Group A3 Lab 2

varsi146,aswma317,akssr921

2022-12-19

Statement of Contribution

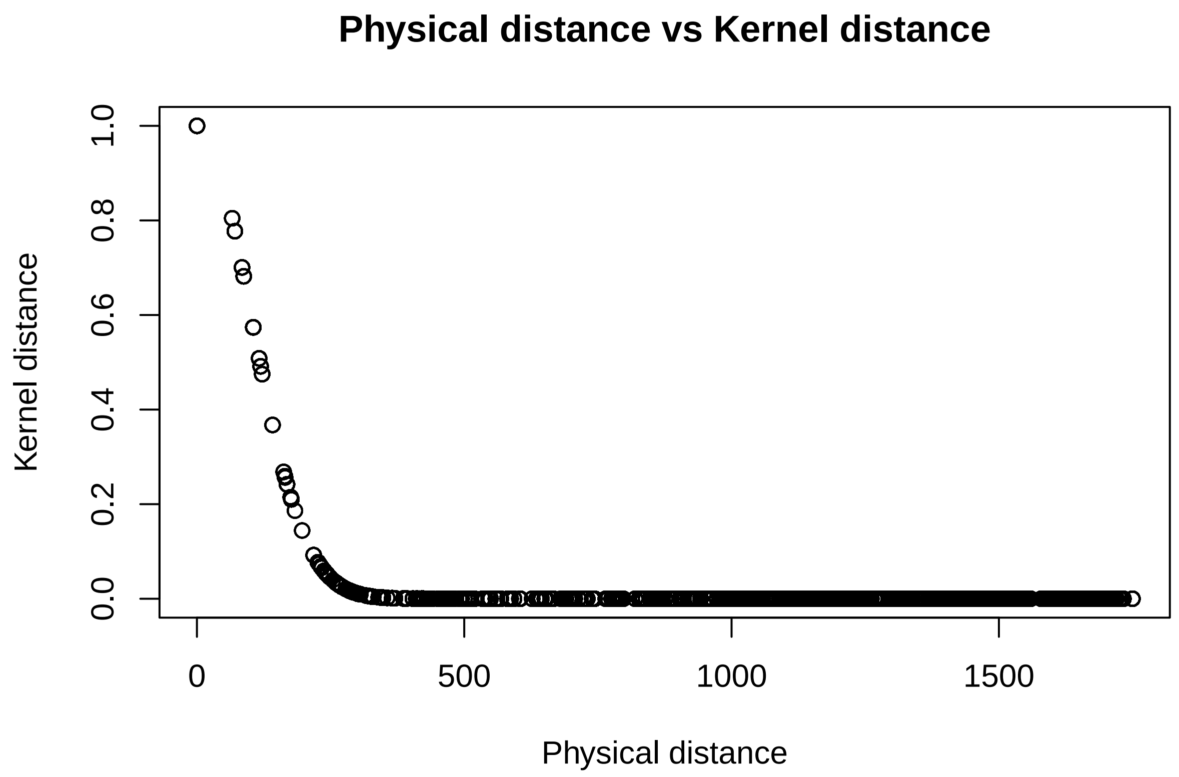
For Lab3, we decided to split the three assignments equally, Akshath completed Assignment 1, Aswath completed Assignment 2 and Varun completed Assignment 3, after which, for verification sake, we completed each other's assignments as well and validated our findings. The report was also compiled by three of us, with each handling their respective assignments.

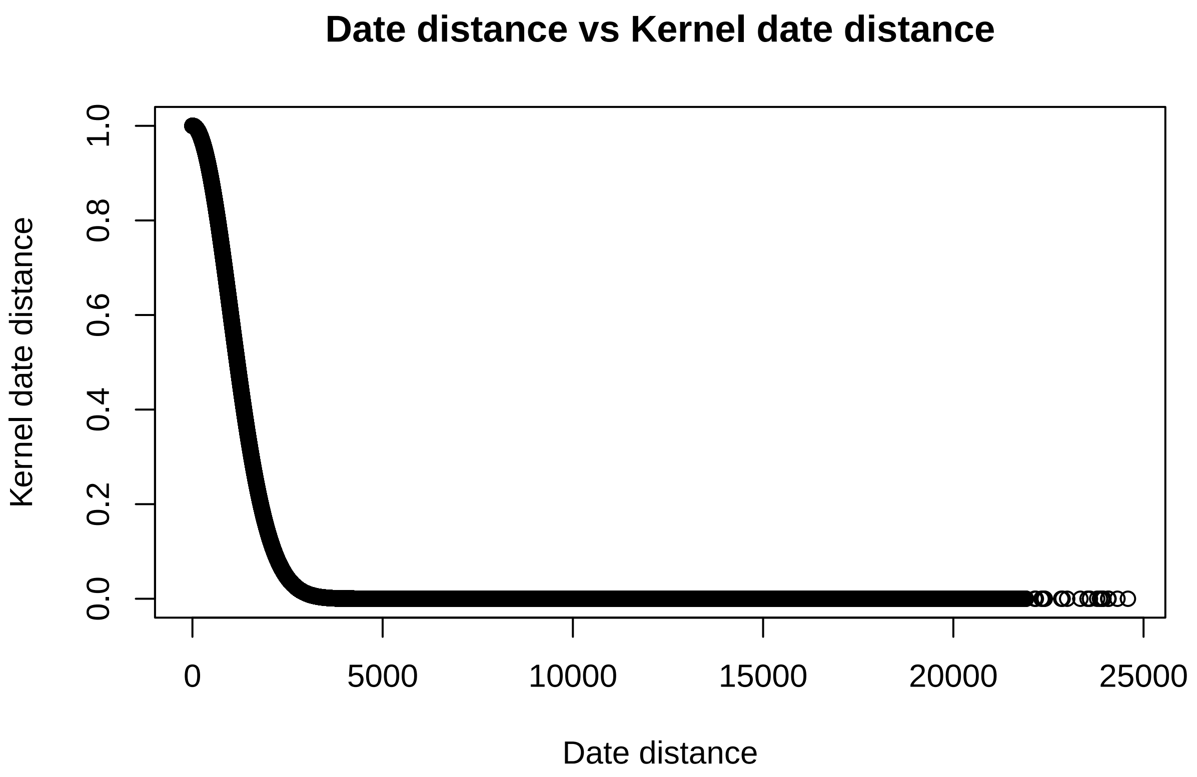
Assignment 1: Kernel methods

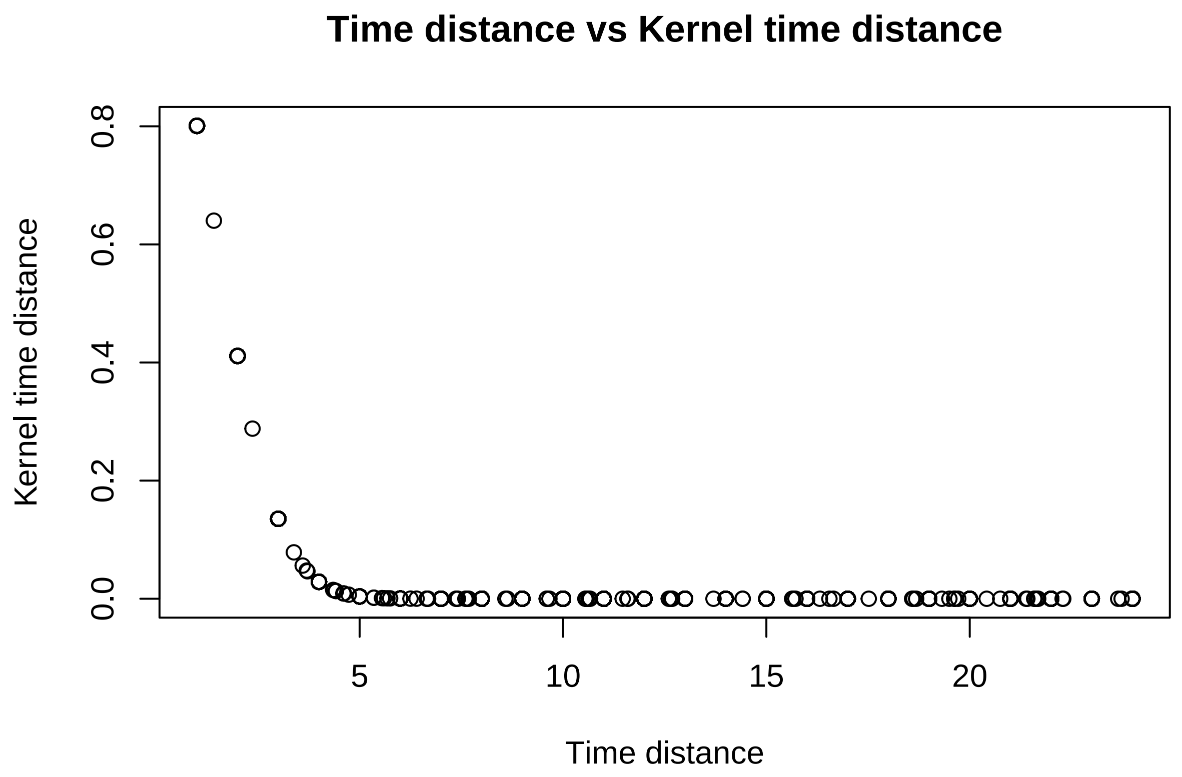
The width values, points, date and times used to predict can be seen below.

```r
h_distance <- 100# These three values are width
h_date <- 1000
h_time <-1.5
a <- 68.4432 # The point to predict
b <- 22.4488
date <- "2008-12-26" # The date to predict
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 to predict
```

The width or smoothing coefficients values are chosen by the below plots which is plotted for kernel value as a function of distance. We selected width values that gives large kernel values to closer points and small values to distant points.


Physical distance vs Kernel distance
Kernel distance
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
Physical distance


Date distance vs Kernel date distance
Kernel date distance
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
Date distance



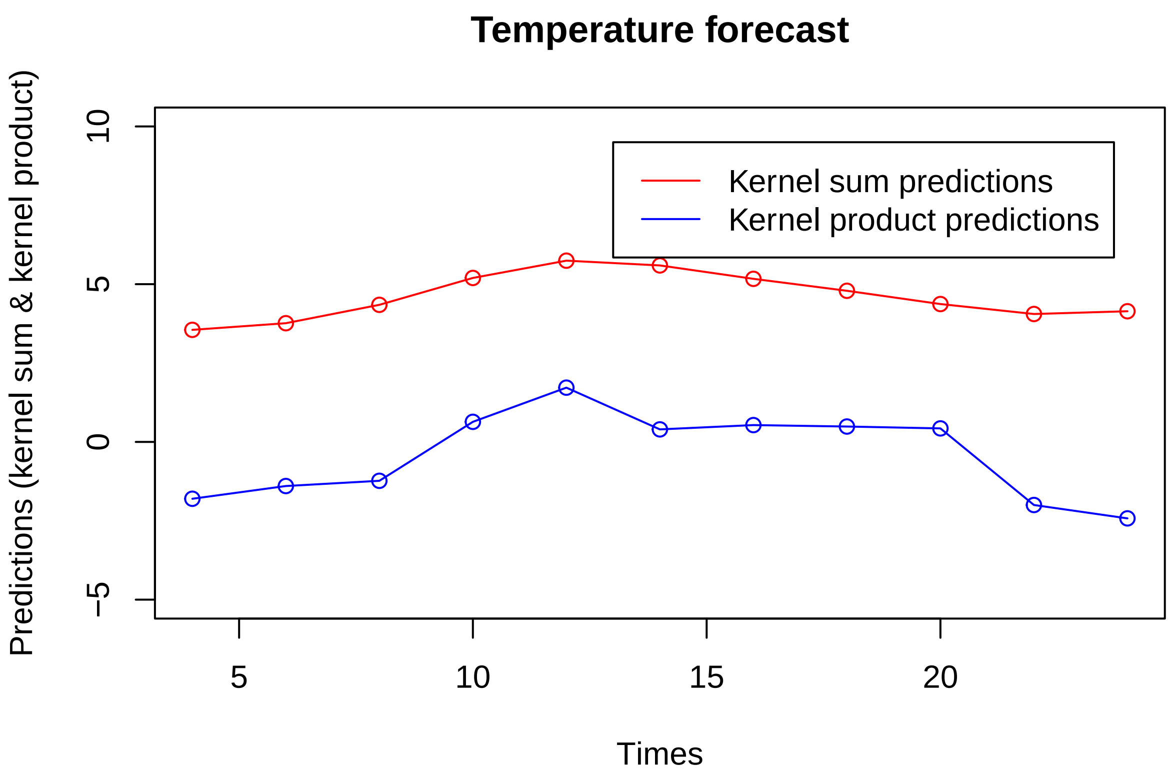
The forecast from **adding kernel** can be seen below.

```
##       times Kernel_sum_predictions
## 1  04:00:00               3.552307
## 2  06:00:00               3.763054
## 3  08:00:00               4.345491
## 4  10:00:00               5.197838
## 5  12:00:00               5.745560
## 6  14:00:00               5.594348
## 7  16:00:00               5.169861
## 8  18:00:00               4.790987
## 9  20:00:00               4.370464
## 10 22:00:00               4.055027
## 11 24:00:00               4.142020
```

The forecast from **multiplying kernel** can be seen below.

```
##       times Kernel_multiplying_predictions
## 1  04:00:00                     -1.8020054
## 2  06:00:00                     -1.3984786
## 3  08:00:00                     -1.2330504
## 4  10:00:00                      0.6363831
## 5  12:00:00                      1.7200928
## 6  14:00:00                      0.3974921
## 7  16:00:00                      0.5335756
## 8  18:00:00                      0.4879336
## 9  20:00:00                    0.4284116
## 10 22:00:00                   -2.0000152
## 11 24:00:00                   -2.4237172
```

From the predictions above and from the plot below, we can conclude that multiplying kernel and then predicting gives more appropriate temperature predictions. We are predicting for the date $2008 - 12 - 26$ which is during Christmas time, the temperatures forecasted after multiplying kernel looks reasonable. Multiplying kernels gives high values only if each of the individual kernel have high values, but in sum we will get high values if any one of the kernel has high value. The better results from summing or multiplying kernels is problem specific. In our case, multiplying kernel works better because each kernel depend only on a single input dimension results in a prior over functions that vary across both dimensions.



Assignment 2: Support Vector Machines

Sub Question 1

Question: Which filter do you return to the user?

```
## Error for filter0 0.0675

## Error for filter1 0.08489388

## Error for filter2 0.082397

## Error for filter3 0.02122347
```

Here, filter0 and filter1 has almost the same configuration; it is getting trained on the training data, and the only difference is the estimation of the E_New - filter0 estimates the E_New from the validation data and filter 1 estimates it from test data.

While filter2 is getting trained on training+validation data & filter3 is getting trained on the entire data.

On the question of which filter needs to be returned, I believe it it should be filter1(or filter0 as it is the same if we want to ignore the E_New estimation part). In filter1, we trained using the training data, found the value of the hyper-parameter(C) by using the validation data, and finally estimate the E_New using the test data.

The problem with filter2 and filter3 is that, we are re-using the C value which was found by training the model on training data and validating via validation data.

If we were to use the filter2, we have to re-calculate the C:

```
## C value calculated with train and validation data(CV): 3.9
```

```
## C value calculated with train+validation and test data(CV): 4.5
```

As seen above, the C values have changed when we train the model on different data set. But once we use the test data(which is supposed to be use to estimate the E_New) to find the hyper-parameter, we don't have a data set to estimate E_New. The issue worsens on filter3 where we use the entire data for training and there is no data left for computing C and estimating E_New

However, if we have to disregard this and provide the filter with the maximum number of support vectors, then filter3 seems like the obvious candidate as the model is getting trained on the entire data.

Sub Question 2

Question: What is the estimate of the generalization error of the filter returned to the user?

For the same reasons mentioned above, err1 should be returned to the user. Here, we trained using the training data, found the value of the hyper-parameter(C) by using the validation data, and finally estimate the E_New using the test data.

Sub Question 3

Question: Make prediction for the first 10 points

```
## Calculated Prediction:  -1.998999 1.560584 1.000278 -1.756815 -2.669577 1.291312 -1.068444 -1.312493
```

```
## Prediction using Predict():  -1.998999 1.560584 1.000278 -1.756815 -2.669577 1.291312 -1.068444 -1.3
```

```
all.equal(actual_pred, pred)
```

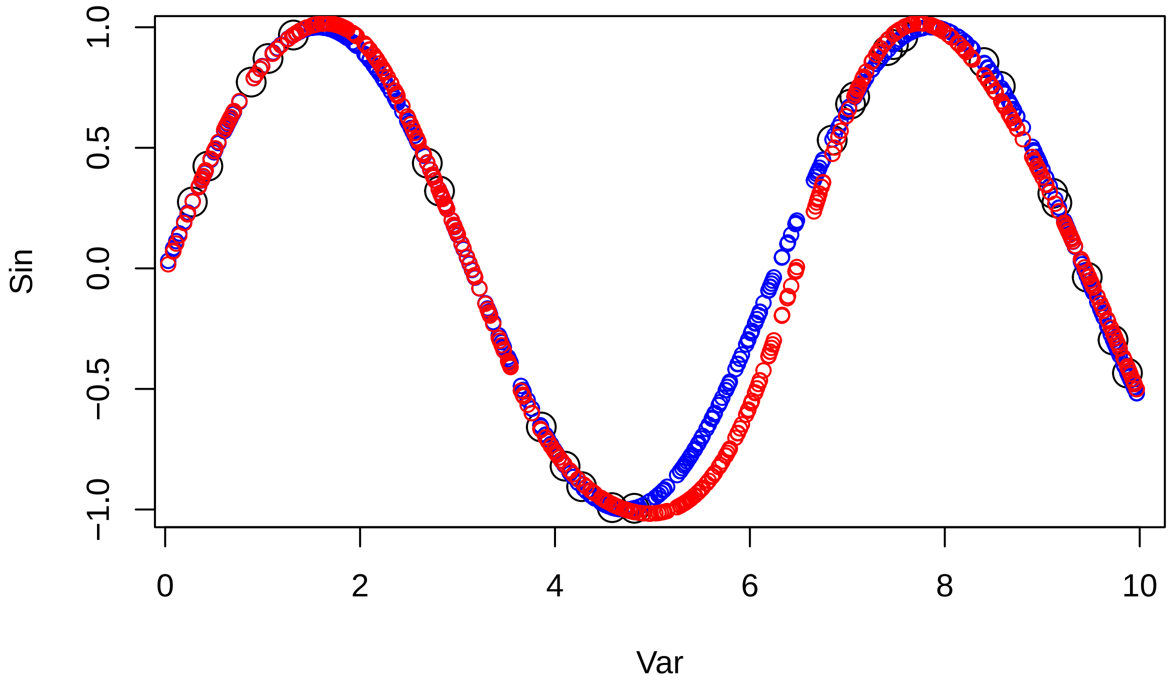
```
## [1] TRUE
```

As seen above, the results match.

Question 3: Neural Networks

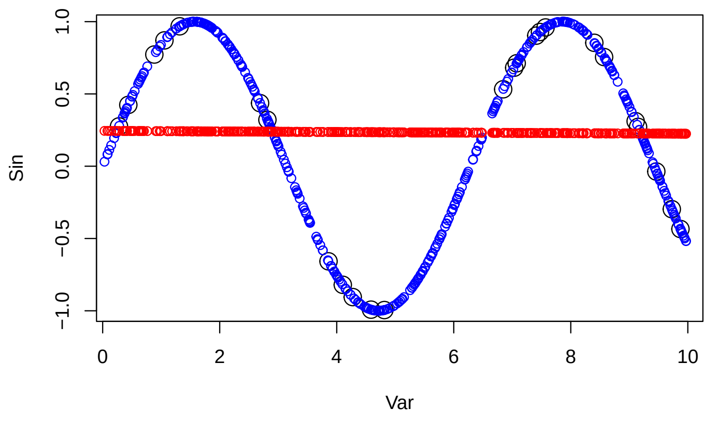
Sub Question 1:

The Neural Network trained is able to predict the trigonometric sine function quite well, however it seems that it is able to capture the peaks of the sine wave better than the trough.

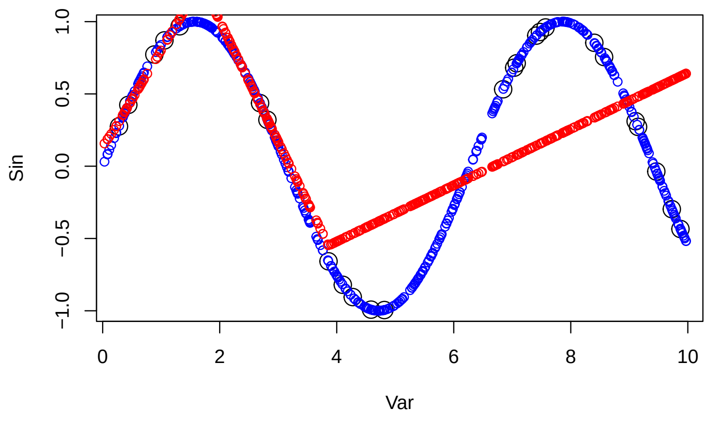


Sub Question 2:

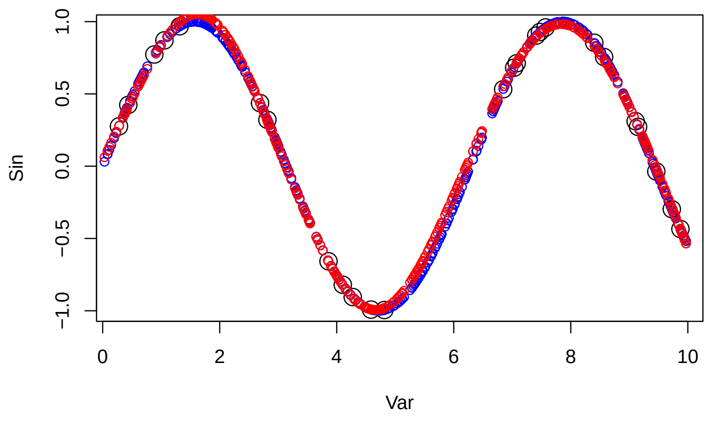
The NN prediction while using a linear activation function:



The NN prediction while using a ReLU activation function:



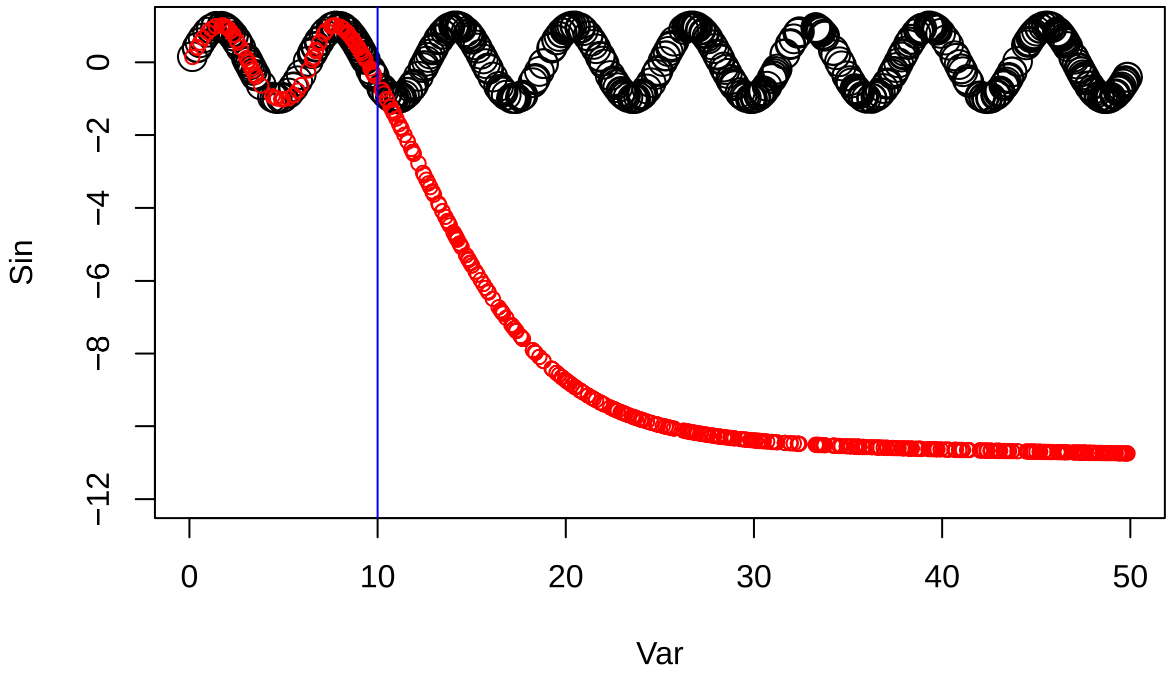
The NN prediction while using a softplus activation function:



From the above plots we can see that it is for the softplus activation function that we get the best prediction from the neural network learning the sine function. The Rectified Linear Output being a piecewise function, cannot capture the sine predictions in its entirety while the linear activation function gives the worst prediction of the three.

Sub Question 3:

From the plot below, we see that the Neural Network is able to perfectly predict values for the variables until 10, after which the predictions seem to move away from the plotted sine-wave seem to converge to a value of around -10.



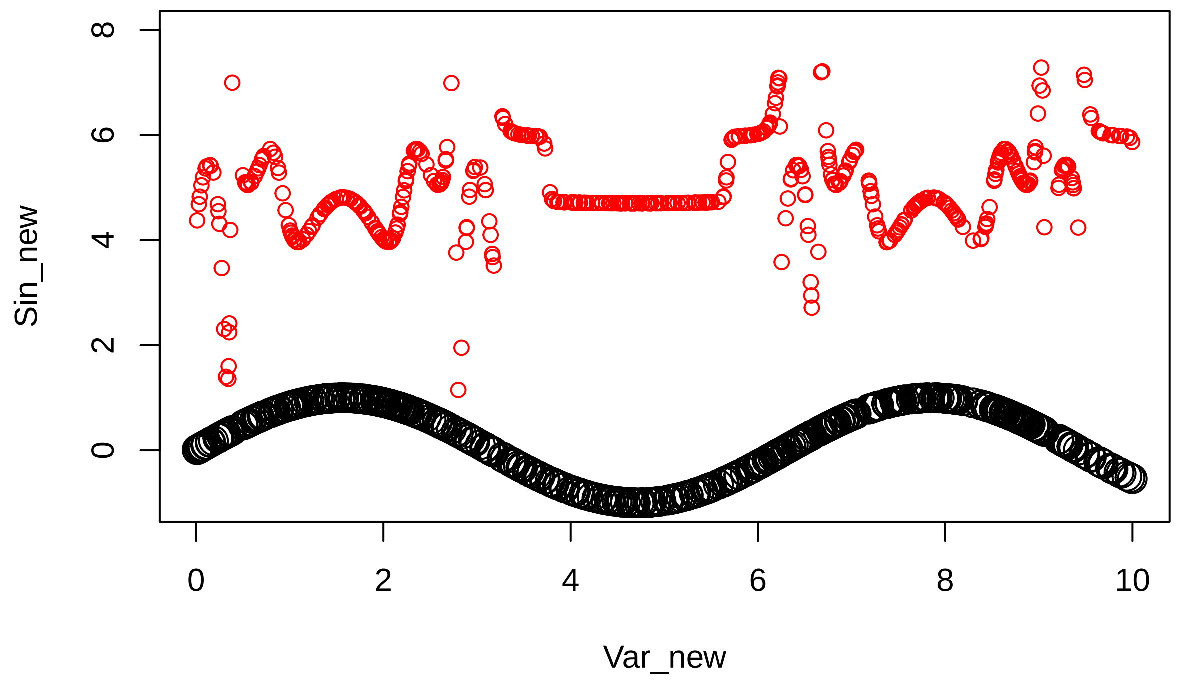
Sub Question 4:

In the previous step, the model is able to predict well upto variables on which the model was trained on. When we look at why the predictions are converging from the perspective of weights, we found out that the for increasing values of x, the already learnt weights seem to have very little to no influence on the predictions.

In other words, the original neural network was trained on data uniformly generated within the interval of $[0, 10]$ and so the weights learned would correspond to that data. But, now we predict using the same model for data generated in interval of $[0, 50]$, and the weights learned would not have much of an influence on these new inputs and this is why after a certain point, the prediction values are converging to a particular value.

Sub Question 5:

As seen from the plot below, the new neural network is trained to predict values of x from $sin(x)$ and results in very poor predictions. A possible explanation as to why the neural network is resulting in poor prediction results is because the model has overfitted to the given set of data without being able to learn the necessary patterns in the data and ends up capturing the noise instead, that is, the neural network is not able predict the non-linear data based on the provided training data.



Appendix

```
library(kernlab)
library(neuralnet)
knitr::opts_chunk$set(echo = TRUE)

#getting data
options(warn=-1)
set.seed(1234567890)
library(geosphere)
library(readr)
stations <- read_csv('stations.csv', locale = locale(encoding = "latin1"),show_col_types = FALSE)
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")
```

```r
h_distance <- 100# These three values are width
h_date <- 1000
h_time <-1.5
a <- 68.4432 # The point to predict
b <- 22.4488
date <- "2008-12-26" # The date to predict
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 to predict


# kernel method function to provide a temperature forecast for a date
# and place in Sweden
kernel_pred<-function(h_distance,h_date,h_time,a,b,date,times){
  pred_temp_sum<-c()
  pred_temp_prod<-c()
  distance_list<-list()
  date_distance_list<-list()
  time_distance_list<-list()
  kernel_distance_list<-list()
  kernel_date_list<-list()
  kernel_time_list<-list()
  j<-0
  for(i in 1:length(times)){
    j<-j+1

    # Filter out temperature measurements made after the forecast time
    st_filtered<-st[(st$date == date & st$time <= times[i]) | st$date<date,]

    # Calculate the distances between the weather stations and the point of interest
    distance <- distHaversine(st_filtered[,c('latitude','longitude')],c(a,b))/1000
    distance_list[[j]]<-distance

    # Calculate the distances in time between the temperature measurements and the forecast date and ti
    date_distance <- abs(as.numeric(difftime(st_filtered$date,date,units='days')))
    date_distance_list[[j]]<-date_distance

    # Convert the character strings to numeric values
    time1 <- strptime(st_filtered$time, format = '%H:%M:%S')
    time2 <- strptime(times[i], format = '%H:%M:%S')

    # Calculate the distance in hours between the two times
    time_distance <- abs(as.numeric(difftime(time1,time2, units = "hours")))
    time_distance_list[[j]]<-time_distance

    #Calculate the kernel values for each temperature measurement using
    #the distances calculated above
    kernel_distance <- exp(-(distance^2)/(2*(h_distance^2)))
    kernel_distance_list[[j]]<-kernel_distance

    kernel_date <- exp(-(date_distance^2)/(2*(h_date^2)))
    kernel_date_list[[j]]<-kernel_date
    kernel_time <- exp(-(time_distance^2)/(2*(h_time^2)))
    kernel_time_list[[j]]<-kernel_time

    # sum of three gaussian kernels
    kernel_sum<-kernel_distance+kernel_date+kernel_time
    #multiplying three gaussian kernels
    kernel_product<-kernel_distance*kernel_date*kernel_time

    #prediction by kernel sum
    prediction_sum<-sum(kernel_sum*st_filtered$air_temperature)/sum(kernel_sum)
    pred_temp_sum<-c(pred_temp_sum,prediction_sum)

    #prediction by kerenel multiplication
    prediction_prod<-sum(kernel_product*st_filtered$air_temperature)/sum(kernel_product)
    pred_temp_prod<-c(pred_temp_prod,prediction_prod)

  }
  return(list(pred_temp_sum,pred_temp_prod,distance_list,kernel_distance_list
              ,date_distance_list,kernel_date_list,time_distance_list,
              kernel_time_list))
}

func_output<-kernel_pred(100,1000,1.5,68.4432,22.4488,"2008-12-26",c("04:00:00", "06:00:00","08:00:00",
                                                                '14:00:00','16:00:00','18:00:00','20:00:00','22
                                                                "24:00:00"))


#plot for selection of width based on distance vs kernel distance
plot(func_output[[3]][[11]],func_output[[4]][[11]],xlab='Physical distance',
     ylab='Kernel distance',main='Physical distance vs Kernel distance')
plot(func_output[[5]][[11]],func_output[[6]][[11]],xlab='Date distance',
     ylab='Kernel date distance',main='Date distance vs Kernel date distance')
plot(func_output[[7]][[11]],func_output[[8]][[11]],xlab='Time distance',
     ylab='Kernel time distance',main='Time distance vs Kernel time distance')


#predictions for sum
Prediction_sum<-func_output[[1]]
sum_df<-data.frame(times=times,Kernel_sum_predictions=Prediction_sum)
sum_df


# predictions for product
prediction_prod<-func_output[[2]]
prod_df<-data.frame(times=times,Kernel_multiplying_predictions=prediction_prod)
prod_df


#plot for prediction by summing kernels
plot(seq(4,24,2),func_output[[1]], type="o",ylim=c(-5,10),col='red',xlab='Times',
     ylab='Predictions (kernel sum & kernel product)',main = "Temperature forecast")
#plot for prediction by multiplying kernel
lines(seq(4,24,2),func_output[[2]], type="o",col='blue')
```

```r
legend(x = 13, y = 9.5,
       legend = c('Kernel sum predictions','Kernel product predictions'),
       col = c('red','blue'), lty = c(1,1))

######### Code for SVM #########
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]#Random selection of data
spam[,-58]<-scale(spam[,-58])#Scaling everything except type
tr <- spam[1:3000, ]#Training data
va <- spam[3001:3800, ]#Validation data
trva <- spam[1:3800, ]#Training/Validation data
te <- spam[3801:4601, ]#Test data

by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){#Finding the least cost
  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))
}
#Use the optimal C to get the error on validation data
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)
# err0#0.0675
cat('Error for filter0', err0)

c1 <- which.min(err_va)*by

#Use the optimal C to get the error on 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#0.08489388
cat('Error for filter1', err1)

#Use the optimal C on train/validation to get the error on test
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#0.082397
cat('Error for filter2', err2)
```

```r
#Use the optimal C on the whole data to get the error on test
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#0.02122347
cat('Error for filter3', err3)
err_va <- NULL
for(i in seq(by,5,by)){#Finding the least cost
  filter <- ksvm(type~.,data=trva,kernel="rbfdot",
                 kpar=list(sigma=0.05),C=i,scaled=FALSE)
  mailtype <- predict(filter,te[,-58])
  t <- table(mailtype,te[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}
c2 <- which.min(err_va)*by
cat('C value calculated with train and validation data(CV):', c1)
cat('C value calculated with train+validation and test data(CV):', c2)
sv<-alphaindex(filter3)[[1]] #Indexes of support vectors
co<-coef(filter3)[[1]] #Linear coefficients of support vector
inte<- - b(filter3) #Negative intercept of linear combination
rbfkernel <- rbfdot(sigma = 0.05) #Declare the kernel
pred <- c()
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  k<-c()
  for(j in 1:length(sv)){
    x <- as.vector(unlist(spam[i,-58]))#Test data to be predicted
    y <- as.vector(unlist(spam[sv[j],-58]))#Support vector
    k <- c(k,rbfkernel(x,y))#Kernel value for each test vs sv
  }
  pred <- c(pred, ((k %*% co) + inte))
}
actual_pred <- as.vector(predict(filter3,spam[1:10,-58], type = "decision"))
cat('Calculated Prediction: ', pred)
cat('Prediction using Predict(): ', actual_pred)
all.equal(actual_pred, pred)
set.seed(1234567890)
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
tr <- mydata[1:25,] # Training
te <- mydata[26:500,] # Test
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)


nn <- neuralnet(formula = Sin~Var, data = tr, hidden = 10, startweights = winit)

plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
# Activation functions for softplus, linear and ReLU:
```

```r
softplus_activ <- function(x) log(1+exp(x))
linear_activ <- function(x) x
relu_activ <- function(x) ifelse(x > 0, x, 0)



nn_func <- function(activ_fn){
  set.seed(42)
  nn_custom <- neuralnet(formula = Sin~Var, data = tr, hidden = c(10), act.fct = activ_fn)
  plot(tr, cex=2)
  points(te, col = "blue", cex=1)
  points(te[,1],predict(nn_custom,te), col='red', cex=1)
}
nn_func(linear_activ)
nn_func(relu_activ)
nn_func(softplus_activ)
set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata_new <- data.frame(Var, Sin=sin(Var))

plot(mydata_new, cex=2, ylim = c(-12,1))
points(mydata_new[,1], predict(nn,mydata_new), col="red", cex=1)
abline(v = 10, col = 'blue')
set.seed(123456789)
Var_new <- runif(500, 0, 10)
mydata_new_2 <- data.frame(Var_new, Sin_new=sin(Var_new))

nn_new <- neuralnet(formula = Var_new~Sin_new, data = mydata_new_2, hidden = 10, threshold = 0.1)

plot(mydata_new_2, cex=2, ylim = c(-1, 8))
points(mydata_new_2[,1], predict(nn_new,mydata_new_2), col="red", cex=1)
```