STK2100 Oblig 1

Håkon Kvernmoen

2/4/2021

Problem 1

a)

First we need to load the data. The code sample for loading did not work for me (got a 400 bad request error). Assuming the datafile "nuclear.dat" is located in the same folder as this file, we load the data and attach it for easier use.

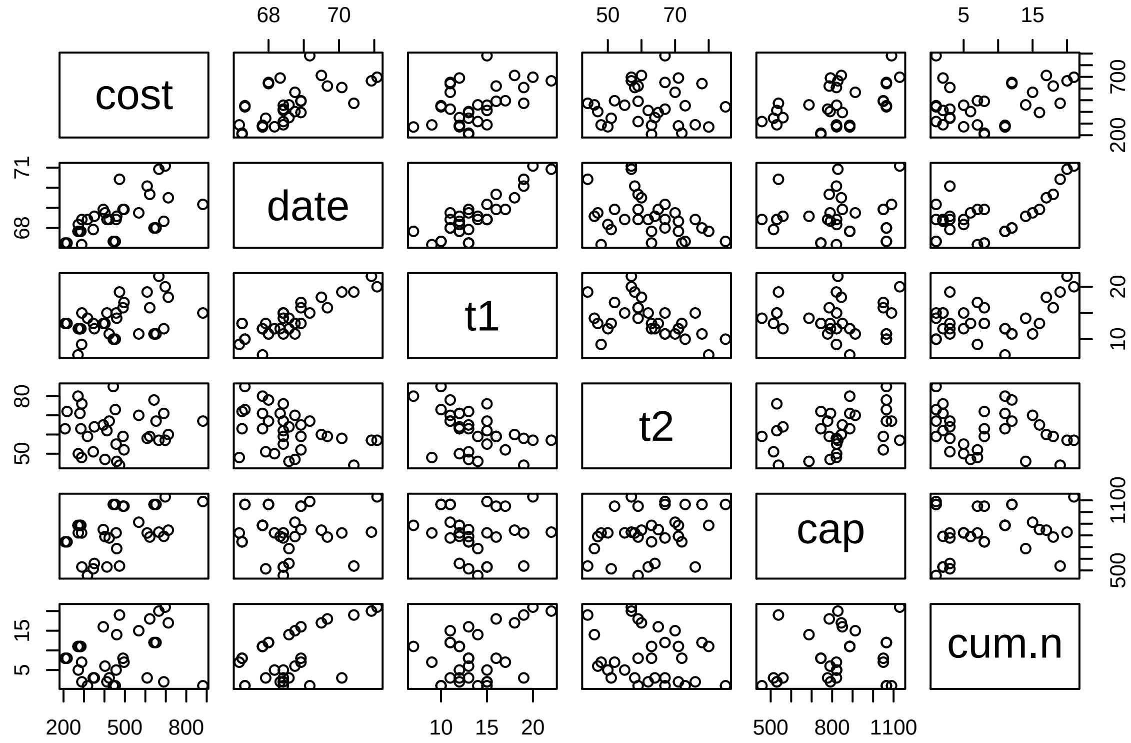
```r
nuclear = read.table("nuclear.dat", sep="\t", header=T)
attach(nuclear)
```

We notice that `pr`, `ne`, `ct`, `bw` and `pt` are binary variables, so we set them as factors.

```r
nuclear$pr = as.factor(nuclear$pr)
nuclear$ne = as.factor(nuclear$ne)
nuclear$ct = as.factor(nuclear$ct)
nuclear$bw = as.factor(nuclear$bw)
nuclear$pt = as.factor(nuclear$pt)
```

To investigate the data we plot the numerical features against each other. There seems to be some correlation between `date` and `t1`

```r
plot(nuclear[,sapply(nuclear, is.numeric)])
```



b)

The standard assumption on the noise terms ϵ_i are.

1. The error terms are normally distributed with a mean of 0
2. The variance σ^2 of this normal distribution is constant
3. The error terms are independent, ϵ_i does not influence ϵ_j

Important?

We will now try to fit the model using all the features. As cost is always positive, we fit the log of the cost as a response variable. With y_i being the i'th observation of the cost, we will try to fit the model.

$$log(y_i) = \beta_0 + \sum_{j=1}^{p} x_{i,j} + \epsilon_i$$

```
all.fit = lm(log(cost)~., data = nuclear)
summary(all.fit)
```

```
##
## Call:
## lm(formula = log(cost) ~ ., data = nuclear)
##
```

```
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.284032 -0.081677  0.009502  0.090890  0.266548
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.063e+01  5.710e+00  -1.862  0.07662 .
## date         2.276e-01  8.656e-02   2.629  0.01567 *
## t1           5.252e-03  2.230e-02   0.236  0.81610
## t2           5.606e-03  4.595e-03   1.220  0.23599
## cap          8.837e-04  1.811e-04   4.878 7.99e-05 ***
## pr1         -1.081e-01  8.351e-02  -1.295  0.20943
## ne1          2.595e-01  7.925e-02   3.274  0.00362 **
## ct1          1.155e-01  7.027e-02   1.644  0.11503
## bw1          3.680e-02  1.063e-01   0.346  0.73261
## cum.n       -1.203e-02  7.828e-03  -1.536  0.13944
## pt1         -2.220e-01  1.304e-01  -1.702  0.10352
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 0.1697 on 21 degrees of freedom
## Multiple R-squared:  0.8635, Adjusted R-squared:  0.7985
## F-statistic: 13.28 on 10 and 21 DF,  p-value: 5.717e-07
```

c)

We will now remove the term with the largest P-value. Observing the summary of the linear model, we see that `t1` has the largest P-value at 0.81610. This is sensible to do since the P-value is a measure of the correctness of the null-hypothesis (H_0). A large P-value as in this case indicates that there is a very little statistical basis for `t1` to be a good predictor for log(`cost`) and is thus neglected.

```
all_no_t1.fit = lm(log(cost)~.-t1, data= nuclear)
summary(all_no_t1.fit)
```

```
##
## Call:
## lm(formula = log(cost) ~ . - t1, data = nuclear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.28898 -0.07856  0.01272  0.08983  0.26537
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.161e+01  3.835e+00  -3.027 0.006187 **
## date         2.431e-01  5.482e-02   4.435 0.000208 ***
## t2           5.451e-03  4.449e-03   1.225 0.233451
## cap          8.778e-04  1.755e-04   5.002 5.25e-05 ***
## pr1         -1.035e-01  7.944e-02  -1.303 0.205922
## ne1          2.607e-01  7.738e-02   3.368 0.002772 **
## ct1          1.142e-01  6.853e-02   1.667 0.109715
## bw1          2.622e-02  9.423e-02   0.278 0.783401
## cum.n       -1.220e-02  7.626e-03  -1.599 0.124034
## pt1         -2.157e-01  1.249e-01  -1.727 0.098181 .
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 0.166 on 22 degrees of freedom
## Multiple R-squared:  0.8631, Adjusted R-squared:  0.8072
## F-statistic: 15.42 on 9 and 22 DF,  p-value: 1.424e-07
```

We observe that there are some change in the P-values for a lot of the features after we excluded `t1`. This is probably due to correlation between the features. We would ideally have linearly independent explanatory variables. In example a change in `cap` should not influence any of the other explanatory variables, but this is not the case. On the other hand, the changes in P-values are not huge and the coefficients estimates seems relatively unchanged. In addition the standard error for the coefficients seems to decrease and we continue these modifications.

d)

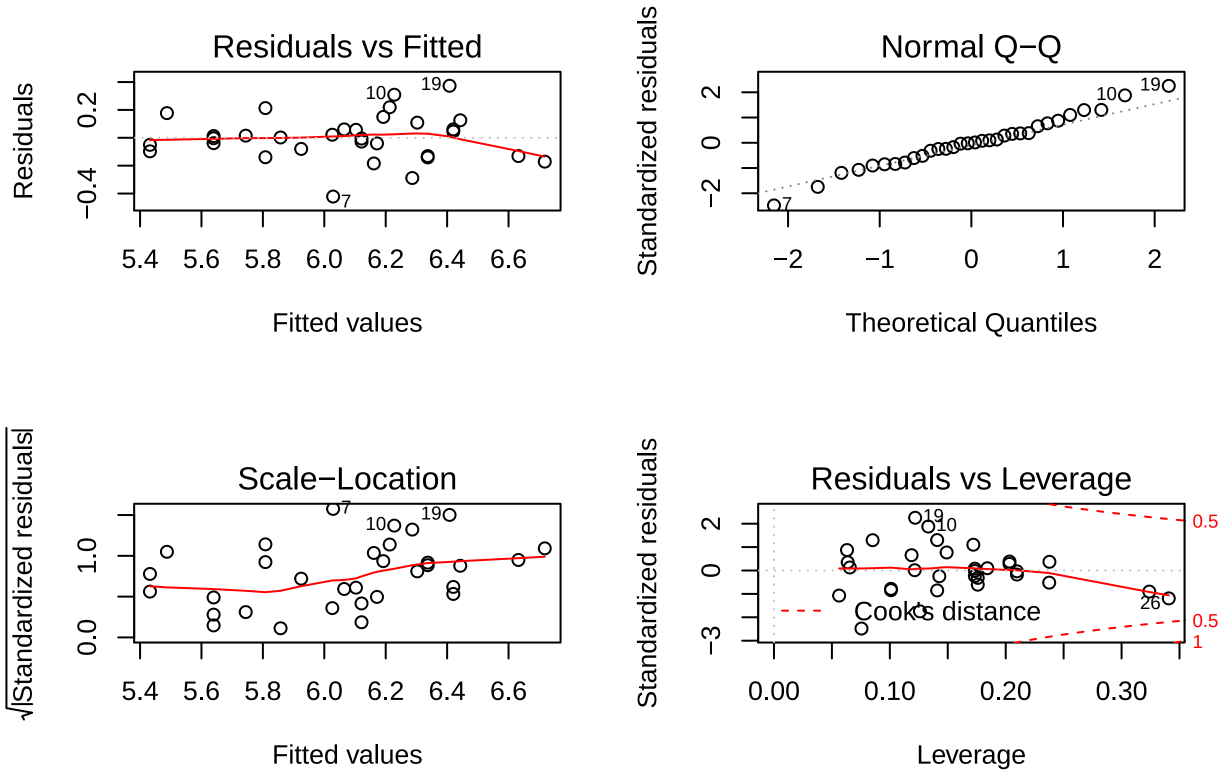
We now want to fit our model, remove the explanatory variable with a P-value larger than 0.05 and repeat this until we have a model where all explanatory variables have P-values smaller than 0.05. We then implement a backward substitution algorithm. We note that we do not want to remove the intercept even tough its P-value can be larger than 0.05.

```r
nuclear_backwards_sub <- data.frame(nuclear)
for (i in 1:ncol(nuclear)) {
    p_values.fit <- lm(log(cost)~., data=nuclear_backwards_sub)
    # -1 since we don’t want to remove intercept
    p_vals <- summary(p_values.fit)$coefficients[-1,4]
    max_idx <- as.integer(which.max(p_vals))

    if(p_vals[max_idx] < 0.05) {
      break
    }
    else {
      # Add one since we don’t want to remove targert variable (cost).
      nuclear_backwards_sub <- nuclear_backwards_sub[,-(max_idx+1)]
    }
}

summary(p_values.fit)
```

```
##
## Call:
## lm(formula = log(cost) ~ ., data = nuclear_backwards_sub)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42160 -0.10554 -0.00070  0.07247  0.37328
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.5035539  2.5022087  -1.800 0.083072 .
## date         0.1439104  0.0363320   3.961 0.000491 ***
## cap          0.0008783  0.0001677   5.238 1.61e-05 ***
## ne1          0.2024364  0.0751953   2.692 0.012042 *
## pt1         -0.3964878  0.0963356  -4.116 0.000326 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1767 on 27 degrees of freedom
## Multiple R-squared:  0.8096, Adjusted R-squared:  0.7814
## F-statistic:  28.7 on 4 and 27 DF,  p-value: 2.255e-09
```

We are then left with 4 explanatory variables. Two of them continues (`date, cap`) and two binary (`ne, pt`).

```
par(mfrow=c(2,2))
plot(p_values.fit)
```



The model seems to fit reasonably well, but the simple linear model does not seem to capture the complexity of the data. From the residuals against fitted values plot we see that the residual mean seems to be around 0 for the lower fitted values, but for higher values log(`cost`) > 6.1 the mean seems to not be centered around 0. This is probably due to a small data set. The QQ-plot confirms this displaying that data point 7, 10 and 19 deviates a lot from the theoretical quantiles. From the Standardized residuals against leverage plot point 10 and 19 again shows a high deviance and should probably be considered outliers. Lastly point 26 deviates from the mean of zero as well as having a high leverage (influence on the model) and should also be considerd as an outlier.

e)

Finally we calculate the MSE

```
MSE = (cost-exp(fitted.values(p_values.fit)))^2
MSE = mean(MSE)
MSE
```

```
## [1] 7443.737
```

This MSE is high and most of the contribution comes from the data points discussed in d). There is also a major drawback to this approach, as we have tested the correctness of our model (using the MSE) on the same data that we used to fit the model. The MSE resulted in a large value either way, but this should generally be avoided as more flexible models can follow the data used too closely (overfitting) and not represent the whole population. A better approach would be to split the data into a training and testing set, using only the training data to fit the model and then evaluate the correctness using the training data. Other techniques such as cross-validation can also be used.

f)

We begin with the likelihood function for a linear model with Gaussian error terms. Writing $\mathbf{x}_i$ as the i'th data point with all q features.

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(y_i - \mathbf{x}_i^T\boldsymbol{\theta})^2}{2\sigma^2}\right)}$$

To avoid the product sum we take the log of the likelihood, giving the log-likelihood. Note that log refers to the natural logarithm.

$$log(L(\boldsymbol{\theta})) = \sum_{i=1}^{n} -\frac{1}{2}log(\sigma^2) - \frac{(y_i - \mathbf{x}^T\boldsymbol{\theta})^2}{2\sigma^2} + const = -\frac{n}{2}log(\sigma^2) - \frac{||\mathbf{y} - \mathbf{X}\boldsymbol{\theta}||^2}{2\sigma^2} + const$$

Inserting the maximum liklihood estimate $\hat{\boldsymbol{\theta}}$ and estimating the variance $\hat{\sigma^2} = ||\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\theta}}||^2/n$, the log likelihood evaluated at the at the maximum likelihood estimate becomes.

$$log(L(\hat{\boldsymbol{\theta}})) = -\frac{n}{2}log(\hat{\sigma^2}) - \frac{n\sigma^2}{2\sigma^2} + const = -\frac{n}{2}log(\hat{\sigma^2}) - \frac{n}{2} + const.$$

As we will use this to preform variable selection, we absorb n into the const term, as it will be constant no matter the amount of variables we include in the model. Rearranging:

$$-2log(\hat{\boldsymbol{\theta}}) = nlog(\sigma^2) + const$$

Inserting this into the two equation for BIC and AIC will retrive the result presented in AIC.LM and BIC.LM

g)

We first use backward selection with the AIC.

```
library("MASS")

AIC.fit = lm(log(cost)~.,data = nuclear)
AIC.fit = stepAIC(AIC.fit, direction = "backward")
```

```
## Start:  AIC=-105.01
## log(cost) ~ date + t1 + t2 + cap + pr + ne + ct + bw + cum.n +
##     pt
##
##         Df Sum of Sq     RSS      AIC
## - t1     1   0.00160 0.60603 -106.930
## - bw     1   0.00345 0.60788 -106.832
## <none>               0.60443 -105.014
## - t2     1   0.04284 0.64727 -104.823
## - pr     1   0.04826 0.65269 -104.556
## - cum.n  1   0.06792 0.67235 -103.607
## - ct     1   0.07781 0.68224 -103.140
## - pt     1   0.08337 0.68781 -102.879
## - date   1   0.19899 0.80343  -97.907
## - ne     1   0.30859 0.91302  -93.815
## - cap    1   0.68497 1.28940  -82.770
##
## Step:  AIC=-106.93
## log(cost) ~ date + t2 + cap + pr + ne + ct + bw + cum.n + pt
##
##         Df Sum of Sq     RSS      AIC
## - bw     1   0.00213 0.60816 -108.818
## <none>               0.60603 -106.930
## - t2     1   0.04135 0.64738 -106.818
## - pr     1   0.04680 0.65283 -106.550
## - cum.n  1   0.07045 0.67648 -105.411
## - ct     1   0.07654 0.68257 -105.124
## - pt     1   0.08216 0.68819 -104.862
## - ne     1   0.31255 0.91858  -95.621
## - date   1   0.54190 1.14793  -88.489
## - cap    1   0.68916 1.29518  -84.627
##
## Step:  AIC=-108.82
## log(cost) ~ date + t2 + cap + pr + ne + ct + cum.n + pt
##
##         Df Sum of Sq     RSS      AIC
## <none>               0.60816 -108.818
## - pr     1   0.05738 0.66554 -107.932
## - t2     1   0.06379 0.67195 -107.626
## - cum.n  1   0.06839 0.67656 -107.407
## - ct     1   0.07440 0.68257 -107.124
## - pt     1   0.08066 0.68882 -106.832
## - ne     1   0.31375 0.92192  -97.505
## - date   1   0.54592 1.15408  -90.318
## - cap    1   0.68739 1.29556  -86.617
```

Then the same with BIC

```
AIC.fit$coefficients
```

```
##   (Intercept)          date            t2           cap           pr1
## -1.168690e+01  2.438055e-01  6.017900e-03  8.739059e-04 -1.098636e-01
##           ne1           ct1         cum.n           pt1
##  2.610950e-01  1.110869e-01 -1.175797e-02 -2.071320e-01
```

```r
n = nrow(nuclear)
BIC.fit = lm(log(cost)~., data=nuclear)
BIC.fit = stepAIC(BIC.fit, direction="backward", k=log(n))
```

```
## Start:  AIC=-88.89
## log(cost) ~ date + t1 + t2 + cap + pr + ne + ct + bw + cum.n +
##     pt
##
##          Df Sum of Sq     RSS     AIC
## - t1      1   0.00160 0.60603 -92.273
## - bw      1   0.00345 0.60788 -92.175
## - t2      1   0.04284 0.64727 -90.166
## - pr      1   0.04826 0.65269 -89.899
## - cum.n   1   0.06792 0.67235 -88.949
## <none>                0.60443 -88.891
## - ct      1   0.07781 0.68224 -88.482
## - pt      1   0.08337 0.68781 -88.222
## - date    1   0.19899 0.80343 -83.250
## - ne      1   0.30859 0.91302 -79.158
## - cap     1   0.68497 1.28940 -68.113
##
## Step:  AIC=-92.27
## log(cost) ~ date + t2 + cap + pr + ne + ct + bw + cum.n + pt
##
##          Df Sum of Sq     RSS     AIC
## - bw      1   0.00213 0.60816 -95.626
## - t2      1   0.04135 0.64738 -93.626
## - pr      1   0.04680 0.65283 -93.358
## <none>                0.60603 -92.273
## - cum.n   1   0.07045 0.67648 -92.219
## - ct      1   0.07654 0.68257 -91.933
## - pt      1   0.08216 0.68819 -91.670
## - ne      1   0.31255 0.91858 -82.430
## - date    1   0.54190 1.14793 -75.297
## - cap     1   0.68916 1.29518 -71.435
##
## Step:  AIC=-95.63
## log(cost) ~ date + t2 + cap + pr + ne + ct + cum.n + pt
##
##          Df Sum of Sq     RSS     AIC
## - pr      1   0.05738 0.66554 -96.207
## - t2      1   0.06379 0.67195 -95.900
## - cum.n   1   0.06839 0.67656 -95.681
## <none>                0.60816 -95.626
## - ct      1   0.07440 0.68257 -95.398
## - pt      1   0.08066 0.68882 -95.106
## - ne      1   0.31375 0.92192 -85.779
## - date    1   0.54592 1.15408 -78.592
## - cap     1   0.68739 1.29556 -74.892
##
## Step:  AIC=-96.21
## log(cost) ~ date + t2 + cap + ne + ct + cum.n + pt
##
```

```
##          Df Sum of Sq     RSS     AIC
## - t2     1   0.02447 0.69001 -98.517
## - cum.n  1   0.05351 0.71905 -97.198
## <none>               0.66554 -96.207
## - ct     1   0.10237 0.76791 -95.094
## - pt     1   0.12015 0.78570 -94.361
## - ne     1   0.28784 0.95339 -88.171
## - date   1   0.49109 1.15664 -81.987
## - cap    1   0.68019 1.34573 -77.141
##
## Step:  AIC=-98.52
## log(cost) ~ date + cap + ne + ct + cum.n + pt
##
##          Df Sum of Sq     RSS     AIC
## - cum.n  1   0.06006 0.75007 -99.312
## <none>               0.69001 -98.517
## - pt     1   0.11719 0.80720 -96.963
## - ct     1   0.12931 0.81932 -96.486
## - ne     1   0.27215 0.96216 -91.343
## - date   1   0.46672 1.15673 -85.450
## - cap    1   0.89456 1.58457 -75.379
##
## Step:  AIC=-99.31
## log(cost) ~ date + cap + ne + ct + pt
##
##         Df Sum of Sq     RSS     AIC
## <none>              0.75007 -99.312
## - ct    1   0.09317 0.84324 -99.031
## - ne    1   0.21478 0.96485 -94.720
## - pt    1   0.37487 1.12494 -89.807
## - date  1   0.55668 1.30675 -85.013
## - cap   1   0.83451 1.58458 -78.845
```

Both AIC and BIC starts with the full model and tries to remove one explanatory variable. The model with the highest likelihood is chosen and the cycle continues until there is no improvement in AIC-value. The two models we ended up with was

AIC: `date,t2, cap, pr, ne, ct, cum.n, pt`

BIC: `date, cap, ne, ct, pt`

h)

Both AIC and BIC contain to parts. Both include a $nlog(\hat{\sigma}^2)$ term, measuring how good the model fits the data, and is equal in both procedures. The difference is in the second terms, $2(q+2)$ and $log(n)(q+2)$ for AIC and BIC respectively. This terms act as a regulator for complexity. Models with high complexity (i.e. many explanatory variables) will be penalized. If we leave one explanatory variable out, $nlog(\hat{\sigma}^2)$ will be equal for the two procedures, assuming we have left out the same explanatory variable. The two complexity regulator terms ($2(q+2)$ and $log(n)(q+2)$) equal for one removal, and does not discriminate which explanatory variables left out. Thus starting at maximum complexity (including all explanatory variables), the order of what explanatory variable to remove, should be equal for the two procedures. The difference is in how much we should penalize the model. We have 32 observations, so $log(32) \approx 3.46 > 2$, thus for this dataset, BIC will punish complex models harder than AIC, which coincides with BIC resulting in fewer explanatory variables than AIC.

i)

Selecting a model using the “arbitrary” criteria of a p-value less than 0.05 might lead to some problems. Our sample size is low (32 observations) and we have quite a lot of explanatory variables relative to samples. P-values are dependent on sample size, so some variables that should have a p-value under this threshold might be picked out even though they are statistically significant (NB: only if the H_0 hypothesis is false). The stepwise AIC and BIC does not intringsantly have this problem as we are only trying to reduce AIC/BIC on the same data set. These step-wise schemes should be used carefully tough, since overfitting can be a problem. Both AIC and BIC tries many different models on the same data set and only reports if improvements are made and thus prone to data dredging.

j)

We define $Z = \mu + \sigma W$ such that $E[W] = 0, Var[W] = 1$ (the standard normal). Since Z/W is drawn from a continuous normal distribution write the expectation value as.

$$E[e^Z] = E[e^{\mu+\sigma W}] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{\mu+\sigma W} e^{-\frac{W^2}{2}} dW = \frac{e^{\mu}}{2\pi} \int_{-\infty}^{\infty} e^{\sigma W} e^{-\frac{W^2}{2}} dW$$

Now we want to complete the square in the exponent.

$$\sigma W - \frac{W^2}{2} = \frac{1}{2}\left(W^2 - 2\sigma W + \sigma^2\right) - \frac{\sigma^2}{2} = \frac{1}{2}\left(W - \sigma\right)^2 - \frac{\sigma^2}{2}$$

Inserting this into our integral

$$E[Z] = \frac{e^{\mu}}{2\pi} \int_{-\infty}^{\infty} e^{-\frac{1}{2}(W-\sigma)^2 + \frac{1}{2}\sigma^2} dW = \frac{e^{\mu+\frac{1}{2}\sigma^2}}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{(W-\sigma)^2}{2}} dW = e^{\mu+\frac{1}{2}\sigma^2}$$

When estimating the $\hat{\eta}$ we can consider two options, one reflecting the calculation just done $\hat{\eta}_2 = e^{\hat{\theta}+\frac{1}{2}\hat{\sigma}^2}$ and one without the standard deviation estimate $\hat{\eta}_1 = e^{\hat{\theta}}$. I think $\hat{\eta}_1$ is the preferred choice here, as the estimated coefficiants should be indipendent of the standard deviate of the Gaussian noise (assuming heteroskedasticity).

k)

We now define the new data point and use the 3 different models to make a prediction.

```r
nuclear.new = data.frame(date=70.0, t1=13, t2=50, cap = 800, pr=1, ne=0, ct=0, bw=1, cum.n=8, pt=1)

nuclear.new$pr = as.factor(nuclear.new$pr)
nuclear.new$ne = as.factor(nuclear.new$ne)
nuclear.new$ct = as.factor(nuclear.new$ct)
nuclear.new$bw = as.factor(nuclear.new$bw)
nuclear.new$pt = as.factor(nuclear.new$pt)

p_values.pred = predict.lm(p_values.fit, nuclear.new, se.fit=TRUE)
AIC.pred = predict.lm(AIC.fit, nuclear.new, se.fit=TRUE)
BIC.pred = predict.lm(BIC.fit, nuclear.new, se.fit=TRUE)

sprintf("P-value estimate: %f SE: %f", exp(p_values.pred$fit),exp(p_values.pred$se.fit))
```

```
## [1] "P-value estimate: 356.490489 SE: 1.122362"
```

```
sprintf("AIC estimate: %f SE: %f", exp(AIC.pred$fit),exp(AIC.pred$se.fit))
```

```
## [1] "AIC estimate: 390.897773 SE: 1.161922"
```

```
sprintf("BIC estimate: %f SE: %f", exp(BIC.pred$fit),exp(BIC.pred$se.fit))
```

```
## [1] "BIC estimate: 360.778910 SE: 1.117556"
```

The results are summerised in table 1.

Table 1: Different model predictions and standard errors

Model	Prediction	SE
P-values	356.490489	1.122362
AIC	390.897773	1.161922
BIC	360.778910	1.117556

Overall, the standard error seems to be relatively equal across all the models. The prediction however, differs. The P-value and BIC a approaches seems to agree, giving almost equal predictions. The AIC model on the other hand, differs with about 30-35 from the two other predictions. Both the BIC and P-value method resulted in fewer explanatory variables giving a simpler model, while the AIC model had more complexity. As the BIC and P-values models agreed, this might suggest a too complex model (using AIC) to predict log(`cost`). However without knowing what value of `cost` this new data-point gives, makes model selection difficult and more data-points should also be considered.

Problem 2

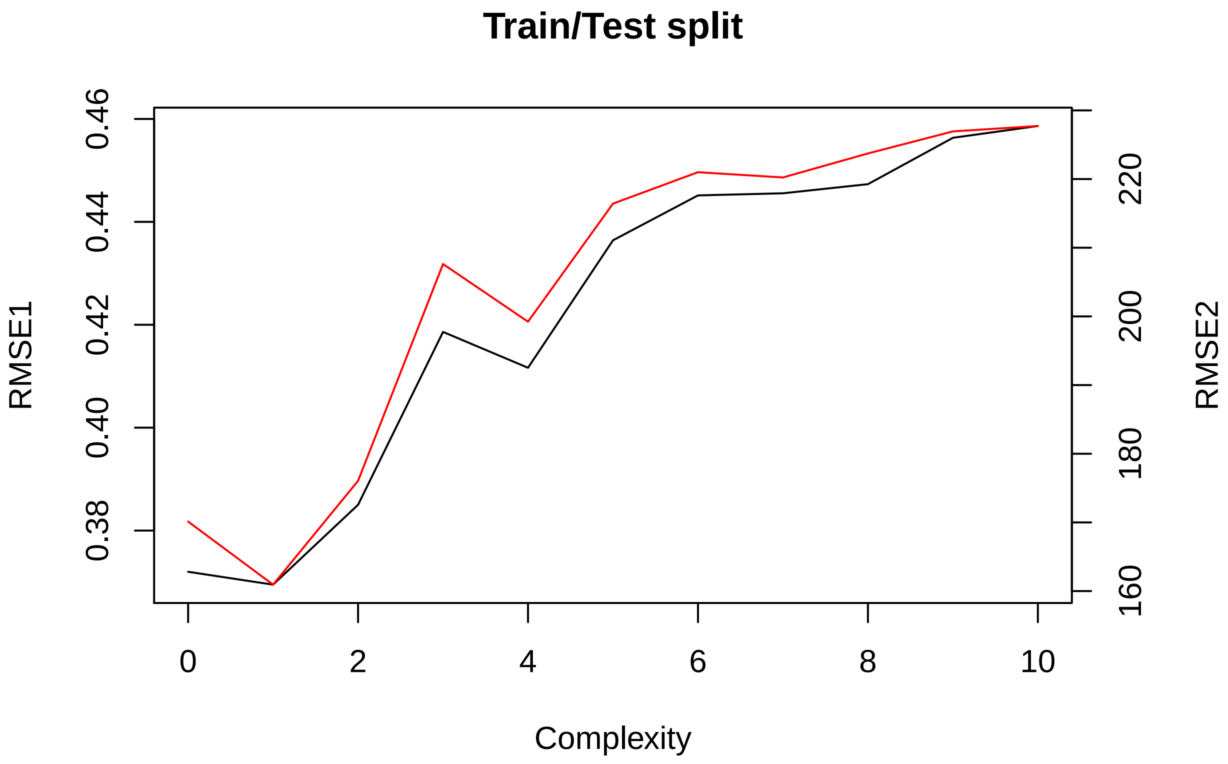
a)

The following code splits the data set (once, randomly) into a training and test set of two equal parts. Using the same stepAIC scheme as before, but opting for forward selection compared to backward selection. The lowest complexity of the model is set as one the simple constant term, while the highest complexity is set to include all the explanatory variables. The stepAIC scheme is preformed 10 times, with $i = 0, 1, ..., 10$. I here represents how many steps the AIC should take (how any explanatory variables should potentially be added) starting from the lowest complexity model. The RMSE from both the log(`cost`) and `cost` is calculated on the test set and then saved (for each i) and then finally plotted against the number of explanatory variables included.

Running the program creates the following plot, suggesting that the best model only has one parameter. There is some problems with this however, as there seems to be some randomness involved. Running the code multiple times produces different plots, but the minimum at 1 seems to be relatively stable. This is due to the test/train split being random. Every time we run the code, we train the model many times and pick the one that best fits the random testing set. The problem with this is that we might not get the best model, but simply the best model for that particularly testing set. The amount of different models tested is very large, so this does seem to be a problem.

```
n = nrow(nuclear)
ind = sample(1:n,n/2,replace=FALSE)
RMSE.test1 = rep(NA,11)
```

```r
RMSE.test2 = rep(NA,11)
model_narrow = lm(log(cost) ~ 1, data = nuclear)
model_wide = lm(log(cost) ~ ., data = nuclear)

for(i in 0:10)
{
 fit = stepAIC(model_narrow, direction="forward", steps=i,data=nuclear[ind,],
               scope=list(lower=model_narrow, upper=model_wide), k = 0)
 pred = predict(fit,nuclear[-ind,])
 RMSE.test1[i+1] = sqrt(mean((log(nuclear$cost)-pred)^2))
 RMSE.test2[i+1] = sqrt(mean((nuclear$cost-exp(pred))^2))
}
par(mar = c(5, 4, 4, 4) + 0.3)  # Leave space for z axis
plot(0:10,RMSE.test1,main="Train/Test split",xlab="Complexity",ylab="RMSE1",type="l") # first plot
par(new = TRUE)
plot(0:10,RMSE.test2, type = "l", axes = FALSE, bty = "n", xlab = "", ylab = "",col=2)
axis(side=4, at = pretty(range(RMSE.test2)))
mtext("RMSE2", side=4, line=3)
```



b)

The next code section uses 10-fold cross-validation. The same steps over i are done and for each step a model is selected (using stepAIC on the whole data set). Then the best model is chosen (for each step), the RMSE errors for both log(`cost`) and `cost` are calculated, but this time using 10-fold CV. The data-set is split into 10 parts, where 9 of these parts are used to train the optimal model found by the stepAIC. Then the RMSE is calculated on the last part (the hold out fold). This repeats until every part has been the hold out fold once, and the mean RSME over the 10-folds are calculated.

Running this code several times yields relatively stable results, but there is some randomness involved. We are interested in the minimum of these two curves, which seems to be around 4-5 explanatory variables. The randomness again come from the division of the folds, as this is done randomly. The trend is relativly stable, but it is dificult to give a definit minimum.

```
# library(lmvar)
# RMSE.cv1 = rep(0,10)
# RMSE.cv2 = rep(0,10)
# for(i in 0:10)
# {
# fit = stepAIC(model_narrow, direction="forward", steps=i,k=0,
#         scope=list(lower=model_narrow, upper=model_wide),trace=0)
#  fit = lm(formula(fit),data=nuclear,x=TRUE,y=TRUE)
#  #Note: the k in the command below has a different meaning than k above!!!
#  RMSE.cv1[i+1] = cv.lm(fit,k=10)$MSE_sqrt$mean
#  RMSE.cv2[i+1] = cv.lm(fit,k=10,log=TRUE)$MSE_sqrt$mean
# }
#
# par(mar = c(5, 4, 4, 4) + 0.3)  # Leave space for z axis
# plot(0:10,RMSE.cv1,xlab="Complexity",ylab="RMSE1",type="l") # first plot
# par(new = TRUE)
# plot(0:10,RMSE.cv2, main="10-fold CV",type = "l", axes = FALSE, bty = "n", xlab = "", ylab = "",col=2
# axis(side=4, at = pretty(range(RMSE.cv2)))
# mtext("RMSE2", side=4, line=3)
```

c)

We now want to implement Leave-one-out cross validation (LOOCV). From the `boot` library we make use of the glm function. This can be fed into a `cv.glm` function that be default uses LOOCV. Since we are fitting a ordinary least squares model we can make use of the fantastic shortcut

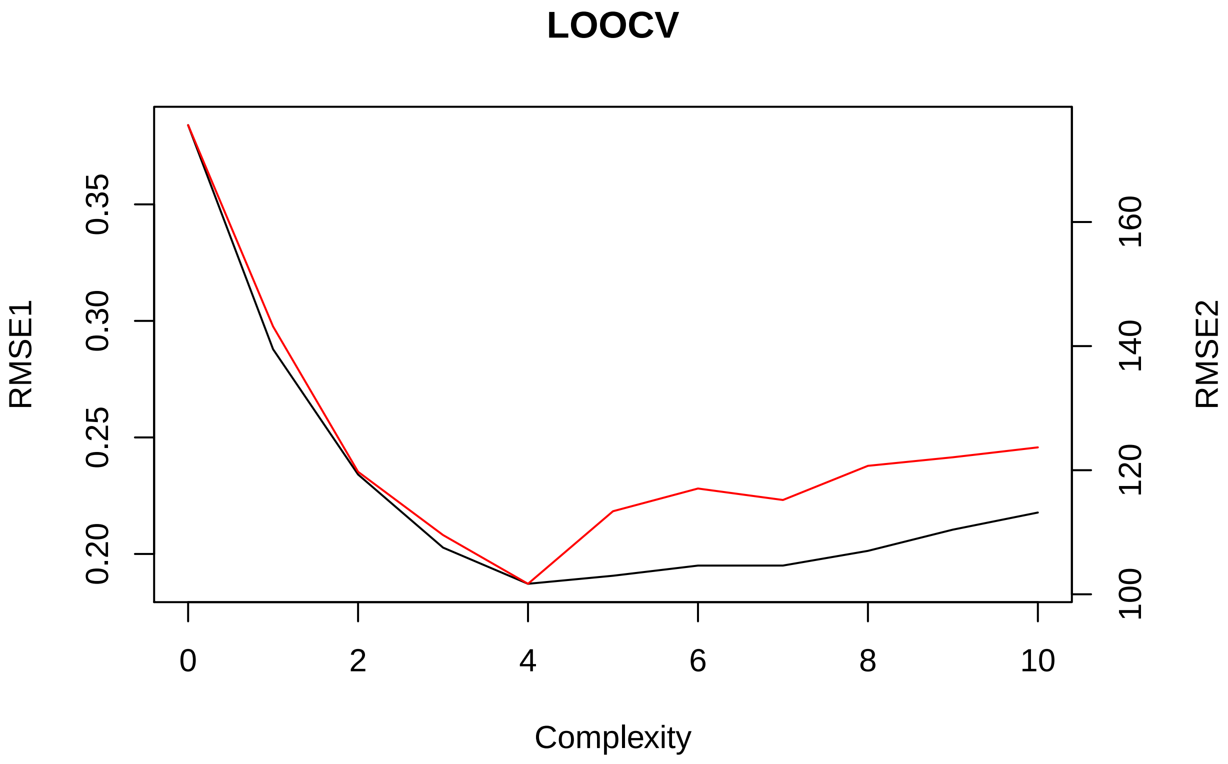
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2 \qquad\qquad RMSE = \sqrt{MSE}$$

Where h_i is the leverage. This actually makes the RMSE calculations a lot faster, since there is no need to fit the same model n times. Running the code many times confirms that these results are more stable than 10-fold CV and way more stable than Train/Test split. This yields a prefered model complexity of 4 explanatory variables.

```
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked _by_ '.GlobalEnv':
##
##     nuclear
```

```
RMSE.LOOCV1 = rep(0,11)
RMSE.LOOCV2 = rep(0,11)
for(i in 0:10) {
  fit = stepAIC(model_narrow, direction="forward", steps=i,k=0,
        scope=list(lower=model_narrow, upper=model_wide),trace=0)
 fit = glm(formula(fit),data=nuclear,x=TRUE,y=TRUE)

 RMSE.LOOCV1[i+1] = sqrt(cv.glm(nuclear ,fit)$delta[1])

 nuclear.diag = glm.diag(fit)
 yhat = exp(fitted(fit))
 y= exp(fit$y)
 RMSE.LOOCV2[i+1] = sqrt(mean((y-yhat)^2 / (1-nuclear.diag$h)^2))
}

par(mar = c(5, 4, 4, 4) + 0.3)  # Leave space for z axis
plot(0:10,RMSE.LOOCV1,xlab="Complexity",ylab="RMSE1",type="l") # first plot
par(new = TRUE)
plot(0:10,RMSE.LOOCV2, main="LOOCV",type = "l", axes = FALSE, bty = "n", xlab = "", ylab = "",col=2)
axis(side=4, at = pretty(range(RMSE.LOOCV2)))
mtext("RMSE2", side=4, line=3)
```



Problem 3

a)

Y_i is the response for the i'th data point, and each data point is a row of zeroes, except where $c_i = j$. Looking at the two models we first set $x_{i1} = 1$, and thus all other $x_{ij} = 0$ for $j = 2, 3, ...K$. If the two models are equal, this implies that $\beta_0 = \alpha_1$. Setting $x_{i2} = 1$ giving all other $x_{ij} = 0$ for $j = 1, 3, 4, ...K$ and thus $\beta_0 + \beta_2 = \alpha_2$. We can do this in general setting $x_{il} = 1$ thus $x_{ij} = 0$ for $j = 1, 2, ..., l-1, l+1, ..., K$ giving the relationship $\beta_j + \beta_0 = \alpha_j$. The relationship between the α and β coefficients is thus.

$$\beta_j = \begin{cases} \alpha_1 & \text{if} \quad j = 0 \\ \alpha_j - \alpha_0 & \text{if} \quad j = 1, 2, ..., K \end{cases}$$

The interpretation of this is that the β model has category $c_j = 1$ as a baseline, corresponding to the coefficient that is always present no matter the value of x_{ij} (β_0). For this model all other β_j is the response y_i from category $c_i = j$ *relative* to category $c_i = 1$.

The α model on the other hand, does not have this baseline value. The coefficients α_j corresponding to category $c_i = j$ is the response y_i, with no baseline (or zero as a baseline).

b)

The design matrix will have the form

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & \dots & \dots & x_{nk} \end{bmatrix}$$

We can then look at an arbitrary element of the product matrix $\mathbf{X}^T\mathbf{X}$. This will result in the sum of the products of the i'th row in $\mathbf{X}^T$ and the j'th row in $\mathbf{X}$

$$[\mathbf{X}^T\mathbf{X}]_{ij} = \sum_{l=1}^{n} x_{il}x_{lj}$$

Both x_{il} and x_{lj} can be equal to 1, but if $i \neq j$ they will never both be one, and thus the sum equals 0 (for $i \neq j$). For $i = j$, $x_{il} = x_{li} = 1$ if $c_i = j$ and thus the sum will count the number of appearances of variable c_i, the final result is then

$$[\mathbf{X}^T\mathbf{X}]_{ij} = \begin{cases} n_j & \text{if} \quad i = j \quad \text{where} \quad j = 1, 2, ..., K \\ 0 & \text{if} \quad i \neq j \end{cases}$$

We will also calculate $\mathbf{X}^T\mathbf{y}$. We can again look at an arbitrary element of the vector $\mathbf{X}^T y$, summing over the product of each row in $\mathbf{X}^T$ and $\mathbf{y}$.

$$[\mathbf{X}^T\mathbf{y}]_j = \sum_{l=1}^{n} x_{li}y_l$$

Again x_{li} will be 1 if $c_i = j$ and zero otherwise, thus effectively each row is the sum of the response y_j corresponding to this explanatory variable.

To find the least-squares estimate we make use of the general formula $\hat{\boldsymbol{\alpha}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. As $\mathbf{X}^T\mathbf{X}$ is a diagonal matrix, inverting this will only result in a diagonal matrix with elements $[(\mathbf{X}^T\mathbf{X})^{-1}]_{jj} = 1/n_j$. We have already calculated $\mathbf{X}^T\mathbf{y}$ so the least-squares estimate is then

$$[\hat{\boldsymbol{\alpha}}]_j = \frac{1}{n_j}\sum_{l=1}^{n} x_{li}y_l$$

$\hat{\boldsymbol{\alpha}}$ is then a vector where each row j is the mean of the response corresponding to category $c_i = j$.

c)

Using the relationship from a) we find that the first coefficient $\hat{\beta}_0$ is.

$$[\hat{\boldsymbol{\beta}}]_0 = [\hat{\boldsymbol{\alpha}}]_1 = \frac{1}{n_1}\sum_{l=1}^{n} x_{l1}y_l$$

The other coefficiants then follow

$$[\hat{\boldsymbol{\beta}}]_j = [\hat{\boldsymbol{\alpha}}]_j - [\hat{\boldsymbol{\alpha}}]_1 = \frac{1}{n_j}\sum_{l=1}^{n} x_{lj}y_l - \frac{1}{n_1}\sum_{l=1}^{n} x_{l1}y_l$$

Since this is a 1-to-1 mapping of the coefficiants, this also has to be the leaste squares estimate for $\boldsymbol{\beta}$.