Machine Learning Lab 3

Assignment 1 - Kernel methods

This assignment was to implement a kernel method in order to predict the hourly temperatures for a date and place in Sweden. Data about weather stations and temperature measurements in the stations at different days and times was provided by the Swedish Meteorological and Hydrological Institute (SMHI).

We used three gaussian kernels:

1. The first to account for the physical distance from a station to the point of interest.

2. The second to account for the distance between the day a temperature measurement was made and the day of interest.

3. The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Smoothing coefficients were chosen manually on what seemed to be reasonable:

1. h_time = **4 hours**
2. h_distance = **200 km**
3. h_date = **15 days**

Plots of the kernel values as a function of distance were made in order to verify that it gives large kernel values to closer points and small values to distant points.

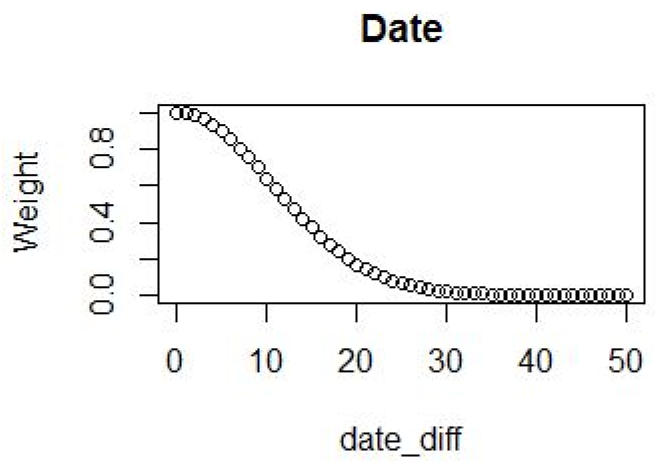


Figure 1. Smoothing factor for dates

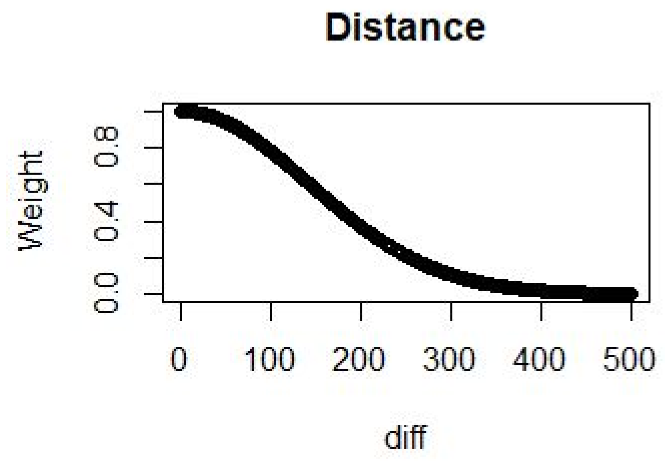


Figure 2. Smoothing factor for distance

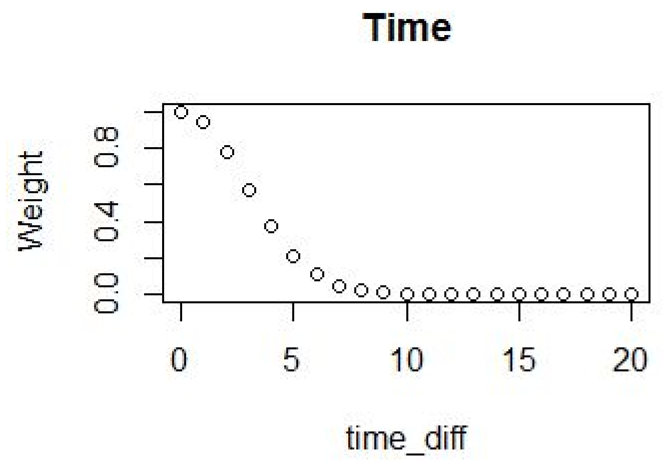


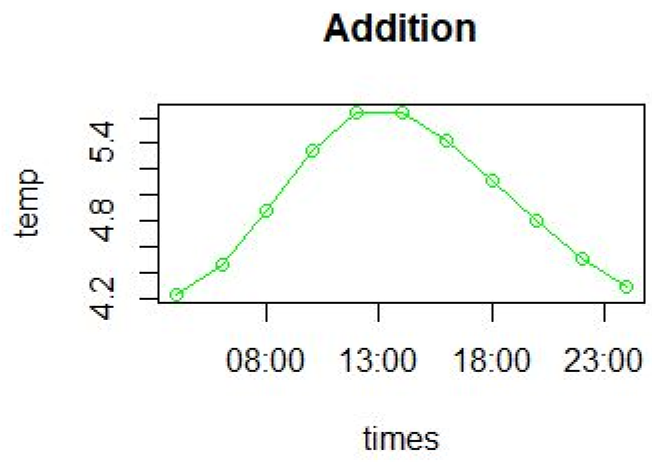
Figure 3. Smoothing factor for time of day.

The Gaussian kernels were calculated using:

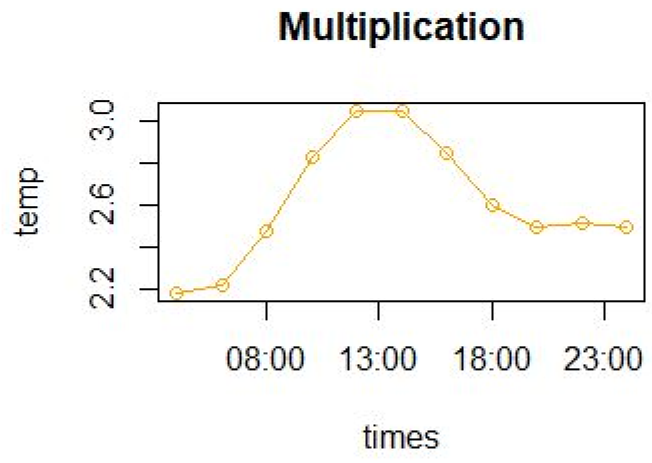
$k = e^{-(u/h)^2}$ where h is the smoothing factor.

The date **2014-11-10** and the coordinate **58.4274 : 14.826** (Vadstena) was chosen. Dates after **2014-11-10** were filtered out so as not to be accounted for in the prediction.

First the the three kernels were summed together to get the predicted temperature: $\frac{(k_{hour} + k_{distance} + k_{days}) * t}{(k_{hour} + k_{distance} + k_{days})}$ where t is the vector of all temperature measurements. The prediction was then normalized.



Next the three kernels were multiplied together. $\frac{(k_{hour} * k_{distance} * k_{days}) * t}{(k_{hour} * k_{distance} * k_{days})}$



We can see that the model captures the trend of a typical day with temperatures rising until noon and then falling towards night. We can also see that the temperatures are in a reasonable range for the time of year that was chosen. However the range of temperature differs between the two models.

Assignment 2

This assignment is to answer two question in the code from Lab3Block1 2020 SVMs.R, which performs SVM model selection by using the function ksvm from the R package kernlab. All the models to select from use the radial basis function kernel with a width of 0.05. The C parameter varies between the models.

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

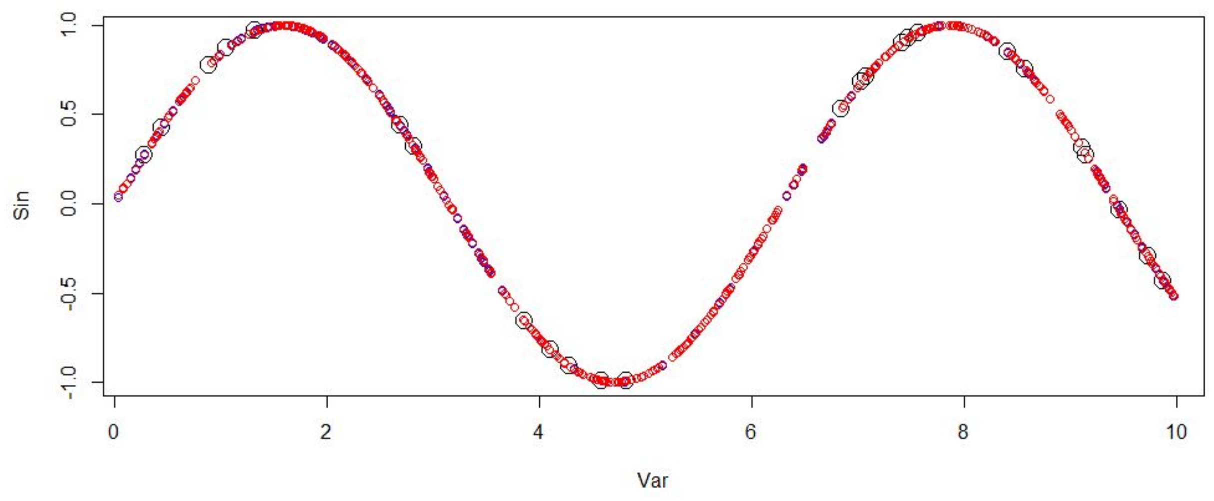
Filter3, the model returned should be trained on all data, of training data,validation data and test data.

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

Err2, generalization error means the model's performance on unknown data. Considering training data and validation data have been used during model selection, the unused data in the known dataset is test data. Filter2 is trained on training data and validation data, its err2 on test data can be the estimate of the generalization error.

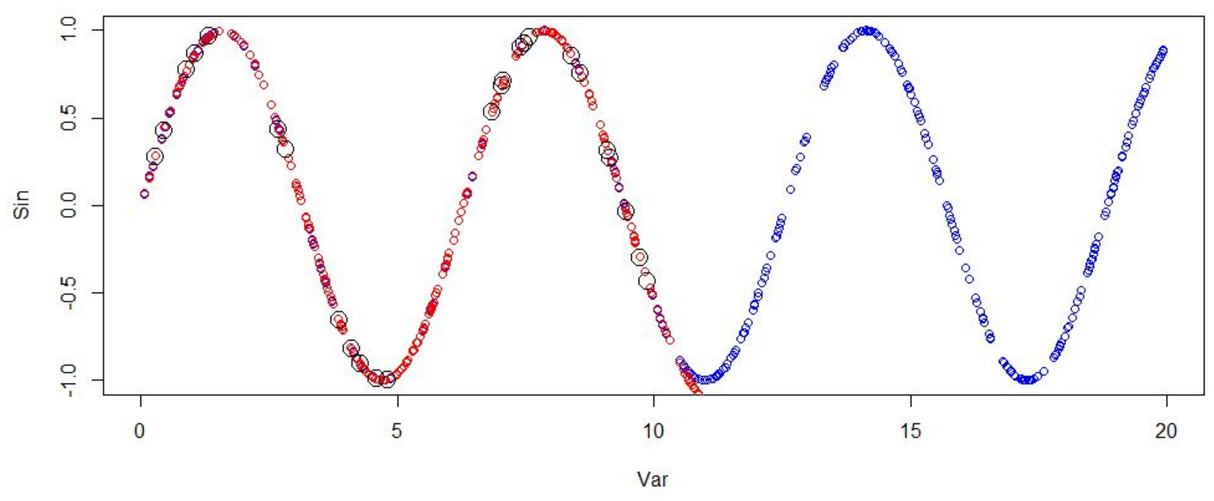
Assignment 3 - Neural networks

The assignment was to train a neural network in order to approximate continuous functions. A neural network with 2 hidden layers of 5 units each was trained to predict sin(x) on the interval [0,10]. 500 random samples of x were generated and the corresponding values of sin(x) was calculated. 25 samples were used as training data while the other 475 were used as validation data. The result is displayed in the figure below.

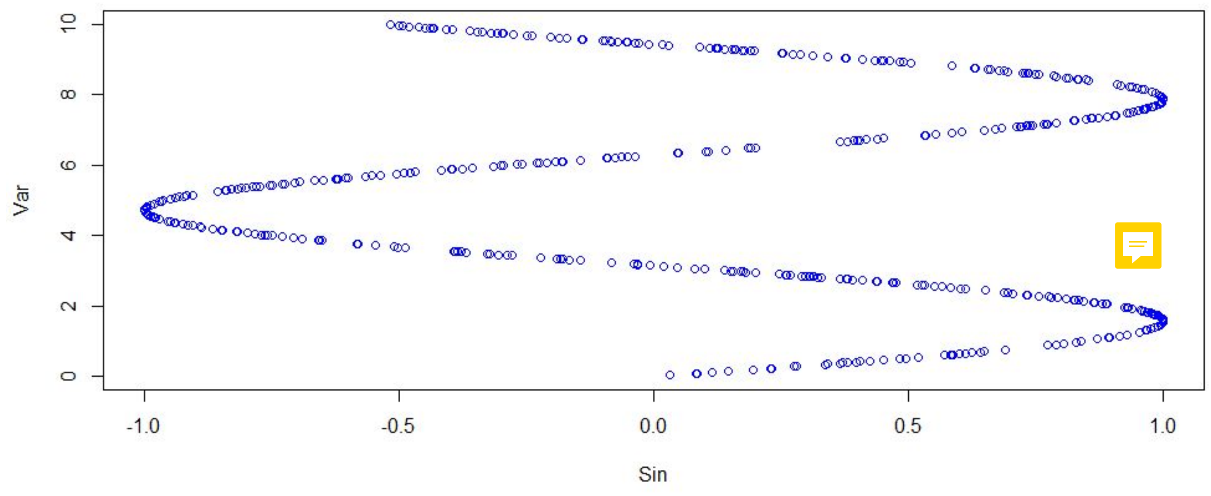


The training samples are the black, larger circles.The predicted values for the training samples are shown in red and the validation data samples are shown in blue (covered by the predicted values in this graph). The graph shows that a neural network with relatively low complexity can be used to fit a cosine function.

The same neural network was used to predict function values for the function sin(x) on the interval [0,20]. The result is displayed below. The fit was great for values of x between 0 and 10, which was also shown in the previous graph. But the prediction diverges immediately after, which suggests that the solution was not generalized to an arbitrary interval, which could be seen as a type of overfit.



The final experiment was to see if the neural network could predict values of the function arcsin(x). A new neural network was trained on 500 random samples where $x \in [0, 10]$. Some of the iterations of the training did not converge, which made training useless. This is thought to be an error in the neuralnet package of R, since iterations that do converge could and should still be used. In the assignment, it was mentioned that the fit was expected to be bad. This is reasonable since each value of x now has multiple possible values of arcsin(x), which is impossible to predict in reality.



Statement of contribution

Filip Isaksson (filis220) - Code and report for assignment 1
Qiming Tang (qimta031) - Code and report for assignment 2
Emil Mårtensson (emima951) - Code and report for assignment 3

Appendix

Code for Assignment 1:

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

#Functions####
gaussianKernel <- function(x_diff, h) {
  u = x_diff/h
  kernal = exp(-(u)^2)
  return (kernal)
}

dist_date <- function(date1, date2){
    x = (as.numeric(difftime(date1, date2, units = "days"))%%365)
    for (i in 1:length(x)){
      if(x[i] > 182)
        x[i] = 365 - x[i]
    }
    return (x)
}

dist_time <- function(time1, time2){
  x = as.numeric(difftime(time1, time2, units = "hours") )
  for (i in 1:length(x)){
    if(x[i] > 12)
      x[i] = 24 - x[i]
  }
  return (x)
}

#Initialize: #####

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


# The point to predict (up to the students)
a <- 58.4274
b <- 14.826

# The date to predict (up to the students)
```

```
date <- "2014-11-10"
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")
times <- strptime(times, "%H:%M:%S")


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

#Filter out dates that come after chosen date:
st_filtered = st[as.Date(st$date) < as.Date(date),]
st_filtered$time = strptime(st_filtered$time, "%H:%M:%S")


#Smoothing factors: ####
h_distance <- 200000 #meters
h_date <- 15 #days
h_time <- 4 #hours

h=c(h_distance,h_date, h_time)


distance = seq(0, 500)
kern_dist=gaussianKernel(distance, h_distance/1000)
plot(distance, kern_dist, xlab = "diff", ylab="Weight", main="Distance" )

date_diff = seq(0, 50, 1)
kern_dist=gaussianKernel(date_diff, h_date)
plot(date_diff, kern_dist, xlab = "date_diff", ylab="Weight", main="Date" )

time_diff = seq(0, 20, 1)
kern_dist=gaussianKernel(time_diff, h_time)
plot(time_diff, kern_dist, xlab = "time_diff", ylab="Weight", main="Time" )

#Predictions with summation: ####

dist_diff = distHaversine(c(b,a), cbind(st_filtered$longitude, st_filtered$latitude))
k_dist <- gaussianKernel(dist_diff, h_distance)

date_diff = dist_date(date, st_filtered$date)
k_date <- gaussianKernel(date_diff, h_date)

for (i in 1:length(times)) {
  time_distance = dist_time(times[i], st_filtered$time)
  k_time = gaussianKernel(time_distance, h_time)

  temperature[i] = sum((k_dist + k_date + k_time)*st_filtered$air_temperature) / sum(k_dist +
k_date + k_time)

  }

plot(times, temperature, type="o", main="Addition", col="green")



#Predictions with multiplication: ####

dist_diff = distHaversine(c(b,a), cbind(st_filtered$longitude, st_filtered$latitude))
k_dist <- gaussianKernel(dist_diff, h_distance)

date_diff = dist_date(date, st_filtered$date)
k_date <- gaussianKernel(date_diff, h_date)

for (i in 1:length(times)) {
  time_distance = dist_time(times[i], st_filtered$time)
  k_time = gaussianKernel(time_distance, h_time)

  temperature[i] = sum((k_dist * k_date * k_time)*st_filtered$air_temperature) / sum(k_dist *
k_date * k_time)
}

plot(times, temperature, type="o", main="Multiplication", col="orange")
```

Code for Assignment 2

```
# Lab 3 block 1 of 732A99/TDDE01 Machine Learning
# Author: jose.m.pena@liu.se
# Made for teaching purposes

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
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))
}

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)
err0

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)
err1

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)
err2
```

```
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)
err3

# Questions

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

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

Code for Assignment 3

```
#Function: calculates the number of weights in a normal neural network given the hidden
numberOfWeights <- function(nInputs, hiddenLayerVec, nOutputs) {
  nodeNumPrev <- nInputs
  weightNum <- 0

  for (nodeNum in hiddenLayerVec) {
    weightNum <- weightNum + (nodeNumPrev+1)*nodeNum
    nodeNumPrev <- nodeNum
  }

  weightNum <- weightNum + (nodeNumPrev+1)*nOutputs
  return(weightNum)
}

#Add the neutralnet package
library(neuralnet)

# Train a neural network to learn the trigonometric sine function. To do so, sample 500 points
# uniformly at random in the interval [0;10]. Apply the sine function to each point. The resulting
# pairs are the data available to you. Use 25 of the 500 points for training and the rest for test.
# Use any number of layers and hidden units that you consider appropriate. You do not need to
# apply early stopping. Plot the training and test data, and the predictions of the learned NN on
# the test data.
set.seed(1234567890)
Var <- runif(500, 0, 10) #500 random values between 0 and 10
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test

hiddenNodes <- c(5,5) #Number of elements specifies how many hidden layers, while the
elements specify how many neurons in each layer
weightNum <- numberOfWeights(nInputs = 1, hiddenLayerVec = hiddenNodes, nOutputs = 1)
winit <- runif(weightNum, -0.1,0.1) #Set random (low) start values on the weights
nn <- neuralnet(data = mydata, formula = Sin ~ Var, hidden = hiddenNodes, act.fct = "tanh",
startweights = winit, rep = 10)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)


# 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.

set.seed(1234567890)
Var <- runif(500, 0, 20)
mydata2 <- data.frame(Var, Sin = sin(Var))
plot(mydata2, col = "blue")
points(mydata2[,1], predict(nn,mydata2), col = "red", cex = 1)
points(tr, cex = 2)

# 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.

set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata3 <- data.frame(Sin = sin(Var), Var)
hiddenNodes3 <- c(5,5)
weightNum3 <- numberOfWeights(nInputs = 1, hiddenLayerVec = hiddenNodes3, nOutputs = 1)
winit3 <- runif(weightNum3, -0.1, 0.1) #Set random (low) start values on the weights

nn3 <- neuralnet(data = mydata3, Var ~ Sin, hidden = hiddenNodes, act.fct = "tanh",
startweights = winit3, rep = 10)
plot(mydata3, col = "blue")
# points(mydata3[,1], predict(nn3, mydata3), col = "red", cex = 1)
# Error in the package, see Fix bug when some replications don't converge #21 below
#https://stackoverflow.com/questions/56254321/error-in-ifncol-matrix-rep-argument-is-of-length-
zero
```