Oblig2

1.a

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

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

r=sum(ifelse(pred==(spam[!spam$train,]$y),1,0))
errror=1-r/length(spam[!spam$train,]$y)

errror
```

```
## [1] 0.08548124
```

Warning comes up because to many of the values of the fitted model are indistinguishable from 1 or 0.

1.b

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

In general one should scale the variables,except if the variables already have the same units. The goal is to produce a result that is independent of different scaling. Without scaling some variables,these variables will have a much higher variance. Much more weight are placed on these variables (principal component loading of these variables are much bigger then the rest). If the purpose is to see how the other variables relate to the data as well,scaling should be done.

```
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

r=sum(ifelse(pred.pc==(spam[!spam$train,]$y),1,0))
error1=1-r/length(spam[!spam$train,]$y)

error1
```

```
## [1] 0.1318108
```

1.c

```
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

error1=rep(0,56)
for (k in 2:57){
fit.pc = glm(y~.-train,data=d[,c(1:k,58,59)],family=binomial,subset=d$train)
pred.pc = predict(fit.pc,d[!d$train,],type="response")>0.5
r=sum(ifelse(pred.pc==(spam[!spam$train,]$y),1,0))
error1[k-1]=1-r/length(spam[!spam$train,]$y)

}
error1
```

```
##  [1] 0.13181077 0.13115824 0.13148450 0.12724307 0.12659054 0.12071778
##  [7] 0.11745514 0.11027732 0.10114192 0.10048940 0.10114192 0.10081566
## [13] 0.10048940 0.09885808 0.10505710 0.10081566 0.10179445 0.09755302
## [19] 0.09983687 0.09820555 0.09853181 0.09885808 0.09853181 0.09722675
## [25] 0.09983687 0.10179445 0.09885808 0.09787928 0.09265905 0.09233279
## [31] 0.09168026 0.08907015 0.08939641 0.08678630 0.08711256 0.08091354
## [37] 0.08417618 0.08221860 0.07993475 0.07993475 0.08026101 0.07895595
## [43] 0.07634584 0.07536705 0.08026101 0.07993475 0.08123980 0.08123980
## [49] 0.08319739 0.08254486 0.08026101 0.08156607 0.08221860 0.08221860
## [55] 0.08221860 0.08548124
```

```
which.min(error1)
```

```
## [1] 44
```

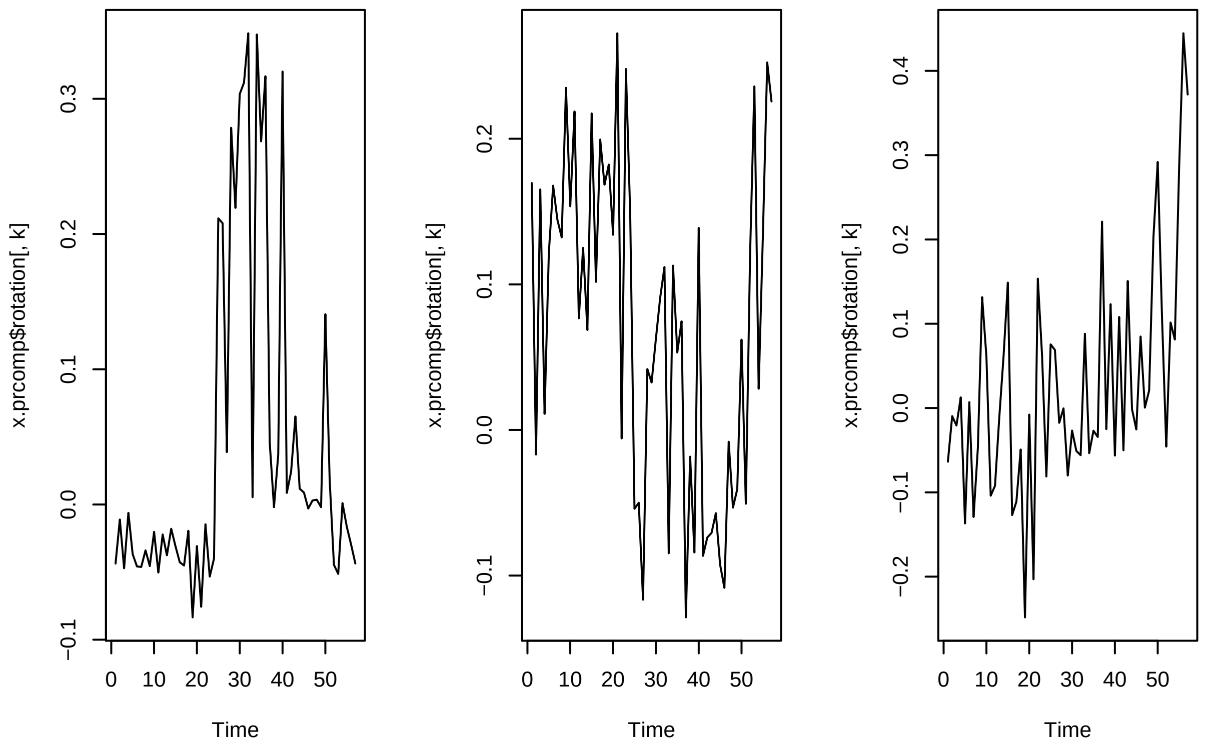
```
error1[44]
```

```
## [1] 0.07536705
```

The goal of PCA is dimensional reduction while retaining most of the information in the data.44 PCA components minimizes the error. Their are 57 PCA components in total.Using all of them gives an error of around 0.08. Using two gives an error of 0.13. If one care about the prediction using 44 components would make sense,but the more variables being used the greater the risk of overfitting.PCA is a unsupervised method which is used as a tool (dimension reduction) for use in supervised method. I would therefore use as few variables as possible but enough to make the error rate small enough.Maybe 19.

1.d,

```
par(mfrow=c(1,3))
for (k in 1:3){
plot.ts(x.prcomp$rotation[,k])
}
```



Plotting the three first principal components. The first plot is showing that for the first principal components,the important covariates are x20 to around x45. The first plot is showing the loadings of the first principal component.The first principal component is the component that when one projects the data onto the component it has the largest total variability from the mean which is set to zero. The second principal component is the component with the second largest variability,and is orthogonal to the first.

Because of this ,one see that in the second principal component the relevant covariates are from x1 to x20.The covariates where the loadings are the largest.The larger the loading of a covariate is,the more it contributes to the principal component.

The third principal component is orthogonal to the first and second principal components. In this component x45 to x57 has the largest loadings. This is the goal for when determining the first principal component:

$maximize \ \ \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n}\phi_{j,1}x_{i,j})^2 \quad subject \ \ to \ \ \sum_{j=1}^{p}\phi_{j,1}^2 = 1$

The other ones are also computed this way,with the additional constraint of orthogonality.

looking at the summary of x.prcomp:

```
summary(x.prcomp)
```

```
## Importance of components:
##                           PC1     PC2     PC3    PC4     PC5     PC6     PC7
## Standard deviation     2.5675 1.80760 1.41533 1.2701 1.24347 1.20936 1.18911
## Proportion of Variance 0.1157 0.05732 0.03514 0.0283 0.02713 0.02566 0.02481
## Cumulative Proportion  0.1157 0.17297 0.20811 0.2364 0.26354 0.28920 0.31401
##                            PC8     PC9   PC10    PC11    PC12   PC13    PC14
## Standard deviation     1.17257 1.13811 1.1300 1.10314 1.06306 1.0544 1.04647
## Proportion of Variance 0.02412 0.02272 0.0224 0.02135 0.01983 0.0195 0.01921
## Cumulative Proportion  0.33813 0.36085 0.3833 0.40460 0.42443 0.4439 0.46315
##                           PC15    PC16   PC17    PC18    PC19    PC20    PC21
## Standard deviation     1.04261 1.03125 1.0240 1.01162 1.00631 1.00141 0.99795
## Proportion of Variance 0.01907 0.01866 0.0184 0.01795 0.01777 0.01759 0.01747
## Cumulative Proportion  0.48222 0.50088 0.5193 0.53723 0.55499 0.57259 0.59006
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.98907 0.98211 0.97023 0.96777 0.96142 0.95674 0.95113
## Proportion of Variance 0.01716 0.01692 0.01651 0.01643 0.01622 0.01606 0.01587
## Cumulative Proportion  0.60722 0.62414 0.64066 0.65709 0.67331 0.68936 0.70524
##                           PC29    PC30    PC31   PC32    PC33    PC34    PC35
## Standard deviation     0.93457 0.93059 0.91460 0.9092 0.89312 0.88414 0.88126
## Proportion of Variance 0.01532 0.01519 0.01468 0.0145 0.01399 0.01371 0.01362
## Cumulative Proportion  0.72056 0.73575 0.75043 0.7649 0.77893 0.79264 0.80626
##                           PC36    PC37    PC38    PC39   PC40    PC41    PC42
## Standard deviation     0.86921 0.85678 0.85033 0.83940 0.8306 0.82134 0.81608
## Proportion of Variance 0.01325 0.01288 0.01269 0.01236 0.0121 0.01184 0.01168
## Cumulative Proportion  0.81952 0.83240 0.84508 0.85744 0.8696 0.88138 0.89307
##                           PC43    PC44    PC45    PC46   PC47    PC48   PC49
## Standard deviation     0.78684 0.78003 0.76280 0.75966 0.7242 0.69908 0.6711
## Proportion of Variance 0.01086 0.01067 0.01021 0.01012 0.0092 0.00857 0.0079
## Cumulative Proportion  0.90393 0.91460 0.92481 0.93494 0.9441 0.95271 0.9606
##                           PC50    PC51    PC52    PC53    PC54    PC55    PC56
## Standard deviation     0.63961 0.61287 0.60474 0.57867 0.55261 0.51031 0.43619
## Proportion of Variance 0.00718 0.00659 0.00642 0.00587 0.00536 0.00457 0.00334
## Cumulative Proportion  0.96779 0.97438 0.98079 0.98667 0.99203 0.99659 0.99993
##                           PC57
## Standard deviation     0.06209
## Proportion of Variance 0.00007
## Cumulative Proportion  1.00000
```

one see that PC1 explains 0.1157 percent of the variability. PC2 0.05732 and PC3 0.03514.These components explains around 20% of the variability alone.Which can be seen in Cumulative Proportion section.

1.e

```r
library(gam)
```

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

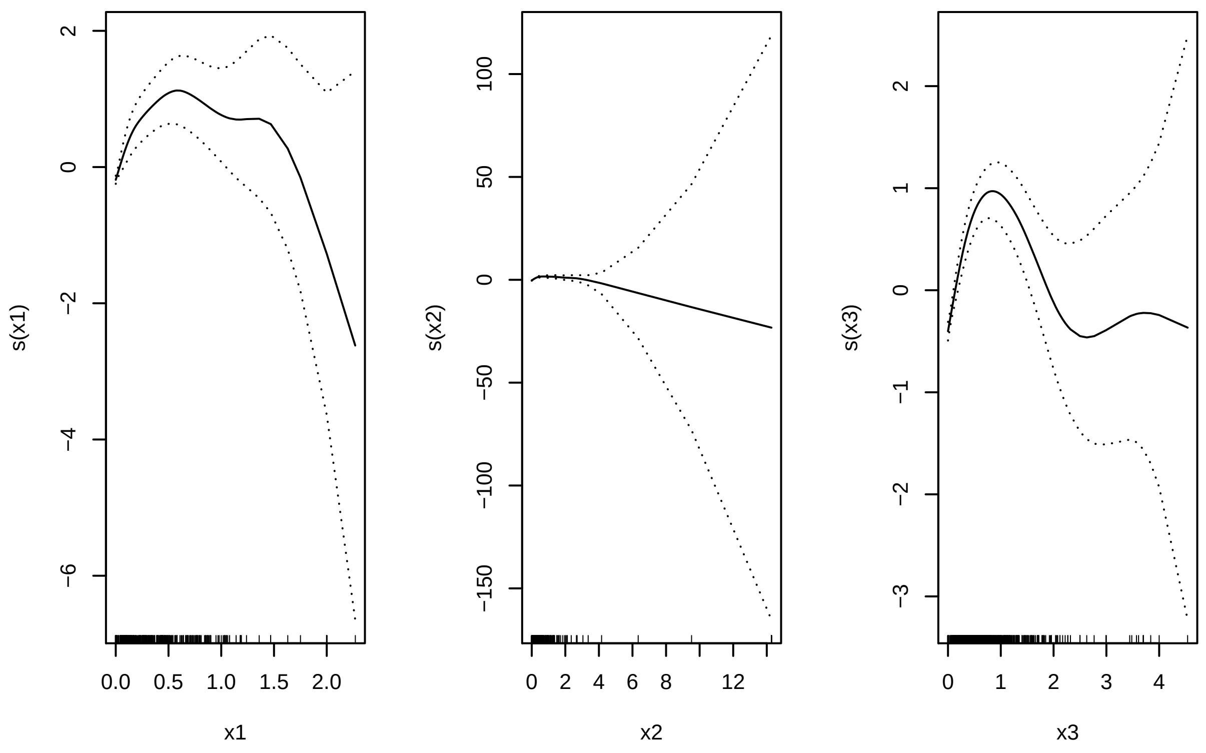
```
## Loading required package: foreach
```

```
## Loaded gam 1.20
```

```r
fit.gam = gam(y~s(x1)+s(x2)+s(x3),data=spam,
              subset=spam$train,family=binomial)
pred.gam = predict(fit.gam,spam[!spam$train,],type="response")>0.5

r=sum(ifelse(pred.gam==(spam[!spam$train,]$y),1,0))
error2=1-r/length(spam[!spam$train,]$y)

error2
```

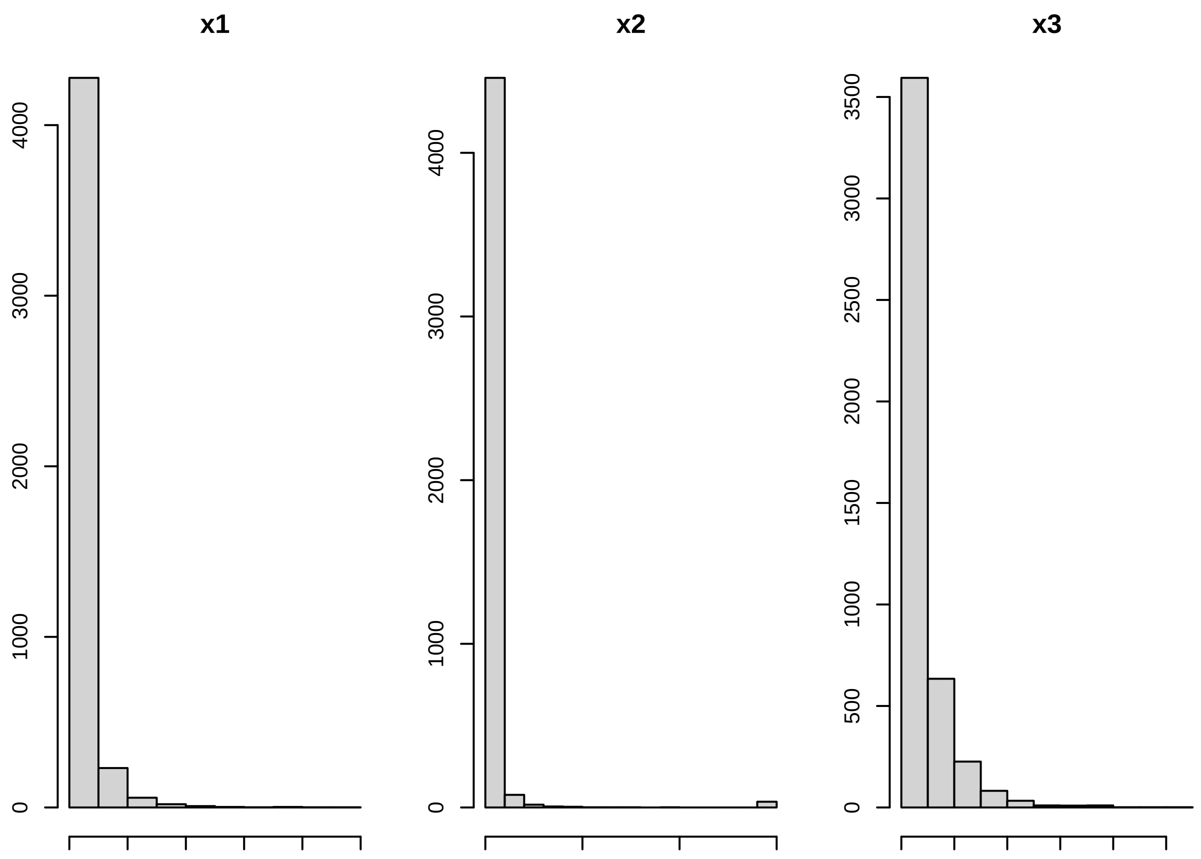
```
## [1] 0.2998369
```

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



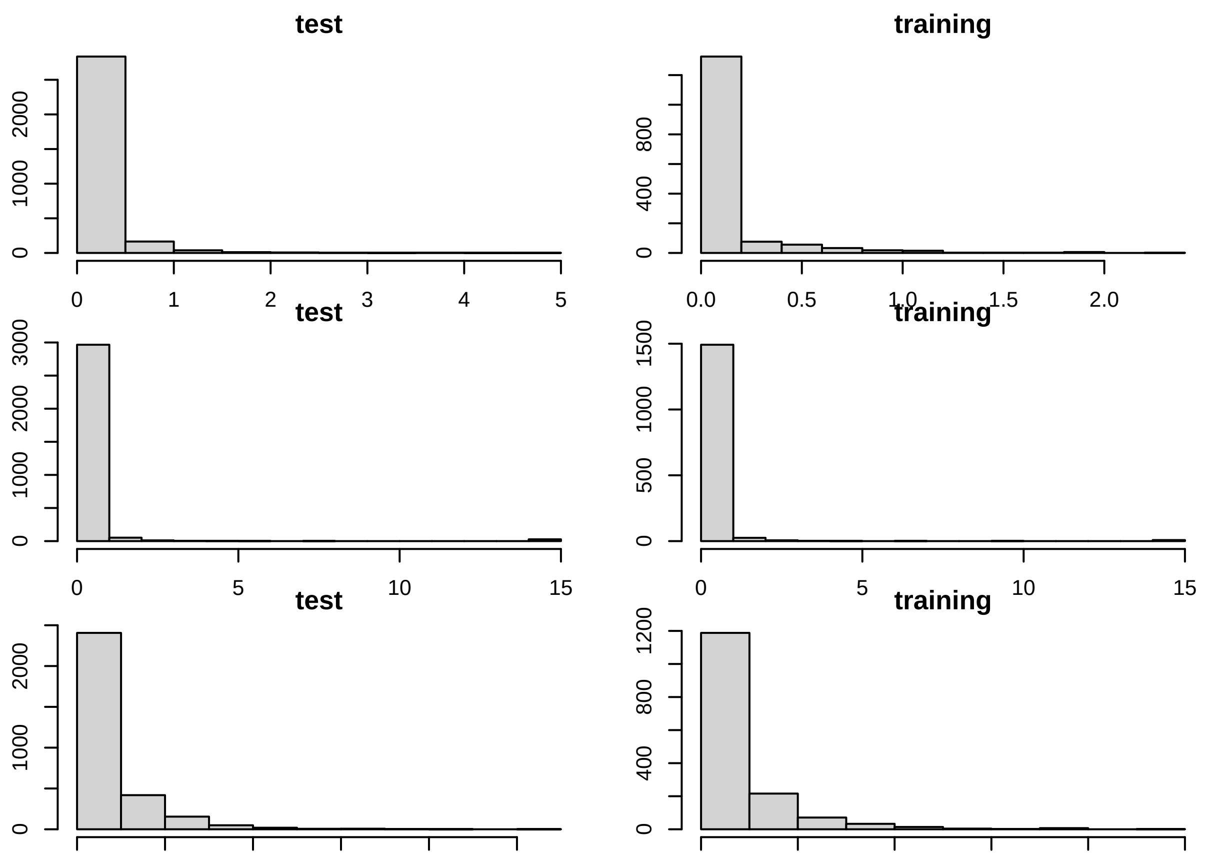
Making histograms

```
par(mfrow=c(1,3),mar=c(1, 2, 2, 2))
hist(spam$x1,main="x1",xlab = "x1")
hist(spam$x2,main="x2",xlab="x2")
hist(spam$x3,main = "x3",xlab="x3")
```



From the histograms one can see that their are frequency values that the data never takes in the the middle.One can see this especially in “x2”. This again leads to large confidence intervals on the plots of the model where their are few data point or where there is no data-points. In these sections their is large uncertainty to weather that part of the curve is linear or not. This is though not the reason for the the small improvement in the error. If one plot the histogram of each variable,one for the training data and one for the test data.One can see that the training data is quite similar to the training data.

```r
f=spam
f$train=NULL
f$y=NULL
r=names(f)

par(mfrow=c(3,2),mar=c(1, 2, 2, 2))
for(i in 1:3){
  s=eval(parse(text=paste("f$",r[i],sep="")))
  hist(s[which(spam$train!=TRUE)],main="test",xlab=NULL)
  hist(s[which(spam$train==TRUE)],main="training",xlab=NULL)

}
```



Thin plate splines are doing a better job on the part where we have a lot of data.Thin plate spline is a generalization of smooth spline.Depending on the value of lambda the spline looks different. As lambda increases the more linear the the look of the spline.Thin plate spline algorithm then have the freedom to find the lambda that minimizes the bias the most,while a linear model is just minimizing the the least squares.The linear model is constricted to be linear while the spline can shapeshift.If the best model is linear bought the linear model and the spline will give equal results.Here i am talking about the bias since one is not using for example k-fold cross validation,to find a model that also takes into account the variance,when it comes to the bias-variance tradeoff.

Another reason for the improvement could be that the angle of the linear model,or the slope of the linear model changes depending on the placement of the leverage points.This again will lead to a worse fit for the rest of the points.This problem those not occur when it comes to splines where the x axes is divided into sections or intervals.On each interval a polynomial is fitted using the data-points in that interval.How the last part of the spline looks, where the leverage points are ,is not effecting the rest of the spline.

```
library(gam)
fit.gam = gam(y~(x1)+(x2)+(x3),data=spam,
              subset=spam$train,family=binomial)

summary(fit.gam)
```

```
##
## Call: gam(formula = y ~ (x1) + (x2) + (x3), family = binomial, data = spam,
##     subset = spam$train)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -2.6371 -0.8762 -0.8762  1.2974  1.5121
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 2050.735 on 1535 degrees of freedom
## Residual Deviance: 1975.112 on 1532 degrees of freedom
## AIC: 1983.112
##
## Number of Local Scoring Iterations: 4
##
## Anova for Parametric Effects
##             Df  Sum Sq Mean Sq F value    Pr(>F)
## x1           1   32.87  32.867 32.0557 1.786e-08 ***
## x2           1    0.00   0.000  0.0004    0.9848
## x3           1   33.95  33.950 33.1116 1.049e-08 ***
## Residuals 1532 1570.78   1.025
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
fit.gam = gam(y~(x1)+s(x2)+(x3),data=spam,
              subset=spam$train,family=binomial)

summary(fit.gam)
```

```
##
## Call: gam(formula = y ~ (x1) + s(x2) + (x3), family = binomial, data = spam,
##     subset = spam$train)
## Deviance Residuals:
##    Min     1Q Median     3Q    Max
## -2.291 -0.787 -0.787  1.095  1.627
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 2050.735 on 1535 degrees of freedom
## Residual Deviance: 1794.836 on 1529 degrees of freedom
## AIC: 1808.836
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##             Df  Sum Sq Mean Sq F value    Pr(>F)
## x1           1   22.08  22.075  22.581 2.204e-06 ***
## s(x2)        1   46.14  46.144  47.202 9.291e-12 ***
## x3           1   23.51  23.506  24.044 1.042e-06 ***
## Residuals 1529 1494.74   0.978
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar Chisq    P(Chi)
## (Intercept)
## x1
## s(x2)             3      101.4 < 2.2e-16 ***
## x3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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

r=sum(ifelse(pred.gam==(spam[!spam$train,]$y),1,0))
error5=1-r/length(spam[!spam$train,]$y)

error5
```

```
## [1] 0.31354
```

Here i have taken a summary where all the covariates are linear.We can see that "x2" is not significant.In the second summary i added spline to "x2" and now "x2" is significant.Looking at the error of the last fit and comparing it to the error when adding splines to all of the three covariates the error rate does not change much,it actually increases a little bit.

```
fit.gam = gam(y~s(x1)+s(x2)+s(x3),data=spam,
              subset=spam$train,family=binomial)
pred.gam = predict(fit.gam,spam[!spam$train,],type="response")>0.5

r=sum(ifelse(pred.gam==(spam[!spam$train,]$y),1,0))
error=1-r/length(spam[!spam$train,]$y)

error
```

```
## [1] 0.2998369
```

My conclusion is therefore.When the the covariates are significant when they are linear,adding a spline to it will not do much,the spline will look linear,which can be seen in the plot of the model were all the terms in the additive model is splines. The plots highly none-linear in the larger covariate values,but they have big confidence intervals around them.The beginning of the plot is is very linear,with very narrow confidence intervals. So in the first knot interval of the spline the function chosen is linear.That is where most of the data is.The larger values of the covariates are the less relevant for the prediction since their is not a lot of point around these values,and the test data and training data closely resemble each other.

The difference in error adding splines to all covariates vs all covariates are linear,only gives around 0.02 of improvement. One would probably see a much better improvement,if more data existed for the larger values,and the training data looked more different from the test data,and also if k-fold cross-validation was applied, because of the flexibility of the spline method.I would therefore choose to go with the model where all terms in the additive model being splines.

1.f

```
fit.pc.gam = gam(y~s(PC1)+s(PC2)+s(PC3),data=d,
                 subset=d$train,family=binomial)
pred.pc.gam = predict(fit.pc.gam,d[!d$train,],type="response")>0.5

r=sum(ifelse(pred.pc.gam==(d[!d$train,]$y),1,0))
error3=1-r/length(d[!d$train,]$y)


fit.pcl.gam = gam(y~(PC1)+(PC2)+(PC3),data=d,
                  subset=d$train,family=binomial)
pred.pcl.gam = predict(fit.pcl.gam,d[!d$train,],type="response")>0.5
r=sum(ifelse(pred.pcl.gam==(spam[!spam$train,]$y),1,0))
error4=1-r/length(spam[!spam$train,]$y)

error3
```

```
## [1] 0.1223491
```

```
error4
```

```
## [1] 0.1311582
```

Principal component analysis is used when one has a large number of covariates. One creates new variables,,where each element in the new variable is a linear combinations of the original variables. This way one is conserving most of the information in the original variables and one can use less variables to explain most of the variability in the original variables.

The first principal component is the normalized linear combination that has the highest variability (variance).

$z_{i,1} = \phi_{1,1}x_{i,1} + \phi_{2,1}x_{i,2} + \phi_{3,1}x_{i,3} + ...\phi_{57,1}x_{i,57}$

each $z_{i,1}$ is the projection of x_i onto the principal component line.Doing this for all individuals gives the first principal component

$Z_1 = [z_{1,1}, z_{2,1}, z_{3,1}, ...z_{n,1}]^T$

Elements $\phi_{1,1}...\phi_{57,1}$ are called the principal component loading.These loadings are normalized so that.$\sum_{j=1}^{p}\phi_{j,1}^2 = 1$ .The vector that is created using these loading are the eigenvector with length 1,and gives the direction of the principal component. We want the distance between data points $x_{i,j}$,(how individual "i" is interpreted by variable "j"), and the mean of all $x_{i,j}$ for the individual "i" to be as big as possible.We do this for all "i" and all "j".Maximize the variance from mean. This mean get shifted to be 0,for simplicity. The One wants to find a principal component vector that

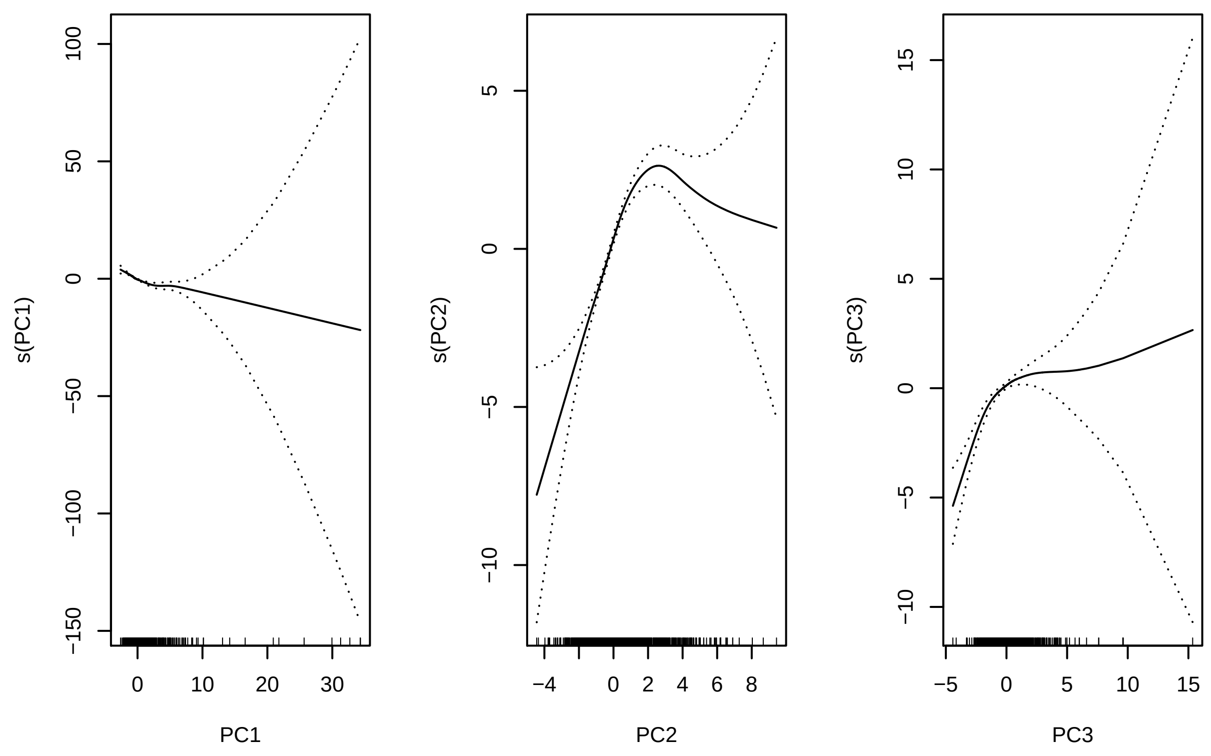
$maximize\ \ \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{n}\phi_{j,1}x_{i,j})^2 \quad subject\ \ to\ \ \sum_{j=1}^{p}\phi_{j,1}^2 = 1$

the expression being maximized is the sum or the total square distances between the projected data-points on to the principal component line and (0,0,. . . ..). One can find the second principal component in the same way as the first,but now one has the extra condition that the second principal component must be orthogonal or uncorrelated to the first. Third,uncorrelated with the second and the first and so on.

One see that the first few principal components explains most of the variability. Therefore i expect to see a huge decrease in error when using the first three principal components compared to just using the first three original explanetory variables.

This improvement might be do to , that no explanetory variable is more important then the other. If it had been, using PC could have worsen result.Lets say "x3" is the most important.Each element in any given PC is a kind of average,one would then down weight the importance of "x3",if one did not use all PC's.

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

plot(fit.pc.gam,se=T)
```



Including the none-linearity does improve the result from 0.1311582 to 0.1223491.One can see from the plot that their seem to be a none-linear relationship,but again the confidence interval are very wide,for large values of the covariates.

The same principal apply here.Since The test set closely resemble the training set test set will be very similar,the method that approximates the training data best will also give the smallest error. Mentioned in the previous exercise about the flexibility(shapeshifting quality) of Splines. Splines will follow the data better and each basis-function those not effect how another basis-function looks,other then that they have to meet each other and be double differentiable,while a linear model will be highly effected by the leverage points in the data,and therefore give a worse approximation for the rest or the most of the data.

Looking at the summary for linear PC1,PC2,PC3.

```
library(gam)

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

summary(fit.pc.gam)
```

```
##
## Call: gam(formula = y ~ s(PC1) + s(PC2) + s(PC3), family = binomial,
##     data = d, subset = d$train)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8164 -0.4849 -0.1729  0.3589  2.8941
##
```

```
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 2050.735 on 1535 degrees of freedom
## Residual Deviance: 974.3573 on 1523 degrees of freedom
## AIC: 1000.358
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## s(PC1)        1  172.93 172.928 183.183 < 2.2e-16 ***
## s(PC2)        1  141.46 141.456 149.844 < 2.2e-16 ***
## s(PC3)        1   41.49  41.492  43.952 4.659e-11 ***
## Residuals 1523 1437.74   0.944
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##              Npar Df Npar Chisq    P(Chi)
## (Intercept)
## s(PC1)             3     18.623 0.0003271 ***
## s(PC2)             3     75.796 2.220e-16 ***
## s(PC3)             3     23.193 3.682e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here one see that PC1 and PC2 is significant,while PC3 is not.

Looking at the summary for none-linear s(PC1),s(PC2),s(PC3).

```
library(gam)

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

summary(fit.pc.gam)
```

```
##
## Call: gam(formula = y ~ (PC1) + (PC2) + (PC3), family = binomial, data = d,
##     subset = d$train)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2785 -0.5583 -0.2271  0.3785  3.4331
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 2050.735 on 1535 degrees of freedom
## Residual Deviance: 1079.022 on 1532 degrees of freedom
## AIC: 1087.022
##
## Number of Local Scoring Iterations: 7
##
## Anova for Parametric Effects
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## PC1          1   144.0 143.981 16.6682 4.682e-05 ***
## PC2          1   192.4 192.402 22.2737 2.580e-06 ***
## PC3          1    14.0  13.956  1.6156    0.2039
## Residuals 1532 13233.5   8.638
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here one see that now all PC's are significant.

We see from the summary,and the errors that adding a spline to the three PC's gives a better model.In this case i would also choose the model where all covariates are none-linear.

1.g,

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

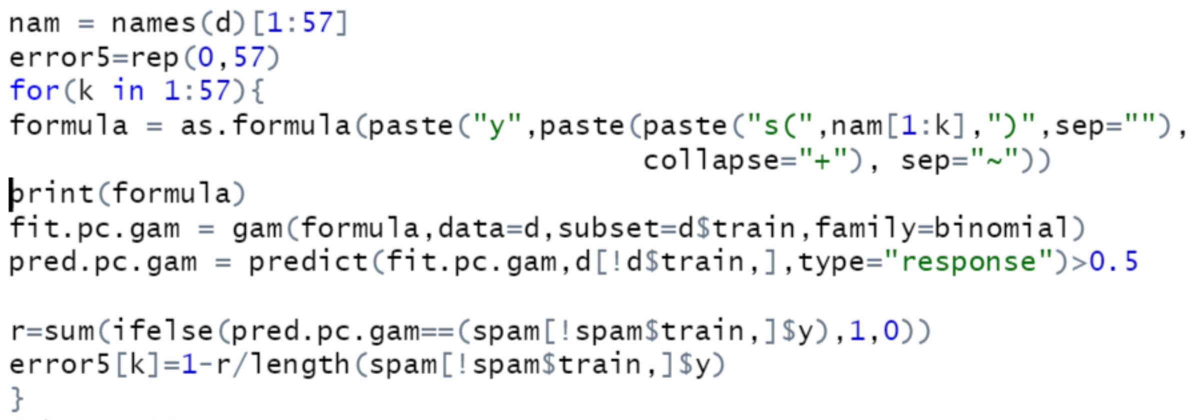
```
## y ~ s(PC1) + s(PC2) + s(PC3) + s(PC4) + s(PC5) + s(PC6) + s(PC7) +
##     s(PC8) + s(PC9) + s(PC10) + s(PC11) + s(PC12) + s(PC13) +
##     s(PC14) + s(PC15) + s(PC16) + s(PC17) + s(PC18) + s(PC19) +
##     s(PC20)
```

```
fit.pc20.gam = gam(formula,data=d,subset=d$train,family=binomial)
pred.pc20.gam = predict(fit.pc20.gam,d[!d$train,],type="response")>0.5

r=sum(ifelse(pred.pc20.gam==(spam[!spam$train,]$y),1,0))
error4=1-r/length(spam[!spam$train,]$y)

error4
```

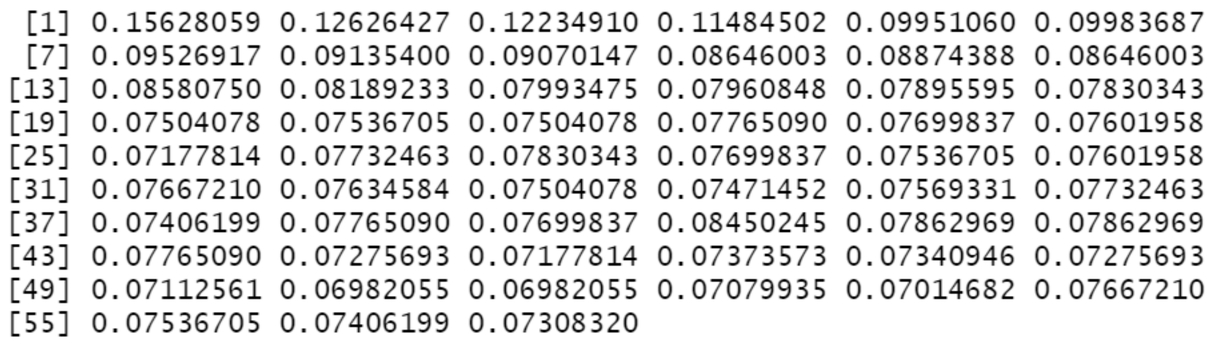
```
## [1] 0.07536705
```

Increasing principal component will reduce the error,since the more component the more variability explained.The training and test data for all covariates look similar,the error is reduced. The more PC's included the more chance of overfitting. If the test data had been a little different then this could be seen. A model with the first 20 principal components makes sense.Using less than 50 percent of the total PC´s.

1.h,



```
nam = names(d)[1:57]
error5=rep(0,57)
for(k in 1:57){
formula = as.formula(paste("y",paste(paste("s(",nam[1:k],")",sep=""),
                                     collapse="+"), sep="~"))
print(formula)
fit.pc.gam = gam(formula,data=d,subset=d$train,family=binomial)
pred.pc.gam = predict(fit.pc.gam,d[!d$train,],type="response")>0.5

r=sum(ifelse(pred.pc.gam==(spam[!spam$train,]$y),1,0))
error5[k]=1-r/length(spam[!spam$train,]$y)
}
```



```
 [1] 0.15628059 0.12626427 0.12234910 0.11484502 0.09951060 0.09983687
 [7] 0.09526917 0.09135400 0.09070147 0.08646003 0.08874388 0.08646003
[13] 0.08580750 0.08189233 0.07993475 0.07960848 0.07895595 0.07830343
[19] 0.07504078 0.07536705 0.07504078 0.07765090 0.07699837 0.07601958
[25] 0.07177814 0.07732463 0.07830343 0.07699837 0.07536705 0.07601958
[31] 0.07667210 0.07634584 0.07504078 0.07471452 0.07569331 0.07732463
[37] 0.07406199 0.07765090 0.07699837 0.08450245 0.07862969 0.07862969
[43] 0.07765090 0.07275693 0.07177814 0.07373573 0.07340946 0.07275693
[49] 0.07112561 0.06982055 0.06982055 0.07079935 0.07014682 0.07667210
[55] 0.07536705 0.07406199 0.07308320
```

cumulative variability increase adding more PC‘s.Looking at for example the length of some of the principal components could give some idea of the contribution of each principal component.I am looking at the length as a rough measure of how each principal component changes the models direction.

PC1+PC2+PC3. . . . .

Thinking about the model as a linear combination of vectors PC1,PC2 . . . .. and so on,subject to least square.

```r
lenght=rep(0,57)
for(i in 1:57){
  lenght[i]=sqrt(sum((x.prcomp$x[,i])^2))

}
lenght
```

```
##  [1] 174.134687 122.597530  95.992181  86.142518  84.335971  82.022454
##  [7]  80.649371  79.527521  77.190109  76.641644  74.818865  72.100405
## [13]  71.513622  70.975045  70.713154  69.942527  69.453275  68.611563
## [19]  68.251394  67.918688  67.684579  67.082152  66.609689  65.804148
## [25]  65.637692  65.206989  64.889456  64.508488  63.385607  63.115744
## [31]  62.031242  61.668182  60.574661  59.965330  59.769978  58.952737
## [37]  58.109960  57.672097  56.930804  56.332359  55.706202  55.349572
## [43]  53.366040  52.904401  51.735763  51.522640  49.117039  47.414124
## [49]  45.513860  43.380430  41.566589  41.015592  39.247068  37.480078
## [55]  34.611179  29.583965   4.210975
```

One can especially see that PC57 is not contributing much to the direction of the model in 57 dimensional space.The further one goes into the PC‘s list,the more orthogonality conditions are put on to the PC.Which gives it less variability to move,and so it will give less contribution to the direction of the model vector. Also the eigenvalue will be smaller for PC2 compared to PC1 and so on,because of the iterative procedure,of the PCA method. Where one is finding the max and then the next max which is smaller then the previous max and so on.

Looking at the error list the model that gives the smallest error rate is the model where the 50 first principal components are included. I would not choose that model because of the risk of overfitting. The first 20 PC‘s makes sense,from exercise 1,g.

1,i,

In exercise 1.a, one used the glm function, using all covariates to create a logistic model. In the next exercises one used principal components as covariates.

In the last part one used the gam function and did kind of the same procedure,with the additional functionality of a gam model,where one could use splines instead of just polynomials.

One can see that bought function(gam and glm) are doing an almost equal job. Looking at the error vectors(calculating error when incrementally increasing PC‘s) in 1.j and in 1.c, gam has a larger max but a smaller min.

A lot of models have been tested.One would use different models for different goals. The data set in this assignment is not that big,so calculating the models have not been very computer intensive. The most computer intensive calculation was in 1.j, where one had to calculate 57 different models incrementally increasing number of PC components.One could imagine a scenario where a lot of computer power is needed,in these situations choosing a models in 1.f with splines on all PC‘s would make sense,if the goal is prediction. For interpretability, i would start with the model in 1.a.The error term in 1,a is competitive with the rest of the models which has less covariates included,and at the same time more interpretable.

Possible weaknesses:

(1).Procedure used only divided the data into a test set and training set. This procedure therefore does not takes into account choosing a model based on the bias-variance tradeoff.

(2)Also the spline function in r has a default knot value of 10.Its a thin plate spline,which does not have the default natural spline setting,where one place a knot on each data-point.Therefore its not only lambda that determines the flexibility ,but also the choice of knots.The knots is a factor that contritibutes to determining the flexibility of the splines together with the optimal lambda which is automatically optimized,after one has chosen a specific number of knots.If i am not mistaken.

One can set k=-1 in s(). s(k=-1),this will then automatically choose the optimal k value(knot value),using cross validation.

(3)The data is skewed.One of the assumptions when applying linear regression with normal errors is that the data should not be skewed.Log transforming the data would be something that could be considered.

Combining (1) and (2) together with (3) could improve the analyses.