Report_Lab3

Karol Wojtulewicz

Assignment 1

In this assignment the task was to predict weather for some coordinates (in Sweden) and date with hours between 4 and 24. For this assignment three gaussian kernels will be used; one for distance from the position in question, one for day difference for the date in question and one for the time difference for the hour in question. For this assignment the data has been provided by the Swedish Meteorological and Hydrological Institute (SMHI) and it consists of stations coordinates, dates, times and much more information not used in this task.

We start with setting different parameters used to build the kernels. First three values provided below are the smoothness factors h for the three different kernels. The comments next to them say how the author has reasoned with their estimation. The date for the estimate is 2004-09-03 and the place is the home town of the author, Sundsvall, posisioned in the very middle of Sweden.

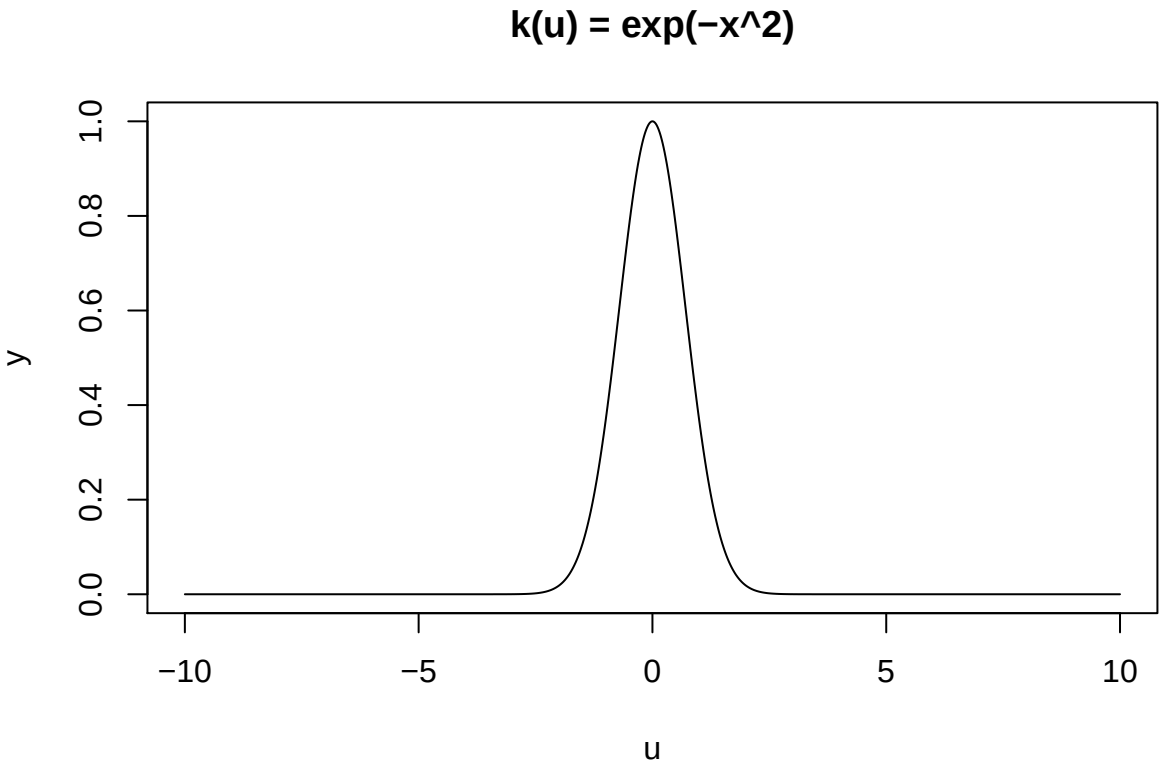
```
h_distance <- 100*1000 # large distances in meters (this is why *1000)
h_date <- 30 #30 days in a month?
h_time <-3 # because we have 2h intervalls
mydate = "2004-09-03"
place.to.predict = c(17.3069, 62.3908) #sundsvall
selected.dates = subset(st, as.Date(st$date) < as.Date(mydate))
```

Next we write the kernels for this task. Below one can see three functions, that each calculates a vector consisting of the y values for the each kernel. The x values are simply the intex of the data posision. In the each kernel the Gaussian kernel is applied on each point in the given dataset, that is the function: $k(u_i) = e^{-u_i^2}$. Here $u_i = \frac{x_i - x}{h}$. For the estimation of the distance between coordinates the function $distHaversine$ is used, that returns the distance between points in meters.

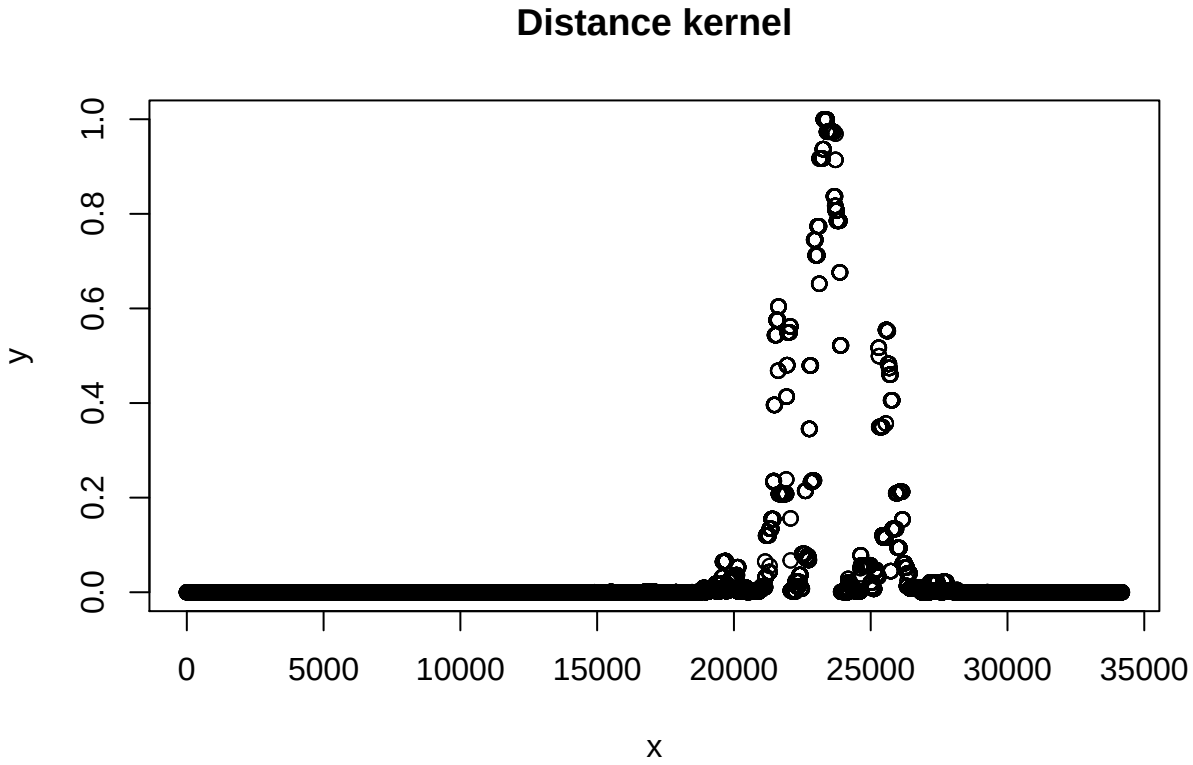
```
get.diff.distance.smooth = function(data, point.coord,h){
  return.vector = 1:length(data[,1])
  for (i in 1:length(data[,1])) {
    x = distHaversine(c(data$longitude[i], data$latitude[i]), point.coord)/h
    return.vector[i] = exp(-x^2)
  }
  return(return.vector)

}

get.diff.date.smooth = function(data, day.of.intresst, h){
  return.vector = 1:length(data[,1])
  for( i in return.vector){
    x = as.numeric(as.Date(day.of.intresst)-as.Date(data$date[i]))/h
    return.vector[i] = exp(-(x)^2)
  }
  return(return.vector)
}

get.diff.time.smooth= function(data, mytime,h){

  return.vector = 1:length(data[,1])
  for( i in return.vector){
```

```
    time1 = as.POSIXct(as.character(data$time[i]), format = "%H:%M:%S", tz = "UTC")
    time2 = as.POSIXct(as.character(mytime), format = "%H:%M:%S", tz = "UTC")
    x = (time1-time2)/h
    return.vector[i] = exp(-as.numeric(x)^2)
    }
  return(return.vector)
}
```

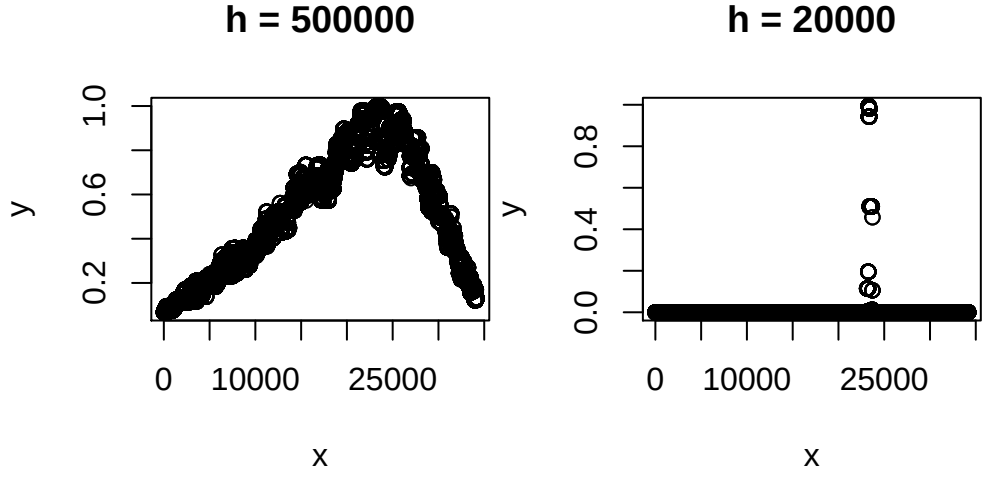
Provided below is the plot of the kernel function mentioned above. After applying the kernel on the dataset should look similar to that function. The smoothing factor h will make the curve wider, or narrower.



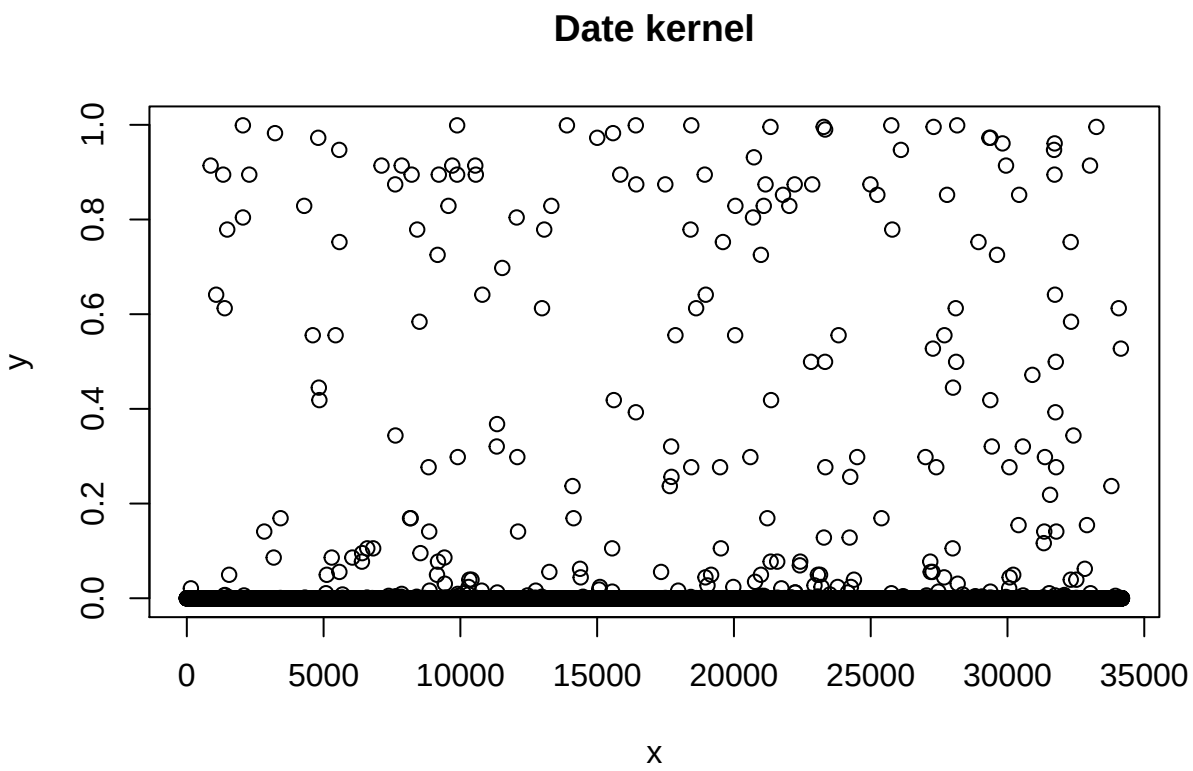
When we apply the kernel on the distance difference and plot the vector recieved from the kernel funtion with the previously given h value, we get the plot seen below. This plot look similar to the kernel funtion seen above, which means that we are on the good track.



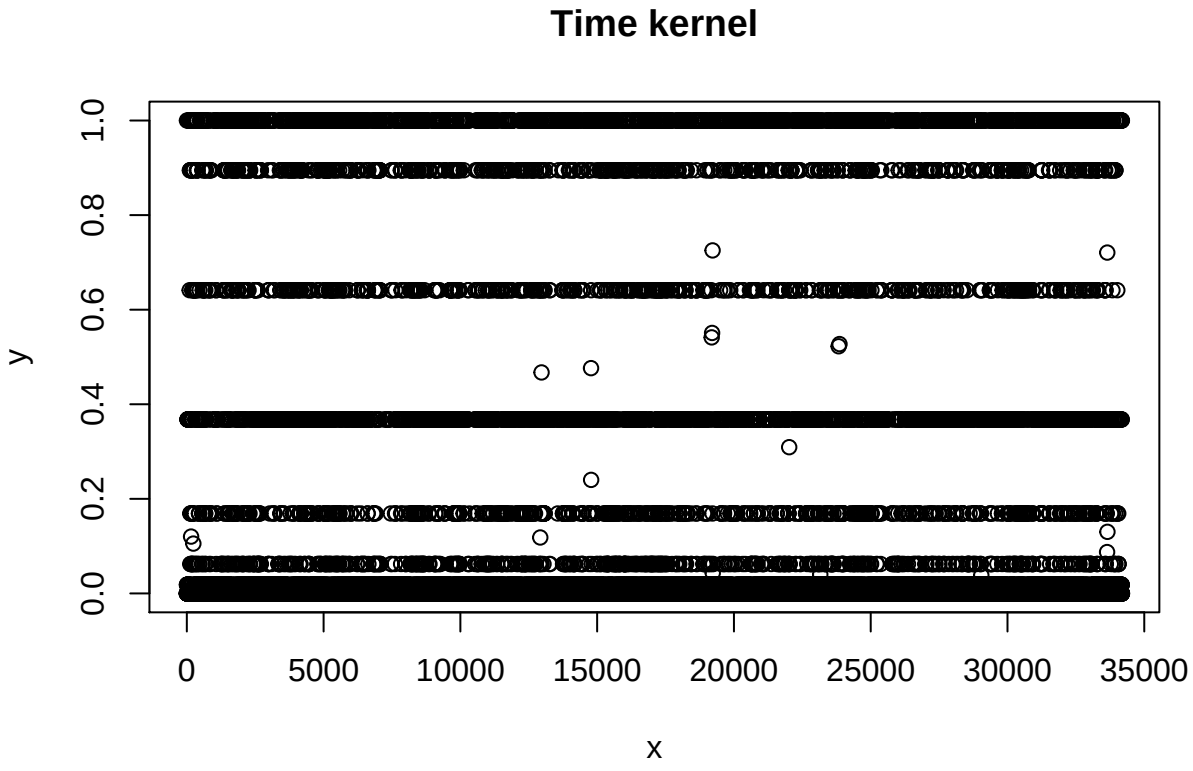
To understand better what kind of changes the h smoothing value causes, the plots with different h values (higher and lower than the original) are provided. Higher h value smooths the curve to, seen on the left, and lower h value narrows the curve to the point where there are just few points visible.



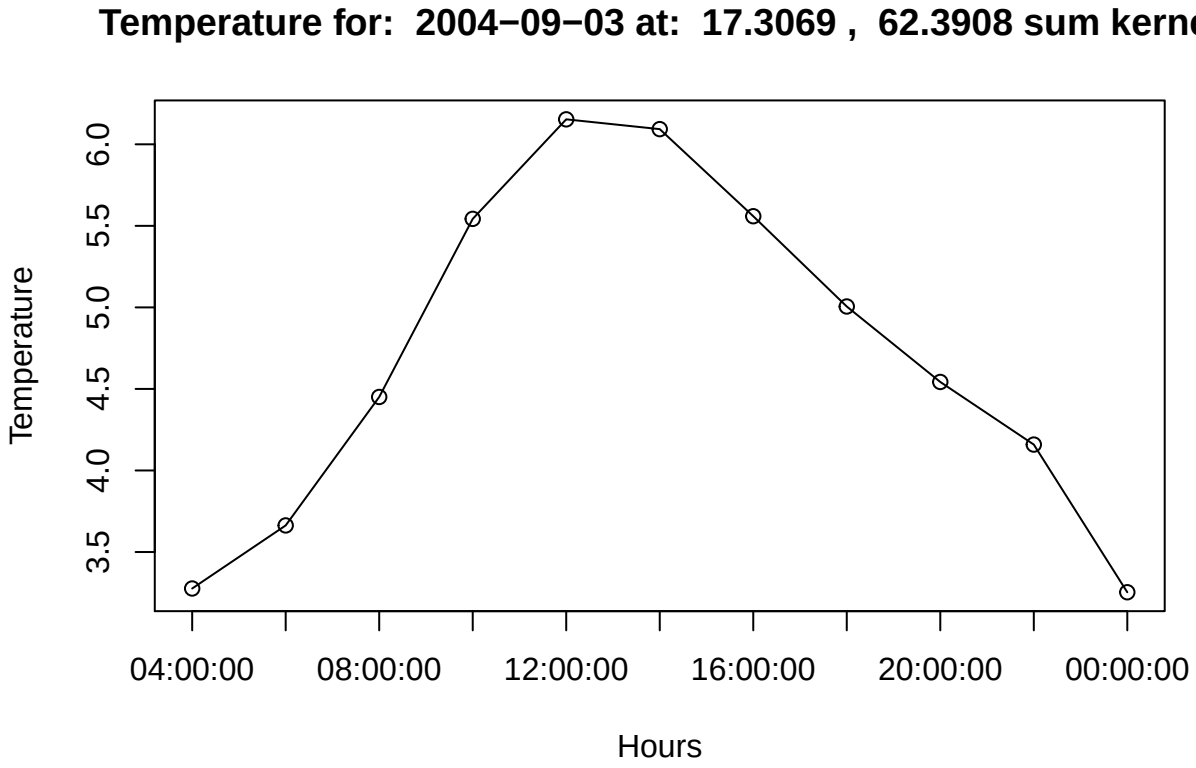
In the graph below is the kernel funtion applied on the date difference. The plot looks dramatically different form the previous one. This is because the data is ordered by the location and not by the date. Even though the kernel is applied correctly the data point is scattered as opposed to the distance difference kernel plot.



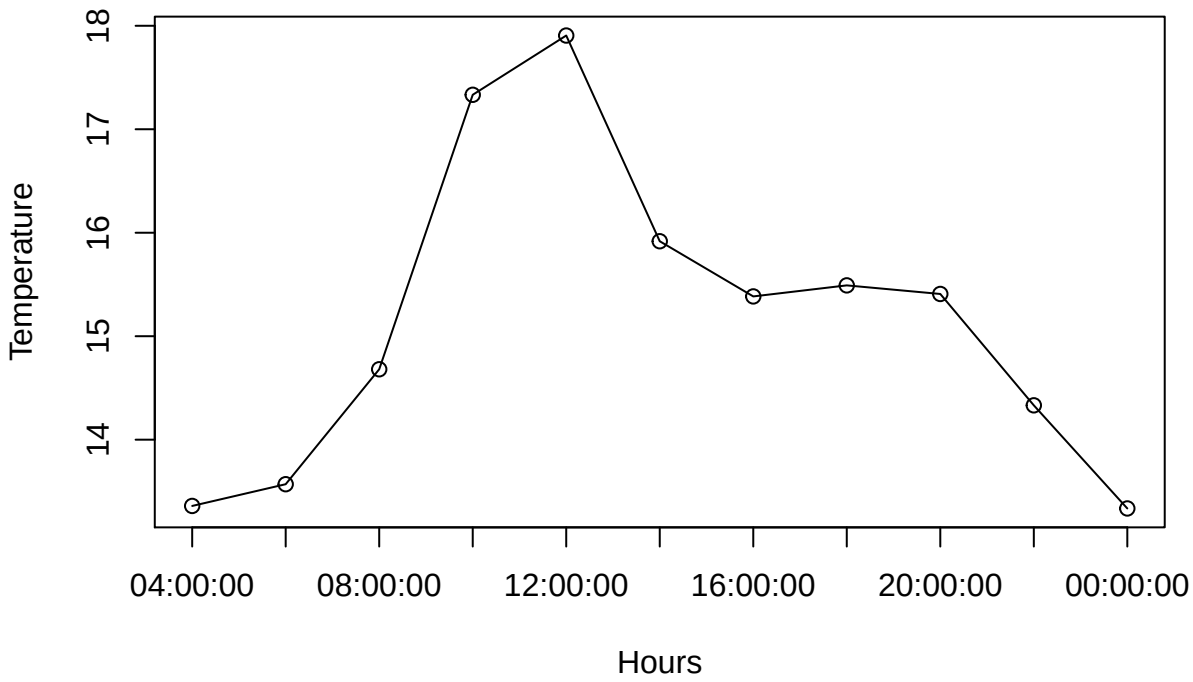
Below we can also see the kernel for the time difference, and it shows a very similar story to the date difference. The plot has "lines" because we have 24 hours in a day everyday opposing to the dates that are always different.



The plot that predicts the temperature for a given date in Sundsvall (where the author is from) by summarizing the kernels.

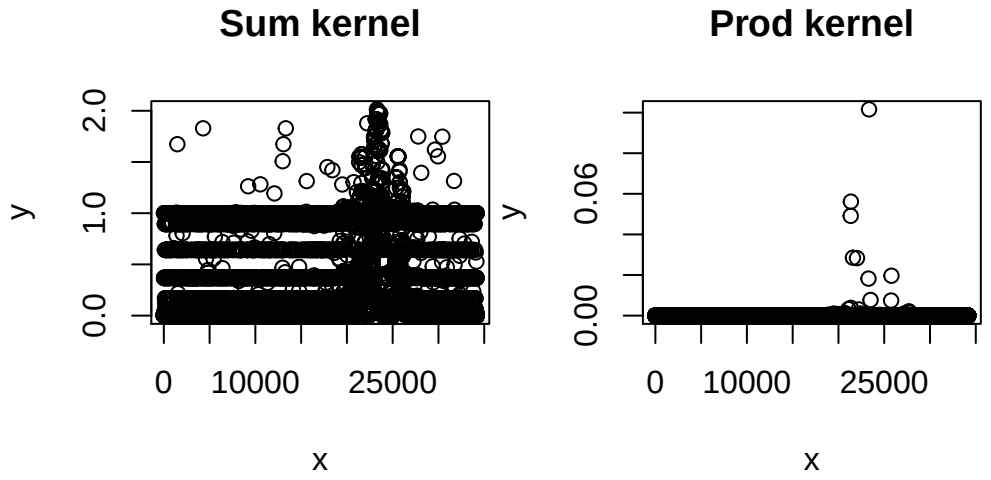


Finally the plot predicting the temperature for the location by multiplying the kernel is provided. By looking at the historical data (also provided by SMHI), multiplying the kernels gives a better prediction for the location of Sundsvall.



The difference between predictions can be easily explained by plotting the summarized and multiplied kernels next to each other. As seen below, summarized kernel, just added all of the point to one vector, that created

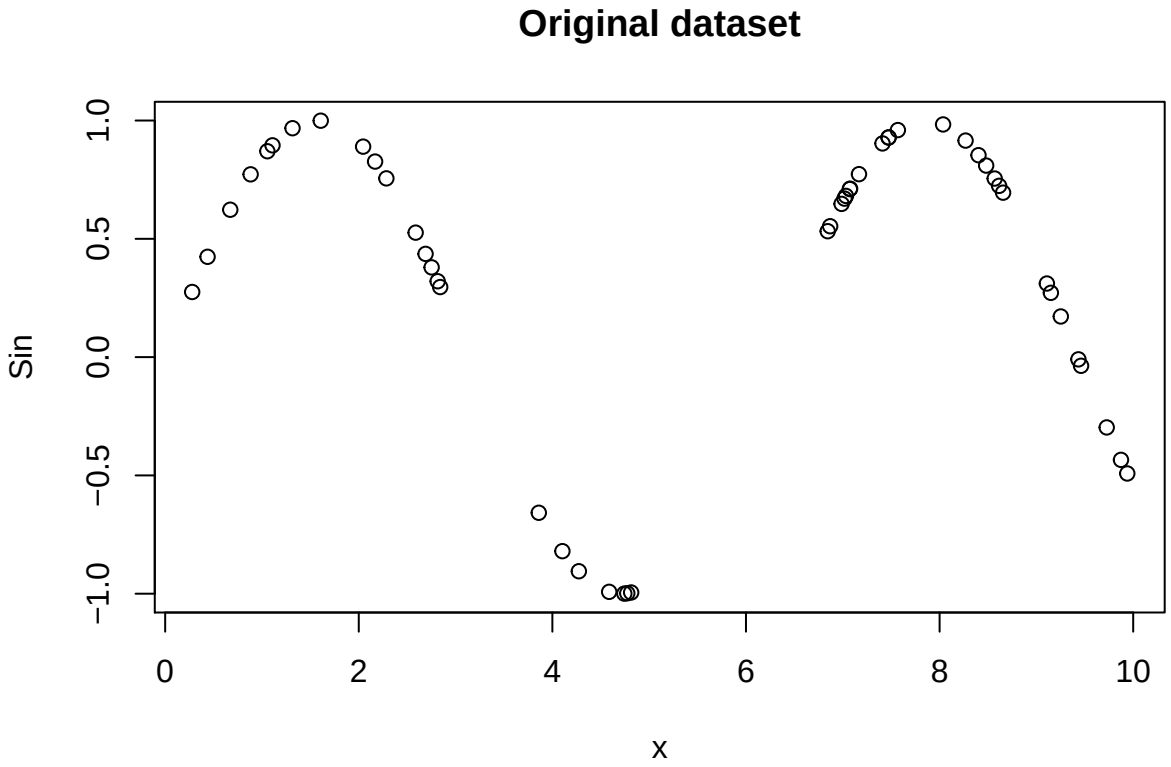
a non specific kernel. By multiplying the kernels, we "boost" the values that overlap in each kernel and the ones that do not are just 0 (because a*0 = 0) which makes kernel easier to predict with.



Assignment 3

In this assignment the task was to predict the sinus curve using a neural network with one hidden layer of 10 units using different thresholds for stopping the gradient descent.

We randomised the points using the given code and $runif$ function and then applied the sinus function on all of them, that created the original dataset shown in a plot below.



After the data has been obtained, we wanted to select the most appropriate threshold. Here we want to look at the lowest MSE value for the validation dataset, that we tested the model on. The MSE value for the training dataset is also provided to check if the model overfitts. As we can see the lowest MSE is given by the

threshold of 0.004. It gives the lowest MSE value for both the training and validation datasets. The starting weights in the network are random values between -1 and 1.

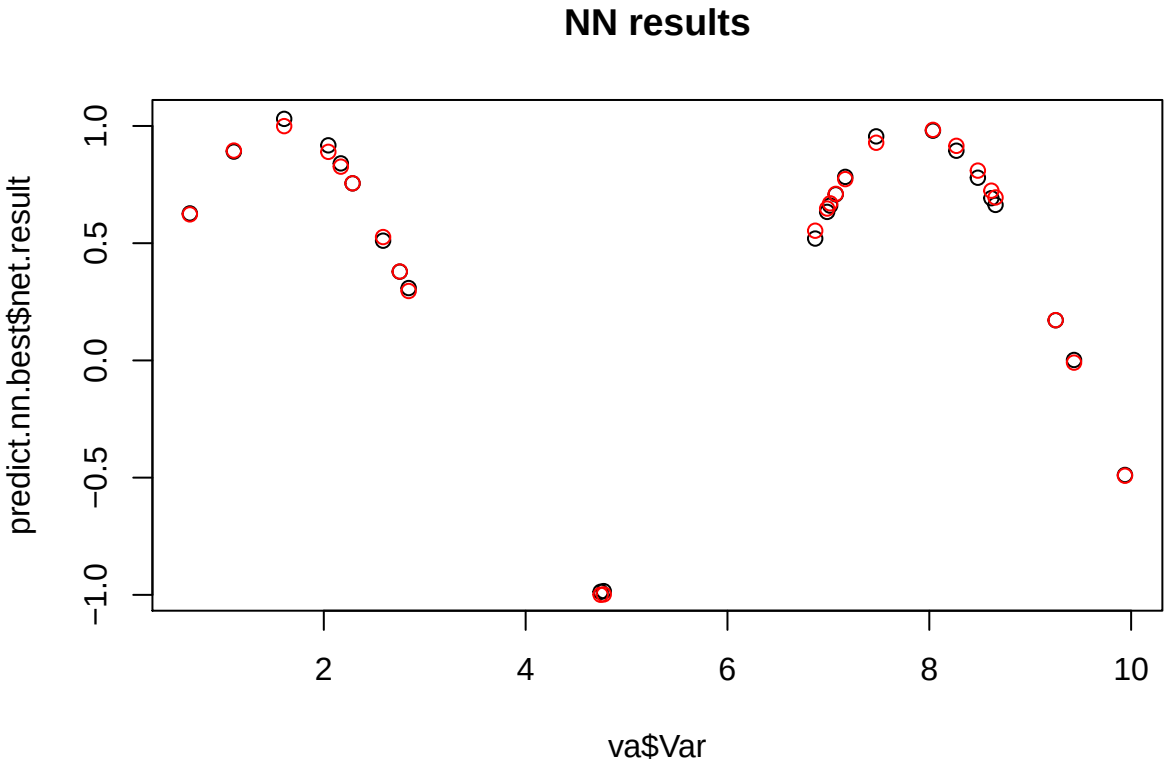
```
for(i in 1:10) {
  thr = i/1000
  threshold.vector[i] = thr
  nn <- neuralnet(tr$Sin~ tr$Var, threshold = thr, data = tr , hidden = 10, startweights =  weights)
    # Your code here
  prediction = compute(nn, va$Var)
  MSE.nn <- sum((va$Sin - prediction$net.result)^2)/length(va$Var)
  print(paste("Threshold: ",thr, "MSE: ", MSE.nn))
  threshold.error[i] = MSE.nn
}
```

```
## [1] "Threshold:  0.001 MSE:  0.000547076610626183"
## [1] "Threshold:  0.002 MSE:  0.000504967983047512"
## [1] "Threshold:  0.003 MSE:  0.000395367560223317"
## [1] "Threshold:  0.004 MSE:  0.000340035769727715"
## [1] "Threshold:  0.005 MSE:  0.000382218297736012"
## [1] "Threshold:  0.006 MSE:  0.000389748839647503"
## [1] "Threshold:  0.007 MSE:  0.000633570742848528"
## [1] "Threshold:  0.008 MSE:  0.000659700966225855"
## [1] "Threshold:  0.009 MSE:  0.000844996150831907"
## [1] "Threshold:  0.01 MSE:  0.00109436382144434"
```

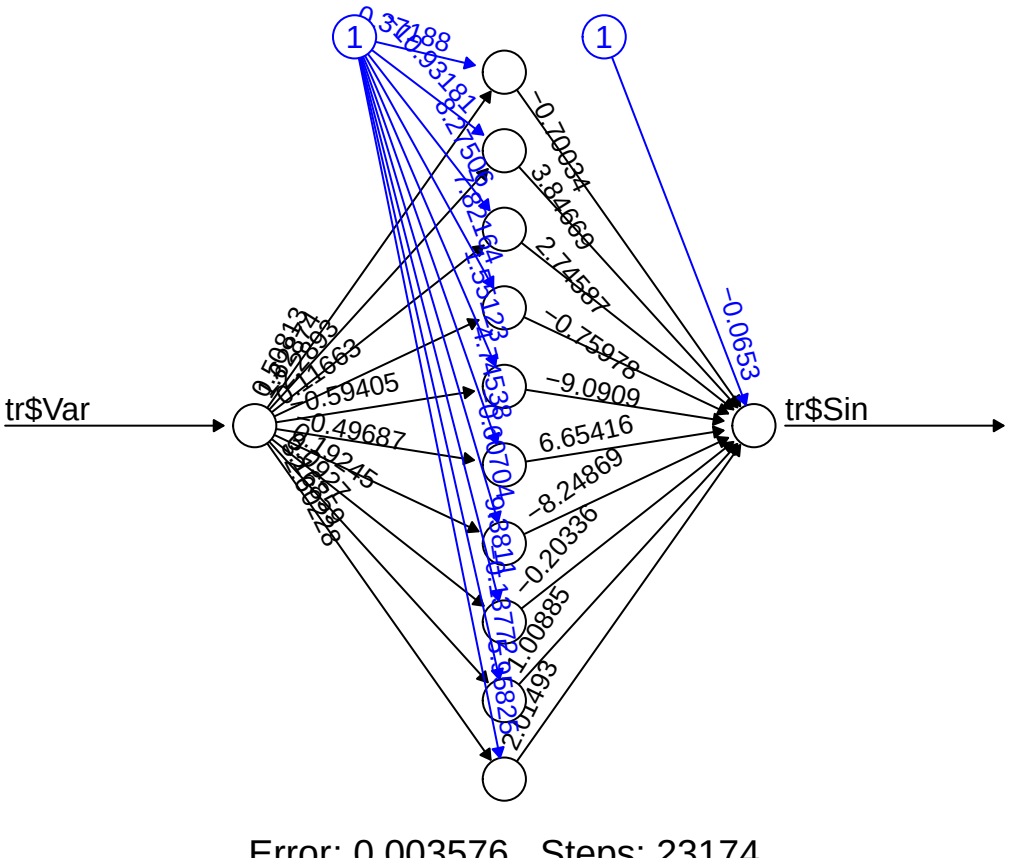
```
best.threshold = threshold.vector[which.min(threshold.error)]
```

After we have selected the threshold we want to plot the model against the original data, that is shown below. As we can see in the plot, the predicted values fit almost exactly in the plot with the original values.

```
## [1] "Selected threshold:  0.004  with MSE:  0.000340035769727715"
```



Shown below is the plot of the final model with the best threshold.



Code

Assignment 1

```
set.seed(1234567890)
library(geosphere)
stations <- read.csv("/Users/karolwojtulewicz/Google\ Drive/skola/TDDE01/Labs/Lab\ 3/Assignment\ 1/stati
temps <- read.csv("/Users/karolwojtulewicz/Google\ Drive/skola/TDDE01/Labs/Lab\ 3/Assignment\ 1/temps50
st <- merge(stations,temps,by="station_number")
#distHaversine
#density(c(-20, rep(0,98), 20))

h_distance <- 100*1000 # large distances in meters (this is why *1000)
h_date <- 30 #30 days in a month?
h_time <-4 # because we have 2h intervalls
#Tried to explain smoothing factors logically, but is not
#place.to.predict <- c(14.826, 58.4274)# The point to predict (up to the students)
#place.to.predict = c(20.2253, 67.8558) #kiruna
#place.to.predict = c(18.0686, 59.3293) # stockholm
place.to.predict = c(17.3069, 62.3908) #sundsvall
#date <- "2013-12-04" # The date to predict (up to the students)
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00", "16:00:00", "18:00:0
temp <- vector(length=length(times))
```

```r
# Students' code here
mydate = "2013-01-14"
mydate = "2004-09-03"

selected.dates = subset(st, as.Date(st$date) < as.Date(mydate))

get.diff.distance.smooth = function(data, point.coord,h){
  return.vector = 1:length(data[,1])
  for (i in 1:length(data[,1])) {
    x = distHaversine(c(data$longitude[i], data$latitude[i]), point.coord)/h
    return.vector[i] = exp(-x^2)
  }
  return(return.vector)

}

get.diff.date.smooth = function(data, day.of.intresst, h){
  return.vector = 1:length(data[,1])
  for( i in return.vector){
    x = as.numeric(as.Date(day.of.intresst)-as.Date(data$date[i]))/h
    return.vector[i] = exp(-(x)^2)
  }
  print(head(return.vector))
  return(return.vector)
}
get.diff.time.smooth= function(data, mytime,h){

  return.vector = 1:length(data[,1])
  for( i in return.vector){
    time1 = as.POSIXct(as.character(data$time[i]), format = "%H:%M:%S", tz = "UTC")
    time2 = as.POSIXct(as.character(mytime), format = "%H:%M:%S", tz = "UTC")
    #apply smoothing factor here? Wonder if it's correct...
    x = (time1-time2)/h
    return.vector[i] = exp(-as.numeric(x)^2)
    }
  return(return.vector)
}
diff.distance = get.diff.distance.smooth(selected.dates, place.to.predict, h_distance)
plot(1:length(diff.distance), diff.distance)
diff.date = get.diff.date.smooth(selected.dates, mydate, h_date)
plot(1:length(diff.date), diff.date)
diff.time.vector = 1:length(selected.dates$time)

for(i in 1:length(times)) {
  #comp_time = convert_to_hours(times[i])
  diff.time.vector = get.diff.time.smooth(selected.dates,times[i], h_time)

  sum.kernels = diff.time.vector+diff.distance+diff.date
  prod.kernels = diff.time.vector*diff.distance*diff.date
  temp[i] = sum(sum.kernels*selected.dates$air_temperature)/sum(sum.kernels)
}
plot(1:length(diff.time.vector), diff.time.vector)
```

```r
plot(temp, type = "o", xlab = "Hours", ylab = "Temperature", xaxt="n", main = paste("Temperature for: "
axis(1, at=1:length(temp), labels=times)
```

Assignment 3

```r
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
plot(trva$Var, trva$Sin)
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# Random initialization of the weights in the interval [-1, 1]

threshold.vector = seq(0,10,by=1)
threshold.error = seq(0,10,by=1)
weights = runif(31,-1,1)

for(i in 1:10) {
  thr = i/1000
  threshold.vector[i] = thr
  nn <- neuralnet(tr$Sin~ tr$Var, threshold = thr, data = tr , hidden = 10, startweights =  weights)
    # Your code here
  prediction = compute(nn, va$Var)
  MSE.nn <- sum((va$Sin - prediction$net.result)^2)/length(va$Var)
  print(paste("Threshold: ",thr, "MSE: ", MSE.nn))
  threshold.error[i] = MSE.nn
}

best.threshold = threshold.vector[which.min(threshold.error)]

nn.best <- neuralnet(tr$Sin~ tr$Var, threshold = best.threshold, data = tr , hidden = 10,
startweights = weights )

predict.nn.best = compute(nn.best, va$Var)

print(paste("Selected threshold: ", best.threshold, " with MSE: ",
sum((va$Sin - predict.nn.best$net.result)^2)/length(va$Var)))

# Plot of the predictions (black dots) and the data (red dots)
plot(va$Var, predict.nn.best$net.result, main = "NN results")
points(va$Var, va$Sin, col = "red")
plot(nn.best, rep = "best") # for r markup nn plot
```