Assignment 3

Group 2

2023-10-22

Task 1

Task 2

We first set up the training data and fix $\hat{f}$.

```
# training data
n = 40
x <- rnorm(n, mean = 0, sd = 1)
x2 <- x^2
noise <- rnorm(n, mean = 0, sd = 1)
y <- x + x2 + noise
df <- data.frame(y = y, x = x, x2 = x2)

# fix f hat
f_hat <- lm(data = df, formula = y ~ .)
```

The test error is given in general as

$$\text{Err}_{\mathcal{T}} = \text{E}[L(Y, \hat{f}(X)) \mid \mathcal{T}]$$

where $mathcalT$ is a particular test data set with data unobserved by the model but drawn from the joint distribution of Y and X and L is a loss function. The squared error loss is given by $(Y - \hat{f}(X))^2$ s.t. we are searching for

$$\text{Err}_{\mathcal{T}} = \text{E}[(Y, \hat{f}(X))^2 \mid \mathcal{T}]$$

To find this quantity we have to integrate over the joint distribution $f(x, y)$ which is given by

$$f(x, y) = f(y \mid x) f(x)$$

which are two quantities we know because $x \sim N(0, \sigma_x^2) \Leftrightarrow f(x) = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{1}{2\sigma_x^2} x^2}$ and $y = x + x^2 + \epsilon \Rightarrow$ E[y | x] $= x + x^2$, $Var[y|x] = \sigma_\epsilon^2$ as x is predetermined and thus constant. As $y|x$ is further just a function of predetermind values and a normal RV it must be normal too, i.e. we have $y|x \sim N(x + x^2, \sigma_\epsilon^2) \Leftrightarrow f(y|x) = \frac{1}{\sigma_\epsilon \sqrt{2\pi}} e^{-\frac{1}{2\sigma_\epsilon^2}(y - x - x^2)^2}$. Thus, we have

$$f(x, y) = f(y|x) f(x) = \frac{1}{\sigma_\epsilon \sqrt{2\pi}} e^{-\frac{1}{2\sigma_\epsilon^2}(y - x - x^2)^2} \cdot \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{1}{2\sigma_x^2} x^2} = \frac{1}{\sigma_\epsilon \sigma_x 2\pi} e^{-\frac{1}{2\sigma_\epsilon^2}\left(y - x - x^2\right)^2 - \frac{1}{2\sigma_x^2} x^2}$$

Further, the squared loss function for this particular problem is given by

$$L(Y, \hat{f}(X)) = (y - \hat{\beta}_0 - \hat{\beta}_1 x - \hat{\beta}_2 x^2)^2$$

such that we finally arrive at

$$\text{Err}_{\mathcal{T}} = \text{E}[(Y, \hat{f}(X))^2 \mid \mathcal{T}] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \frac{1}{\sigma_\epsilon \sigma_x 2\pi} e^{-\frac{1}{2\sigma_\epsilon^2}\left(y-x-x^2\right)^2 - \frac{1}{2\sigma_x^2}x^2} (y - \hat{\beta}_0 - \hat{\beta}_1 x - \hat{\beta}_2 x^2)^2 dxdy$$

Solving this integral by hand does not appear to be very easy so we opt for numerical integration.

```r
# library cubature allows multivariate numerical integration
if(!require(cubature)) install.packages("cubature")
```

```
## Lade nötiges Paket: cubature
```

```r
library(cubature)

sigma_x <- 1
sigma_e <- 1

# setup the integral as a function of two variables
 int <- function(z) {
  # setup the
  x <- z[1]
  y <- z[2]
  # setup the joint joint density
  jdens <- {
    (1 / (2 * pi * sigma_x * sigma_e)) * exp(-x^2 / (2 * sigma_x^2) -
                                               (y - x - x^2)^2 / (2 * sigma_e^2))}

  # calculate swuared error with coefficients from f hat
  sq_loss <- (y - coef(f_hat)[1] - coef(f_hat)[2] * x - coef(f_hat)[3] * x^2)^2
  return(jdens * sq_loss)
 }

# solve the integral
solved <- adaptIntegrate(f = int, lowerLimit = c(-Inf, -Inf), upperLimit = c(Inf, Inf))
```

```r
# Simulation
# make test data set of equal size
x_test <- rnorm(40, mean = 0, sd = 1)
x2_test <- x^2
noise_test <- rnorm(40, mean = 0, sd = 1)
y_test <- x + x2 + noise
df_test <- data.frame(y = y_test, x = x_test, x2 = x2_test)

# make 10000 test data sets of n = 10
test_data <- lapply(1:10000, function(i) {
  n = 10
  x <- rnorm(n, mean = 0, sd = 1)
  x2 <- x^2
  noise <- rnorm(n, mean = 0, sd = 1)
  y <- x + x2 + noise
  data.frame(y = y, x = x, x2 = x2)
```

```
})

# using f hat predict y for the test data and
# calculate the squared error loss
mean_squared_loss <- lapply(test_data, function(tt) {
  pred <- predict(f_hat, tt)
  test_error <- mean((pred - tt$y)^2)
})

# test error of the simulation is the mean over
# the individual test errors
mean(unlist(mean_squared_loss))
```

```
## [1] 1.066954
```

```
# compare to integral
solved$integral
```

```
## [1] 1.046636
```

We see that the two values are very similar meaning that we can approximate the true test error with simulation methoid in a pretty accurate way.

Task 3

Task 4

Task 5

Task 6

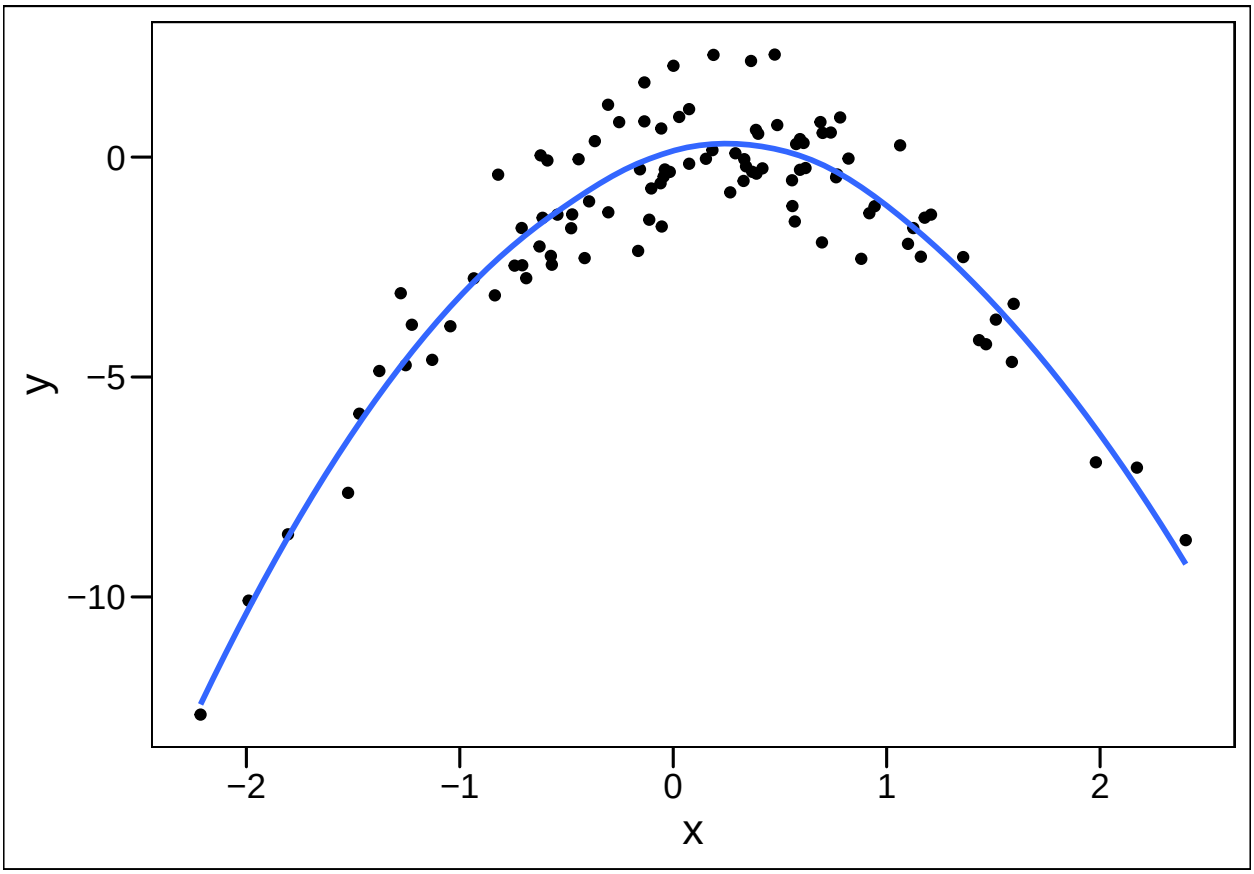
```
# a)
# data
set.seed(1)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

```
# assuming you dont mean orthogonal polynomials we set up a
#data.frame with all the polynomials inside
dat <- data.frame(y = y, x = x, x2 = x^2, x3 = x^3, x4 = x^4)

# b)
# we then create a scatterplot and we dare to use
# ggplots in built loess implementation to draw a curve
# that fits the data as good as possible

dat %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "loess", se = F) +
  theme_base()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Looking at the scatterplot we observe that the data follows an inverted U-shaped pattern that could be expected if you choose $f(x) = x - 2x^2$. Another notable fact is that most of the data points are located around the turning point of the inverted U. However, this behavior of y is also expectable as $x, \epsilon \sim N(0, 1)$ both have most of their mass around 0 and hence the most common value should be around 0 looking on the data generating process.

```r
# fit the models
fits <- lapply(2:5, function(i) {
  lm(data = dat[, 1:i], y ~ .)
})

# calculate AIC stepwise
# A get number of coefficients + 1 (we estimate the variance in addition)
K <- unlist(lapply(fits, function(f) length(f$coef))) + 1

# get the LogLikelihood of the models
LL <- unlist(lapply(fits, logLik))

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

```
## [1] 478.8804 280.1670 282.0886 282.2963
```

```r
# print model summary with for lowest AIC
summary(fits[[which.min(AIC)]])
```

```
##
## Call:
## lm(formula = y ~ ., data = dat[, 1:i])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.9650 -0.6254 -0.1288  0.5803  2.2700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05672    0.11766   0.482    0.631
## x            1.01716    0.10798   9.420  2.4e-15 ***
## x2          -2.11892    0.08477 -24.997  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.958 on 97 degrees of freedom
## Multiple R-squared:  0.873,  Adjusted R-squared:  0.8704
## F-statistic: 333.3 on 2 and 97 DF,  p-value: < 2.2e-16
```

```r
# d)

# get fitted values from models in c
pred_c <- lapply(fits, function(f) f$fitted)

# get residual standard deviations
sigma_c <- lapply(fits, function(f) summary(f)$sigma)

# create 100 test data sets
new_y <- lapply(1:100, function(i) {
  y <- x - 2*x^2 + rnorm(100)
  y
})

# now the assumption is that errors in linear regression are gaussian hence
# the likelihood must be that of a gaussian distribution

# for each of the model specifications calculate the in sample error as the
# -2 log(LL) where log(LL) for each test data set is determined by inserting
# the new y values into the normal density with a mean equal to the fitted value
# of the particular observation and standard deviation equal to the residual
# standard deviation of the training fits. This gives us then the individual
# log likelihood for every newly generated y to come from the assumed data
# generating process. Summing over these then gives the log-likelihood of the
# whole test data set. Then we average over the log likelihoods of the 100 test
# data sets and multiply with -2.

Err <- rep(NA, 4)
for(i in 1:length(pred_c)) {
LL <- lapply(new_y, function(ny) {
```

```
  sum(log(dnorm(x = ny, mean = pred_c[[i]], sd = sigma_c[[i]])))
})
Err[i] <- -2 * mean(unlist(LL))
}

# print results
Err
```

```
## [1] 465.4097 288.7366 288.7320 290.5469
```

Looking at the in sample errors for the 4 models we calculated using this particular loss function we see that they are quite similar to the AIC values from task b). This is not very surprising as also the AIC builds on the $-2 * LL$ if you remember the formula to calculate AIC.

Task 7

Task 8

Task 9

Task 10