Advanced Methods for Regression and Classification

Exercise 4

Anonymous

11/3/2020

Initial setup. Load the data, set seed, build train and test split indexes.

```r
set.seed(11941109)
load("../dat.RData")

# split to train and test
# all sample indexes
indexes <- c(1:nrow(d))
# train sample count should be 2/3s of the total sample count
train_sample_count <- as.integer(nrow(d)*2/3)
# calculate the train indexes
train_indexes <- sample(indexes, train_sample_count)
# extract the test indexes as the train complement
test_indexes <- setdiff(indexes, train_indexes)
```

Create mse function that trims only the largest 10%.

```r
mse <- function(prediction, ground_truth, trim =TRUE) {
  squared_residuals <- 0
  for(i in 1:length(prediction)) {
    delta_squared <- (ground_truth[i] - prediction[i])^2
    squared_residuals <- c(squared_residuals, delta_squared)
  }
  if (trim) {
    # remove top 10%
    squared_residuals <- squared_residuals[squared_residuals < quantile(squared_residuals, 0.9)]
  }

  # return mean
  result <- mean(squared_residuals)
  return(result)
}
```

Task 1) Full model

a)

We will create the full model using only the train data for fitting.

```r
full_model <- lm(y~., d, subset = train_indexes)
#summary(full_model)
```

There are quite a few linearly dependent variables such as X60, X66, X97, X98, X104, X194 and possibly others. Looking at the model fit (and comparing it to my results from the previous exercises) I see that the

R-squared and the adjusted R-squared are rather high indicating a good fit - we should check this on the test data to ensure the model is not overfitting due to high number of independent variables.

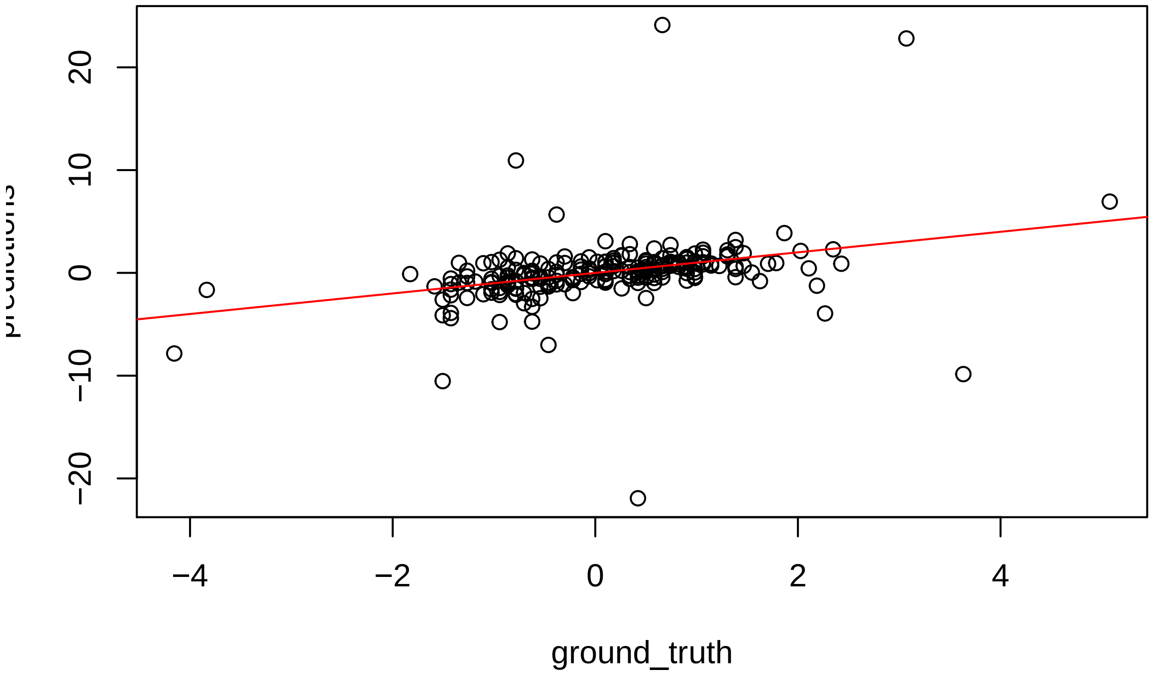
b) Predict with full model

Now I will use the fitted model to predict the test data. I will also add a red line with y=x where the points should lie if the predictions match the ground truth. The worse the prediction the further away from this line the loint will lie.

```
predictions <- predict(full_model, d[test_indexes,])
```

```
## Warning in predict.lm(full_model, d[test_indexes, ]): prediction from a
## rank-deficient fit may be misleading
```

```
ground_truth <- d[test_indexes,]$y
plot(ground_truth, predictions)
abline(0,1, col="red")
```



We see that the data is predicted somewhat well but there are huge outliers which can be seen by the scale change and the distance of the points from the red line.

Let's take a look at the MSE and trimmed MSE results next. My expectation would be that the trimmed MSE will bu much lower as the visible outliers are less than 10% (22 points as we have a test set of ~220 observations).

```
# not trimmed
non_trimmed <- mse(predictions, ground_truth, trim = FALSE)
non_trimmed
```

```
## [1] 10.1483
```

```
# trimmed
trimmed <- mse(predictions, ground_truth)
trimmed
```

```
## [1] 0.8819244
```

```
paste("nontrimmed:trimmed ratio", non_trimmed/trimmed)
```

```
## [1] "nontrimmed:trimmed ratio 11.5069939752494"
```

The results are in line with the expectations as the non-trimmed MSE is about 11 times higher than the trimmed one. I'd conclude that the trimming removed most if not all of the outliers as the 10% highest squared residuals (points furthest away from the x=y line) were ignored in the MSE calculation.

Task 2) Principal component regression (PCR)

a + b)

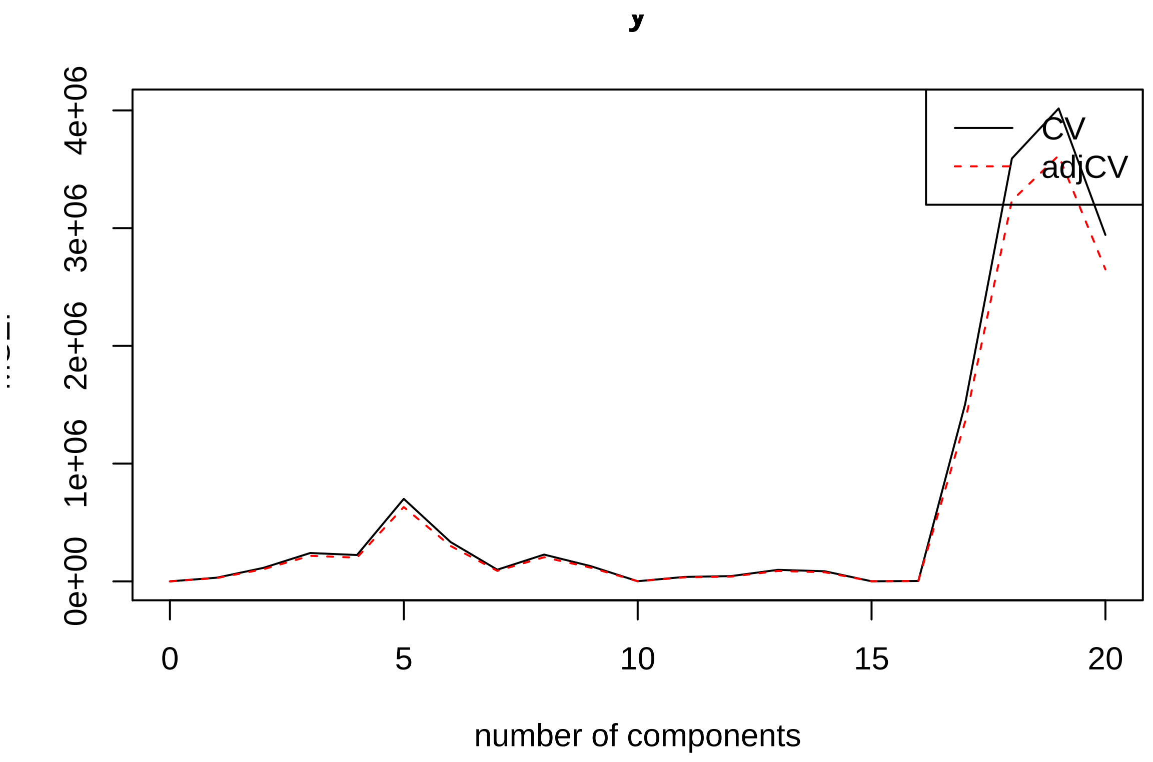
I used the pcr function as suggested for training data only with scaled variances to 1 whilst utilizing cross-validation with 10 folds and observed the prediction error for up to 20 components.

```
pcr_model <- pcr(y~., data = d, subset = train_indexes, scale = TRUE,
                 validation = "CV", segments = 10, ncomp = 20, segment.type="random")
summary(pcr_model)
```

```
## Data:    X dimension: 442 392
##  Y dimension: 442 1
## Fit method: svdpc
## Number of components considered: 20
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.9552      176    338.3    490.7    472.9    837.0    577.9
## adjCV       0.9552      167    321.0    465.6    448.7    794.3    548.4
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       314.4    476.9    358.8     31.52     193.5     211.7     312.8
## adjCV    298.4    452.5    340.4     29.91     183.6     200.9     296.9
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        292.9     24.09     53.56      1226      1895      2004
## adjCV     278.0     22.86     50.83      1164      1798      1902
##        20 comps
## CV         1715
## adjCV      1628
##
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X    21.52    32.91    41.84    48.33    53.86    57.88    61.34    64.33
## y    21.45    23.17    27.34    27.45    43.77    48.44    49.65    52.82
##    9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X    66.76     69.03     71.02     72.87     74.51     75.92     77.17
## y    58.43     59.16     60.09     60.12     60.17     60.18     61.00
##    16 comps  17 comps  18 comps  19 comps  20 comps
## X     78.38     79.45     80.37     81.24     82.09
## y     61.00     64.26     65.95     66.45     66.46
```

From the summary we can see that the minimum prediction error from CV is achieved with 10 components. . Let's visualize this in a plot as well. Note that we use MSEP - mean squared prediction error in this case not the RMSEP (root of MSEP) that was used in the pcr function itself. This just means that we get squares of the CV and adjCV values for the components but other than that this plot is just a visualization of the results from the pcr.

```
plot(pcr_model, plottype="validation", val.type = "MSEP", legend = "topright")
```



Just as we suspected the plot also suggests that the minimum error is achieved at 10 components. To get the resulting MSE we need to extract the cross-validated predictions for 10 components and just compare them with the ground truth.

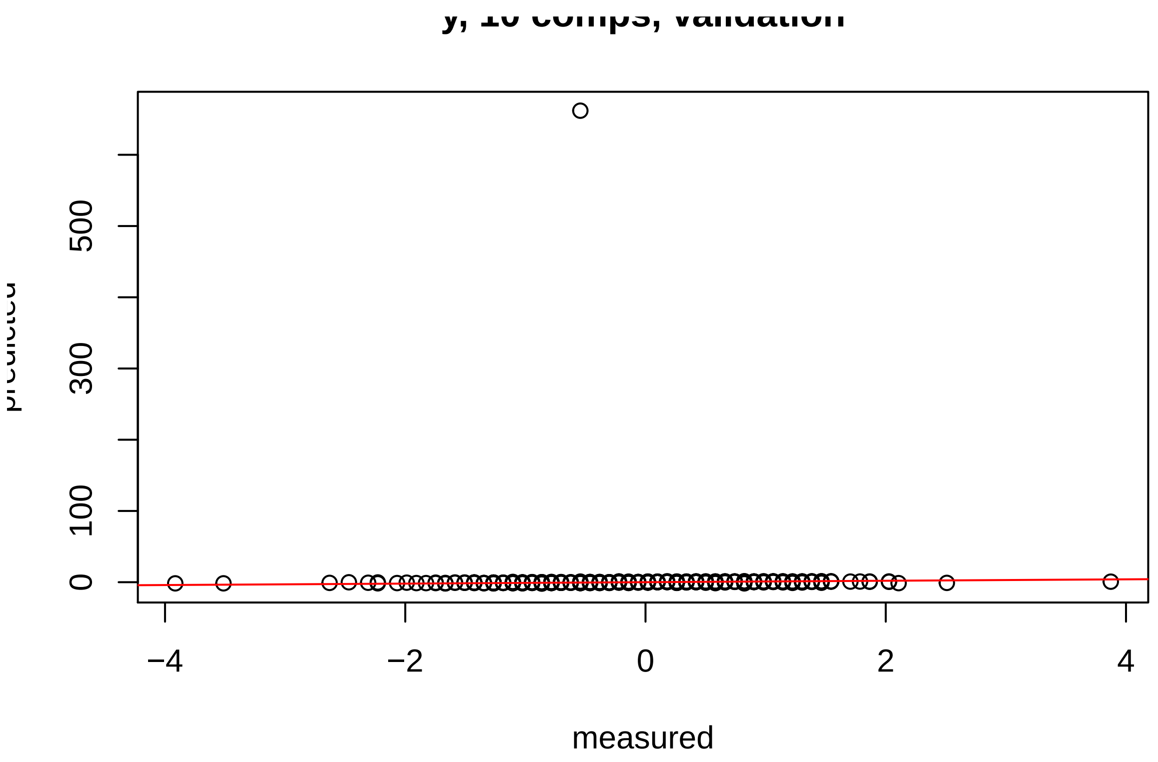
```
pcr_10_mse <- mse(pcr_model$validation$pred[,1,10], d$y[train_indexes], trim = FALSE)
pcr_10_mse
```

```
## [1] 991.0846
```

As we can see the error is much higher than with the full model. An alternate (less precise) way of obtaining the MSE is to just square the CV RMSE from the summary => $74.91^2 = 5611.508$. Now why did this happen? Let's take a look in the plot in the next subtask.

c)

```
predplot(pcr_model, ncomp = 10)
abline(0,1, col="red")
```



As we can see the plot doesn't tell us much because we have a massive outlier. Now we know why the MSE was so high. As we have only 442 observations in the train 1 outlier covers almost the entire MSE we got.

As we have a very obvious outlier let's try and remedy it with the trimmed mse (so that we can thn compare with d as well how well our train and test compare ~ how well the model generalizes).

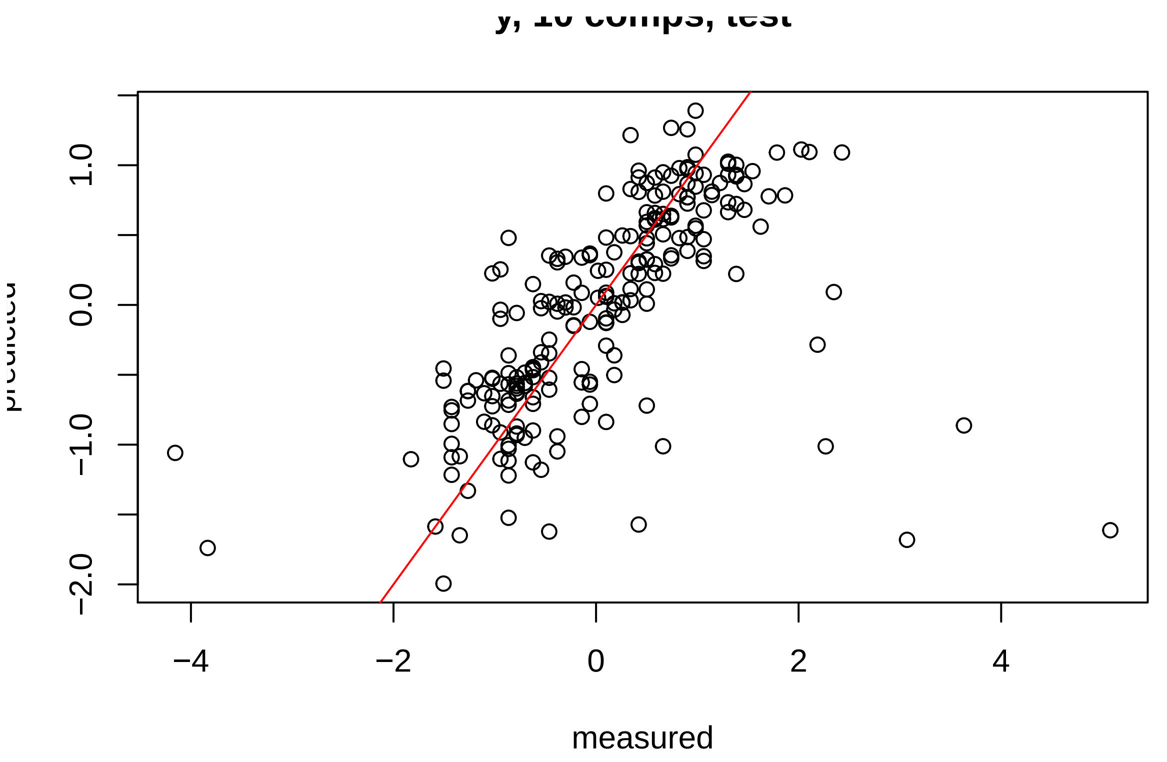
```
trimmed_pcr_train <- mse(pcr_model$validation$pred[,1,10], d$y[train_indexes])
trimmed_pcr_train
```

```
## [1] 0.1639827
```

As can be seen this estimate us much lower than without trimming further proving our point that the outlier is to blame for the bad train performance. This value vastly outperforms the full model (0.159 vs 0.9 for the full model) BUT this is still on training set only (even though it was taken as test set for our models in the appropriate CV folds). This means if we want to have a fair comparison we need to calculate the trimmed mse on the test set as well. Let's take a look at that in d)

d)

```
predplot(pcr_model, ncomp = 10, newdata = d[test_indexes,])
abline(0,1, col="red")
```



As we can see the plot looks much better than the training one. This indicates that the model actually fits the data somewhat ok even though we can still see obvious outliers for positive measured values higher than 2 our model always underpredicts. Let's take a look at the trimmed MSE.

```
test_predictions <- predict(pcr_model, newdata = d[test_indexes, ], ncomp = 10)
trimmed_pcr <-  mse(test_predictions, d$y[test_indexes])
trimmed_pcr
```

```
## [1] 0.1605652
```

As we can see the trimmed mse is a lot more reasonable. This looks comparable to the full model. Let's take a look:

```
trimmed_pcr/trimmed
```

```
## [1] 0.1820624
```

As we can see the pcr model has only 20.72% of the full model error measure when using trimmed MSE. Furthermore we can take a look how close the train (well validation) and test error are for the pcr model - the train trimmed MSE was about 0.159 and our test is 0.187. This is about 17.6% increase which is not bad at all and suggests the model generalizes rather well.

Task 3) Partial least squares

a + b)

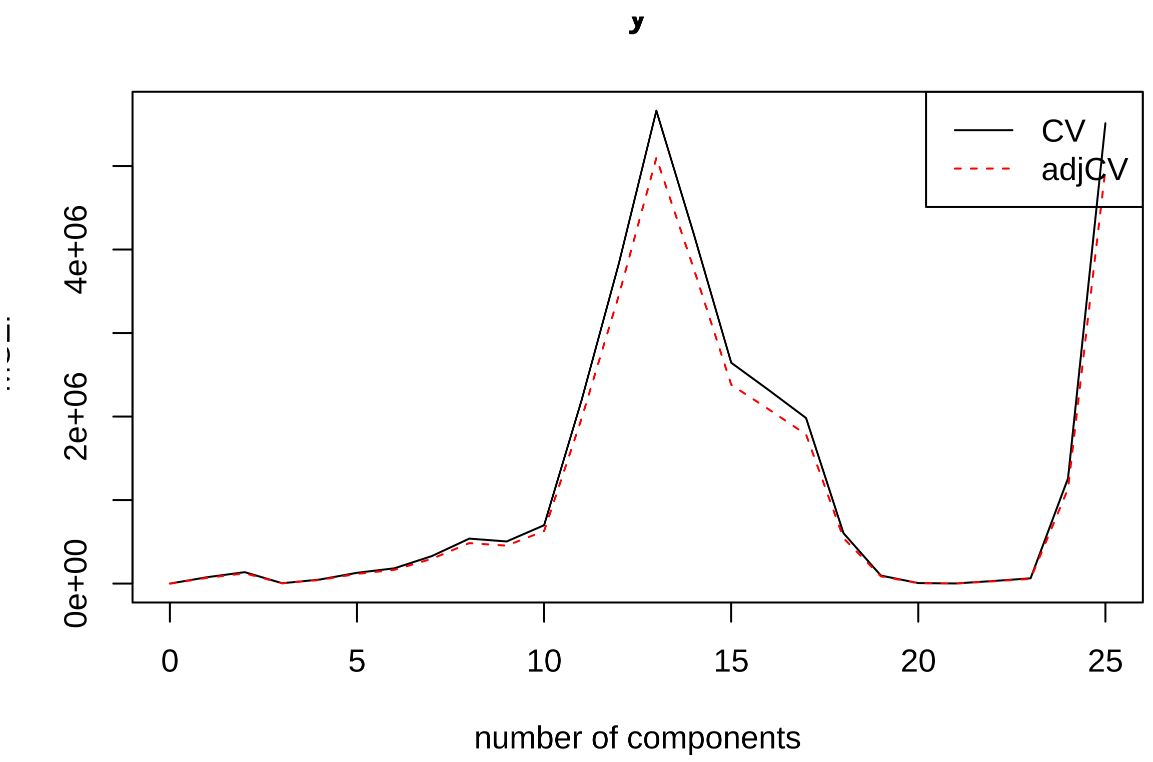
In this task we do the same thing as we did in 2 but we have a little different goal in mind. In pcr we were choosing the orthogonal components maximizing the variance but that does not necessarily have to better explain y. In plsr we take y into regard and try to optimize for that.

```r
plsr_model <- plsr(y~., data = d, subset = train_indexes, scale = TRUE,
                  validation = "CV", segments = 10, ncomp = 25, segment.typr="random")
summary(plsr_model)
```

```
## Data:    X dimension: 442 392
##  Y dimension: 442 1
## Fit method: kernelpls
## Number of components considered: 25
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.9552    276.8    369.4    64.55    218.4    358.8    427.5
## adjCV       0.9552    262.7    350.6    61.25    207.2    340.5    405.7
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       573.7    733.7    710.3     835.9      1481      1959      2380
## adjCV    544.4    696.2    674.0     793.2      1405      1859      2258
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV         2046      1626      1522      1408     777.0     309.3
## adjCV      1941      1543      1445      1336     737.3     293.5
##        20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## CV        76.67     37.46     174.5     251.0      1123      2348
## adjCV     72.75     35.55     165.6     238.2      1065      2228
##
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X    19.95    27.69    35.23    40.88    47.72    51.22    54.09    56.85
## y    40.28    61.29    67.32    71.89    74.49    77.49    79.20    80.80
##    9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X    61.35     64.06     65.54     67.66     69.49     70.94     71.91
## y    81.54     82.21     83.20     83.74     84.26     84.74     85.26
##    16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
## X     73.29     74.49     75.50     76.52     77.34     78.40     79.46
## y     85.64     86.08     86.53     86.95     87.38     87.68     87.94
##    23 comps  24 comps  25 comps
## X     80.04     81.01     81.66
## y     88.30     88.52     88.81
```

As can be seen from the summary the model with 5 components has the lowest CV mse. What we can see here however is the discrepancy of the % variance explained between x and y - we explain ~46.5% of the variance of x but ~76% of the variance of y which is higher than what we had in case of pcr. Actually there we were explaining higher portion of the x variance compared to the y variance. Let's validate our observation with a plot.

```r
plot(plsr_model, plottype="validation", val.type = "MSEP", legend = "topright")
```



It sure does look like the sweetspot is the 5 components. We see models with higher component counts getting somwhat close (18 for example) but 5 is the clear winner with the best CV pred error as well as the low component count on top.

```
plsr_5_mse <- mse(plsr_model$validation$pred[,1,5], d$y[train_indexes], trim = FALSE)
plsr_5_mse
```

```
## [1] 128479.3
```

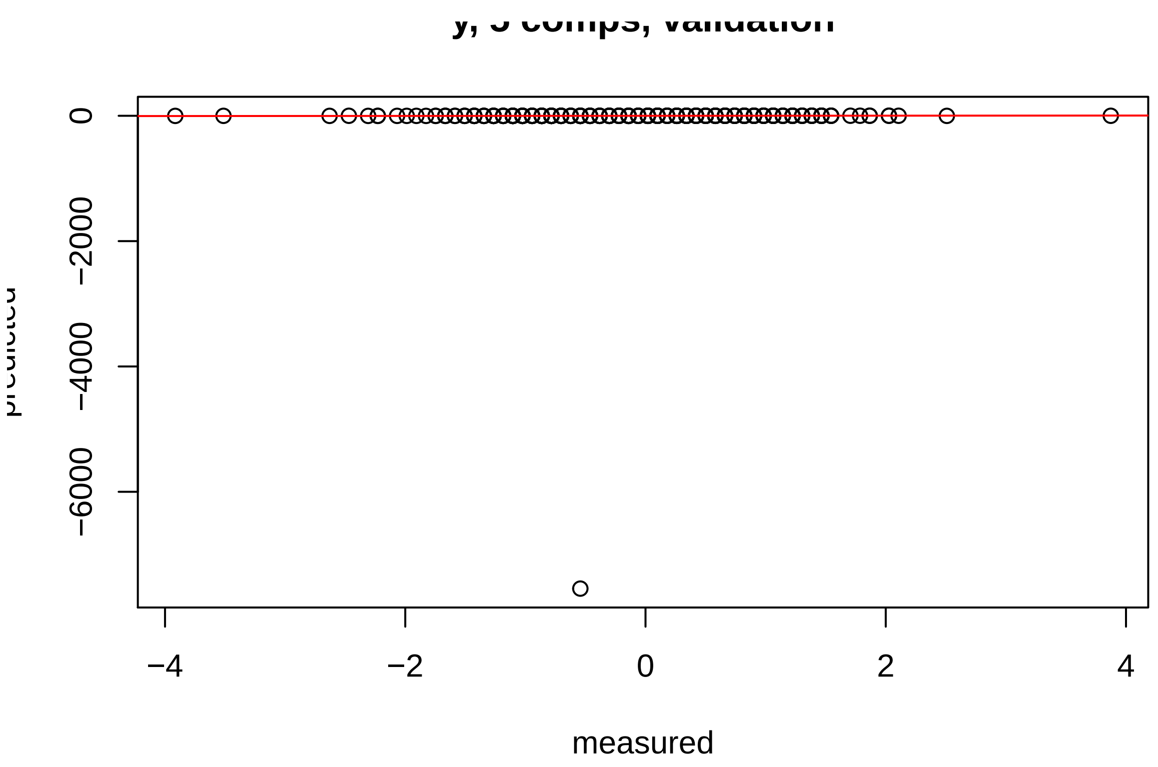
```
paste("pcr:plsr mse ratio", pcr_10_mse/plsr_5_mse)
```

```
## [1] "pcr:plsr mse ratio 0.00771396597494312"
```

As we can see the MSE is much lower than in case of the pcr. I believe this is due to use trying to explain the variability of y not that of x but let's confirm with a plot.

c)

```
predplot(plsr_model, ncomp = 5)
abline(0,1, col="red")
```



Just like in case of pcr the plot has an outlier but this time the value is much smaller (about 10 times!). Because we are paying more attention to y and thus allow our model to be more affected by the outlier resulting in better prediction for that 1 observation.

Let's calculate the trimmed mse and compare them with pcr (as we did that in 2) as well).

```
trimmed_plsr_train <- mse(plsr_model$validation$pred[,1,5], d$y[train_indexes])
trimmed_plsr_train
```

```
## [1] 0.1436389
```

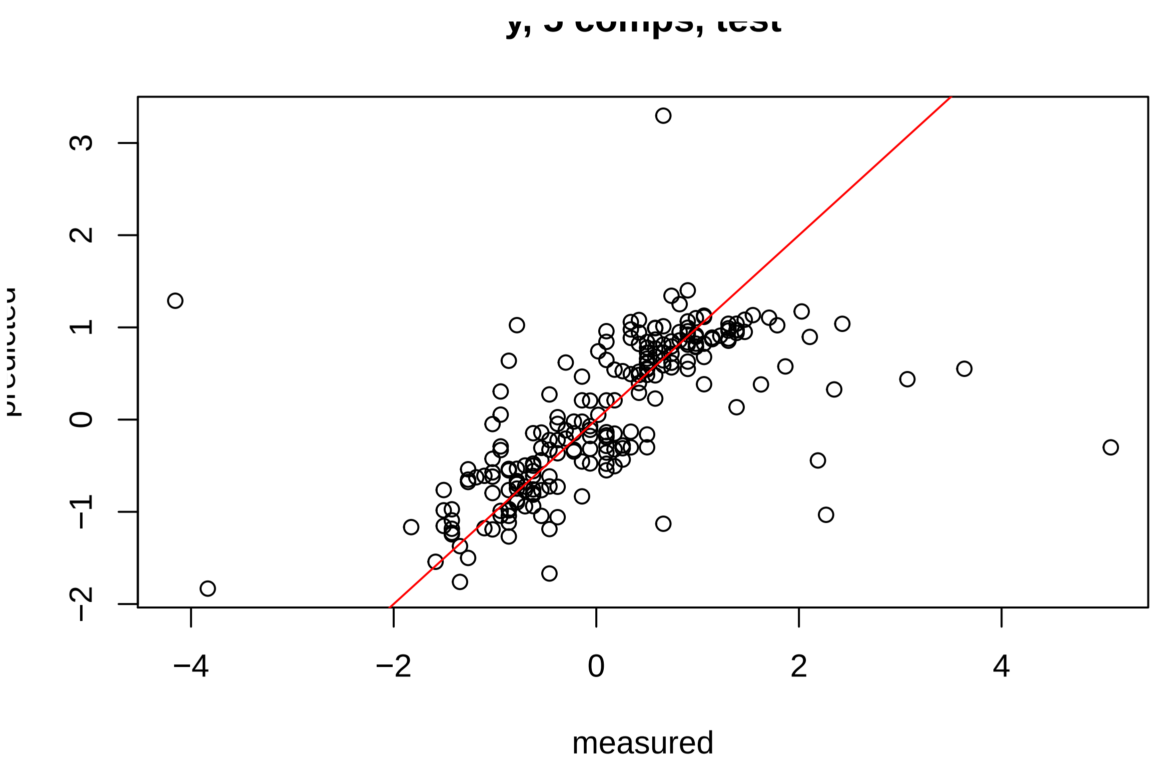
```
paste("pcr:plsr mse ratio", trimmed_pcr_train/trimmed_plsr_train)
```

```
## [1] "pcr:plsr mse ratio 1.14163158024705"
```

We can see that even the trimmed MSE is better than the one from pcr though the difference is much smaller as expeced since the removed outlier by trimming had much higher impact in pcr.

d)

```
predplot(plsr_model, ncomp = 5, newdata = d[test_indexes,])
abline(0,1, col="red")
```



As we can see data is more tightly packed towards the red line than in pcr meaning we achieve lower variability in y. However, the same problem just like we had in pcr occurs for y values over 2 - all of them are underpredicted by our model.

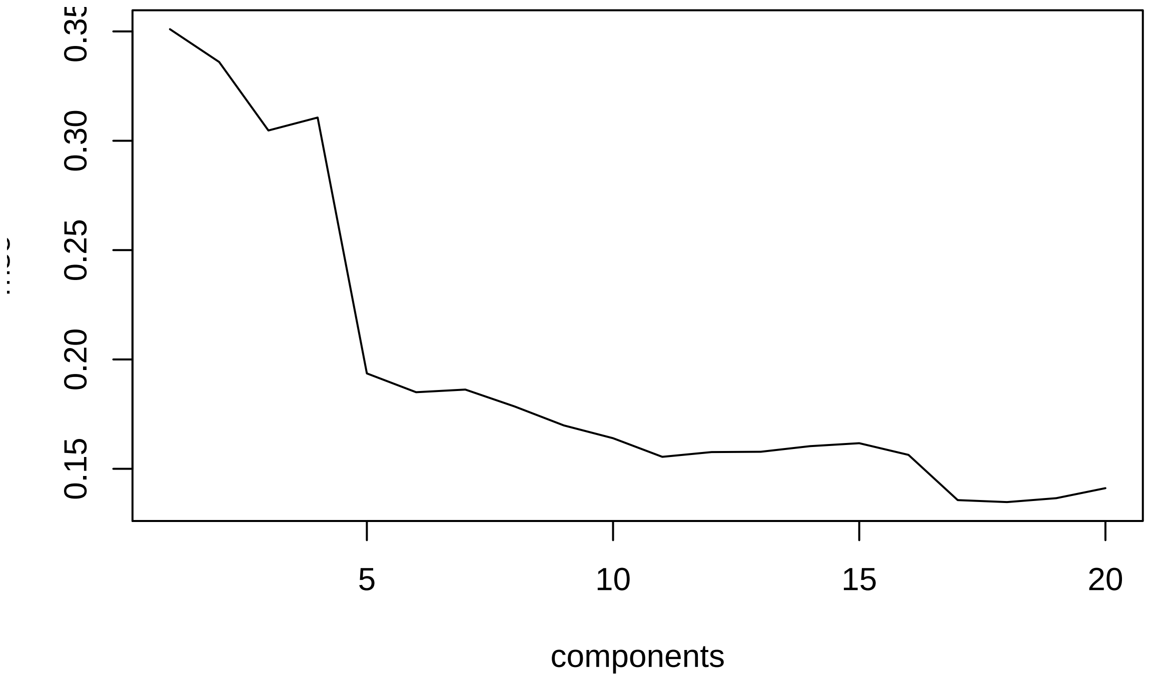
e)

As the outliers might have affected our decision making let's try and reconstruct the mse plot for different numbers of components with trimmed mse not the actual mse.

- **PCR**

```
trimmed_mses_for_all_components <- function(model, component_count) {
  mses <- c()
  for(i in 1:component_count) {
    current_mse <- mse(model$validation$pred[,1,i], d$y[train_indexes])
    mses <- c(mses, current_mse)
  }
  return(mses)
}

components <- 1:20
pcr_mses <- trimmed_mses_for_all_components(pcr_model, 20)
plot(components, pcr_mses, type="l", xlab="components", ylab="mse")
```



As we can see from the plot this was deffinitely the case. Our component selection was affected as our chosen components (10) are nowhere near the lowest trimmed mse. Let's find the component count with the min trimmed mse.

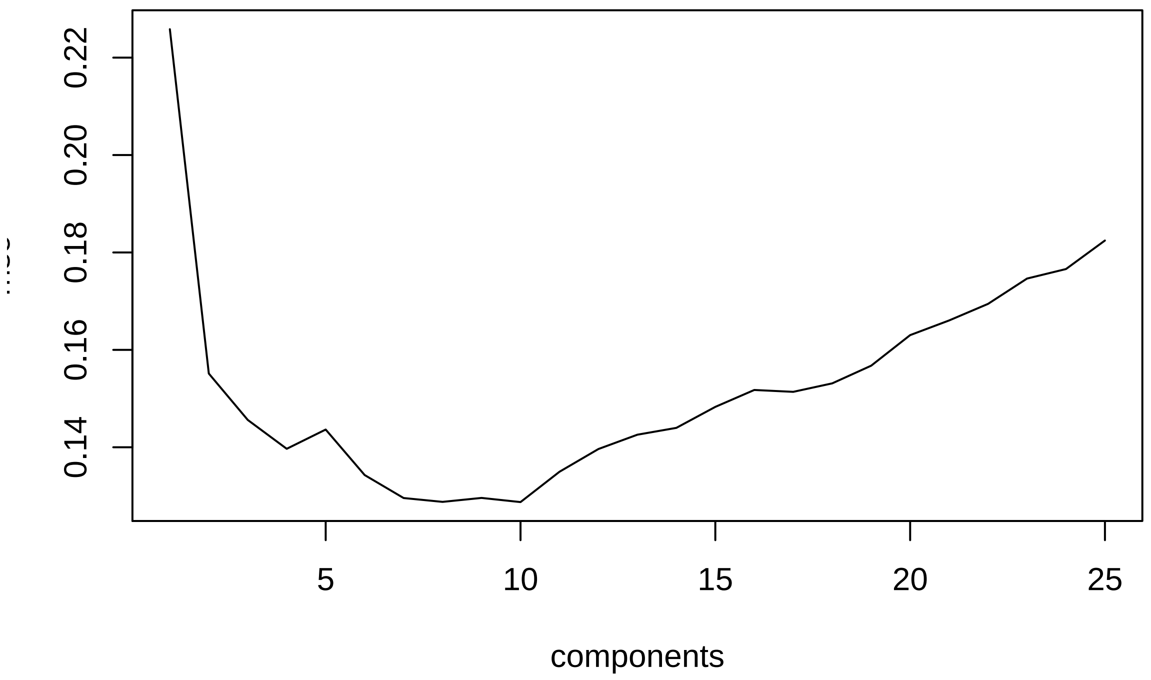
```
order(pcr_mses)[1]
```

```
## [1] 18
```

The best number of components would be 19 but If I was to use this approach I would increase the number of components observed to ensure that there is not another lower value down the line and if there is I'd have to consider how much lower it is and if it is acceptable.

- PLSR

```
components <- 1:25
plsr_mses <- trimmed_mses_for_all_components(plsr_model, 25)
plot(components, plsr_mses, type="l", xlab="components", ylab="mse")
```



Yet again we can see that the 5 is not a minimum. It seems like we were closer this time however and the minimum will be 6.

```
order(plsr_mses)[1]
```

```
## [1] 10
```

Indeed it does look like the 6 component model has the lowest trimmed mse on the training data (validation sets in CV).

Task 4)

In this task we will try to do the PCA by hand. We need to get the Z matrix as input for our LM input so we have have to get V matrix that we will multiply with the X matrix (for test) and for train we can take it directly from the princomp output. We also need to make sure we scale and center both train and test the same way and that we get the scaling and centering coefficients from the training data only. Once we build our lm model with the Z matrix input we can use to to predict on the Z_test built from X and V and predict the outcomes.

```
# extract the model matrix
X <- model.matrix(y~.-1, data = d)

# get principal components, center and scale, also cap max components
pca <- princomp(X[train_indexes, ], cor=TRUE)

# take first 10 components
train_input <- data.frame(pca$scores[, 1:10])
train_input$y <- d$y[train_indexes]

# train lm with PCA components as input and y as output
pca_lm <- lm(y~., train_input)
```

```r
# Extract V to create test Z
V <- pca$loadings[, 1:10]

# prepare Z
X_test_scaled_centerd <- scale(X[test_indexes,], center = pca$center, scale = pca$scale)
Z <- X_test_scaled_centerd %*% V
Z_df <- data.frame(Z)

# predict
pca_predictions <- predict(pca_lm, newdata = Z_df)

# calculate trimmed_mse
manual_mse <- mse(pca_predictions, d$y[test_indexes])
manual_mse
```

```
## [1] 0.1605652
```

```r
trimmed_pcr
```

```
## [1] 0.1605652
```

As we can see the MSEs are the same as expected.