Lab3.Report

Arun Uppugunduri

2019-12-17

Assignement 1 - Kernel Methods

In this assignment we were to use three gaussian kernels in order to predict weather. The weather is predicted using three different gaussian kernels. In the task we chose a location to predict and 11 different times to predict 04:00:00, 06:00:00…. 24:00:00. The predicted temperature is calculated by either summing or multiplying the values from the kernels. Each kernels give a weight to based on the difference in distance, days and hours.

For the calcualtions o each kernel we select a smoothing factor h. These are used to adjust how much wight we want to give each measure. Below can be seen the values for the smoothing factors h, date which is predicted and longitude/latitude of location to be predicted.

a <- 18.0632240  
b <- 59.334591   
location = data.frame(a,b) # Longitude/Latitude for the city of Stockholm  
  
mydate = "2016-01-14"  
  
h\_date <- 4  
h\_time <- 5  
h\_distance <- 90000

To make the predictions more reasonable we do not want to base any predictions on data from dates which are posterior or selected date of prediction. We therefore begin by creating a new dataset only holding data up to our selected date.

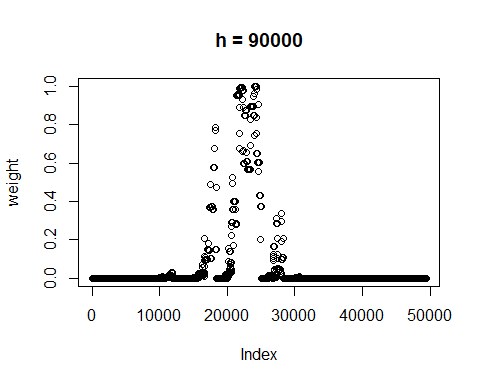
selected.dates = subset(st, as.Date(st$date) <= as.Date(mydate))

Thinking logically we do not want data measures from a location far a way to give a big weight to our prediction. A lower value on h results in a lower bias and higher varance. A value on h results in a higher bias but lower varance. This is due to the gaussian kernel formula. A lower value on h will make the kernel value smaller while bigger values on h will make the kernel value bigger. The same idea applies to day and hours. Below can be seen the function which is applied to each kernel.

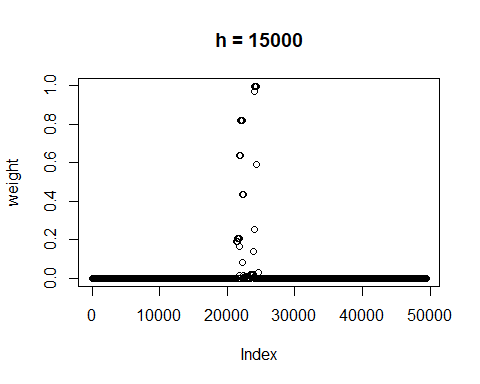
kernel (u) = e^(-u^2)  
u = (Xi - x ) / h

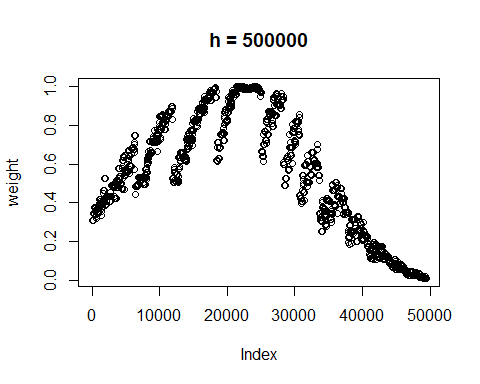
Distance Kernel Below can be seen the functions for the different gaussian kernels. The distance kernel below uses the function distHaversine to calcualate the difference in distance from our location to all other datapoints based on their longitude/latitude.

#Distance Kernel  
h = 9000  
kernel\_distance = function(Xn, x, h){  
 u = distHaversine(data.frame(Xn$longitude, Xn$latitude), x)/h  
 k = exp(-u^2)  
 return (k)  
}

Plotting the kernel with h = 90000 gives us the following apperance. We want to choose a value on h which takes data which is around the stockholm area but not too far away. 

If we compare to see what would happen when changing the h values to higher and lower, h = 500000, h = 15000 we see that the lower value has a very small amount of data points. This means that we choose to only give value to data measures that are failry close to our location. Comparing this to h = 500000 we can see that there are alot of data points meaning that we choose that even though the distance is big, we still want it to give weight to our model.

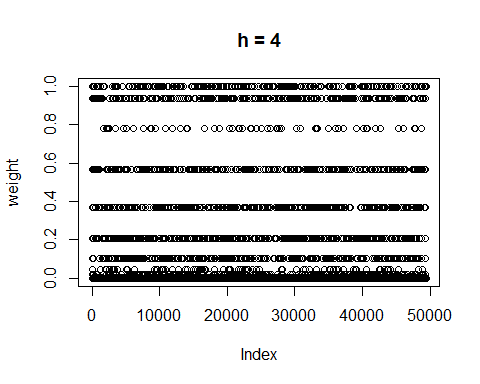


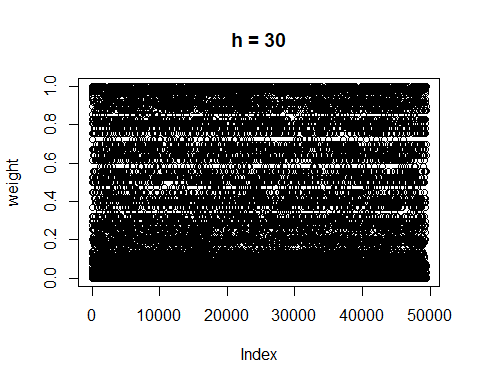


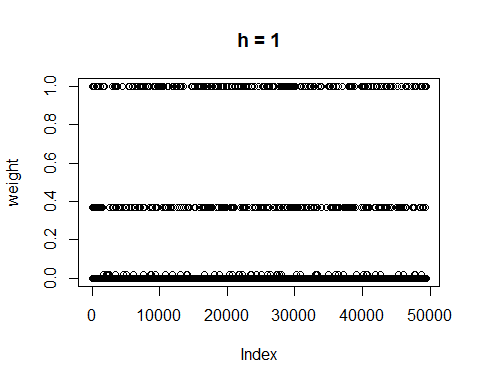
Date kernel Next we created our function for the date kernel. It is basically the same calculations as before but witht the difference that we adjust the amount of days. If the amount of days is bigger than 183 we adjust the difference to be (365 - difference). This seemed to be reasonable since for instance January and December weather should have impact on each other.

#Date kernel  
h\_date = 4  
date = as.Date(mydate)  
kernel\_day = function(date, selected.dates, h\_date){  
 u = difftime(date, selected.dates$date)  
 u = as.numeric(u)  
 u = u%%365  
 diff = 365-u  
 u = ifelse(u>183,diff,u)  
 u = u/h\_date  
 k = exp(-u^2)  
 return(k)  
}

Plotting the kernel and applying it on the date difference gives the plot seen below. We see that the plot appears differently. According to how the model was built all data that has the same difference in days is given the same weight. This means that the model does not see it differently which year the measure was done. This is probably not optimal



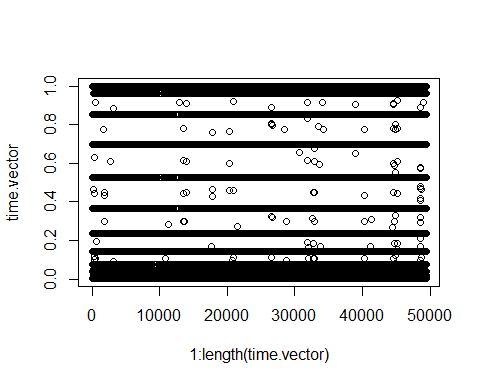
As before we can see the impact that h has on the plot. A low value on h only gives weight to measures made very close to our selected date while a bigger h takes more data points into account 



Time Kernel Lastly we create our time kernel. As before we adjust the difference since it seemed reasonable that for example hours 23.00 and 01.00 should be related.

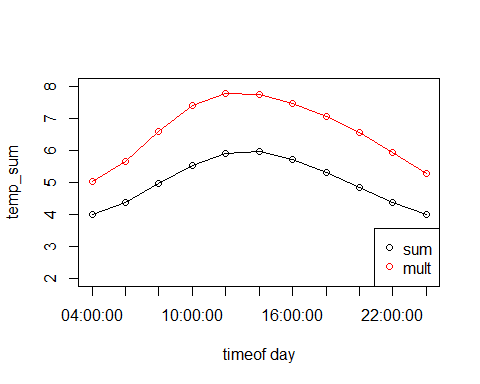
#Time Kernel  
h\_time = 5  
kernel\_hours = function(selected.dates, times){  
 time\_diff = difftime(strptime(selected.dates$time, format = "%H:%M:%S"),  
 strptime(times[1], format = "%H:%M:%S"), units = "hours")  
 time\_abs = abs(time\_diff)  
 fix\_time = 24-time\_abs  
 fixtime =as.numeric(fix\_time)  
 time\_diff = ifelse(time\_abs > 12, fix\_time, time\_abs)  
 u = time\_diff/h\_time  
 k = exp(-u^2)  
return(k)  
}

Below can be seen the plotted kernel over when summed over all hours. It has a very similar apperance to the previous time vector. This is since the vast majority will always have multiples regarding the difference in time, hence getting the same weight.



Temperature prediction When all kernels are done we can make the prediction. The predictions where to be made by first summing the the kernels and then by multiplying the kernels.

time.vector = c()  
temp\_sum = c()  
temp\_mult =c()  
for(i in 1:11){  
 time.vector = kernel\_hours(selected.dates, times[i])  
 sum.kernels = time.vector + distance + day\_kernel  
 product.kernels = time.vector\*distance\*day\_kernel  
 temp\_sum[i] = sum(sum.kernels\*selected.dates$air\_temperature)/sum(sum.kernels)  
 temp\_mult[i] = sum(product.kernels\*selected.dates$air\_temperature)/sum(product.kernels)  
   
}

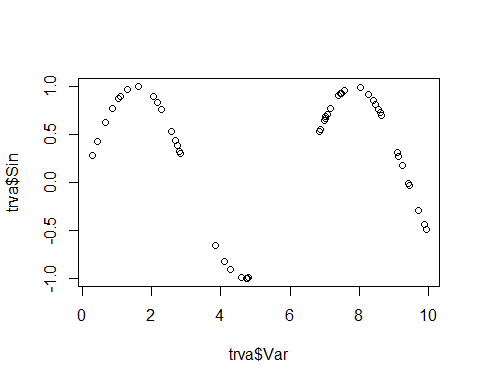
Below can be seen the graphs for the summed vs multiplied kernel values 

Assignment 3

In this assingment we create a neural network. The task was to predict a sinus curve by using the neural network with a hidden layer of 10 units. The model is selected by trying the model with different values for the threshold ranging from 1/1000 to 10/1000. By using the supplied code we create 50 randomised points. We then create a data set where the sinus value of this value is created. The aim is to create a neural network which can predict this function without knowing that sinus has been applied to the value. The data set is then split in half testing and validation set.

Plotting the data set, values and sin(values) gives us the following plot

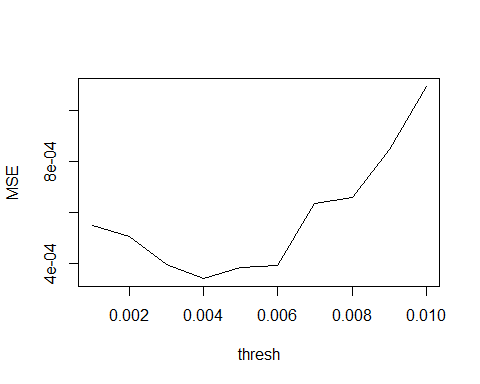
plot(trva$Var, trva$Sin)

 When data has been created we want to create the neural model and find what threshold gives us the lowest mean square error. The model is created using the test data “tr” and based on this we make a prediction. The prediction is done on the validation data and gives us the predicted sinus value. Based on the prediction we calculate the mean square error and this is stored in a vector. Finding which value in the vector holds the lowest MSE value we get the best threshold value.

The group assignment was a completion since the calculation of Mean Square Error was incorrect. In the report the calculation was made by taking the mean of the squared values difference. Mean(Prediction.val^2 – va$Sin^2). This resulted in an incorrect MSE value and consequently also an incorrect threshold. I had however not made this error in my individual report.

for(i in 1:10){  
 thresh = i/1000  
 thresh.vector[i] = thresh  
 nn = neuralnet(Sin~ Var, threshold = thresh, data = tr, hidden =c(10), startweights = winit)  
 prediction.val = predict(nn, va)  
 MSE = mean((prediction.val - va$Sin)^2)  
 thresh.error[i] = MSE  
}  
bestthresh = thresh.vector[which.min(thresh.error)]

The code results in bestthresh = 0.004. Plotting the threshold values against the resulting MSE gives the following which further confirmst that the lowest MSE is given by threshold 4/1000.



When the threshold value has been obtained we repeat the process. Creating the neural model but using the optimal threshold value. Again a prediction is done on the validation data

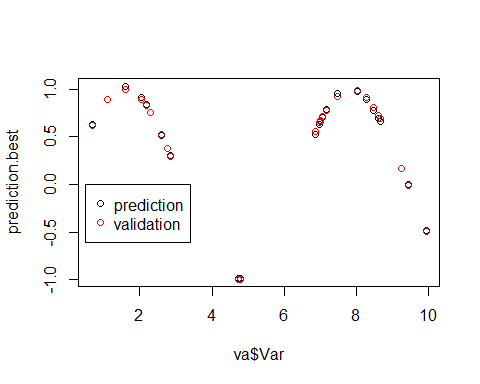
nnbest = neuralnet(Sin~ Var, threshold = bestthresh, data = tr, hidden =c(10), startweights = winit)  
prediction.best = predict(nnbest, va)  
MSE\_best = mean((prediction.best - va$Sin)^2)

print(paste("Best Threshold:", bestthresh, "MSE:", MSE\_best))

## [1] "Best Threshold: 0.004 MSE: 0.000340035769727715"

plotting the neural model gives us the following. We can see the 10 layers and the weight which is given to each layer. Plotting the prediction against the original validation data we see that the

Plotting the predicted sinus values against the validation data we see that the model predicts well. The data has the form of a sinus curve and is also more or less identical to the valiadiation points.



RNGversion('3.5.1')  
  
set.seed(1234567890)  
library(geosphere)  
stations <- read.csv("stations.csv", fileEncoding="latin1")  
temps <- read.csv("temps50k.csv")  
st <- merge(stations,temps,by="station\_number")  
  
# The point to predict (up to the students)  
a <- 18.0632240  
b <- 59.334591   
location = data.frame(a,b)  
   
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","00:00:00")  
mydate = "2016-01-14"  
  
#Filter dataset to not include dates that are after the time of prediction  
selected.dates = subset(st, as.Date(st$date) <= as.Date(mydate))  
  
  
#med data.frame behöver man inte köra for loopen utan räcker med  
#distance2 = kernel\_distance(selected.dates, location, h\_distance)  
#for( i in 1:nrow(selected.dates)){  
 # distance[i] = kernel\_distance(selected.dates[i,], location, h\_distance)  
#return(distance)  
#}  
  
#gaussan kernel  
kernel\_distance = function(Xn, x, h){  
 u = distHaversine(data.frame(Xn$longitude, Xn$latitude), x)/h  
 k = exp(-u^2)  
 return (k)  
}  
  
h\_distance <- 90000  
distance = kernel\_distance(selected.dates, location, h\_distance)  
plot(distance, xlab = "distance", ylab = "weight")  
  
  
  
#Date kernel  
date = as.Date(mydate)  
kernel\_day = function(date, selected.dates, h\_date){  
 u = difftime(date, selected.dates$date)  
 u = as.numeric(u)  
 u = u%%365  
 diff = 365-u  
 u = ifelse(u>183,diff,u)  
 u = u/h\_date  
 k = exp(-u^2)  
 return(k)  
}  
h\_date <- 4  
day\_kernel = kernel\_day(date, selected.dates, h\_date)  
plot(day\_kernel, xlab = "dates", ylab = "weight")  
  
  
  
#Time Kernel  
kernel\_hours = function(selected.dates, times, h\_time){  
 time\_diff = difftime(strptime(selected.dates$time, format = "%H:%M:%S"),  
 strptime(times[1], format = "%H:%M:%S"), units = "hours")  
 time\_abs = abs(time\_diff)  
 fix\_time = 24-time\_abs  
 fixtime =as.numeric(fix\_time)  
 time\_diff = ifelse(time\_abs > 12, fix\_time, time\_abs)  
 u = time\_diff/h\_time  
 k = exp(-u^2)  
return(k)  
}  
h\_time <- 5  
time\_kernel = kernel\_hours(selected.dates, times, h\_time)  
plot(time\_kernel, xlab = "time", ylab = "weight")  
  
time.vector = c()  
temp\_sum = c()  
temp\_mult =c()  
for(i in 1:11){  
 time.vector = kernel\_hours(selected.dates, times[i], h\_time)  
 sum.kernels = time.vector + distance + day\_kernel  
 product.kernels = time.vector\*distance\*day\_kernel  
 temp\_sum[i] = sum(sum.kernels\*selected.dates$air\_temperature)/sum(sum.kernels)  
 temp\_mult[i] = sum(product.kernels\*selected.dates$air\_temperature)/sum(product.kernels)  
   
}  
plot(1:length(time.vector), time.vector)  
  
plot(temp\_sum, type ="o", ylim = c(2, 6), xlab = "timeof day", xaxt ='n')  
axis(1, at=1:length(times), labels =times)  
points(temp\_mult, type = "o", col = "red")  
legend("bottomright", col = c("black", "red"), legend = c("sum", "mult"), pch = c(1, 1))  
  
  
  
  
##ASSIGNMENT 3  
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  
thresh.vector = rep(0,10)  
thresh.error = rep(0,10)  
#Random varaiable initialization of the weights in the interval [-1,1]  
winit = runif(31, -1,1)  
  
  
  
  
MSE = function(predcition, observation){  
 return(mean(predcition-observation)^2)  
}  
for(i in 1:10){  
 thresh = i/1000  
 thresh.vector[i] = thresh  
 nn = neuralnet(Sin~ Var, threshold = thresh, data = tr, hidden =c(10), startweights = winit)  
 prediction.val = predict(nn, va)  
 MSE = mean((prediction.val - va$Sin)^2)  
 thresh.error[i] = MSE  
 #   
}  
bestthresh = thresh.vector[which.min(thresh.error)]  
which.min(thresh.error)  
# i = 4 => thresh = 1/1000  
plot(thresh.vector, thresh.error, xlab ="thresh", ylab = "MSE", type ="l")  
axis(1, at=(1:10), labels = thresh.vector)  
  
nnbest = neuralnet(Sin~ Var, threshold = bestthresh, data = tr, hidden =c(10), startweights = winit)  
prediction.best = predict(nnbest, va)  
MSE\_best = mean((prediction.best - va$Sin)^2)  
  
  
 plot(nnbest)  
 #plot the prediction, black dots and the data red dots  
 plot(va$Var, prediction.best)  
 points(va$Var, va$Sin, col ="red")