Lab 4

Attila Lazar

04.11.2020

data

We load the dataset and split train and test datasets

```
load("data/dat.RData")
data <- d
set.seed(1234)
n <- nrow(data)
train <- sample(1:n, round(n*2/3))
test <- (1:n) [-train]
```

1)

a)

We train a LS model on the training whole dataset

```
model1 <- lm(y~., data, subset=train)
#summary (model1)
```

for calculating trimmed MSE we define following functions

```
rtmean <- function(x,trim = 0) {
  x <- sort(x)
  v <- x[1:floor(length(x)*(1-trim))]
  mean(v)
}

mse <- function(y.true,y.pred, trim=0){
  return(rtmean((y.true - y.pred)^2, trim =trim))
}
```

we calculate MSE on the test dataset

```
mse(data[test, 'y'], predict(model1, data[test,]))
```

```
## Warning in predict.lm(model1, data[test, ]): prediction from a rank-
## deficient fit may be misleading
```

```
## [1] 8.781102
```

Then the 10% trimmed MSE on the test dataset

```
mse(data[test, 'y'], predict(model1, data[test,]), trim=0.1)
```

```
## Warning in predict.lm(model1, data[test, ]): prediction from a rank-
## deficient fit may be misleading
```

```
## [1] 0.6878118
```

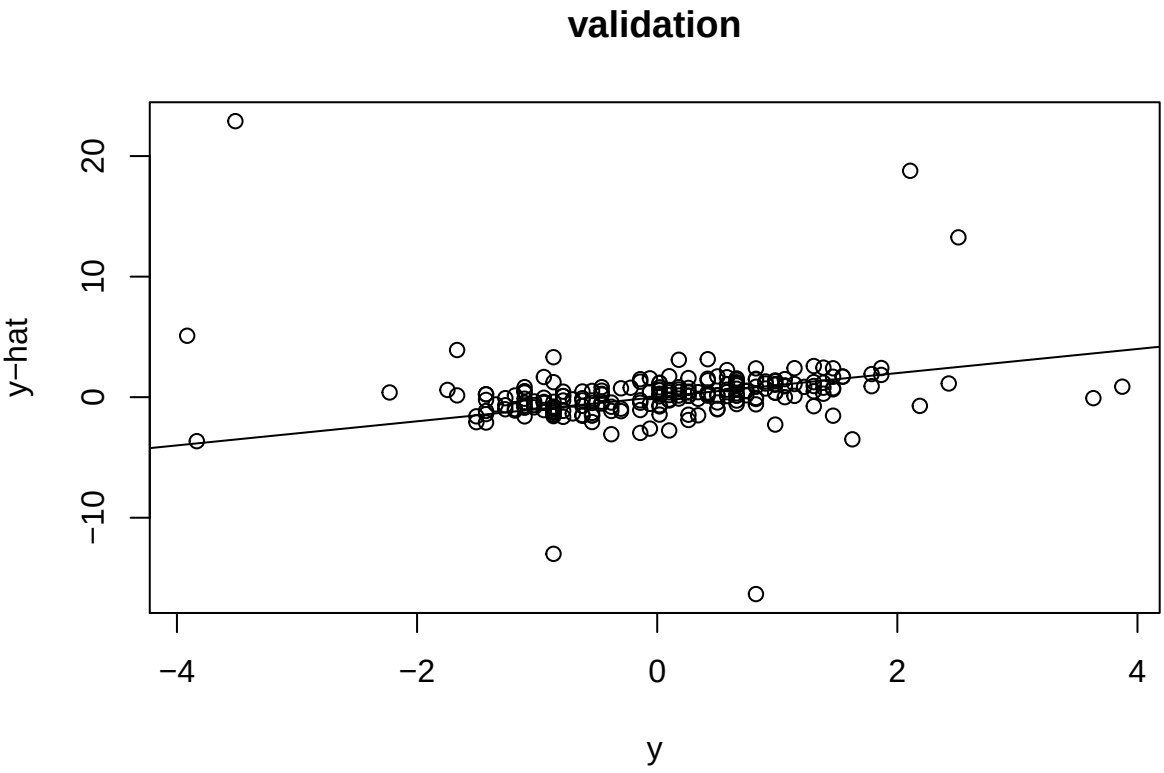
The trimmed MSE is - as expected - much better. R warns about the model fit beeing rank-deficient because we do not have full rank in our input matrix

We plot the predicted values agaist the actual y values

```
plot(data[test, 'y'], predict(model1, data[test,]), xlab='y' ,ylab='y-hat', main="validation")
```

```
## Warning in predict.lm(model1, data[test, ]): prediction from a rank-
## deficient fit may be misleading
```

```
abline(c(0,1))
```



We see there are a few outliers with bad predictions

2)

a)

We train pcr regression models with up to 100 components with the train dataset

```
library(pls)
```

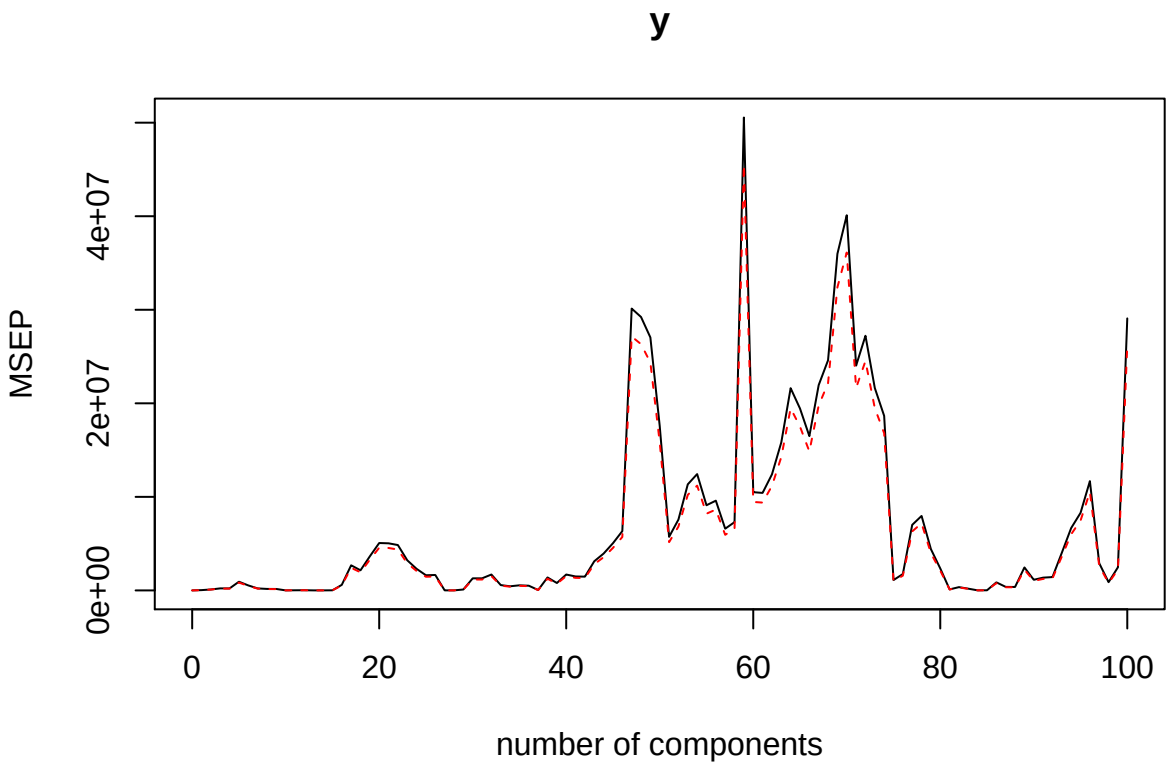
```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```
ncomp <- 100
model2 <- pcr(y~., ncomp=ncomp, data=data, subset=train, scale=TRUE, validation="CV", segments=10,  segm
```

b)

```
plot(model2, plottype="validation", val.type="MSEP")
```

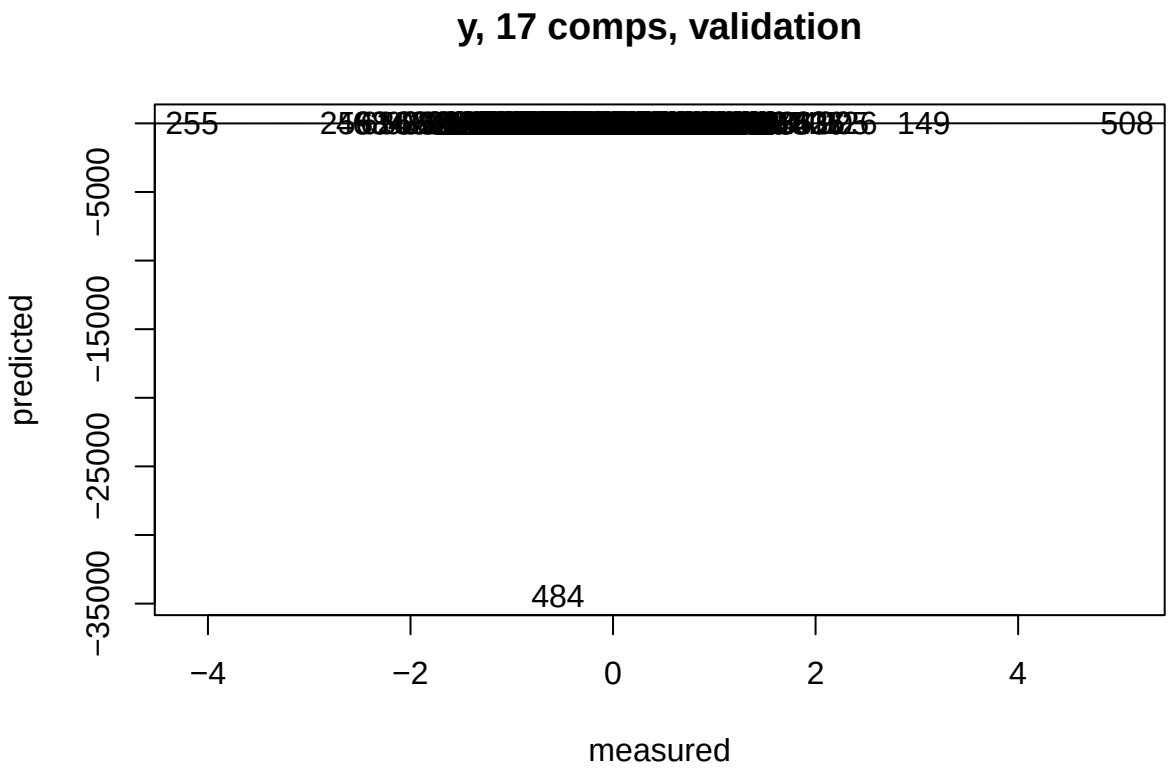


The plot does not look like expected, the MSEP increases and reaches very high numbers. This is probably because of outliers which disturb the fitting process. We cannot use this plot to select the optimal number of components.

c)

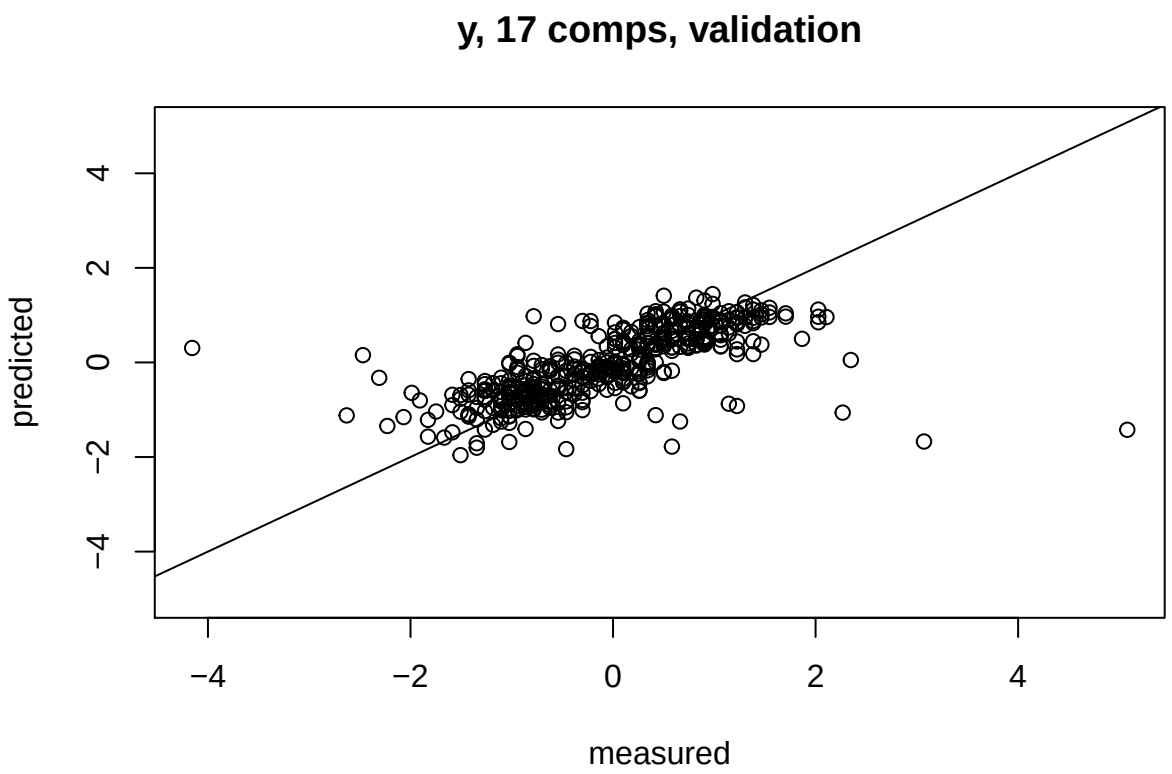
Now we look on the predictions on the train data

```
selcomp <- 17
predplot(model2, line=TRUE, ncomp=selcomp, labels=rownames(data[train,]))
```



There is a very big prediction error for one datapoint. To better see the result we replot restricting the y axix

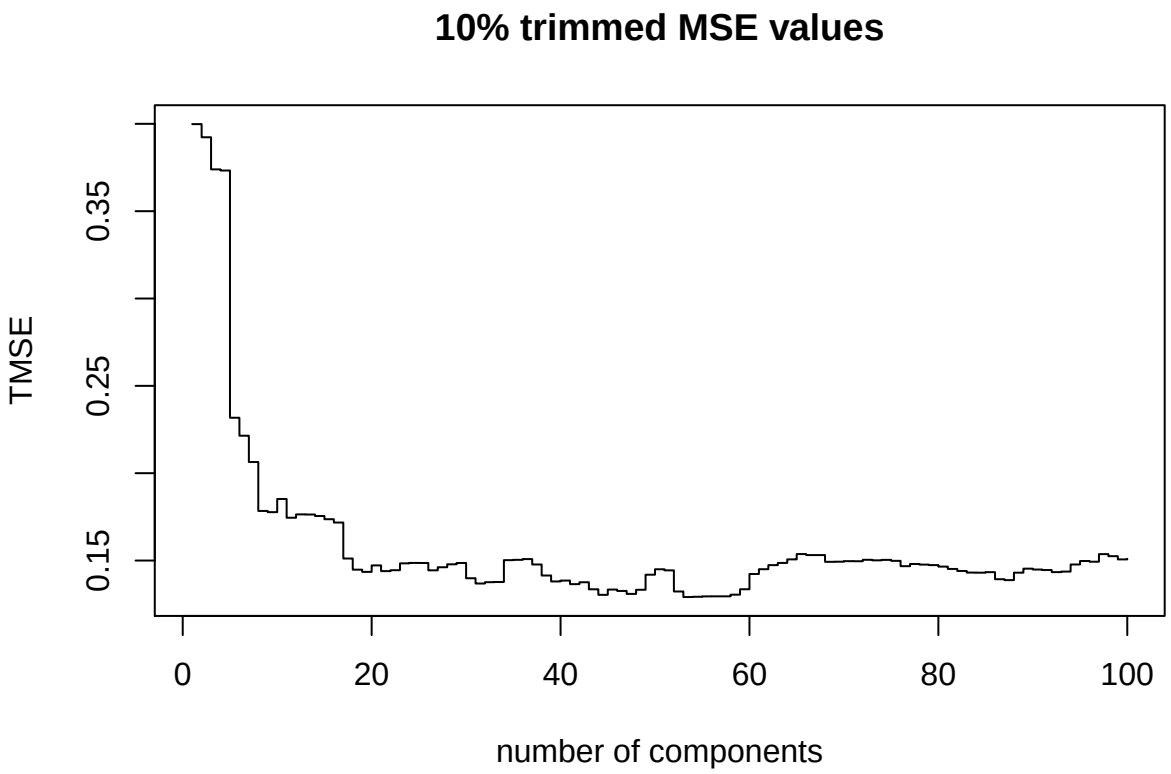
```
plot(model2, ncomp=selcomp, line=TRUE, ylim=c(-5,5))
```



d)

To better deal with outliers we recreate the plot from 2b using 10 % trimmed MSE. We use the test dataset for the MSE calculations

```
res <- vector(length = ncomp)
for (i in 1:ncomp) {
  res[i] <- mse(data[test,'y'], predict(model2, data[test,], ncomp=i), trim=0.1)
}
plot(res, type='s', xlab="number of components", ylab="TMSE", main="10% trimmed MSE values")
```



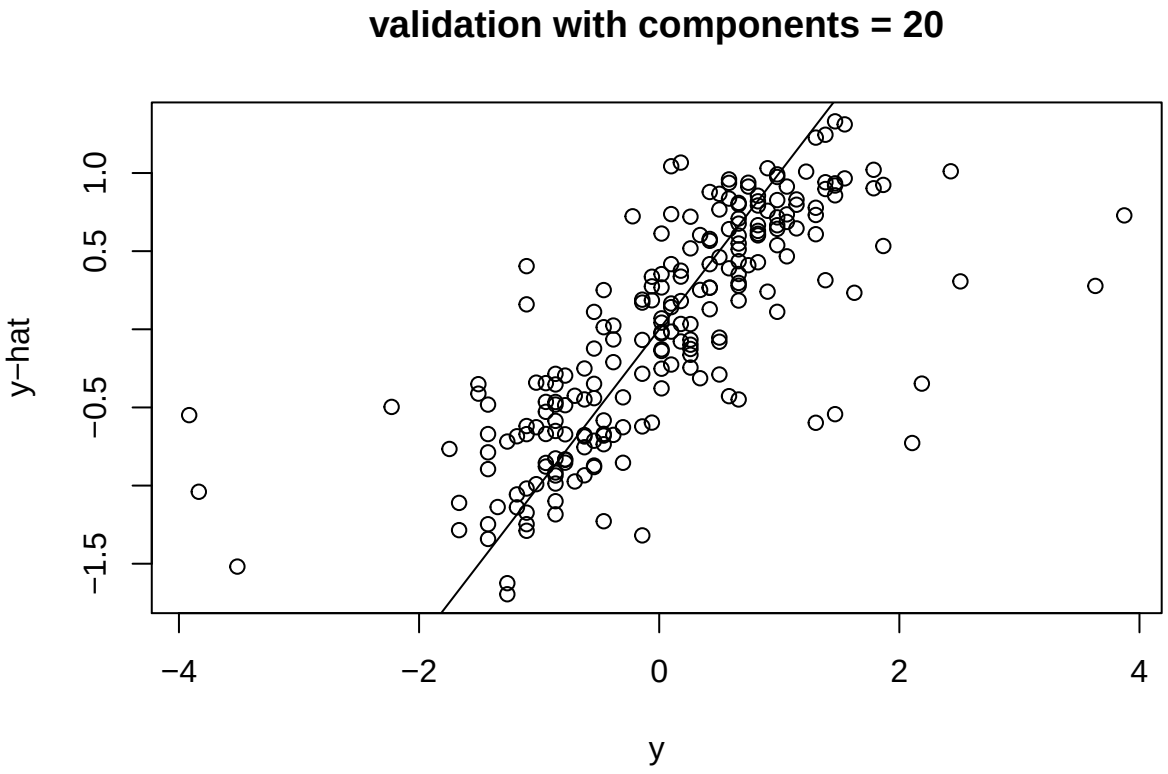
Here we see a sharp drop on MSE to around 20 components. After that the MSE only improves slightly. We choose 20 as optimal number of compoents and calculate the trimmed MSE again

```
selcomp <- 20
mse(data[test, 'y'], predict(model2, newdata=data[test,], ncomp=selcomp), trim=0.1)
```

```
## [1] 0.1472177
```

We plot the predictions

```
plot(data[test, 'y'], predict(model2, data[test,], ncomp=selcomp), xlab='y' ,ylab='y-hat', main=paste0(
abline(c(0,1))
```



3)

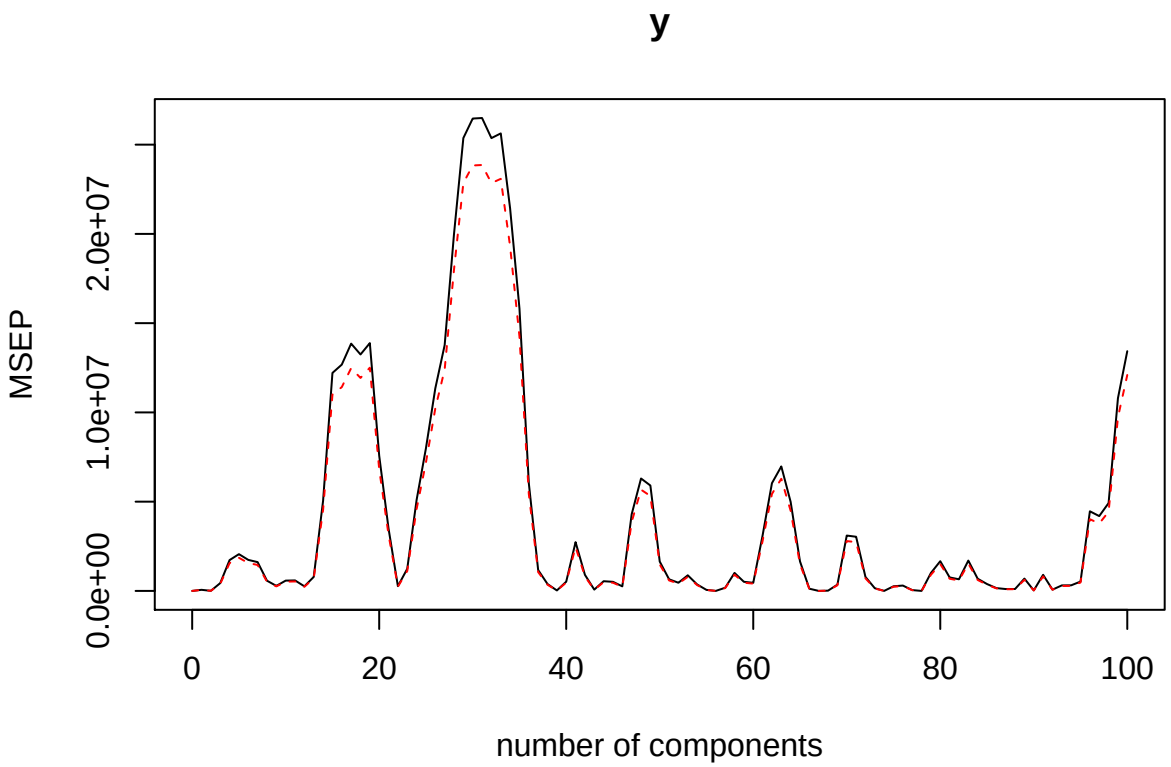
a)

We use the same argumenst then in 2) to train a Partial Least Squares model with up to 100 components

```
ncomp <- 100
model3 <- plsr(y~., ncomp=ncomp, data=data, subset=train, scale=TRUE, validation="CV", segments=10,  seg
```

b)

```
plot(model3, plottype="validation", val.type="MSEP")
```

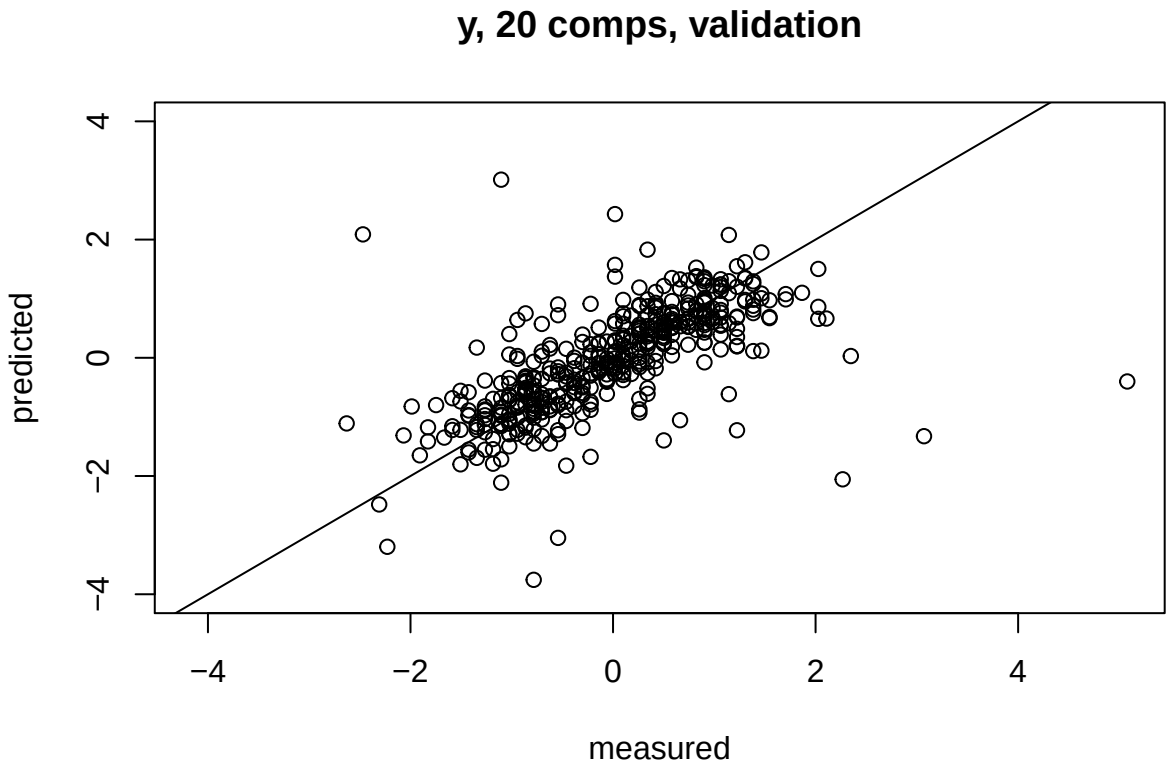


Again we see a simular plot then in 2) which is not usable to select the optimal number of components.

c)

We plot the prediction with 20 compoents, to get an interpretable plot we limit the y axis

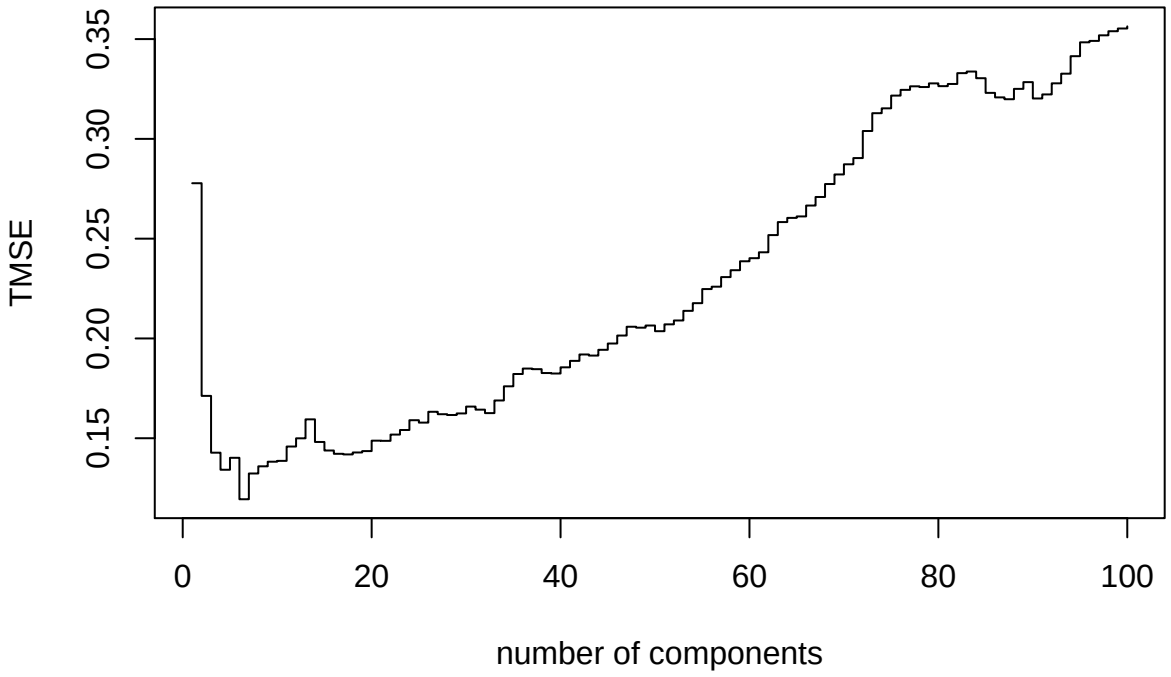
```
predplot(model3, line=TRUE, ncomp=20, ylim=c(-4, 4))
```



d)

We recalculate 10% timmed mse-s with the test dataset for up to 100 components

```
res <- vector(length = ncomp)
for (i in 1:ncomp) {
  res[i] <- mse(data[test,'y'], predict(model3, data[test,], ncomp=i), trim=0.1)
}
plot(res, type='s', xlab="number of components", ylab="TMSE", main="10% trimmed MSE values")
```



We see that there is a clear choise for the best number of components

```
selcomp <- which.min(res)
selcomp
```

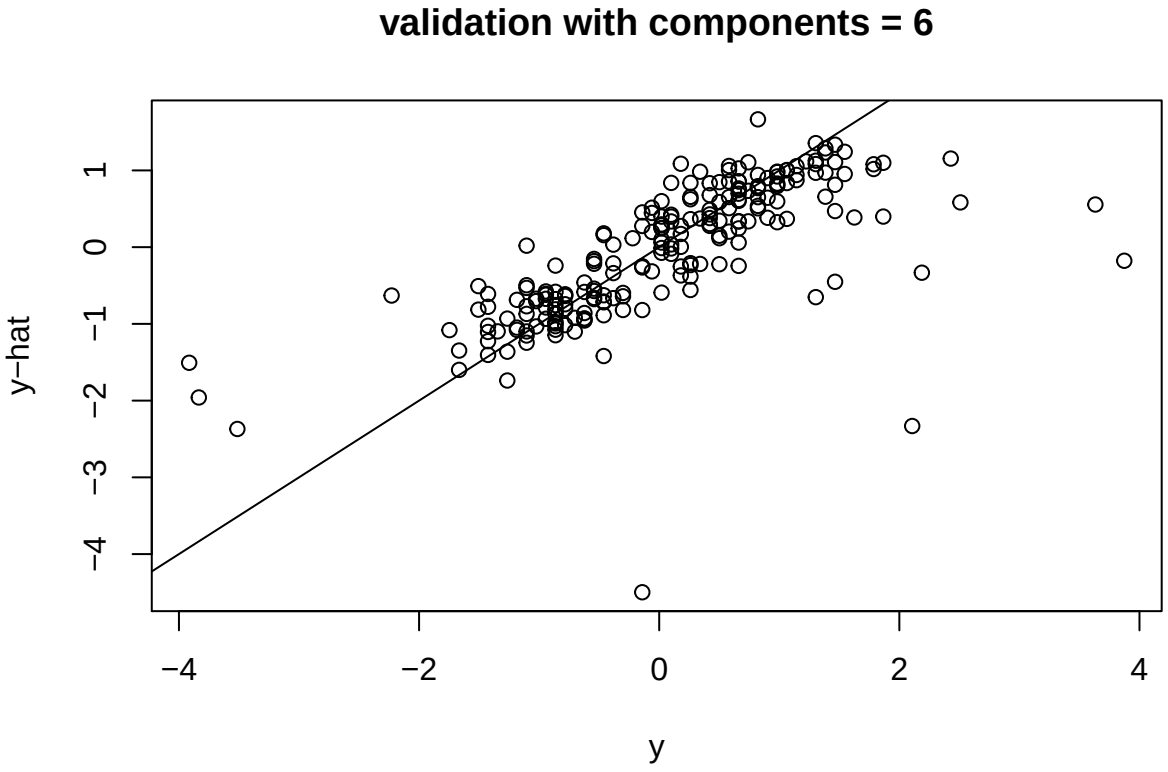
```
## [1] 6
```

with trimmed MSE

```
min(res)
```

```
## [1] 0.119433
```

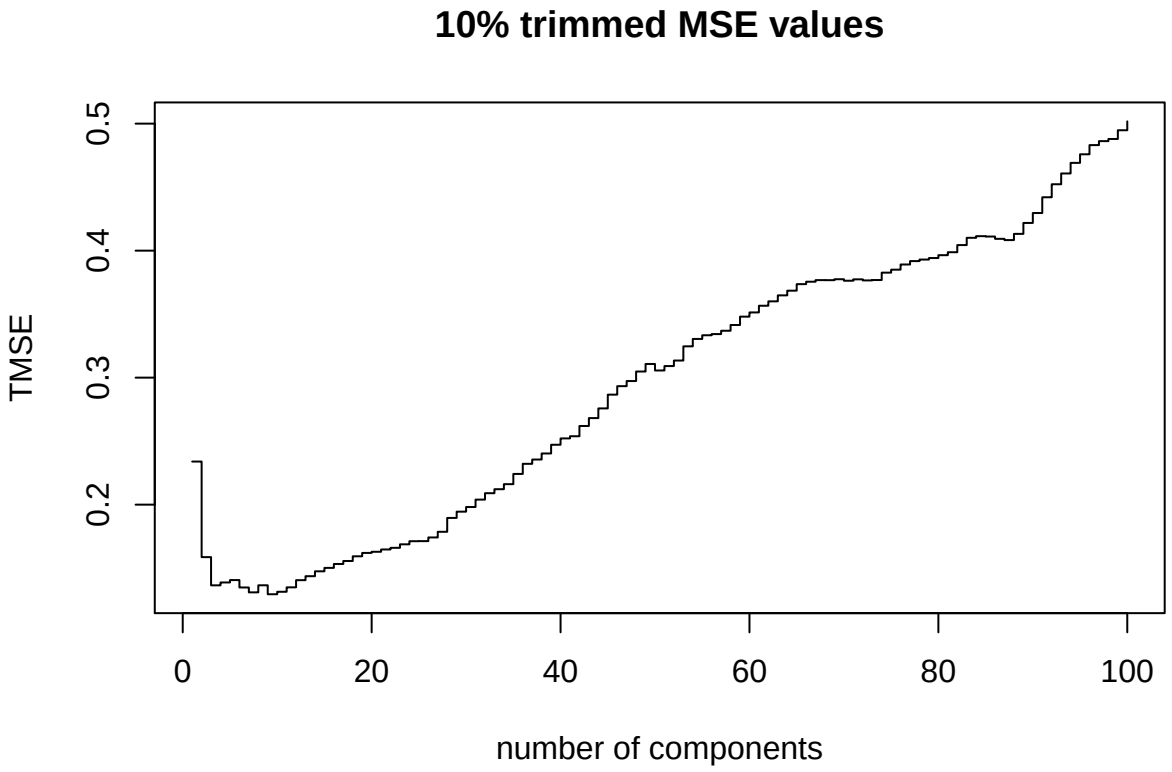
```
plot(data[test, 'y'], predict(model3, data[test,], ncomp=selcomp), xlab='y' ,ylab='y-hat', main=paste0(
abline(c(0,1))
```



e)

We replot the plot from 3b) using 10 % trimmed MSE

```
res <- vector(length = ncomp)
for (i in 1:ncomp) {
  res[i] <- mse(data[train,'y'], model3$validation$pred[,1,i], trim=0.1)
}
plot(res, type='s', xlab="number of components", ylab="TMSE", main="10% trimmed MSE values")
```



The MSE dropps sharply till 3 and rises again after about 10 components

the optimal nr of components seems to be around 3:

with the 10% trimmed MSE of

```
res[3]
```

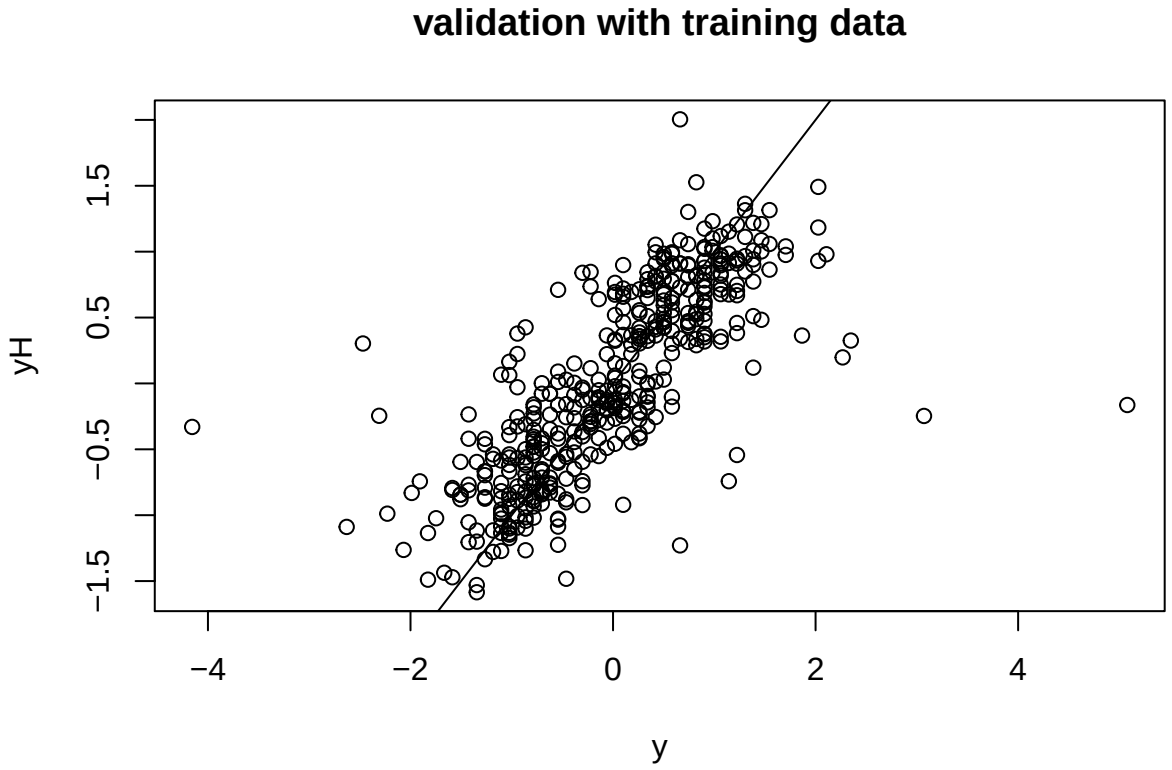
```
## [1] 0.1364572
```

The result with 3 components is slightly worse than with 6 components selected in 3d)

4)

We select 20 components to creat ore pcr model. We scale the input data, then calculate the regression coeficients using the scores from princomp. Then we plot our predictions

```
selcomp = 20
data.s <- scale(data)
X <- data.s[train, 2:393]
pc <- princomp(X)
Z <- pc$scores[,1:selcomp]
y <- data.s[train, 'y']
betaH <- solve(t(Z)%*%Z)%*%crossprod(Z,y)
yH <- Z %*% betaH
plot(y, yH, main="validation with training data")
abline(c(0,1))
```

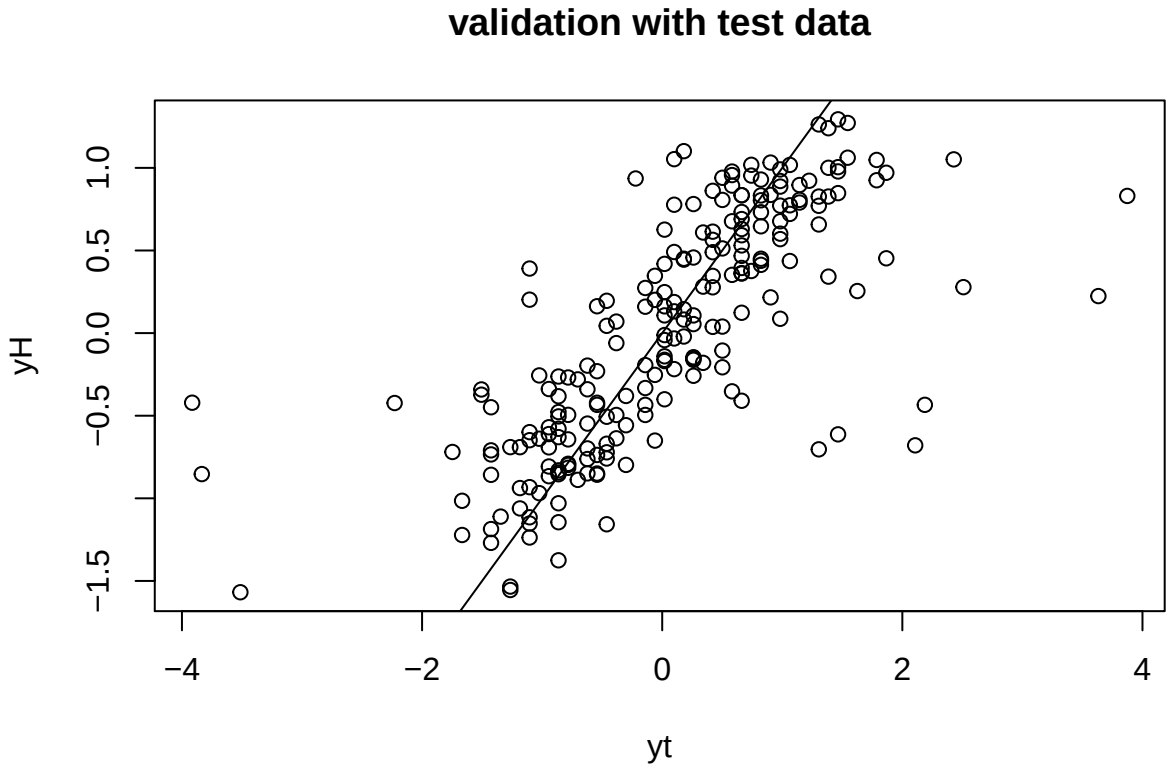



```
mse(y, yH, trim=0.1)
```

```
## [1] 0.1256515
```

We repeat with the test dataset

```
Xt <- data.s[test, 2:393]
yt <- data.s[test, 'y']
Zt <- Xt %*% pc$loadings[,1:selcomp]
yH <- Zt %*% betaH
plot(yt, yH, main="validation with test data")
abline(c(0,1))
```




```
mse(yt, yH, trim=0.1)
```

```
## [1] 0.1427142
```

The MSE is very simular to the MSE from 2d)