Machine Learning

Assignment 1

Hypothesis, Target and Candidate Sets and Overfit Measuring.

Julian Albert
ALBJUL005

Department of Statistical Sciences
University of Cape Town
South Africa

September 19, 2019

Contents

Introduction

In this assignment we will consider various datasets and target functions to assess bias-variance and overfit measures. To do this we must first define some terminology, first we define our dataset of size N to be $\mathcal{D}_N = \{(x_1, y_1),\ \ldots,\ (x_N, y_N)\}$. Next we can define some target function $f : \mathcal{X} \to \mathcal{Y}$ which maps our inputs in domain $\mathcal{X}$ to our output domain $\mathcal{Y}$. The target function represents the ideal model (true pattern) which is unknown to us. If the target function was known no learning would be required. So how do we learn the target function? We use historical (training) data.

We can define a hypothesis set $\mathcal{H}$ which contains $g : \mathcal{X} \to \mathcal{Y}$ where we want $g \approx f$. In other words we create — and thus know — g and we hope it accurately approximates the target function f.

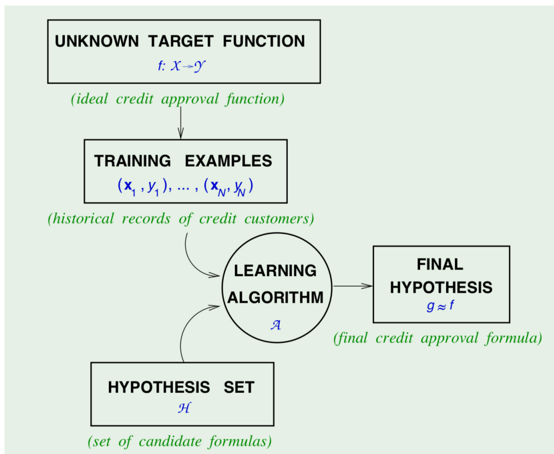

Figure 1: Learning Algorithm Diagram from [1].

From Figure (1) we can see that we have some unknown target function, as we do not know this function we only "observe" it through our dataset $\mathcal{D}$. We want to produce $g : \mathcal{X} \to \mathcal{Y}$ our final hypothesis for $g \in \mathcal{H}$ s.t $g \approx f$. To do this we need a learning algorithm $\mathcal{A}$. The learning algorithm selects the formula $g \in \mathcal{H}$ where $\mathcal{H}$ is our hypothesis set (set of candidate formulae). Selecting from $\mathcal{H}$ is important as it helps define if we can and how we learn. The learning algorithm and hypothesis set define the *learning model*, important here is the realisation that when approaching learning problems the only part over which we have control is the learning model.

Question 1

In this Question we consider two types of polynomial target functions,

$$f(x) = \sum_{q=0}^{Q_f} \alpha_q x^q \tag{1}$$

$$= \sum_{q=0}^{Q_f} \beta_q L_q(x) \tag{2}$$

where $L_q(x)$ is a Legendre polynomial of order q give by $L_q(x) = 2^q \sum_{k=0}^{q} x^k \binom{q}{k} \binom{\frac{q+k-1}{2}}{q}$

i) Legendre

Here we will plot $L_q(x)$ over $[-1;1]$ for $q = 0,\ 1,\ \ldots,\ 5$. Figure (2) shows this plot, we can see that when $q = 0$ we have a flat line, as q increases we get increasing orders of the polynomial for $L_q(x)$.

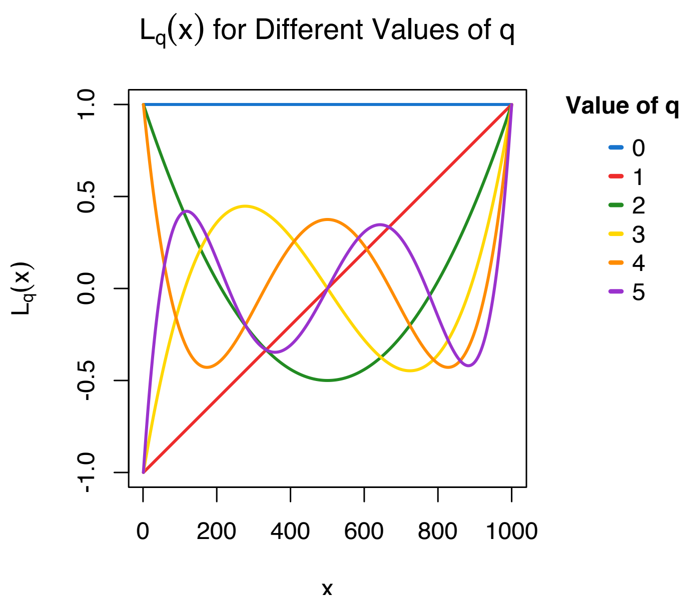

Figure 2: Legendre Polynomial Plotted for Different Values of q.

ii) Polynomial vs Legendre

Using Equations (1) and (2) we can generate target functions $f(x)$, this is done by sampling α_q and β_q from a Uniform distribution over $[-1;1]$. We can then use the two equations for different values of q, below I consider $q = 2,\ 4,\ 10$. Figure (3) shows three target functions for the polynomial and Legendre polynomial — represented by Equations (1) and (2) respectively.

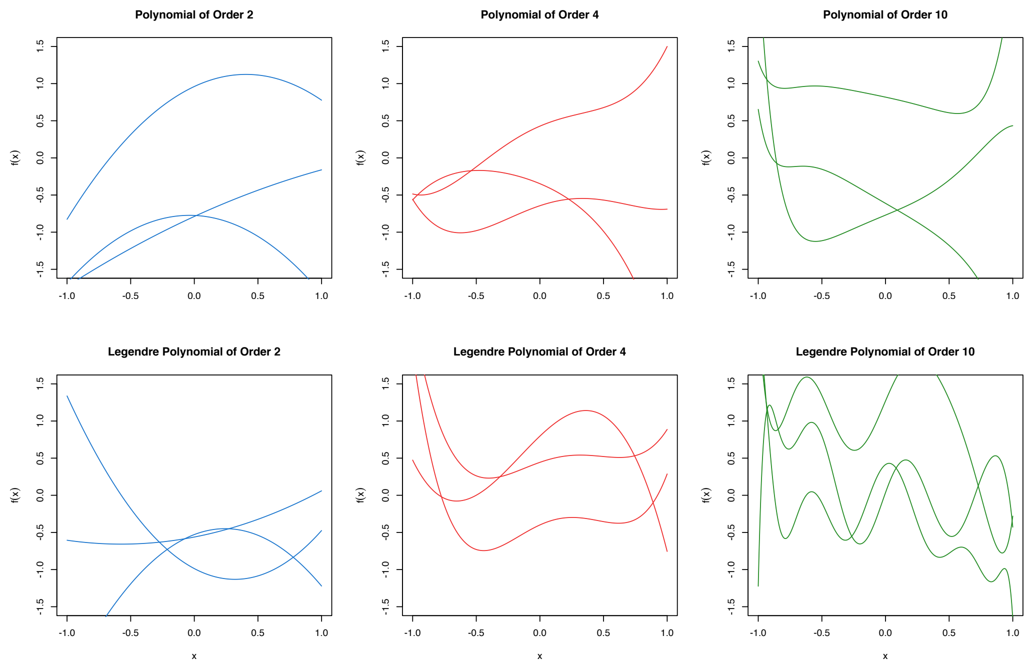

Figure 3: Polynomials using Equation (1) and Legendre Polynomials using Equation (2) Plotted for Different Values of q.

From Figure (3) we can see that for the polynomials (top row) the target functions do not seem to become overly “wiggly” for orders of $q > 3$. In contrast to the Legendre polynomials (bottom row) where the target functions increase in wiggliness with increasing orders of q. From this we can determine that a limitation of the standard polynomials is that the target functions appear bounded by the third order of q whereas the Legendre polynomials flexibility is unbounded. This is further supported by Figure (4)

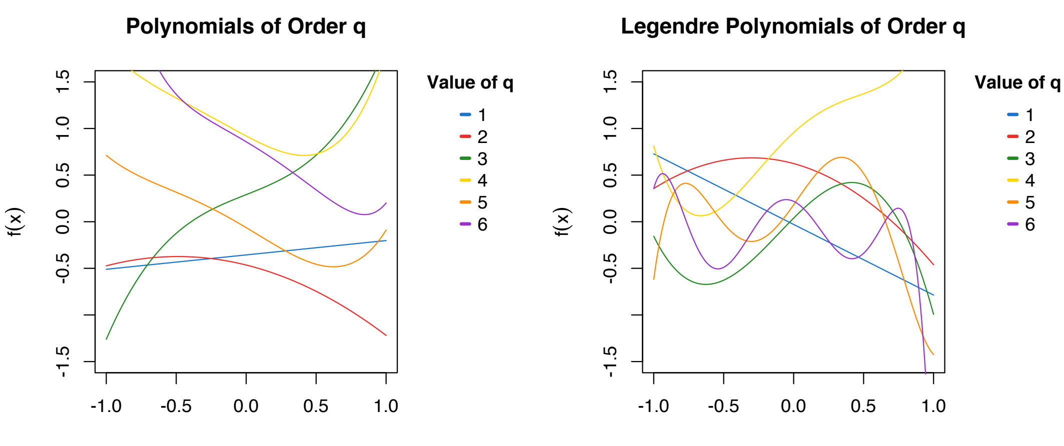

Figure 4: Target Functions from Polynomials using Equation (1) and Legendre Polynomials using Equation (2) Plotted for Different Values of q.

Question 2

Here we are going to consider $\mathcal{D}$ and Legendre-based target function f of order Q_f with noise level σ. We assume that $x_i \sim U(-1, 1)$. We will explore two candidate hypotheses

$$\mathcal{H}_2 : 2^{nd} \text{ order polynomials}$$
$$\mathcal{H}_{10} : 10^{th} \text{ order polynomials}$$

This Question is concerned with how the level of noise (σ), the order of the target function (Q_f) and the size of the dataset (N) relate to over-fitting. We compare the final hypothesis $g_{10} \in \mathcal{H}_{10}$ to the final hypothesis $g_2 \in \mathcal{H}_2$. We would expect $E_{in} g_{10} \leq E_{in} g_2$ as g_{10} has more degrees of freedom to fit the data.

We will first generate a random 10^{th} order target function over $-1 \leq x \leq 1$ using Equation (2). Next we will generate M datasets $\mathcal{D}_N = \{(x_1, y_1),\ \ldots,\ (x_N, y_N)\}$ where $20 \leq N \leq 110$. For each dataset $\mathcal{D}_N$ we will have P different noise levels $0.2 \leq \sigma < 1.1$. For each N and σ combination we will calculate an overfit measure defined as

$$\mathbb{E}_{\mathcal{D}} \left[E_{\text{out}} \left(g_{10}^{\mathcal{D}_N^{\sigma}} \right) - E_{\text{out}} \left(g_2^{\mathcal{D}_N^{\sigma}} \right) \right] \tag{3}$$

where $g_q^{\mathcal{D}}(x) \in \mathcal{H}_q$ represents a fitted model. The more positive this measure is, the more severe over fitting would be. The result will be an $P \times M$ matrix which can be plotted as a colour map. Figure (5) shows the generation of a target function $L_{10}(x)$, $\mathcal{D}_{15}$ and the resulting 2^{nd} and 10^{th} order polynomial fits with in-sample error, out-of-sample error and overfit measures calculated. Here we see that despite $L_{10}(x)$ remaining the same throughout the data generated and the resulting fits for $N = 15$, $\sigma = 0.5$ vary. This is what leads us to generate data many times for each N and σ and take an average overfit measure over the data generations — corresponding to the $\mathcal{D}$ subscript on the expectation in Equation (3). We also note that where the overfit measure is unconstrained we see very high values, this is largely due to the high-order polynomial going to extreme values in order to try fit the data. When we leave the problem unconstrained the colour map is not interpretable due to the scale being distorted. As a result we set a bound δ on the overfit measure.

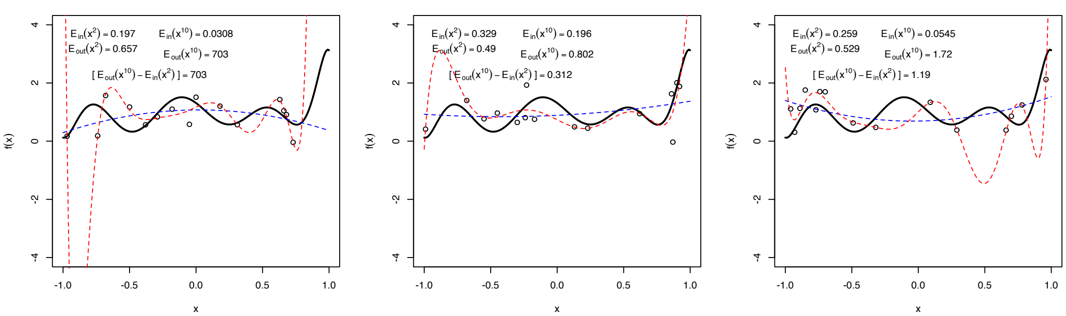

Figure 5: Random Legendre-based Target Function $L_{10}(x)$ with Linear Model Fit of 2^{nd} (Blue) and 10^{th} (Red) Order Polynomials.

i and ii) Colour Maps

Figure (6) illustrates how the level of noise (σ), the order of the target function (Q_f) and the size of the dataset (N) relate to over-fitting.

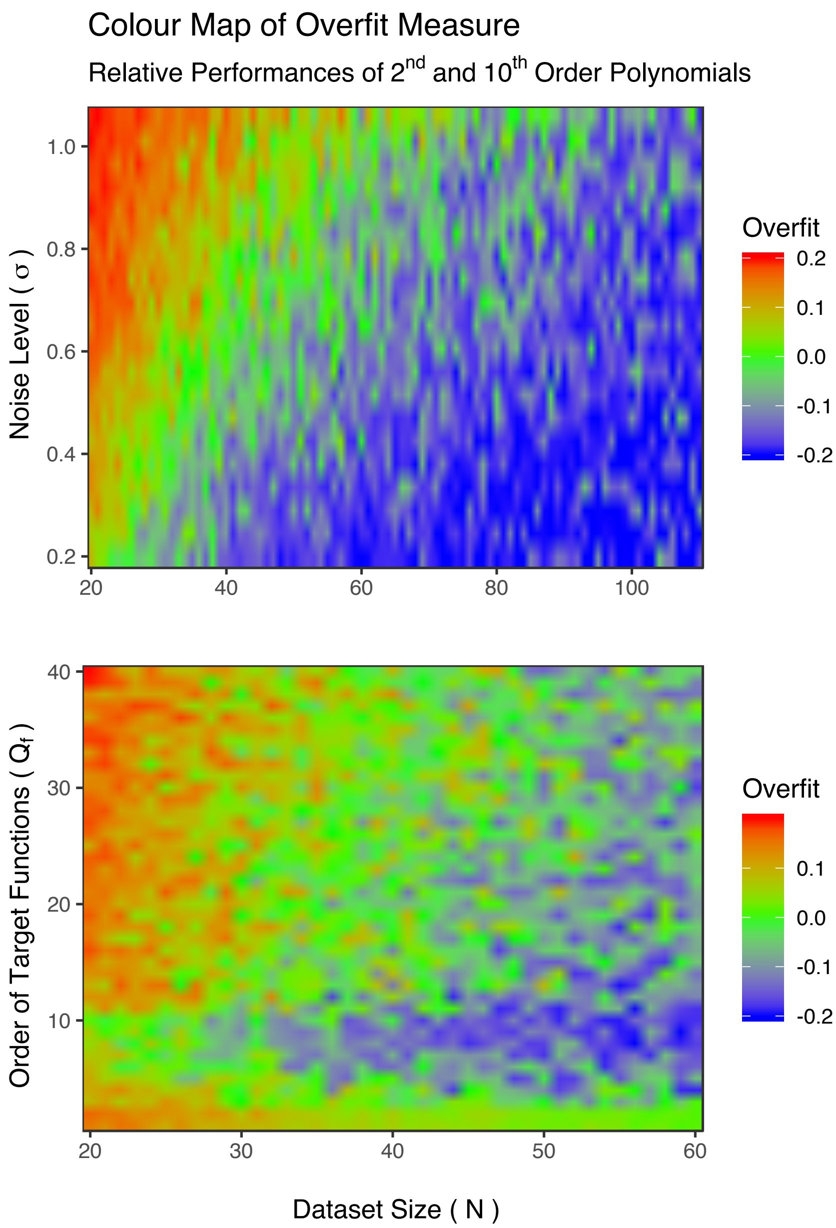

Figure 6: Colour Map of Overfit Measure as Defined by Equation (3) for Different Values of N and σ, Q_f.

iii) Discussing the Colour Maps

The colour map as depicted in the top row of Figure (6) shows the impact of noise level and dataset size for fixed $Q_f = 10$. The result is somewhat expected, for increased dataset size we see that our overfit measure is relatively smaller. With increased noise level we observe a higher overfit measure. Noise distorts the learning with the more complex model being more susceptible to noise than the simpler model. The bottom row of Figure (6) reveals that target function complexity Q_f affects overfitting in a similar way to noise [1]. We can deduce that for the case of σ vs N we observe a more linear relationship, however, for Q_f vs N we observe a more non-linear relationship. This is supported by Figure (4.3) in [1]. Note here that the relationships are not necessarily as strong as those found in the textbook, however, a key observation is that the pattern does appear to be similar and tending towards the textbook solution. Further improvements could be made in the form of data pre-processing or scaling the target functions coefficients such that $\mathbb{E}[f^2] = 1$. This scaling would likely resolve the shape discrepancies.

- END -

References

[1] Abu-Mostafa, Y. S. *Learning from data*, vol. 4.

Appendix

R Code

```r
##
#
# Author: Julian Albert
# Date: 09 September 2019
#
# Description:
# Machine Learning Assignment 1 consists of two problems,
# the first is to do target functions and coefficient approximation using order
# polynomials. The second considers candidate hypothesis and explores problems
# with overfitting.
#
#------------------------------------------------------------------------------#

# 0. Clean Workspace, Directory and Library ----

## Clean Workspace
rm(list=ls())
dev.off() # Close Plots
setwd("~") # Clear Path to User

## Locations
project_folder <- "/Documents/UCT/Coursework/MachineLearning"
loc_script <- "/Assignment_1/UCT_Assignment/Code/MachineLearning_Ass1"
loc_figs <- "/Assignment_1/UCT_Assignment/Figs"

## Directories
dir_script <- paste("~", project_folder, loc_script, sep = '')
dir_figs <- paste("~", project_folder, loc_figs, sep = '')

## Set Working Directory to Script Location
setwd(dir_script)

## Libraries - Lazyload
if (!require("pacman")) install.packages("pacman")
p_load(tidyverse, data.table, Cairo, ggplot2, viridis)

# 1. Problem 1 ----

func.tmp_of_k <- function(k, x, q) # k in q, x is data, q is order of polynomial
{
  x^k * choose(q, k) * choose( (q+k-1)/2, q)
}

func.legendre_p1 <- function(q, x)
{

  k <- as.matrix(0:q, ncol = 1) # to have dimension qx1

  tmp.rhs <- apply(k, 1, func.tmp_of_k, x, q) # to each k
  Lq <- (2^q) * rowSums(tmp.rhs) # Lq(x)

  return(Lq)
}

nx <- 1000 # smooth polynomials require lots of x's
nk <- 5 # assignment wants q = 0, 1, ..., 5

x <- seq(-1, 1, length.out = nx)
q <- as.matrix(seq(0, nk, by = 1), ncol = 1)
```

```r

res.Lq <- apply(q, 1, func.legendre_p1, x) # to each q get Lq(x)
# round(res.Lq, 3)
colour_vec <- c("dodgerblue3", "firebrick2", "forestgreen", "gold",
                "darkorange", "darkorchid3") # colour for pretty plot

# i) plot Lq over -1, 1 for different value of q

setwd(dir_figs)
cairo_pdf("ML_Ass1_fig_Lq_for_diff_q.pdf", width = 5, height = 5)
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
plot(res.Lq[, 1], ylim = c(-1, 1), xlab = "x", ylab = expression(L[q](x)),
     main = expression(paste(L[q](x), " for Different Values of q")),
     type = "l", col = colour_vec[1], lwd = 2) # titles
for(i in 2:dim(res.Lq)[2]){lines(res.Lq[, i], col = colour_vec[i], lwd = 2)}
legend("topright", title = expression(paste(bold("Value of q"))),
       inset = c(-0.4, 0), legend = c("0", "1", "2", "3", "4", "5"),
       col = colour_vec, lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       y.intersp = 1, x.intersp = 0.5, seg.len = 0.5) # legend stuff
dev.off() # turn-off plot

# ii) plot using two equations. Polynomial vs Legendre, 3 targets
x <- seq(-1, 1, length.out = 1000)

## creates random polynomial target functions > Using equation 2
func.poly_target <- function(x, nk)
{
  alpha_vec <- runif(nk + 1, -1, 1) # represent q = 0, 1, ..., nk
  target_f <- 0

  for(i in 1:(nk + 1)){
    target_f <- target_f + alpha_vec[i] * x^(i-1)
  }

  return(target_f)

}

cairo_pdf("ML_Ass1_fig_Polynomials.pdf", width = 12, height = 4)
layout(matrix(1:3, 1, 3, byrow = TRUE), respect = TRUE)
plot(x, 1.5*x, type = "n", ylab = expression(f(x)), main = "Polynomial of Order 2")
for(i in 1:3) {lines(x, func.poly_target(x, 2), col = colour_vec[1])}
plot(x, 1.5*x, type = "n", ylab = expression(f(x)), main = "Polynomial of Order 4")
for(i in 1:3) {lines(x, func.poly_target(x, 4), col = colour_vec[2])}
plot(x, 1.5*x, type = "n", ylab = expression(f(x)), main = "Polynomial of Order 10")
for(i in 1:3) {lines(x, func.poly_target(x, 10), col = colour_vec[3])}
dev.off()

cairo_pdf("ML_Ass1_fig_Poly_for_diff_q_wx.pdf", width = 5, height = 5)
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
plot(x, 1.5*x, type = "n", ylab = expression(f(x)),
     main = "Polynomials of Order q")
for(i in 1:length(colour_vec)){lines(x, func.poly_target(x, i), col = colour_vec[i])}
legend("topright", title = expression(paste(bold("Value of q"))),
       inset = c(-0.4, 0), legend = c("1", "2", "3", "4", "5", "6"),
       col = colour_vec, lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       y.intersp = 1, x.intersp = 0.5, seg.len = 0.5) # legend stuff
dev.off() # turn-off plot

# creates random Legendre target functions > Using equation 3
func.legendre_target <- function(x, nk)
{
  beta_vec <- runif(nk + 1, -1, 1) # represent q = 0, 1, ..., nk
  target_f <- 0

  for(i in 1:(nk + 1)){
    target_f <- target_f + beta_vec[i] * func.legendre_p1((i-1), x)
  }

  return(target_f)

}

```

```r
cairo_pdf("ML_Ass1_fig_LegendrePolynomials.pdf", width = 12, height = 4)
layout(matrix(1:3, 1, 3, byrow = TRUE), respect = TRUE)
plot(x, 1.5*x, type = "n", ylab = expression(f(x)), main = "Legendre Polynomial of Order 2")
for(i in 1:3) {lines(x, func.legendre_target(x, 2), col = colour_vec[1])}
plot(x, 1.5*x, type = "n", ylab = expression(f(x)), main = "Legendre Polynomial of Order 4")
for(i in 1:3) {lines(x, func.legendre_target(x, 4), col = colour_vec[2])}
plot(x, 1.5*x, type = "n", ylab = expression(f(x)), main = "Legendre Polynomial of Order 10")
for(i in 1:3) {lines(x, func.legendre_target(x, 10), col = colour_vec[3])}
dev.off() # turn-off plot

cairo_pdf("ML_Ass1_fig_LegPoly_for_diff_q_wx.pdf", width = 5, height = 5)
par(mar = par()$mar + c(0,0,0,5), pty = 's') # larger margins so legend on side
plot(x, 1.5*x, type = "n", ylab = expression(f(x)),
     main = "Legendre Polynomials of Order q")
for(i in 1:length(colour_vec)){lines(x, func.legendre_target(x, i), col = colour_vec[i])}
legend("topright", title = expression(paste(bold("Value of q"))),
       inset = c(-0.4, 0), legend = c("1", "2", "3", "4", "5", "6"),
       col = colour_vec, lwd = 3, cex = 1, xpd = TRUE, bty = "n",
       y.intersp = 1, x.intersp = 0.5, seg.len = 0.5) # legend stuff
dev.off() # turn-off plot

setwd(dir_script)

# 2. Problem 2 ----

## Generate 10-th order target using legendre over -1, 1
generator <- function(n, # Dataset Size
                      x, # X values for the Data
                      targetfunction,
                      sigma # the sd of noise
                      )
{

  l <- length(targetfunction) # how many data points there are in the target
  dat <- matrix(rep(NA, 2*n), ncol = n) # N columns of data, 2 rows for X and Y
  xdat_indices <- sample(1:l, n) # get n indices for data
  ydat <- targetfunction[xdat_indices] + rnorm(n, 0, sigma) # y =fx + e
  xdat <- x[xdat_indices] # x = x[indexed]
  Data <- data.frame(xdat, ydat)

  return(Data)

}

#fitted model of the data, in this case lm give B0 + B1X1 + B2X^2
func.fitted_model <- function(x, model)
{

  fitted_model <- 0
  coeff_vec <- as.numeric(model$coefficient)

  for(i in 1:length(model$coefficient)){
    fitted_model <- fitted_model + (coeff_vec[i]*(x^(i-1)))
  }

  return(fitted_model)
}

# gives the bias for a given fitted model >> (g-f)^2 / N
func.fit_target_bias <- function(x, target, model)
{

  fitted_model <- func.fitted_model(x, model)
  bias <- (t(fitted_model - target) %*% (fitted_model - target))/(length(x))

  return(bias)
}

# Test Case
x_dat <- seq(-1, 1, 0.01)
sigma <- 0.5
n <- 15
```

```r
targetfunction <- func.legendre_target(x_dat, 10)

setwd(dir_figs)

cairo_pdf("ML_Ass1_fig_Prob2i_fits.pdf", width = 12, height = 4)
layout(matrix(1:3, 1, 3, byrow = TRUE), respect = TRUE)
for(i in 1:3){
plot(c(-1, 1), c(-4, 4), main = "",
     type = "n", xlab = "x", ylab=expression(f(x)))
lines(x_dat, targetfunction, type = "l", lwd = 2) # target function
data_gen <- generator(n, x_dat, targetfunction, sigma) # generated data
points(data_gen$xdat, data_gen$ydat) # plot points

model_2 <- lm(data_gen$ydat ~ data_gen$xdat + I(data_gen$xdat^2)) #quadratic fit
# IS and OOS erros
err.in_2 <- (t(model_2$residuals)%*%model_2$residuals)/n
err.out_2 <- func.fit_target_bias(x_dat, targetfunction, model_2) + (sigma^2)
lines(x_dat, func.fitted_model(x_dat, model_2),
      lty = 2, col = "blue", lwd = 1) # plot lines for fitted quadratic
# Labels for IS, OOS
text1 = bquote(italic(E)["in"](x^2) == .(format(err.in_2, digits = 3)))
text2 = bquote(italic(E)["out"](x^2) == .(format(err.out_2, digits = 3)))
text(x=-0.7,y=3.7,labels=text1)
text(x=-0.7,y=3.2,labels=text2)

model_10 <- lm(data_gen$ydat ~ data_gen$xdat + I(data_gen$xdat^2) +
                 I(data_gen$xdat^3) + I(data_gen$xdat^4) +I(data_gen$xdat^5) +
                 I(data_gen$xdat^6) + I(data_gen$xdat^7) + I(data_gen$xdat^8) +
                   I(data_gen$xdat^9) + I(data_gen$xdat^10)) # 10-th order
# IS and OOS erros
err.in_10 <- (t(model_10$residuals)%*%model_10$residuals)/n
err.out_10 <- func.fit_target_bias(x_dat, targetfunction, model_10) + (sigma^2)
lines(x_dat, func.fitted_model(x_dat, model_10),
      lty = 2, col = "red", lwd = 1) # plot lines for fitted 10-th order
# Labels for IS, OOS
text3 = bquote(italic(E)["in"](x^10) == .(format(err.in_10, digits = 3)))
text4 = bquote(italic(E)["out"](x^10) == .(format(err.out_10, digits = 3)))
text(x=0,y=3.7,labels=text3)
text(x=0,y=3,labels=text4)
# Overfit Measure and Labels for it
overfit_measure <- err.out_10 - err.out_2
text5 = bquote("["~italic(E)["out"](x^10) - italic(E)["in"](x^2)~"]" == .(format(overfit_measure, digits = 3)))
text(x=-0.35,y=2.3,labels=text5)
}
dev.off()

setwd(dir_script)

## Need a function that takes in sigma and n to make colour map
func.overfit_measure <- function(sigma, n, k, order_q, delta)
{

  overfit <- numeric() # initialise
  targetfunction <- func.legendre_target(x_dat, order_q) # same target

  # Generate k times to take average
  for(i in 1:k){

  data_gen <- generator(n, x_dat, targetfunction, sigma) # generate data

  # fit 2nd order >> get IS, OOS
  model_2 <- lm(data_gen$ydat ~ data_gen$xdat + I(data_gen$xdat^2))
  err.in_2 <- (t(model_2$residuals)%*%model_2$residuals)/n
  err.out_2 <- func.fit_target_bias(x_dat, targetfunction, model_2) + (sigma^2)

  # fit 10-th order >> get IS, OOS
  model_10 <- lm(data_gen$ydat ~ data_gen$xdat + I(data_gen$xdat^2) +
                   I(data_gen$xdat^3) + I(data_gen$xdat^4) +I(data_gen$xdat^5) +
                   I(data_gen$xdat^6) + I(data_gen$xdat^7) + I(data_gen$xdat^8) +
                   I(data_gen$xdat^9) + I(data_gen$xdat^10))
  err.in_10 <- (t(model_10$residuals)%*%model_10$residuals)/n # In sample error for the quadratic model
```

```
  err.out_10 <- func.fit_target_bias(x_dat, targetfunction, model_10) + (sigma^2) # Out of sample error

  overfit[i] <- err.out_10 - err.out_2   # store overit measure
  }

  overfit <- ifelse(overfit > delta, delta, overfit) # error threshold
  overfit <- ifelse(overfit < -delta, -delta, overfit) # error threshold
  overfit_measure <- mean(overfit)

  return(list(E2in = err.in_2, E2out = err.out_2,
              E10in = err.in_10, E10out = err.out_10,
              Overfit = overfit_measure))
}

x_dat <- seq(-1, 1, 0.01)
sigma_vector <- seq(0.2, 1.1, length.out = 21)[-21]
N_vector <- seq(20, 110, by = 1)
grid <- expand.grid(N_vector, sigma_vector)

dat_prob2i <- apply(grid, 1, function(x){
  func.overfit_measure(sigma = x[2], n = x[1],
                       k = 50, order_q = 10, delta = 0.2)
})

dat_prob2i <- bind_rows(dat_prob2i)
grid$Overfit <- dat_prob2i$Overfit

setwd(dir_figs)
cairo_pdf("ML_Ass1_fig_Prob2i_colourmap.pdf", width = 5, height = 5)
ggplot(data = grid, aes(x = Var1, y = Var2, fill = Overfit)) +
  geom_raster(interpolate = TRUE) +
  scale_fill_gradient2(low = "blue", mid = "green", high = "red") +
  theme_bw() +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
  labs(x = bquote("\n Dataset Size ( N )"),
       y = bquote("Noise Level ("~sigma~")"~"\n"),
       title = "Colour Map of Overfit Measure",
       subtitle = bquote("Relative Performances of" ~ 2^"nd" ~ "and" ~ 10^"th" ~ "Order Polynomials")) +
  theme(aspect.ratio = 0.75)
dev.off()

## Problem 2ii

x_dat <- seq(-1, 1, 0.01)
Qf_vector <- seq(1, 40, by = 1)
N_vector <- seq(20, 60, by = 1)
grid_ii <- expand.grid(N_vector, Qf_vector)

dat_prob2ii <- apply(grid_ii, 1, function(x){
  func.overfit_measure(sigma = 0.2, n = x[1],
                       k = 50, order_q = x[2], delta = 0.2)
})

dat_prob2ii <- bind_rows(dat_prob2ii)
grid_ii$Overfit <- dat_prob2ii$Overfit

cairo_pdf("ML_Ass1_fig_Prob2ii_colourmap.pdf", width = 5, height = 5)
ggplot(data = grid_ii, aes(x = Var1, y = Var2, fill = Overfit)) +
  geom_raster(interpolate = TRUE) +
  scale_fill_gradient2(low = "blue", mid = "green", high = "red") +
  theme_bw() +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
  labs(x = bquote("\n Dataset Size ( N )"),
       y = bquote("Order of Target Functions ("~"Q"[f]~")"~"\n"), title = "") +
  theme(aspect.ratio = 0.75)
dev.off()

setwd(dir_script)
```

Plagiarism Declaration Form

A copy of this form, completed and signed, to be attached to all coursework submissions to the Statistical Sciences Department.

COURSE CODE:	**STA5068Z**
COURSE NAME:	**Machine Learning**
STUDENT NAME:	**Julian Albert**
STUDENT NUMBER:	**ALBJUL005**
GROUP NUMBER:	**1**

PLAGIARISM DECLARATION

- I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
- I have used a generally accepted citation and referencing style. Each contribution to, and quotation in, this tutorial/report/project from the work(s) of other people has been attributed, and has been cited and referenced.
- This tutorial/report/project is my own work.
- I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.
- I acknowledge that copying someone else's assignment or essay, or part of it, is wrong, and declare that this is my own work.
- Agreement to this statement does not exonerate me from the University's plagiarism rules.

Signature: Date: September 19, 2019