732A99 - Lab 3, block 1 Group L4

Keshav Padiyar, Jacob Welander & Henrik Olofsson

14 December, 2020

Contents

statement of contribution

Keshav Padiyar: Assignment 3

Jacob Welander: Assignment 1

Henrik Olofsson: Assignment 2

Assignment 1 Kernel methods

You are asked to provide a temperature forecast for a date and place in Sweden. The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels:

The kernel is based on measure distances to, in this case there are 3 distances: *Physical* (spatial), *Date* which assume within a year, maximum 182 or 183 depending of leap year. And also *Time* 24 hours (assuming maximum distance of 12 hours). These distances $||x - x'||$ are later referred to as d.

```
# Calculate the time distance, this is not trivial since 0:23 can't be
# subtracted as usual but need to implement a statement that gives us
# a absolute time distance for both ways ( max distance 12 hours)
time_distance <- function(time_point,pred_time){
 abs_hour <- abs(hour(hms(time_point) - hms(pred_time)))
 ifelse(abs_hour < 12, abs_hour, 24 - abs_hour)
}

# The same logic as before is implemented with years
# This function somewhat takes in account leap years
# Starting by setting all time differences to the same year
# for calculating the difference in days ( with start of first of January)
# then also adding a day if leap year is present

date_distance <- function(date_point,pred_date ){
if(leap_year(date_point)){
  # we known that the year is a leap year
  fix_year <- as.Date(paste0(year(date_point),"-01-01"))
  date_diff <-  date_point - fix_year
  pred_diff <-  as.Date(paste0(year(date_point),"-",
                               month(pred_date),"-",day(pred_date))) - fix_year
  abs_diff <- abs(pred_diff-date_diff)
  ifelse(abs_diff < (182 + leap_year(date_point)) ,
         abs_diff, 365 + leap_year(date_point) - abs_diff )

} else{
  # we known that the year is not a leap year
  fix_year <- as.Date(paste0(year(date_point),"-01-01"))
  date_diff <-  date_point - fix_year
  pred_diff <-  as.Date(paste0(year(date_point),"-", month(pred_date),"-",
                               day(pred_date))) - fix_year
  abs_diff <- abs(pred_diff-date_diff)
  ifelse(abs_diff < 182, abs_diff, 365 - abs_diff )
}
}
```

A kernel which can be written as:

$$k\left(\frac{x - x'}{h}\right)$$

A simplified Gaussian kernel is implemented by taking the distance d and a specified h which is creating u.

$$k(u) = \exp\left(-\|u\|^2\right)$$

```
gaussian_kernel <- function(d,h){
exp(-(d/h)^2)
}
```

Creating the function kernel that take the data, the specified h values, and date and time to predict. Uses the simplified Gaussian kernel for all instances and loops over all specified time point and returns a data frame with the kernel of all time points.

```
kernel <- function(data,spatial_h,date_h,time_h,a,b,pred_times,pred_date){

  data$spatial_distance <- distHaversine(p1=c(a,b),
                            p2 = matrix(c(data$latitude,data$longitude),ncol=2))

data$date_distance <- date_distance(date_point = data$date, pred_date = pred_date)

spatial_kernel <-  gaussian_kernel(d= data$spatial_distance, h= spatial_h)

date_kernel <- gaussian_kernel(d= data$date_distance, h = date_h)

  kernel_list <- list()[1:length(pred_times)]

  for(t in 1:length(pred_times)){

  data$time_distance <- time_distance(time_point = data$time, pred_time=pred_times[t])
  time_kernel <- gaussian_kernel(d= data$time_distance, h= time_h)

  kernel_list[[t]] <- data.frame(spatial_kernel,date_kernel,time_kernel)

  }

  return(kernel_list)

}
```

Predicting the given coordinates (58.4274,14.826), the physical h is defined by a half standard deviation due to lack of knowledge of spatial impact of the air temperature, one can see in the following map (map) that the radius of h seems reasonable. Date h is given by 3 weeks, which will capture most of the weeks in a month but still not be influenced by other months to much. Time h is set to 3 hours which also is a subjective choice that make sense due to the rapid changes of temperature over day and night.

By multiplying each kernel observation with the air temperature one can predict the air temperature for the unknown weather station and time.

```
mult_kern <- function(kern,respons,time_pred){
result <- data.frame(time_pred, temp_pred=NA)
for( t in 1:length(kern)){
kern_pred <-  apply(kern[[t]],1,prod)
result[t,2]<- sum(kern_pred * respons)/sum(kern_pred)
}
return(result)
}
```

In the same way as before one can predict the air temperature for the unknown weather station and time, but instead of multiplying each kernel one can sum them up observation wise in the following way.

```
sum_kern <- function(kern,respons,time_pred){
result <- data.frame(time_pred, temp_pred=NA)
for( t in 1:length(kern)){
kern_pred <-  rowSums(kern[[t]])
result[t,2]<- sum(kern_pred * respons)/sum(kern_pred)
}
return(result)
}
```

The first date that is predicted is 2013-11-04 which is given by the assignment, the position is always fixed on the given coordinates.

```
mult <- mult_kern(kern = kern, respons = st2$air_temperature,time_pred = times)
sum <- sum_kern(kern = kern, respons = st2$air_temperature,time_pred = times)

ggplot(mult,aes(x=time_pred, y=temp_pred)) + geom_point(color="blue") +
geom_point(y=sum$temp_pred,color="red") + ylim(c(min(mult$temp_pred,sum$temp_pred),
                                                 max(mult$temp_pred,sum$temp_pred))) +
labs(y="Predicted Air Temperature",x="Time")
```

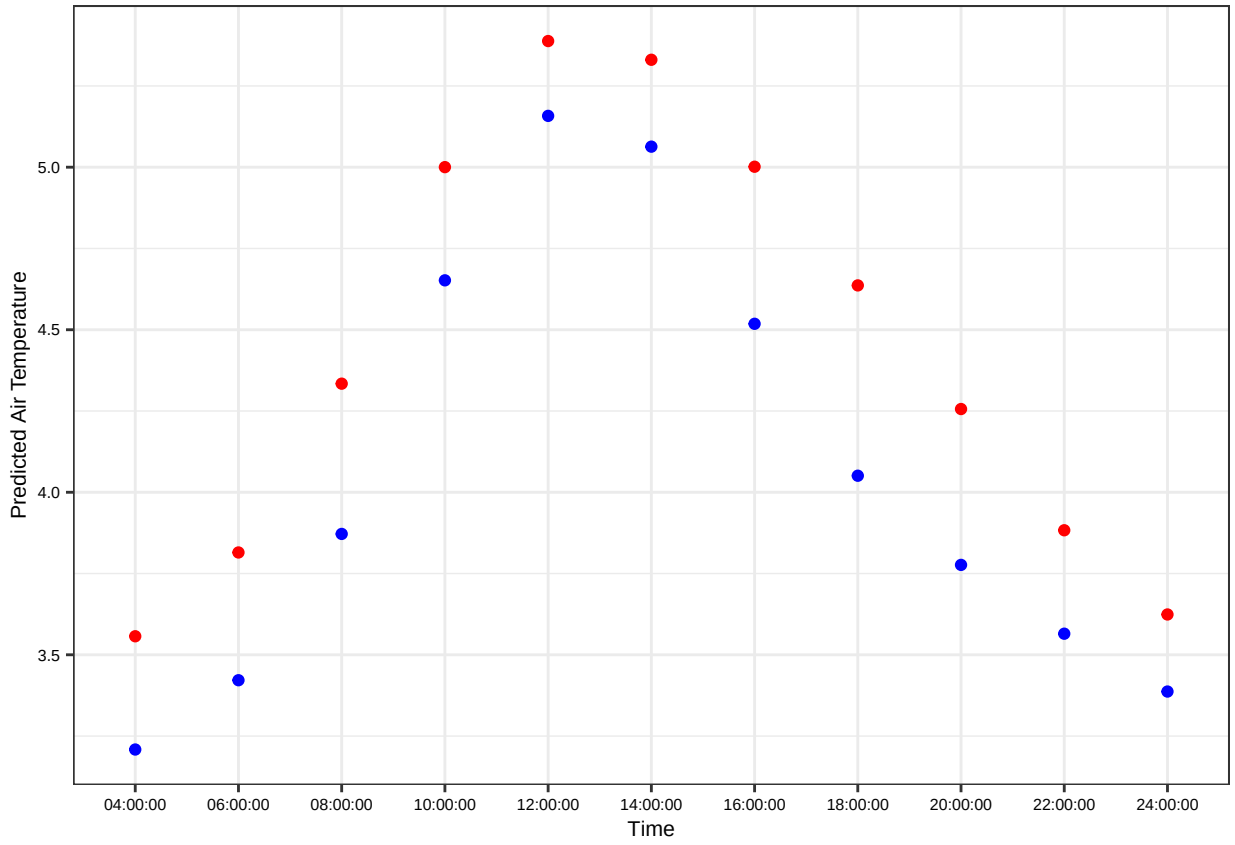


Figure 1: Kernel prediction 1

The two prediction follows closely, where the red indicates summation and blue indicates the multiplying kernel. In this case air temperature seems to peak at around 13:00 around 6 degrees celsius and drops down to 3 degrees celsius at night, which seems to be a logic value for this time of the year.

The second date that is predicted is 2013-06-28 which in comparison is during the summer. The position is fixed on the given coordinates. This demonstrates how the kernel models compares when high temperatures are introduced.

```
mult <- mult_kern(kern = kern, respons = st2$air_temperature,time_pred = times)
sum <- sum_kern(kern = kern, respons = st2$air_temperature,time_pred = times)

ggplot(mult,aes(x=time_pred, y=temp_pred)) + geom_point(color="blue") +
geom_point(y=sum$temp_pred,color="red") + ylim(c(min(mult$temp_pred,sum$temp_pred),
                                                 max(mult$temp_pred,sum$temp_pred))) +
labs(y="Predicted Air Temperature",x="Time")
```

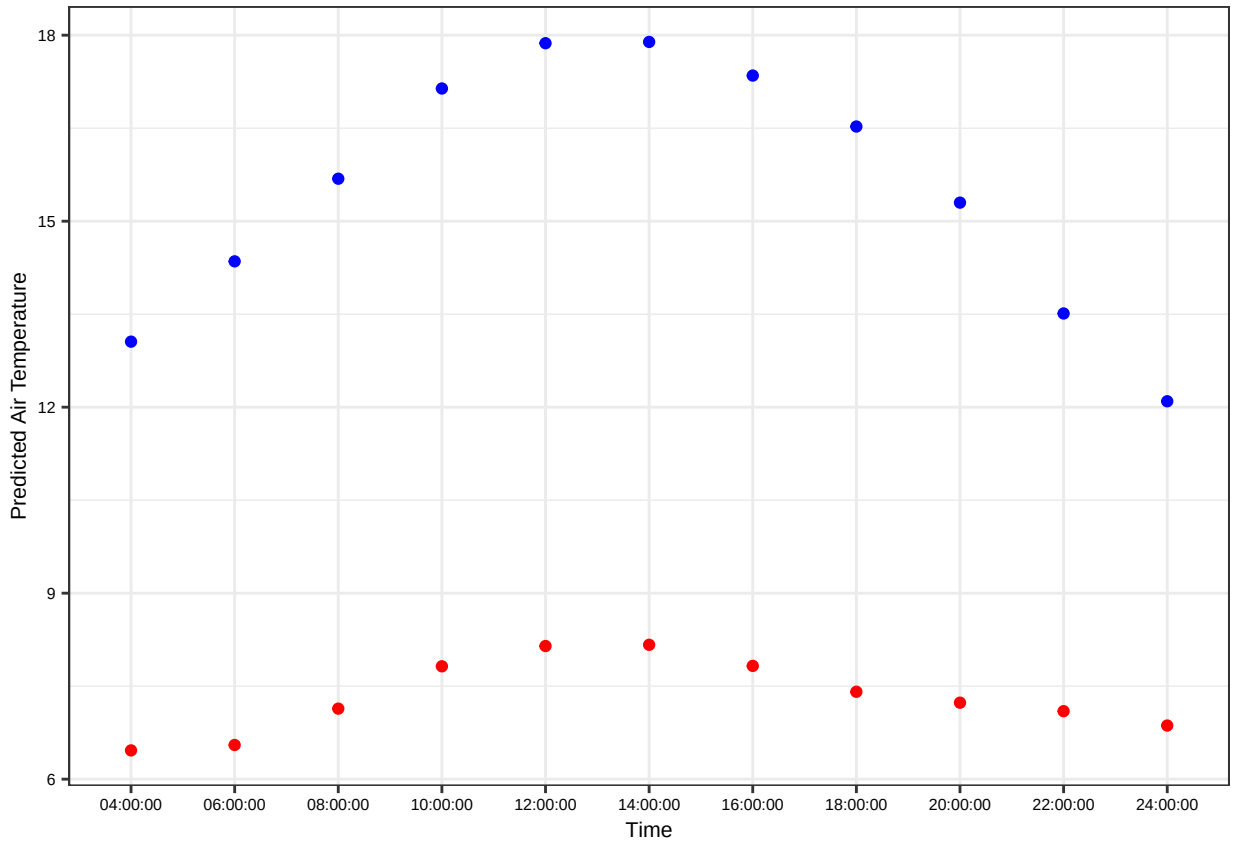


Figure 2: Kernel prediction 2

In this case the two prediction does not follows closely as before, where the red indicates summation and blue indicates the multiplying kernel. Air temperature seems to peak at around 13:00 around 18 degrees celsius and drops down to 12 degrees celsius at night for the multiplying kernel, which seems to be a logic value for this time of the year. However the summation kernel peaks at 8.5 degrees celsius which does not seems reasonable for that time of the year.

The reason behind this may be that the summation kernel are independent of each other due to the addition of each kernel, and does not affect the final kernel that is used to predict the air temperature. In comparison when the multiplying each kernel they will affect each other and to get a better estimate over all due to the kernels being combined.

Lastly, all of the Gaussian kernels are visualized with their selected h from distance 0 to their maximum distance in the data. Here one can see how the smoothing parameter h gives more weights to observations with small distances and lower weights to observations with distances that are far away.

```
par(mfrow=c(1,3))
curve(gaussian_kernel(x,h=h_distance),from = 0,to=max(st2$spatial_distance),
      xlab="Physical distance")
curve(gaussian_kernel(x,h=h_time),from = 0,to=12, xlab="Time distance")
curve(gaussian_kernel(x,h=h_date),from = 0,to=183, xlab="Day distance")
```

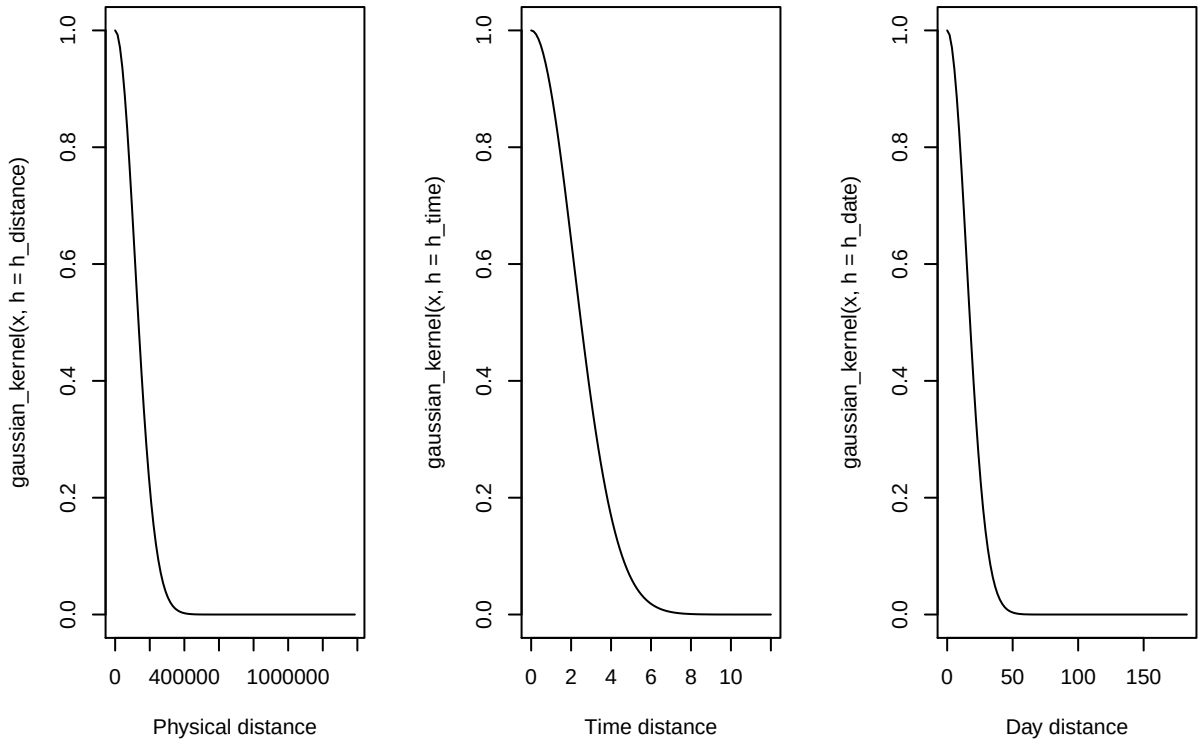


Figure 3: Gaussian kernels with different h and distances

Assignment 2 Support vector machine

1. Introduction

In this assignment, the `spam` data set is used, containing the frequencies for different words in emails. This information will be used to train a support vector machine (SVM) models to predict these emails as spam or non-spam (binary classification). An SVM is a sparse (i.e. uses only a subset of the points to build the model) method that uses a margin-maximizing approach to solve both classification and regression tasks. The assignment focuses more on the theory of machine learning than SVM's, leading us to explore why we use the holdout method to assess model performance.

In the first step, the data is split into some different sets:

- training
- validation
- training and validation
- test

2. Model Training

We then proceed to train a number of models using a range of values for `C`, a cost parameter. At this stage, training is done using the training dataset and errors are calculated using the validation set. This is a simple grid search that produces an error statistic for each of the values tried for `C`, from 0.3 to 4.8. We then select the `C` which gives the lowest error.

Optimal `C` selected:

```
## [1] 1.2
```

We then use this value of C to refit the model and calculate errors in different configurations:

- Case 0: train using **training** data and calculate errors for **validation** set
- Case 1: train using **training** data and calculate errors for **test** set
- Case 2: train using **training+validation** data and calculate errors for **test** set
- Case 3: train using all data and calculate errors using **test** set

Remember, all configurations uses the same value for the cost parameter, which has been estimated from the **validation** set.

The different error rates for the filters:

```
## err0

## [1] 0.07

## err1

## [1] 0.08489388

## err2

## [1] 0.08364544

## err3

## [1] 0.03370787
```

3. Questions and Answers

1. Which filter do we return to the user? filter0, filter1, filter2 or filter3? Why?

We return `filter3` as we are now done with model estimation and are ready to use the filter. At this stage we use all available data to finalize our model. We only use the holdout method with different data sets to tune the model and to verify that the method of fitting a model to data is suitable - SVM in this case - or perform model selection if there are many different models. Please see motivation below for further explanation.

2. What is the estimate of the generalization error of the filter returned? err0, err1, err2 or err3? Why?

The best estimate of the generalization error would be that of `err1` (0.085).

Motivation:

Let us examine the different scenarios:

In all four cases, the models are fitted with a hyperparameter `C` of 1.2, which has been determined using training and validation data, i.e. the value for `C` that minimizes the error of the validation set.

The hyperparameter `C` is a user-defined parameter that controls regularization. Larger values of `C` adjusts the slack, that is it penalises the model for points that are misclassified or in the margin more. The effect of this, for large values of `C`, is that the model will attempt to fit every point despite that fit not being generalizable, i.e. overfitting occurs. On the other end of the spectrum a low `C` generalizes the model well as misclassified (or near-misclassified) points do not incur that much extra loss. This smoothens the model a lot. As we can see, `C` is directly related to the regularization and therefore the generalization error of the model. This parameter has been selected using the validation set, excellent!

As we arrive at the optimal cost hyperparameter using the validation dataset, using the same set to test the model performance (as done with `err0`) would underestimate the generalization error. Why? We would then be using the same set to both tune hyperparameters (e.g. decide cost parameter) and test the model, creating a leak. This should exclude `err0` from consideration for this purpose.

Why not use the error calculated from `filter3`? Well, now we are again using the test data itself to fit the model, which would introduce a huge leakage if we were to assess the model performance using the error it gives. Indeed, the test error of this filter is much lower than the other filters' and close to the error of the training set.

Why then would we not pick `err2`, or do we expect the best approximation of the generalization error to be that of `err1`? For `err2` we are again risking some bleeding when using the same set multiple times (testing on a model trained on training and validation sets and optimized for the validation set). Utilizing three isolated sets and only using them for their specific purposes and then discarding them would avoid the possibility of data leakage. This is exactly what we are doing to calculate `err1`. The generalization error obviously does not change depending on what data we use to train the final model, leading us to pick the *theoretically* best approximation of the error, i.e. `err1`, but the filter that uses all data available to us, `filter3`.

In short, the training set lets us train a model, the validation set lets us configure this model in an optimal way and the test set lets us see how well the method of training the model generalizes. The most important lesson here, and what machine learning is all about, is that for actual finalization and deployment of a model, none of our sets are important. We are doing all of this work just to figure out what method using which settings for the hyperparameters are appropriate to model the type of data we will feed it - is the machine able to learn from future data of the same kind? If our model fitting approach seems to work well, we apply the method to our data.

Assignment 3 Neural Networks

1.

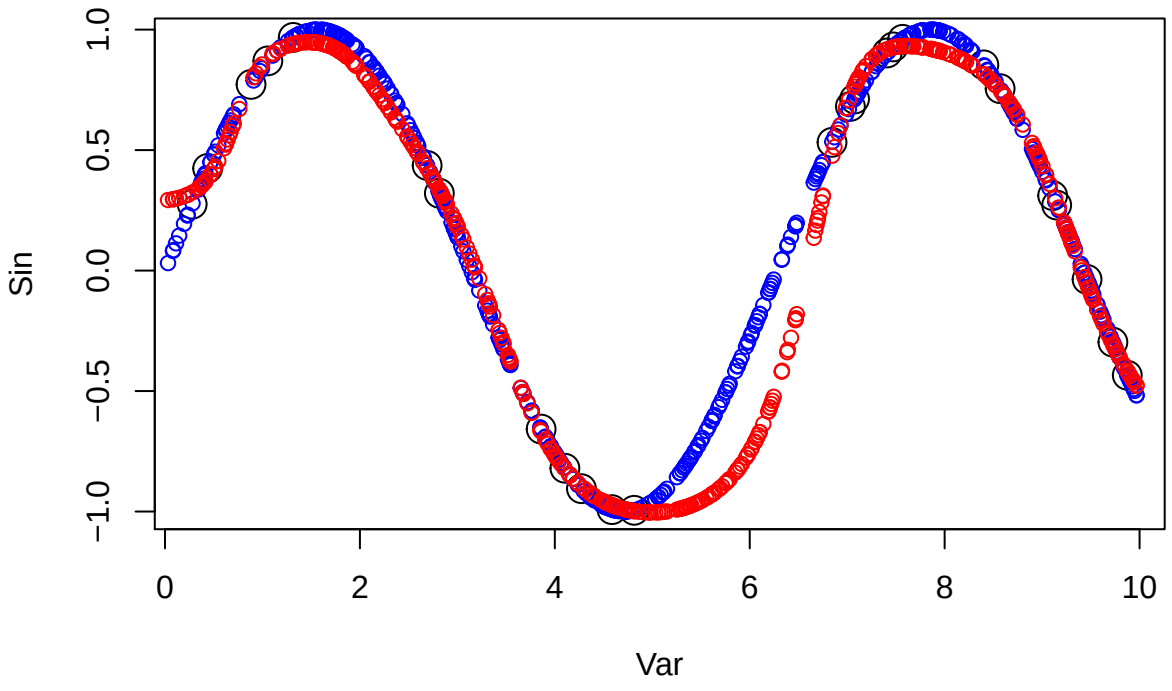
Train a neural network to learn the trigonometric sine function. To do so, sample 500 points uniformly at random in the interval [0,10].

```
Var <- runif(500, 0, 10) # sample 500 points uniformly at random in the interval [0;10]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
tr <- mydata[1:25,] # Use 25 of the 500 points for training
te <- mydata[26:500,] # Test


# Random initialization of the weights in the interval [-1, 1]
winit <- runif(25, -1, 1)# Your code here

# Use any number of layers and hidden units that you consider appropriate

nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit)
pred <- predict(nn,te)
#plot(nn)
#resError <- sum((te[,2]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2) # Plot the training
points(te, col = "blue", cex=1) # Plot test data
points(te[,1],predict(nn,te), col="red", cex=1) # plot predictions
```




```
# Comment your results:
```

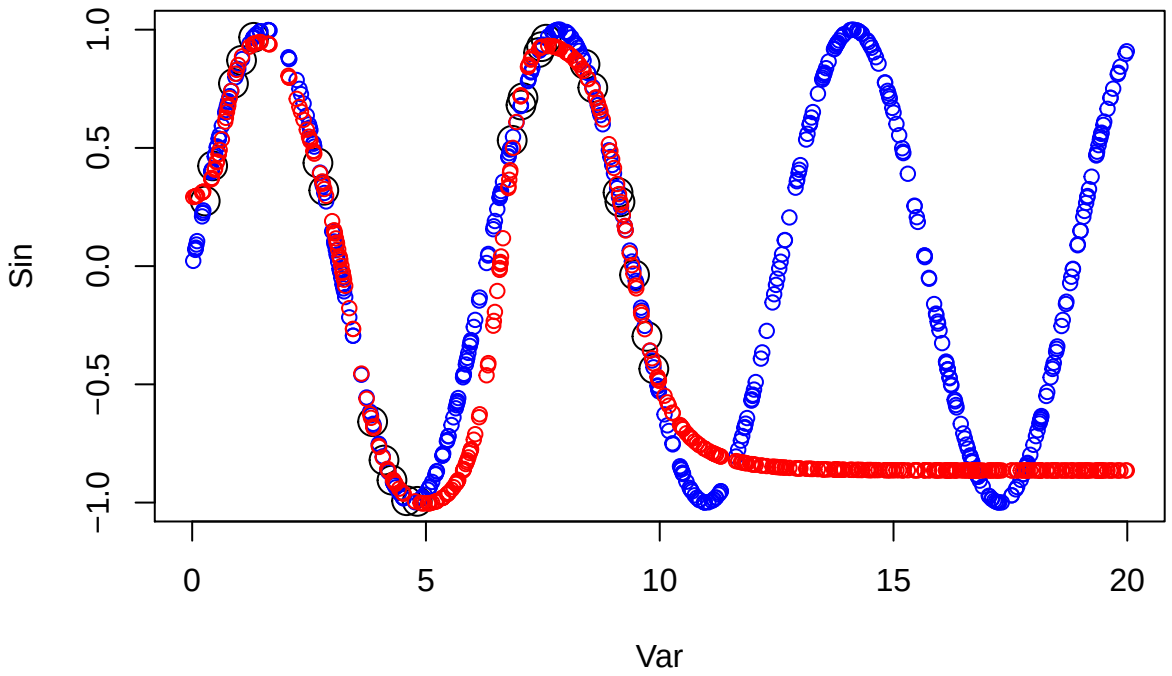
Above graph depicts that, the predicted points approximately interpolate the true values. However there

are certain points where we could see some deviation, for example, the interval between [5,7.5]. This is due to very less training points.

2

Then, sample 500 points uniformly at random in the interval [0,20], and apply the sine function to each point. Use the previously learned NN to predict the sine function value for these new 500 points.

```
Var <- runif(500, 0, 20) # sample 500 points uniformly at random in the interval [0;20]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
pred <- predict(nn,mydata)
#resError <- sum((mydata[,2]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2,xlim=c(0,20),ylim=c(-1,1)) # Plot the training
points(mydata, col = "blue", cex=1) # Plot test data
points(mydata[,1],pred, col="red", cex=1) # plot predictions
```

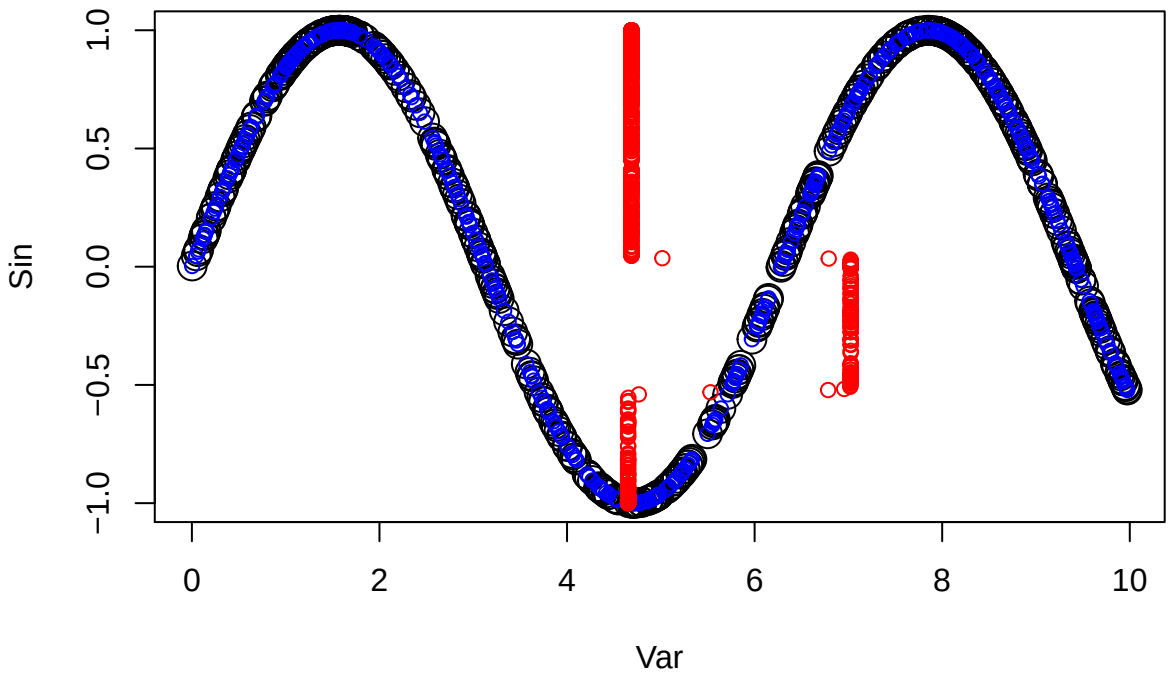


We have trained the model with the uniformly distributed points within the interval 0 and 10. Here, we are trying to fit the model with the data points that are from interval 0-20. Therefore, in the above graph, we could see that for the data points within 0-10 range the model fits well. And beyond that there are errors. That is model couldn't extrapolate those data points. Hence we see a mixed result.

3

Finally, sample 500 points uniformly at random in the interval [0,10], and apply the sine function to each point. Use all these points as training points for learning a NN that tries to predict x from sin(x), i.e. unlike before when the goals was to predict sin(x) from x. You should get bad results.

```
Var <- runif(500, 0, 10) # sample 500 points uniformly at random in the interval [0;10]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
nn <- neuralnet(formula = Var ~ Sin, data = mydata, hidden = 2, startweights = winit)
#plot(nn)
pred <- predict(nn,mydata)
#resError <- sum((mydata[1,]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(mydata, cex=2) # Plot the training
points(mydata, col = "blue", cex=1) # Plot test data
points(pred,mydata$Sin, col="red", cex=1) # plot predictions
```



In this case, we are trying to predict the x values given its *Sin* function. In the above graph Let us consider *Sin(0)*, *Sin(0)* has multiple (around 4) X values, neural network will take the average of those 4 X values and it is true with majority of data points. Hence we could see a straight line at the center of the plot.

In addition, there is another line at the right side, this is because, there are slightly more data points on the right side (for the interval [-0.5,0] on Y axis).

Finally, we could see some individual points in the interval ∼[5,7]. Because there are some gaps in the training points in that region.

The $\sin(x)$ is periodic in nature, neural network tries to average out the repeated values and converges with high error rates given less nodes (up to 3) and single hidden layer. Beyond this setting the model is not converging at all.

Code Appendix

```
library("ggplot2")
library("geosphere")
library("lubridate")
library("dplyr")

theme_set(theme_bw())
theme_update(plot.title = element_text(size=16,face="bold",hjust = 0.5),
             axis.text = element_text(size=6,colour="black"),
             axis.title = element_text(size=8),
             legend.title = element_text(size=15))


# Calculate the time distance, this is not trivial since 0:23 can't be
# subtracted as usual but need to implement a statement that gives us
# a absolute time distance for both ways ( max distance 12 hours)
time_distance <- function(time_point,pred_time){
 abs_hour <- abs(hour(hms(time_point) - hms(pred_time)))
 ifelse(abs_hour < 12, abs_hour, 24 - abs_hour)
}

# The same logic as before is implemented with years
# This function somewhat takes in account leap years
# Starting by setting all time differences to the same year
# for calculating the difference in days ( with start of first of January)
# then also adding a day if leap year is present

date_distance <- function(date_point,pred_date ){
if(leap_year(date_point)){
  # we known that the year is a leap year
  fix_year <- as.Date(paste0(year(date_point),"-01-01"))
  date_diff <-  date_point - fix_year
  pred_diff <-  as.Date(paste0(year(date_point),"-",
                               month(pred_date),"-",day(pred_date))) - fix_year
  abs_diff <- abs(pred_diff-date_diff)
  ifelse(abs_diff < (182 + leap_year(date_point)) ,
         abs_diff, 365 + leap_year(date_point) - abs_diff )

} else{
  # we known that the year is not a leap year
  fix_year <- as.Date(paste0(year(date_point),"-01-01"))
  date_diff <-  date_point - fix_year
  pred_diff <-  as.Date(paste0(year(date_point),"-", month(pred_date),"-",
                               day(pred_date))) - fix_year
  abs_diff <- abs(pred_diff-date_diff)
  ifelse(abs_diff < 182, abs_diff, 365 - abs_diff )
}
}

gaussian_kernel <- function(d,h){
exp(-(d/h)^2)
}
```

```
kernel <- function(data,spatial_h,date_h,time_h,a,b,pred_times,pred_date){

  data$spatial_distance <- distHaversine(p1=c(a,b),
                            p2 = matrix(c(data$latitude,data$longitude),ncol=2))

data$date_distance <- date_distance(date_point = data$date, pred_date = pred_date)

spatial_kernel <-  gaussian_kernel(d= data$spatial_distance, h= spatial_h)

date_kernel <- gaussian_kernel(d= data$date_distance, h = date_h)

  kernel_list <- list()[1:length(pred_times)]

  for(t in 1:length(pred_times)){

  data$time_distance <- time_distance(time_point = data$time, pred_time=pred_times[t])
  time_kernel <- gaussian_kernel(d= data$time_distance, h= time_h)

  kernel_list[[t]] <- data.frame(spatial_kernel,date_kernel,time_kernel)

  }

  return(kernel_list)

}

mult_kern <- function(kern,respons,time_pred){
result <- data.frame(time_pred, temp_pred=NA)
for( t in 1:length(kern)){
kern_pred <-  apply(kern[[t]],1,prod)
result[t,2]<- sum(kern_pred * respons)/sum(kern_pred)
}
return(result)
}


sum_kern <- function(kern,respons,time_pred){
result <- data.frame(time_pred, temp_pred=NA)
for( t in 1:length(kern)){
kern_pred <-  rowSums(kern[[t]])
result[t,2]<- sum(kern_pred * respons)/sum(kern_pred)
}
return(result)
}


stations <- read.csv("stations.csv",fileEncoding="latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
st$date <- as.Date(st$date)

pred_date <- "2013-11-04"
times <- c("04:00:00", "06:00:00", "08:00:00",paste(seq(10,24,2),":00:00",
```

```
                                                    sep = ""))
# filtering every observation to a given date
# also include the first hours if they exists, 02:00:00 etc.
st2 <- st %>% filter(date < pred_date | date==pred_date & time< times[1])


a <- 58.4274
b <- 14.826

st2$spatial_distance <- distHaversine(p1=c(a,b),
                                      p2 = matrix(c(st2$latitude,st2$longitude),
                                                  ncol=2))
st2$date_distance <- date_distance(date_point = st2$date, pred_date = pred_date)
st2$time_distance <- time_distance(time_point = st2$time, pred_time=times[1]) # obs, only first time

h_distance <- sd(st2$spatial_distance)/2
h_date <- 21
h_time <- 3
kern <- kernel(st2, spatial_h = h_distance, date_h=h_date, time_h=h_time,
               a=a,b=b,pred_times = times, pred_date = pred_date)


mult <- mult_kern(kern = kern, respons = st2$air_temperature,time_pred = times)
sum <- sum_kern(kern = kern, respons = st2$air_temperature,time_pred = times)

ggplot(mult,aes(x=time_pred, y=temp_pred)) + geom_point(color="blue") +
geom_point(y=sum$temp_pred,color="red") + ylim(c(min(mult$temp_pred,sum$temp_pred),
                                                 max(mult$temp_pred,sum$temp_pred))) +
labs(y="Predicted Air Temperature",x="Time")
stations <- read.csv("stations.csv",fileEncoding="latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
st$date <- as.Date(st$date)

pred_date <- "2013-06-28"
times <- c("04:00:00", "06:00:00", "08:00:00",paste(seq(10,24,2),":00:00",
                                                    sep = ""))

st2 <- st %>% filter(date < pred_date | date==pred_date & time< times[1])


a <- 58.4274
b <- 14.826

st2$spatial_distance <- distHaversine(p1=c(a,b),
                                      p2 = matrix(c(st2$latitude,st2$longitude),
                                                  ncol=2))
st2$date_distance <- date_distance(date_point = st2$date, pred_date = pred_date)
st2$time_distance <- time_distance(time_point = st2$time, pred_time=times[1]) # obs, only first time

h_distance <- sd(st2$spatial_distance)/2
h_date <- 21
h_time <- 3
```

```r
kern <- kernel(st2, spatial_h = h_distance, date_h=h_date, time_h=h_time, a=a,b=b,pred_times = times, pr


mult <- mult_kern(kern = kern, respons = st2$air_temperature,time_pred = times)
sum <- sum_kern(kern = kern, respons = st2$air_temperature,time_pred = times)

ggplot(mult,aes(x=time_pred, y=temp_pred)) + geom_point(color="blue") +
geom_point(y=sum$temp_pred,color="red") + ylim(c(min(mult$temp_pred,sum$temp_pred),
                                                 max(mult$temp_pred,sum$temp_pred))) +
labs(y="Predicted Air Temperature",x="Time")

par(mfrow=c(1,3))
curve(gaussian_kernel(x,h=h_distance),from = 0,to=max(st2$spatial_distance),
      xlab="Physical distance")
curve(gaussian_kernel(x,h=h_time),from = 0,to=12, xlab="Time distance")
curve(gaussian_kernel(x,h=h_date),from = 0,to=183, xlab="Day distance")
######################### Assignment 2 ########################
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(message = TRUE)

library(kernlab)
set.seed(1234567890)

data(spam)

index <- sample(1:4601)
tr <- spam[index[1:3000], ]
va <- spam[index[3001:3800], ]
trva <- spam[index[1:3800], ]
te <- spam[index[3801:4601], ]

by <- 0.3
err_va <- NULL
# tunes the cost/slack by grid search from .3 to 4.8 by .3
# trains on the training data and uses validation data to predict
# error calculated using validation data
for(i in seq(by,5,by)){
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=i)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}

which.min(err_va)*by
# = 1.2
# uses training data to train, validation data to predict and calculates error on validation
filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
message("err0")
err0
```

```
# uses training data to train, test data to predict and calculates error on test
filter1 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by)
mailtype <- predict(filter1,te[,-58])
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
message("err1")
err1

# uses training+validation data to train, test data to predict and calculates error on test
filter2 <- ksvm(type~.,data=trva,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by)
mailtype <- predict(filter2,te[,-58])
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
message("err2")
err2

# uses all data to train, test data to predict and calculates error on test
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by)
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
message("err3")
err3

######################### End of code for assignment 2 ########################
knitr::opts_chunk$set(echo = TRUE)
library(neuralnet)
set.seed(1234567890)
Var <- runif(500, 0, 10) # sample 500 points uniformly at random in the interval [0;10]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
tr <- mydata[1:25,] # Use 25 of the 500 points for training
te <- mydata[26:500,] # Test


# Random initialization of the weights in the interval [-1, 1]
winit <- runif(25, -1, 1)# Your code here

# Use any number of layers and hidden units that you consider appropriate

nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit)
pred <- predict(nn,te)
#plot(nn)
#resError <- sum((te[,2]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2) # Plot the training
points(te, col = "blue", cex=1) # Plot test data
points(te[,1],predict(nn,te), col="red", cex=1) # plot predictions
# Comment your results:
Var <- runif(500, 0, 20) # sample 500 points uniformly at random in the interval [0;20]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
pred <- predict(nn,mydata)
#resError <- sum((mydata[,2]-pred)^2)/2
```

```
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2,xlim=c(0,20),ylim=c(-1,1)) # Plot the training
points(mydata, col = "blue", cex=1) # Plot test data
points(mydata[,1],pred, col="red", cex=1) # plot predictions
Var <- runif(500, 0, 10) # sample 500 points uniformly at random in the interval [0;10]
mydata <- data.frame(Var, Sin=sin(Var)) # Apply the sine function to each point
nn <- neuralnet(formula = Var ~ Sin, data = mydata, hidden = 2, startweights = winit)
#plot(nn)
pred <- predict(nn,mydata)
#resError <- sum((mydata[1,]-pred)^2)/2
#resError
# Plot of the training data (black), test data (blue), and predictions (red)
plot(mydata, cex=2) # Plot the training
points(mydata, col = "blue", cex=1) # Plot test data
points(pred,mydata$Sin, col="red", cex=1) # plot predictions
```