Compulsory exercise 2: Group XX

TMA4268 Statistical Learning V2018

NN1, NN2 and NN3

14 mars, 2018

1a)

- Q1: We have d possible predictors. For each predictor, we can either use it, or we can not use it. Hence we have two unique possibilities in d variables, resulting in a total of 2^d possible combinations. Thus, the number of different regression models is given by 2^d. Another way to see this is by investigating each model complexity independently. Say we use k predictors in our model. We then choose k different predictors from d possible ones, and we can not choose the same twice. Hence, the number of possible combinations using k predictors is given by the standard nCr-formula $\binom{d}{k}$. Now we have our d different choices of k, which all gives different linear regression models. In total, the number of models is

$$\binom{d}{0} + \binom{d}{1} + \cdots + \binom{d}{d}.$$

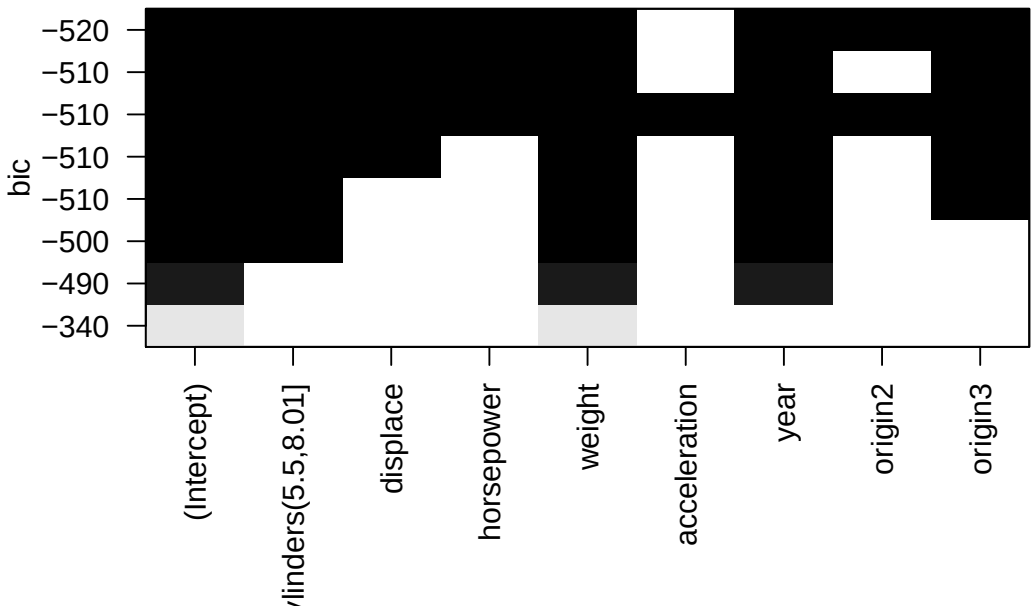
  The binomial theorem states that

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k.$$

  . By setting $x = 1$, we see that the number of different linear regression models is given by

$$\sum_{k=1}^{d} \binom{d}{k} = 2^d.$$

- Q2: The following algorithm illustrates how the best subset method finds the best model amongst the 2^d possible ones: Denote first by $\mathcal{M}_0$ the model using no predictors. This model simply predicts the sample mean for new observations. Next, for $k = 1, 2, \ldots, d$, fit all $\binom{d}{k}$ models, by solving the equation $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$, where $\boldsymbol{X}$ and $\boldsymbol{Y}$ only contains the predictors for one specific model. Calculate the RSS, and choose the model with the lowest RSS amongst the $\binom{d}{k}$ models, to be considered later. Now after the for-loop, we have $d+1$ possible models to consider. Calculate $BIC = \frac{1}{n}(RSS + log(n)k\hat{\sigma}^2)$ for each, where $\hat{\sigma}^2$ is a unbiased estimator for the variance, and choose the model with the lowest BIC. This is the best model by the best subset method, using BIC-criterion. R^2 is given by $1 - RSS/TSS$. In this expression, we do not care how many predictors are being used. This is a problem, because training error is a poor estimate for test error, since it may cause overfitting. A possible solution to this is to add penalties, depending on how many predictors we use. If we were to use R^2 instead of BIC to choose the best model, our best model would surely contain all d predictors since over-fitting would make our model fit the training data better. This is unwanted as we would rather have a model that best fit test data, which BIC better simulates.



1b)

- Q3: As explained in Q2, RSS is calculated for all models, and for each model complexity, we choose the model with the lowest RSS as our best model. The `regsubsets`-function in R performs the actual calculations, and in the summary we can see which predictors corresponds to the lowest RSS for each model complexity. We see from this summary, which is printed in the task, that the best model with two predictors uses the "weight" and "year" columns, in addition to the intercept.
- Q4: Now assuming we have the best model for each model complexity, we still want to choose which model complexity to use. As discussed in Q2, we expect to have the smallest RSS for the highest possible model complexity in the training set, but since we want to avoid overfitting, we add penalties to more complex models. Therefore, for each model complexity, we add the penalties according to the BIC-criterion, and then choose the model with the lowest value as our best method. The *regsubsets*-function in R also does the calculations for the penalties, using BIC-criterion in addition to other penalty-adding methods. Thus we can from the `regsubsets`-function also retrieve the BIC-values for each model complexity, and choose the lowest one as our model. The figure above, which is the result of plotting the `regsubsets`-data also contains the necessary information for choosing the best model. On the y scale, we see BIC-values for the models of different complexity, and on the x-scale we see which predictors are contained in the best model of this complexity, as a black box. We choose the model with the lowest BIC-value, which we see has $BIC \approx -520$, and contains seven predictors, all but acceleration. Hence, this is our best model according to the best subset method using BIC-criterion on the training data. From the prints in the task we see the more exact value $BIC = -516.9417$, which was rounded off to -520. The following R-code fits this model to the training data, by using the `lm`-function. The summary output gives us information about how well the model fit. We see that all of the t-values are fairly high, and the p-values for all but one are within the 0.001-significance level, while the last one is within the 0.01 significance level. It is therefore reasonable to assume there is a linear connection between miles per gallon and the other variates. We also see R-squared-values relatively

close to 1, which indicates a good model fit. We have $RSS = RSE^2(n-k-1)$, so that

$$MSE = \frac{RSS}{n} = \frac{RSE^2(n-k-1)}{n} = \frac{3.233^2 \cdot 305}{313} = 10.185.$$

The train-MSE is also calculated in the code below, giving the result 10.184, which is different probably because of roundoff-error. We have also performed an Anderson-Darling test for Normality. The p-value for this test is $9.241 \cdot 10^{-6}$, which is very small, and indicates that our data does not have normal distributed errors. This puts some doubt in our originl assumption that our data follows a linear relationship with normal distributed noise.

```
library(nortest)
trainLm=lm(mpg~. -acceleration, data=ourAutoTrain)
summary(trainLm)
ad.test(rstudent(trainLm))
yhat_train=predict(trainLm,ourAutoTrain)
yhat_test=predict(trainLm,ourAutoTest)
MSE_train=mean((ourAutoTrain$mpg-yhat_train)^2)
MSE_train
MSE_test=mean((ourAutoTest$mpg-yhat_test)^2)
MSE_test
```

```
##
## Call:
## lm(formula = mpg ~ . - acceleration, data = ourAutoTrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.7254 -2.0561 -0.3412  1.7122 12.9550
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -1.941e+01  4.589e+00  -4.230 3.09e-05 ***
## cylinders(5.5,8.01] -3.533e+00  7.469e-01  -4.730 3.43e-06 ***
## displace             3.279e-02  6.749e-03   4.858 1.90e-06 ***
## horsepower          -4.883e-02  1.184e-02  -4.124 4.80e-05 ***
## weight              -5.824e-03  6.185e-04  -9.415  < 2e-16 ***
## year                 7.849e-01  5.699e-02  13.771  < 2e-16 ***
## origin2              1.794e+00  6.204e-01   2.892  0.00411 **
## origin3              2.906e+00  5.943e-01   4.891 1.63e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.233 on 305 degrees of freedom
## Multiple R-squared:  0.8344, Adjusted R-squared:  0.8306
## F-statistic: 219.6 on 7 and 305 DF,  p-value: < 2.2e-16
##
##
##  Anderson-Darling normality test
##
## data:  rstudent(trainLm)
## A = 2.2684, p-value = 9.241e-06
##
## [1] 10.18351
## [1] 8.931018
```

- Q5: The code above also predicts new values for the test set, and calculates the test-MSE. Here, we got $MSE = 8.931$, which actually is smaller than the training MSE, indicating that we indeed have managed to avoid overfitting, and made a model which predicts new values pretty well.

1c)

- Q6. K-fold cross-validation involves dividing the data into k folds, $C_1, C_2, \ldots, C_k$. We then want to divide the data into a training and test set, and we choose one of the k parts as the test set, while we let the rest be training data. After fitting the model, we calculate MSE for the specific test folder, as

$$MSE_j = \frac{1}{n_k} \sum_{i \in C_j} (y_i - \hat{y}_i)^2,$$

where $n_k = n/k$. Furthermore, we want to use all the folds as testing data in turn, while we let the $k-1$ remaining folds be training data to fit different models. Thus, we calculate $MSE_1, MSE_2, \ldots, MSE_k$, and finally we give the cross-validation a value as

$$CV_k = \frac{n_k}{n} \sum_{j=1}^{k} MSE_j.$$

In our regression setting, when doing this for different complexities, we want to find which complexity gives the lowest CV_k, and we choose this as our best model.

- Q7. LOOCV suffers from quite high variance, since the training sets only differ with one observation. This makes the training sets have a very high correlation, and we choose our model based on an average of these corrolated models. This, in the end, can give a high variance. By using k-fold cross-validation instead we can lower the variance. Note however that this is at the cost of a higher bias, since the dataset used for fitting the models are smaller by a factor of $(k-1)/k$. k-fold cross-valudation with $k = 5$ or $k = 10$ is often seen as a compromise between the high variance and the high bias. In addition, it is more expensive to implement LOOVC, since you have to fit n data sets, instead of k, resulting in more calculations. It is thus less work using k-fold cross-validation.

1d

- Q8. The R-code for performing 10-fold cross validation is below.

```
library(caret)
library(leaps)
predict.regsubsets=function(object,newdata,id){
  form=as.formula(object$call[[2]])
  mat=model.matrix(form,newdata)
  coefi=coef(object,id=id)
  xvars=names(coefi)
  mat[,xvars]%*%coefi
}
k=10
set.seed(4268)
folds=sample(1:k,nrow(ourAutoTrain),replace=TRUE)
cv.errors=matrix(NA,k,8,dimnames=list(NULL,paste(1:8)))
for (j in 1:k){
  best.fit=regsubsets(mpg~.,data=ourAutoTrain[folds!=j,])
  for (i in 1:8){
    pred=predict.regsubsets(best.fit,ourAutoTrain[folds==j,],id=i)
```

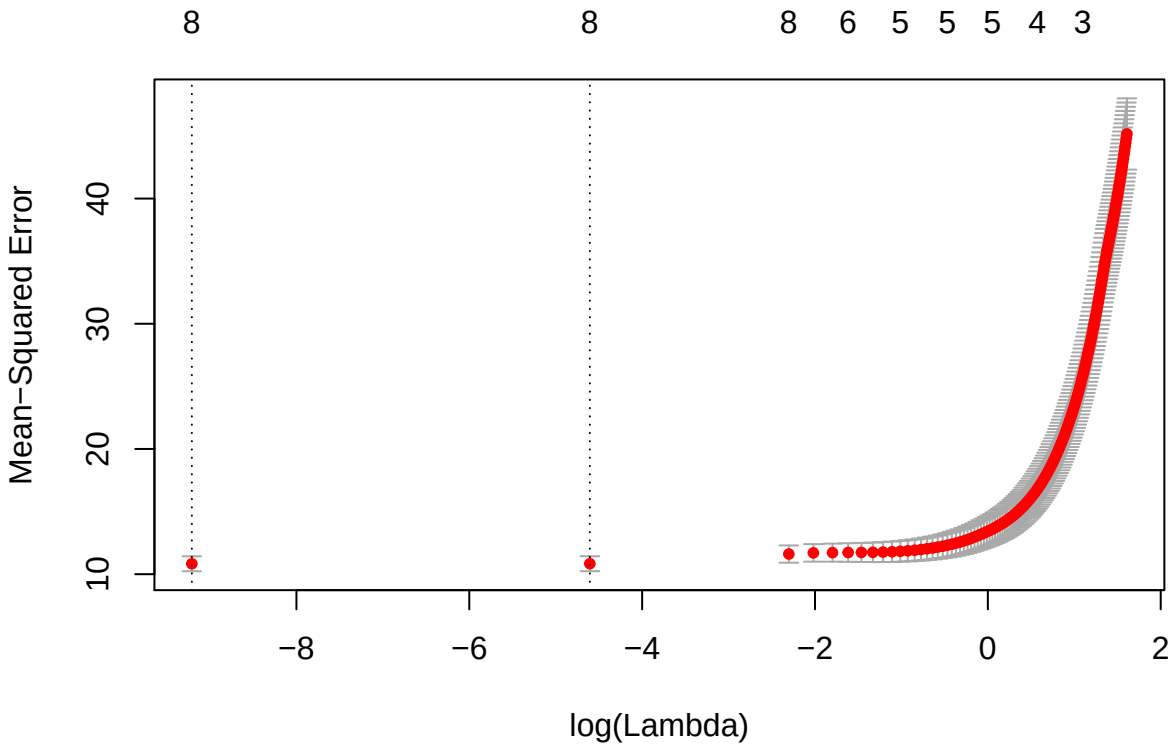
```
    cv.errors[j,i]=mean((ourAutoTrain$mpg[folds==j]-pred)^2)
  }
}
mean.cv.errors=apply(cv.errors,2,mean)
mean.cv.errors
which.min(mean.cv.errors)
```

```
##        1        2        3        4        5        6        7        8
## 20.04605 11.98844 11.92601 11.29254 11.71515 10.72818 10.56209 10.61032
## 7
## 7
```

- Q9. We see from the output of the code the MSE for the different complexities, and we also see we get the lowest MSE when using seven predictors. Hence, we want to use the same model complexity as we did for the BIC-criterion.
- Q10. Now that we have established which model complexity we want to use, we would like to fit this on our entire training set, to get the best model possible. We would therefore use the `regsubsets`-function on the training set, and choose the model using seven predictors. We do however note that this will give us the exact same model as we did for the BIC-criterion. Our comments on the model fit for the training set, and the MSE for the training set, is therefore identical to our results in Q4. Also, our predicted new values for the test set, and the MSE for the test set, is identical to our results in Q5. We therefore refer to Q4 and Q5 in this task.

2a) Explain figures

- Q11. λ is a parameter which draws the estimated coeffiecients towards zero in both ridge and lasso regression. For very large λ's, all the estimated coefficients are essentially zero. However, in ridge regression, none of the coefficients will be set exactly to zero. In comparison , the penalty term in lasso regression may set some of the coefficient to be exactly zero when λ is large enough. In Figure 1, we can see that this is the case, while the coefficients in Figure 2 do not seem to be exactly zero for any value of λ in the given interval. Thus, Figure 1 corresponds to lasso regression while Figure 2 corresponds to ridge regression.
- Q12. As stated, we can see that the tuning parameter λ draws the β's towards zero as λ increases. In lasso regression, all of the coefficients seem to become exactly zero, while in ridge regression, the coefficients are drawn towards zero, but are not exactly zero. When $\lambda = 0$, the penalty term in both ridge and lasso regression will disappear, so the coefficients will just be the same as with least squares estimation. At this value for λ, the bias is zero, but the variance may be high. As λ increases, the variance will become lower as the coefficients are shrinked and flexibility decreases. However, the bias will increase at the same time. In both ridge and lasso regression, when $\lambda \to \infty$, we have the null model, which means that all of the coefficient estimates except the intercept are zero. Then the variance becomes low, but the bias becomes large. At some point in between, the MSE will be at its lowest.
- Q13. Ridge regression will include all p covariates, and thus can not be used to perform model selection. However, lasso regression can be used in the same way as best subset selection, as the variables are forced to be exactly zero when λ is large. As the penalty λ increases, as we see in Figure 1, lasso regression may zero out some of the estimated coefficients and it thus performs a variable selection. We can say that the coefficients going slowest to zero represent the most important covariates. Here, both `year` and `cylinders(5.5,8.01]` seem to be important variables, similar to what we saw in Problem 1b. It is for some of the other coefficients in this case however difficult to see exactly for which value of λ they become zero, as they



2b) Finding the optimal λ

- Q14: The function cv.glmnet performs a k-fold cross-validation for a sequence of λ's, produces a plot and returns an object with different ingredients.
- Q15. The plot shows the cross-validation curve, including the upper and lower standard deviation curves along the sequence of λ's. The λ with the lowest cross-validated MSE in the plot can be chosen as the optimal λ:

```
cv.out$lambda.min
```

```
## [1] 1e-04
```

The `1se-rule`, is another way to choose which λ is optimal. In the object returned by `cv.glmnet`, one of the values is `lambda.1se`, which is the largest value of λ such that the error is within one standard error of the minimum.

- Q16: By using the `1se-rule`, we get that the optimal λ is

```
cv.out$lambda.1se
```

```
## [1] 0.01
```

2c) Prediction

- Q17: Using lasso regression with the optimal value of λ according to the `1se-rule`, $\lambda = 0.01$, we can fit the model. The coefficient estimates are

```
coef(cv.out,s="lambda.1se")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                                      1
## (Intercept)          -21.407540632
## cylinders(5.5,8.01]   -3.374630096
## displace               0.030954456
## horsepower            -0.035558382
## weight                -0.006071262
## acceleration           0.122203782
## year                   0.781813677
## origin2                1.675613651
## origin3                2.832358817
```

Thus, we get the model fit $\hat{y} = -21.408 - 3.375x_{cylinders} + 0.031x_{displace} - 0.036x_{horsepower} - 0.006x_{weight} + 0.122x_{acceleration} + 0.782x_{year} + 1.676x_{origin2} + 2.832x_{origin3}$.

- Q18: Using this model with optimal λ chosen according to the `1se-rule`, we can predict `mpg` for cars based on their data. For a car with 4 cylinders, `displace=150`, `horsepower=100`, `weight=3000`, `acceleration=10`, `year=82` and which comes from Europe, `mpg` is predicted as

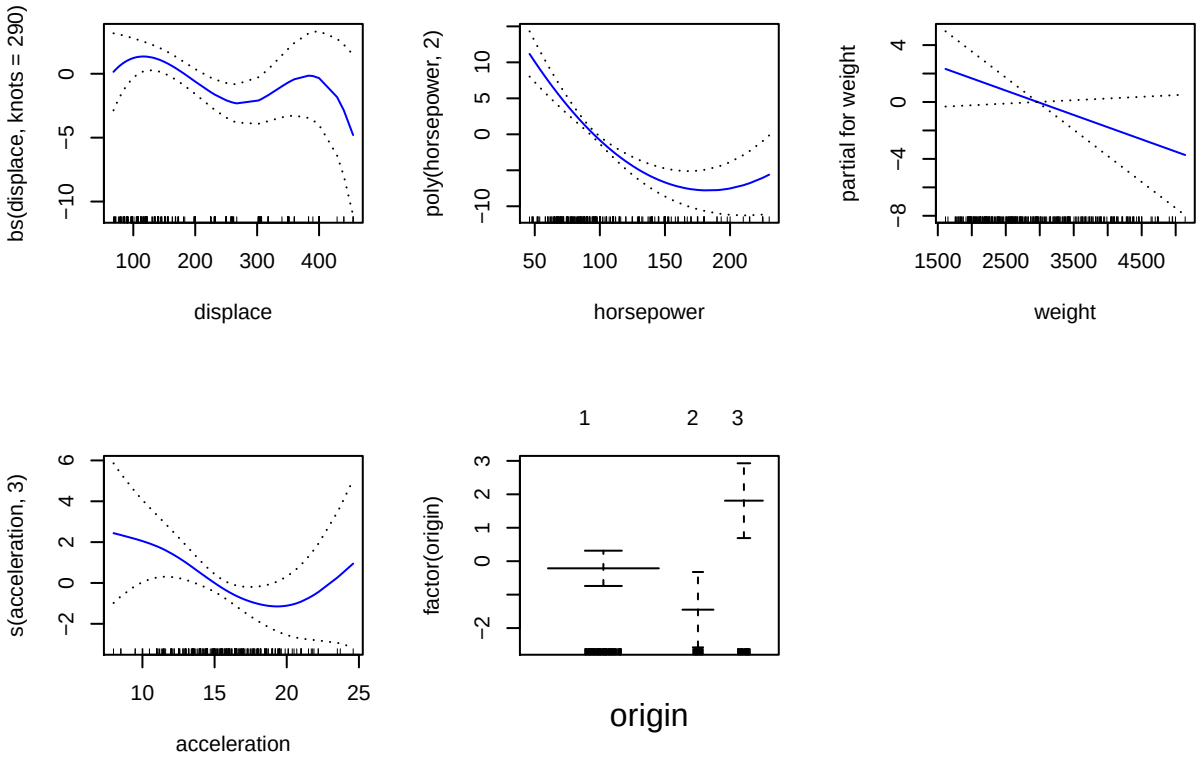
```
predict(cv.out, newx =matrix(c(0,150,100,3000,10,82,1,0),nrow=1), s = "lambda.1se")
```

```
##             1
## [1,] 28.47238
```

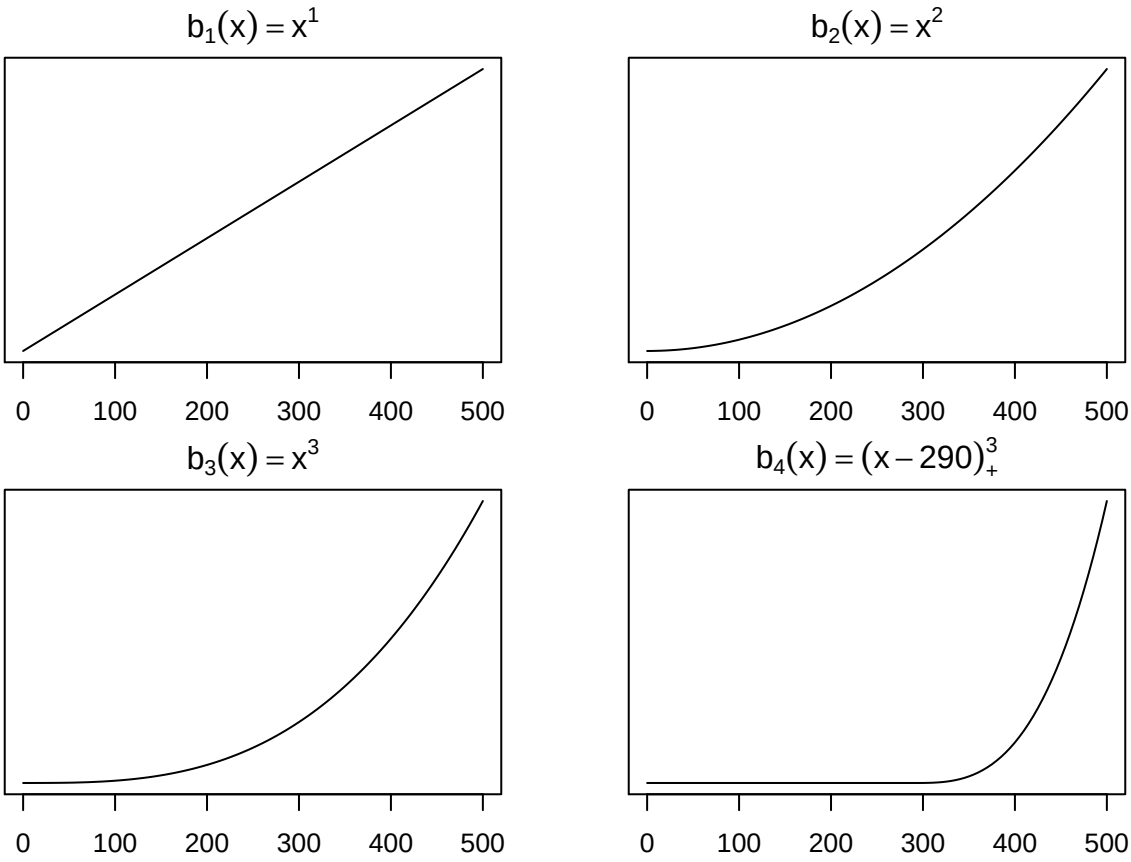
3a)

- Q19: Fitting the specified `gam`

```
library(gam)
gamobject = gam(mpg ~ bs(displace, knots  = 290) + poly(horsepower, 2)  + weight + s(acceleration, 3) +
par(mfrow=c(2,3))
plot(gamobject,se=TRUE,col="blue")
M = 4
cp = c(290)
l = c(0,500)
par(mfrow = c(2,2), mar = c(2,2,2,2))
```




```
s=sapply(1:(M-1), function(y) curve(x^y, l[1], l[2], yaxt = "n",
 ylab = "", xlab = "", main = bquote(b[.(y)](x) == x^.(y))))
s=sapply(1:length(cp), function(y) curve(pmax(0,x-cp[y])^3, l[1], l[2], yaxt = "n",
 ylab = "", xlab = "displacement", main = bquote(b[.(y+M-1)](x) == (x-.(cp[y]))["+"]^.(M-1))))
```



- Q20: A basis for the cubic spline (displace) will be a combination of a step function at the knot and polynomials of degree, will be $X, X^2, X^3, (X - 290)_+^3$.