University of Cape Town

STA5068Z

Machine Learning

Assignment 1

Author:
Raisa Salie

Student Number:
SLXRAI001

October 19, 2020

Contents

1 Simulating Target Functions

We assume we have some target function $f(x)$, a polynomial of order Q_f. The target function can be represented in the following ways

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

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

Where L_q is a Legendre polynomial represented by

$$L_q(x) = 2^q \sum_{k=0}^{q} x^k \binom{q}{k} \binom{\frac{q+k-1}{2}}{q}. \tag{3}$$

1.i Legendre Polynomials

We use the representation in (3) to generate Legendre polynomials over $[-1, 1]$ (see Figure 1), with reference to the `Legendre.R` script provided. Legendre polynomials are either even ($L_q(-x) = L_q(x)$) or odd ($L_q(-x) = -L_q(x)$) (George et al. 2005). Among the polynomials plotted those which are even have even q, and those which are odd have odd q. This is consistent with the property that $L_q(-x) = (-1)^q L_q(x)$ (George et al. 2005), which shows that this is indeed true for all Legendre polynomials. Although not immediately evident from the Figure 1 itself, we also have the property that $\int_{-1}^{1} L_q(x)dx = 0$ for $q \geq 1$, and $\int_{-1}^{1} L_i(x)L_j(x)dx = 0$ for $i \neq j$.

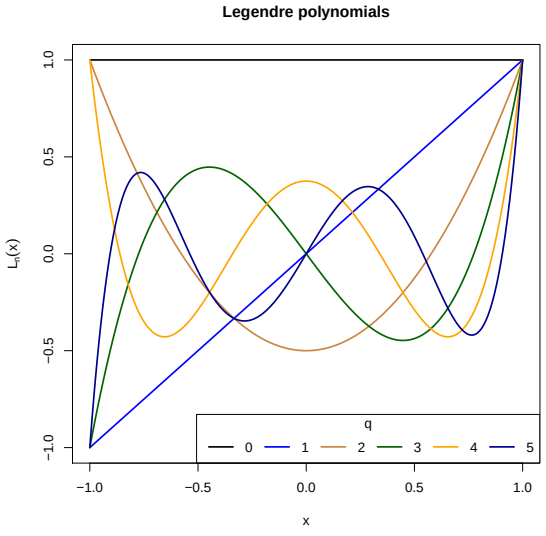

Figure 1: Legendre polynomials of order q

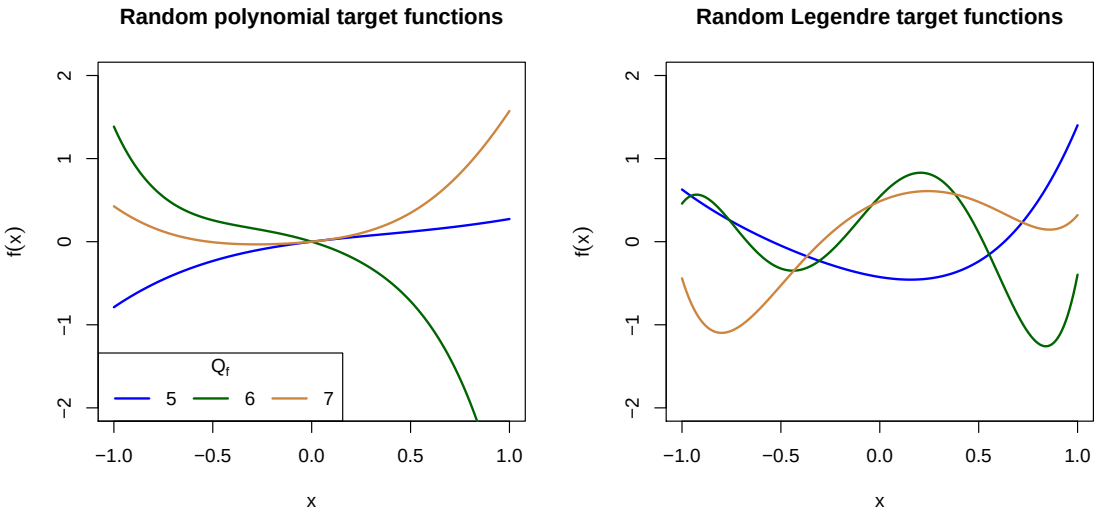

Figure 2: Comparison of target function representations (1) and (2) using three random simulations of different order

1.ii Comparing Random Polynomials to Random Legendre Polynomials

We generate random target functions using the representations in (1) and (2) by drawing each β_q and α_q from a uniform distribution on $[-1, 1]$. The target functions are plotted in Figure 2. The polynomials arising from representation (1) are monotonic over $[-1, 1]$. They are increasing (for odd Q_f) or decreasing (for even Q_f) over $[-1, 1]$. In contrast, the Legendre target functions (2) are not monotonic, and have complex patterns because of their orthogonality over the domain. This makes them more interesting subjects for our analyis.

2 Comparing Models of Different Complexities

Suppose we have data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ arising from the Legendre target function expressed in (2), with order Q_f and noise level σ. We posit the following hypothesis sets.

$$\begin{aligned} \mathcal{H}_2 &: \quad \text{2nd-order polynomials} \\ \mathcal{H}_{10} &: \quad \text{10th-order polynomials} \end{aligned}$$

We will denote the models arising from these hypotheses as $g_2^{\mathcal{D}}(x) \in \mathcal{H}_2$ and $g_{10}^{\mathcal{D}}(x) \in \mathcal{H}_{10}$.

2.i Overfit Measure (varying σ, N)

Here we attempt to understand the impact of noise on the overfitting measure at different N. Our goal is to calculate a measure of overfitting for each (σ, N) configuration based on $E_{\mathcal{D}}\left[E_{out}\left(g_{10}^{\mathcal{D}}\right) - E_{out}\left(g_2^{\mathcal{D}}\right)\right]$. We use the values $N \in \{20, \ldots, 110\}$, and $0.2 \leq \sigma \leq 1.1$ in increments of 0.01. We do this by executing the following steps, with reference to the code provided in `Overfit.R`.

1. Generate a random 10^{th} order target function

$$f(x) = \sum_{q=0}^{10} \beta_q L_q(x),$$

with β_q drawn from $U(-1, 1)$.

2. For each $i \in \{1, \ldots, 100\}$
   (a) For each $N \in \{20, \ldots, 110\}$
      i. For each $\sigma \in \{0.2, 0.21, \ldots, 1.09, 1.1\}$
         A. Simulate data ($\mathcal{D}$ of size N) from a process having target function $f(x)$, with noise $\epsilon(x) \sim N(0, \sigma)$.
         B. Fit a 2^{nd} order polynomial to $\mathcal{D}$, $g_2^{\mathcal{D}}(x)$.
         C. Fit a 10^{th} order polynomial to $\mathcal{D}$, $g_{10}^{\mathcal{D}}(x)$.
         D. Calculate $E_{out,i}(g_2^{\mathcal{D}}(x))$ and $E_{out,i}(g_{10}^{\mathcal{D}}(x))$.
         E. Store overfit measure $E_{out,i}\left(g_{10}^{\mathcal{D}}\right) - E_{out,i}\left(g_2^{\mathcal{D}}\right)$.
3. Store the mean overfit measure over i repetitions of each (N, σ) combination, that is

$$M_{(\sigma,N)} = \frac{1}{100} \sum_{i=1}^{100} \left(E_{out,i}\left(g_{10}^{\mathcal{D}}\right) - E_{out,i}\left(g_2^{\mathcal{D}}\right) \right).$$

Hence, we simulate 100 datasets from a single target function f, and find the measure of overfit for each datset, for each (σ, N) combination. We take the average overfit measure over the 100 simulations of $\mathcal{D}$, and use this to produce the colour map given in Figure 3.

2.ii Overfit Measure (varying Q_f, N)

Here we attempt to understand the impact of model complexity on the overfitting measure at different N. We aim to find a measure of overfitting based on $E_{\mathcal{D},f}\left[E_{out}\left(g_{10}^{\mathcal{D}}\right) - E_{out}\left(g_2^{\mathcal{D}}\right)\right]$, and will hence require multiple simulations of target functions for each $Q_f \in \{1, \ldots, 40\}$ and multiple simulations of data for each Q_f. We use the values $N \in \{20, \ldots, 60\}$, and $Q_f \in \{1, \ldots, 40\}$, with fixed σ.

1. Set $\sigma = 0.2$.
2. For each $Q_f \in \{1, \ldots, 40\}$
   (a) Generate a random target function

$$f(x) = \sum_{q=0}^{Q_f} \beta_q L_q(x),$$

   with β_q drawn from $U(-1, 1)$.
   (b) For each $i \in \{1, \ldots, 20\}$
      i. For each $N \in \{20, \ldots, 60\}$

A. For each $j \in \{1, \ldots, 10\}$
   - Simulate a dataset ($\mathcal{D}$ of size N) from $f(x)$ having noise $\epsilon \sim N(0, \sigma)$.
   - Fit a 2^{nd} order polynomial to $\mathcal{D}$, $g_2^{\mathcal{D}}(x)$.
   - Fit a 10^{th} order polynomial to $\mathcal{D}$, $g_{10}^{\mathcal{D}}(x)$.
   - Calculate $E_{out,(i,j)}(g_2^{\mathcal{D}}(x))$ and $E_{out,(i,j)}(g_{10}^{\mathcal{D}}(x))$.
   - Store $E_{out,(i,j)}\left(g_{10}^{\mathcal{D}}\right) - E_{out,(i,j)}\left(g_2^{\mathcal{D}}\right)$.

3. Store the average overfit measure for each (Q_f, N) combination, that is

$$M_{Q_f,N} = \left(\frac{1}{20}\right)\left(\frac{1}{10}\right)\sum_{i=1}^{20}\sum_{j=1}^{10}\left(E_{out,(i,j)}\left(g_{10}^{\mathcal{D}}\right) - E_{out,(i,j)}\left(g_2^{\mathcal{D}}\right)\right).$$

This is then used as the metric to populate the colour map given in Figure 4.

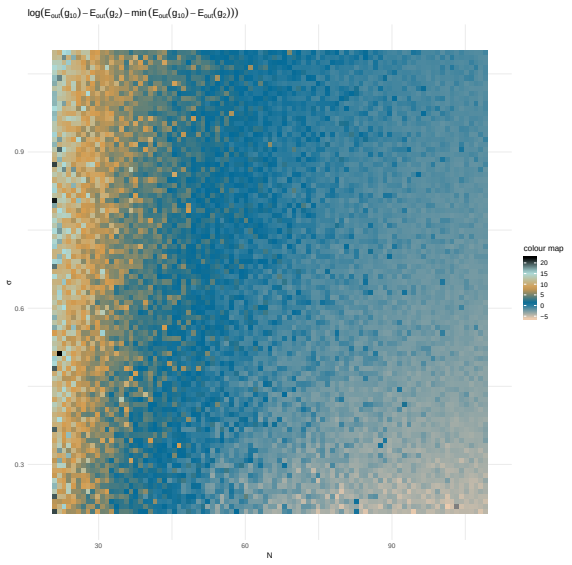

Figure 3: Colour map of scaled measure of overfitting based on $E_{\mathcal{D}}\left[E_{out}\left(g_{10}^{\mathcal{D}}\right) - E_{out}\left(g_2^{\mathcal{D}}\right)\right]$

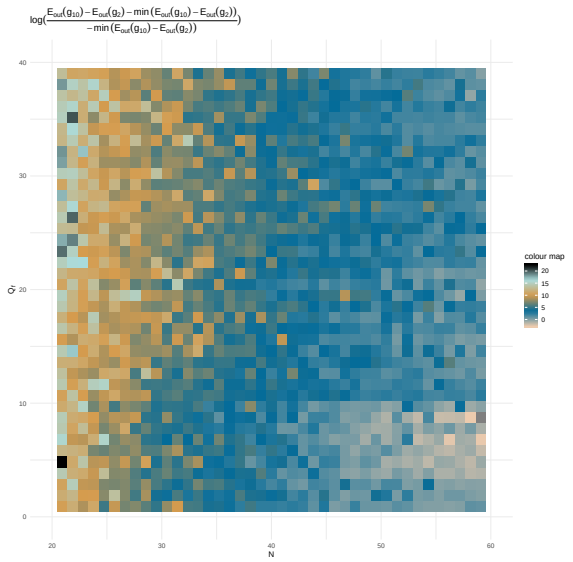

Figure 4: Colour map of scaled measure of overfitting based on $E_{\mathcal{D},f}\left[E_{out}\left(g_{10}^{\mathcal{D}}\right) - E_{out}\left(g_{2}^{\mathcal{D}}\right)\right]$

2.iii Discussion

Figure 3 illustrates that if we fix the target function and increase noise, so does the level of overfitting at fixed N (number of data points). If we fix the noise, σ overfitting increases as N decreases. Hence, to avoid overfitting with data which is higher in noise (high σ), we require more data (higher N) than when there is less noise (lower σ). However, in practise we do not know the level of noise.

Figure 4 illustrates that at a fixed noise level σ, overfitting increases as the target complexity increases at fixed N. In both Figures it is illustrated that the measure of overfitting decreases as we increase the number of data points. The emergence of the horizontal line at $Q_f = 10$ indicates that beyond this target complexity overfitting is high even at the largest values of N explored here.

The plots generated by Abu-Mostafa (2012) for an analogous problem appeared more smooth after tens of millions of runs. The plots we have generated echo the patterns that emerge, but with far less smoothness since we used far less runs. Even so, the computations incurred significant run time. Abu-Mostafa (2012) argues that the plots highlight that there is another factor aside from conventional noise that contributes to overfitting, which needs to be characterised. The noise which we have simulated to produce Figure 3 is defined as stochastic noise. The noise which is associated with target complexity in Figure 4 is defined as deterministic noise. This is described by Abu-Mostafa (2012) as the part of the target function which is not captured by $\mathcal{H}$. This can be observed in the top left region of Figure 2, where the 10^{th} order polynomial is incapable of capturing certain aspects of higher order polynomials

References

Abu-Mostafa, Y. (2012), 'Lecture 11 - overfitting'. Last accessed 19 October 2020.
URL: *https://www.youtube.com/watch?v=EQWr3GGCdzwhd=1*

George, B. et al. (2005), 'Mathematical methods for physics'.

3 Appendix

3.i R Code: Preamble

```
rm(list = ls(all = TRUE))
install.packages("wesanderson")
library(ggplot2)
library(ggthemes)
library(RColorBrewer)
library(wesanderson)
library(ggpubr)
```

3.ii R Code: Question 1

```
# QUESTION 1 i)
# Legendre polynomials
Legendre=function(x,n){
  val=0
  for(i in 0:n){
    val=val+((x^i)*choose(n,i)*choose((n+i-1)/2,n))
  }
  return((2^n)*val)
}

# plot Legendre polynomials of different orders
x<-seq(-1,1,0.01)
tans <- c("blue", "tan3", "darkgreen", "orange","darkblue")
plot(x,Legendre(x,1),type="l",
     xlab=expression(x),
     ylab=expression(L[n](x)),
     col=tans[1],lwd=2,
     main="Legendre polynomials")
lines(x,Legendre(x,0),type="l",col="black",lwd=2)
lines(x,Legendre(x,2),type="l",col=tans[2],lwd=2)
lines(x,Legendre(x,3),type="l",col=tans[3],lwd=2)
lines(x,Legendre(x,4),type="l",col=tans[4],lwd=2)
lines(x,Legendre(x,5),type="l",col=tans[5],lwd=2)
legend("bottomright", title = expression(q),
       legend = c(0:5),
       lty = 1, lwd = 2,
       col=c("black", tans),
       horiz = T)

# QUESTION 1 ii)
# creates random polynomial target functions
rLegfunc=function(x,n){
  val=0
  vec=runif(n,-1,1)
  for(i in 1:n){
```

```
    val=val+(Legendre(x,i)*vec[i])
  }
  return(val)
}

rPolyfunc=function(x,n){
  val=0
  vec=runif(n,-1,1)
  for(i in 1:n){
    val=val+(x^i*vec[i])
  }
  return(val)
}

# plot random target functions
x <- seq(-1,1,by = 0.01)
set.seed(2020)
par(mfrow=c(1,2))
# polynomial targets
tans <- c("blue", "darkgreen", "tan3")
plot(x,rPolyfunc(x,4),type="l",
     ylab=expression(f(x)),
     xlab=expression(x),
     col=tans[1],
     ylim=c(-2,2),
     lwd=2,
     main="Random polynomial target functions")
lines(x,rPolyfunc(x,5),type="l",col=tans[2], lwd=2)
lines(x,rPolyfunc(x,6),type="l",col=tans[3], lwd=2)
legend("bottomleft", title = expression(Q[f]),
       legend = c(5:7),
       lty = 1, lwd = 2,
       col=tans, horiz = T)
# Legendre polynomial targets
plot(x,rLegfunc(x,4),type="l",
     ylab=expression(f(x)),
     xlab=expression(x),
     col=tans[1], lwd=2,
     ylim=c(-2,2),
     main="Random Legendre target functions")
lines(x,rLegfunc(x,5),type="l",col=tans[2], lwd=2)
lines(x,rLegfunc(x,6),type="l",col=tans[3], lwd=2)
par(mfrow=c(1,1))
```

3.iii R Code: Question 2

```
# QUESTION 2i)
# generate random 10th order target function over [-1,1]
set.seed(2020)
func <- rLegfunc(x, 10)
plot(x, func, type='l')

#simulates a dataset with target "func" of size n with noise sig
generator=function(n,x,func,sig){
  l<-length(func)
  dat<-matrix(rep(NA,2*n),ncol=n)
  xdat<-floor(runif(n)*l)+1
  ydat<-func[xdat]+rnorm(n,0,sig)
  xdat<-x[xdat]
  Data<-data.frame(xdat,ydat)
  return(Data)
}

#fitted model of the data
fit=function(x,model){
  v=0
  for(i in 1:length(model$coefficient)){
    v=v+(model$coefficient[i]*(x^(i-1)))
  }
  return(v)
}

# gives the bias for a given fitted model
fdiff=function(x,target,model){
  f=fit(x,model)
  return((t(f-target)%*%(f-target))*(x[2]-x[1]))
}

# --------------------

Nseq <- seq(20,110,1)
sigseq <- seq(0.2,1.1,0.01)

# results
res1long <- data.frame(N=0, sigma=0, i=0,overfit=0)
colnames(res) <- paste(Nseq)

set.seed(2020)
for (i in 1:100){
  for (n in 1:length(Nseq)){
    N <- Nseq[n]
    for (s in 1:length(sigseq)){
      sig <- sigseq[s]
```

```
      #generate data
      data <- generator(n = N, x = x, func = func, sig = sig)
      # fit second/ten order polynomial to data
      two <- lm(I(data$ydat) ~ data$xdat + I(data$xdat^2))
      ten <- lm(I(data$ydat) ~ data$xdat + I(data$xdat^2) + I(data$xdat^3) +
      I(data$xdat^4)+ I(data$xdat^5)+
      I(data$xdat^6)+ I(data$xdat^7)+
      I(data$xdat^8)+ I(data$xdat^9)+
      I(data$xdat^10))
      err.out2<-fdiff(x,func,two)+(sig^2) # Out of sample error
      err.out10<-fdiff(x,func,ten)+(sig^2) # Out of sample error
      # difference between E_out for each model
      overfit <- err.out10 - err.out2
      # store
      res1long <- rbind(res1long, c(N=N, sigma = sig, i=i ,overfit = overfit))
    }
  }
}

beep()
res1long <- res1long[-1,]

# find means across different i
res1df <- data.frame(sigma =0, N=0, E_overfit = 0)
for (s in sigseq){
  sig.res <- res1long[which(res1long$sigma==s),]
  for (this.N in Nseq){
    res1df <- rbind(res1df,
                    cbind(sigma = s,
                          N =this.N,
                          E_overfit = mean(sig.res[which(sig.res$N==this.N),]$overfit)))
  }
}
res1df <- res1df[-1,]

# scale for better colour mapping
res1df$E_overfit <- log(res1df$E_overfit - min(res1df$E_overfit))

# make a palette
pal <- wes_palette("Darjeeling2", 100000, type = "continuous")

# colour map
plot1 <- ggplot(res1df, aes(N, sigma)) +
  geom_raster(aes(fill = E_overfit)) +
  theme_minimal() +
  ylab(expression(sigma))+
  scale_fill_gradientn(colors=pal, name = "colour map")+
  xlim(20,110) +
```

```
  ylim(0.2,1.1) +
  ggtitle(expression(log(E[out](g[10]) - E[out](g[2]) -
  min(group("(",E[out](g[10]) - E[out](g[2]),")")) + epsilon)))

plot1

#--------------------
# QUESTION 2 ii.)
Qfseq <- seq(1,40,by=1)
Nseq <- seq(20,60,by=1)
sig <- 0.2
Nreps <- 20 # number of repeats per value of Qf
Ndatasets <- 10 # number of datasets per Qf, N combination

# storage for results
res2long <- c()

set.seed(2020)
for (q in 1:length(Qfseq)){
  Qf <- Qfseq[q]
  func <- rLegfunc(x, Qf)

  for (i in 1:Nreps){
    for (n in 1:length(Nseq)){
      N <- Nseq[n]

      for (j in 1:Ndatasets){
        # generate data from target function
        data <- generator(n = N, x = x, func = func, sig = sig)
        # fit second/tenth order polynomial to data
        two <- lm(I(data$ydat) ~ data$xdat + I(data$xdat^2))
        ten <- lm(I(data$ydat) ~ data$xdat + I(data$xdat^2) + I(data$xdat^3) +
                    I(data$xdat^4)+ I(data$xdat^5)+
                    I(data$xdat^6)+ I(data$xdat^7)+
                    I(data$xdat^8)+ I(data$xdat^9)+
                    I(data$xdat^10))
        err.out2<-fdiff(x,func,two)+(sig^2) # Out of sample error
        err.out10<-fdiff(x,func,ten)+(sig^2) # Out of sample error
        # difference between E_out
        overfit <- err.out10 - err.out2
        res2long <- rbind(res2long, c(Qf, N, i, j, overfit))

      }
    }
  }
}

colnames(res2long) <- c("Qf", "N", "i", "j", "overfit")
res2long <- as.data.frame(res2long)
```

```
# find means across different i,j
res2df <- data.frame(Qf =0, N=0, E_overfit = 0)
for (this.q in Qfseq){
  Qf.res <- res2long[which(res2long$Qf==this.q),]
  for (this.N in Nseq){
    res2df <- rbind(res2df,
                    cbind(Qf = this.q,
                          N =this.N,
                          E_overfit = mean(Qf.res[which(Qf.res$N==this.N),]$overfit)))
  }
}
res2df <- res2df[-1,]

# scale for better colour mapping
res2df$E_overfit <-log((res2df$E_overfit-min(res2df$E_overfit))/(-min(res2df$E_overfit)))

# create a palette
pal2 <- wes_palette("Darjeeling2", n = 100000, type = "continuous")

# colour map
plot2<- ggplot(res2df, aes(N, Qf)) +
  geom_raster(aes(fill = E_overfit)) +
  theme_minimal() +
  ylab(expression(Q[f]))+
  scale_fill_gradientn(colors=pal2, name="colour map")+
  xlim(20,60) +
  ylim(0,40) +
  ggtitle(expression(log(frac(E[out](g[10]) - E[out](g[2]) -
                            min(group("(",E[out](g[10]) - E[out](g[2]),")")), -
                            min(group("(",E[out](g[10]) - E[out](g[2]),")")) + epsilon))))

plot2
```