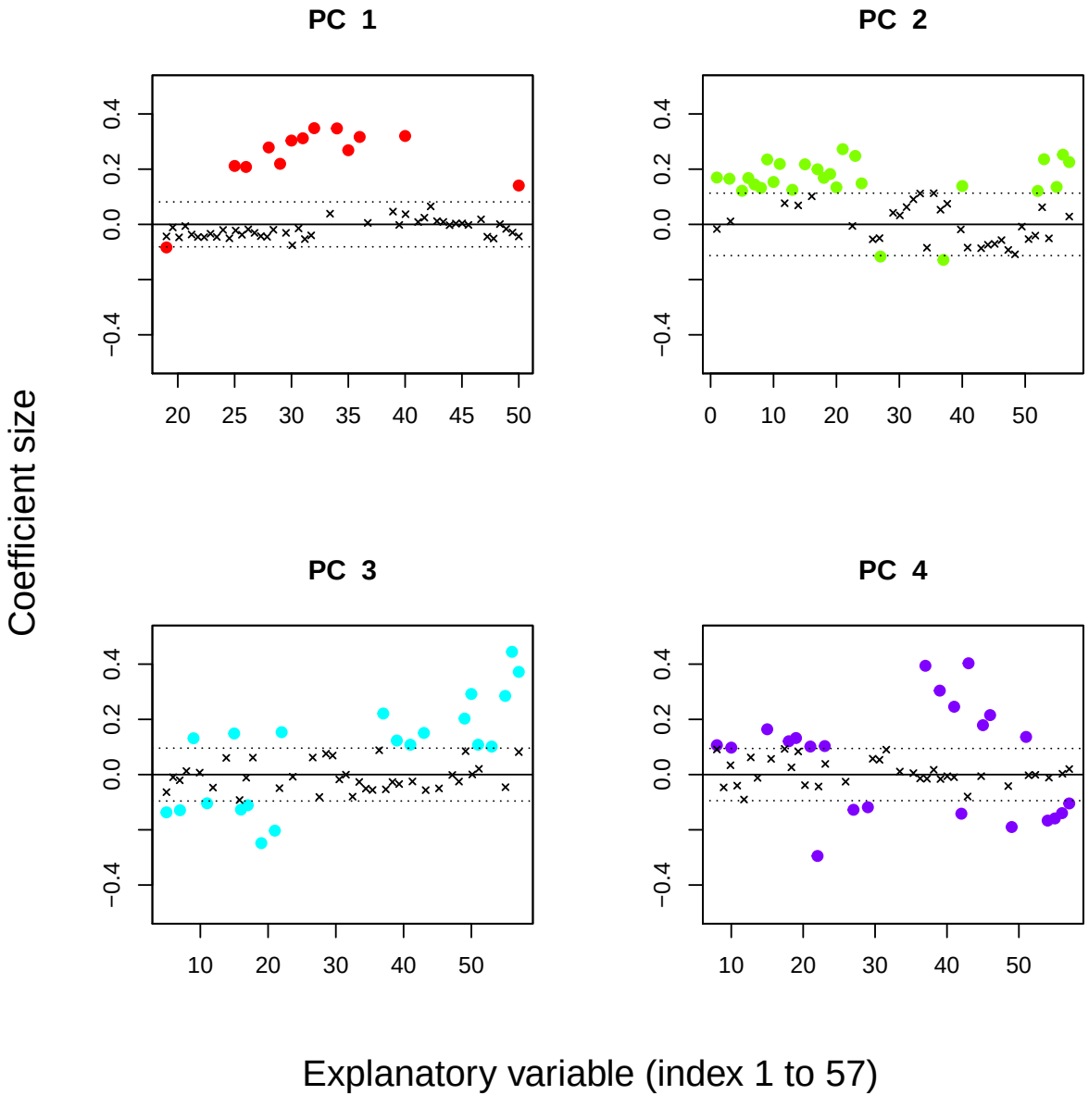
d)

When using the most important principal components we are granted with new variables that explain more of the data than the original variables. These new variables are linear combinations of the original variables, so the trade off is that we loose some interpretation of the model. Some analysis of the principal components is thus desired to achieve better interpretability. Since we use quite a lot of principal components, deep diving into the structure of each of these will be time consuming so we will here focus on the 4 most important (with the largest eigenvalue).

Each principal component will have 57 coefficients corresponding to the 57 explanatory variables. Some will be large (meaning this variable is important for this principal component) and some small or (very close to) zero (meaning this variable is **not** important to this principal component). To pick out the "large"

components we plot for each principal component the coefficients that are larger than the mean of the absolute value of all the coefficients for that principal component. All other coefficients are "small" and are plotted as crosses, while the "large" ones are plotted as colored circles. The solid line is zero while the two dotted lines are ± the mean of the absolute value of the coefficients.

```
n = 4
cols = rainbow(n)
par(mfrow=c(2,2), oma=c(2,2,0,0))
for( i in 1:n ) {
  avrg_coef_size = mean(abs(x.prcomp$rotation[,i]))
  pc.eig= x.prcomp$rotation[,i]

  pc.x = seq(1, length(pc.eig), length.out = length(pc.eig))
  pc.eig.large = pc.eig[abs(pc.eig) > avrg_coef_size]
  pc.x.large = pc.x[abs(pc.eig) > avrg_coef_size]
  pc.eig.small = pc.eig[abs(pc.eig) < avrg_coef_size]
  pc.x.small = pc.x[abs(pc.eig) < avrg_coef_size]

  plot(pc.x.large, pc.eig.large, type="p", col=cols[i], ylab="", main=paste("PC ", i), xlab="", pch=19,
  par(new=T)
  plot(pc.x.small, pc.eig.small, type="p",pch=4, cex=0.6,  yaxt="n", xaxt="n",ann=F, ylim=c(-0.5,0.5))
  abline(0,0)
  abline(avrg_coef_size, 0, lty="dotted")
  abline(-avrg_coef_size, 0, lty="dotted")
}

mtext("Explanatory variable (index 1 to 57)", side=1, line=0, outer=T, cex=1.3)
mtext("Coefficient size", side=2, line=0, cex=1.3, outer=T)
```



To interpret this we quickly summarize what each explanatory variable means. The first 48 variables correspond to the frequency of a specific word (**# specific word**/**# total words** $\times 100$). This checks for common spam words such as "money" and "free". The six variables (49 to 54) checks for the frequency of specific charters (**# character** / **total characters** $\times 100$). The last three measures use of capital letters in different forms. From this we can give a vague intepretation of the 4 most important principal compoenents.

1. Contains mostly specific words (from the 48 first variables). In addition it contains the 50 variable meaning some specific character is important
2. Again a lot of different words (from the 48 first variables), but with little overlap with the first principal component. In addition the variables 52, 53, 55, 56 and 57 are also included. The first two (52, 53) correspond to some specific character (not the same as the first principal component) while the last three correspond to different occurrences of words in capital letters.
3. This again includes mostly specific words but again some specific characters and capital letters are

important. All the variables corresponding to capital letters are duplicates of the second principal component

4. This is very similar to the third and second principal components. Some specific words are important, in addition to specific characters and words in capital letters.

e)

We now try some non-linear models. First we fit the model using the 3 first explanatory variables using smoothing splines with 4 degrees of freedom. Then we use the same 3 explanatory variables to fit a logistic regression model. We can then check if these non-linear terms yields any improvements.

```
library(gam)
```

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

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

```
## Loaded gam 1.20
```

```
fit.gam = gam(y~s(x1)+s(x2)+s(x3), data=spam, subset =spam$train, family = binomial) # Smoothing spline
fit.log = gam(y~x1+x2+x3, data=spam, subset=spam$train, family = binomial) # Logistic regression

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

confusion.gam = table(pred.gam, spam$y[!d$train])
confusion.log = table(pred.log, spam$y[!d$train])

print(paste("Natural Splines -> ", true_PN_rate(confusion.gam, display = T)))
```

```
## [1] "Natural Splines ->  TPR =  0.4729 TNR =  0.8500"
```

```
confusion.gam
```

```
##
## pred.gam FALSE TRUE
##    FALSE  1570  642
##    TRUE    277  576
```

```
print(paste("Logistic regression -> ", true_PN_rate(confusion.log, display = T)))
```

```
## [1] "Logistic regression ->  TPR =  0.2233 TNR =  0.9123"
```

```
confusion.log
```

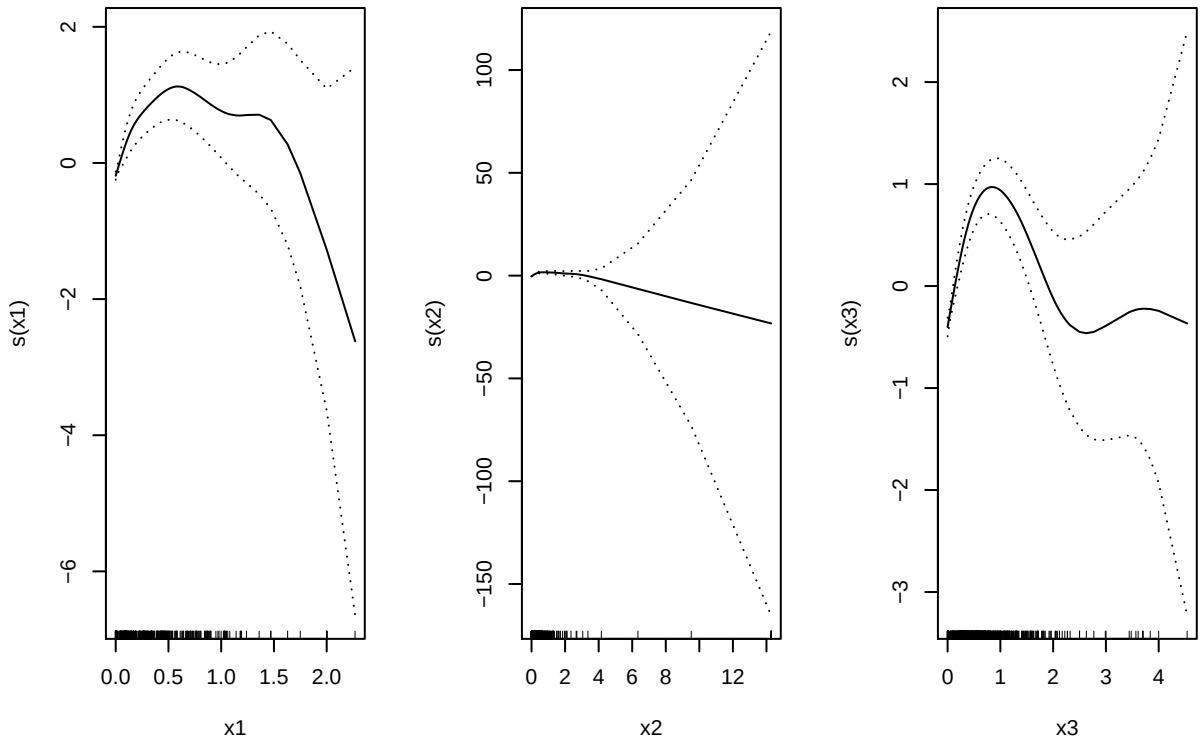
```
##
## pred.log FALSE TRUE
##    FALSE  1685  946
##    TRUE    162  272
```

We can see that the results are drastically worse than the approach using many principal components. The logistic regression actually had a higher TNR than the natural splines, but the TPR of logistic regression is very bad. There might be some improvement as the TPR is higher for the splines, but the TPR is still ≈ 0.47 so it is actually worse than guessing. Deciding if there is any definite benefit for including a more complex relationship between the explanatory variables and the target is not feasible when we restrict ourselves to only using 3 variables.

This is actually not too surprising, as we have arbitrarily chosen three explanatory variables to predict whether or not an e-mail can be classified as spam. There is not necessarily anything specifically interesting about these 3 variables. I think I would have a hard time deciding if an e-mail was spam or not if I were only allowed to see the frequency of three words!

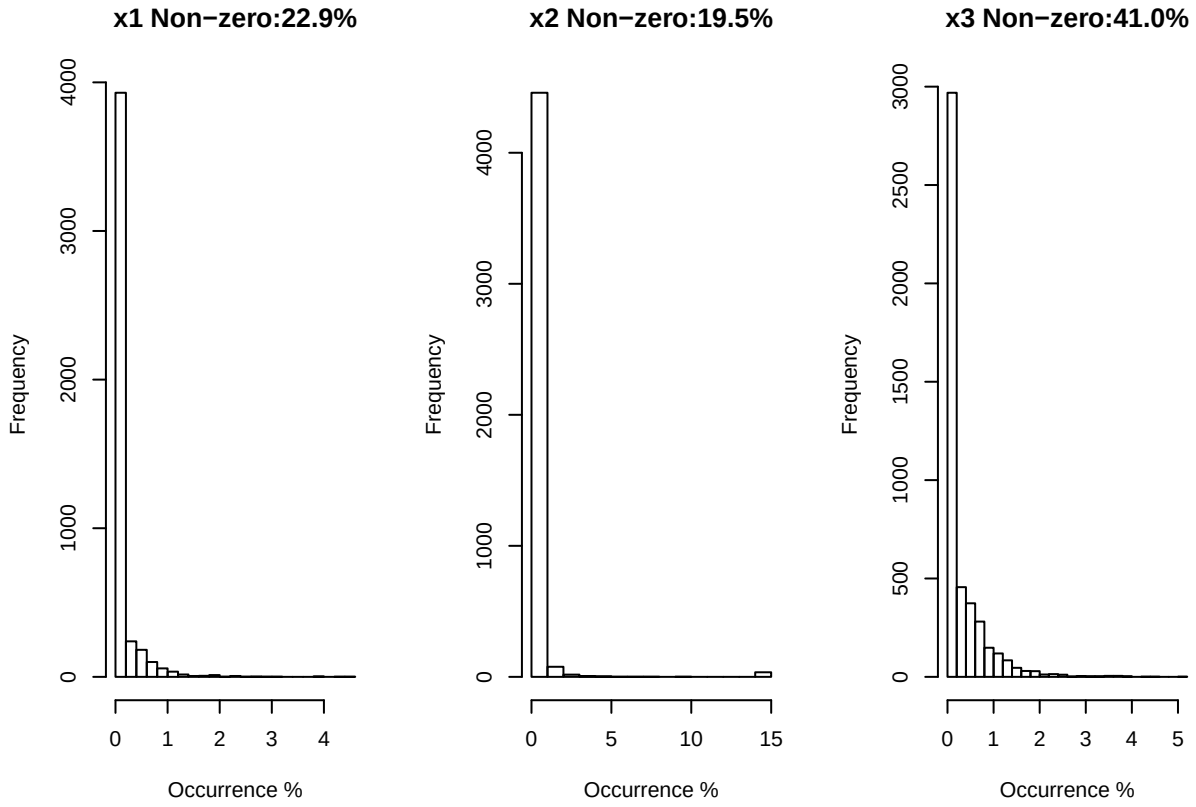
We should take a closer look how the splines behave as a function of the explanatory variables. Below we see a plot of each spline polynomial for each of the three variables. It becomes clear that a lot of the data points contains close to zero or zero entries, such that for any large value of x_1, x_2 or x_3 the standard error is very large.

```
par(mfrow=c(1,3))
plot(fit.gam, se=T)
```



We can also make a histogram for each of the three variables. This is seen below. In the title the percentage of non-zero entries for each variable is presented. The amount of "large" values for these explanatory variables (frequency of a specific word in this case) are few and it is thus difficult to fit for these values. Thus the standard error for each spline (displayed above) for large values goes off the charts. This is an indication that our data is very sparse, confirmed by that only 23%, 20% and 41% of entries for the first, second and third explanatory variable respectively, are non-zero,

```r
par(mfrow=c(1,3))

for(i in 1:3) {
  nonzero = 100*length(spam[,i][spam[,i] != 0])/length(spam[,i])
  hist(spam[,i], breaks=20, xlab="Occurrence %", main=paste("x", i, " Non-zero:", set_dec(nonzero, 1),
}
```



f)

We will now repeat the process from **e** but instead of using the first three explanatory variables, we will use the first three principal components. Again we make one model using natural splines with 4 degrees of freedom and one using linear terms.

```r
fit.pc.gam = gam(y~s(PC1)+s(PC2)+s(PC3), data=d, subset=d$train, family=binomial)
fit.pc.log = gam(y~PC1+PC2+PC3, data=d, subset=d$train, family=binomial)

pred.pc.gam = predict(fit.pc.gam, d[!spam$train, ], type="response") > 0.5
pred.pc.log = predict(fit.pc.log, d[!spam$train, ], type="response") > 0.5

confusion.pc.gam = table(pred.pc.gam, spam$y[!spam$train])
confusion.pc.log = table(pred.pc.log, spam$y[!spam$train])

print(paste("PC gam ->", true_PN_rate(confusion.pc.gam, display=T)))
```

```
## [1] "PC gam -> TPR =  0.8103 TNR =  0.9220"
```

```
confusion.pc.gam
```

```
##
## pred.pc.gam FALSE TRUE
##       FALSE  1703  231
##       TRUE    144  987
```

```
print(paste("PC log ->", true_PN_rate(confusion.pc.log, display=T)))
```

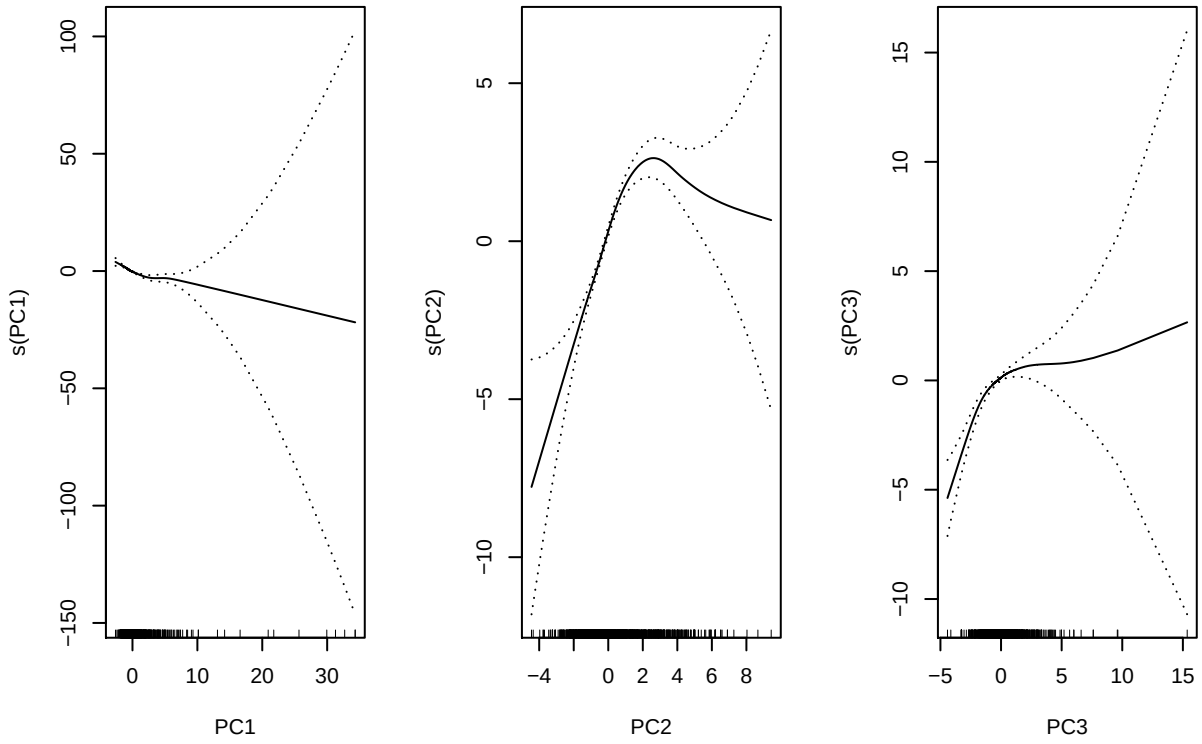
```
## [1] "PC log -> TPR =  0.7685 TNR =  0.9350"
```

```
confusion.pc.log
```

```
##
## pred.pc.log FALSE TRUE
##       FALSE  1727  282
##       TRUE    120  936
```

This gives a great improvement compared with the results from **e**. Both TPR and TNR increases compared with using the "raw" explanatory variables". The non-linear terms gives a decent improvement for TPR, while the logistic regression model still preforms (very slightly) better for TNR. As we did in **e**, we take a closer look on the splines as a function of the principal components.

```
par(mfrow=c(1,3))
plot(fit.pc.gam, se=T)
```



We see that using the principal components gives a smaller standard error for a bigger range. As the variables have been standardized, we see that the inputs (PC1, PC2 and PC3) are centered around 0. There is still a large error on the tails, especially for the first principal component. Especially in our case where most of the explanatory variables are bound by $[0, 100]$ a PCA is very beneficial and a non-linear structure seems to slightly better represent the relationship between the principal components and the target.

g)

We will now look at a non-linear structure (natural splines with 4 degrees of freedom) for more than 3 principal components. These models are more complex and from our results in **f** we would expect a slight improvement.

```
nam = names(d)[1:57]
k = 20
formula = as.formula(paste("y", paste(paste("s(", nam[1:k], ")", sep=""), collapse = "+"), sep="~"))
print(formula)
```

```
## y ~ s(PC1) + s(PC2) + s(PC3) + s(PC4) + s(PC5) + s(PC6) + s(PC7) +
##     s(PC8) + s(PC9) + s(PC10) + s(PC11) + s(PC12) + s(PC13) +
##     s(PC14) + s(PC15) + s(PC16) + s(PC17) + s(PC18) + s(PC19) +
##     s(PC20)
```

```
fit.gam.pc = gam(formula, data=d, subset = d$train, family = binomial)
pred.pc.gam = predict(fit.gam.pc, d[!d$train,], type="response") > 0.5
```

```
confusion.pc.gam = table(pred.pc.gam, spam$y[!spam$train])
confusion.pc.gam
```

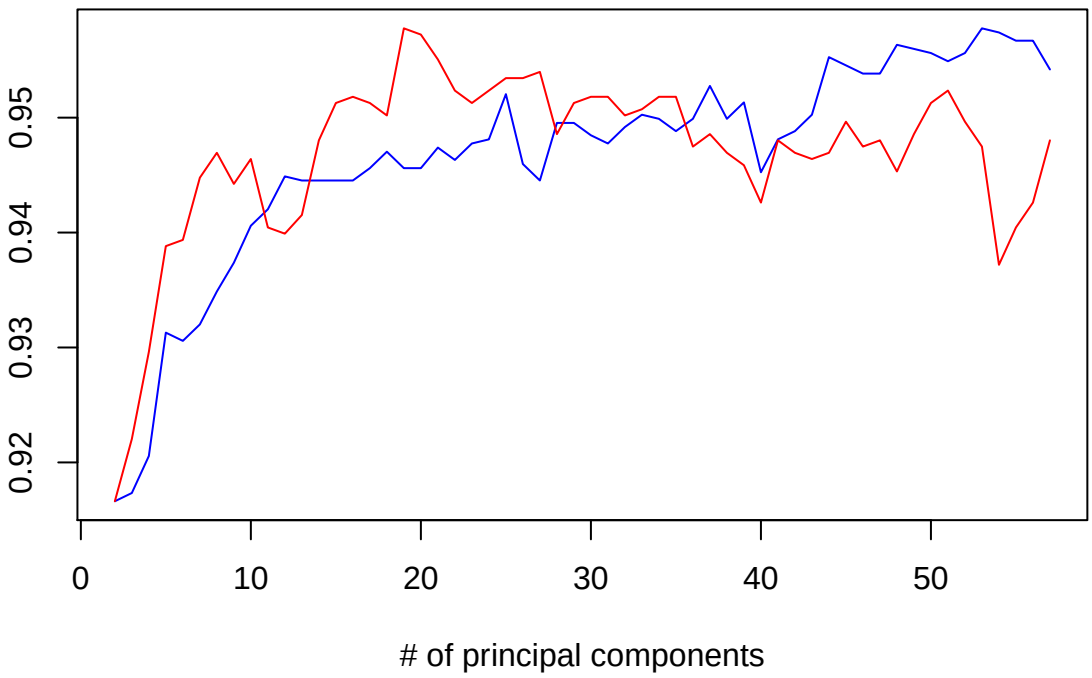
```
##
## pred.pc.gam FALSE TRUE
##       FALSE  1768  152
##       TRUE     79 1066
```

```
print(true_PN_rate(confusion.pc.gam, display = T))
```

```
## [1] "TPR =  0.8752 TNR =  0.9572"
```

We see a good performance for both TPR and TNR. $TNR \approx 0.96$ assures that very few e-mails will be labeled as spam even tough they are not. The TPR is still worse than using $k = 45$ with a linear model. ## h) Lastly we apply the scheme from **c**, but this time using the natural splines for all the principal components.

```
min_pc = 2
max_pc = 57
n_pc = max_pc-min_pc+1
n_pc_vec = seq(min_pc, max_pc, length.out = n_pc)
TPR_vec = seq(0, 0, length.out = n_pc)
TNR_vec = seq(0, 0, length.out = n_pc)

for( i in 1:n_pc) {
  k = i+1
  formula = as.formula(paste("y", paste(paste("s(", nam[1:k], ")", sep=""), collapse = "+"), sep="~"))
  fit.temp = gam(formula, 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])
  TR = true_PN_rate(fit.temp.confusion_mat)
  TPR_vec[i] = TR[1]
  TNR_vec[i] = TR[2]
}


cols = c("blue", "red")
plot(n_pc_vec, TPR_vec, type="l", col=cols[1], axes=FALSE, xlab="# of principal components", ylab="")
par(new=TRUE)
plot(n_pc_vec, TNR_vec, type="l", col=cols[2], xlab="", ylab="")
```


0.92
0.93
0.94
0.95
0
10
20
30
40
50
# of principal components