Machine Learning - Block 1 Assignment 3

Agustín Valencia

12/13/2019

Kernel Methods

Implement a kernel method to predict the hourly temperatures for a date and place in Sweden. To do so, you are provided with the files `stations.csv` and `temps50k.csv`. These files contain information about weather stations and temperature measurements in the stations at different days and times. The data have been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI).

You are asked to provide a temperature forecast for a date and place in Sweden. The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels:

- The first to account for the distance from a station to the point of interest.
- The second to account for the distance between the day a temperature measurement was made and the day of interest.
- The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Choose an appropriate smoothing coefficient or width for each of the three kernels above. Answer to the following questions:

- 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.
- 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.

Note that the file `temps50k.csv` may contain temperature measurements that are posterior to the day and hour of your forecast. You must filter such measurements out, i.e. they cannot be used to compute the forecast.

Solution - Kernel Methods

Let $\mathbf{obs} = (position, date, time)$ denote an observation where:

$$position = (latitude, longitude)$$

$$date = Year, Month, Day$$

$$time = Hour :: Minutes :: Seconds$$

In the same sense it can be defined $\mathbf{pred} = (position, date, time)$ the non-observed data desired to be predicted.

Let the distance functions :

$$d_1 = d_{pos}(a, b) = \sqrt{(long_a - long_b)^2 + (lat_a - lat_b)^2}$$

$$d_2 = d_{date}(a, b) = \mathbf{mod}(|date_a - date_b|, 365)$$

$$d_3 = d_{time}(a, b) = \mathbf{mod}(|time_a - time_b|, 24)$$

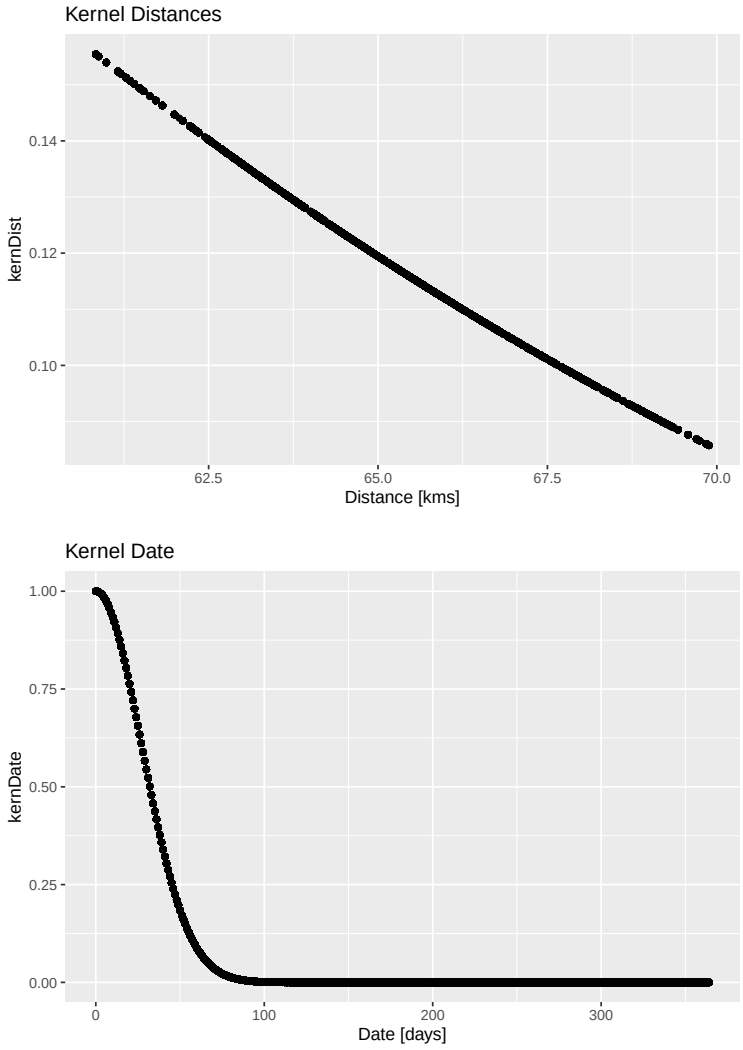
The following Gaussian Kernels can be defined:

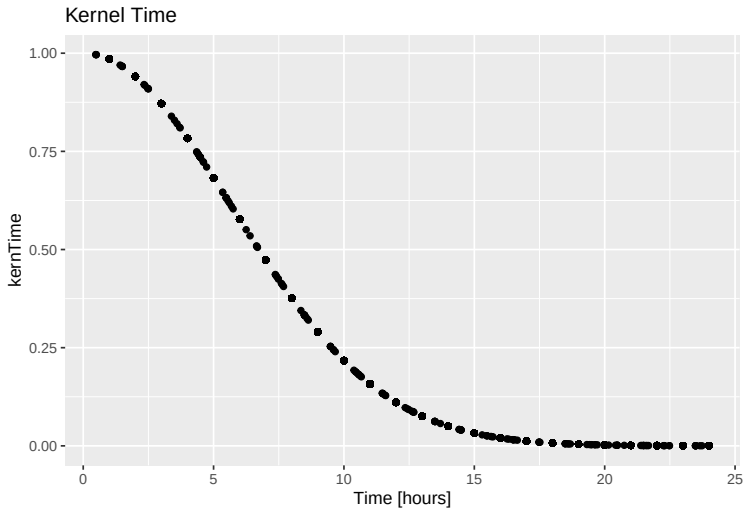
$$\phi_1 = \phi_{pos} = e^{\frac{-|d_{pos}(\mathbf{pred},\mathbf{obs})|^2}{2\sigma^2 h_{distance}}}$$

$$\phi_2 = \phi_{date} = e^{\frac{-|d_{date}(\mathbf{pred},\mathbf{obs})|^2}{2\sigma^2 h_{date}}}$$

$$\phi_3 = \phi_{date} = e^{\frac{-|d_{time}(\mathbf{pred},\mathbf{obs})|^2}{2\sigma^2 h_{time}}}$$

For the given date they are computed as following:





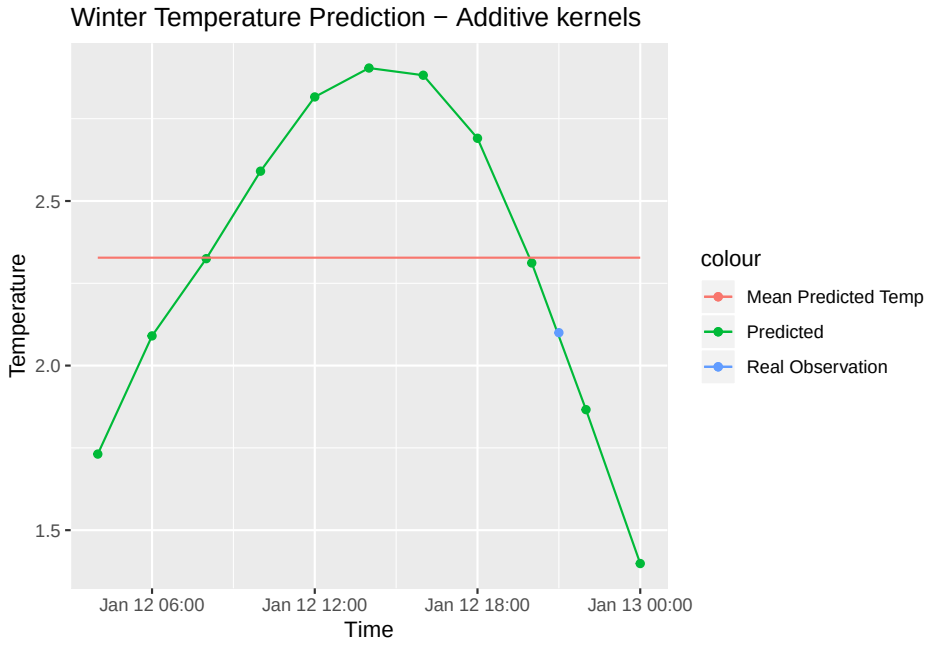
It can be observed from the kernels shape how the closest observations incide more in the prediction than the ones far from the prediction target in terms of temporal and spatial distances by using the beforementioned distance functions.

If the predictions for a day in winter are computed as

$$y = \frac{1}{N}\sum_{i=1}^{3}\phi_i\left(\frac{d_i(\mathbf{obs},\mathbf{pred})}{h_i}\right)$$

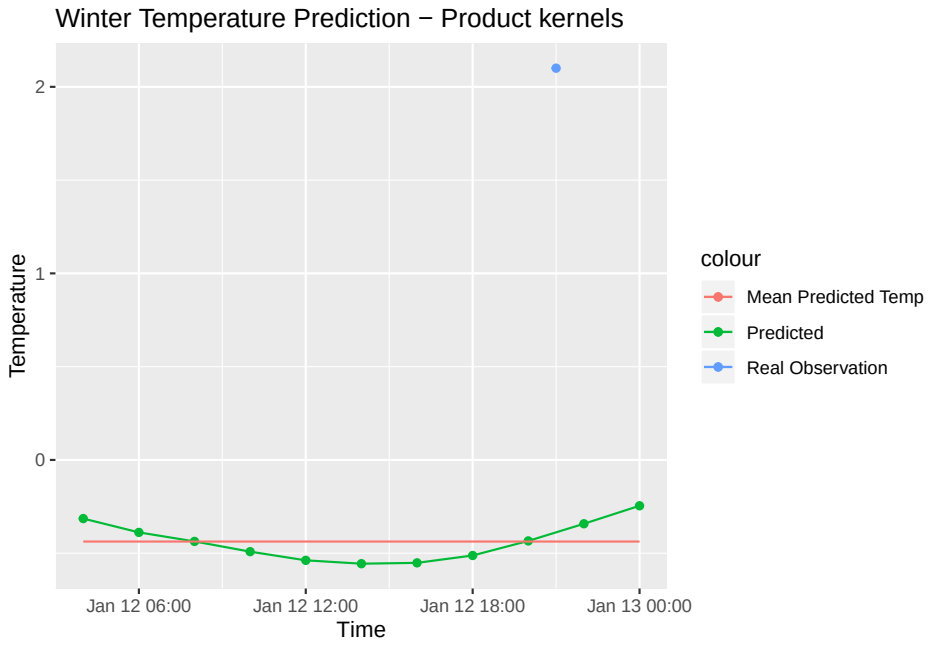
The following results are obtained

```
## The closer station to (66.3002,19.2) is 614
## The temperature measures for 16463 is/are 2.1
## The mean predicted temperature for 16463 using sum of kernels is 2.327836
## The mean predicted temperature for 16463 using product of kernel is -0.4374
```



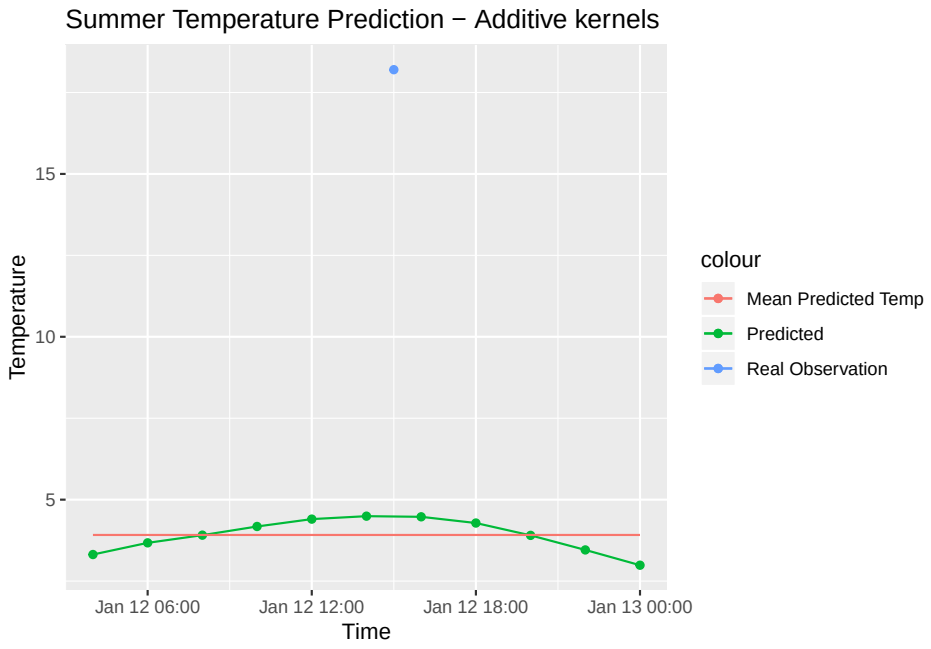
On the other hand, if the kernels are multiplied instead of added:

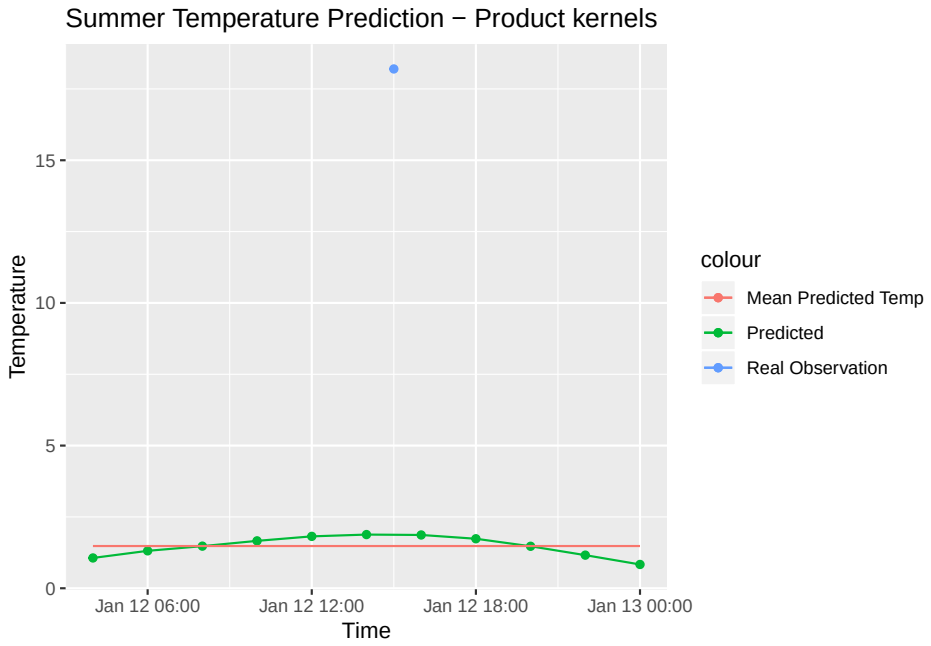
$$y = \frac{1}{N} \prod_{i=1}^{3} \phi_i \left(\frac{d_i(\mathbf{obs}, \mathbf{pred})}{h_i} \right)$$



If the same exercise is done for a day in summer it is obtained that:

```
## The closer station to (66.3002,19.2) is 135
## The temperature measures for 16642 is/are 18.2
## The mean predicted temperature for 16642 using sum of kernels is 3.915799
## The mean predicted temperature for 16642 using product of kernel is 1.479926
```





It can be seen that the predictions done by adding the different kernels seem to have better results than the ones multiplying them. This might be intepreted as assuming each kernel as independent from the others, the errors will also be independent. As per the multiplication case, errors from one weak prediction will affect the other errors and therefore have a big impact in the overall error.

Nonetheless there might be some dependencies in the predictors as seasons over the years or temperatures changes during day and night, for this case it is better to assume them as independent and get a lower overall error.

Support Vector Machines

Use the function `ksvm` from the R package `kernlab` to learn a SVM for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5.

This implies that you have to consider three models.

- Perform model selection, i.e. select the most promising of the three models (use any method of your choice except cross-validation or nested cross-validation).
- Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or nested cross-validation).
- Produce the SVM that will be returned to the user, i.e. show the code.
- What is the purpose of the parameter C ?

Solution - Support Vector Machines

The spam dataset has been split in a ratio of 70%/30% for training and testing purposes respectively.

In order to compare the predictors performance, it have been ran experiments to get the training error, number of support vectors, true positives , false negative , false positive, false negative rates and overall misclassification from all the predictors

The code of the experiments is detailed in Appendix B and the results are summarized in the following table (rates are given in % format).

C	trainingError	numSV	TPR	TNR	FPR	FNR	Misclassification
0.5	4.41	1365	93.09	91.11	6.91	8.89	8.18
1.0	3.76	1278	91.76	92.08	8.24	7.92	8.04
5.0	1.96	1187	92.32	92.21	7.68	7.79	7.75

Taking into account that our classifier is meant to detect spam emails, it has to be noticed that, from the users perspective, loosing true email in the spambox is way worse than getting spam email in the inbox. That is measured by the False Positive Rate, for which $C = 0.5$ get the best score.

Analyzing the overall misclassification it is seen that $C = 5.0$ gets the better score, though, it also gets a higher False Positive rate than $C = 0.5$, which has been said is the most important score for this application. Thus, we drop it anyway.

Regarding the amount of support vectors chosen by the model training, one might tend to think that the highest the amount of support vectors the overfitted the model, which is not false at all, though it is not always true. Having a wider soft margin might also produce more support vectors in the sense of the dynamics of *pushing* (support vectors in the right side of the hyperplane) and *pulling* (support vectors in the wrong side of the hyperplane). This can be interpreted as the highest training error for model $C = 0.5$

Therefore the chosen model to be deployed into the customer's system would be the following:

```
svmToDeploy <- ksvm(
    type ~ .,
    data=spam,
    type="C-svc",
    kernel="rbfdot",
    C = 0.5,
    kpar = list(sigma = 0.05)
)
```

Parameter C denotes the inverse of the regularization penalization. The bigger the C the smaller the soft

margin between the classes to learn, thus, the more strict the learning process which could lead to overfit the data. Fortunately that is not the case for this classifier, since it gets even better scores under testing data than the more relaxed models.

Cross-validation still recommended in order to get the best results possible for the classifier, though for this assignment we are not allowed to perform that type of analisys.

Neural networks

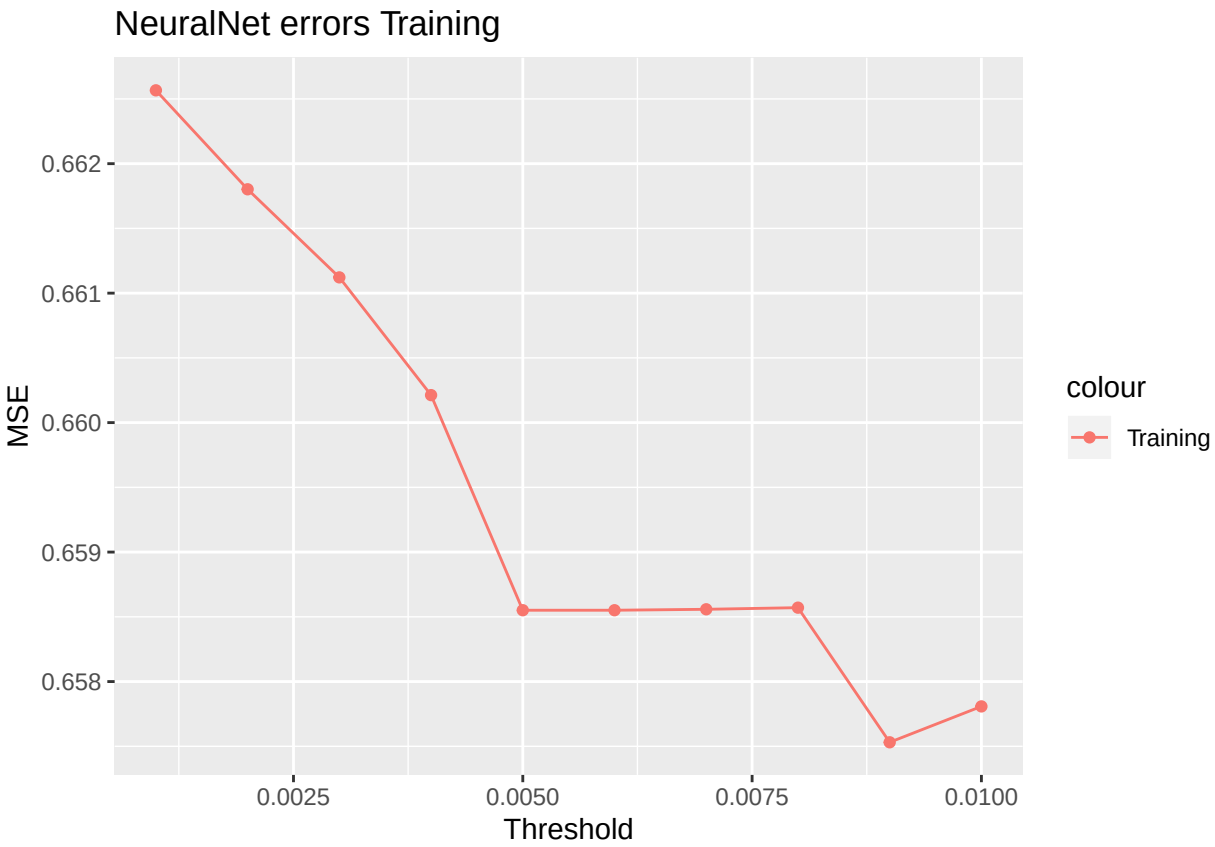
Train a neural network to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0,10]. Apply the sine function to each point. The resulting pairs are the data available to you. Use 25 of the 50 for training and the rest for validation. The validation set is used for early stop of the gradient descent. That is, you should use the validation set to detect when to stop the gradient descent and so avoid overfitting.

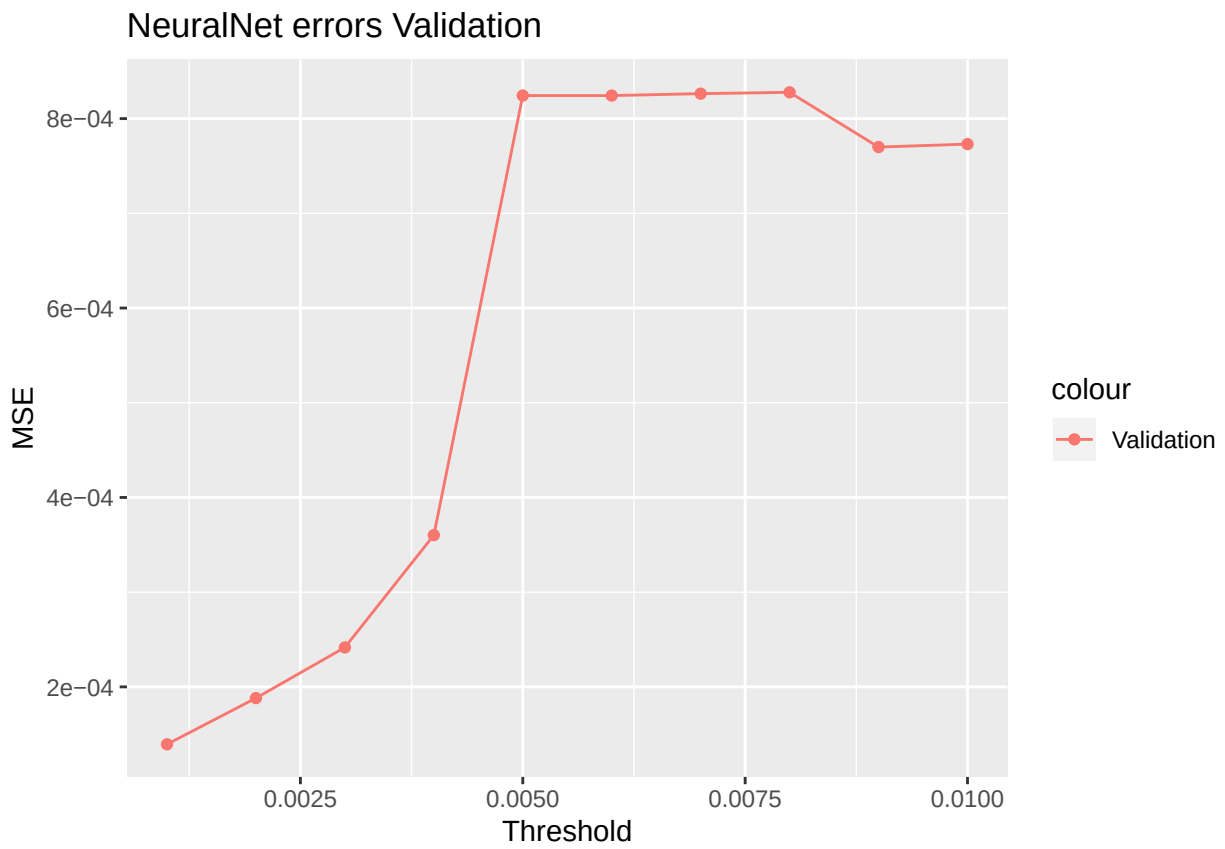
Stop the gradient descent when the partial derivatives of the error function are below a given threshold value. Check the argument `threshold` in the documentation. Consider threshold $i/1000$ with $i = 1, ..., 1000$. Initialize the weights of the neural network to random values in the interval [-1,1]. Use a neural network with a single hidden layer of 10 units. **Use the default values for the arguments not mentioned here**. Choose the most appropriate value for the threshold. Motivate your choice. Provide the final neural network learned with the chosen threshold. Feel free to use the following template:

```
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# Random initialization of the weights in the interval [-1, 1]
winit <- # Your code here
for(i in 1:10) {
  nn <- neuralnet(# Your code here)
  # Your code here
}
plot(nn <- neuralnet(# Your code here))
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1)
points(trva, col = "red")
```

Solution - Neural Networks

After experimenting with the given thresholds, the following errors have been found


NeuralNet errors Training
MSE
0.662
0.661
0.660
0.659
0.658
0.0025
0.0050
0.0075
0.0100
Threshold
colour
Training



As seen, although that the highest training error is obtained at $i = 1$, also the best validation error occurs at that treshold. This is the best scenario because we have a good generalization (given the training error) and a nice behavior in unseen data.

Thus, the final network to deliver will be

```
nn <- neuralnet (
    Sin ~ Var,
    data = tr,
    hidden=10,
    threshold = 1/1000,
    startweights = winit
)
```

Checking its predictions

Appendix A : Code for Assignment 1 - Kernel Methods

```
stations <- read.csv("data/stations.csv")
temps <- read.csv("data/temps50k.csv")
st <-merge(stations, temps, by="station_number")
h_distance <- 500
h_date <- 7
h_time <- 5
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")



# utils functions
getPositionDistance <- function(posLatVect, posLongVect, a, b) {
    res <- sqrt((posLatVect - a)^2 + (posLongVect - b)^2)
    return(res)
}

getDateDistance <- function(dateVect, now) {
    res <- as.numeric(difftime(dateVect, now, units="days")) %% 365
    return(res)
}

getTimeDistance <- function(timeVect, now) {
    d <- as.numeric(difftime(timeVect, now, units="hours"))
    d <- abs(ifelse(d < 12, d, 24-d))
    return(d)
}

# kernel estimations
kernelPredict <- function(data, a, b, date, h_distance, h_date, h_time, times) {
    diffDistance <- getPositionDistance(data$longitude, data$latitude, a, b)
    diffDate <- getDateDistance(data$date, date)
    kernDist <- exp(-abs(diffDistance)^2/(h_distance*2*sd(diffDistance)))
    kernDate <- exp(-abs(diffDate)^2/(h_date*2*sd(diffDate)))
    predDist <- 1/nData * sum(kernDist * data$air_temperature)
    predDate <- 1/nData * sum(kernDate * data$air_temperature)
    predTempSum <- vector(length = length(times))
    predTempProd <- vector(length = length(times))
    for(i in 1:length(times)) {
        diffTime <- getTimeDistance(data$time, times[i])
        kernTime <- exp(-abs(diffTime)^2/(h_time*2*sd(diffTime)))
        predTime <- 1/nData * sum(kernTime * data$air_temperature)
        predTempSum[i] <- (predDist + predDate + predTime)
        predTempProd[i] <- (predDist * predDate * predTime)
    }
    return( list(
                predTempSum = predTempSum,
                predTempProd = predTempProd,
                diffDistance = diffDistance,
                diffDate = diffDate,
                diffTime = diffTime,
```

```
                kernDist = kernDist,
                kernDate = kernDate,
                kernTime = kernTime
            )
    )
}

# Date to predict
date <- "2015-01-28"
a <- 66.30020
b <- 19.20000

# Filter the future
st$date <- as.Date(st$date)
date <- as.Date(date)
data <- st[which(st$date < date),]
nData <- nrow(data)
times <- strptime(times, format = "%H:%M:%S")
data$time <- strptime(data$time, format = "%H:%M:%S")
resWinter <- kernelPredict(data, a, b, date, h_distance, h_date, h_time, times)
predTempSum <- resWinter$predTempSum
predTempProd <- resWinter$predTempProd
diffDistance <- resWinter$diffDistance
diffDate <- resWinter$diffDate
diffTime <- resWinter$diffTime
kernDist <- resWinter$kernDist
kernDate <- resWinter$kernDate
kernTime <- resWinter$kernTime
# Kernel plotting
ggplot() + geom_point(aes(x=diffDistance, y=kernDist)) +
    ggtitle("Kernel Distances") + xlab("Distance [kms]")
ggplot() + geom_point(aes(x=diffDate, y=kernDate)) +
    ggtitle("Kernel Date") + xlab("Date [days]")
ggplot() + geom_point(aes(x=diffTime, y=kernTime)) +
    ggtitle("Kernel Time") + xlab("Time [hours]")
# Getting real temperatures for the given date
trueData <- st[which(st$date == date),]
# Getting closer station to compare against
realDists <- getPositionDistance(trueData$longitude, trueData$latitude, a, b)
closerStationNum  <- trueData[which.min(realDists),]$station_number
closerStationName <- trueData[which.min(realDists),]$station_name
trueData <- trueData[which(trueData$station_number == closerStationNum),]
realTemps <- trueData$air_temperature
realTimes <- trueData$time
meanPredictedSum <- rep(mean(predTempSum), length(times))
meanPredictedProd <- rep(mean(predTempProd), length(times))
realTimes <- as.POSIXct(strptime(realTimes, format = "%H:%M:%S"))
posixTimes <- as.POSIXct(times)

cat("The closer station to (",a,",",b,") is ", closerStationName,"\n", sep="")
cat("The temperature measures for ", date," is/are ", realTemps,"\n", sep="")
cat("The mean predicted temperature for ", date," using sum of kernels is ",
    mean(predTempSum),"\n", sep="")
```

```
cat("The mean predicted temperature for ", date," using product of kernel is ",
    mean(predTempProd),"\n", sep="")

ggplot() +
    geom_line(aes(x=posixTimes, y=predTempSum, color="Predicted")) +
    geom_point(aes(x=posixTimes, y=predTempSum, color="Predicted")) +
    geom_line(aes(x=posixTimes, y=meanPredictedSum, color="Mean Predicted Temp")) +
    geom_point(aes(x=realTimes, y=realTemps, color="Real Observation")) +
    ggtitle("Winter Temperature Prediction - Additive kernels") +
    xlab("Time") + ylab("Temperature")
ggplot() +
    geom_line(aes(x=posixTimes, y=predTempProd, color="Predicted")) +
    geom_point(aes(x=posixTimes, y=predTempProd, color="Predicted")) +
    geom_line(aes(x=posixTimes, y=meanPredictedProd, color="Mean Predicted Temp")) +
    geom_point(aes(x=realTimes, y=realTemps, color="Real Observation")) +
    ggtitle("Winter Temperature Prediction - Product kernels") +
    xlab("Time") + ylab("Temperature")
# Date to predict
date <- "2015-07-26"

# Filter the future
st$date <- as.Date(st$date)
date <- as.Date(date)
data <- st[which(st$date < date),]
nData <- nrow(data)
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00",
           "16:00:00", "18:00:00", "20:00:00", "22:00:00", "24:00:00")
times <- strptime(times, format = "%H:%M:%S")
data$time <- strptime(data$time, format = "%H:%M:%S")
resSummer <- kernelPredict(data, a, b, date, h_distance, h_date, h_time, times)
predTempSum <- resSummer$predTempSum
predTempProd <- resSummer$predTempProd
diffDistance <- resSummer$diffDistance
diffDate <- resSummer$diffDate
diffTime <- resSummer$diffTime
kernDist <- resSummer$kernDist
kernDate <- resSummer$kernDate
kernTime <- resSummer$kernTime

# Getting real temperatures for the given date
trueData <- st[which(st$date == date),]
# Getting closer station to compare against
realDists <- getPositionDistance(trueData$longitude, trueData$latitude, a, b)
closerStationNum  <- trueData[which.min(realDists),]$station_number
closerStationName <- trueData[which.min(realDists),]$station_name
trueData <- trueData[which(trueData$station_number == closerStationNum),]
realTemps <- trueData$air_temperature
realTimes <- trueData$time
meanPredictedSum <- rep(mean(predTempSum), length(times))
meanPredictedProd <- rep(mean(predTempProd), length(times))
realTimes <- as.POSIXct(strptime(realTimes, format = "%H:%M:%S"))
posixTimes <- as.POSIXct(times)
```

```
cat("The closer station to (",a,",",b,") is ", closerStationName,"\n", sep="")
cat("The temperature measures for ", date," is/are ", realTemps,"\n", sep="")
cat("The mean predicted temperature for ", date," using sum of kernels is ",
    mean(predTempSum),"\n", sep="")
cat("The mean predicted temperature for ", date," using product of kernel is ",
    mean(predTempProd),"\n", sep="")

ggplot() +
    geom_line(aes(x=posixTimes, y=predTempSum, color="Predicted")) +
    geom_point(aes(x=posixTimes, y=predTempSum, color="Predicted")) +
    geom_line(aes(x=posixTimes, y=meanPredictedSum, color="Mean Predicted Temp")) +
    geom_point(aes(x=realTimes, y=realTemps, color="Real Observation")) +
    ggtitle("Summer Temperature Prediction - Additive kernels") +
    xlab("Time") + ylab("Temperature")


ggplot() +
    geom_line(aes(x=posixTimes, y=predTempProd, color="Predicted")) +
    geom_point(aes(x=posixTimes, y=predTempProd, color="Predicted")) +
    geom_line(aes(x=posixTimes, y=meanPredictedProd, color="Mean Predicted Temp")) +
    geom_point(aes(x=realTimes, y=realTemps, color="Real Observation")) +
    ggtitle("Summer Temperature Prediction - Product kernels") +
    xlab("Time") + ylab("Temperature")
```

Appendix B : Code for Assignment 2 - Support Vector Machines

```
## Utils
splitData <- function(data, trainRate) {
    n <- dim(data)[1]
    idxs <- sample(1:n, floor(trainRate*n))
    train <- data[idxs,]
    test <- data[-idxs,]
    return (list(train = train, test = test))
}

get_performance <- function(targets, predictions, text) {
    t <- table(targets, predictions)
    tn <- t[1,1]
    tp <- t[2,2]
    fp <- t[1,2]
    fn <- t[2,1]
    total <- sum(t)
    tpr <- tp/(tp+fp) * 100
    tnr <- tn/(tn+fn) * 100
    fpr <- fp/(tp+fp) * 100
    fnr <- fn/(tn+fn) * 100

    return (
        list(
            tpr = tpr,
            tnr = tnr,
            fpr = fpr,
            fnr = fnr,
            misclass = (fp+fn)/total * 100
        )
    )
}

# Data split
data(spam)
split <- splitData(spam, .7)
train <- split$train
test  <- split$test

train.x <- train[,-ncol(train)]
train.y <- train[,ncol(train)]
test.x  <- test[,-ncol(test)]
test.y  <- test[,ncol(test)]

Cs <- c(.5, 1, 5)
kWidth <- 0.05
svmModels <- list()
svmScores <- data.frame (
    C = vector(length = 3),
    trainingError = vector(length = 3),
    numSV = vector(length = 3),
    TPR = vector(length = 3),
    TNR = vector(length = 3),
```

```
    FPR = vector(length = 3),
    FNR = vector(length = 3),
    Misclassification = vector(length = 3)
)
for (i in 1:length(Cs)) {
    svmModel <- ksvm(
                    type ~ .,
                    data=train,
                    type="C-svc",
                    kernel="rbfdot",
                    C = Cs[i],
                    kpar = list(sigma = kWidth)
                )
    predictions <- predict(svmModel, test)
    performance <- get_performance(test.y, predictions)
    svmScore <- c(
                    Cs[i],
                    error(svmModel) * 100,
                    nSV(svmModel),
                    performance$tpr,
                    performance$tnr,
                    performance$fpr,
                    performance$fnr,
                    performance$misclass
                  )
    svmScores[i,] <- svmScore
    svmModels[[i]] <- svmModel
}
kable(svmScores, digits = 2)
```

Appendix C : Code for Assignment 3 - Neural Networks

```
getMSE <- function (target, prediction) {
    if(length(target) != length(prediction)) {
        stop("lengths must match")
    }
    res <- mean((target-prediction)^2)
    return(res)
}

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

# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
trError <- vector(length=10)
valError <- vector(length=10)
for(i in 1:10) {
  nn <- neuralnet(
      Sin ~ Var,
      data = tr,
      hidden=10,
      threshold = i/1000,
      startweights = winit
    )
  trPred <- predict(nn, tr)
  valPred <- predict(nn, va)
  trError[i] <- getMSE(va$Sin, trPred)
  valError[i] <- getMSE(va$Sin, valPred)
}

pTr <- ggplot() + ggtitle("NeuralNet errors Training") +
        geom_point(aes(x=(seq(1:10)/1000), y=trError, color="Training")) +
        geom_line(aes(x=(seq(1:10)/1000), y=trError, color="Training")) +
        xlab("Threshold") + ylab("MSE")
pVal <- ggplot() + ggtitle("NeuralNet errors Validation") +
        geom_point(aes(x=(seq(1:10)/1000), y=valError, color="Validation")) +
        geom_line(aes(x=(seq(1:10)/1000), y=valError, color="Validation")) +
        xlab("Threshold") + ylab("MSE")
pTr
pVal

#plot(nn <- neuralnet(# Your code here))
## Plot of the predictions (black dots) and the data (red dots)
#plot(prediction(nn)$rep1)
#points(trva, col = "red")

nn <- neuralnet (
    Sin ~ Var,
    data = tr,
    hidden=10,
```

```
    threshold = 1/1000,
    startweights = winit
)
```

Appendix D : Environment setup Code

```
knitr::opts_chunk$set(echo = FALSE)
suppressWarnings(RNGversion('3.5.1'))
set.seed(1234567890)
library(geosphere)
library(kernlab)
library(knitr)
library(ggplot2)
library(neuralnet)
```