TDDE01: Machine Learning, LAB3

Max Lund

December 11, 2017

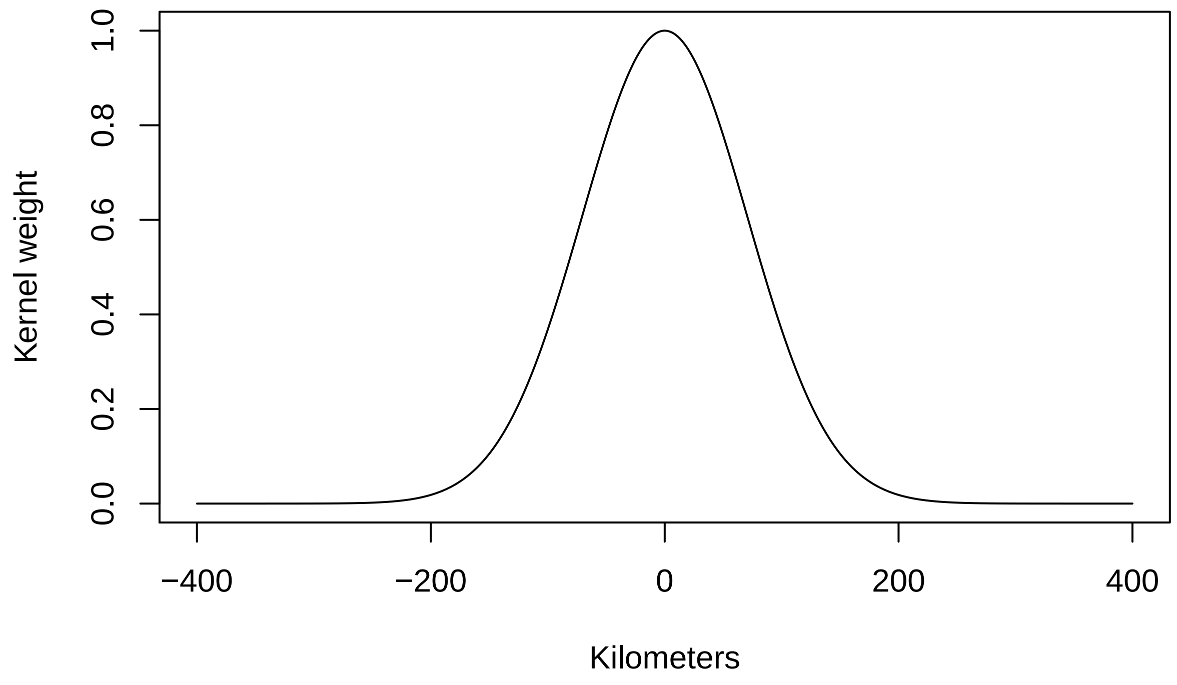
1 Assignment 1: Kernel classifier

The dataset for the first assignment contains informations from weather stations around the country. Our task is to create a gaussian kernel classifier that uses three different kernels, each using a different metric for measuring the distance between a data point to predict and an observation. The distance measures are either distance between the locations, in the day of the year, or in the time of the day. The objective to predict is the air temperature at a given location, at some date, at some time of the day. The general gaussian kernel that all three kernels use is the following:

```
# the general function used by all kernels
gaussian=function(distance, h_value)
{
  return (exp(-(distance^2 / h_value)))
}
```

The h-value of the function determines the width of our distribution, i.e. when the weights of the kernel will start becoming very small and thus not contributing much to the prediction. For the *distance* kernel, we reasoned by looking at some weather maps that any location more than 200 km away from our target for prediction should be disregarded. By trying a few different h-values, we plotted until we found a suitable value:

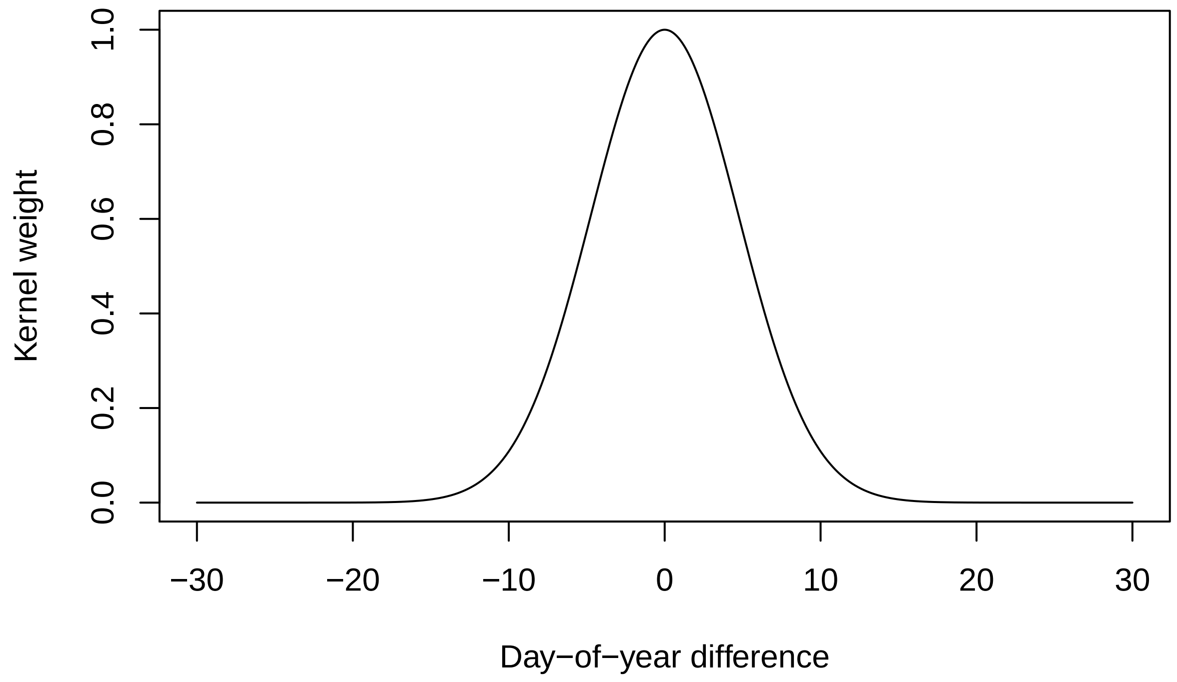
```
dist = seq(-400, 400, by=1)
plot(dist, gaussian(dist, 10000), type="l", xlab="Kilometers", ylab="Kernel weight")
```



```
# a good width/h-value for the distance kernel seems to be around 10000
h_distance = 10000
```

Next we repeated the process to find a good h-value for our kernel measuring the *day of the year* difference. Looking at some diagrams of average temperatures throughout the year, we reasoned that around +/- 15 days should be a good threshold for our predictions:

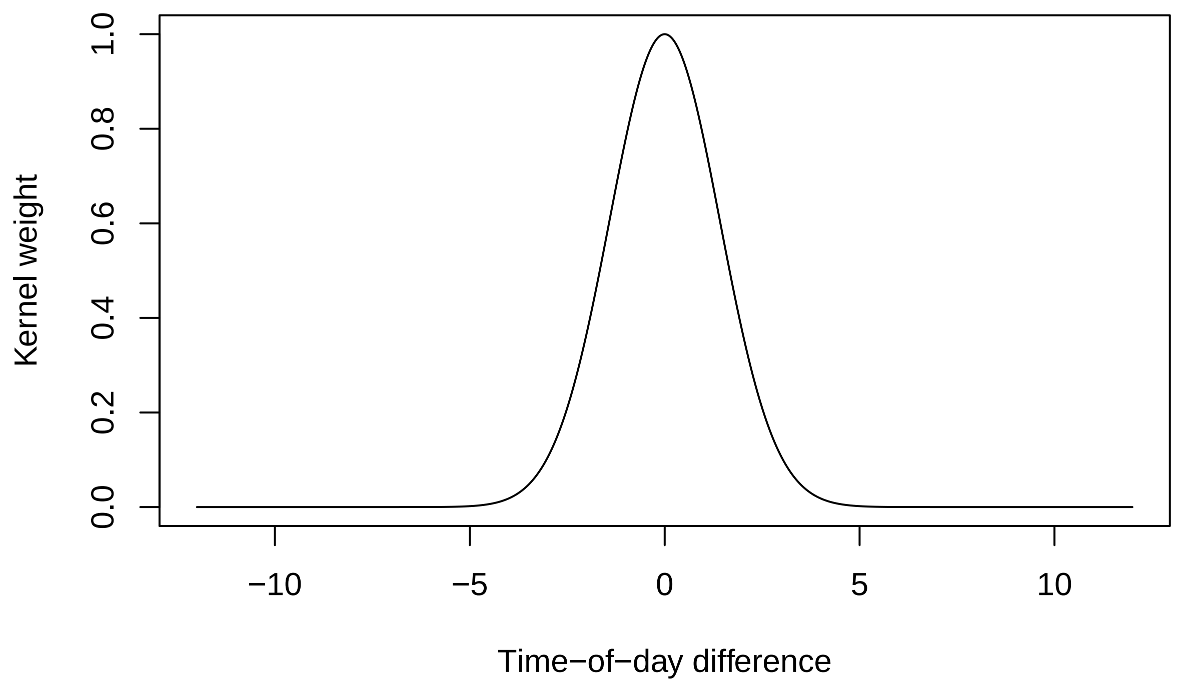
```
dist = seq(-30, 30, by=0.1)
plot(dist, gaussian(dist, 45), type="l", xlab="Day-of-year difference", ylab="Kernel weight")
```



```
# a good h-value/width for the date/day-of-year kernel seems to be around 45
h_date = 45
```

Finally we repeat the process to find the h-value for our kernel measuring the *time of day* difference. Although this can be tricky since day-light hours in Sweden vary considerably between the seasons, we reasoned that a good threshold would be around +/- 4 hours:

```
dist = seq(-12, 12, by=0.01)
plot(dist, gaussian(dist, 4), type="l", xlab="Time-of-day difference", ylab="Kernel weight")
```



```
# a good h-value/width for the time-of-day kernel seems to be about 4
h_time = 4
```

1.1 Kernel functions

The distance kernel uses the *distHaversine* function to get the distance between two latitude- and longitude values.

```
distance_kernel=function(lat1, long1, lat2, long2)
{
  location1 = c(lat1, long1)
  location2 = c(lat2, long2)
  distance_km = distHaversine(location1, location2) / 1000
  return (gaussian(distance_km, h_distance))
}
```

The day-of-year/date kernel needs a few helper function to deal with leap-years when getting the difference in *days of the year* between two dates.

```
leaps = rev(c(
  "1932-2-29"
  ,"1936-2-29"
  ,"1940-2-29"
  ,"1944-2-29"
  ,"1948-2-29"
  ,"1952-2-29"
  ,"1956-2-29"
  ,"1960-2-29"
  ,"1964-2-29"
  ,"1968-2-29"
  ,"1972-2-29"
  ,"1976-2-29"
  ,"1980-2-29"
  ,"1984-2-29"
  ,"1988-2-29"
  ,"1992-2-29"
  ,"1996-2-29"
  ,"2000-2-29"
  ,"2004-2-29"
  ,"2008-2-29"
  ,"2012-2-29"
  ,"2016-2-29"
  ))

get_leapyears=function(d){
  for(i in 1:length(leaps)){
    if(as.Date(d) > as.Date(leaps[i])){
      return(i - 1)
    }
  }
}

get_day_diff=function(date1, date2){
  year1 = as.numeric(substr(date1, 0, 4))
  year2 = as.numeric(substr(date2, 0, 4))
  if (year1 > year2)
  {
    diff = difftime(strptime(date1, format="%Y-%m-%d"),
                    strptime(date2, format="%Y-%m-%d"), units="days")
  }
  else
  {
    diff = difftime(strptime(date2, format="%Y-%m-%d"),
                    strptime(date1, format="%Y-%m-%d"), units="days")
  }
  date1.leaps = get_leapyears(date1)
  date2.leaps = get_leapyears(date2)
  diff = diff - abs(date1.leaps - date2.leaps)
  diff = as.numeric(diff) %% 365
  return(min(diff, 365 - diff))
}

date_kernel=function(date1, date2)
{
  day_diff = get_day_diff(date1, date2)
  return (gaussian(day_diff, h_date))
}
```

Our time of the day kernel also needs a helper function to get the difference in hours between two times of the day:

```
get_time_diff=function(t1, t2)
{
  t1 = as.numeric(as.difftime(as.character(t1)), units="hours")
  t2 = as.numeric(as.difftime(as.character(t2)), units="hours")
  diff1 = t1 - t2
```

```r
  diff2 = t2 - t1
  return (min(diff1 %% 24, diff2 %% 24))
}

time_kernel=function(time1, time2)
{
  time_diff = get_time_diff(time1, time2)
  return (gaussian(time_diff, h_time))
}
```

1.2 Making predictions

The points for prediction we chose was at central Linköping on the 11th of December, making one prediction every two hours from 02:00:00 until 24:00:00.

```r
latitude = 58.41
longitude = 15.618
target_date = "2017-12-11"
times = c("02:00:00", "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")
```

We filter out the observations that lie later in time compared to our date.

```r
valid = subset(st, as.Date(date) < as.Date(target_date))
```

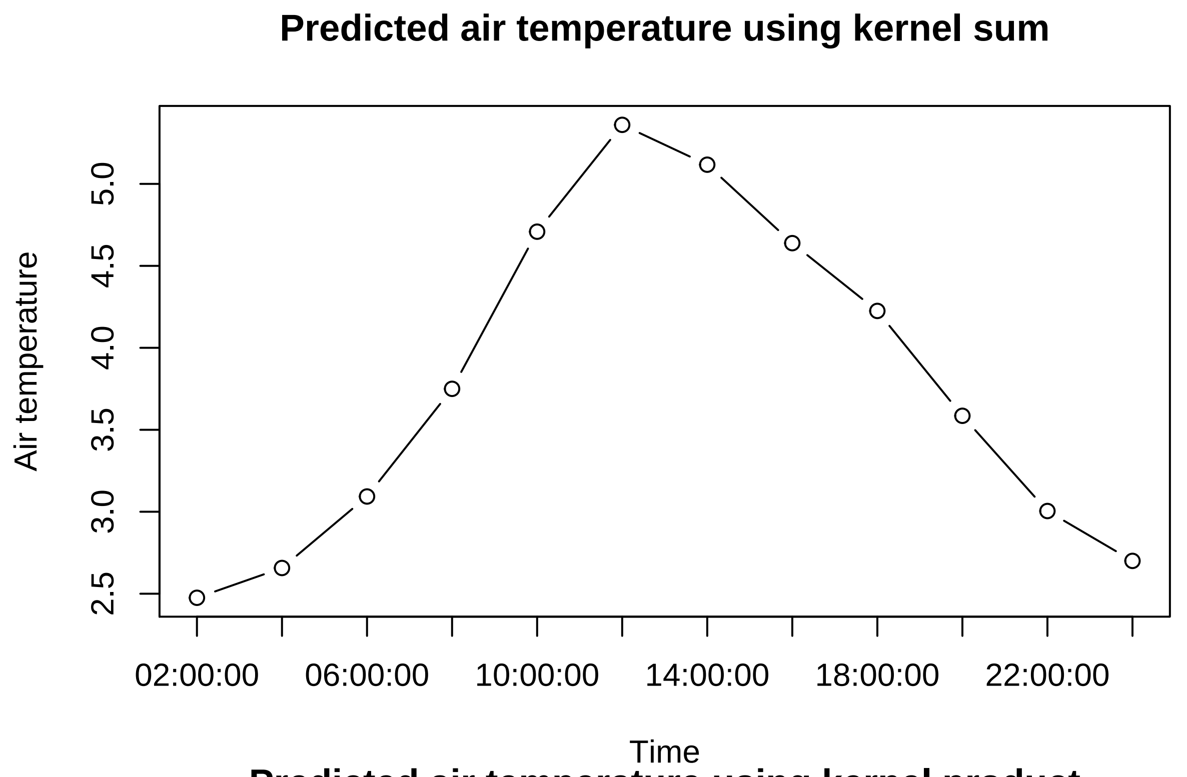
Then we get the weights from our and distance- and date kernels for every observation in the valid dataset.

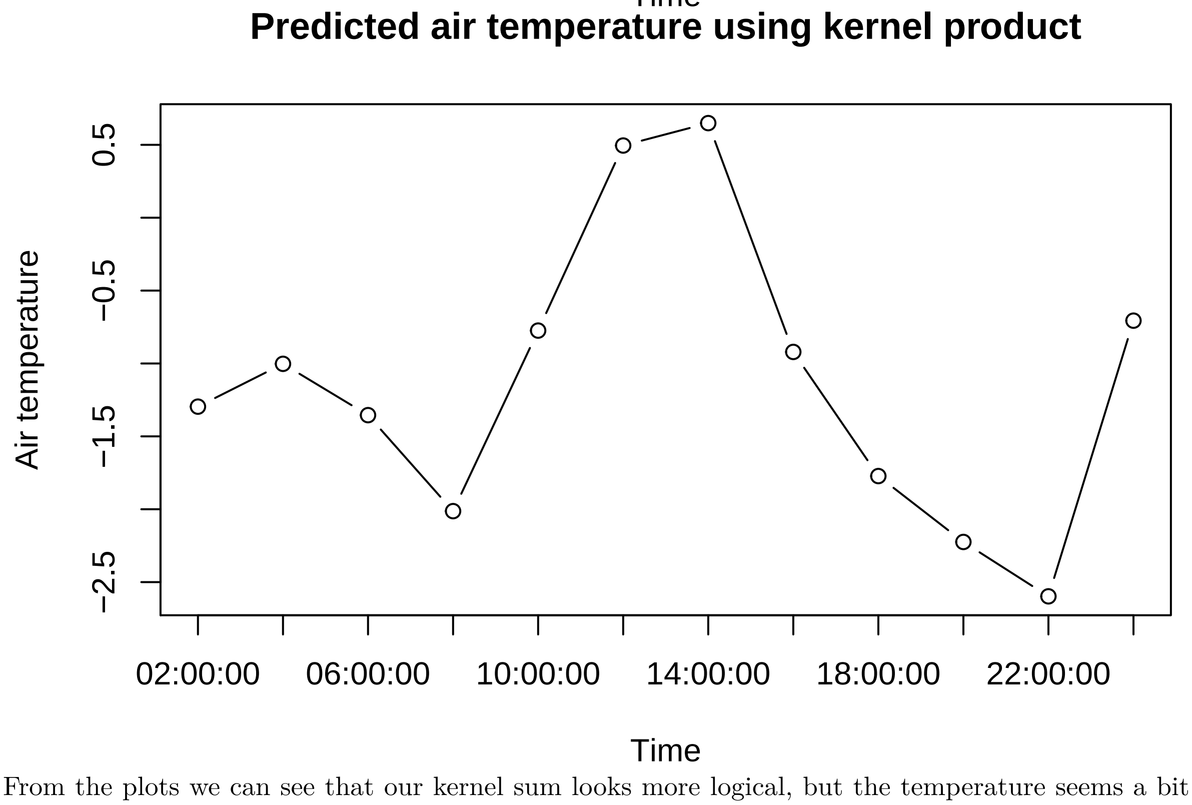
```r
distance_weights = mapply(distance_kernel,
                          valid$latitude, valid$longitude,
                          latitude, longitude)
date_weights = mapply(date_kernel, valid$date, target_date)
```

We then set up to get the kernel weights for each of the different times of day that we are predicting on. For making the final predictions for air temperature, we use both the sum and the product of our three kernel weights.

```r
preds_sum = numeric(length(times)) # the predictions using sum of kernels
preds_prod = numeric(length(times)) # .. using product

for (i in 1:length(times))
{
  # get weights from time kernel for each time
  time_weights = mapply(time_kernel, valid$time, times[i])
  kernel_sum = date_weights + distance_weights + time_weights
  kernel_prod = date_weights * distance_weights * time_weights
  # get the predicted temperature using both kernel sum and kernel product
  preds_sum[i] = sum(kernel_sum * valid$air_temperature) / sum(kernel_sum)
  preds_prod[i] = sum(kernel_prod * valid$air_temperature) / sum(kernel_prod)
}
```





From the plots we can see that our kernel sum looks more logical, but the temperature seems a bit high (although it was fairly accurate for today's date). The kernel product seems to have values that are closer to the average of this time of year, but the plot is strange since there is a distinct temperature increase towards midnight which doesn't seem reasonable.

1.3 Kernel sum vs. kernel product

The difference between using the kernel sum or the kernel product in our case is that the kernel product will consider the joint impact of all metrics used in an observation, while the sum will view every kernel weight as discrete. For example, say that the locations of the point we want to predict and an observation are very similar, then the distance kernel might output something close to 1. But if the other metrics for date and time differed greatly between the point for prediction and the observation, such as one being the the middle of summer and the other middle of winter, and the time of days being middle of the night or middle of the day, the two other kernels will output something close to 0 like 0.0001. The kernel sum will then still be close to 1, but the kernel product will be close to 0 since it takes 1 * 0.0001 * 0.0001.

2 Assignment 2: Neural network

The second assignment was to implement a neural network to estimate the sinus function using 50 observations split 50/50 into training and validation sets.

```r
library(neuralnet)
set.seed(1234567890)

# generate 50 random numbers between 0-10
samples = runif(50, 0, 10)
# get the sinus of samples
df = data.frame(Var=samples, Sin=sin(samples))
# split 50/50 training/validation sets
train = df[1:25, ]
validation = df[26:50, ]
```

The network we are using has one hidden layer of ten nodes. It is fully connected with bias weights on each node, including the output node. We therefore have 10 weights on the inputs, 10 on the output, and 11 bias weights, for a total of 31 weights. We randomly initialize the weights with values between -1 to 1 before starting to train the network.

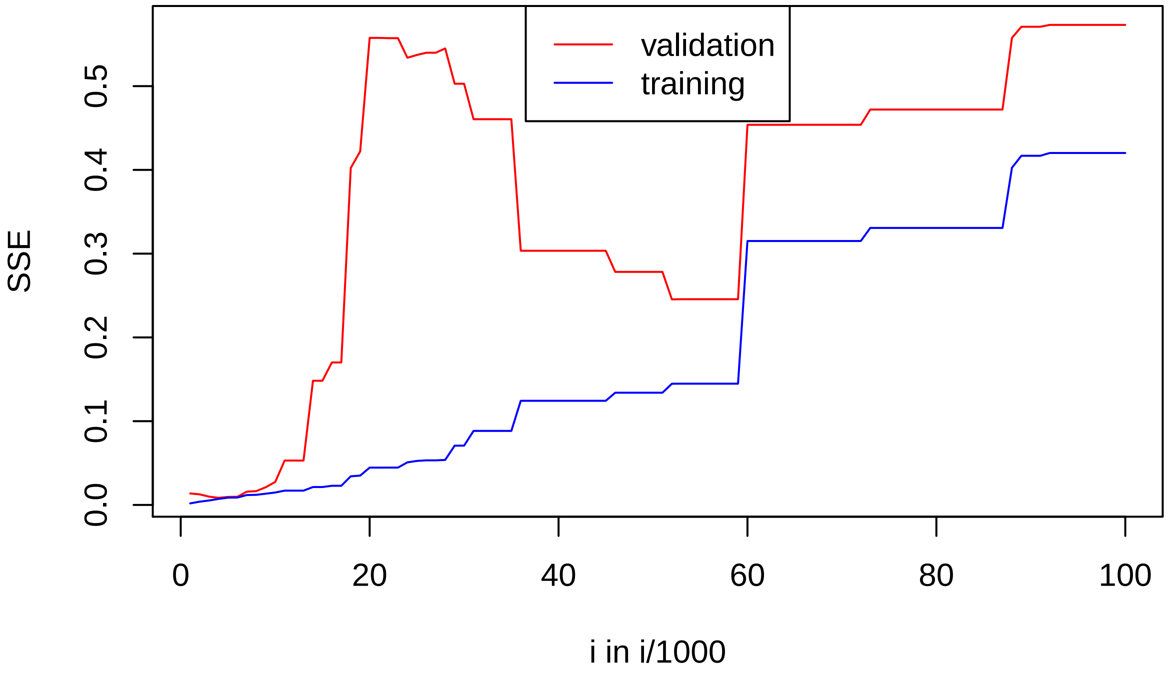
```r
# intialial random weights for NN (31 total)
init_weights = runif(31, -1, 1)
```

To find the best model, we retrain the network using progressively higher threshold for the partial derivatives of the error function, i.e. we make the threshold for the allowed error higher and therefore reduce overfitting successively. We then test the model on the validation set to see when the SSE for the predictions will be lowest.

```r
errors.valid = numeric(100)
errors.train = numeric(100)
# run NN with threshold i/1000 for i in 1:100
for(i in 1:100) {
  nn = neuralnet(
    formula = Sin ~ Var,
    data = train,
    startweights = init_weights,
    hidden = 10,
    threshold = i / 1000
  )

  # make predictions
  res.train = compute(nn, train[ ,"Var"])
  res.valid = compute(nn, validation[ ,"Var"])

  # calculate the SSE for each threshold
  errors.train[i] = sum((res.train$net.result - train[ ,"Sin"])^2)
  errors.valid[i] = sum((res.valid$net.result - validation[ ,"Sin"])^2)
}
```

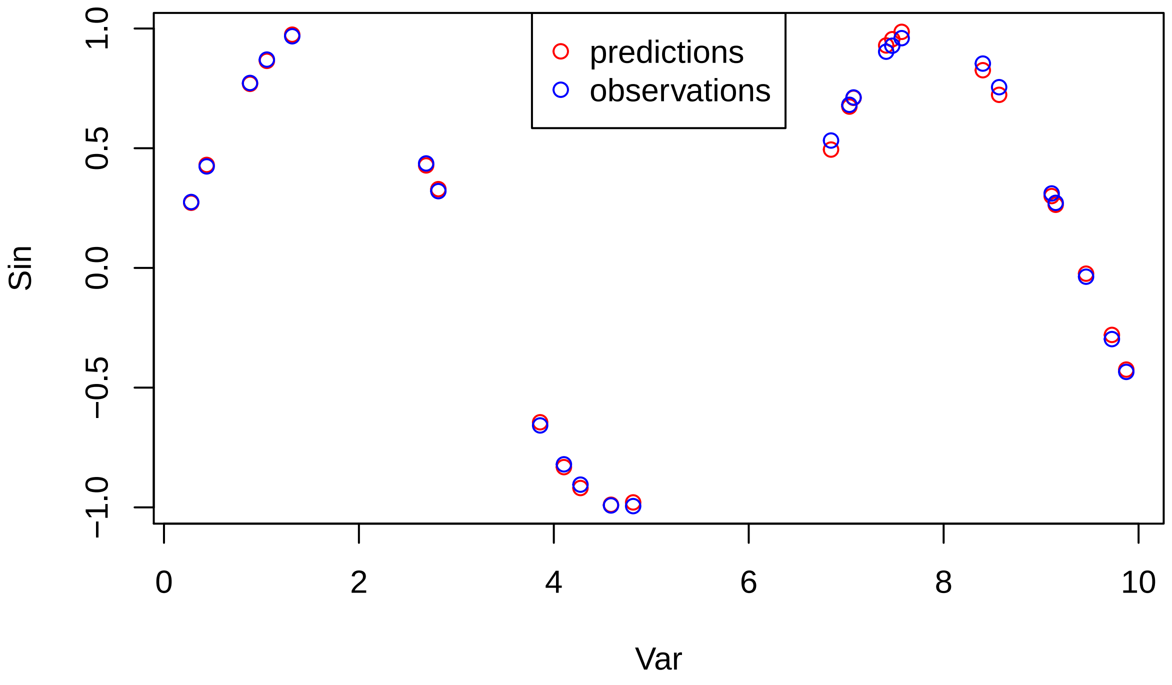


We then chose the best value for the threshold to select the best model. We plot this neural net model and the predictions it made together with the true observations.

```
## the i value in 1/1000 that gave lowest error: 4
```

```r
# use the best threshold to train our NN
best.threshold = which.min(errors.valid) / 1000
best.nn = neuralnet(
  formula = Sin ~ Var,
  data = train,
  startweights = init_weights,
  hidden = 10,
  threshold = best.threshold
)
```

```
## Data Error:  0;
```



```r
# plot NN
plot(best.nn)
```