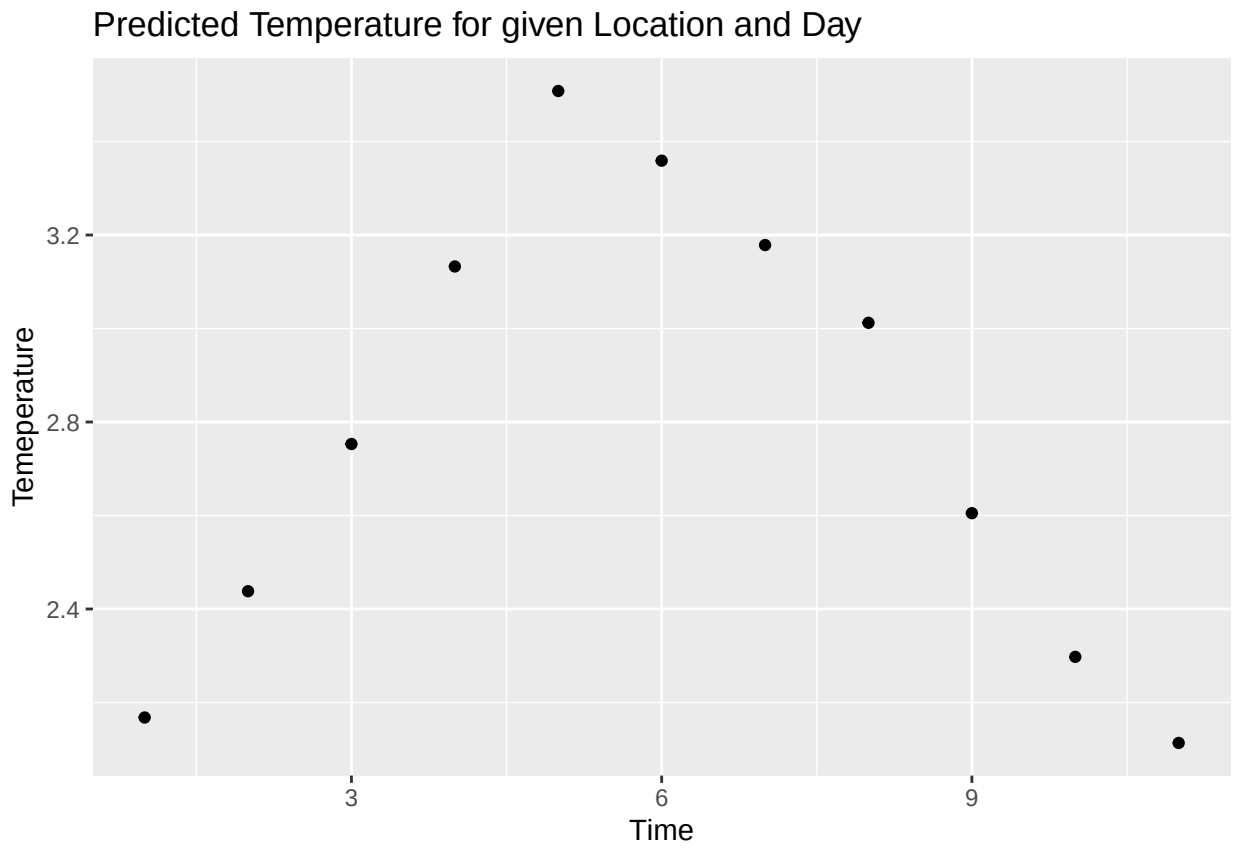
Lab 3 Block 1

Aman Kumar Nayak (amana551)

1/02/2020

Part 1

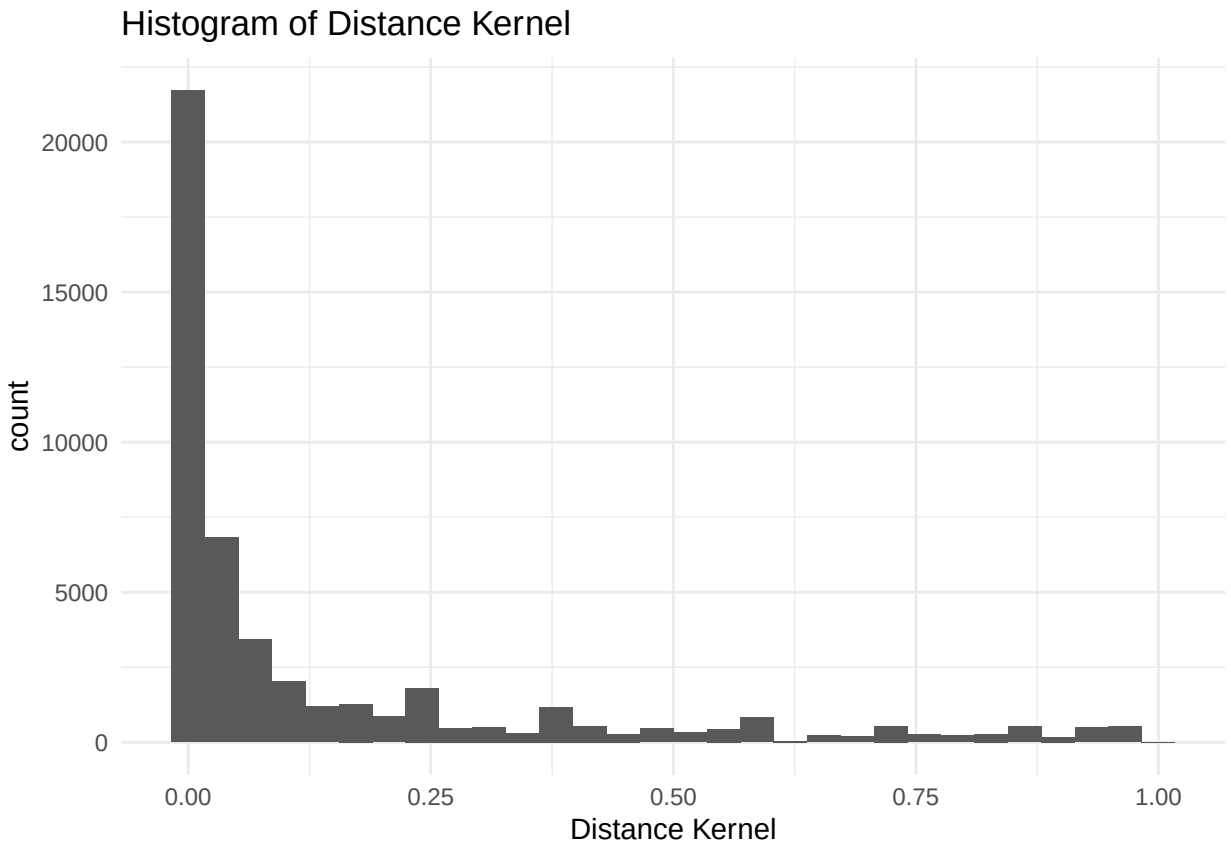
Distribution of temperature predicted for the given day and location. We have selected date as "2015-03-03"



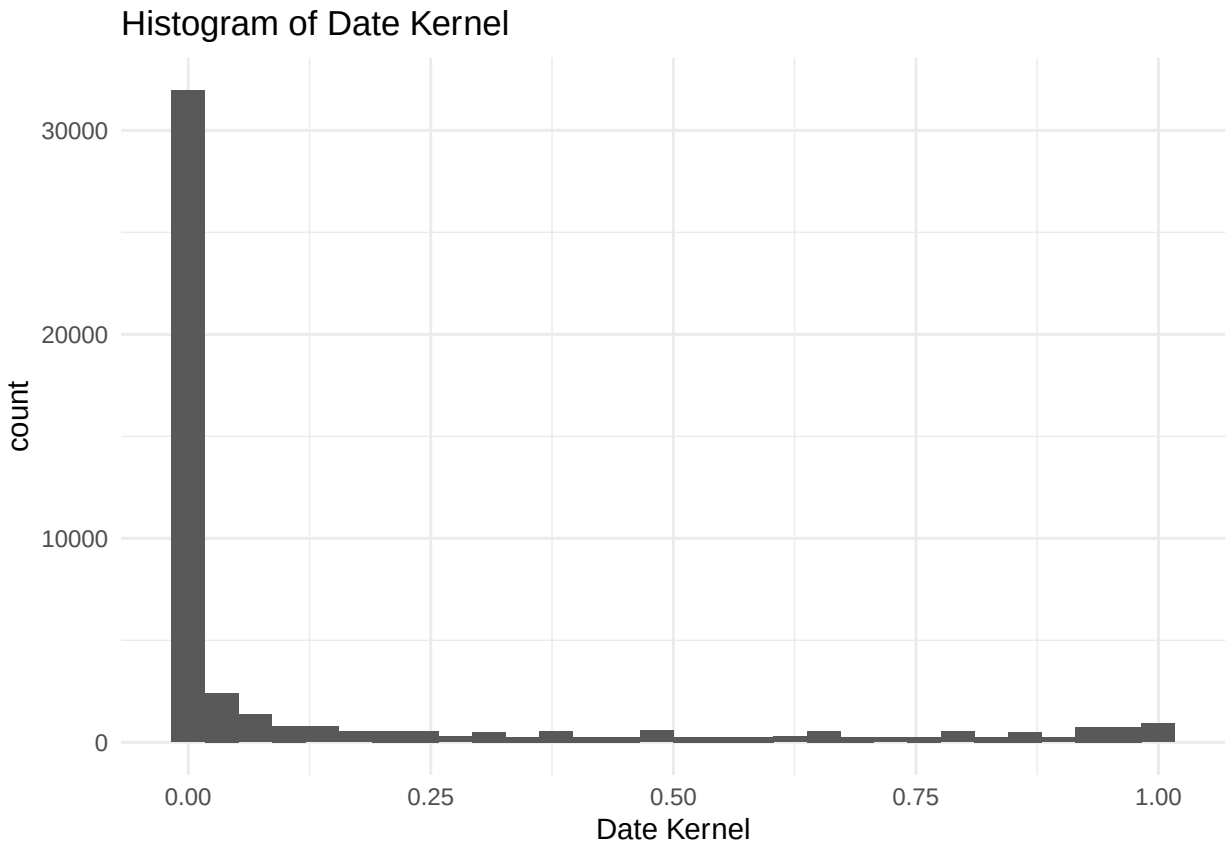
Choice Of Kernel Width

Kernel width for *distance kernel* selected as 200000 (200km) which was selected in order assign more weight to stations which is close to target location.

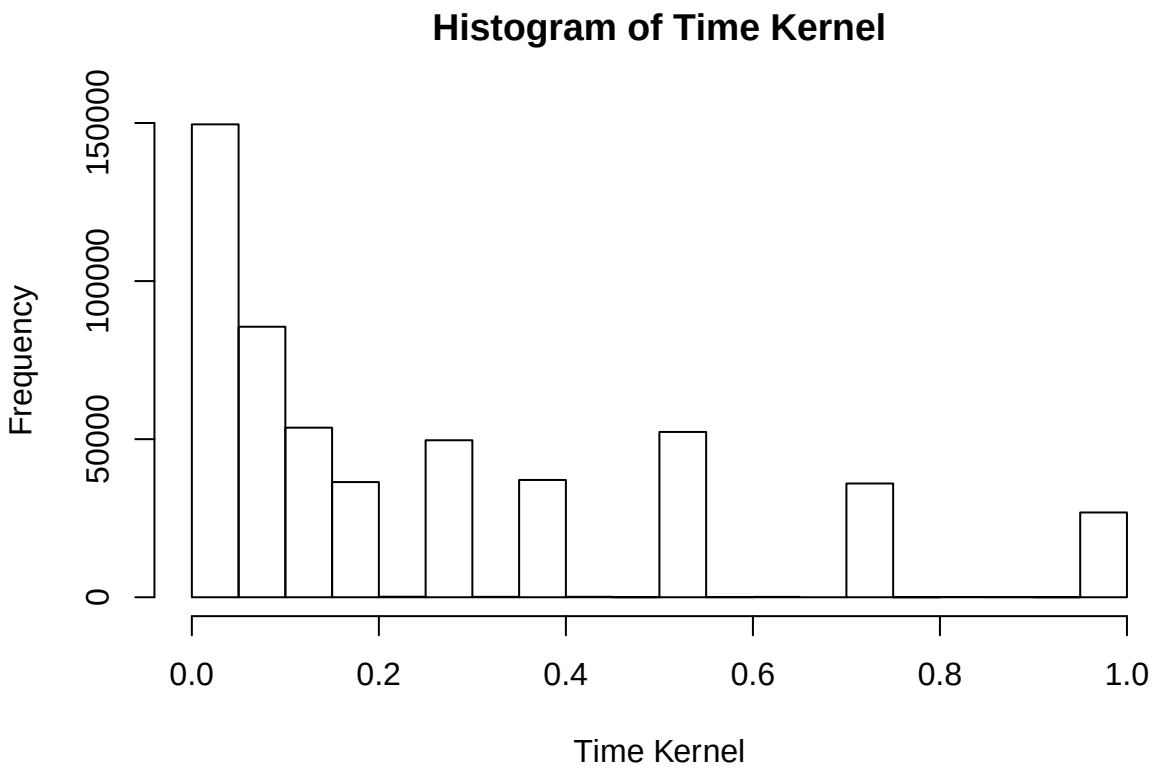
Below histograms of kernels suggest that kernel value represented in terms of the probability density function is close to 0 for the maximum number of stations that are further away from target location while stations close to target location given more weight.



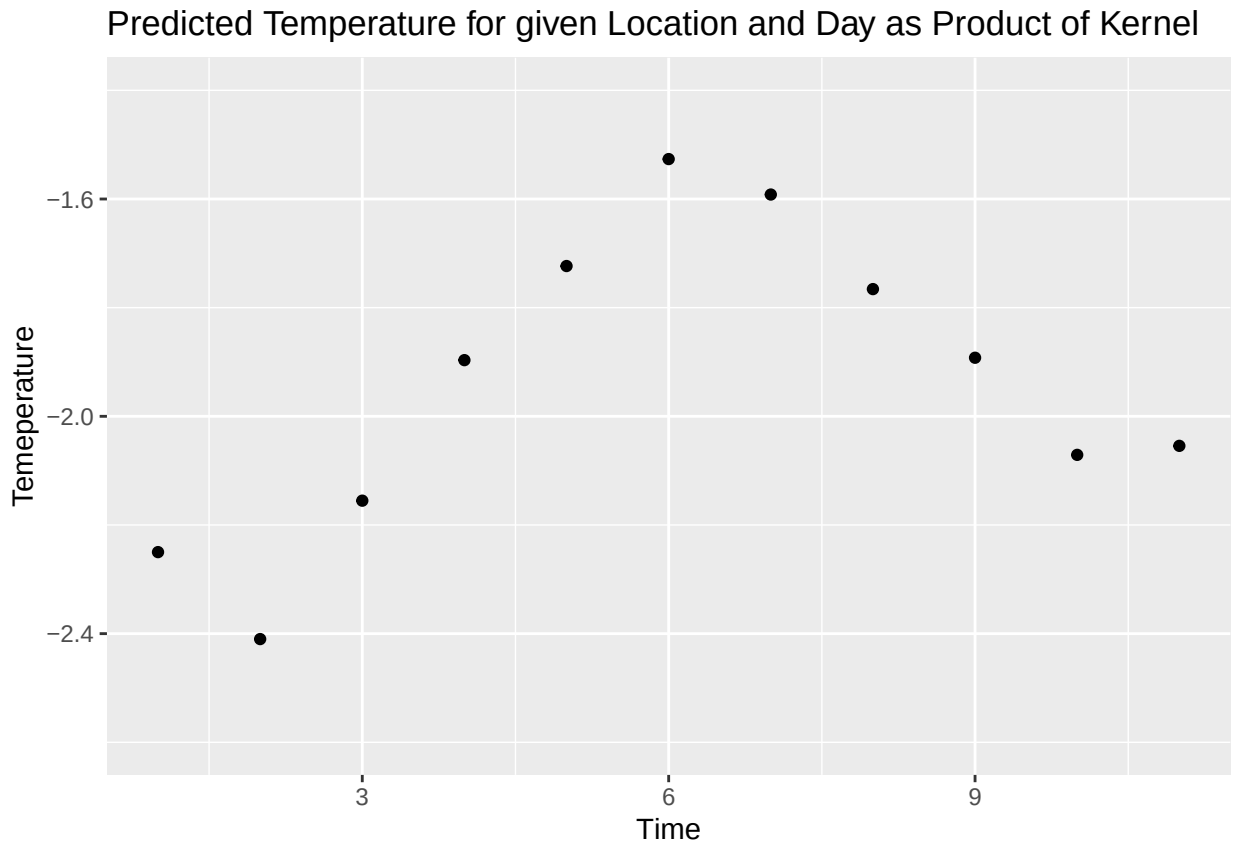
Kernel width of date kernel selected as 30. Thus more weight is assigned to date, which is in 10 days proximity with a target date and rest are assigned small weight.



Kernel width of time kernel selected as 3 in order to give more weight in the 3hour cycle on both side.



As the product of kernel



In the product of kernels, we got range as -2.4 to -1.5, which is slightly higher in terms of prediction range when compared with the sum of the kernel but values are in the negative range.

Reason for change is that when kernel with minimal values or negative value multiplied with extremely low or high value, the kernel value changes drastically whereas in case of sum kernel is not affected to such sudden changes.

##**Part 2**

Divided data between train(50% of original data) , validate (25% original data) and test (25% of original data).

```
library(kernlab)
data("spam")
suppressWarnings(RNGversion("3.5.1"))
set.seed(123456789)
data = spam

#data$type = as.factor(data$type)
n = length(data$type)

id = sample(1:n , floor(n*.50))
train = data[id,]

#Validate
id1 = setdiff(1:n,id)
set.seed(123456789)
id2 = sample(id1 , floor(n*.25))
```

```
valid = data[id2 , ]

#Test
id3 = setdiff(id1 , id2)
test = data[id3 , ]
```

2.1

Below is table of comparison table for **different miscalculation rate** in case of SVM with different soft penalty factor C

```
#At C = 0.5
model_C0.5 = ksvm(x = as.matrix(train[,-58]) , y = train[,58] ,
                  kernel = "rbfdot" , kpar=list(sigma = .05), C = 0.5 )

pred0.5 = predict(model_C0.5 , newdata = valid[,-58])

con0.5 = table(pred0.5, valid[,58])

misCal0.5 = 1 - (sum(diag(con0.5))/sum(con0.5))

#At C = 1
model_C1 = ksvm(x = as.matrix(train[,-58]) , y = train[,58] ,
                kernel = "rbfdot" , kpar=list(sigma = .05), C = 1 )

pred1 = predict(model_C1 , newdata = valid[,-58])

con1 = table(pred1, valid[,58])

misCal1 = 1 - (sum(diag(con1))/sum(con1))


#At C = 5
model_C5 = ksvm(x = as.matrix(train[,-58]) , y = train[,58] ,
                kernel = "rbfdot" , kpar=list(sigma = .05), C = 5 )


pred5 = predict(model_C5 , newdata = valid[,-58])

con5 = table(pred5, valid[,58])

misCal5 = 1 - (sum(diag(con5))/sum(con5))

ComparessionTable = data.frame("C" = c("0.5" , "1" , "5") ,
                               "Miss-calculation Rate" = c(misCal0.5 ,
                                                           misCal1 , misCal5))


ComparessionTable
```

```
##     C Miss.calculation.Rate
## 1 0.5            0.08000000
## 2   1            0.07826087
## 3   5            0.07217391
```

Looking at comparison table we can say that with keeping **penalty C = 5** we get lowest misclassification rate which allow us to select this as optimal model.

2.2

Calculating generalization error using Hold Out Method at C=5 for SVM

```
genData = rbind(train , valid)

selectedModel = ksvm(x = as.matrix(genData[,-58]) , y = genData[,58] ,
                     kernel = "rbfdot" , kpar=list(sigma = .05), C = 5 )


pred = predict(selectedModel , newdata = test[,-58])

con = table(pred, test[,58])

misCal = 1 - (sum(diag(con))/sum(con))

print("Generalization Error calculated in terms of Miss-classification Rate")
```

```
## [1] "Generalization Error calculated in terms of Miss-classification Rate"
```

```
misCal
```

```
## [1] 0.06950478
```

Generalization when modeled with Train and Validation data(75% of original data) and tested with Test data(25% of original data) is coming as 0.0695.

2.3

SVM Model which will be reverted to user will contain model which is trained using 100% of original data.

```
userModel = ksvm(x = as.matrix(spam[,-58]) , y = spam[,58] , kernel = "rbfdot" , kpar=list(sigma = .05)
```

```
print("User model details")
```

```
## [1] "User model details"
```

```
userModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 5
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.05
##
## Number of Support Vectors : 1547
##
## Objective Function Value : -2082.468
## Training error : 0.022169
```

2.4

Purpose of C parameter

C is cost parameter which is used to optimize, in order to reduce misclassification during training. Value of C will decide hyper-plane margin, if large value of C is chosen (as chosen above, in context to available 3 values), it will produce smaller-margin hyper-plane while classifying the data and on the other hand if smaller value of C is chosen than optimizer will produce comparatively larger-margin hyper-plane.

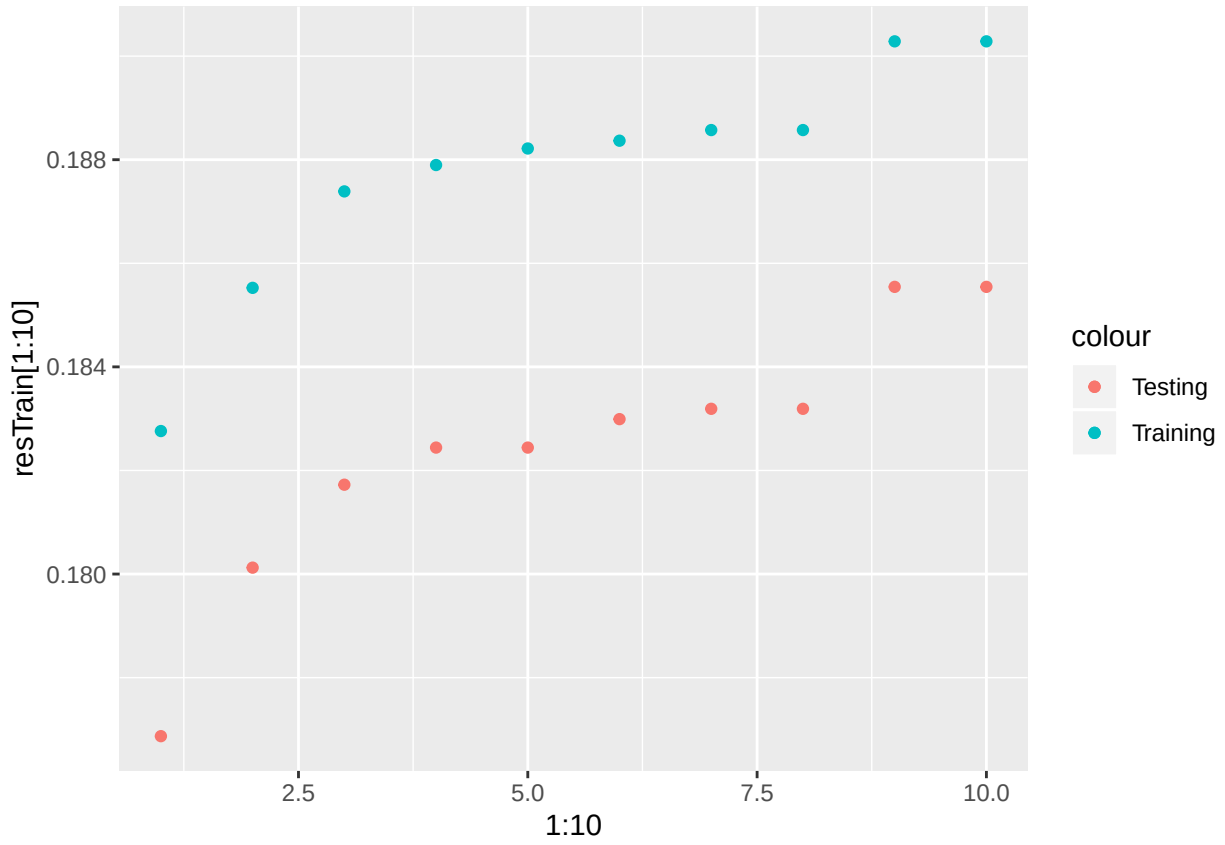
There is no fix formula to set C's value and model should be checked with multiple values in order to find optimal value.

##**Part 3 Neural Network**

```
library(neuralnet)
suppressWarnings(RNGversion("3.5.1"))
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 <- runif(25 , -1 , 1)
resTrain = numeric()
resVali = numeric()
for(i in 1:10)
{
  nn <- neuralnet(formula = Sin ~ Var , data = tr , hidden = c(3,3),
                  startweights = winit, threshold = i/1000 ,
                  lifesign = "full"
  )
  #prediction for training set and there squared error
  pred = compute(nn , as.data.frame(tr$Var))
  pred = pred$net.result
  resTrain[i] = sum((tr[,2] - pred)^2)/2

  pred = compute(nn , as.data.frame(va$Var))
  pred = pred$net.result
  resVali[i] = sum((va[,2] - pred)^2)/2
}

ggplot()+
  geom_point(aes(x = 1:10 , y = resTrain[1:10] , color = "Training"))+
  geom_point(aes(x = 1:10 , y = resVali[1:10] , color = "Testing"))
```

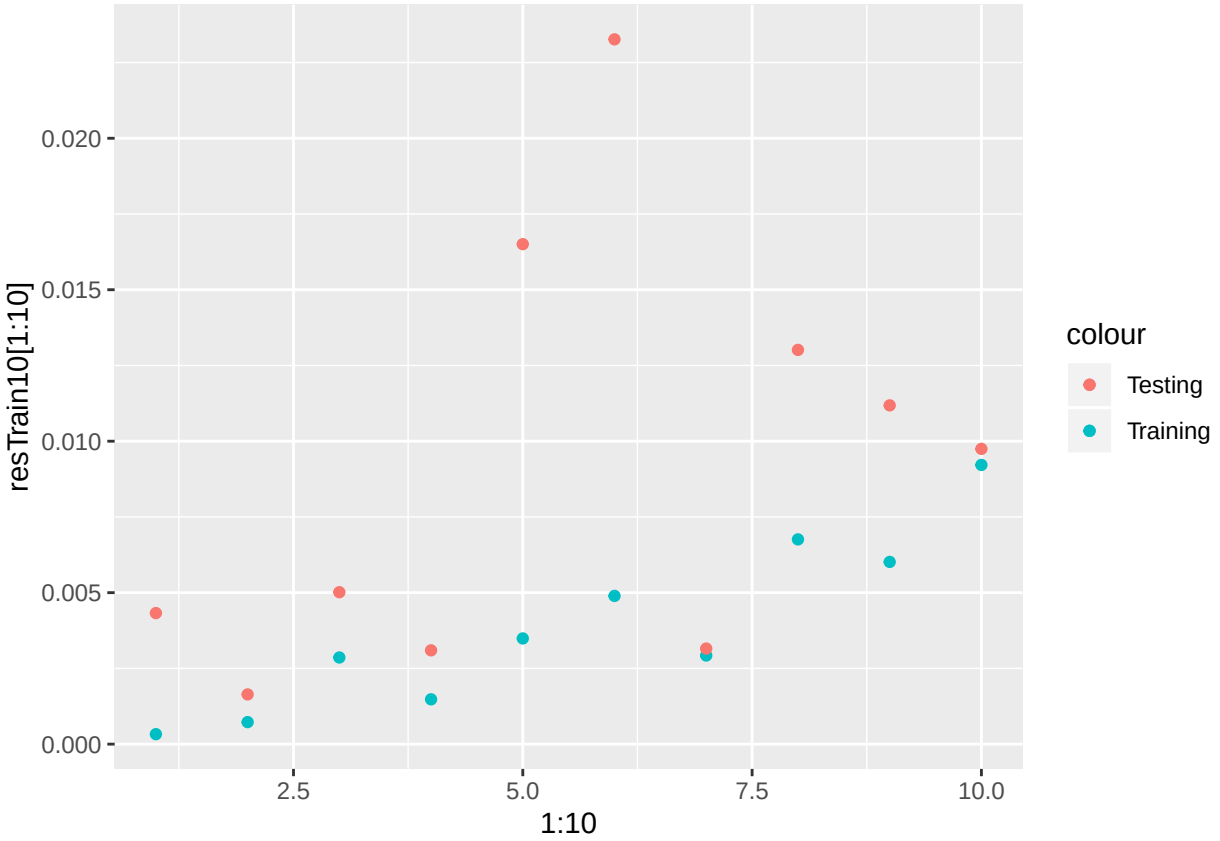



```
#plot(nn)

#with one hidden layer
winit <- runif(25 , -1 , 1)
resTrain10 = numeric(10)
resVali10 = numeric(10)
for(i in 1:10)
{
  nn <- neuralnet(formula = Sin ~ Var , data = tr , hidden = 10,
                  startweights = winit, threshold = i/1000 ,
                  lifesign = "full"
  )
  #prediction for data set and there squared error
  pred = compute(nn , as.data.frame(tr$Var))
  pred = pred$net.result
  resTrain10[i] = sum((tr[,2] - pred)^2)/2

  pred = compute(nn , as.data.frame(va$Var))
  pred = pred$net.result
  resVali10[i] = sum((va[,2] - pred)^2)/2
}

ggplot()+
  geom_point(aes(x = 1:10 , y = resTrain10[1:10] , color = "Training"))+
  geom_point(aes(x = 1:10 , y = resVali10[1:10] , color = "Testing"))
```




```
plot(nn)

winit1 = runif(50 , -1 , 1)
nnFinal = neuralnet(formula = Sin ~ Var , data = trva, hidden = 10 ,
                    startweights = winit1, threshold = 4/1000 ,
                    lifesign = "full")

predFinal = compute(nnFinal , as.data.frame(trva[,1]))
predFinal = predFinal$net.result
genError = sum((trva[,2] - predFinal)^2)/2

plot(nnFinal)

ggplot()+
  geom_point(aes(x = 1:50 , y = trva[,2] , color = " Original"))+
  geom_point(aes(x = 1:50 , y = predFinal , color = "Predicted"))
# geom_line(aes(x = 1:50 , y = trva[,2] , color = " OriginalLine"))+
# geom_line(aes(x = 1:50 , y = predFinal , color = " PredLine"))
```

Appendix

```
knitr::opts_chunk$set(echo = TRUE)

library(geosphere)
stations <- read.csv("G:/MS Machine Learning/Term/Term2/ML/ML Assignment/3/stations.csv")
```

```r
temps <- read.csv("G:/MS Machine Learning/Term/Term2/ML/ML Assignment/3/temps50k.csv")
st <- merge(stations,temps,by="station_number")


# These three values are up to the students

#latitude
a = 58.4274
#Logitude
b = 14.826

#Test Date
date = "2015-01-01"

#time
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","2

h_distance = 200000 #200Km
h_date = 30 #30 Days
h_time = 3 # 3Hours band

#Selecting data which is prior to predicted date
#Day
dateData = as.Date(st$date , format = "%Y-%m-%d")
userDate = as.Date(date , format = "%Y-%m-%d")

filteredDateSt = st[-which(dateData >= userDate),]
#filteredDateSt$date = as.Date(filteredDateSt$date , format = "%Y-%m-%d")

distDiff = distHaversine(filteredDateSt[ , 4:5] , c(a,b))

# h_dtest = sd(distDiff)
# distKernel = exp(-(distDiff/h_dtest)**2)
# hist(distKernel)

distKernel = exp(-(distDiff/h_distance)**2)

#hist(distKernel)
# dateDiff = as.numeric(difftime(userDate , filteredDateSt[,9] , units = "days"))/365
#
# dayKernel = exp(-(dateDiff/h_date)**2)

trDate = numeric()
dateDiff = numeric()

dateDS = as.Date(filteredDateSt$date)

for(i in 1:length(filteredDateSt$date)){
  trDate[i] = difftime(userDate , dateDS[i] , units = "day")
  dateDiff[i] = min((365 - trDate[i]%%365) , trDate[i]%%365)
}

dayKernel = exp(-(dateDiff/h_date)**2)
```

```r
#hist(dayKernel)

#time Kernel
#userTime = as.difftime(times,           #format = "%H:%M:%S" ,   units = "hours")
userTime = as.numeric(as.difftime(times, units = "hours"))
#userTime = times
timeDiff   = matrix(NA, nrow = length(filteredDateSt$time) , ncol = length(userTime))
timeKernel = matrix(NA, nrow = length(filteredDateSt$time) , ncol = length(userTime))

dataTime = as.numeric(as.difftime(as.character(filteredDateSt$time) ,format = "%H:%M:%S", units = "hours

for (i in 1:length(dataTime))
{
  for(j in 1:length(userTime))
  {
    timeD = abs(userTime[j] - dataTime[i])
    timeDiff[i,j] = as.numeric(min(timeD , (24 - timeD)))
  }
}

for(i in 1:ncol(timeDiff))
  timeKernel[,i] = exp(-((as.numeric(timeDiff[,i]))/h_time))

#hist(timeKernel)


#Prediction
predictedTemp = numeric(length = length(times))

kernalSum = numeric()



for (i in 1:length(userTime))
{
  kernalSum = distKernel + dayKernel + timeKernel[,i]
  predictedTemp[i] = sum(kernalSum * filteredDateSt$air_temperature)/sum(kernalSum)

}

#plot(predictedTemp)


#Product Part
KernalProd = numeric()
predictedTemp2 = numeric(length = length(times))
for (i in 1:length(userTime))
{
  KernalProd = distKernel * dayKernel * timeKernel[,i]
  predictedTemp2[i] = sum(KernalProd * filteredDateSt$air_temperature)/sum(KernalProd)
}

library(ggplot2)
```

```
ggplot()+
  geom_point(aes(x = 1:11 , y = predictedTemp)) +
  theme_grey()+
  xlab(" Time ")+
  ylab("Temeperature")+
  ggtitle("Predicted Temperature for given Location and Day")

ggplot()+
  geom_histogram(aes(x = distKernel))+
  xlab("Distance Kernel")+
  ggtitle("Histogram of Distance Kernel")+
  theme_minimal()

ggplot()+
  geom_histogram(aes(x = dayKernel))+
  xlab("Date Kernel")+
  ggtitle("Histogram of Date Kernel")+
  theme_minimal()


hist(timeKernel , xlab = "Time Kernel" , main = "Histogram of Time Kernel")


ggplot()+
  geom_point(aes(x = 1:11 , y = predictedTemp2)) +
  theme_grey()+
  xlab(" Time ")+
  ylab("Temeperature")+
  ggtitle("Predicted Temperature for given Location and Day as Product of Kernel")+
  ylim(-2.6 , -1.4)


library(kernlab)
data("spam")
suppressWarnings(RNGversion("3.5.1"))
set.seed(123456789)
data = spam

#data$type = as.factor(data$type)
n = length(data$type)

id = sample(1:n , floor(n*.50))
train = data[id,]

#Validate
id1 = setdiff(1:n,id)
set.seed(123456789)
id2 = sample(id1 , floor(n*.25))
valid = data[id2 , ]

#Test
id3 = setdiff(id1 , id2)
test = data[id3 , ]
```

```
#At C = 0.5
model_C0.5 = ksvm(x = as.matrix(train[,-58]) , y = train[,58] ,
                  kernel = "rbfdot" , kpar=list(sigma = .05), C = 0.5 )

pred0.5 = predict(model_C0.5 , newdata = valid[,-58])

con0.5 = table(pred0.5, valid[,58])

misCal0.5 = 1 - (sum(diag(con0.5))/sum(con0.5))

#At C = 1
model_C1 = ksvm(x = as.matrix(train[,-58]) , y = train[,58] ,
                kernel = "rbfdot" , kpar=list(sigma = .05), C = 1 )

pred1 = predict(model_C1 , newdata = valid[,-58])

con1 = table(pred1, valid[,58])

misCal1 = 1 - (sum(diag(con1))/sum(con1))


#At C = 5
model_C5 = ksvm(x = as.matrix(train[,-58]) , y = train[,58] ,
                kernel = "rbfdot" , kpar=list(sigma = .05), C = 5 )


pred5 = predict(model_C5 , newdata = valid[,-58])

con5 = table(pred5, valid[,58])

misCal5 = 1 - (sum(diag(con5))/sum(con5))

ComparessionTable = data.frame("C" = c("0.5" , "1" , "5") ,
                               "Miss-calculation Rate" = c(misCal0.5 ,
                                                           misCal1 , misCal5))


ComparessionTable

genData = rbind(train , valid)

selectedModel = ksvm(x = as.matrix(genData[,-58]) , y = genData[,58] ,
                     kernel = "rbfdot" , kpar=list(sigma = .05), C = 5 )


pred = predict(selectedModel , newdata = test[,-58])

con = table(pred, test[,58])

misCal = 1 - (sum(diag(con))/sum(con))

print("Generalization Error calculated in terms of Miss-classification Rate")
```

```
misCal

userModel = ksvm(x = as.matrix(spam[,-58]) , y = spam[,58] , kernel = "rbfdot" , kpar=list(sigma = .05)

print("User model details")

userModel


library(neuralnet)
suppressWarnings(RNGversion("3.5.1"))
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 <- runif(25 , -1 , 1)
resTrain = numeric()
resVali = numeric()
for(i in 1:10)
{
  nn <- neuralnet(formula = Sin ~ Var , data = tr , hidden = c(3,3),
                  startweights = winit, threshold = i/1000 ,
                  lifesign = "full"
  )
  #prediction for training set and there squared error
  pred = compute(nn , as.data.frame(tr$Var))
  pred = pred$net.result
  resTrain[i] = sum((tr[,2] - pred)^2)/2

  pred = compute(nn , as.data.frame(va$Var))
  pred = pred$net.result
  resVali[i] = sum((va[,2] - pred)^2)/2
}

ggplot()+
  geom_point(aes(x = 1:10 , y = resTrain[1:10] , color = "Training"))+
  geom_point(aes(x = 1:10 , y = resVali[1:10] , color = "Testing"))

#plot(nn)

#with one hidden layer
winit <- runif(25 , -1 , 1)
resTrain10 = numeric(10)
resVali10 = numeric(10)
for(i in 1:10)
{
  nn <- neuralnet(formula = Sin ~ Var , data = tr , hidden = 10,
                  startweights = winit, threshold = i/1000 ,
                  lifesign = "full"
  )
  #prediction for data set and there squared error
```

```
  pred = compute(nn , as.data.frame(tr$Var))
  pred = pred$net.result
  resTrain10[i] = sum((tr[,2] - pred)^2)/2

  pred = compute(nn , as.data.frame(va$Var))
  pred = pred$net.result
  resVali10[i] = sum((va[,2] - pred)^2)/2
}

ggplot()+
  geom_point(aes(x = 1:10 , y = resTrain10[1:10] , color = "Training"))+
  geom_point(aes(x = 1:10 , y = resVali10[1:10] , color = "Testing"))

plot(nn)

winit1 = runif(50 , -1 , 1)
nnFinal = neuralnet(formula = Sin ~ Var , data = trva, hidden = 10 ,
                    startweights = winit1, threshold = 4/1000 ,
                    lifesign = "full")

predFinal = compute(nnFinal , as.data.frame(trva[,1]))
predFinal = predFinal$net.result
genError = sum((trva[,2] - predFinal)^2)/2

plot(nnFinal)

ggplot()+
  geom_point(aes(x = 1:50 , y = trva[,2] , color = " Original"))+
  geom_point(aes(x = 1:50 , y = predFinal , color = "Predicted"))
# geom_line(aes(x = 1:50 , y = trva[,2] , color = " OriginalLine"))+
# geom_line(aes(x = 1:50 , y = predFinal , color = " PredLine"))
```