Machine Learning - Block 1, Lab 3

Group A13: Connor Turner, Arash Haratian, Yi-Hung Chen

2022-12-09

Statement of Contribution: *For this lab, Assignment 1 was completed by Yi-Hung, Assignment 2 was completed by Arash, and Assignment 3 was completed by Connor. Each member then collaborated with each other to produce the final report.*

Assignment 1: Kernel Methods

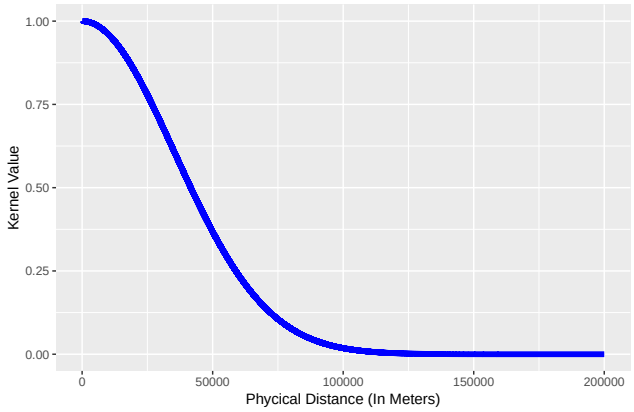
In this assignment, the task was to build a model to predict the hourly temperature for a certain date and place in Sweden. To do this, a kernel method was implemented based on daily temperature and station data from the Swedish Meteorological and Hydrological Institute.

Three different Gaussian kernels were used in this assignment:

1. The physical distance from a station to the point of interest.
2. The distance between the day a temperature measurement was made and the day of interest.
3. The distance between the hour of the day a temperature measurement was made and the hour of interest.

The first task was to choose an appropriate smoothing coefficient for each of three kernels above. No cross-validation was used in this analysis; instead, the widths were manually chosen so that points closer to the target point have larger kernel values, and vice versa. Figures 1-3 below show the kernel value as a function of distance, and the descriptions for each explain why each particular value was chosen.

Figure 1: Kernel Value With Respect to Physical Distance

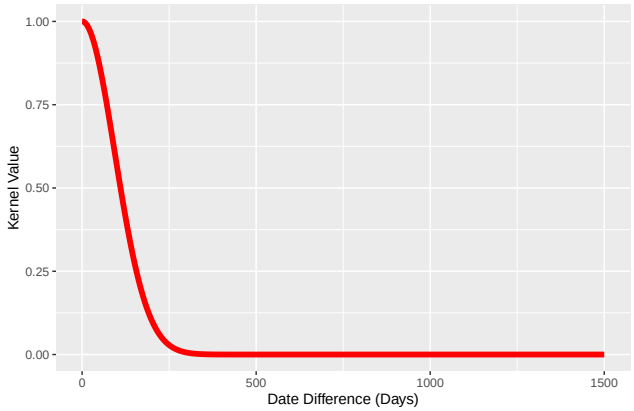


.

Distance Kernel: For the distance kernel, the width was set at 50 kilometers, so that points in the nearby vicinity of the station (< 50 km) received larger kernel values, and points further away received a smaller kernel value. Intuitively, this makes sense, as the weather varies greatly across large distances. Because of this, data coming from a station located more than 100 km away hardly affects the prediction given by the

model, as Figure 1 shows.

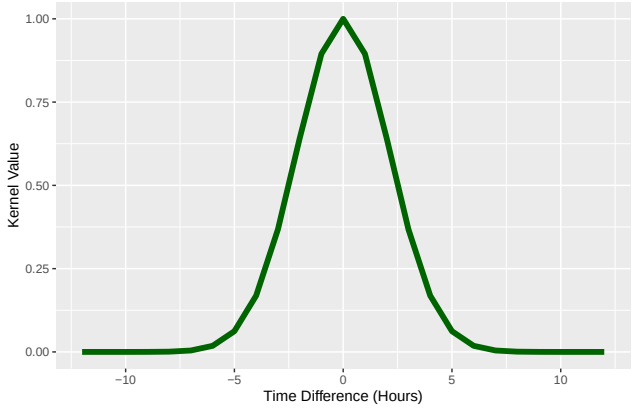
Figure 2: Kernel Value With Respect to Date Difference



.

Date Kernel: The width of the date kernel was set at 182.5 (365/2), meaning that observations within the previous six months were given a larger kernel value, and later observations were given smaller kernel values. This is also intuitive, as the current weather is more likely to follow more recent trends in that area. As Figure 2 shows above, as the date difference approaches 365 days, the kernel value becomes almost nonexistent. However, it should be noted that this is not ideal in real life scenario, as the same date in previous years should have a larger impact on prediction than a date in different season.

Figure 3: Kernel Value With Respect to Hour Difference



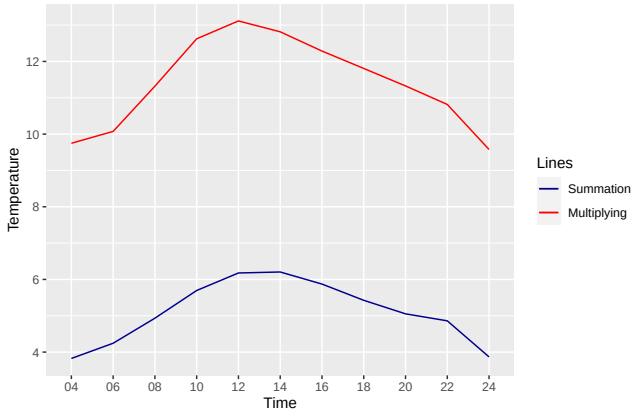
.

Hour Kernel: The width of the hour kernel was set at 3 hours, meaning that observations within the last 3 hours had the largest kernel values and later observations had smaller kernel values. This also makes intuitive sense, as predictions made based on the weather in the previous hour are much more likely to be accurate than ones made on the weather 12 hours ago. As such, data that is ahead or behind by 6 hours or more is given almost no weight in the model, as Figure 3 shows.

After the kernels were built and the widths were set, the next step was to combine the three Gaussian kernels – first by summation, and then by multiplication. These two models were then used to make temperature predictions over a full day based on the given data. These predictions are mapped in Figure 4 below.

Note: because the data includes measurements that are posterior to the day and hour of the target time, filtering must be done before prediction being made.

Figure 4: Comparison of Summation and Multiplication of Kernels



.

According to Figure 4, the predicted temperature was much higher in the multiplication model compared to the summation model. Additionally, the difference between the highest and lowest temperatures was higher in the multiplication model compared to the summation model. In other words, the daily temperature prediction for the summation model was lower and had less variance compared to the other model.

The reason for these differences had to do with the nature of the kernels and how they are combined. In the summation model, if any of three kernels had a large value, the combined weight would also be large because the three kernels are added together. However, in the multiplication model, if any one of three kernels produced a small kernel value, it resulted in a smaller overall weight because the three decimals would be multiplied together.

For example, take a landmark where the nearest station is over 500km away, but the station has data from the same day and previous hour before the target time. The summation approach will given this data a large weight, because the values of the date and hour kernel are large; however, the multiplication approach will result in an overall weight near 0, as the distance is too far away for a reasonable prediction to be made. Because of this, it appears that the multiplication approach will produce more reasonable predictions, as all three kernels would have to be large for the data to have any effect on the prediction value.

Assignment 2: Support Vector Machines

The aim of this assignment was to train and evaluate 4 different support vector machine (SVM) models with different partitions of the data as the training and testing data. All 4 models used Gaussian (or radial basis function) kernels where $\sigma = 0.05$.

First, the data was divided into 4 different partitions:

- *training data (`tr`)*: containing the first 3000 observations
- *validation data (`va`)*: containing observations 3001 to 3800
- *training + validation data (`trva`)*: containing the first 3800 observations
- *test data (`te`)*: containing observations 3801 to 4601

To find the best value of the regularization parameter C, a series of candidate values for C were used to train an SVM model on the training data, and the misclassification rate was calculated for each value of C on the validation data. The best value of C was the candidate value that has the lowest misclassification rate on the validation data – in this case, that value was 3.9.

A total of four different SVM models were trained using the data in the training partition and the given value of σ. The following table shows which partitions were used to train and evaluate each model:

The model (`name`)	training partition	evaluating partition
First model (`filter0`)	training data (`tr`)	validation data (`va`)
Second model (`filter1`)	training data (`tr`)	testing data (`te`)
Third model (`filter2`)	training + validation data (`trva`)	testing data (`te`)
Fourth model (`filter3`)	full data (`spam`)	testing data (`te`)

The misclassification rates for each model on the validation data are as follows:

The model (`name`)	error rate
First model (`err0`)	0.0675
Second model (`err1`)	0.0849
Third model (`err2`)	0.0824
Fourth model (`err3`)	0.0212

The question here is twofold:

1) Which filter would be best for future predictions (i.e. gives the best generalization)?

2) Which prediction of the error rate is most accurate (i.e less biased)?

In regard to the first question, Model 4 (`filter3`) would provide the best predictions. When it comes to Support Vector Machines, the more data that is used to train the model, the better that model is at generalization. Since Model 4 is the only model that used the entire dataset for training, it can reasonably be assumed that it will perform better with new data compared to the other models.

In regard to the second question, Model 3 (`err2`) provides the most unbiased prediction of the misclassification rate. Model 1 (`err0`) was biased and underestimated the real generalization error, as the partition that was used to calculate the error rate was the same partition that was used to find the best value of the regularization parameter C. Model 4 (`err3`) underestimated the real generalization error as well, as the testing data was also used to train the model. When the partition that is used to calculate the error rate is also used to train the model, the error is not an unbiased estimate of real generalization error. Between `err1` and `err2`, `err2` was the most acceptable choice, as the training partition for third model (`filter2`) had more observations compared to second model (`filter1`).

To predict the class of any new data point x_*, the following formula was used:

$$\hat{y}(x_*) = \sum_{i=1}^{n} \hat{\alpha}_i * K(x_i, x_*)$$

where $\hat{\alpha}$ is the linear coefficients for the support vectors, x_i the i^{th} observation of the support vector, and n is equal to the length of the support vector. Furthermore, to predict the class of each observation, one can use the formula $\hat{g}(x_*) = sign(\hat{y}(x_*))$. The predictions for the top ten rows of the full dataset are shown in the table below:

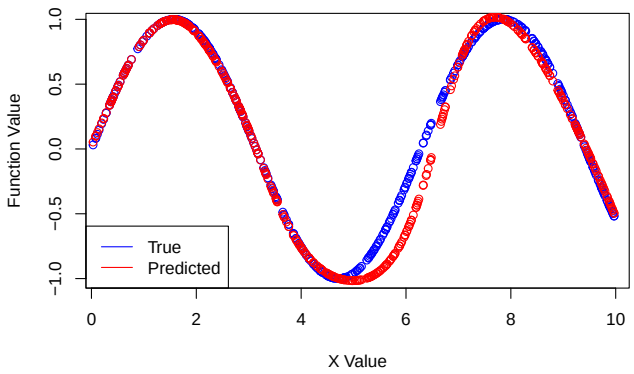
manual_predictions	predict_function
-1.998999	-1.998999
1.560584	1.560584
1.000278	1.000278
-1.756815	-1.756815
-2.669577	-2.669577
1.291312	1.291312
-1.068444	-1.068444
-1.312493	-1.312493
1.000183	1.000183
-2.208639	-2.208639

Assignment 3: Neural Networks

For this assignment, the goal was to train neural networks with different settings using the `neuralnet` package in R.

In the first exercise, a neural network was trained to learn the trigonometric sine function $f(x) = sin(x)$. To do so, 500 data points were randomly generated from the Uniform distribution on the interval $[0, 10]$ and then run through the sine function. Of these data points, 25 were used as training data, and the remaining 475 were used for testing. The training data were used to train a neural network with one hidden layer that contained 10 hidden units. This trained model was then used to predict the sine values in the testing data. The results are plotted in Figure 5 below, where the real values of the testing data are plotted in blue and the predicted values are plotted in red:

Figure 5: True vs Predicted Sine Values from Neural Net



.

As is shown in the plot above, the predicted values follow the true values quite closely, with the only slight deviation coming coming when x was between 5 and 7.

Building off of this original model, the next step was to observe the behavior of neural networks using different activation functions, denoted as $h(x)$. This activation function is used in each hidden unit in the neural network to generate the output value for that unit. These output values are then multiplied by the output weights and combined to create the predicted value, $\hat{y}$. In the `neuralnet` package, the default activation function is the logistic/sigmoid function, but in this case, three other functions were used instead: the linear activation function ($h_1(x) = x$), the recified linear unit (ReLU) activation function ($h_2(x) = max(0, x)$), and the softplus activation function ($h_3(x) = ln(1 + e^x)$). These activation functions were used to repeat the process of the previous exercise, and the results of these activation functions are shown in Figures 6, 7, and 7 below:

Figure 6: True vs Predicted Values − Linear Activation

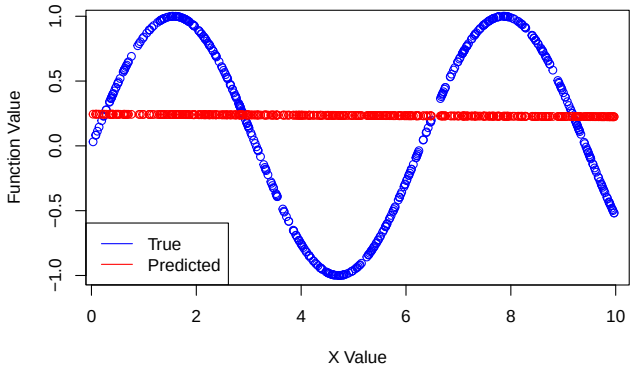


Figure 7: True vs Predicted Values − ReLU Activation

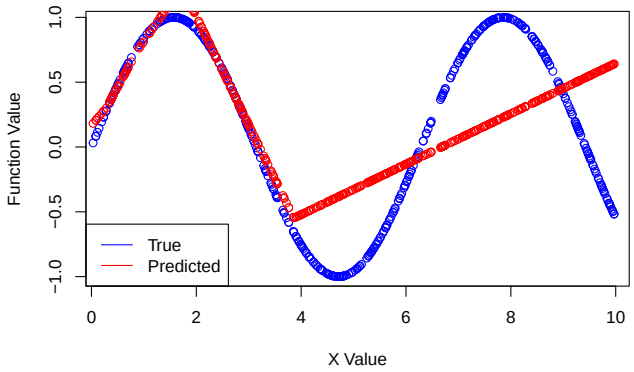
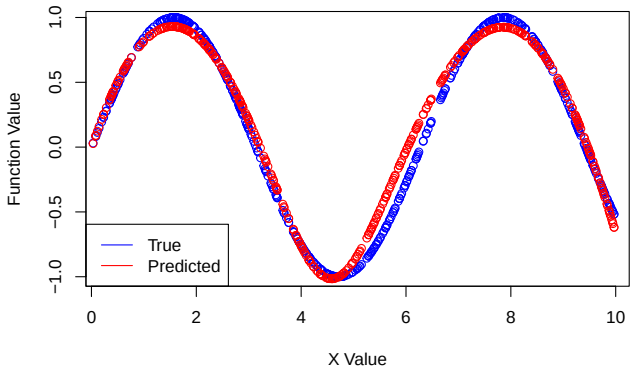


Figure 8: True vs Predicted Values − Softplus Activation



.

Since the relationship being predicted is non-linear, it makes sense that the linear activation function does not provide an accurate prediction for the testing data. Since the derivative of the linear function is a constant, the backpropagation algorithm results in a model that cannot capture more complex relationships. And in the case of this one-input, one-layer neural network, the predictions of this model form a straight line instead of following the sine curve. However, contrary to what my be seen from the plot, the model does not produce the same predicted value for every value of x. Instead, the predicted values show a very slight negative relationship between x and the prediction of $sin(x)$.

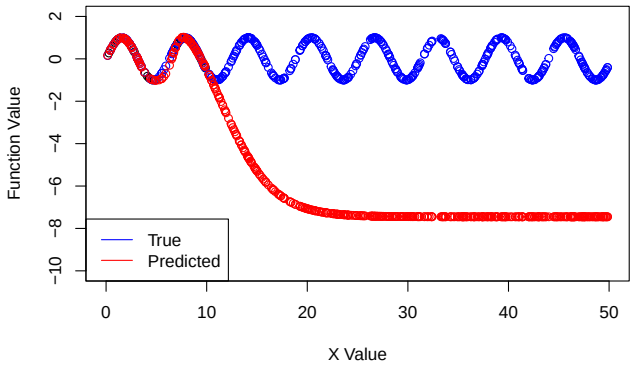
The ReLU activation function is a popular alternative to the logistic activation function in more complex neural networks, as its simpler nature makes it easier to train deep neural networks, and its nonlinearity allows it to capture more complex relationships than a linear activation function. But for this single-layer neural network, it was not very helpful. As seen in Figure 3, the predicted values followed the true values somewhat well through the first curve (over the interval $[0, 4]$). However, once $x > 4$, the sign of each input

value stayed the same across each subsequent value of x, effectively making $h_2(x)$ a linear activation function. As a result, the predictions over the interval $[4, 10]$ became practically useless for predicting the non-linear sine function.

The softplus activation function, on the other hand, provided a fairly accurate prediction of the sine curve. This result may seem surprising at first, because the shape of the softplus curve is extremely similar to the ReLU. In fact, the outputs of both functions are almost identical as $x \to \infty$ and $x \to -\infty$. However, the key difference is in the derivatives of each function. Whereas the ReLU faces the same issue as the linear activation function beyond a certain value of x, the derivative of the softmax is actually the logistic activation function. This function allowed for a more effective backpropagation algorithm in the training process that better captured the relationship between x and $sin(x)$. Thus, the predicted values in Figure 8 are quite similar to the ones seen in Figure 5.

Next, the predictive power of the neural network was tested for values beyond its training domain. In this exercise, the neural network from the first exercise was fitted to another set of testing data consisting of 500 data points randomly generated from the Uniform distribution. The only difference is that in this exercise, the testing data is over the interval $[0, 50]$ instead of the training domain of $[0, 10]$. The true and predicted values are shown in Figure 9 below:

Figure 9: True vs Predicted Sine Values from Neural Net



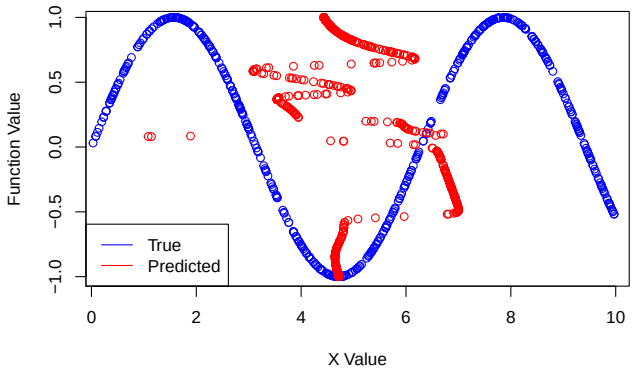
.

As expected, the neural network does a great job of predicting values over the interval $[0, 10]$. However, the predictions veer way off course as soon as x goes beyond that interval. In particular, the predicted values decrease monotonically over the interval $[10, 50]$, eventually converging around -7.45.

To understand why this happens, it is helpful to understand the nature of the logistic activation function and how the weights in the neural network are learned. The weights that are learned by the neural network are trained on the training domain, in this case [0,10]. This means that when the value of x leaves this domain, it will lead to extreme values being inputs in the activation function – in this case the logistic. For this specific activation function, large positive input values produce an output that is very close to 1, and large negative values produce an output very close to 0. Thus, if the input weights are positive (regardless of the bias), the output of the activation function will go to 1 for that hidden unit as $x \to \infty$. For negative input weights, the output of the activation function will go to 0 for that hidden unit as $x \to -\infty$. As a result, the output converges on the sum of the output weights for the hidden units where the input weights are positive. In the case of this particular neural network, that value is -7.4511.

For the final exercise, another neural network was trained to do the exact opposite of the first neural network. In this case, 500 data points were sampled in the exact same way as the first exercise. However, this time the network was trained to predict the value of x from $sin(x)$. All 500 points were used as the training data for this exercise, and the resulting neural network was then used to predict the x values in the training data given the sine values. The results are shown in Figure 10 below:

Figure 10: True vs Predicted X Values from Neural Net



.

Since the model is being used to predict the data that it was trained on, one would expect an almost perfect recreation of the data. But as Figure 10 plainly shows, this was not the case for this model. The reason for this has mainly to do with what is being used as the input and output for this model. Training the model to predict $sin(x)$ from x is quite simple, as each value of x in the training data is associated with only one output value. In fact, when this exercise was recreated to predict $sin(x)$ from x like the other models, the prediction values were almost a perfect match with the true values. However, predicting x from $sin(x)$ is incredibly confusing for the model. In the latter case, the range of input values is quite small [-1,1], and since $sin(x)$ is a periodic function, each of these inputs are associated with two or more output values. For example, if $sin(x) = 0$, x could potentially be 0, π, 2π, or 3π. As a result, the model cannot properly train itself, because it does not know what output value it is supposed to assign for a given value of $sin(x)$ based on the training data. This results in a plot like Figure 10, where the predicted values are wildly inaccurate and do not show any kind of pattern other than the fact that they are loosely clustered around the middle of the training domain of x.

Appendix 1: Code for Assignment 1

```
#Question 1
library(geosphere)
library(ggplot2) #TA said ggplot is allowed for visualization

set.seed(1234567890)
stations <- read.csv("./stations.csv", fileEncoding = "latin1")
temps50k <- read.csv("./temps50k.csv")

data_full <- merge(stations, temps50k, by = "station_number")

#choose target location
set.seed(1378)
target_station <- 84260
target_loc <- stations[stations$station_number == target_station, c("longitude", "latitude")]
target_data <- data_full[data_full == target_station, ]
target_date <- "2013-11-04"

hours <- seq(as.POSIXct("2013-11-04 04:00"), as.POSIXct("2013-11-04 24:00"), by = "2 hour")
hours <- format(hours, format = "%H:%M:%S")
#find the max physical difference between the target station and the other station
max_distance <- max(distHaversine(target_loc, data_full[,  c("longitude", "latitude")]))

#gives the maximum date difference to "previous date"
max_date_diff <- max(as.vector(difftime(as.Date(target_date),as.Date(data_full$date), units = "day")))

# Check if the weight is suitable for our application
h_distance <- 50000 ; h_date <- 132.5; h_time <- 3
distance <- seq(0,200000,1)/h_distance
k_dist <- exp(-(distance)^2)
plotdf_distance <- data.frame(k_dist,seq(0,200000,1))

plot_distance <- ggplot(plotdf_distance,aes(x=seq(0,200000,1),y=k_dist))+
  geom_line()+
  ggtitle("Figure 1: Kernal value with respect to physical distance")+
  xlab ("Phycical Distance in meters")+
  ylab ("kernel values")

# Since many of the stations are simply too far away, limit the x-value to make observation easier


# kernel of date
max_date_diff <- max(as.vector(difftime(as.Date(target_date),as.Date(data_full$date), units = "day")))

#Here, calculate the max/min difference of date with target date, this doesn't account for season since
date <- seq(0,1500,1)/h_date
k_date <- exp(-(date)^2)
plotdf_date <- data.frame(k_date,seq(0,1500,1))

plot_date <- ggplot(plotdf_date,aes(x=seq(0,1500,1),y=k_date))+
  geom_line()+
  ggtitle("Figure 2:Kernal value with respect to date distance")+
  xlab ("Date Difference")+
```

```
  ylab ("kernel values")


# we only present t closest 2000 day, since the day that is before that has too little affect to the va


# kernel of time
time <- seq(-12,12,1)/h_time
k_time <- exp(-(time)^2)
plotdf_time <- data.frame(k_time,seq(-12,12,1))

plot_hour <-ggplot(plotdf_time,aes(x=seq(-12,12,1),y=k_time))+
  geom_line()+
  ggtitle("Figure 3:Kernal value with respect to hour distance")+
  xlab ("Hour Difference")+
  ylab ("kernel values")


#Summation Kernel
i <- 1
temperature_sum <- vector(length = length(hours))
for(i in seq_along(hours)){
  #filter posterior data
  available_data <- data_full[(

      data_full$date < target_date|
      data_full$time <= hours[i] & data_full$date == target_date


  ), ]

  physcial_distance <- distHaversine(target_loc, available_data[,  c("longitude", "latitude")])
  date_distance <- difftime(as.Date(target_date), as.Date(available_data$date), units = "day")
  hour_distance <- difftime(as.POSIXct(hours[i], format = "%H:%M:%S"),
                    as.POSIXct(available_data$time, format = "%H:%M:%S"),
                    units = "hour")
  #Here we referred to page10 of "Lecture 3a Block 1:  Kernel Methods" Lecture Slide
  weight_sum <- exp(-(physcial_distance^2/h_distance^2/2)) + exp(-(as.numeric(date_distance)/h_date)^2/
  temperature_sum[i] <- sum(available_data$air_temperature * weight_sum) / sum(weight_sum)
}

#Multiplying Kernel

temperature_multiply <- vector(length = length(hours))
for(i in seq_along(hours)){
   available_data <- data_full[(

      data_full$date < target_date|
      data_full$time <= hours[i] & data_full$date == target_date


  ), ]
```

```
  physcial_distance <- distHaversine(target_loc, available_data[,  c("longitude", "latitude")])



  date_distance <- difftime(as.Date(target_date), as.Date(available_data$date), units = "day")
  hour_distance <- difftime(as.POSIXct(hours[i], format = "%H:%M:%S"),
                    as.POSIXct(available_data$time, format = "%H:%M:%S"),
                    units = "hour")

  #Here we referred to page10 of "Lecture 3a Block 1:  Kernel Methods" Lecture Slide
  weight_multiply <- exp(-(physcial_distance^2/h_distance^2/2)) * exp(-(as.numeric(date_distance)/h_dat


  temperature_multiply[i] <- sum(available_data$air_temperature * weight_multiply) / sum(weight_multiply
}
plot_hours <- c("04","06","08", "10", "12", "14", "16", "18", "20", "22" ,"24")
temperature_sum <- data.frame(cbind(plot_hours,temperature_sum))

plotdata <- data.frame(cbind(temperature_sum,temperature_multiply))
ggplot(data =plotdata)+
  geom_line(aes(x=plot_hours,y=as.numeric(temperature_sum),group=1,colour="Summation"))+
  geom_line(aes(x=plot_hours,y=as.numeric(temperature_multiply),group=1 ,colour="Multiplying"))+
  scale_color_manual(name = "Lines", values = c("Summation" = "darkblue", "Multiplying" = "red"))+
  ylab("Temperature")+
  ggtitle("Figure 4:Comparesion of Sumatinon and Multiplying kernels")
```

Appendix 2: Code for Assignment 2

```
library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo, ]
spam[, -58] <- scale(spam[, -58])
tr <- spam[1:3000, ]
va <- spam[3001:3800, ]
trva <- spam[1:3800, ]
te <- spam[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, scaled = FALS
  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,
                scaled = FALSE)
mailtype <- predict(filter0, va[, -58])
t <- table(mailtype, va[, 58])
err0 <- (t[1, 2] + t[2, 1]) / sum(t)
err0

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

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

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)
```

```r
err3

sv <- alphaindex(filter3)[[1]]
co <- coef(filter3)[[1]]
inte <- -b(filter3)

k <- NULL
for (i in 1:10) {
  # We produce predictions for just the first 10 points in the dataset.
  x_star <- spam[i, -58]
  k2 <- NULL
  for (j in 1:length(sv)) {
    k2 <- c(k2,
            exp( -0.05 * sum( (x_star - spam[sv[j], -58])^2 ) )
    )
  }

  k <- c(k, sum(co * k2) + inte)
}
k
predict(filter3,spam[1:10,-58], type = "decision")

# the vectorized version

sv <- alphaindex(filter3)[[1]]
co <- coef(filter3)[[1]]
inte <- -b(filter3)

important_obs <- spam[sv, -58]
sv_mat <- as.matrix(important_obs)

result <- vector("numeric", 10)
for(i in 1:10){
  x_star <- unname(unlist(spam[i, -58]))
  dist <- colSums((x_star - t(sv_mat))^2)
  k <- exp(-dist * 0.05)
  result[i] <- sum(k*co) + inte
}

knitr::kable(data.frame("manual_predictions" = result, "predict_function" = predict(filter3,spam[1:10,-5
```

Appendix 3: Code for Assignment 3

```
library(neuralnet)

# Question 1 ------------------------------------------------------------------

# Generate initial data set and initial weights for the neural network
set.seed(1234567890)
Var <- runif(500, 0, 10)
set.seed(1234567890)
winit <- runif(31, -1, 1)
mydata <- data.frame(Var, Sin=sin(Var))

# Split data into training and testing datasets
tr <- mydata[1:25,]
te <- mydata[26:500,]

# Train the neural network and predict sine values for the testing data
nn <- neuralnet(formula = Sin ~ Var, data = tr,
                hidden = 10, startweights = winit)
te$predictions <- predict(nn, newdata = te)

# Plot the results
plot(tr, cex=1, col = "blue",
     main = "Figure X: True vs Predicted Function Values from Neural Net",
     xlab = "X Value",
     ylab = "Function Value")
points(te, col = "blue", cex=1)
points(te[,1], te[,3], col="red", cex=1)
legend(x = "bottomleft",
       legend = c("True", "Predicted"),
       lty = c(1, 1),
       col = c("blue", "red"))


# Question 2 ------------------------------------------------------------------

# Create linear, ReLU, and softplus activation functions
linear <- function(x){
  x
}

relu <- function(x){
  ifelse(x <= 0, 0,x)
}

softplus <- function(x){
  log(1+exp(x))
}

# Train neural networks with each of the activation functions
nnlin <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10,
                   act.fct = linear, startweights = winit)
```

```
nnrelu <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10,
                    act.fct = relu, startweights = winit)
nnsp <- neuralnet(formula = Sin ~ Var, data = tr, hidden = 10,
                  act.fct = softplus, startweights = winit)

# Predict sine values of the testing data with each model
te$predlin <- predict(nnlin, newdata = te)
te$predrelu <- predict(nnrelu, newdata = te)
te$predsp <- predict(nnsp, newdata = te)

# Plot results of linear activation function
plot(tr, cex=1, col = "blue",
     main = "True vs Predicted Values - Linear Activation",
     xlab = "X Value",
     ylab = "Function Value")
points(te, col = "blue", cex=1)
points(te[,1], te[,4], col="red", cex=1)
legend(x = "bottomleft",
       legend = c("True", "Predicted"),
       lty = c(1, 1),
       col = c("blue", "red"))

# Plot results of ReLU activation function
plot(tr, cex=1, col = "blue",
     main = "True vs Predicted Values - ReLU Activation",
     xlab = "X Value",
     ylab = "Function Value")
points(te, col = "blue", cex=1)
points(te[,1], te[,5], col="red", cex=1)
legend(x = "bottomleft",
       legend = c("True", "Predicted"),
       lty = c(1, 1),
       col = c("blue", "red"))

# Plot results of softplus activation function
plot(tr, cex=1, col = "blue",
     main = "True vs Predicted Values - Softplus Activation",
     xlab = "X Value",
     ylab = "Function Value")
points(te, col = "blue", cex=1)
points(te[,1], te[,6], col="red", cex=1)
legend(x = "bottomleft",
       legend = c("True", "Predicted"),
       lty = c(1, 1),
       col = c("blue", "red"))


# Question 3 ------------------------------------------------------------------

# Generate 500 data points across the interval [0,50]
set.seed(1234567890)
Var <- runif(500, 0, 50)
newdata <- data.frame(Var, Sin=sin(Var))
```

```r
# Use first neural network to predict sine values of the new data
newdata$predictions <- predict(nn, newdata = newdata)

# Plot the results
plot(tr, cex=1, xlim = c(0,50), ylim = c(-10, 2),
     main = "Figure X: True vs Predicted Function Values from Neural Net",
     xlab = "X Value",
     ylab = "Function Value")
points(newdata, col = "blue", cex=1)
points(newdata[,1], newdata[,3], col="red", cex=1)
legend(x = "bottomleft",
       legend = c("True", "Predicted"),
       lty = c(1, 1),
       col = c("blue", "red"))


# Question 4 ------------------------------------------------------------

# View weights of trained model and find which input weights are negative
nn$weights

# Find sum of output weights for hidden units with positive input weights
sum(nn$weights[[1]][[2]][-c(7, 9, 11)])

# Check that the model converges to this value as x -> infinity
predict(nn, data.frame(Var = 100000, Sin = sin(100000)))

# Question 5 ------------------------------------------------------------

# Create 500 new data points
set.seed(1234567890)
Var <- runif(500, 0, 10)
tr2 <- data.frame(Var, Sin=sin(Var))

# Train neural network to predict x from sin(x)
nninv <- neuralnet(formula = Var ~ Sin, data = tr2, hidden = 10,
                   threshold = 0.1, startweights = winit)
tr2$predictions <- predict(nninv, newdata = tr2)

# Predict x values from training data and plot results
plot(tr, cex=1, col = "blue",
     main = "Figure X: True vs Predicted Function Values from Neural Net",
     xlab = "X Value",
     ylab = "Function Value")
points(tr2, col = "blue", cex=1)
points(tr2[,3], tr2[,2], col="red", cex=1)
legend(x = "bottomleft",
       legend = c("True", "Predicted"),
       lty = c(1, 1),
       col = c("blue", "red"))

# Recreate this exercise, but predict sin(x) from x instead:
nn2 <- neuralnet(formula = Sin ~ Var, data = tr2, hidden = 10,
```

```
                threshold = 0.1, startweights = winit)
tr2$predictions2 <- predict(nn2, newdata = tr2)

# Predict sine values from training data and plot results
plot(tr, cex=1, col = "blue",
     main = "Figure X: True vs Predicted Function Values from Neural Net",
     xlab = "X Value",
     ylab = "Function Value")
points(tr2, col = "blue", cex=1)
points(tr2[,1], tr2[,4], col="red", cex=1)
legend(x = "bottomleft",
       legend = c("True", "Predicted"),
       lty = c(1, 1),
       col = c("blue", "red"))
```