TDDE01 - Laboration 3 Block 1

Olof Simander (olosi122), Gustaf Lindgren (gusli281), Anton Jervebo (antje840)

Dec 2022

Statement of Contribution

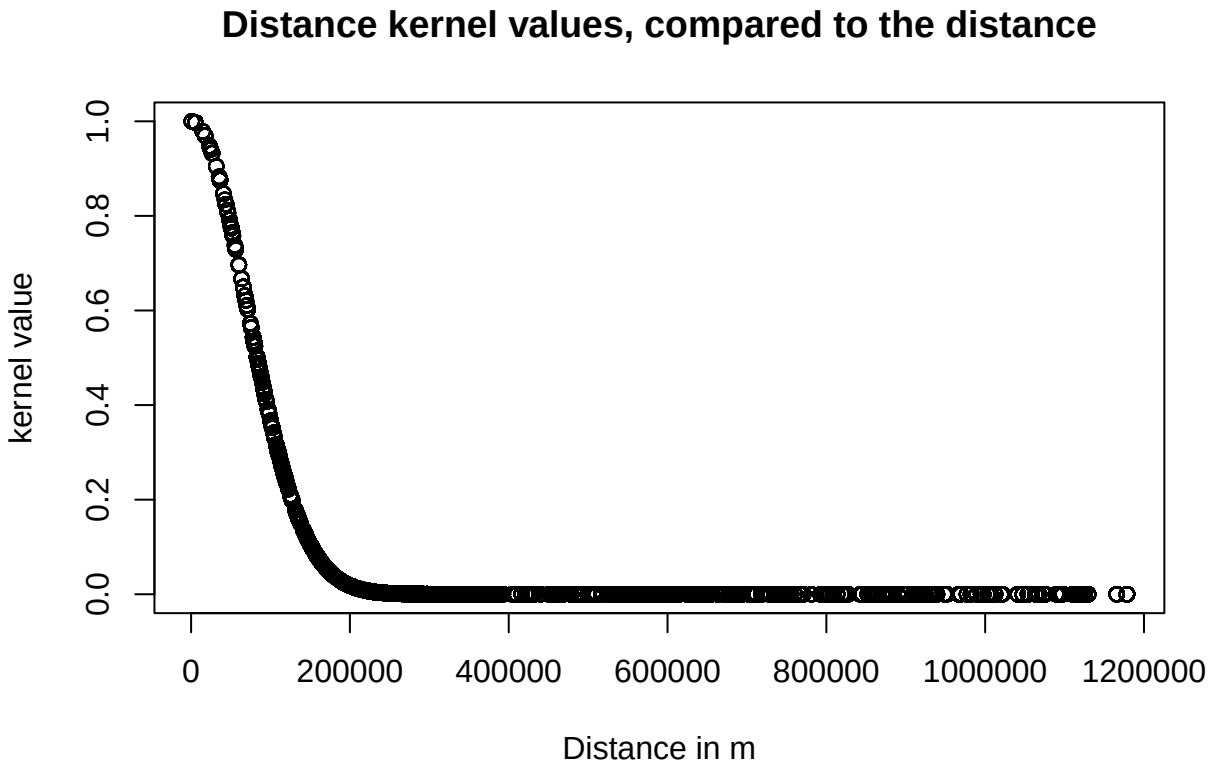
The code and writing of assignment 1 was mainly contributed to by Anton Jervebo. Gustaf Lindgren was responsible for the coding and writing of Assignment 2. Assignment 3 was mainly contributed to by Olof Simander. During various stages of completion, including during the analyses and after completion of writing, the assignments were discussed within the group. These discussions has influenced the report.

Assignment 1

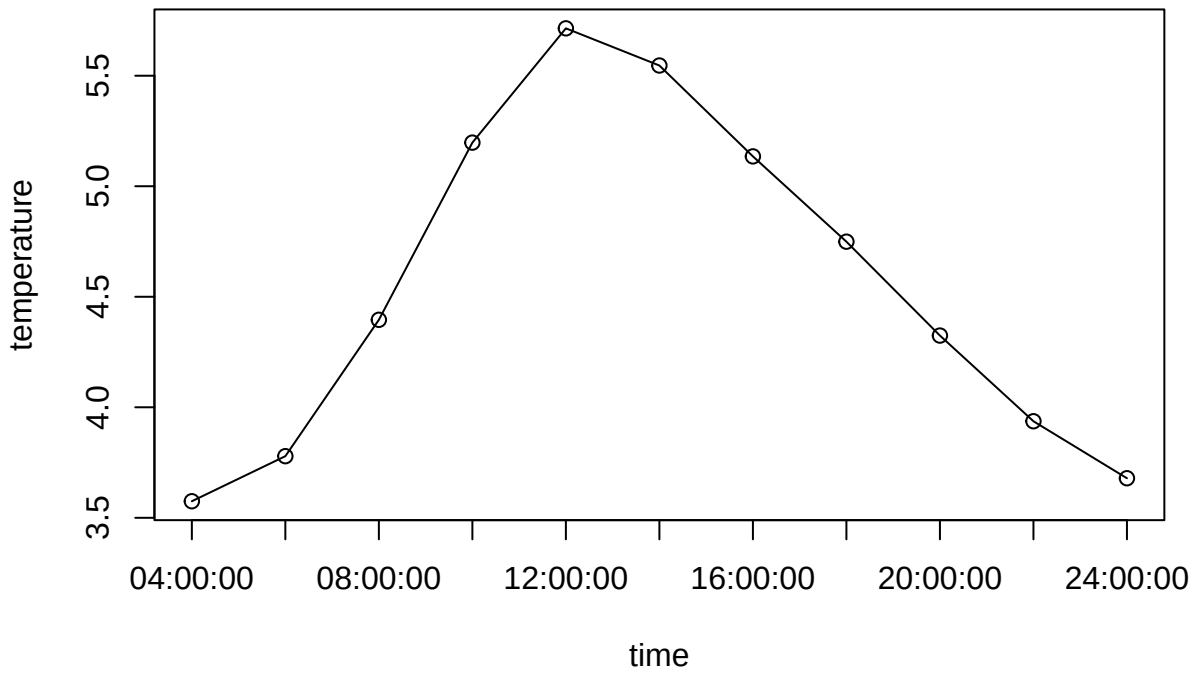
- Create three Gaussian kernels
  - distance, day_difference, hour_difference
- Create a sum of the three kernels
- Create a product of the three kernels
- Elaborate on the reason the may differ

The first step is to choose smoothing factors for the distance, date and time. For the distance between the station and the predicted point, we choose a value to favor the closer measurements over the measurements from stations far from the position we want to predict. In the first plot below, the correlation between distance and kernel value can be seen. For the date, it's feels important to favor the measurement that are closer in time to the date we want to predict, than those that are from a completely different season. Finally, for the smoothing factor for the hours of the day, we chose a smoothing factor that is favoring measurements closer in time of day since the temperatures can fluctuate quite a bit over the day.

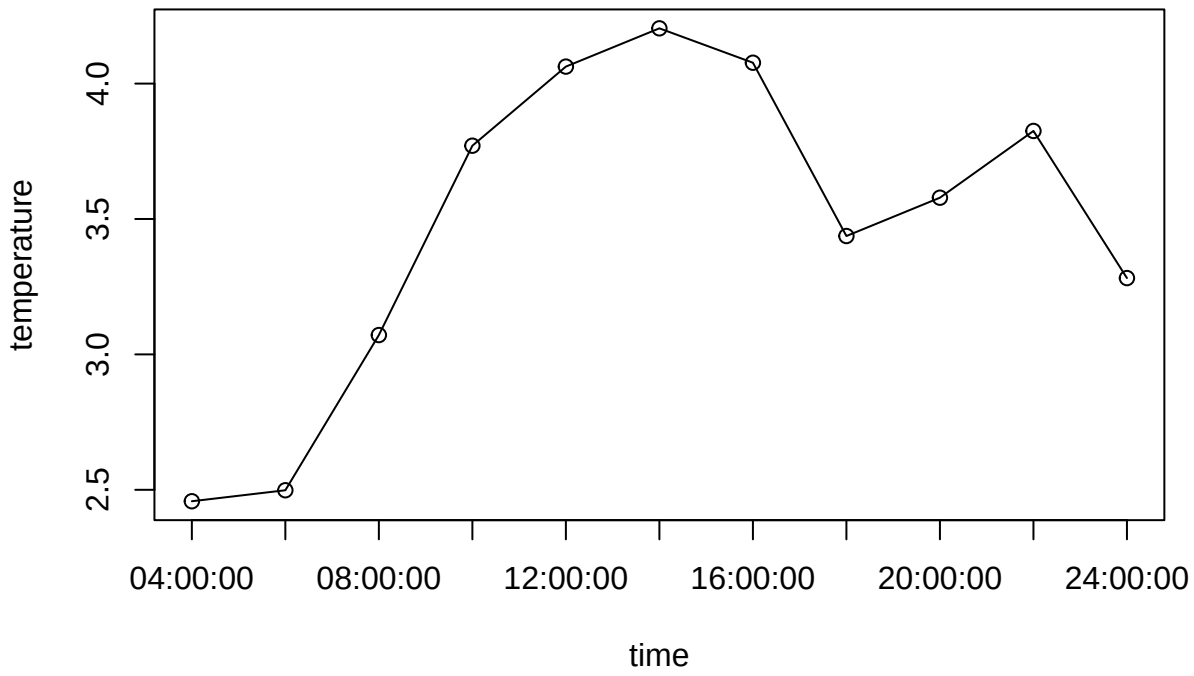
For the predicted position and date, we chose to predict the temperatures in Linköping on 2012-11-03.



The plots below show the predicted temperatures for Linköping at that date. The first plot uses a kernel that is a sum of three Gaussian kernels, and the second one uses a kernel that is the product of three Gaussian kernels.



Looking at the kernel that is a sum of three Gaussian kernels, we can see that the temperatures are starting at a bit cooler than 5 degrees, and are rising to 5.3 degrees at noon. It then predicts the temperatures to steadily decrease for the rest of the day. This doesn't feel like an unreasonable prediction for a November day in Linköping.



Looking at the plot that is the product of three Gaussian kernels, we can see that it's a bit different compared to the previous plot. The first thing to note is the fact that the temperature is decreasing between 4 and 6 in the morning. This feels reasonable considering the sun isn't up at that time. Other things that we can see is that the temperatures are lower than in the previous plot and that the decrease in temperature seems to stop at 17:00, and even starts rising again.

The method to calculate the air temperature is using the score from the three kernels as weights. When using the sum of three Gaussian kernels, the weight has the possibility to be high as long as the point and time we want to predict is close in either distance, time of year or time of day. This is not true while using the multiplication, where the weight is close to zero as long as one of the values from the three kernels are close to zero. This means that the temperature prediction is highly dependent on the physical distance and difference in time of day or year when using the multiplication method.

Appendix A

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

set.seed(1234567890)
library(geosphere)

# Had to save the stations.csv with the correct encoding (UTF-8) to be able to read it
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
```

```r
st <- merge(stations, temps, by = "station_number")
h_distance <- 100000 # These three values are up to the students
h_date <- 10
h_time <- 2
a <- 58.41086 # The point to predict (up to the students)
b <- 15.62157

prediction_point <- t(c(b, a))

pred_date <- "2012-11-03" # The date to predict (up to the students)
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"
  )
temp_sum <- vector(length = length(times))
temp_prod <- vector(length = length(times))

# Students' code here

# Filtering measurements on date
st$date <- as.Date(st$date, format = "%m/%d/%Y")
st_filtered <- subset(st, st$date < as.Date(pred_date))

dist_kernel <- function(measured_pos, pred_pos) {
  distance <- distHaversine(measured_pos, pred_pos)
  plot(
    distance,
    exp(-(distance / h_distance) ^ 2),
    type = "p",
    main = "Distance kernel values, compared to the distance",
    xlab = "Distance in m",
    ylab = "kernel value"
  )
  return (exp(-((
    distance / (h_distance)
  ) ^ 2)))
}

day_kernel <- function(measured_date, pred_date) {
  date_diff <-
    as.numeric(as.Date(measured_date) - as.Date(pred_date)) %% 365
  date_diff <-
    ifelse(date_diff < 365 / 2, date_diff, 365 - date_diff)
```

```
  return (exp(-((
    date_diff / (h_date)
  ) ^ 2)))
}

hour_kernel <- function(measured_hour, pred_hour) {
  hour_diff <-
    as.numeric(abs(difftime(
      strptime(measured_hour, format = "%H:%M:%S"),
      strptime(pred_hour, format = "%H:%M:%S"),
      units = "hours"
    )))
  hour_diff <- ifelse(hour_diff <= 12, hour_diff, 24 - hour_diff)
  return (exp(-((
    hour_diff / (h_time)
  ) ^ 2)))
}


measured_positions <-
  data.frame(as.numeric(st_filtered$longitude),
             as.numeric(st_filtered$latitude))


dist_kernel_score <-
  dist_kernel(measured_positions, prediction_point)
day_kernel_score <- day_kernel(st_filtered$date, as.Date.character(pred_date))

hour_kernel_score <-
  matrix(nrow = nrow(st_filtered), ncol = length(times))

col <- 1
for (time in times) {
  hour_kernel_score[, col] <- hour_kernel(st_filtered$time, time)
  col <- col + 1
}

# sum of kernels
for (i in 1:length(times)) {
  k_sum <-
    dist_kernel_score + day_kernel_score + hour_kernel_score[, i]
  k_sum <- k_sum / sum(k_sum)
  temp_sum[i] <- sum(k_sum %*% st_filtered$air_temperature)
}

plot(
  temp_sum,
  type = "o",
  xlab = "time",
  ylab = "temperature",
  xaxt = "n"
)
axis(1, at = 1:length(times), labels = times)
```

```r
# mult of kernels
for (i in 1:length(times)) {
  k_prod <-
    dist_kernel_score * day_kernel_score * hour_kernel_score[, i]
  k_prod <- k_prod / sum(k_prod)
  temp_prod[i] <- sum(k_prod %*% st_filtered$air_temperature)
}

plot(
  temp_prod,
  type = "o",
  xlab = "time",
  ylab = "temperature",
  xaxt = "n"
)
axis(1, at = 1:length(times), labels = times
)
```

Assignment 2

Which filter do you return to the user ? filter0, filter1, filter2 or filter3? Why?

At first glance, it would make sense to return *filter3* since it has a significantly lower error (0.0287) than the other three filters. However, this model has been trained on the entire spam data set and its performance is then evaluated on the test data set, which is a subset of the spam data set i.e., the model is evaluated on data points it has already been trained on. This naturally results in a low error rate for the model.

In comparison, the other models are trained and evaluated on different data sets which makes the calculated errors for those models more meaningful. Though for some reason the *validation* data set is used for the predict function in *filter0* instead of the *test* data set which is used in *filter1* and *filter2* which means that we cannot compare them directly. Based on this, we would choose either *filter0* or *filter2.*

Another aspect to consider, is the complexity of the model. Since *filter0* (and *filter1*) is only trained on the train data set, the number of support vectors are fewer than because the models are trained on fewer data points.

What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why?

- **err0:** 0.0625
- **err1:** 0.07615481
- **err2:** 0.06991261
- **err3:** 0.02871411

As mentioned in the previous question, the generalization error rate is significantly lower for *filter3* since it is trained and tested on the same data. The other models are trained and validated on different data sets which naturally result in a higher error rate. Interestingly, *filter2* which is trained on both the train and validation data sets, performs worse than *filter0* which is only trained on the train data set. This is probably because *filter0* is evaluated on a different data set than the other filters.

Implementation of SVM predictions.

Our own implementation of the predict function yields the exact same results as the predict function. This makes sense since our implementation and the predict function uses the same data points, support vectors and kernel function with the same value for sigma.

Appendix: Code for assignment 2

```
## [1] 0.0625
```

```
## [1] 0.07615481
```

```
## [1] 0.06991261
```

```
## [1] 0.02871411
```

```
## [1] "Our implementation"
```

```
##             [,1]
##  [1,]  0.8185775
##  [2,]  0.9999825
##  [3,]  2.2142890
##  [4,] -2.5851401
##  [5,]  1.7727880
##  [6,] -1.2596537
##  [7,]  1.0026088
##  [8,]  2.5794347
##  [9,]  0.9998688
## [10,]  1.2206699
```

```
## [1] "prediction function"
```

```
##             [,1]
##  [1,]  0.8185775
##  [2,]  0.9999825
##  [3,]  2.2142890
##  [4,] -2.5851401
##  [5,]  1.7727880
##  [6,] -1.2596537
##  [7,]  1.0026088
##  [8,]  2.5794347
##  [9,]  0.9998688
## [10,]  1.2206699
```

```
#Code from Lab3Block1_2021_SVMs_St.R with some modifications
set.seed(12345)
library(kernlab)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
spam[,-58]<-scale(spam[,-58])
tr <- spam[1:3000, ] #train
va <- spam[3001:3800, ] #validation
trva <- spam[1:3800, ] #train and validation
te <- spam[3801:4601, ]  #test

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=FALSE)
  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=FA
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=FA
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3

# Questions

# 1. Which filter do we return to the user ? filter0, filter1, filter2 or filter3? Why?

# 2. What is the estimate of the generalization error of the filter returned to the user? err0, err1, e

# 3. Implementation of SVM predictions.

sv<-alphaindex(filter3)[[1]] #return index of support vectors
co<-coef(filter3)[[1]] # linear coefficients of support vectors
inte<- - b(filter3) # negative linear intercept of linear combination
train_10 = spam[1:10, -58]
kernel = rbfdot(sigma = 0.05) #Radial Basis kernel function "Gaussian"
pred=matrix(0, nrow=10, ncol=1)
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset

  sample = unlist(train_10[i,])
  pred_new = inte
  for(j in 1:length(sv)){
    sv_values <- unlist(spam[sv[j], -58]) #return all columns except 58 of the sv for the given index
    pred_new <- pred_new + co[j]*kernel(sample, sv_values)
  }
  pred[i] = pred_new
}
print("Our implementation")
pred
print("prediction function")
predict(filter3,spam[1:10,-58], type = "decision")
```

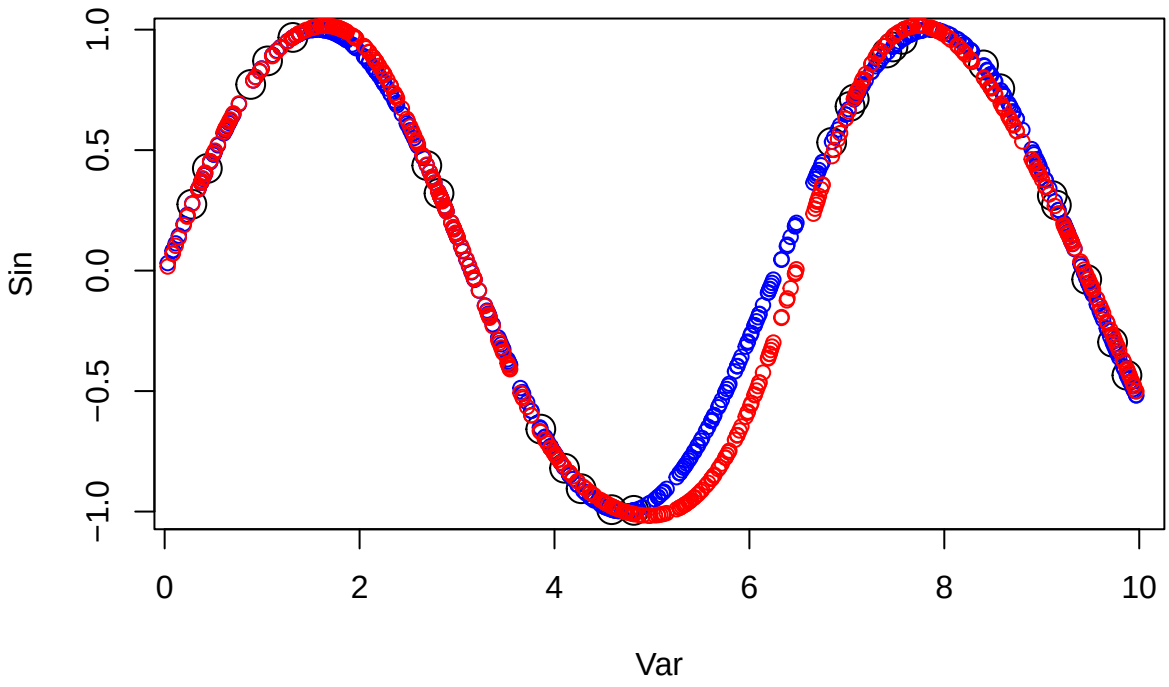
```
<style>
body {
text-align: justify}
</style>
```

Assignment 3

Task 1

- Train a neural network to learn the trigonometric sine function.
- Use 25 of the 500 points for training and the rest for test. Use one hidden layer with 10 hidden units. You do not need to apply early stopping. Plot the training and test data, and the predictions of the learned NN on the test data. You should get good results.
- Comment your results.

Overall, with the small amount of training data, the model fits the test data well. The interval between about 5 and 7 has a slightly worse fit. This might be due to the lack of data points in the training data for this range.



Task 2

- Repeat question (1) with the following custom activation functions: $h_1(x) = x$, $h_2(x) = max(0, x)$, $h_3(x) = ln(1 + e^x)$ (a.k.a. linear, ReLU and softplus). See the help file of the neural net package to learn how to use custom activation functions.

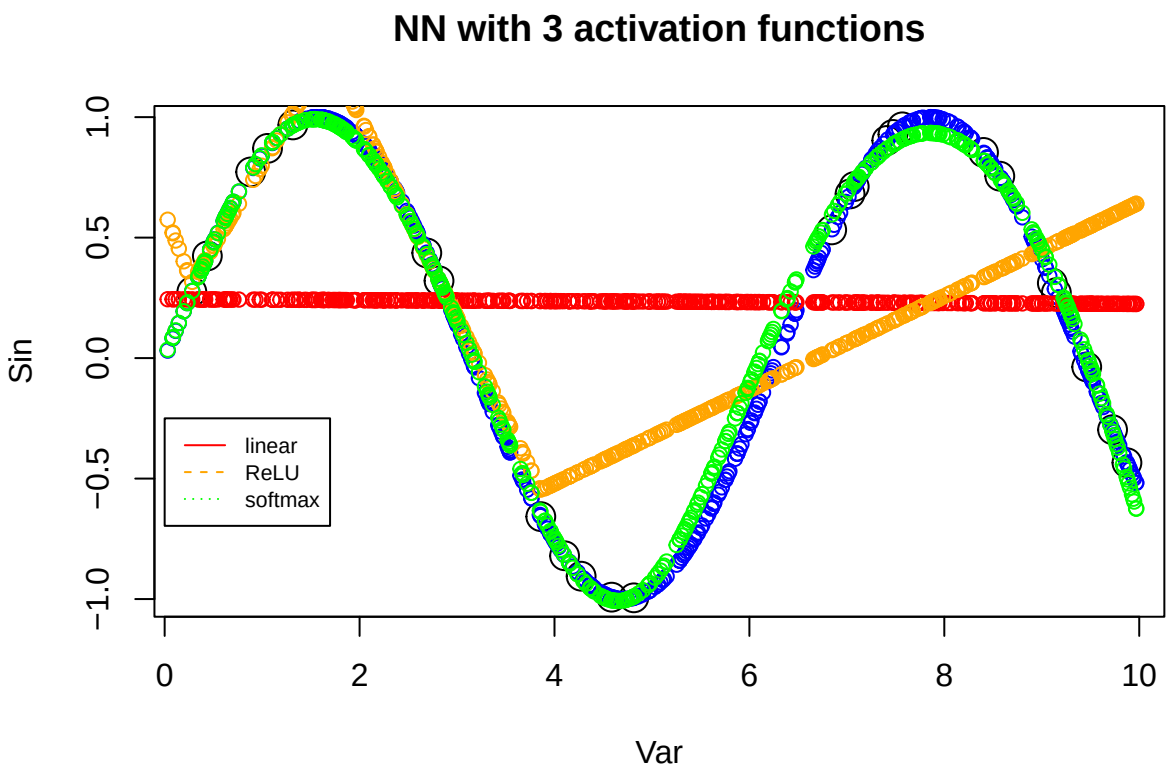
- Plot and comment your results.

The three activation functions have very different results.

The first, linear, has a near constant value of 0.2343153. This is because it is explicitly linear, which naturally will not fit a sine function very well. The reason it does not approximate 0 is because there are more data points in the first quadrant then the fourth.

The second, ReLU, is clearly linear in segments of the data. However, it has points for which it is non-linear, which is explained by the lack of a defined derivative of the max function. It does not predict the test data very well.

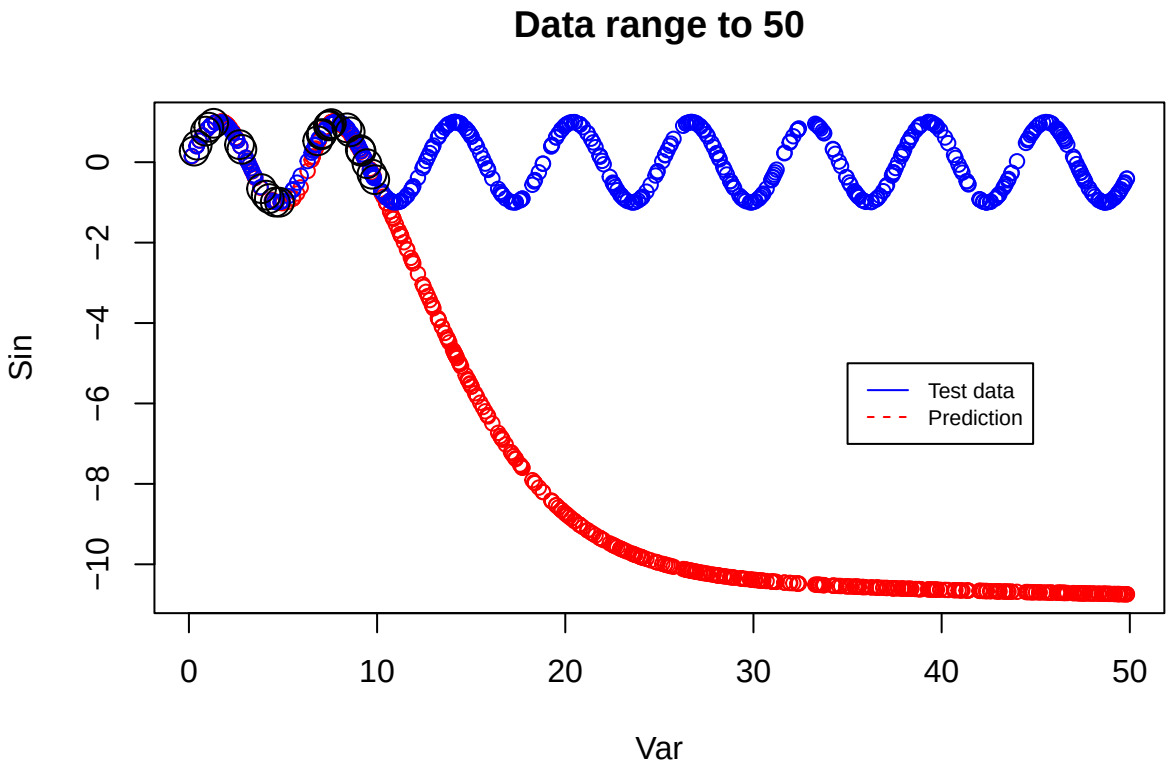
The third, softmax, approximates the test data well. It has infinitely many derivatives and arguably performs better than the sigmoid activation function in task 1, especially in the interval 5 to 7.



Task 3

- Sample 500 points uniformly at random in the interval [0,50], and apply the sine function to each point. Use the NN learned in question (1) to predict the sine function value for these new 500 points. You should get mixed results.
- Plot and comment on your results.

Observing the graph, the predictions from the first neural network does predict the data well on the interval of 0 to 10. This is because the model is trained on this range. However, beyond 10, the sigmoid activation function results in an extrapolated logistic curve.



Task 4

- In question (3), the predictions seem to converge to some value. Explain why this happens.

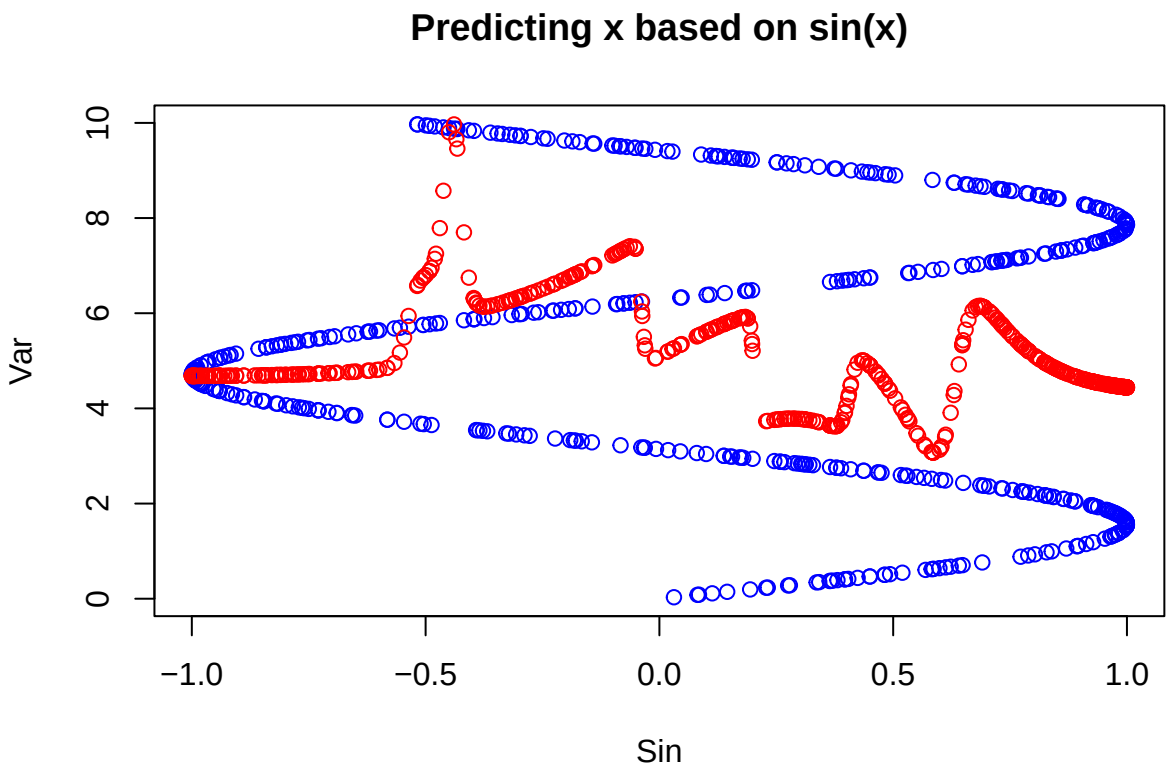
We did not choose a specific seed for the starting weights for the NN. One randomization gives -10.74471 as the minimum value which the prediction converges towards. As explained above, the reason the prediction deviates much more after Var = 10 is because it has not been trained on this data. Instead, it will be based on the derivatives from the sigmoid function

$$h(x) = \frac{1}{1 + e^{-x}}.$$

Task 5

- Sample 500 points uniformly at random in the interval [0,10], and apply the sine function to each point. Use all these points as training points for learning a NN that tries to predict x from sin(x), i.e. unlike before when the goal was to predict sin(x) from x. Use the learned NN to predict the training data. You should get bad results.
- Plot and comment your results.

This is not a good fit. The main reason is because of the nature of the arcsine function: it has multiple results for a given x, unless the output is constrained. In this case, it is constrained between 0 and 10. But this includes multiple periods of the sine function. Therefore, the NN will have problems trying to predict multiple outcomes for a given input.



Assignment 1 Appendix

```
library(neuralnet)
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
# 31 randomized weights with 20 for hidden nodes and 11 for outcome node
# This is evident when plotting the network
winit = runif(31,-1,1)

# Create a neural network model with 1 hidden layer and 10 hidden units
nn = neuralnet(Sin~., tr, hidden = 10, startweights = winit)
#plot(nn)
# Plot of the training data (black), test data (blue), and predictions (red)
plot(tr, cex=2)
points(te, col = "blue", cex=1)
points(te[,1],predict(nn,te), col="red", cex=1)
linear = function(x) x
ReLU = function(x) ifelse(x>0,x,0)
softmax = function(x) log(1 + exp(x))

# Create a neural network model with 1 hidden layer and 10 hidden units
# with 3 different activation functions
nn_r = neuralnet(
```

```
  Sin~., tr, hidden = 10, act.fct = ReLU, startweights = winit, threshold = 0.1)
pred_r = predict(nn_r,te)

nn_l = neuralnet(
  Sin~., tr, hidden = 10, act.fct = linear, startweights = winit)
pred_l = predict(nn_l,te)
#mean(pred_l)

nn_s = neuralnet(
  Sin~., tr, hidden = 10, act.fct = softmax, startweights = winit)
pred_s = predict(nn_s,te)

plot(tr, cex=2, main = "NN with 3 activation functions")
points(te, col = "blue", cex=1)
points(te[,1],pred_l, col="red", cex=1)
points(te[,1],pred_r, col="orange", cex=1)
points(te[,1],pred_s, col="green", cex=1)
legend(
  0,-0.25, legend = c("linear", "ReLU","softmax"),
  col = c("red", "orange", "green"), cex = 0.7, lty = 1:3)
set.seed(1234567890)
Var = runif(500,0,50)
mydata_2 = data.frame(Var, Sin = sin(Var))
preds = predict(nn, mydata_2)
#min(preds)

plot(
  mydata_2[,1], preds, col="red", cex=1, xlab = "Var",
  ylab = "Sin", main = "Data range to 50")
points(mydata_2[,1], mydata_2[,2],  col = "blue", cex=1)
points(tr, cex = 2)
legend(
  35,-5, legend = c("Test data", "Prediction"),
  col = c("blue", "red"), lty = 1:2, cex = 0.7)
plot(nn)
set.seed(1234567890)
Var = runif(500,0,10)
mydata_3 = data.frame(Var, Sin = sin(Var))

nn_2 = neuralnet(
  Var~., mydata_3, hidden = 10, startweights = winit,
  threshold = 0.1)

preds_2 = predict(nn_2, mydata_3)

plot(
  mydata_3[,2], mydata_3[,1], col = "blue", cex=1,
  xlab = "Sin", ylab = "Var", main = "Predicting x based on sin(x)")
points(mydata_3[,2], preds_2, col="red", cex=1)
```