Computer Lab 3 Block 1

Linköping University

Jakob Fjellström
Mowniesh Asokan
Tim Yuki Washio

December 14, 2020

———-

Jakob worked on assignment 1, Mowniesh on assignment 2, and Yuki on assignment 3.

———-

Assignment 1: Kernel Methods

In this assignment, we were tasked to implement a kernel method to predict the hourly temperatures for a date and place in Sweden. This forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours, by using a kernel that is 1) the sum of three Gaussian kernels and 2) the product of the same three kernels. To start off, we define the gaussian kernel below.

$$K(u) = exp(-||u||^2)$$

Above, $\|\cdot\|$ denotes the euclidean norm. We had to take into account three things: first, the distance - measured in days - between the day which we want to make predictions, and the day of recorded temperature measurement. Second, we need to take into account the physical distance - measured in meters - between the weather station and the point of interest. Thirdly, the distance - measured in hours - between the time of the day that the temperature was measured, and the hour of interest. Hence, we will apply the gaussian kernel above on all three of these cases. The smoothing coefficients - usually the standard deviation for the gaussian kernel - we got to try out manually and choose an appropriate one. Lastly, we needed to filter out all the temperature data that was measured posterior to the day of interest. To do that, we simply partitioned the data frame as follows.

```
filtered_data <- st[as.Date(st$date) < as.Date(date),]
```

Then, to find the physical distance between to points, we used the function *distHaversine* from the *geosphere* package. We wrote the following function.

```
calc_real_dist <- function(date,p2){

    p1 <- filtered_data[,c(5,4)]
    physical_distance <- distHaversine(p1, p2)
    return(physical_distance)
  }
```

Here, $p2$ is the longitude & latitude of the point which we want to predict the temperature for. Then, to find the difference in days between the day of interest and all the days which we have recorded temperature date, we used the following function.

```
diff_in_days <- function(date) {
     day_difference <- as.numeric(difftime(filtered_data$date,
     date, units="days"))
     return(day_difference)
  }
```

Further, to find the difference in hours between the hour of interest, and the hour of temperature measure, we used the below function.

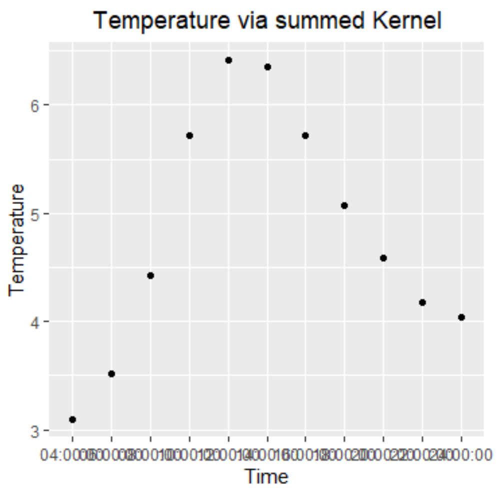
```
diff_in_time <- function(data, hour) {
     hour_difference <- as.numeric(difftime(strptime(data$time,
     format="%H:%M:%S"), strptime(hour, format="%H:%M:%S"),
     units="hours"))
     return (hour_difference)
  }
```

And then, it was easy to find the kernels.

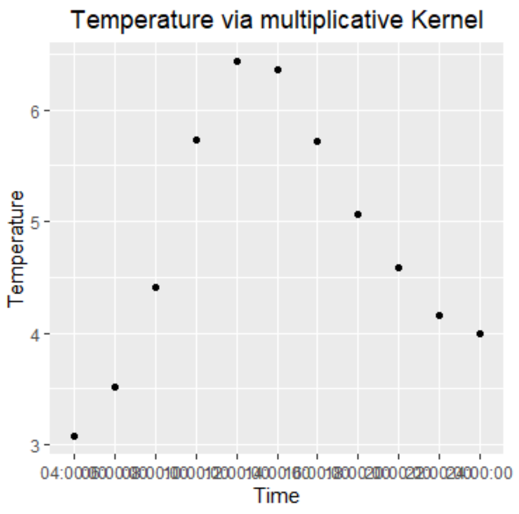
```
dist_kernel <- exp(-(1/s_c1)*(h_distance)^2)
date_kernel <- exp(-(1/s_c2)*(h_date)^2)

summed_forcast <- vector(length=length(time_interval))
prod_forcast <- vector(length=length(time_interval))
for(i in 1:length(time_interval)){

     h_times <- diff_in_time(data=filtered_data,
     hour= time_interval[i])
     time_kernel <- exp(-(1/s_c3)*(h_times)^2)

     summed_kernel <- dist_kernel + date_kernel + time_kernel
     summed_kernel_w_temp <-  summed_kernel * old_temp

     prod_kernel <- dist_kernel %*% date_kernel %*% time_kernel
```

```
    prod_kernel_w_temp <- prod_kernel %*% old_temp

    summed_forcast[i] <-
    round(sum(summed_kernel_w_temp)/(sum(summed_kernel)),2)
    prod_forcast[i] <- round(prod_kernel_w_temp/(sum(prod_kernel)),2)
}
```

Above, $sc1, sc2, sc3$ are the smoothing coefficients. And below follows the predicted temperatures at the weather station Malmö A, for 8th of February, 1996.



(a) Predicted temperature between 0400 and 2400 with summed Kernel.



(b) Predicted temperature between 0400 and 2400 with multiplicated Kernel.

Figure 1: The figure shows the predicted temperatures for the 8th of February, 1996, for two different Kernels.

As we can see, the difference between the two methods are very slim. This is a result of when calculating the *summed_kernel* and *prod_kernel*. Because the values in the vectors *dist_kernel*, *date_kernel* and the *time_kernel* is either very small, and gets round down to zero, or very close to 1, the result of adding the vectors and matrix multiply them becomes very similar.

Assignment 2: Support Vector Machines

svm is a supervised learning algorithm, which is widely used for classification algorithm.SVM is applicable for data that are linearly separable and for non-linear data it uses kernel method for classification.

- It classifies the two classes using hyperplane,The hyperplane should have the largest margin in a high dimensional space to separate a given data into a classes.The margin between the two classes represent the longest distance between the closest data point of the classes.

- Based on the number of the input features/variables, the decision boundary can be a line (if we had only 2 features) or a hyperplane. A hyperplane is an (N-1)-dimensional subspace for an N-dimensional space.

- SVM pick the decision boundary that maximizes the distance to the support vectors. The decision boundary is drawn in a way that the distance to support vectors are maximized. If the decision boundary is too close to the support vectors then, it will be sensitive to noise and not generalize well.

```
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], ]
```

Error

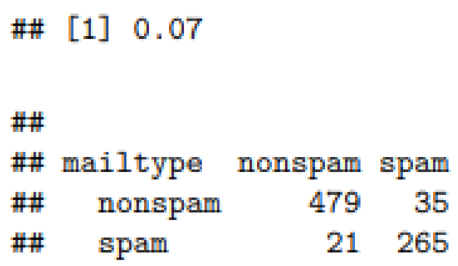
Error= (c)Classification Error + Margin Error(soft margin)

- Margin help to increase the distance of the decision boundary to the support vectors. the longest distance between the closest data point of the classes.

- Maximize the number of points that are correctly classified in the training set.

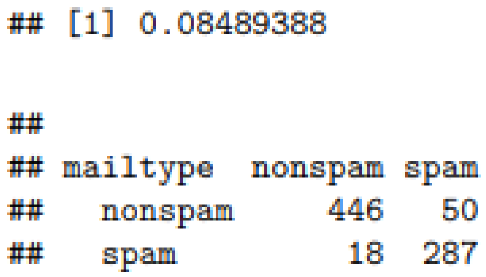
C-Parameter

C is small, the penalty for misclassified data point is low so decision boundary with large margin is choosen at the expense of the great number of misclassifications. If c is large, SVM tries to minimize the number of misclassifed samples and results in decision boundary with smaller margin.



```
## [1] 0.07

##
## mailtype  nonspam spam
##   nonspam     479   35
##   spam         21  265
```

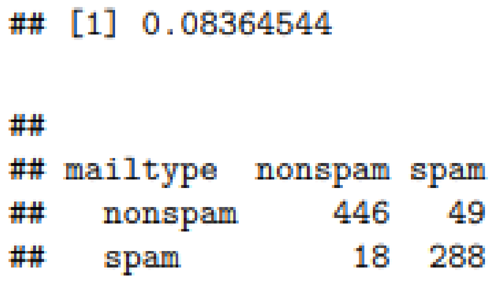
(a) Error value of Filter0



```
## [1] 0.08489388

##
## mailtype  nonspam spam
##   nonspam     446   50
##   spam         18  287
```

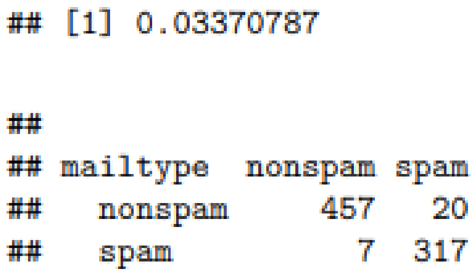
(b) Error Value of Filter1

Figure 2



```
## [1] 0.08364544

##
## mailtype  nonspam spam
##   nonspam     446   49
##   spam         18  288
```

(a) Error value of Filter2



```
## [1] 0.03370787

##
## mailtype  nonspam spam
##   nonspam     457   20
##   spam          7  317
```

(b) Error Value of Filter3

Figure 3

Questions

1.

Which filter do we return to the user ? filter0, filter1, filter2 or filter3 ? Why ?
Filter2 is performing better compare with all the other filter in the list, because this filter is trained by trva data and it is tested by the te data that is unseen to the model.
In the filter0 ,the error value is very low compare with all other filters but the filter is trained by using the tr data and tested by using the validation va data.The c parameter in this filter choose by using the va data in the previous model.So only it performs better than the other filters.Even filter3 also gives good error value but the problem is filter3 is trained by original data and is tested by the data that is taken from the original one.

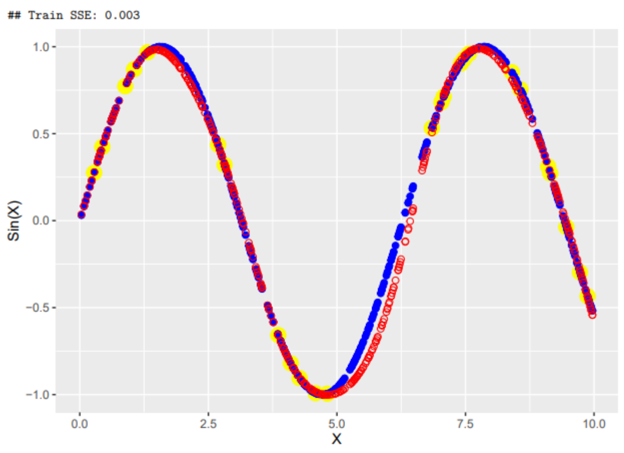
2.

What is the estimate of the generalization error of the filter returned ? err0, err1, err2 or err3 ? Why ?
Error1 is the generalized error of the all filters, because that filter is trained by the training data and while testing it is generalized to give a prediction and it is trained by unseen data at those stages.Even the best c parameter for this data is estimated by the filter is trained by using the training data.

In the RBFDOT kernel that decrease the distance between the two classes and project the values in the hyper plane. Distance between the class of data is separated by the euclidean distance identifier formula.

Assignment 3: Neural Networks

We prepare our data and split into train and test set. We use randomly generated and uniformly distributed sample points from [-1, 1] as our initial weights. Also, we train our model on 2 hidden layers with 10 and 5 neurons because this constellation returned good results.



```
## Train SSE: 0.003
```

Figure 4

First, we can see that our SSE on our training data is very low, which is good. Looking at the generated plot we observe that our model predicts $\sin(x)$ pretty well. Most of the computed points are either directly on the desired value or very much nearby. Therefore, we can conclude that our neural net is working properly on computing the sine function based on values between 0 and 10.

```
## Warning: Removed 20 rows containing missing values (geom_point).
```

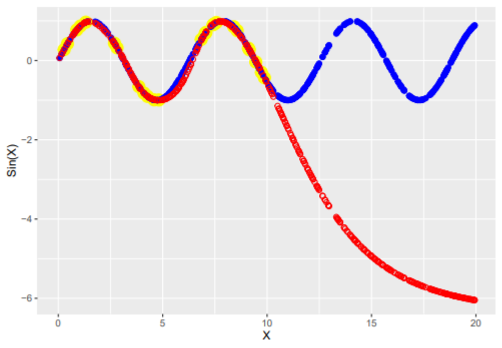


Figure 5

Our model is delivering good results in the range [0, 10] which is not surprising since we already found out about that in the previous exercise. Though, for bigger values of x our model (since it's only trained on 0,10 range) delivers bad results. When x > 10, we observe an exponentially decreasing line that converges at around *-2.363*.

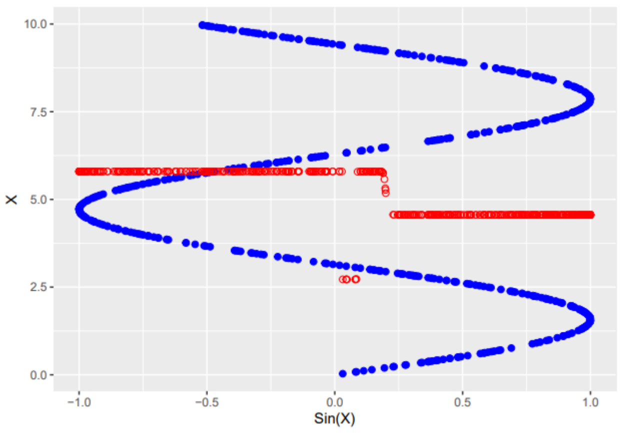


Figure 6

In this last case we tried to fit a neural net to predict x values based on input $\sin(x)$. In this case we use 2 hidden layers containing 3 and 2 neurons. When trying to use the same hidden layers as before the algorithm was not able to converge within the maximum number of steps. As we can see the model fails to predict the true values for x. This is because we train our neural net using predictors that have multiple correct output values. For example if we look at $\sin(x) = 1$ we can either expect x $\approx$ 1.57 or x $\approx$ 7.85.

R Code

Assignment 1

```
temps <- read.csv("temps50k.csv")
stations <- read.csv("stations.csv")

library(geosphere)

Kernel_Methods <- function(p1, p2, predict_date, s_c1, s_c2, s_c3){

  set.seed(1234567890)
  st <- merge(stations,temps,by="station_number")
  date <- predict_date
  # creating a two hour interval for the time sequence

  time_interval = 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")

  # Filter out the posterior data
  filtered_data <- st[as.Date(st$date) < as.Date(date),]

  #Distance function between the points

  calc_real_dist <- function(date,p2){

    p1 <- filtered_data[,c(5,4)]
    physical_distance <- distHaversine(p1, p2)
    return(physical_distance)
  }

  # Creating a function to find the difference
  between the date to predict and day of measure

  diff_in_days <- function(date) {
    day_difference <- as.numeric(difftime(filtered_data$date, date,
    units="days"))
    return(day_difference)
```

```
}


# Creating a function to find the difference between time to
predict and time of measure

diff_in_time <- function(data, hour) {
  hour_difference <- as.numeric(difftime(strptime(data$time,
  format="%H:%M:%S"), strptime(hour, format="%H:%M:%S"),
  units="hours"))
  return (hour_difference)
}


#using above functions

old_temp <- filtered_data$air_temperature
h_distance <- calc_real_dist(date, p2)
h_date <- diff_in_days(date=date)

## Calculating the (gaussian) kernels

dist_kernel <- exp(-(1/s_c1)*(h_distance)^2)
date_kernel <- exp(-(1/s_c2)*(h_date)^2)


summed_forcast <- vector(length=length(time_interval))
prod_forcast <- vector(length=length(time_interval))
for(i in 1:length(time_interval)){

  h_times <- diff_in_time(data=filtered_data,
  hour= time_interval[i])
  time_kernel <- exp(-(1/s_c3)*(h_times)^2)

  summed_kernel <- dist_kernel + date_kernel + time_kernel
  summed_kernel_w_temp <-  summed_kernel * old_temp

  prod_kernel <- dist_kernel %*% date_kernel %*% time_kernel
  prod_kernel_w_temp <- prod_kernel %*% old_temp

  summed_forcast[i] <-
```

```
    round(sum(summed_kernel_w_temp)/(sum(summed_kernel)),2)
    prod_forcast[i] <- round(prod_kernel_w_temp/(sum(prod_kernel)),2)
  }

  ret <- data.frame(Summed = summed_forcast,
  Prod = prod_forcast, Time = time_interval)

  return(ret)
}
Temp <- Kernel_Methods(p1=c(12.8203,55.3836),p2=c(12.9843,
55.6049),predict_date="1996-02-08",
s_c1 = 10000,s_c2=10,s_c3=20)

library(ggplot2)
ggplot(data=Temp, aes(x=Time)) +
  geom_point(aes(y = Summed)) +
  ylab("Temperature") +
  ggtitle("Temperature via summed Kernel") +
  theme(plot.title = element_text(hjust = 0.5))
ggplot(data=Temp, aes(x=Time)) +
  geom_point(aes(y = Prod)) +
  ylab("Temperature") +
  ggtitle("Temperature via multiplicative Kernel") +
  theme(plot.title = element_text(hjust = 0.5))
```

Assignment 2

```
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
t
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
t
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
t
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
t
```

Assignment 3

```
library(neuralnet)
library(ggplot2)

# data preparation
set.seed(1234567890)
sample_10 = runif(500, 0, 10)
mydata = data.frame(X=sample_10, Sin=sin(sample_10))
train = mydata[1:25,]
test = mydata[26:500,]

# Generate random startweights [-1,1]
set.seed(1234567890)
startweights = runif(1000,-1,1)

# Train neural net
nn = neuralnet(Sin~X, data=train, hidden=c(10,5),
startweights=startweights)

# Calculate SSE on training data
train_sse = sum((unlist(nn$net.result) - train[,2])^2)/2
cat("Train SSE:", round(train_sse, 4))

# Compute predictions on test data
test_predictions = predict(nn, test)
results = data.frame(X=test$X, actual=test$Sin,
prediction=test_predictions)

# Plot train values (yellow), test values (blue)
and test predictions (red)
ggplot() +
  geom_point(aes(x=train$X, y=train$Sin), color="yellow", size=5)+
  geom_point(aes(x=test$X, y=test$Sin), color="blue", size=2) +
  geom_point(aes(x=results$X, y=results$prediction), shape=1,
  color="red", size=2) +
  xlab("X") + ylab("Sin(X)")

# data preparation
set.seed(1234567890)
```

```
sample_20 = runif(500, 0, 20)
mydata20 = data.frame(X=sample_20, Sin=sin(sample_20))

# Compute predictions on new test data
test_predictions_20 = predict(nn, mydata20)
results_20 = data.frame(X=mydata20$X, actual=mydata20$Sin,
prediction=test_predictions_20)

# Plot train values (yellow), test values (blue) and
test predictions (red)
ggplot() + xlim(0, max(mydata20[,1])) +
 ylim(min(results_20$prediction), max(results_20$prediction)) +
  geom_point(aes(x=train$X, y=train$Sin), color="yellow", size=5) +
  geom_point(aes(x=results_20$X, y=results_20$actual),
  color="blue", size=2) +
  geom_point(aes(x=results_20$X, y=results_20$prediction),
  shape=1, color="red", size=2) +
  xlab("X") + ylab("Sin(X)")

 # Generate random startweights [-1,1]
set.seed(1234567890)
startweights = runif(1000,-1,1)
nn_x = neuralnet(X~Sin, data=mydata, hidden=c(3,2),
startweights=startweights)

# Compute predictions on test data
test_predictions_x = predict(nn_x, mydata)
results_x = data.frame(X=mydata$Sin, Y=mydata$X,
prediction=test_predictions_x)

# Plot train values (blue) and train predictions (red)
ggplot() +
  geom_point(aes(x=results_x$X, y=results_x$Y), color="blue", size=2)
  + geom_point(aes(x=results_x$X, y=results_x$prediction),
  shape=1, color="red", size=2) +
  xlab("Sin(X)") + ylab("X")
```