STK2100 - Machine learning and statistical methods

Mandatory assignment 2 of 2

Rohullah Akbari

4/29/2021

Problem 1)

We will start with loading the data.

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

a)

We will now run standard logistic regression with fitting based on the training data and evaluating based on the test data.

```
fit = glm(y~.-train,data=spam,subset=spam$train,family=binomial)
pred = predict(fit,spam[!spam$train,],type="response")>0.5
```

For this assignment we will use the MSE error rate for prediction performance. The test error is given by

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Using this to code the error:

```
ytest = spam[(!spam$train),"y"]
test_error = mean((ytest - pred)^2)
print(test_error)
```

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

In this case we obtain the measure to be at ca. 0.085.

b)

An alternative to use all the explanatory variables is to reduce the dimension by principal components. In our case we have 57 different explanatory variables, and maybe not all of them are necessary to have. Therefore, we use principal components which can be computed by:

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

where the option **scale=TRUE** result in that the variables are scaled to have variances equal to 1 before transformation. This scale transformation is a reasonable option because, short summarized, it aims to find the directions of maximum variance in the spam data and then it will project it onto a new subspace with

fewer or equal dimensions than the original one. This results in reduction of the dimensions by transforming the old variables into a set of new variables of smaller sets, and at the same time we try to lose the unnecessary information.

Now we will try a logistic model based on the first 2 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

#computing error:
ytest1 = spam[(!d$train),"y"]
test_error1 = mean((ytest1 - pred.pc)^2)
print(test_error1)
```

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

In this case, by using the first 2 principal components, we get a MSE error rate at ca. 0.132 which is higher compared to task above.

c)

We will try logistic regression by using the first k principal components for different values of k. To do that, as shown above, we start by transforming the variables into the principal components.

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

After that, we will apply logistic regression with the first k principal components. We will loop over every k values while trying to find the k values that gives the least MSE error.

```
min_error = 1000 #choosing a big value for min_error in order to find the min error
best_k = 0
for(k in 1:57){
  #fiting and predict.
  glm_fit = glm(y~.-train, data = d[,c(1:k,58,59)],family=binomial,subset=d$train)
  y_pred = predict(glm_fit,d[!d$train,],type="respons") > 0.5
  #computing error
  error = mean((d[(!d$train),"y"] - y_pred)^2)

  #if-statement to find the min_error
  if(error < min_error){
    min_error = error
    best_k = k
  }
}

cat("Best k-value: ", best_k, " with MSE: ",min_error)
```

```
## Best k-value:  45  with MSE:  0.07536705
```

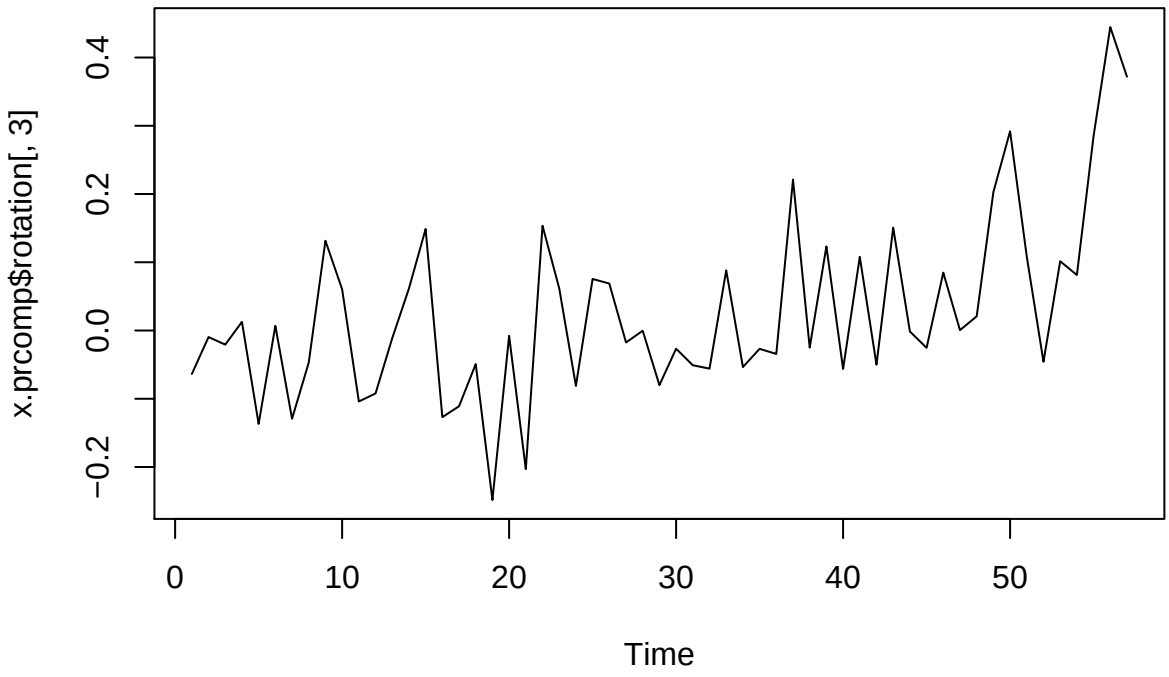
As the output from R shows above, the k value that gives the best result is $k = 45$ with an error at $MSE = 0.075$. By comparing this measure with the result from the task above, it is clear that logistic regression with using of k principal components gives definitely better result.

d)

The command below shows the weights for the 57 different explanatory variables for the first principal component (and similarly for the other components).

But I am not sure how to interpret the plot or the components.

```
plot.ts(x.prcomp$rotation[,3])
```



e)

In this task, we will concentrate on non-linear terms in the model. A generalized additive model (GAM) based on the first three explanatory variables can be fitted and predicted by the following commands:

```
library(gam)
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
```

We will compare this prediction with a similar model with only linear terms, logistic regression.

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

By comparing the MSE error, we get:

```
MSE.gam = mean((spam[(!spam$train),"y"] - pred.gam)^2)
MSE.glm = mean((spam[(!spam$train),"y"] - pred.glm)^2)
cat("MSE error for GAM: ",MSE.gam)
```

```
## MSE error for GAM:  0.2998369
```

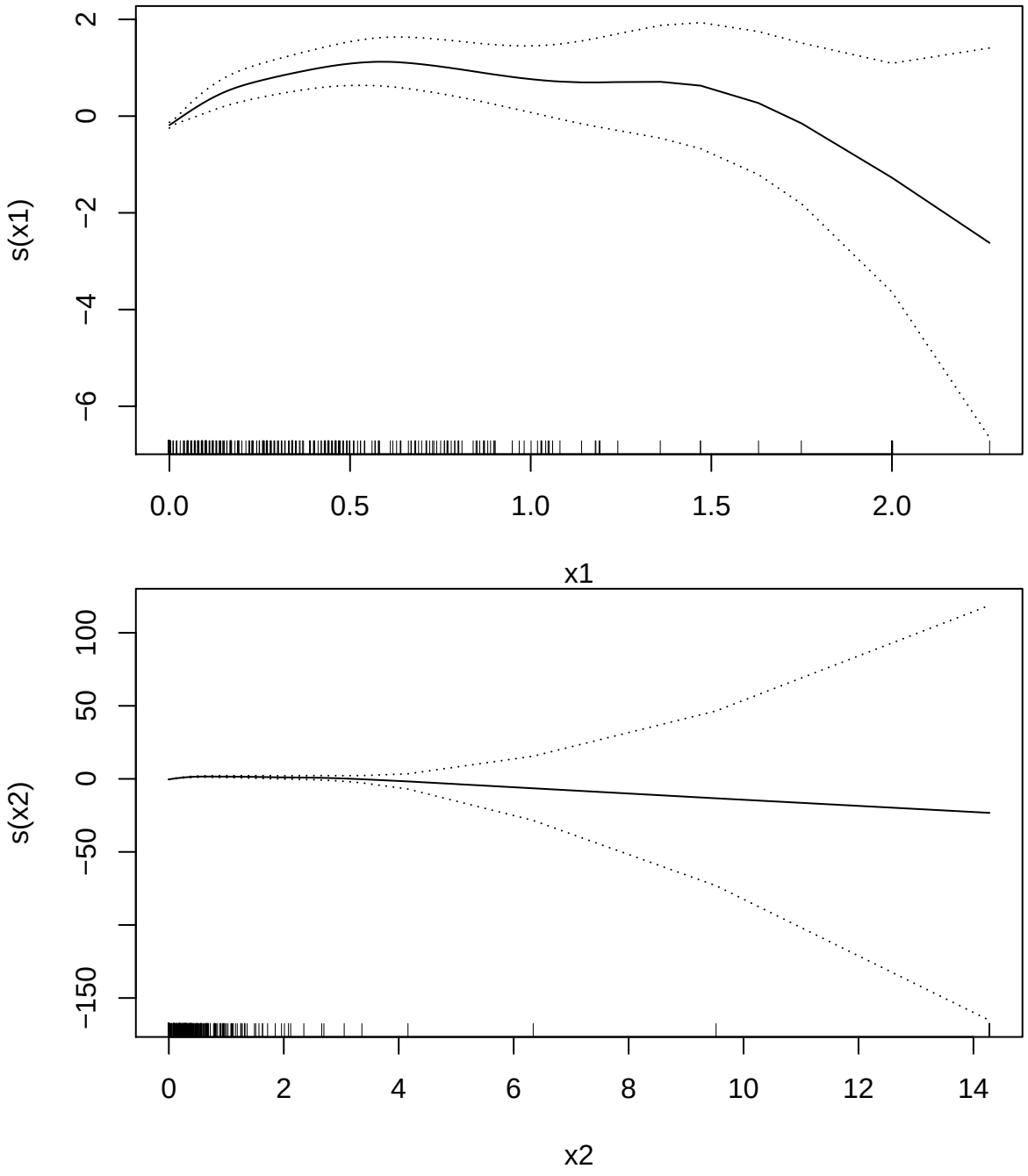
```
cat("MSE error for GLM: ",MSE.glm)
```

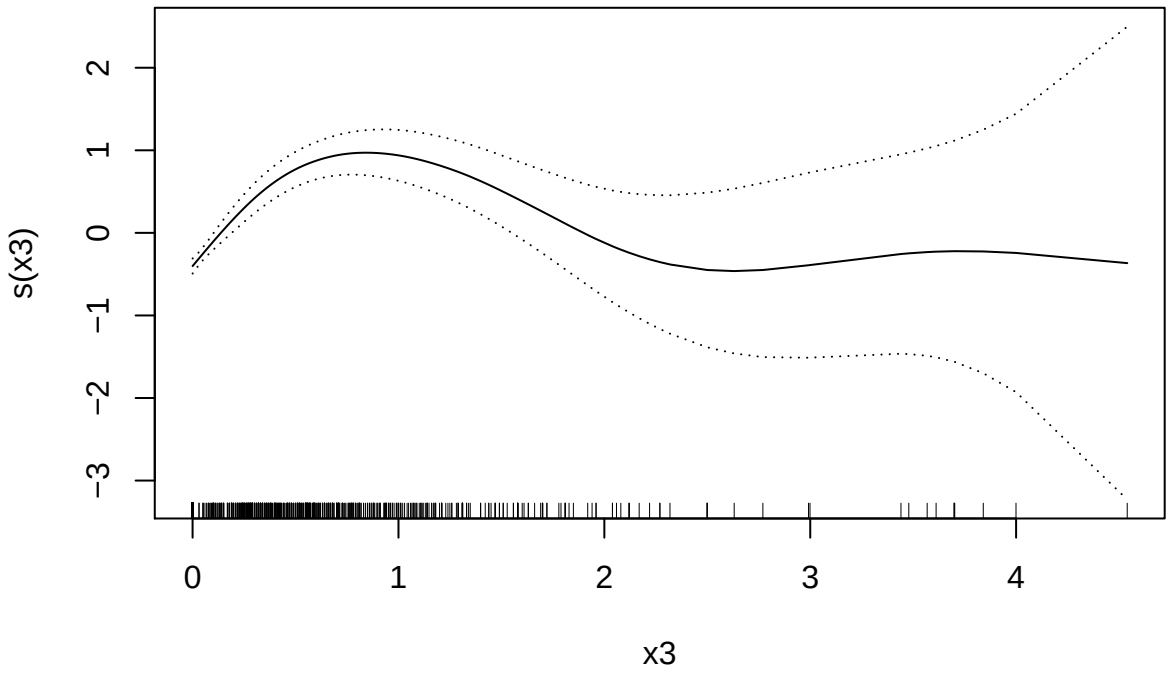
```
## MSE error for GLM:  0.3615008
```

From the output above we have that the MSE error for GAM is lower compared to the GLM. We conclude with we get improvement by including the non-linear terms.

The plot of the estimated functions for the non-linear terms:

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

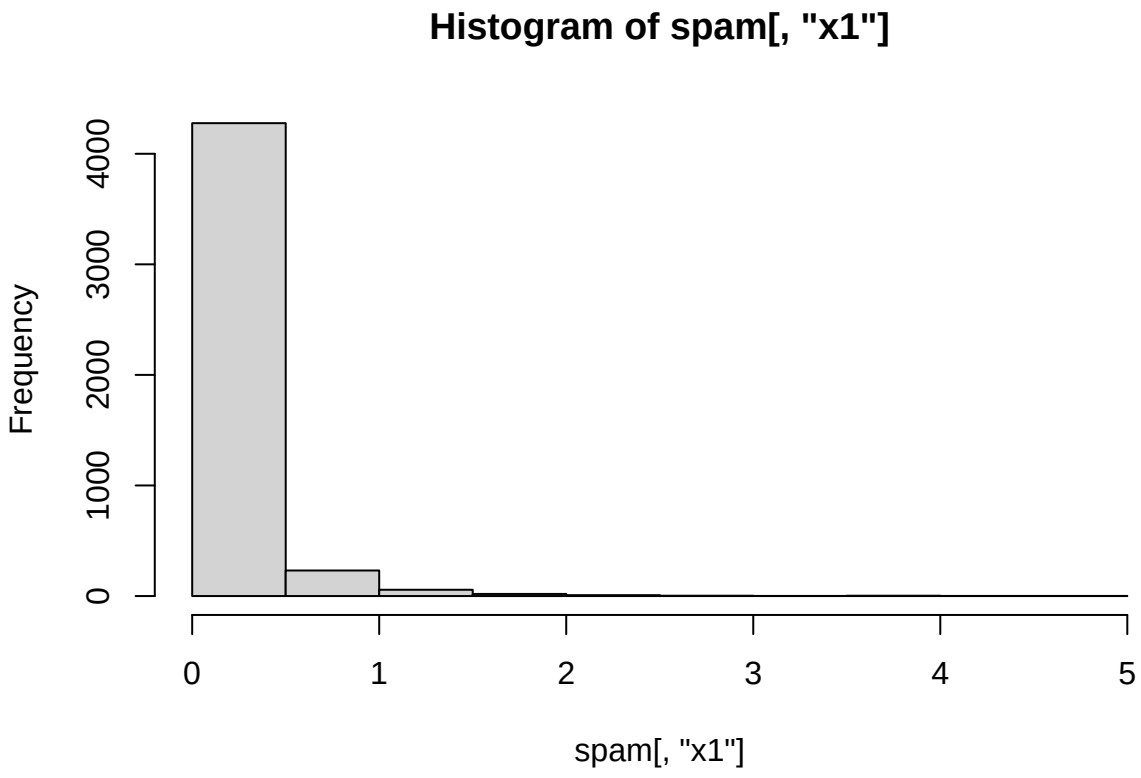




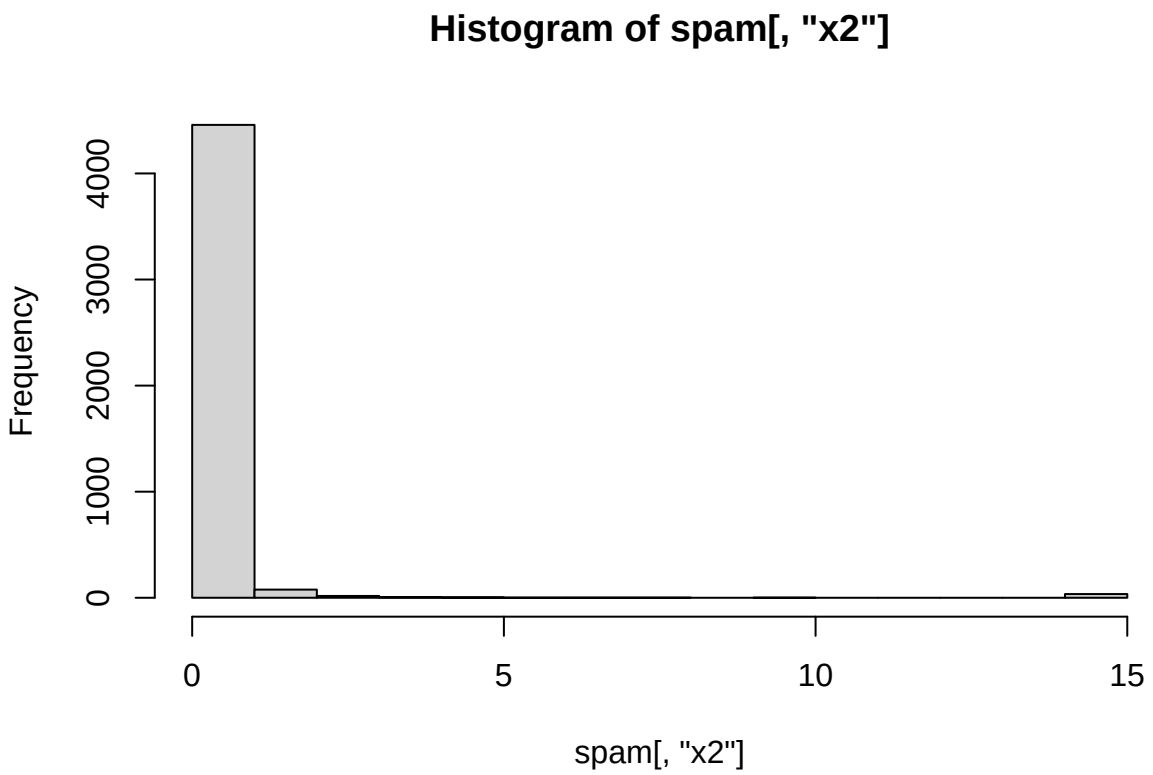
The figures are showing the fitted GAM models as function of non-linear terms. It is easy to observe the non-linearity relation between explanatory and response variables.

We can also make histogram for the variables (see below). From the histograms we can see high peaks on the left side, which in this case, describes the frequencies of some specific words. Since we have only few high peaks in the histogram means that there is very few words that are repeated frequently.

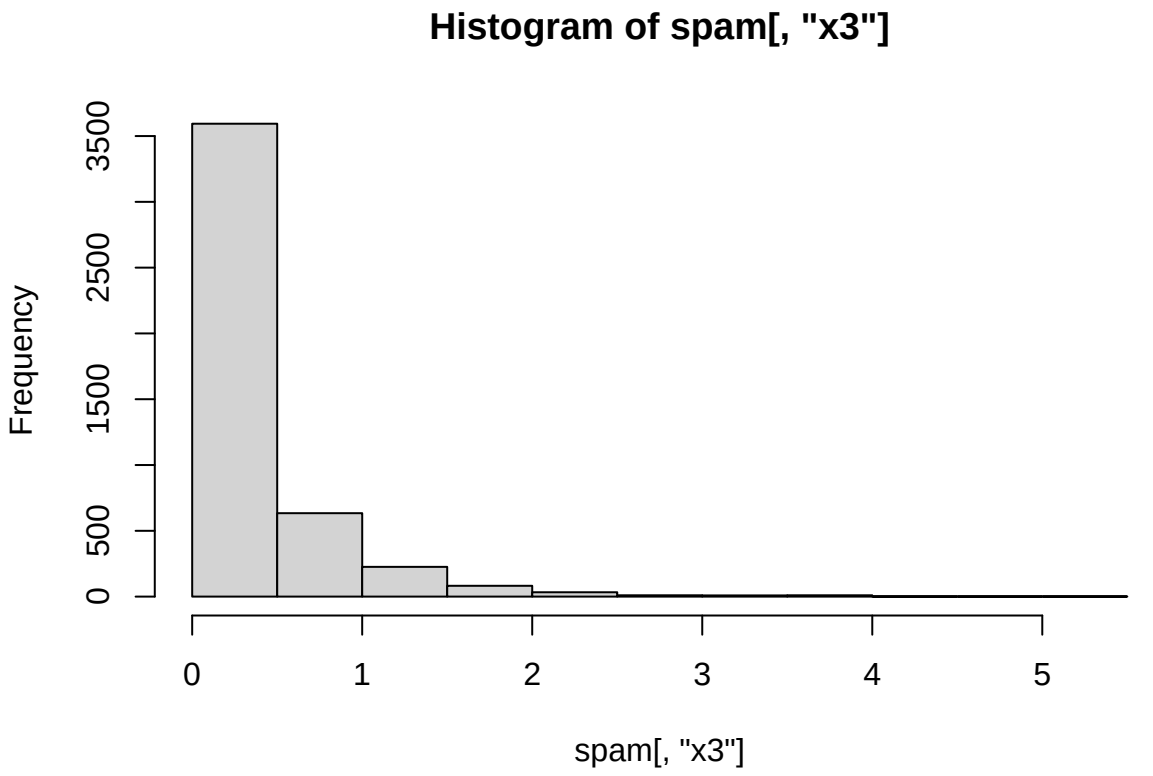
```
#The histogram for the variables:
hist(spam[,"x1"])
```



```
hist(spam[,"x2"])
```




```
hist(spam[,"x3"])
```



f)

Here we repeat the procedure from (e) but now based on the first three principal components.

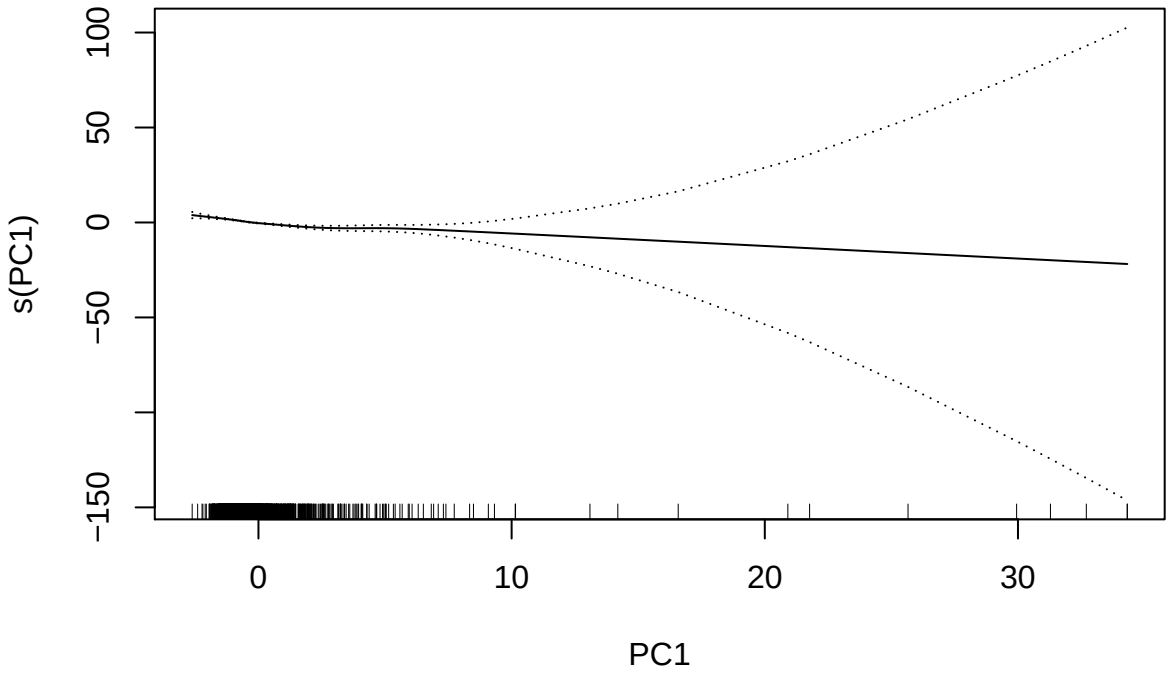
```
fit.pc.gam = gam(y~s(PC1)+s(PC2)+s(PC3),data=d,subset=d$train,family=binomial)
pred.pc.gam = predict(fit.pc.gam,d[!d$train,],type="response")>0.5

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

MSE.pc.gam = mean((d[(!d$train),"y"] - pred.pc.gam)^2)
MSE.pc.glm = mean((d[(!d$train),"y"] - pred.pc.glm)^2)
cat("MSE error for GAM: ",MSE.pc.gam)
```

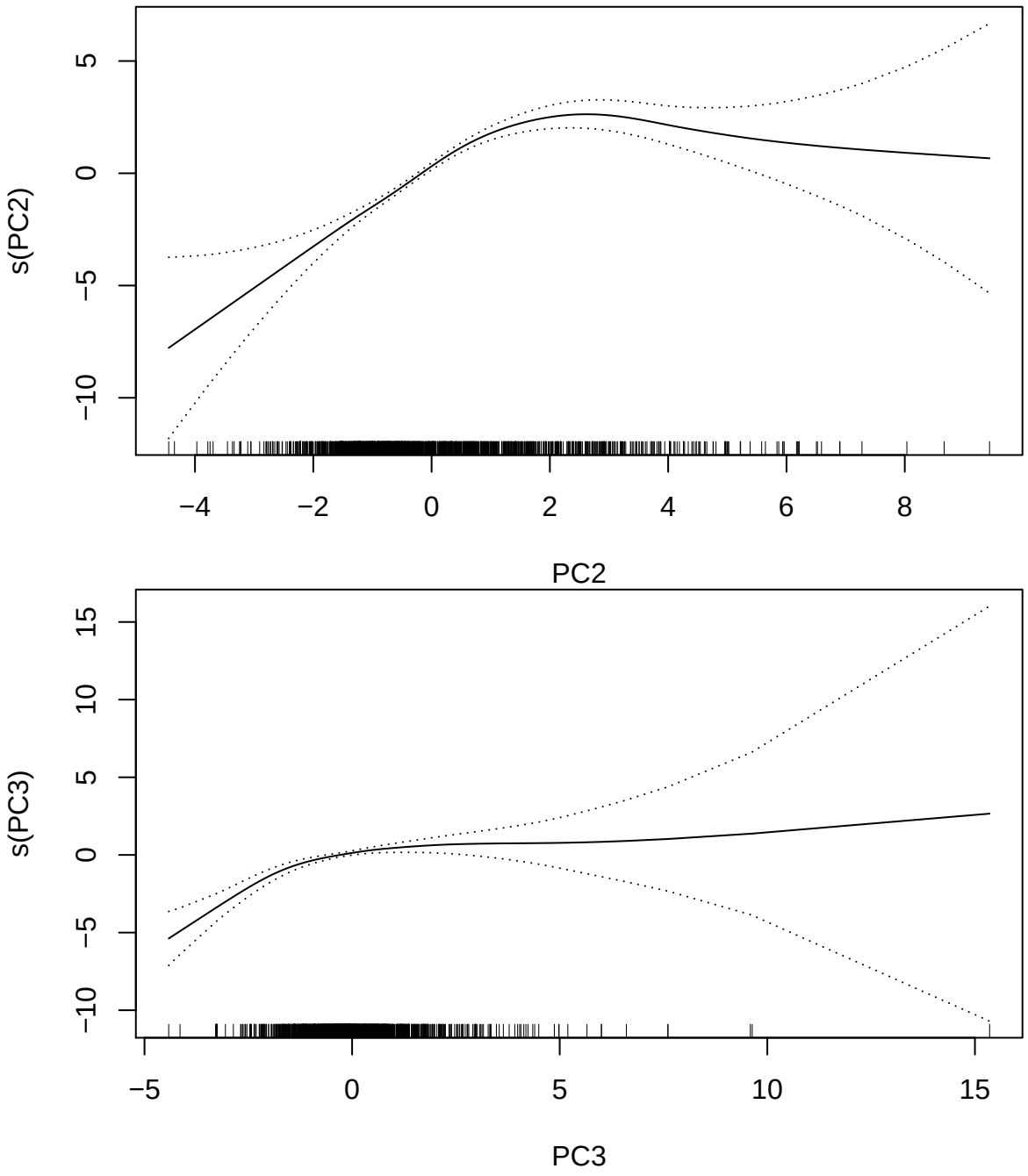
```
## MSE error for GAM:  0.1223491
```

```
cat("MSE error for GLM: ",MSE.pc.glm)
```

```
## MSE error for GLM:  0.1311582
```

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





We get the same result as above, got lower for GAM model. It means that by adding non-linear terms we get improvements and more precise model. Another interesting finding was that the MSE error for both GAM and GLM model decrease with using 3 principal components instead of the original variables.

The figures are showing the fitted GAM models as function of first three principal components.

g)

By adding more principal components we get

```
for(k in c(10,15,20,25)){
  nam = names(d)[1:57]
```

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

  #computing error:
  MSE.pc.gam = mean((d[(!d$train),"y"] - pred.pc.gam)^2)
  cat("k = ",k," MSE error for GAM: ",MSE.pc.gam)
  print("|")
}
```

```
## k =  10  MSE error for GAM:  0.08646003[1] "|"
## k =  15  MSE error for GAM:  0.07993475[1] "|"
## k =  20  MSE error for GAM:  0.07536705[1] "|"
## k =  25  MSE error for GAM:  0.07177814[1] "|"
```

Here the code has been executed with different k-values: 10,15,20 and 25. We can see that the GAM model with increasing k-value will result in lower in MSE error rate. Therefore, every k-value in this task does better than k = 3 from task above.

h)

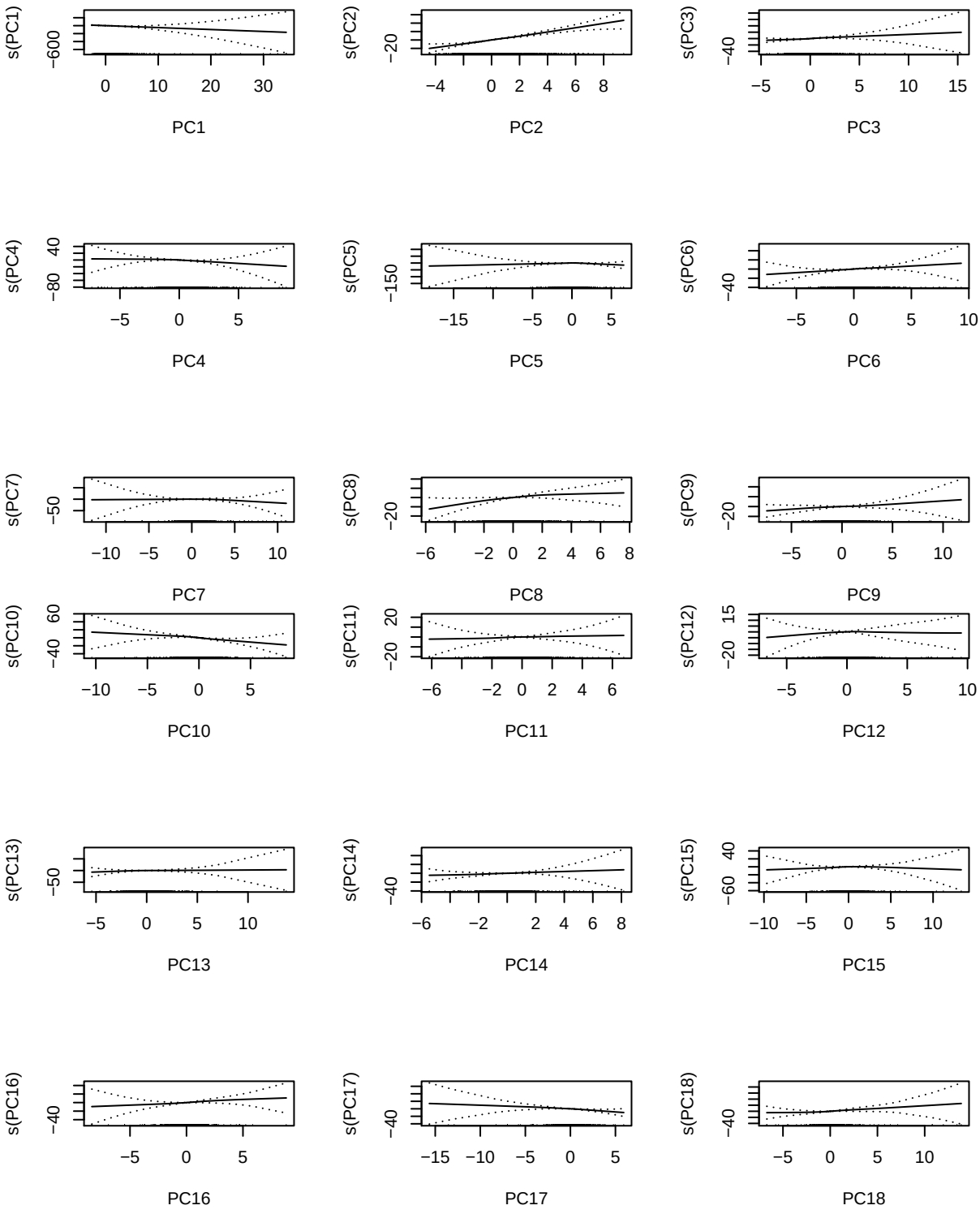
Finally, trying out non-linear structures on the principal components for different values of k. To do this, we can use the same procedure as in task c) but with a little modification inspired from task g).

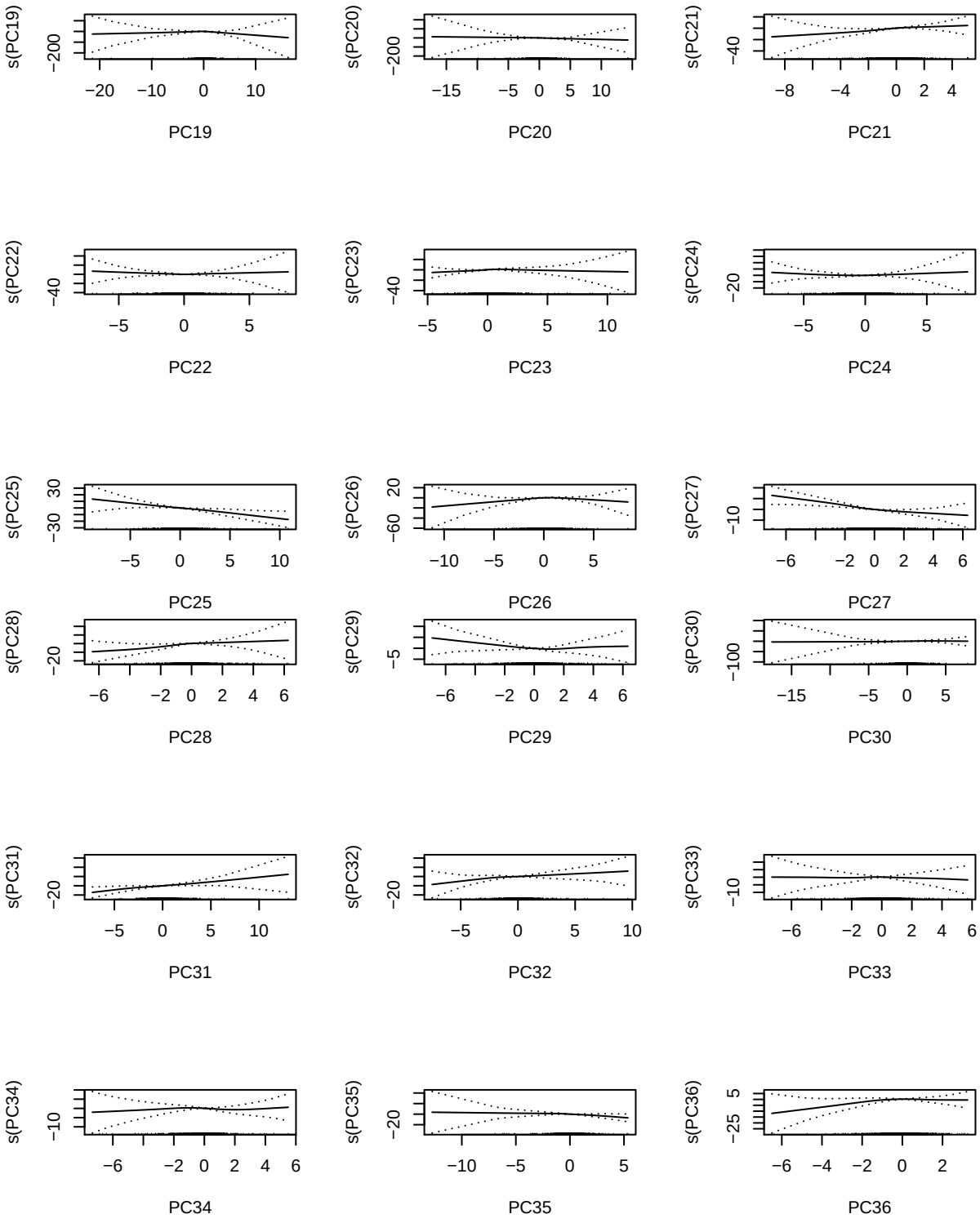
```
min_error = 1000 #choosing a big value for min_error in order to find the min error
best_k = 0
for(k in 1:57){
  nam = names(d)[1:57]
  formula = as.formula(paste("y",paste(paste("s(",nam[1:k],")",sep=""), collapse="+"), sep="~"))
  fit.pc.gam.1 = gam(formula,data=d[,c(1:k,58,59)],subset=d$train,family=binomial)
  pred.pc.gam.1 = predict(fit.pc.gam.1,d[!d$train,],type="response")>0.5

  #computing error:
  MSE = mean((d[(!d$train),"y"] - pred.pc.gam.1)^2)

  #if-statement to find the min_error
  if(MSE < min_error){
    min_error = MSE
    best_k = k
  }
}

cat("Best k-value: ", best_k, " with MSE: ",min_error)
```

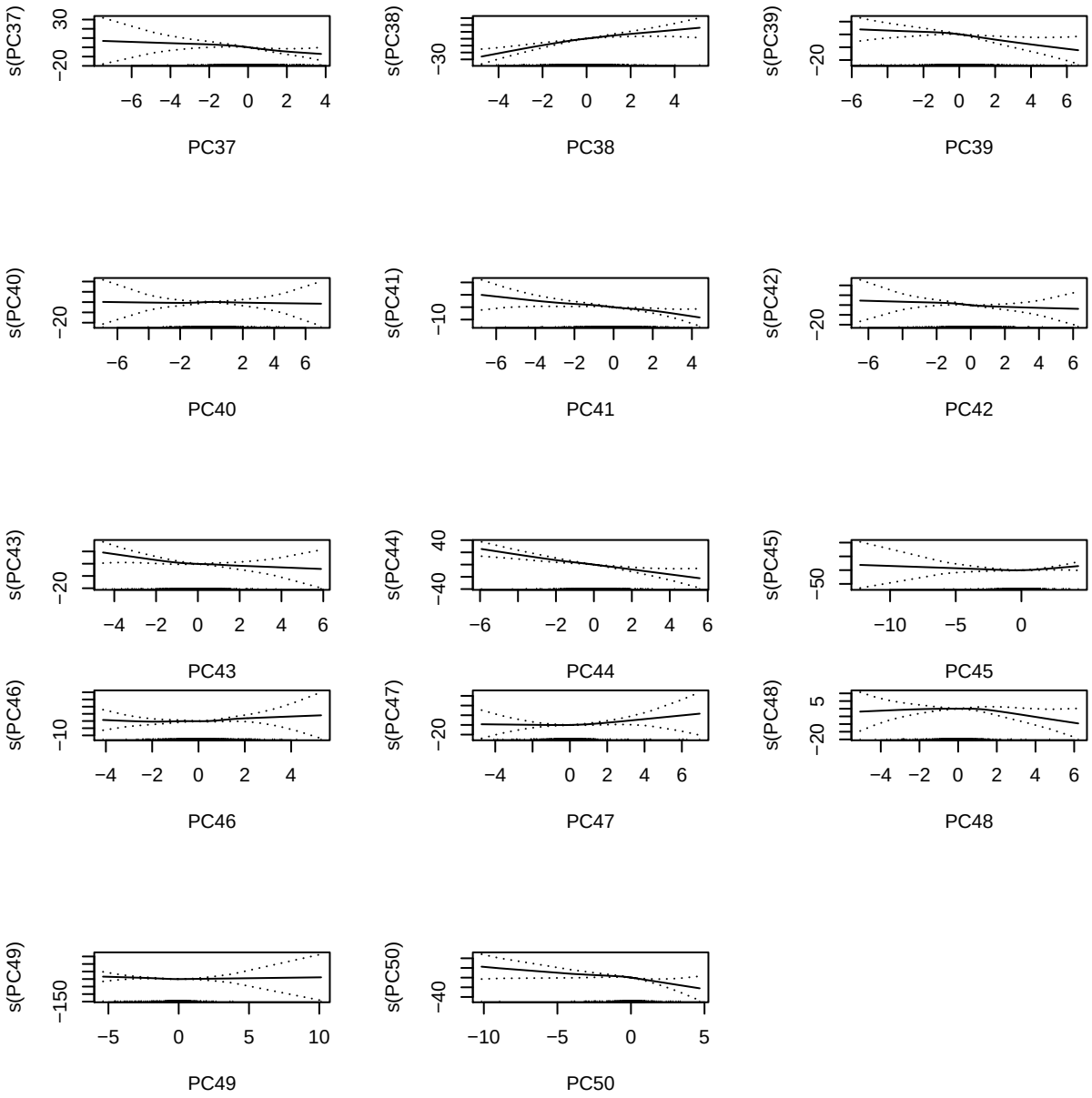
```
## Best k-value:  50  with MSE:  0.06982055
```

By using this procedure we obtain that k = 50 is the most optimal value which guarantees lowest error rate compared to the other values. The MSE error was founded to be at around 0.067 for k = 50.

```
k = 50
nam = names(d)[1:57]
  formula = as.formula(paste("y",paste(paste("s(",nam[1:k],")",sep=""), collapse="+"), sep="~"))
  fit.50.gam = gam(formula,data=d,subset=d$train,family=binomial)

  par(mfrow=c(3,3))
  plot(fit.50.gam,se=T)
```


s(PC1)
−600
0 10 20 30
PC1
s(PC2)
−20
−4 0 2 4 6 8
PC2
s(PC3)
−40
−5 0 5 10 15
PC3
s(PC4)
40
−80
−5 0 5
PC4
s(PC5)
−150
−15 −5 0 5
PC5
s(PC6)
−40
−5 0 5 10
PC6
s(PC7)
−50
−10 −5 0 5 10
PC7
s(PC8)
−20
−6 −2 0 2 4 6 8
PC8
s(PC9)
−20
−5 0 5 10
PC9
s(PC10)
60
−40
−10 −5 0 5
PC10
s(PC11)
20
−20
−6 −2 0 2 4 6
PC11
s(PC12)
15
−20
−5 0 5 10
PC12
s(PC13)
−50
−5 0 5 10
PC13
s(PC14)
−40
−6 −2 2 4 6 8
PC14
s(PC15)
40
−60
−10 −5 0 5 10
PC15
s(PC16)
−40
−5 0 5
PC16
s(PC17)
−40
−15 −10 −5 0 5
PC17
s(PC18)
−40
−5 0 5 10
PC18


s(PC19)
−200
−20 −10 0 10
PC19
s(PC20)
−200
−15 −5 0 5 10
PC20
s(PC21)
−40
−8 −4 0 2 4
PC21
s(PC22)
−40
−5 0 5
PC22
s(PC23)
−40
−5 0 5 10
PC23
s(PC24)
−20
−5 0 5
PC24
s(PC25)
−30 30
−5 0 5 10
PC25
s(PC26)
−60 20
−10 −5 0 5
PC26
s(PC27)
−10
−6 −2 0 2 4 6
PC27
s(PC28)
−20
−6 −2 0 2 4 6
PC28
s(PC29)
−5
−6 −2 0 2 4 6
PC29
s(PC30)
−100
−15 −5 0 5
PC30
s(PC31)
−20
−5 0 5 10
PC31
s(PC32)
−20
−5 0 5 10
PC32
s(PC33)
−10
−6 −2 0 2 4 6
PC33
s(PC34)
−10
−6 −2 0 2 4 6
PC34
s(PC35)
−20
−10 −5 0 5
PC35
s(PC36)
−25 5
−6 −4 −2 0 2
PC36



i)

We started by using standard logistic regression with fitting on the training data and evaluating on test data. By using this method, we got an MSE at ca. 0.085. Then, we tried to run logistic regression by using the first k principal components for different values of k which resulted MSE at 0.075. By comparing this two methods, we obtained better result for using the first k principal components. Using this findings, we can conclude that using principal components analysis can improve the model when we have many variables that, more or less, measure similar things.

The next step was to look at some non-linear terms in the model. We started with a generalized additive model (GAM) based on the first three explanatory variables and compared it to a similar model with only linear terms. This resulted in a lower MSE error rate for the GAM model. The very same procedure was repeated for based on the first three principal components, and yet again adding non-linear terms resulted in lower MSE error rate. So we can conclude with by adding non linear terms can improve the prediction performance model. An interesting finding with method was that, we got lower MSE error for increasing

k-value.

There are possible weaknesses with the experiments that are made. One of them could be that we used only one training-test split which could lead unstable result. For example the result could change if we split the data set again. If the same test set is used to choose between models, the final error measure will be too optimistic.