Assignment 1

Group 2

2023-10-08

Task 1

Task 2

Task 3

Task 4

Task 5

Task 6

First we write functions to perform the steps of the precodure.

```
library(caret) # we get knnreg from here

# function to get y and X
get_xy <- function(p, sigma, N = 500) {
  X <- sapply(1:p, function(x) runif(n = N, min = -1, max = 1))
  epsilon <- rnorm(n = N, sd = sigma)
  Y <- exp(-8 * apply(X^2, MARGIN = 1, FUN = sum)) + epsilon
  df <- data.frame(Y = Y, X = X)
  return(df)
}

# function to get predictions at x0 = 0 for every f hat
get_fx0 <- function(x0 = 0, sigma, data) {
  x0_df <- as.data.frame(matrix(rep(x0, ncol(data)-1), nrow = 1))
  names(x0_df) <- names(data)[-1]
  f <- as.formula(paste0(names(data)[1], "~", paste(names(data)[-1], collapse = "+")))
  # estimate linear model
  l <- lm(data = data, formula = f)
  l_x0 <- predict.lm(l, newdata = x0_df) # its just the intercept what was clear
                                         # but if we specify other x0 the function
                                         # still works
  # estimate knn
  knn_mod <- knnreg(formula = f, data = data, k = 1)
  knn_x0 <- predict(knn_mod, newdata = x0_df)

  # output the fx0 value
```

```
  data.frame(linear = l_x0, knn = as.numeric(knn_x0))
}

  # function to calculate EPE
 epe <- function(fx0, x_0 = 0, p, mu = 0, sigma, f = function(x) exp(-8 * sum(x^2))) {
   x_0 <- rep(x_0, p)
   noise <- rnorm(n = nrow(fx0), mean = mu, sd = sigma)
   epe_lm <- mean((f(x_0) + noise  - fx0$linear)^2)
   epe_knn <- mean((f(x_0) + noise - fx0$knn)^2)
   return(data.frame(epe_linear = epe_lm, epe_knn = epe_knn))
}
```

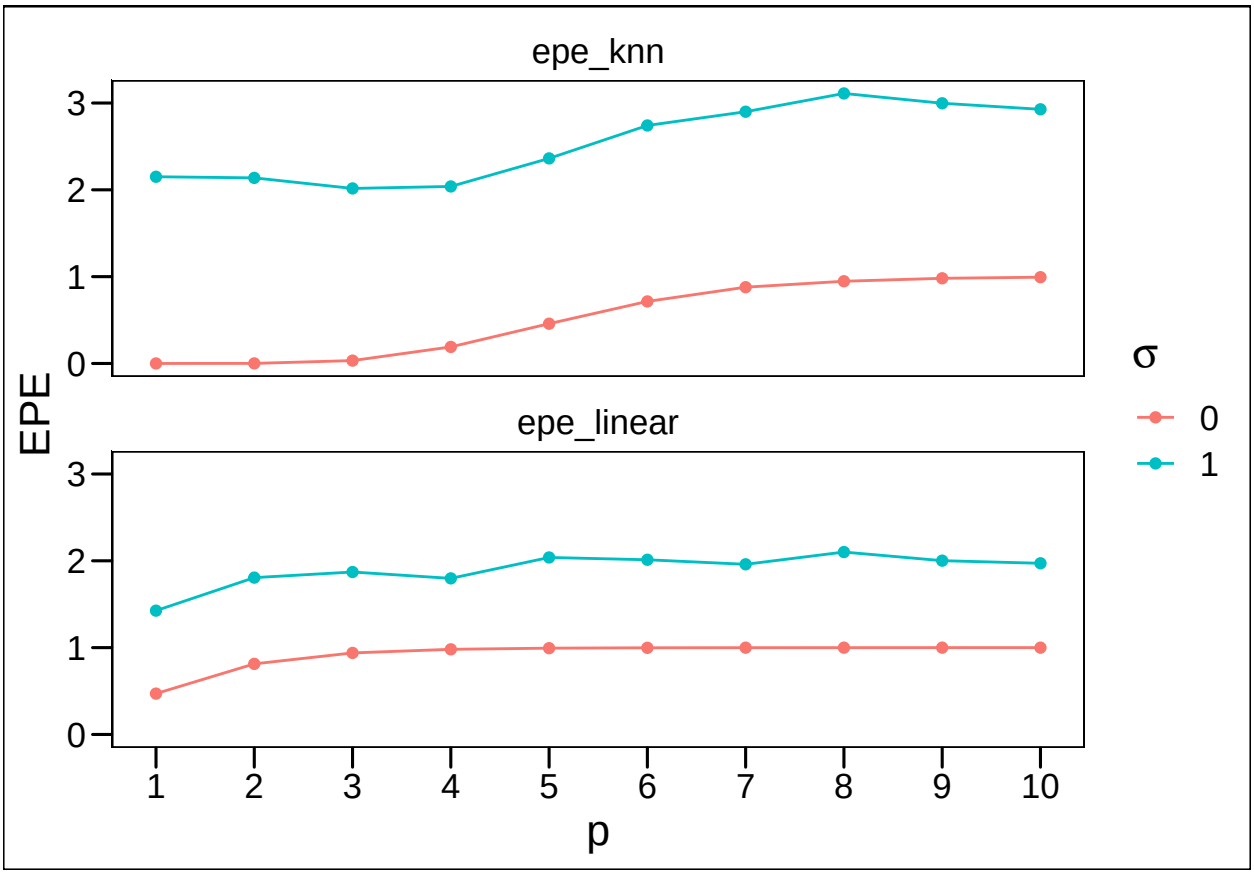
Now that we have the functions we can iterate over p and σ and estimate in each of the 1000 iterations a linear model as well as a $KNN(1)$.

```
run <- F # only change if you want to do the simulation again

if(run == T) {
# get all combinations of p and sigma
grid <- expand.grid(p = 1:10, sigma = 0:1)

grid_split <- with(grid, split(x = grid, f = list(p, sigma)))

# create 1000 datasets per p-sigma combination

results <- lapply(grid_split, function(g) {
  # generate specific data sets
  spec <- vector(mode = "list", length = 1000)
  for(m in 1:1000) {
    spec[[m]] <- get_xy(p = g$p, sigma = g$sigma)
  }

  # evaluate f hat at x_0 = 0
  fx0 <- do.call("rbind", lapply(spec, function(s) get_fx0(data = s, sigma = g$sigma)))

  epe_run <- epe(fx0 = fx0, x_0 = 0, p = g$p, mu = 0, sigma = g$sigma)

  data.frame(p = g$p, sigma = g$sigma, epe_run)
})


results_bind <- do.call("rbind", results)

saveRDS(results_bind, file = "results_task6.rds")
}
```

We can now inspect the results of the simulation.

```
# make a nice plot
library(ggplot2)
library(tidyr)
library(ggthemes)
library(latex2exp)
```

```r
# load the saved results
results_bind <- readRDS(file = "results_task6.rds")

results_bind |>
  pivot_longer(cols = contains("epe")) |>
  ggplot(aes(x = p, y = value, color = as.factor(sigma), group = as.factor(sigma))) +
  geom_point() +
  geom_line() +
  scale_x_continuous(breaks = 1:10) +
  facet_wrap(~ name, nrow = 2) +
  labs(y = "EPE", color = TeX("$\\sigma$")) +
  theme_base()
```



There are two observations here. First, dimensionality increases EPE for both methods. However the linear model experiences the increase earlier but then further increases become smaller while for KNN the increases are smaller but last longer with increasing p. Second, an increase in σ shifts up EPE almost by its magnitude. This illustrates the irreducible error that is introduced by the noise term which is usually part of the models we think of when describing data generating processes.

Task 7

```r
# load data
data("diabetes", package = "lars")
```

```r
# matrices can apparently be columns in df..
# good to know
y <- rnorm(100)
x1 = rnorm(100)
x2 = rnorm(100)
x <- cbind(x1, x2)
df <- data.frame(y = y, x = I(x))

# but this complicates things so we break up the
# structure and make a nice df
# function to get rid of Asis
unAsIs <- function(X) {
    if("AsIs" %in% class(X)) {
        class(X) <- class(X)[-match("AsIs", class(X))]
    }
    X
}

# extract y and x
y <- diabetes$y
x <- unAsIs(diabetes$x)

# make a new df with all the data
diabetes_df <- as.data.frame(cbind(y, x))
```

As instructed, we now set a seed and sample row indices from the set of integers running from 1 to the number of observations with equal probability.

```r
# set seed
set.seed(12)

# split the data into train and test
# step 1: sample 400 indices
ind <- sample(x = 1:nrow(diabetes_df), size = 400)

# subset the datasets as instructed
train <- diabetes_df[ind, ]
test <- diabetes_df[-ind, ]
```

The reason why random sampling is a good idea is that we are not really familiar with the dataset. Specifically we do not know whether observations were sorted by any of the variables available and if so we do not know at all by which one. Just taking the 400 first observations then would lead the information contained in training and test data to be biased by sorting leading ultimately to sampling bias in our estimations.

With respect to the question about standardized data lets start what happens if we use unstandardized data to be able to highlight why it is sensible to use standardized data instead in the subsequent analysis. Lets say we have a regression of the form $y = X\beta + \epsilon$. If X contains unstandardized data, the unit of measurement is likely to be different between variables, for instance age is measured in years but height is measured in cm. As regression coefficients can be interpreted ceteris paribus as the change of y wrt to a change in X_i but dependent on the unit of measurement. Thus it is not possible to disentangle the strength of the effects based on unstandardized regression coefficients. This is especially a problem if we want to perform variable selection, as we want to select those variables with the greatest influence on the response variable. But as we cannot really measure how large the influence (read: effect) of the variable is based on unstandardized regression, we cannot perform variable selection like this in a sensible way.

Table 1: Correlation matrix for y and X

	y	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
y											
age	0.18										
sex	0.07	0.21									
bmi	0.6	0.17	0.09								
map	0.44	0.31	0.26	0.41							
tc	0.22	0.26	0.06	0.25	0.25						
ldl	0.19	0.23	0.16	0.26	0.21	0.9					
hdl	-0.4	-0.09	-0.39	-0.36	-0.21	0.04	-0.2				
tch	0.42	0.22	0.36	0.41	0.29	0.56	0.67	-0.74			
ltg	0.56	0.27	0.16	0.44	0.39	0.52	0.33	-0.4	0.61		
glu	0.37	0.29	0.24	0.38	0.4	0.33	0.28	-0.27	0.41	0.48	

Standardized regression coefficients however represent the change y in terms of standard deviations for a one-standard-deviation change in the corresponding standardized X_i. They allow for direct comparison of the relative importance of different variables and help assess the impact of predictors while accounting for differences in scale and units. Thus standardization solves the problem caused by differences in units of measurement. This ultimately allows sensible variable selection.

To analyse the correlation structures we simply calculate a matrix with correlation of all columns in our dataset. The first column contains the correlations of the variables in X with y and the other columns and rows respectively contain the correlations between the columns in X. In general high absolute values of $corr(X_k, y)$ are desirable because this implies high co- or countermovement of the dependent and independent variables. This at least hints at predictive power of X_k, where k is the column index. Contrary, low values for $corr(X_k, X_j)$, $k \neq j$ are desirable as high values would introduce all the problems associated with multicollinearity, most prominently however the variance of the estimates will become inflated. This means nothing else than a loss in precision of estimates. Another huge problem is that multicollinearity is associated with "almost rank deficient" $X'X$ what can lead to problems if we run our model on a computer system.

```
library(kableExtra)
# exploration of correlation
correlation_matrix <- round(cor(train), 2)
# eliminate redundancies and make a nice table for the pdf
correlation_matrix[!lower.tri(correlation_matrix)] <- ""
kable(correlation_matrix, booktabs = T, caption = "Correlation matrix for y and X")
```

Looking at Table 1 we can clearly identify variables that seem to have explanatory power w.r.t. y, namely *bmi*, *ltg*, *map*, *tch*, *hdl* and *glue*. Looking at the correlation structure of this subset of the columns in X we find that especially those variables exhibit substantial correlation among themselves. In the light of variable selection this causes the familiar problems of multicollinearity and specifically in terms of model selection the problem is, that the effect captured by a regression model with respect to a certain variable depends heavily on the inclusion of variables that are highly correlated with this particular variable. This can destabilize variable selection procedures.

```
# use training data to fit the full model
# get model formula from column names
f <- as.formula(paste0("y~", paste(colnames(train)[-1], collapse = "+")))

fit_full <- lm(data = train, formula = f)
```

```r
# get variables significant at alpha = 0.05
summary_full <- summary(fit_full)
coefficients <- summary_full$coefficients
significant <- which(coefficients[, 4] < 0.05)[-1]

# in sample MSE
MSE_full_in <- mean(fit_full$residuals^2)

# out of sample MSE
pred_full <- predict(fit_full, newdata = test)
MSE_full_out <- mean((test$y - pred_full)^2)

message(paste("In sample MSE is", MSE_full_in, sep = ": "))
```

```
## In sample MSE is: 2876.35849729569
```

```r
message(paste("Out of sample MSE is", MSE_full_out, sep = ": "))
```

```
## Out of sample MSE is: 2809.87939232709
```

```r
# use the significant variables only
f2 <- as.formula(paste0("y~", paste(colnames(train)[significant], collapse = "+")))

# estimate smaller model
fit_sig <- lm(data = train, formula = f2)

# in sample MSE
MSE_sig_in <- mean(fit_sig$residuals^2)

# out of sample MSE
pred_sig <- predict(fit_sig, newdata = test)
MSE_sig_out <- mean((test$y - pred_sig)^2)

message(paste("In sample MSE is", MSE_sig_in, sep = ": "))
```

```
## In sample MSE is: 3030.1546413137
```

```r
message(paste("Out of sample MSE is", MSE_sig_out, sep = ": "))
```

```
## Out of sample MSE is: 3222.23929484251
```

```r
# F-Test
anova(fit_sig, fit_full)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ sex + bmi + map + ltg
## Model 2: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##   Res.Df     RSS Df Sum of Sq      F   Pr(>F)
## 1    395 1212062
```

```
## 2    389 1150543  6     61518 3.4666 0.002383 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We find that the additional variables in the full model yield a statistically significant improvement over the model with just the variables being statistically significant.

Task 8

The Akaike criterion is in general given by

$$AIC = 2K - ln(L)$$

where K is the number of regressors and L is the likelihood of the model. Since $2K$ is a penalty term, AIC is to be minimized to find the most appropriate model.

Best Subset Selection

Best subset selection is basically just getting all 2^P possible combinations of explanatory variables available to us and estimating the corresponding regression models. It would be conventient to use the leaps package for this task but there the AIC-criterion is not implemented but only SIC and BIC. So we have to construct the models ourselves, estimate them and calculate AIC.

First we write a function that calculates all possible combinations of regressors and gives back the associated regression formulas. We will not use the leaps and bounds algorithm as the dataset is not to big.

```r
# write function to get formulas
bs_formulas <- function(x = train, dep = "y", intercept_only = T) {
  # extract variables names
  vars <- names(x)
  exps <- vars[vars != dep]

  # get all combinations
 f <- lapply(1:length(exps), function(k) {
    # get combinations for given k
    combinations <- combn(exps, m = k, simplify = F)
    # make it a regression formula
    formulas <- lapply(combinations, function(c) {
      paste(dep, paste(c, collapse = "+"), sep = "~")
    })
    # make it a vector again
    unlist(formulas)
    })
  # dissolve list again
  output <- unlist(f)

  if(intercept_only == T) output <- c(as.formula(paste0(dep, "~ 1")), output)

  return(output)
}
```

Then we estimate the models

```r
# get formulas
formulas <- bs_formulas() # look at defaults set above

# estimate all models
fits <- lapply(formulas, function(f) lm(data = train, formula = f))

# get log likelihoods
LL <- unlist(lapply(fits, function(f) logLik(f)[1]))

# get number of regressors (K)
K <- unlist(lapply(fits, function(fit) length(fit$coefficients)))

# get AIC
AIC <- 2 * K - 2 * LL

# get the most appropriate model
best_index <- which.min(AIC)

# display it
message(formulas[[best_index]])
```

```
## y~sex+bmi+map+tc+ldl+ltg
```

Now that we have identified the best model we can assess its in- and out-of-sample performance using MSE again.

```r
# get in sample MSE
MSE_BS_in <- mean(fits[[best_index]]$residuals^2)
message(paste("Within sample MSE is for Best Subset Selection:", MSE_BS_in))
```

```
## Within sample MSE is for Best Subset Selection: 2880.81482384062
```

```r
# get out of sample MSE
MSE_BS_out <- mean((test$y - predict(fits[[best_index]], newdata = test))^2)
message(paste("Out of sample MSE is for Best Subset Selection:", MSE_BS_out))
```

```
## Out of sample MSE is for Best Subset Selection: 2901.55718360118
```

Finally we conduct an F-Test of the identified model against the full model.

```r
anova(fits[[best_index]], fits[[length(fits)]]) # last model is the full one by construction
```

```
## Analysis of Variance Table
##
## Model 1: y ~ sex + bmi + map + tc + ldl + ltg
## Model 2: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1    393 1152326
## 2    389 1150543  4    1782.5 0.1507 0.9626
```

Based on the p-value of the F-Test we can conclude that adding the remaining variables to the model selected by best subset selection does not lead to a statically significant improvement.

Backward Stepwise

Here we are lucky because the MASS package provides us with a function that does stepwise regression based on AIC.

```
library(MASS)

# conduct backwards stepwise regression
backwards_step <- stepAIC(fit_full, direction = "backward", trace = F)

length(backwards_step$coefficients)
```

```
## [1] 7
```

```
length(fit_full$coefficients)
```

```
## [1] 11
```

```
# get in sample MSE
MSE_backwards_in <- mean(backwards_step$residuals^2)
message(paste("Within sample MSE is for Backwards Stepwise:", MSE_backwards_in))
```

```
## Within sample MSE is for Backwards Stepwise: 2880.81482384062
```

```
# get out of sample MSE

MSE_backwards_out <- mean((test$y - predict(backwards_step, newdata = test))^2)
message(paste("Out of sample MSE is for Backwards Stepwise:", MSE_backwards_out))
```

```
## Out of sample MSE is for Backwards Stepwise: 2901.55718360118
```

```
anova(backwards_step, fit_full)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ sex + bmi + map + tc + ldl + ltg
## Model 2: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1    393 1152326
## 2    389 1150543  4    1782.5 0.1507 0.9626
```

The result of the F-Test is the same as for best subset selection as the chosen model is the same again. The interpretation thus is the same as for best subset selection.

Finally all results can be found in Table 2.

```
# prepare the
df_table <- data.frame(variable = names(fit_full$coefficients),
                       full_model = fit_full$coefficients)
```

```r
best_subset_df <- data.frame(variable = names(fits[[best_index]]$coefficients),
                             best_subset = fits[[best_index]]$coefficients)

backward_df <- data.frame(variable = names(backwards_step$coefficients),
                             backward = backwards_step$coefficients)

# merge tables
merge1 <- merge(df_table, best_subset_df, by = "variable", all.x = T)
df_table_final <- merge(merge1, backward_df, by = "variable", all.x = T)

# add MSE within and out of sample
MSE <- data.frame(variable = c("In Sample MSE",
                               "Out of Sample MSE"),
                  full_model = c(MSE_full_in,
                                 MSE_full_out),
                  best_subset = c(MSE_BS_in,
                                  MSE_BS_out),
                  backward = c(MSE_backwards_in,
                               MSE_backwards_out))

# bind together
df_table_final_mse <- rbind(df_table_final, MSE)

# do some formatting
df_table_final_mse[, -1] <- round(df_table_final_mse[, -1], 2)
df_table_final_mse[is.na(df_table_final_mse)] <- ""

# create table with kable
kable(df_table_final_mse, booktabs = T, caption = "Results for Different Selection Methods.",
      col.names = c("", "Full model", "Best Subset Selection", "Backwards Stepwise"), digits = 2) |>
  pack_rows("Regression Coefficients", 1, 11, hline_after = T, latex_align = "c") |>
  pack_rows("Performance", 12, 13, hline_after = T, latex_align = "c")
```

We see that Best Subset Selection as well as Backwards Stepwise Selection yield the same model and thus consequently the same MSE within and out of sample. The selected models are much more sparse than the full one. We can also see, that only those variables have been selected by the two methods which have large coefficients in absolute terms. This seems sensible in light of the discussion about standardized regression in Task 7. Interestingly however, both within and out of sample MSE are slightly better for the full model.

Task 9

Task 10

Table 2: Results for Different Selection Methods.

	Full model	Best Subset Selection	Backwards Stepwise
Regression Coefficients			
(Intercept)	153.13	153.15	153.15
age	-2.90		
bmi	548.13	554.38	554.38
glu	41.54		
hdl	14.33		
ldl	443.66	510.75	510.75
ltg	732.82	778.67	778.67
map	296.95	301.06	301.06
sex	-190.13	-180.49	-180.49
tc	-677.58	-715.46	-715.46
tch	66.48		
Performance			
In Sample MSE	2876.36	2880.81	2880.81
Out of Sample MSE	2809.88	2901.56	2901.56