TDDE01 : Computer lab 3 report

Group B.15

Charlie Elf - chael086 - chael086@student.liu.se
Christoffer Gärdin - chrga084 - chrga084@student.liu.se
Felix Lindgren - felli675 - felli675@student.liu.se

October 26, 2022

Contents

1 Statement Of Contribution

In this lab the main responsibility of the 3 assignments was divided onto the group members. Assignment 1 was held responsible for by Felix Lindgren. Assignment 2 was held responsible for by Christoffer Gärdin. Assignment 3 was held responsible for by Charlie Elf. The load was in that way divided onto the group members. The assignment responsibility included for each group member making sure that code was working to solve the problem and the writing of each assignment in the lab report.

2 Assignment 1 - Kernel Methods

2.1 Implementation of kernel method

For this task we implemented a kernel method that predicted the weather at a given latitude, longitude and date from 04:00 to 24:00 at said date. This was to be done by creating three gaussian kernels, one to account for the distance between the measurement and the given point, one to account for the differance between the date of the measurement and the one in the data and finally one kernel to account for the difference in the time of day. The physical distance used was the haversine distance between two latitude-longitude points. The distance in dates was seen as the difference inside a given year, this was under the assumption that while there could be some changes in the year to year temperatures, like global warming, it made more sense to look at what day during the year a measurement was taken. Time of day was the time between the measurement and the time of prediction. These kernels were then combined, giving the relevance of each weather measurement, they were all added together to get the final prediction of the weather at a given location, date and time. Measurements that were taken after the given date were taken out and not used.

2.2 Smoothing coefficient

When choosing smoothing coefficients it was mostly done by trial and error while using the plots of the kernel function as guidance. The three kernel functions are shown below in figure 1. For the physical distance it needed to be quite large since the nearest station could be quite far away so $h_{distance} = 30000$. For the date it was quite small since it seemed better to give more weight to dates close to the one specified, $h_{date} = 7.5$. The time coefficient was the smallest since there will be quite a big difference in temperature during the day, and the differences were generally not very large, $h_{time} = 3$.

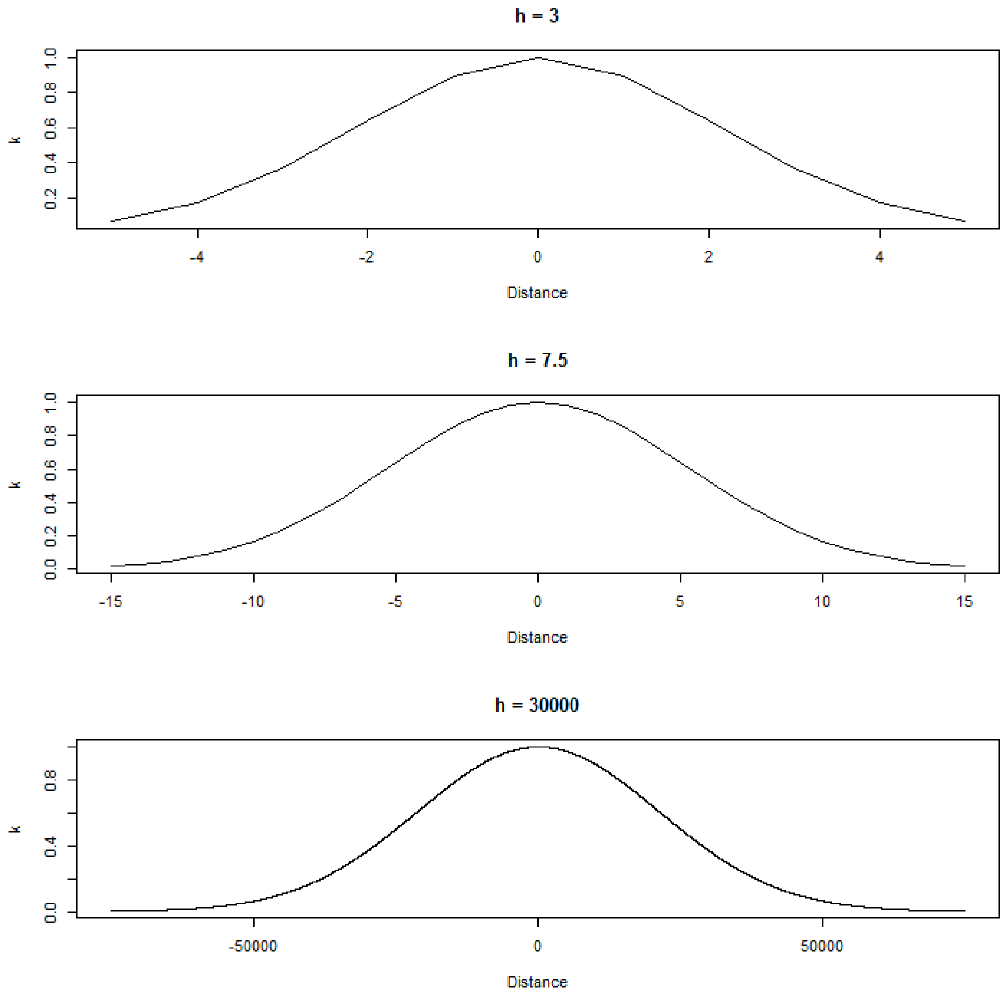

Figure 1: Plots showing the kernel function output for time, date and physical distance.

2.3 Additive vs multiplicative kernels

When combining the results from the different kernels this was at first done by adding them all together. Next we were supposed to combine the different kernels by multiplying them instead. The results for both methods are shown in figure 2 for the date 1998-08-15 at a lat/lon around the Stockholm area. The reason why multiplying should give a better result is the relevance of each point will scale each other depending on it's importance. So say that a measurement is very close to the given point and it is on the correct date, but it is during the wrong part of the day, then the kernel value of the time will reduce it's relevance, while if it all added together being on the correct day and location would be enough to give a "high" relevance for said score. In the figure we can see that the additive kernel seems to have a lower average and the highest temperature around 20:00. Compare this to the multiplicative kernel that instead has a much smoother curve where the highest point is reached around 14:00, this seems like a more reasonable estimate of the weather. The higher temperature in the multiplicative kernel also seems more reasonable for the weather in

August.

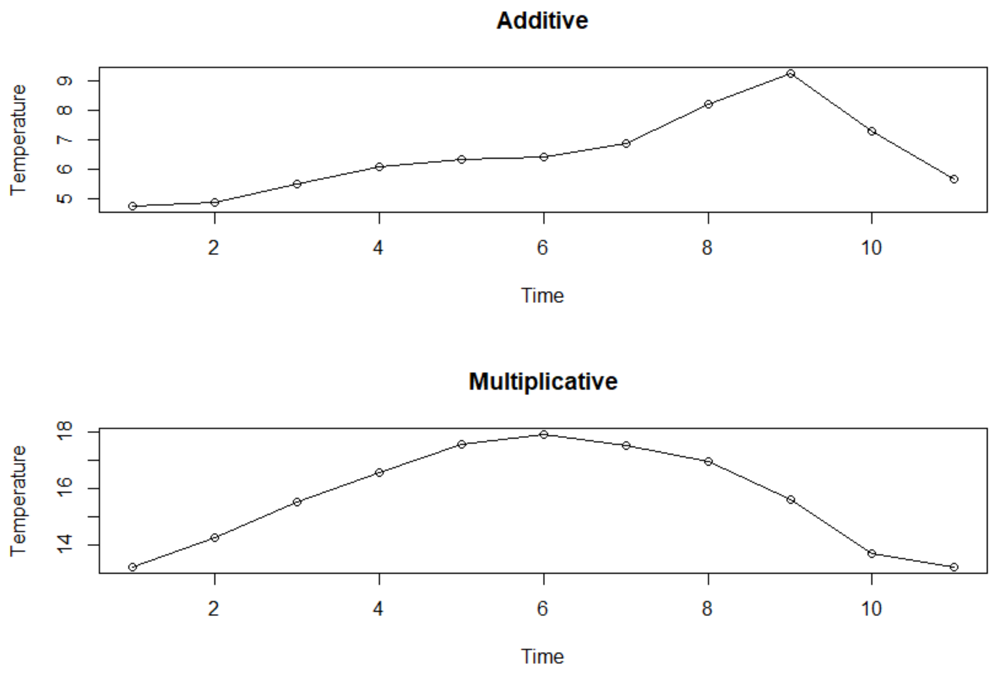

Figure 2: Plots showing the predicted temperature from 04:00 to 24:00 with two hour steps.

3 Assignment 2 - Support Vector Machines

In this assignment, we were supposed to analyze an SVM model that classifies the spam dataset. All the models use the radial basis function kernel with a width of 0.05.

3.1 Filter choice

The difference we can find between the filters is that some are trained and predicted on different data sets. The parameters: kernel, kpar, and C are all the same between the filters. Filter 0 is trained on the training set and predicted on the validation set, filter 1 is trained on the training set and predicted on the test set, filter 2 is trained on both the training and validation set and predicted on the test set, lastly filter 3 is trained on the whole spam data set and predicted on the test set. The validation data set is already used to find the best C parameter in the ksvm function. Therefore the best filter to return to the user would be filter 1 or filter 0 which is fitted on the training data and later can be evaluated by the test data. If we would use filter 3, we would make predictions on a small part of the data that has already been fitted to the model which not may calculate optimal estimations on new data and lead to overfitting. Also, in filter 2 we have already used the validation set to find optimal values for the C parameter. The best choice would therefore be filter 1 or filter 0 as it can be used on the test data when we determining how good our model is and later be sent to the user.

Filter 3 should be returned which has the smallest amount of error. But the data is overfitted but as the future predictions are optimal as the support vectors are optimal.

3.2 Estimate of the generalization error of the filter returned to the user

The estimate of the generalization errors can be seen in Table 1. The estimated generalization error returned to the user is gotten by predicting on the test data which is 8.489388% (Err1) and trained on the training data. The test data is used to finally evaluate how good the model is. Err3 would make prediction on some of the data that has already been trained on the model which could lead to overfitting. Err0 make prediction on the validation set which is used to tune the model. Err2 is predicting on both the training and validation data set which may not lead to optimal C parameter. Err1 is tested on unseen data which avoids overfitting and can help us determining how good the model is.

Table 1: Estimated generalization error for filters 0-3.

	Err0	**Err1**	**Err2**	**Err3**
Estimated generalization error	6.75%	8.489388%	8.2397%	2.122347%

3.3 Implementation of the linear combination for filter 3

In the first equation below we can see how we implemented the linear combination for filter 3. $\hat{\alpha}$ is the linear coefficients for the support vectors, X is the support vectors from the fitted SVM model, $x_\star$ is the new point that is going to be classified, and b is the negative intercept. The second equation shows the Radial basis kernel function where the denominator is a design parameter set as 0.05.

$$\hat{y}(x_\star) = sign(\hat{\alpha}^T K(X, x_\star) + b) \tag{1}$$

$$K(X, x_\star) = exp\left(-\frac{||X - x_\star||_2^2}{2\delta^2}\right) \tag{2}$$

In Table 2 we can observe the predictions done by the implemented linear combination for filter 3 on the 10 first points in the spam data set. The sign of the value determines if it is a spam or not.

Table 2: Results for the implemented linear combination for filter 3.

	Predicted value
Iteration 1	-1.998999
Iteration 2	1.560584
Iteration 3	1.000278
Iteration 4	-1.756815
Iteration 5	-2.669577
Iteration 6	1.291312
Iteration 7	-1.068444
Iteration 8	-1.312493
Iteration 9	1.000184
Iteration 10	-2.208639

4 Assignment 3 - Neural Networks

In this assignment we are to create Neural Network models that from x values will predict the sin(x) values and vice versa. 10 hidden units with one hidden layer is used in all Neural Networks. The activation function and data sets differs between the tasks.

4.1 Neural Network with Sigmoid activation function

A dataset with 500 random numbers between 0 - 10 is created and split up into training- and test-data with 5% as training and 95% as test. The goal is to predict sin(x) from values of x. A neural network (NN) model with 10 hidden layers in 1 hidden layer is trained on the training data. The NN model here uses the standard logistic activation function $h(x) = \frac{1}{1+e^{-x}}$. In Figure 3 we see the NN with 10 hidden units in 1 hidden layer predicting the sine value very well.

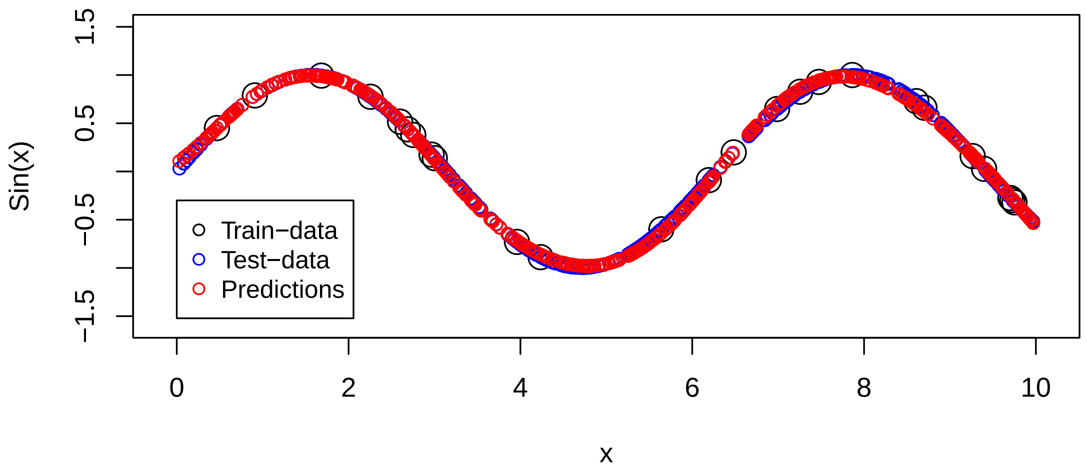

Figure 3: Sin(x) predictions from x with Sigmoid activation function.

4.2 Neural Network with Linear, ReLU and Softplus activation function

Here the same training data sample is used to train 3 different NN models. The NN models are trained with different activation functions. Linear, ReLU and Softplus. The results are shown in Figures 4, 5 and 6.

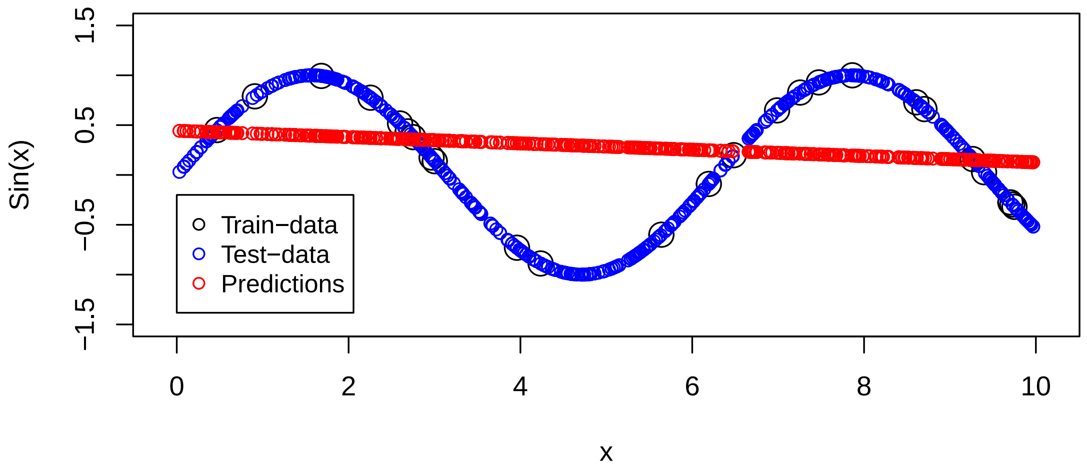

Figure 4: Sin(x) predictions from x with linear activation function.

It is clear from Figure 4 that the linear activation function is not a good alternative for predicting sin(x). The activation function here is linear and thus the model basically becomes a standard linear regression model whereas the sine function is not linear.

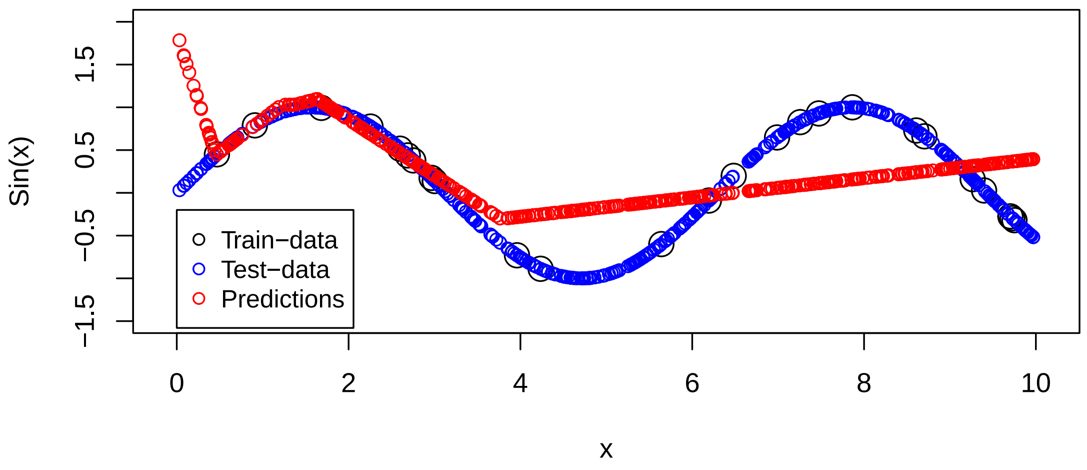

Figure 5: Sin(x) predictions from x with ReLU activation function.

In Figure 5 the predictions are somewhat ok for $x \in [0.5, 3]$ and then seems to become a standard linear regression. The ReLU function performs better than the linear one but is still to simple to get good predictions for the sine function.

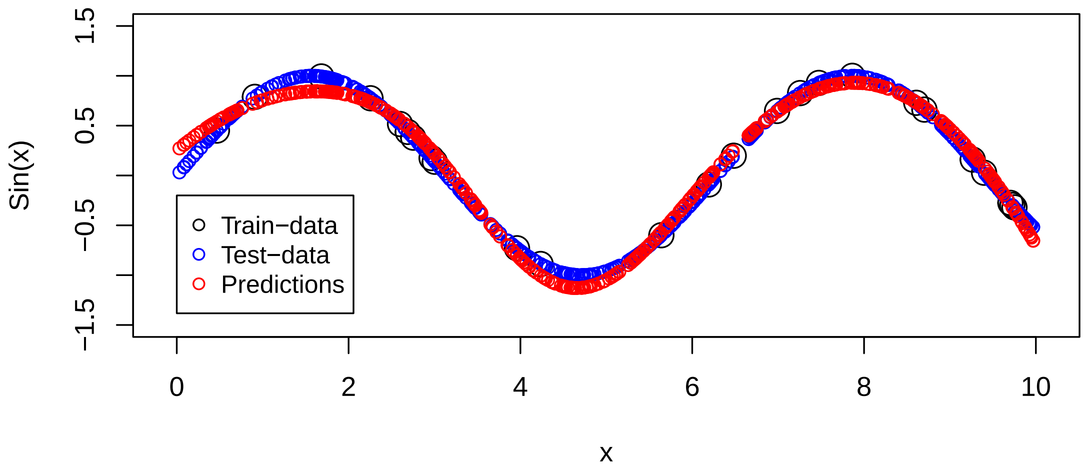

Figure 6: Sin(x) predictions from x with Softplus activation function.

Figure 6 shows fairly good predictions from the NN model indicating that the nonlinear Softplus activation works well for predicting the nonlinear sine function. The predictions here show that for response values having a nonlinear relation to the inputs it is required to have a nonlinear activation function. This is also seen in the first example in Figure 3 where the results of the nonlinear Sigmoid function also is good.

4.3 Neural Network with Sigmoid activation function and bigger span in data set

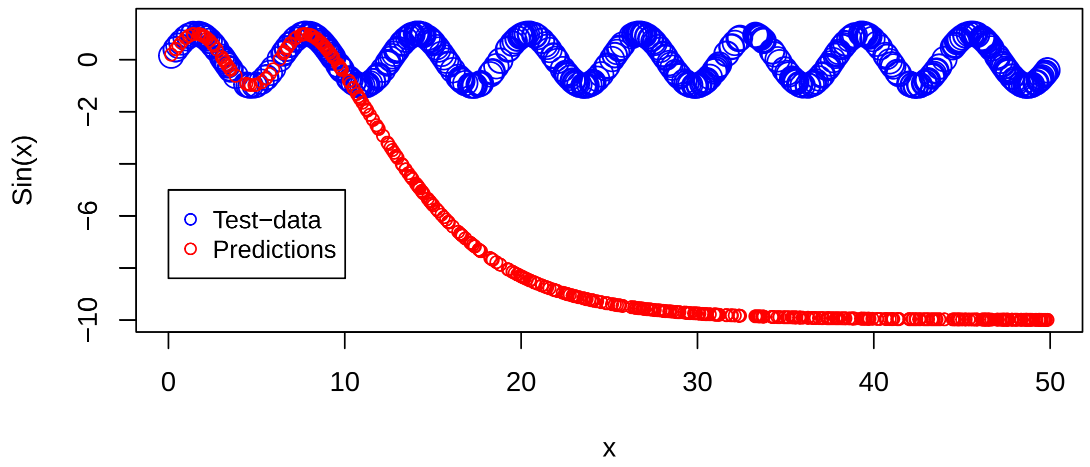

Figure 7: x predictions from sin(x) with Sigmoid activation function and trained model from Section 4.1

A new data sample of 500 random numbers between 0 - 50 is created. The NN model trained in section 4.1 is used for predicting the whole new data sample of 500 inputs. The result is shown in Figure 7 with show varying results. The NN model performs well for $x < 10$ which is expected as the model was trained with those values of x in the training data sample. For $x > 10$ however the model does not perform well and the predictions converge to about -10 as x grows.

4.4 Convergence of Neural Network model

For $x > 10$ in the Sigmoid activation function multiplied with the weights causes the hidden unit output to be either 1 or 0 depending on the weight being either poisitive or negative. When x grows the function output becomes more and more binary (1 or 0). This is because of the Sigmoid function, see Equation 3 having the exponent causing the output of 1 or 0 depending on positive or negative W. This in turn causes the the predictions to converge to about -10 because of the weights in the final tensor.

$$h(z) = \frac{1}{1+e^{-(z)}}, z = b + \sum_{j=1}^{p} x_j * W_j \tag{3}$$

4.5 Neural Network predictions of x from sin(x)

A new data sample of 500 random numbers between 0 - 10 is created. All numbers in the data sample are used to train a NN model to instead of predicting sin(x) from x it is trained to predict x from sin(x). The predictions are made from the same data sample for sine value of the 500 input variables. The NN model here uses the Sigmoid activation function.

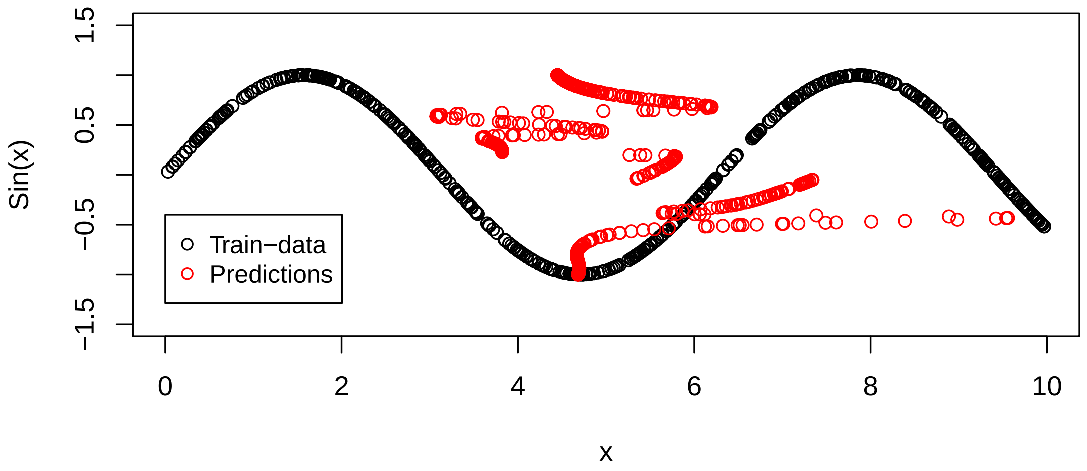

Figure 8: x predictions from sin(x) with Sigmoid activation function.

In Figure 8 the result is shown. It shows very bad predictions even with the activation function that worked well for predictions of sin(x) from x. A reason for this is the sine function being periodical which the NN model has troubles understanding (example: $sin(1) = sin(1 + 2\pi)$). The NN model should then choose between multiple possible x values which get problematic. This is not a problem in the previous parts because of the data samples not having to deal with the periodical factor.

A Source code

A.1 Assignment 1

```r
library(ggplot2)
library(geosphere)
set.seed(1234567890)

stations <- read.csv("stations.csv", fileEncoding="latin1")
temps <- read.csv("temps50k.csv", fileEncoding="latin1")
st <- merge(stations,temps,by="station_number")
h_distance <- 3e4# These three values are up to the students
h_date <- 7.5
h_time <- 3
a <- 59.334591 # The point to predict (up to the students)
b <- 18.063240
date <- "1998-08-15" # 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))

N = length(st$air_temperature)
#plot(temp, type="o")

filter_dates <- function(){
  numeric_date <- as.numeric(as.POSIXct(date))
  numeric_dates <- as.numeric(as.POSIXct(st$date))
  fd <- replicate(N, numeric_date) - numeric_dates
  filtered_indexes <- which(fd > 0)
  return(st[c(filtered_indexes),])
}


distances <- function(st){
  ## Distance, physical, between station and sample point
  N <- length(st$air_temperature)
  dist = replicate(N,0)
  for (i in 1:N) {
    dist[i] =  distHaversine(c(st$latitude[i], st$longitude[i]), c(a,b))
  }
  # Distance between days
  day_dist = replicate(N,0)
  date_numeric <- as.numeric(strftime(date,"%j"))
  days_numeric <- as.numeric(strftime(st$date, "%j"))
  days_dist = replicate(N, date_numeric) - days_numeric

  hours_numeric <- as.numeric(vapply(strsplit(unlist(st$time),":"), `[`, 1, FUN.
    VALUE=character(1)))
  hour_diff = 4 - hours_numeric
  hour_diff = apply(as.matrix(hour_diff), 1, function(x) {ifelse(any(x < -12), -24 -
     x, x)})
  return(data.frame(phys=dist, days=days_dist,time=hour_diff))
}

k <- function(d){
  return(exp(-norm(as.matrix(d))^2))
}

kernel_add <- function(dist, days_dist, hour_diff){
  k_dist <- vapply(dist/h_distance, k, FUN.VALUE = numeric(1))
  k_days <- vapply(days_dist/h_date, k, FUN.VALUE = numeric(1))
  k_hour <- NULL
  #vapply(hour_diff[[times[1]]]/h_time, k, FUN.VALUE = numeric(1))
  r <- replicate(length(times), 0)
  for(i in 1:length(times)){
    hours <- hour_diff + 2*(i-1)
    hours <-apply(as.matrix(hours), 1, function(x) {ifelse(any(abs(x) > 12), 24 -
    abs(x), x)})
    k_hour <- vapply(unlist(hours)/h_time, k, FUN.VALUE = numeric(1))
    k_tot = (k_hour + k_days + k_dist)/3
    res <- sum(k_tot*(f_data$air_temperature)) / sum(k_tot)
    r[i] <- res
  }
  return(r)
}

kernel_mult <- function(dist, days_dist, hour_diff){
  k_dist <- vapply(dist/h_distance, k, FUN.VALUE = numeric(1))
  k_days <- vapply(days_dist/h_date, k, FUN.VALUE = numeric(1))
  k_hour <- NULL
  #vapply(hour_diff[[times[1]]]/h_time, k, FUN.VALUE = numeric(1))
  r <- replicate(length(times), 0)
  for(i in 1:length(times)){
    hours <- hour_diff + 2*(i-1)
    hours <-apply(as.matrix(hours), 1, function(x) {ifelse(any(abs(x) > 12), 24 -
    abs(x), x)})
    k_hour <- vapply(unlist(hours)/h_time, k, FUN.VALUE = numeric(1))
    k_tot = k_hour * k_days * k_dist
    res <- sum(k_tot*(f_data$air_temperature)) / sum(k_tot)
    r[i] <- res
  }
  return(r)
}

f_data <- filter_dates()
d <- distances(f_data)
kr_add <- kernel_add(unlist(d[1]), unlist(d[2]), d[3])
kr_mult <- kernel_mult(unlist(d[1]), unlist(d[2]), d[3])
print(kr_add)
print(kr_mult)
par(mfrow=c(2,1))
plot(kr_add, type='o', main="Additive",
     xlab="Time",
     ylab="Temperature")
plot(kr_mult, type='o',main="Multiplicative",
     xlab="Time",
     ylab="Temperature")

k_dist <- vapply(unlist(d[1])/h_distance, k, FUN.VALUE = numeric(1))
k_days <- vapply(unlist(d[2])/h_date, k, FUN.VALUE = numeric(1))
k_hour <- vapply((unlist(d[3]) + 18)/h_time, k, FUN.VALUE = numeric(1))
k_tot = (k_hour * k_days * k_dist)

sum(k_hour)
sum(k_dist)
sum(k_days)


par(mfrow=c(3,1))
x <- -5:5
plot(x, exp(-(x/h_time)^2), main="h = 3",
     xlab="Distance",
     ylab="k",
     type='l')
x <- -15:15
plot(x, exp(-(x/h_date)^2), main="h = 7.5",
     xlab="Distance",
     ylab="k",
     type='l')
x <- -75000:75000
plot(x, exp(-(x/h_distance)^2), main="h = 30000",
     xlab="Distance",
     ylab="k",
     type='l')
```

A.2 Assignment 2

```
# Lab 3 block 1 of 732A99/TDDE01/732A68 Machine Learning
# Author: jose.m.pena@liu.se
# Made for teaching purposes

library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58]) #scale all columns except the predicted column type
tr <- spam[1:3000, ] #training: first 3 000 rows
va <- spam[3001:3800, ] #validation: rows between 3 001 and 3 800
trva <- spam[1:3800, ] #training and validation: first 3 800 rows
te <- spam[3801:4601, ] #test: rows between 3 801 and 4 601

by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){
  print(i)
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=i,scaled=
    FALSE)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}

#Filter 0 is trained on the training data set and tested in the validation data set
filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err
    _va)*by,scaled=FALSE)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t) #misclassification error
err0

#Filter 1 is the best: is trained on training data and predicts on the test data
filter1 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err
    _va)*by,scaled=FALSE)
mailtype <- predict(filter1,te[,-58])
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
err1

#Filter 2 is trained on the training and validation data
filter2 <- ksvm(type~.,data=trva,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(
    err_va)*by,scaled=FALSE)
mailtype <- predict(filter2,te[,-58])
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
err2

#Filter 3 is trained on the whole spam data set
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(
    err_va)*by,scaled=FALSE)
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?

  #The differences we can find between the filters is that the some are trained and
    predicted on different data sets.
  #The most appropriate one is the one that is trained on the training data set and
    predicted on the test data set which is filter1.

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

# 3. Implementation of SVM predictions.

sv<-alphaindex(filter3)[[1]] #Indexes of the of the support vectors
co<-coef(filter3)[[1]] # The linear coefficients for the support vectors
inte<- - b(filter3) #The negative intercept of the linear combination
spam.features <- spam[-58] #Remove the predicted value
lambda <- 0.05 #Tuned parameter for RBF
k<-NULL
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset
    .
  k2 <- NULL
  k2<-inte
  for(j in 1:length(sv)){
    k2<- k2 + co[j]*exp(-lambda*(sum((spam.features[sv[j],]-spam.features[i,])^2)))
    #intercept + weights * RBF kernel function
  }
  k<-c(k,k2) #Concatenate k2
  print(k)
}
k
predict(filter3,spam[1:10,-58], type = "decision")
```

A.3 Assignment 3

```
# Lab 3
# Assignment 3

library(neuralnet)
set.seed(1234567890)

# 3.1
sample <- runif(500, min =0, max=10) # 500, random vector 0-10
sin_sample <- sin(sample) #  Applying sine-function to the vector
data <- as.data.frame(cbind(sample, sin_sample))

# Dividing into training and test data with proportion 25:475
n=dim(data)[1]
id=sample(1:n, floor(n*0.05))
train=data[id,]
test=data[-id,]

# Creating neuralnet model with 1 layer and 10 hidden units.
model_1 = neuralnet(sin_sample ~ sample ,data = train, hidden = 10)

```

```r
# Plotting train-, test-data and test predictions.
plot(train, cex=2, xlab = "x", ylab = "Sin(x)", xlim=c(-0.1,10.1), ylim =c(-1.5,1.5)
    )
points(test, col = "blue", cex=1)
points(test[,1], predict(model_1, test), col="red", cex=1)
legend(0, -0.3, c("Train-data", "Test-data", "Predictions"),cex = 0.9, col=c("black"
    ,"blue","red"),pch=c(1,1,1))

# 3.2 Creating neuralnet models with different activation functions.
h1 <- function(x) x # Linear
model_h1 = neuralnet(sin_sample ~ sample ,data = train, hidden = 10, act.fct = h1)
h2 <- function(x) ifelse(x>=0, x, 0) # ReLU
model_h2 = neuralnet(sin_sample ~ sample ,data = train, hidden = 10, act.fct = h2)
h3 <- function(x) log(1 + exp(x)) # Softplus
model_h3 = neuralnet(sin_sample ~ sample ,data = train, hidden = 10, act.fct = h3,
    threshold = 0.1)

# Plot with Linear activation function
plot(train, cex=2, xlab = "x", ylab = "Sin(x)",xlim=c(-0.1,10.1), ylim =c(-1.5,1.5))
points(test, col = "blue", cex=1)
points(test[,1], predict(model_h1, test), col="red", cex=1)
legend(0, -0.2, c("Train-data", "Test-data", "Predictions"),cex = .9, col=c("black",
    "blue","red"),pch=c(1,1,1))

# Plot with ReLU activation function
plot(train, cex=2, xlab = "x", ylab = "Sin(x)",xlim=c(-0.1,10.1), ylim =c(-1.5,2))
points(test, col = "blue", cex=1)
points(test[,1], predict(model_h2, test), col="red", cex=1)
legend(0, -0.2, c("Train-data", "Test-data", "Predictions"),cex = .9, col=c("black",
    "blue","red"),pch=c(1,1,1))

# Plot with Softplus activation function
plot(train, cex=2, xlab = "x", ylab = "Sin(x)", xlim=c(-0.1,10.1), ylim =c(-1.5,1.5)
    )
points(test, col = "blue", cex=1)
points(test[,1], predict(model_h3, test), col="red", cex=1)
legend(0, -0.2, c("Train-data", "Test-data", "Predictions"),cex = .9, col=c("black",
    "blue","red"),pch=c(1,1,1))

# 3.3
set.seed(1234567890)
sample <- runif(500, min =0, max=50) # 500, random vector 0-50
sin_sample <- sin(sample) #  Applying sin-function to the vector
data_2 <- as.data.frame(cbind(sample, sin_sample))

plot(data_2, cex=2, xlab = "x", ylab = "Sin(x)", ylim=c(-10,1.5), col="blue")
points(data_2[,1], predict(model_1, data_2), col="red", cex=1)
legend(0, -5, c("Test-data", "Predictions"),cex = .9, col=c("blue","red"),pch=c(1,1)
    )

# 3.4
model_1$weights

# 3.5
set.seed(1234567890)
sample_3 <- runif(500, min =0, max=10) # 500, random vector 0-10
sin_sample_3 <- sin(sample_3) #  Applying sine-function to the vector
data_3 <- as.data.frame(cbind(sample_3, sin_sample_3))

model_3 = neuralnet(sample_3 ~ sin_sample_3 ,data = data_3, hidden = 10, threshold =
     0.1)

plot(data_3, cex=1, xlab = "x", ylab = "Sin(x)", ylim=c(-1.5,1.5))
points(predict(model_3, data_3), data_3[,2] , col="red", cex=1)
```

```r
legend(0, -0.4, c("Train-data", "Predictions"),cex = .9, col=c("black","red"),pch=c
    (1,1))
```