TDDE01 Lab3

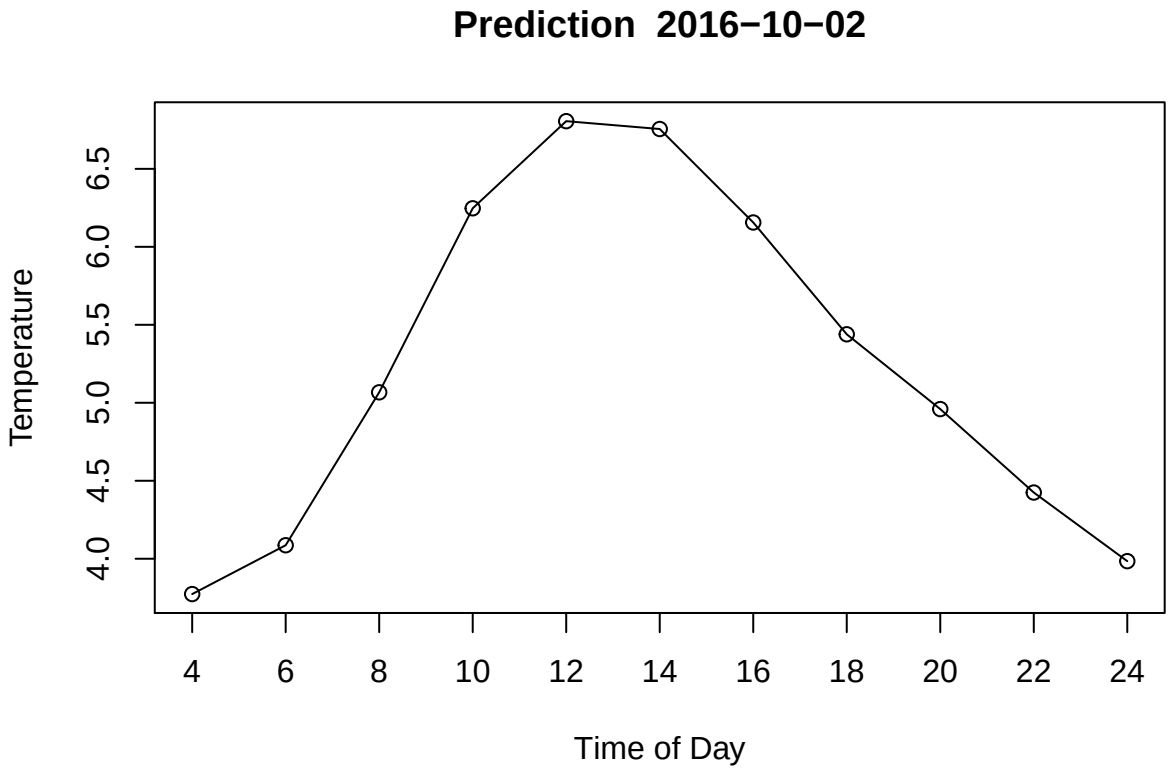
Teodor Ganestål (teoga849) and Andreas Ryefalk (andry954)

10 December 2019

1. Kernel Methods

```
set.seed(1234567890)
library(geosphere)

stations <- read.csv("stations.csv", header = TRUE, fileEncoding = "ISO-8859-1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

#smoothing coefficients
h_distance <- 10000
h_date <- 7
h_time <- 2

#Danmark
a <- 59.832979
b <- 17.744537

date <- "2016-10-02" # 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", "18:00:00", "20:00:00", "22:00:00", "24:00:00")

temp <- vector(length=length(times))

#gaussian kernel
k = function(u) exp(-(u**2))

#distance in
u.distances = geosphere::distHaversine(c(a, b), st[,4:5])

same_year = function(X) gsub('[0-9]{4}', "2000", X)

#distance of dates in days
d_calc = function(x) min(366 - x, x)
u.days = sapply(as.numeric(difftime(same_year(date), same_year(st$date)), units = "days") %% 366, d_cal

#dinstance in time in hours (abs)
t_calc = function(t1, t2) min((t1 - t2) %% 24, (t2 - t1) %% 24)
measured_times = as.numeric(as.difftime(sapply(st$time, toString), units = "hours"))


for (i in seq(1:length(times))) {
  time1 = as.numeric(as.difftime(times[i], units = "hours"))
  u.hours = sapply(measured_times, function(t) t_calc(time1, t))
  kernel = k(u.distances/h_distance) + k(u.days/h_date) + k(as.numeric(u.hours)/h_time)
```

```
  temp[i] = sum(kernel * st$air_temperature) / sum(kernel)
}

plot(y=temp, x=as.numeric(as.difftime(times, units = "hours")), type="o", xlab = "Time of Day", ylab="T
axis(side=1, at=as.numeric(as.difftime(times, units = "hours")))
```



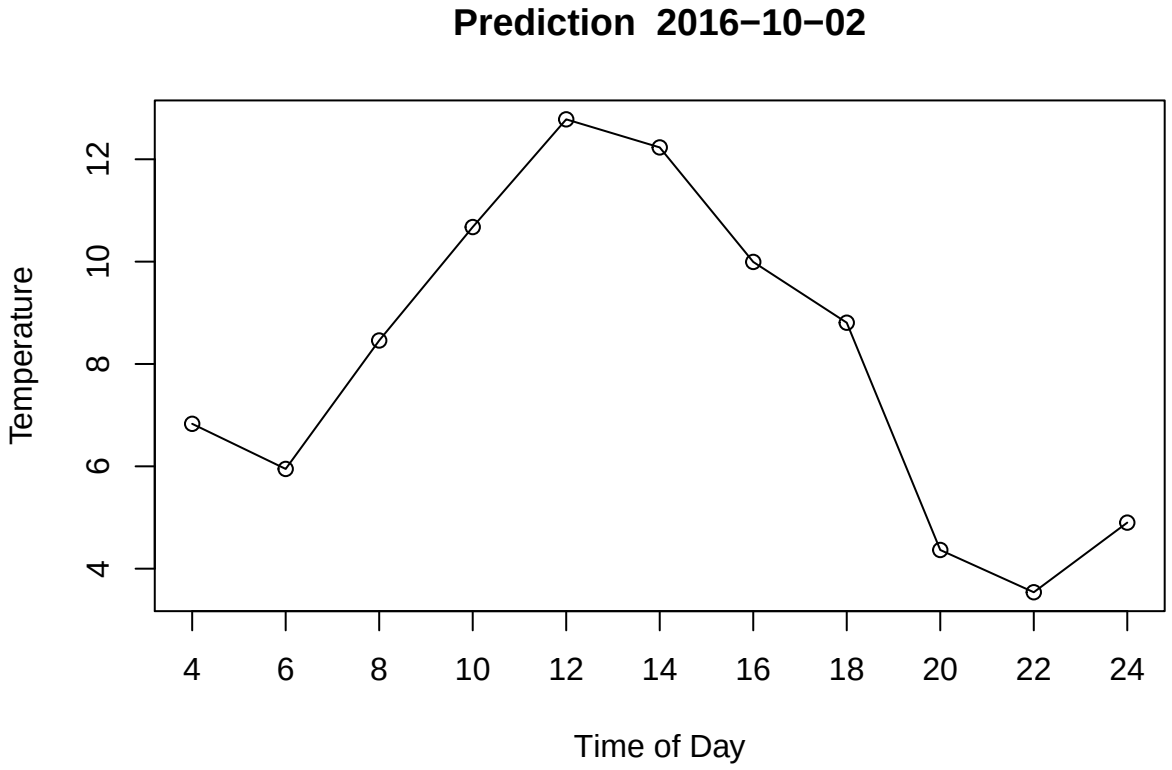
Show that your choice for the kernels' width is sensible, i.e. that it gives more weight to closer points. Discuss why your of definition of closeness is reasonable.

The smoothing factor can be seen as "the value where data is close", i.e. where the kernel value becomes $e^{-1} \approx 0.3678794$, and everyhing closer than that, contributes even more.

The smoothing factors where chosen as the following. 10000m for distance as the weather is usualy pretty similar for 1mil but beond that it will start to differ. The date was chosen as 7 days as the weather one week previous has a high probability to continue for the next day. Lastly hours were choosen as 2 hours as beyond that we will start to flatten the curve tou much with the cooler temperatures.

Instead of combining the three kernels into one by summing them up, multiply them. Compare the results obtained in both cases and elaborate on why they may differ.

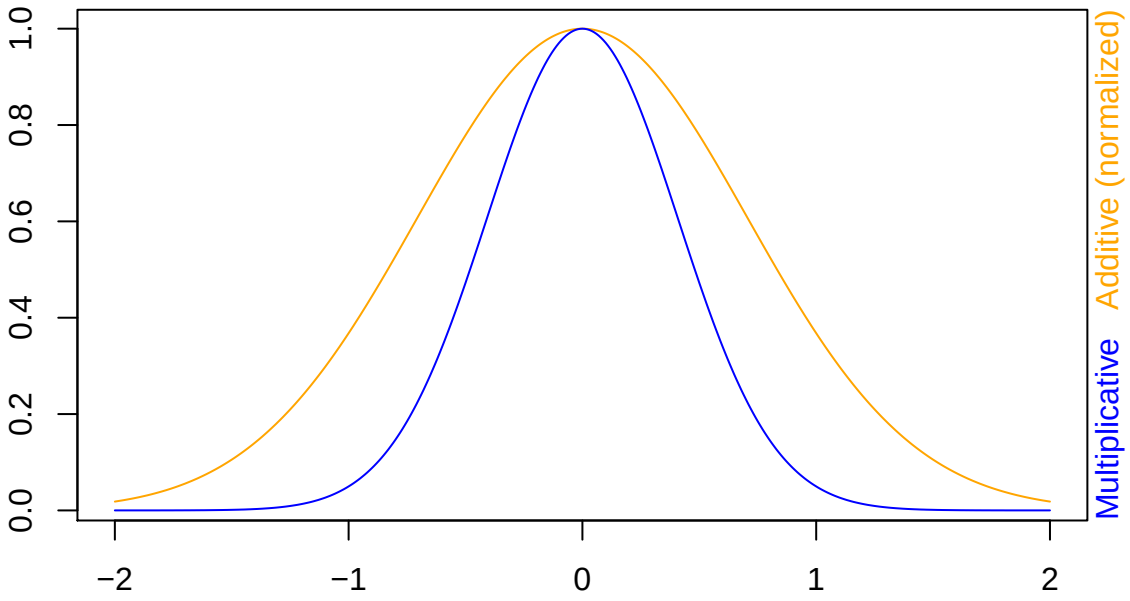
```
for (i in seq(1:length(times))) {
  time1 = as.numeric(as.difftime(times[i], units = "hours"))
  u.hours = sapply(measured_times, function(t) t_calc(time1, t))
  kernel = k(u.distances/h_distance) * k(u.days/h_date) * k(as.numeric(u.hours)/h_time)
  temp[i] = sum(kernel * st$air_temperature) / sum(kernel)
}

plot(y=temp, x=as.numeric(as.difftime(times, units = "hours")), type="o", xlab = "Time of Day", ylab="T
axis(side=1, at=as.numeric(as.difftime(times, units = "hours")))
```



We can clearly see a better performance for the multiplicative kernels, they give values much closer to the actual result (looked it up in old weather data).

When multiplying kernels we get a much steeper peak than when adding them, and added kernels instead get a slightly fatter tail. This means that the multiplied kernel is more strict, and a data point needs to be even closer to make an impact on the prediction. As such, we see the predicted data with the first kernel being more smoothed out, as it gives more wight to data further away. The second prediction, in contrast, varies more since it only considers very close data points, and thus is more fitted to those points specificly. See below the difference between adding and multiplying three Gaussian kernels. The added kernel is normalized to 1 in the graph to clearer display the difference in width to the multiplied kernel.

```
xval = seq(-2, 2, 0.01)
plot(xval, (k(xval) + k(xval) + k(xval)) / 3, type="l", xlab = "", ylab = "", col="orange" )
lines(y=k(xval) * k(xval) * k(xval), x=xval, col="blue")
mtext("Additive (normalized)", col = "orange", side = 4, adj = 1)
mtext("Multiplicative", col = "blue", side = 4, adj = 0)
```

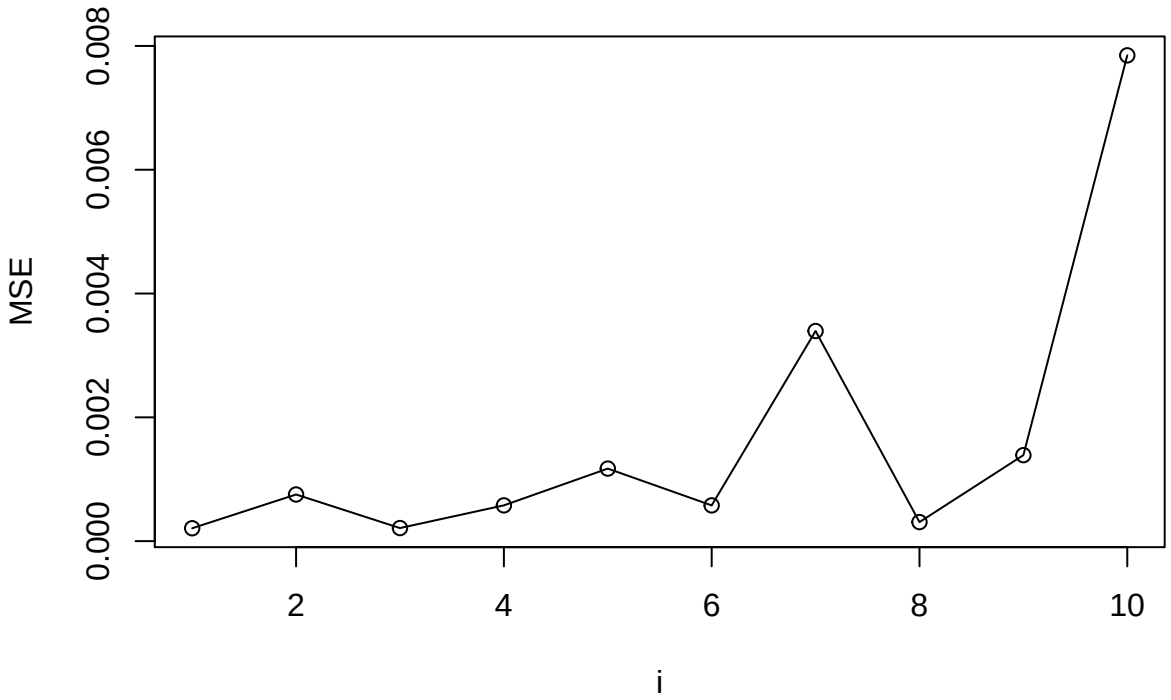


As we can se in this graph, the additive kernel has a much higher h value than the multiplicative one. Therefore it will produce a much more biased graph, which we don't want in this case.

3. Neural Networks

The threshold value was chosen by comparing Mean Square Errors (MSE) of the thresholds $i/1000$ with $i = 1, ..., 10$ as seen in the graph below. Using the threshold $1/1000$ we got the lowest MSE of 0.000208343.

```
library(neuralnet)
set.seed(1234567890)

Var = runif(50, 0, 10)
trva = data.frame(Var, Sin=sin(Var))
train = trva[1:25,] # Training
valid = trva[26:50,] # Validation

# Random initialization of the weights in the interval [-1, 1]
winit = runif(10, -1, 1)

mse <- function(X, X1) mean((X-X1)**2)

MSEs = vector(length=10)
for(i in 1:10) {
  nn = neuralnet(Sin~Var, train, threshold = i/1000, startweights = winit, hidden = 10)
  MSEs[i] = mse(predict(nn, valid), valid$Sin)
}
best.i = which.min(MSEs)

plot(MSEs, ylab="MSE", xlab = "i", type="o")
```

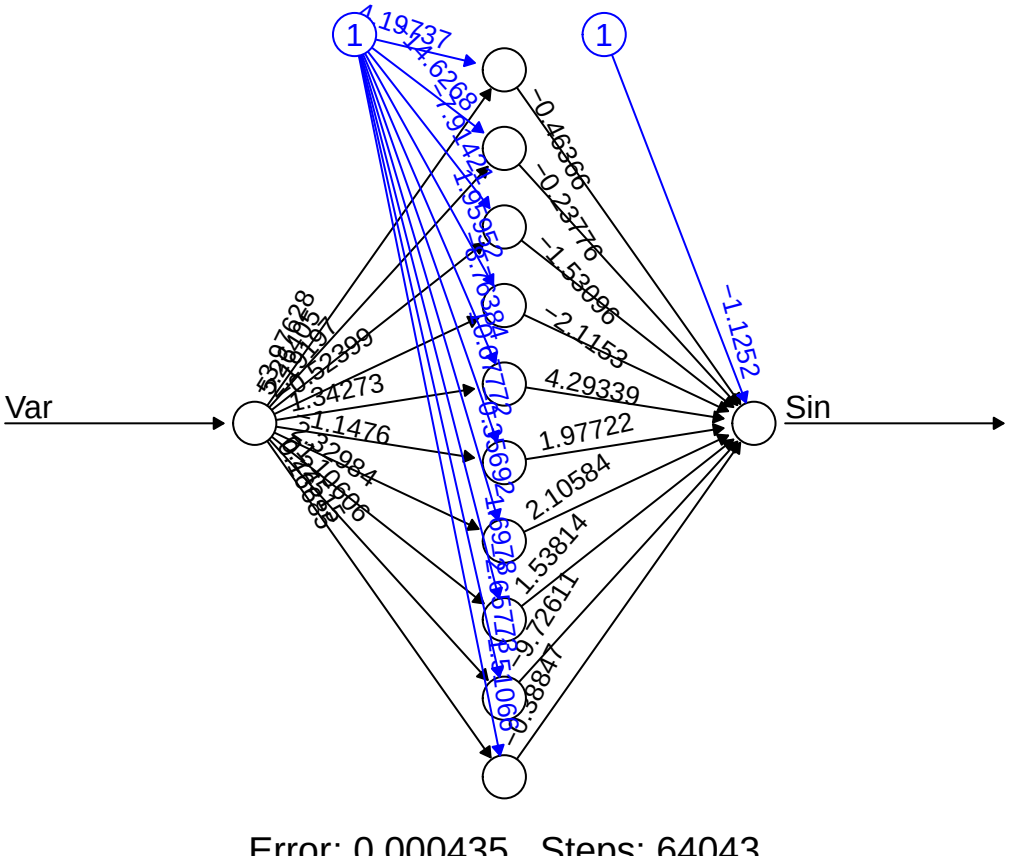



```
print(paste("Best threshold:", best.i, "/ 1000 ( MSE =", min(MSEs), ")"))
```

```
## [1] "Best threshold: 1 / 1000 ( MSE = 0.000208342955645015 )"
```

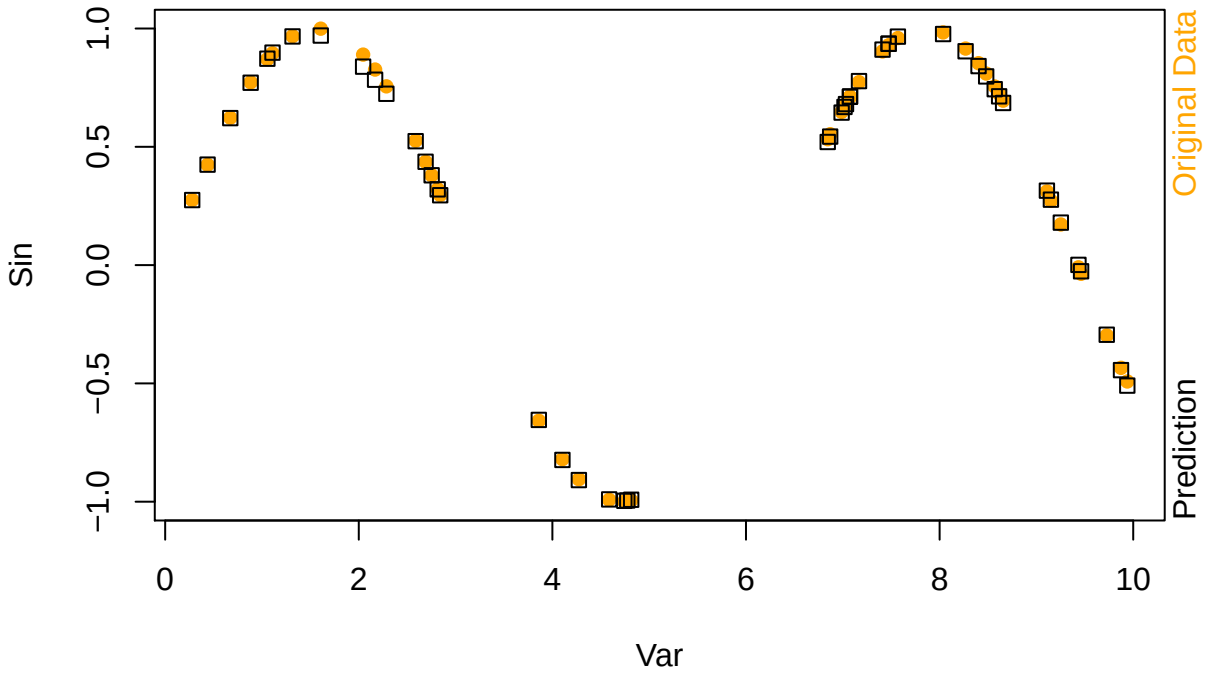
The final Neural Network is presented below.

```
nn = neuralnet(Sin~Var, train, threshold = best.i/1000, startweights = winit, hidden = 10)

plot(nn, rep="best")
```



The predicted data is shown below.

```
xfit = seq(0, 10, .2)

plot(trva, col = "orange", pch=16)
points(trva$Var, predict(nn, trva), pch=0)
mtext("Original Data", col = "orange", side = 4, adj = 1)
mtext("Prediction", side = 4, adj = 0)
```


Sin
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
Var
Original Data
Prediction