Group_A11

Computer lab 3

Group A11 - Victor Guillo, Christian Kammerer, Jakob Lindner

Statement of contribution

Assignments 1 and 3 were mainly contributed by Victor Guillo. Christian Kammerer worked on assignment 2 and Jakob Lindner on assignment 4. Results from all assignments have been discussed afterwards between all group members.

1 THEORY

1 What is the kernel trick?

The kernel trick is a technique in machine learning that allows the use of non-linear models without explicitly transforming the input data into a higher-dimensional space. Instead of directly computing the transformation $\phi(x)$, the kernel function $\kappa(x, x')$ efficiently computes the inner product in the transformed space. This avoids the computational burden of handling the transformed features while enabling non-linear decision boundaries, such as in **Support Vector Machines (SVMs)**. The kernel trick works because models rely only on inner products, which can be replaced by a suitable kernel function. A common example is the Radial Basis Function (RBF) kernel.
Page 194

2 In the literature, it is common to see a formulation of SVMs that makes use of a hyperparameter C. What is the purpose of this hyperparameter?

The hyperparameter C serves as a regularization parameter in the SVM formulation. It controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights. A larger C gives more weight to achieving a low training error, which can lead to a smaller margin but less generalization. A smaller C emphasizes a larger margin at the cost of allowing some misclassifications on the training data.
Page 211

3 In neural networks, what do we mean by mini-batch and epoch?

A mini-batch is a small, random subset of the training data used to compute the gradient and update the model parameters during one iteration of stochastic gradient descent. It balances computational efficiency and accuracy. An epoch is one complete pass through the entire training dataset. In the mini-batch approach, an epoch consists of multiple iterations, each processing a mini-batch. Pages 124-125

2 KERNEL METHODS

We are implementing a kernel method to predict the hourly temperature of a set of coordinates in Sweden, for a given time and date, based on historic temperature records. For this, we are going to use three different kernels which are based on:

1. the haversine distance ($d_{spatial}$), between the given coordinates and the coordinates of the historic records.
2. the distance in day time (d_{time}) between the given time and the time of the historic record, by also accounting for the cyclical nature of the clock (00:00 and 23:00 are close to each other, not far).
3. the date distance (d_{date}) between the given date and the historic records.

After computing the three different types of distances, we apply the gaussian kernel function with respect to an appropriate smoothing coefficient for the metric.

$$K(d_{var,i}) = \exp\left(-\frac{d_{var,i}^2}{2h_{var}^2}\right), var \in (spatial, time, date)$$

where i denotes the i-th data point and and h_{var} is the smoothing coefficient for the variable var.

```r
kernel_function <- function(distances, h) {
  weights <- lapply(distances, function(distance)
    exp(-(distance ^ 2 / (2 * h ^ 2))))
  return(unlist(weights))
}
```

where $d_{var,j}$ is the distance value of the i-th data point w.r.t to the variable var and h_{var} the smoothing coefficient for variable var. This gives us the kernel value $K(d_{var,i})$ which acts as a weight of data point i w.r.t var.

We then have to aggregate the three different $K(d_{var,i})$ values.

For experiment purposes, we do this in two different fashions, by either summing, or multiplying them. If summed:

$$w_i = K(d_{spatial,i}) + K(d_{time,i}) + K(d_{date,i})$$

If multiplied:

$$w_i = K(d_{spatial,i}) * K(d_{time,i}) * K(d_{date,i})$$

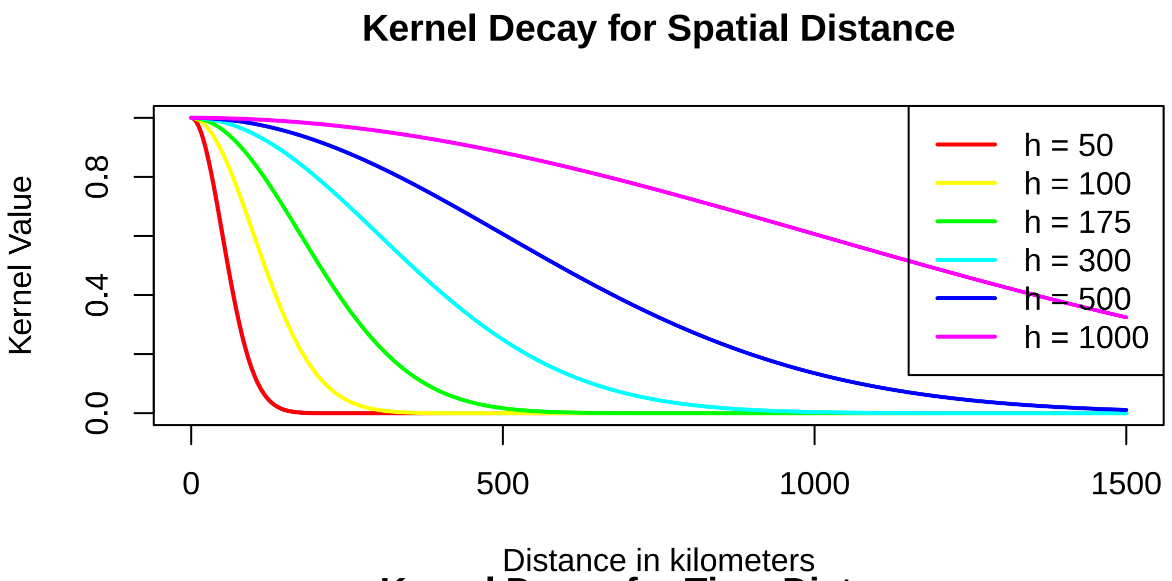
```r
if (weight_aggregation == "sum") {
    aggregated_weights <- spatial_weights + date_weights + time_weights
  } else if (weight_aggregation == "multiply") {
    aggregated_weights <- spatial_weights * date_weights * time_weights
  }
```

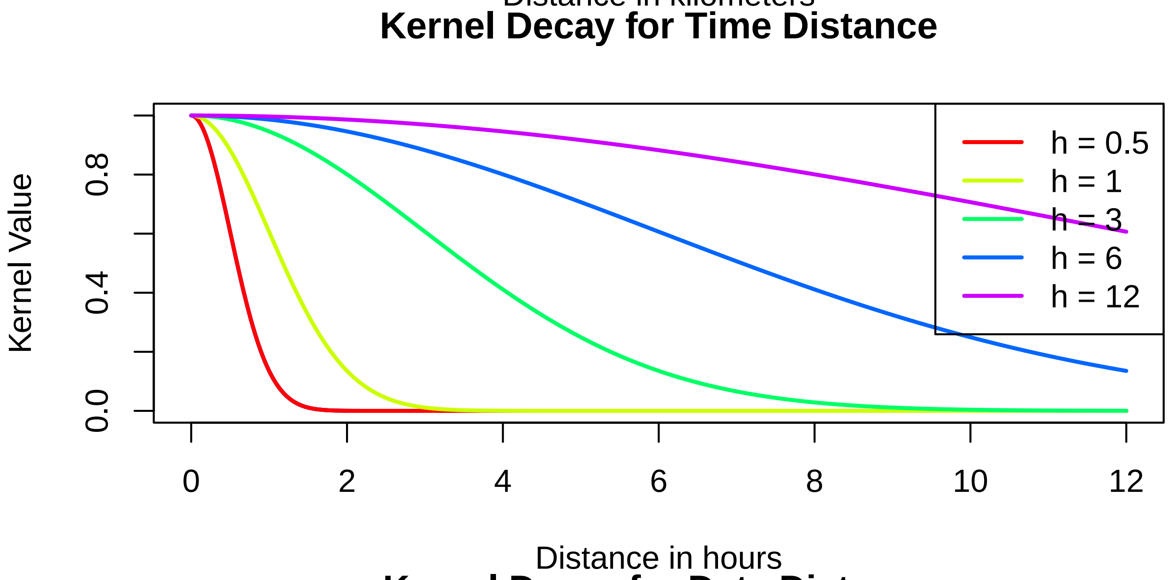
This aggregated weight value determines the weight of the data point in the prediction as shown below.

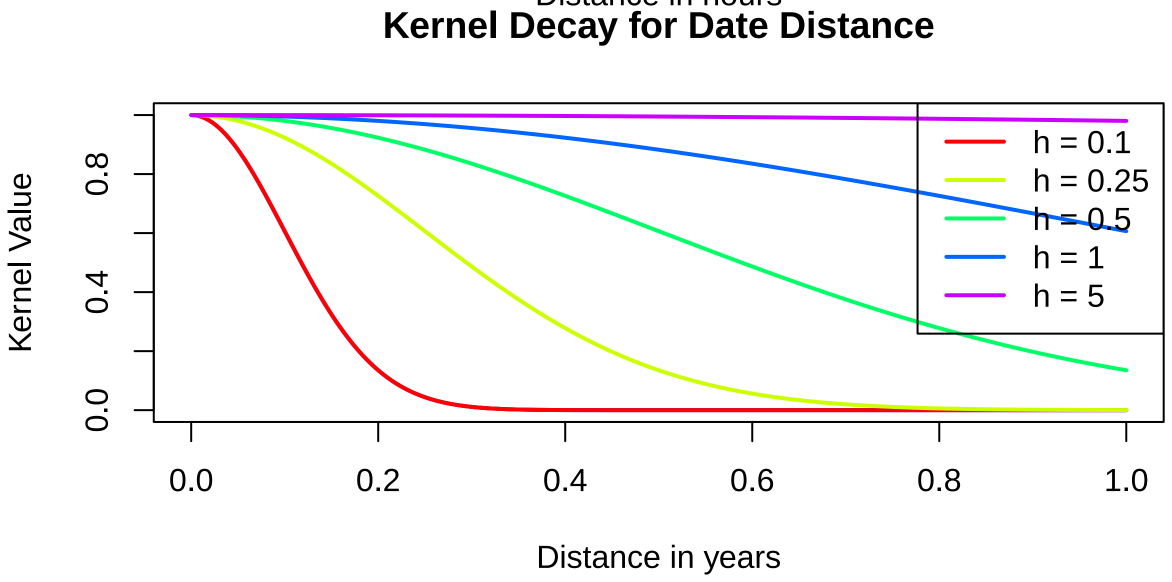
$$\hat{y} = \frac{\sum_i^n w_i * y_i}{\sum_i^n w_i}$$

```r
y_hat = sum(aggregated_weights * y) / sum(aggregated_weights)
```

Now that we have established the functionality of the kernel method, we still need to determine appropriate smoothing coefficients. We do this by plotting how the kernel value decays given a set of smoothing coefficients for every variable. *Note:* If the distance value of a given point is equal to the smoothing coefficient, the kernel value for that point will be 0.5.







We ended up using $h_{distance} = 175$ (kilometers) as that is around 10% of the maximum distance possible between two points in Sweden and approximately the straight line distance between Linköping and Stockholm. This means that when we predict the weather in Stockholm, a weather recording in Linköping will have roughly 50% of the weight of a weather recording in Stockholm (if we disregard date and time).

We went with $h_{time} = 3$ (hours), as this somewhat divides a day into quarters (morning, noon, evening, night).

And with $h_{date} = 1/12$ (years) as this corresponds to a month. This means all data points that are over a year old, will be heavily decayed and have almost no influence in the prediction. However, this was necessary, otherwise we would completely lose the seasonal aspect of the weather.

It would make more sense to further break down the date variable to account for seasonal variability and global warming, instead of using it the way we are doing.

Results

We chose ten random weather stations in Sweden and chose ten random points in time, for which we could find temperature data and compared the real life data to our predictions.

Table 1: Predictions with Kernel Sum

Location	Date	Time	Predicted_Temperature	Actual_Temperature	Deviation
Bromma Airport	1999-01-08	05:00:00	3.93	-5	-8.93
Linköping Airport	2012-03-13	06:00:00	4.17	5	0.83
Arlanda Airport	2001-07-13	04:00:00	3.85	13.9	10.05
Gothenburg Airport	2004-09-04	20:00:00	4.56	15	10.44
Luleå Airport	2014-03-19	02:00:00	2.98	-11	-13.98
Malmö Airport	2011-05-29	08:00:00	5.15	12.2	7.05
Västerås Airport	2010-01-25	00:00:00	3.17	-8.9	-12.07
Dalarna Airport	2015-02-03	02:00:00	3.11	-7.8	-10.91
Kiruna Airport	2012-12-31	08:00:00	4.76	-3.9	-8.66
Jönköping Airport	2011-09-09	10:00:00	5.94	15.5	9.56
Average absolute deviation	NA	NA	NA	NA	9.25

Table 2: Predictions with Kernel Multiply

Location	Date	Time	Predicted_Temperature	Actual_Temperature	Deviation
Bromma Airport	1999-01-08	05:00:00	-1.18	-5	-3.82
Linköping Airport	2012-03-13	06:00:00	-0.01	5	5.01
Arlanda Airport	2001-07-13	04:00:00	12.31	13.9	1.59

Location	Date	Time	Predicted_Temperature	Actual_Temperature	Deviation
Gothenburg Airport	2004-09-04	20:00:00	16.63	15	-1.63
Luleå Airport	2014-03-19	02:00:00	-2.69	-11	-8.31
Malmö Airport	2011-05-29	08:00:00	7.16	12.2	5.04
Västerås Airport	2010-01-25	00:00:00	-3.49	-8.9	-5.41
Dalarna Airport	2015-02-03	02:00:00	-10.66	-7.8	2.86
Kiruna Airport	2012-12-31	08:00:00	-9.38	-3.9	5.48
Jönköping Airport	2011-09-09	10:00:00	16.14	15.5	-0.64
Average absolute deviation	NA	NA	NA	NA	3.98

From the results above we can draw two conclusions: The method which leverages the summation is considerably too "cautious", meaning it's predictions have a very small range in which they fall, which corresponds roughly to the yearly mean temperature in Sweden.

The method which leverages multiplication performs significantly better, which we can attribute to the fact, that a very low kernel score, for one variable, has a larger effect on the weight of the data point.

Example: If we want to predict the weather in Linköping for the 1st of January 2015 at 08:00 o'clock and have a data point in Linköping on the 31st of August 2004, the kernel values might look as follows: $K_{spatial} = 1$, $K_{date} = 0$, $K_{time} = 1$. The aggregated kernel value / weight of the data point would be $w_{sum} = 2$, $w_{multiply} = 0$. Clearly, this data point would not provide good predictive capabilities for predicting the weather in August, however it would still have a significant weight when using summation, while having 0 weight when using multiplication.

3 SUPPORT VECTOR MACHINES

```r
# Calculate the best C
best_C <- which.min(err_va) * by
cat("The best C is:", best_C)
```

```
## The best C is: 1.8
```

For the next part we keep the best C for all filters.

```
## err0 is: 0.165
```

```
## err1 is: 0.1672909
```

```
## err2 is is: 0.1498127
```

```
## err3 is is: 0.01373283
```

Filter 0 : Verify how well the SVM model with the best C performs on the validation set. err0 helps validate the tuning of C.

Filter 1 : Measure the generalization error of the model on unseen data (test set). err1 shows into how well the model performs outside the validation set.

Filter 2 : By using more data (training + validation), the model has a potentially stronger fit. err2 tests whether this improved training size leads to better generalization on the test set.

Filter 3 : Train the final model using all available data, maximizing the information for training. Evaluate on the test set to observe the generalization error.

Questions

1. Which filter do we return to the user ? filter0, filter1, filter2 or filter3? Why? The filter returned to the user is filter3 because it is trained on the entire dataset, including training, validation, and test portions, allowing it to utilize all available information and achieve the lowest error. In the context of this exercise, the goal is to perform SVM model selection using only the given spam dataset, and no truly unseen data is expected. While training on all data may risk overfitting in other scenarios, it is acceptable here as the focus is solely on optimizing performance within the provided dataset.
2. What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why? The estimate of the generalization error of the filter returned to the user is err3, err3 = 0.008739076. Since this filter is trained on the most data and uses the best C from cross-validation, it is expected to generalize better than Filter 0, Filter 1, or Filter 2.

err0: Measures error on the validation set (va), which was only used to tune C. It does not estimate the model's performance on unseen data. err1: Measures error on the test set (te), but the model is trained only on 3000 training samples, not the full dataset. err2: Measures error on the test set (te), but the model is trained on 3800 samples (trva), which is larger than filter1, but still smaller than the entire dataset. err3: Measures error on the test set (te), and since filter3 is trained on all 4601 samples, this is the best and most reliable model for generalization.

err0 = 0.175, err1 = 0.1610487, err2 = 0.1573034, err3 = 0.008739076. The err3 value is much smaller than the other errors, but this is due to the fact that it is trained on all the data, including the test set (te). While this gives it a lower error, it might not reflect "true" generalization error, but it is still the error reported for Filter 3, which is the model returned to the user.

3. SVM Decision Function

The SVM decision function is defined as:

$$f(x) = \sum_{i=1}^{N} \alpha_i K(x_i, x) + b$$

Where:

- α_i are the coefficients for the support vectors,
- $K(x_i, x)$ is the kernel function,
- b is the intercept term.

The **RBF kernel** is given by:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

Here: - $\|x - x'\|^2$ is the squared Euclidean distance between two points x and x', - σ is the kernel width parameter

```r
# Extract support vector indices, coefficients, and intercept
sv <- alphaindex(filter3)[[1]] # Support vector indices
co <- coef(filter3)[[1]]       # Coefficients for the support vectors
inte <- -b(filter3)            # Negative intercept
```

```r
# Initialize a vector to store the decision values
k <- NULL

# Compute decision values for the first 10 points
for (i in 1:10) {
  k2 <- NULL # Initialize contributions for the current data point
  for (j in 1:length(sv)) {
    # Compute the RBF kernel value between point i and support vector j
    diff <- as.numeric(spam[i, -58] - spam[sv[j], -58]) # Difference between features
    rbf_kernel <- exp(-sum(diff^2) / (2 * 0.05^2))      # RBF kernel computation
    k2 <- c(k2, co[j] * rbf_kernel)                    # Contribution from support vector j
  }
  # Compute final decision value for data point i
  k <- c(k, sum(k2) + inte)
}

# Output the computed decision values
k
```

```
##  [1]  0.006292512  0.457293502  0.854669625 -0.730198576 -0.860385532
##  [6]  0.404159399 -1.793707488 -0.674704164  0.576064455 -0.964727603
```

```r
# Compare the computed decision values with predict()
predict(filter3, spam[1:10, -58], type = "decision")
```

```
##             [,1]
##  [1,] -1.0702965
##  [2,]  1.0003450
##  [3,]  0.9995908
##  [4,] -0.9999648
##  [5,] -0.9995379
##  [6,]  1.0000612
##  [7,] -0.8585873
##  [8,] -0.9997047
##  [9,]  0.9998209
## [10,] -1.0000973
```

Positive value: The point is classified as belonging to the positive class. Negative value: The point is classified as belonging to the negative class.

The manually computed decision values are close but not identical to the predict() results due to: Floating-point precision differences. Implementation-specific optimizations in kernlab::ksvm.

The values from predict() have a higher magnitude compared to the manually computed values. This suggests that predict() incorporates additional numerical optimizations or higher precision, leading to more confident predictions.

The manually computed values demonstrate the underlying mechanics of how SVM predictions are made. The classification results are consistent with the predictions from predict(), validating the correctness of your implementation.

4 NEURAL NETWORKS

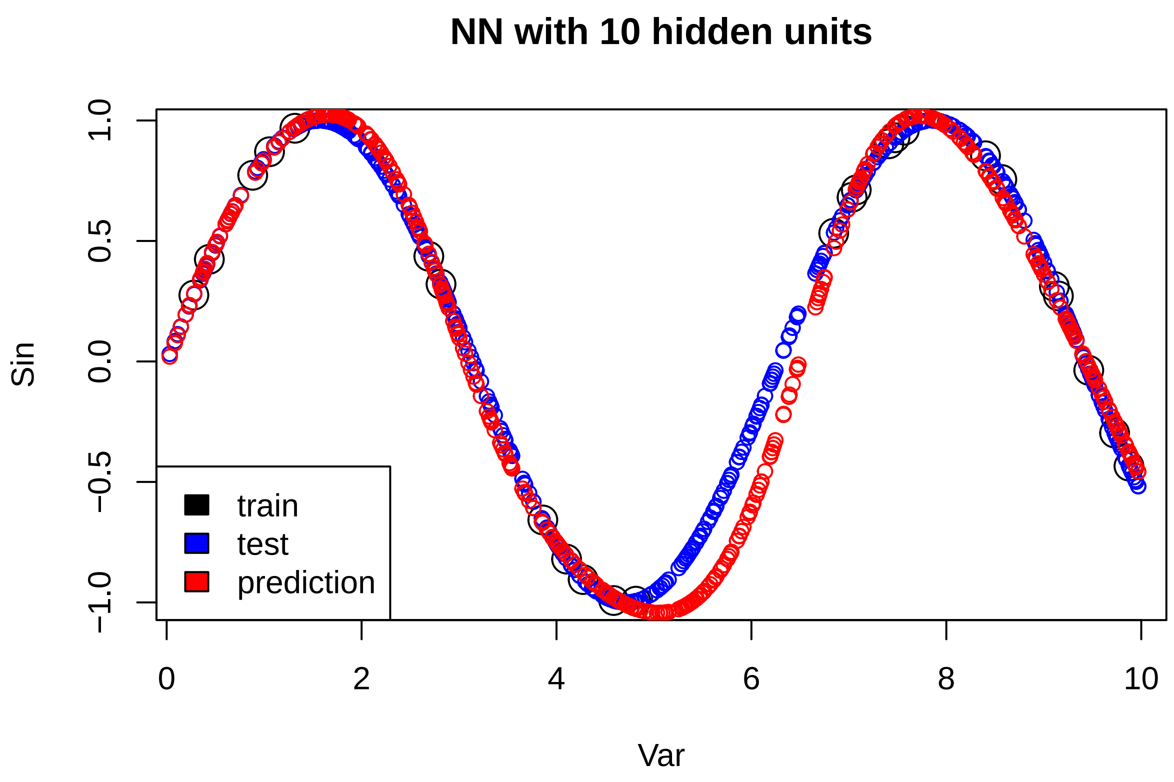
Task 1

We used the given code template to create the dataset containing 500 points in the interval [0,10], to split it in training and test data and to train the neural network with 10 hidden units. For the start weights we sampled ten random values between -1 and 1.

```r
set.seed(1234567890)
# create data
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
# split data
train <- mydata[1:25,]
test <- mydata[26:500,]
# Random initialization of the weights in the interval [-1, 1]
set.seed(1234567890)
winit <- runif(10,-1,1)

# train data with one hidden layer and 10 hidden units
nn <- neuralnet(Sin~Var, train, hidden=10, startweights=winit)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train, cex=2)
points(test, col = "blue", cex=1)
points(test[,1],predict(nn,test), col="red", cex=1)
title("NN with 10 hidden units")
legend(x="bottomleft", legend=c("train", "test","prediction"), fill=c("black","blue", "red"))
```



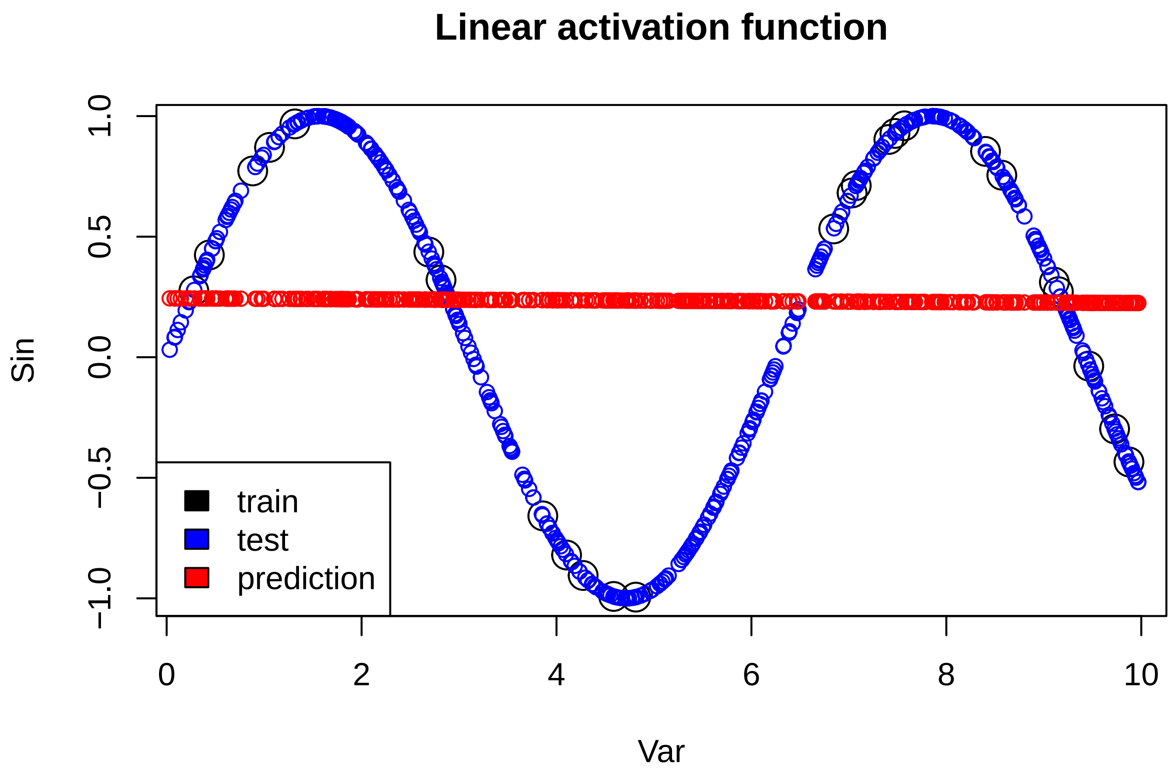
The overall prediction quality is good. Between approximately Var = 5 and Var = 6.5 it is a little bit off though. This might be explained by the missing training data in this interval.

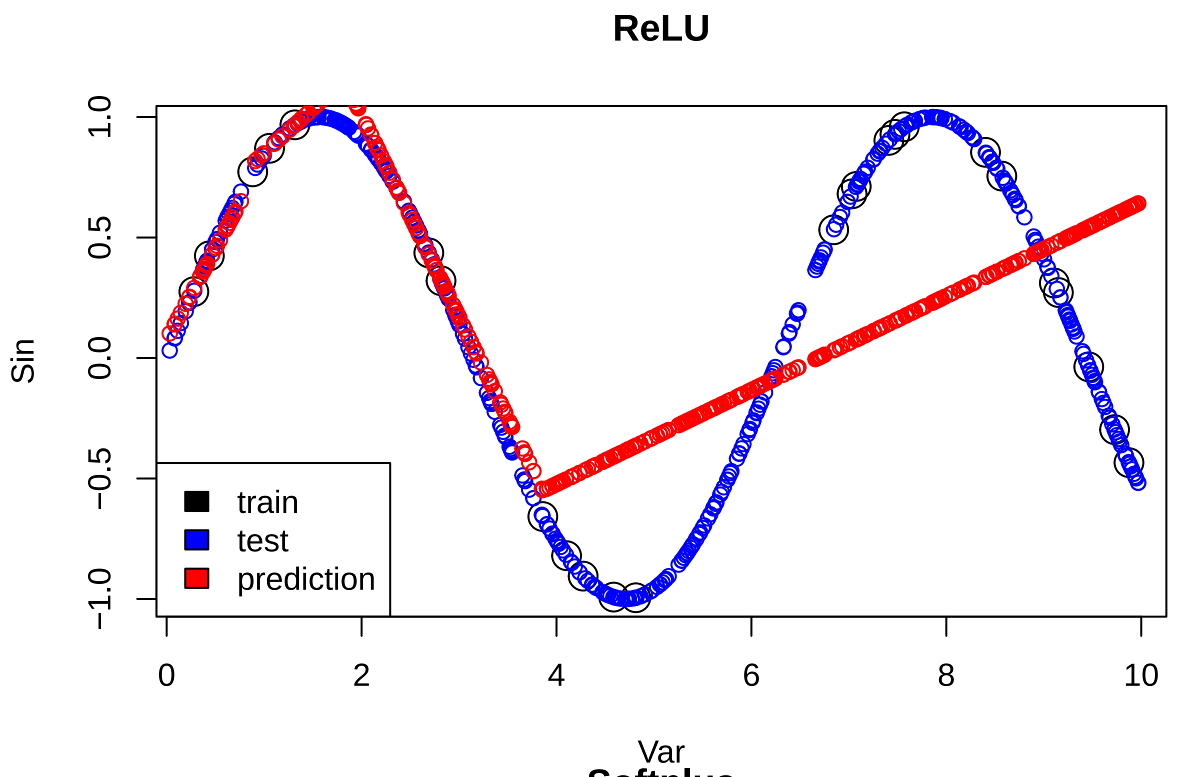
Task 2

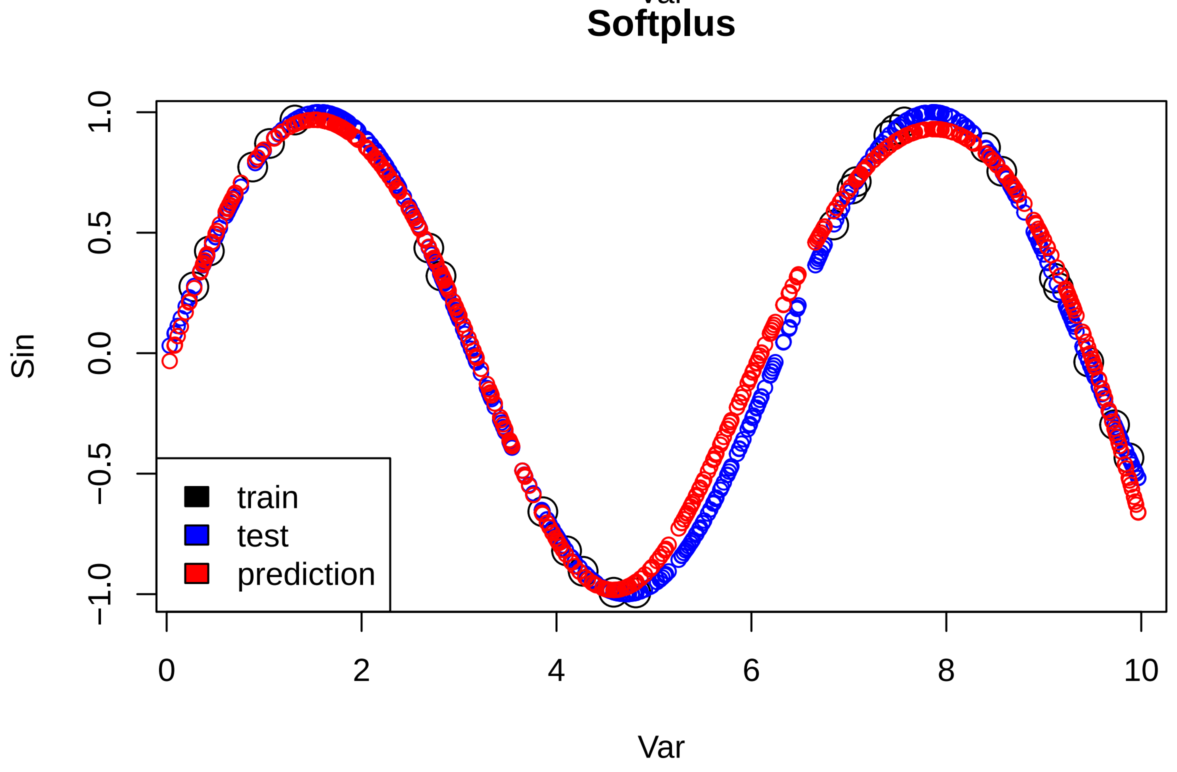
The linear activation function, ReLU and softplus are implemented and used for three different neural nets. For ReLU we used the "ifelse" construct , as the max()-function is not differentiable and causes an error in the training.

```r
h1 <- function(x) x
h2 <- function(x) ifelse(x > 0, x, 0)
h3 <- function(x) log(1+exp(x))

nn_h1 <- neuralnet(Sin~Var, train, hidden=10, act.fct=h1, startweights=winit)
nn_h2 <- neuralnet(Sin~Var, train, hidden=10, act.fct=h2, startweights=winit)
nn_h3 <- neuralnet(Sin~Var, train, hidden=10, act.fct=h3, startweights=winit)
```







For the linear activation function the underlying pattern is not learned at all. This is not surprising as it does not add any non-linearity.

Using ReLU as activation function results in a slightly better prediction. But from Var=4 on it is just a linear line. Reasons for that might be that ReLU is still piece wise linear and the model does not have enough neurons to adjust to the non-linear sinus function.

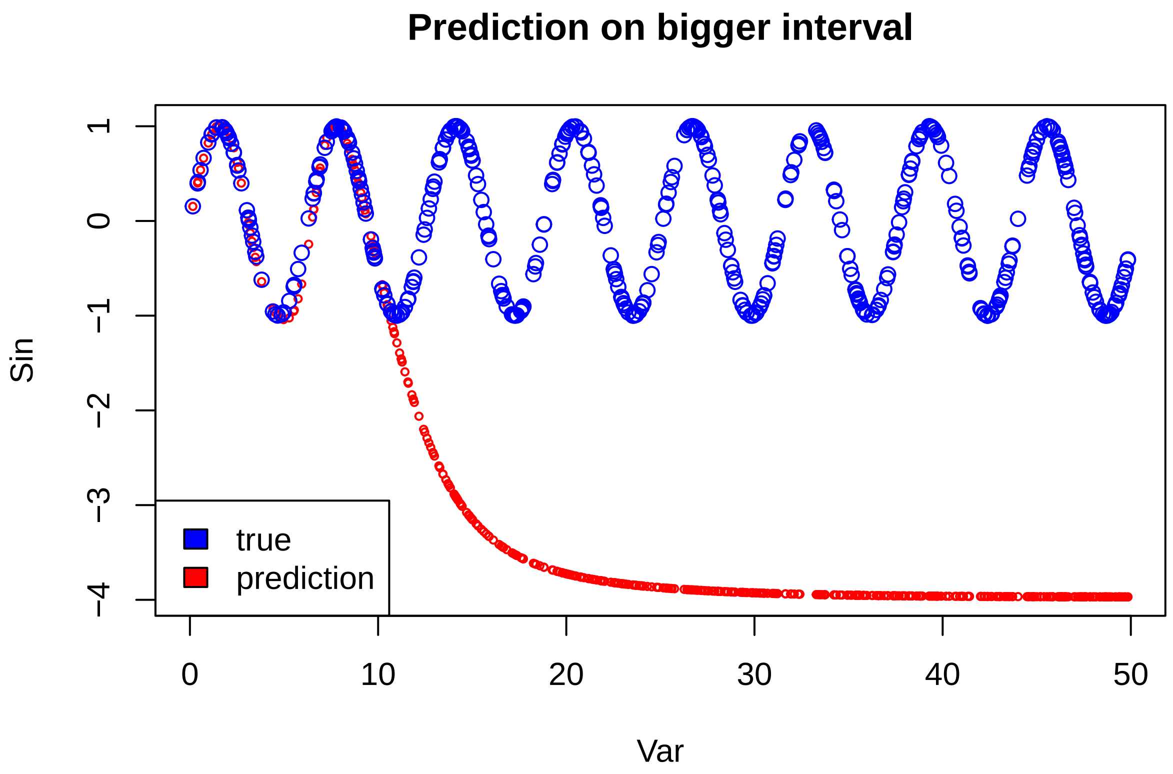
The results of the neural net with softplus are similar to the net with sigmoid. But the interval where the predictions are a bit further from the actual values is a bit bigger for softplus, roughly from Var = 5 to Var = 8.5.

Task 3

For this task we need a new dataset with 500 points, but in the range from 0 to 50.

```
# create new data
set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata_3 <- data.frame(Var, Sin=sin(Var))
```

Then the neural network from task 1 is used to predict the sinus on the new data.



In the interval, that the net was trained for, the prediction is very good. But for Var > 10 the prediction converges towards -3.97.

```
## Minimum value of prediction: -3.969336
```

Task 4

As there is no training data for values bigger than 10, the prediction fails for these. But why it converges to this exact value is not clear. We cannot find any indicator in the weights:

```
## Weights for input to hidden layer:
##  11.513 4.272 -0.8795 -0.8276 5.5674 0.2863 0.2569 0.9651 -0.5697 1.6987

## Weights for bias to hidden layer:
##  -1.7517 -0.4719 -0.1691 -0.8734 -1.9694 0.1537 0.8253 0.1511 -0.1557 -0.6942

## Weights for output layer:
##  0.5203 -4.7498 6.6557 2.9049 -6.7634 2.3065 -3.5924 0.6954 -1.5948 0.7492 -1.0623
```

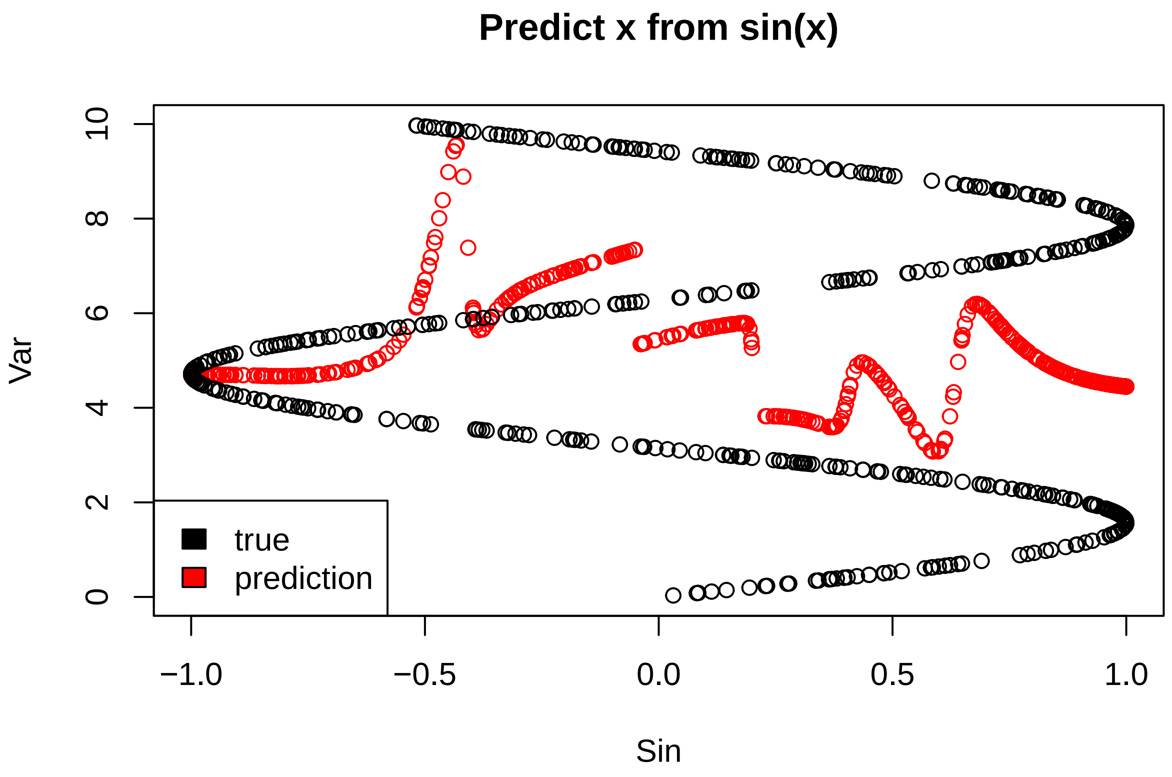
As the hidden layers are “black boxes” that cannot be explained by humans in most cases, this is not surprising.

Task 5

The same procedure as for the previous tasks is used here as well. Only the target and feature variables are switched to predict x from sin(x).

```r
set.seed(1234567890)
Var <- runif(500,0,10)
mydata_5 <- data.frame(Var, Sin=sin(Var))

nn5 <- neuralnet(Var~Sin, mydata_5, hidden=10, threshold=0.1, startweights=winit)
```



The results are very bad. The predictions are not close to the actual value at all, which makes sense as many x values result in the same sin(x).

Appendix

```r
## Assignment 2

set.seed(1234567890)
library(geosphere)

# Data preprocessing
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations, temps, by = "station_number")
# st$coords <- apply(st, 1, function(row) list(c(row['longitude'], row['latitude'])))
st <- subset(st, select = c(latitude, longitude, date, time, air_temperature))
```

```r
st$date <- as.Date(st$date)
st$time <- strptime(st$time, format = "%H:%M:%S")


# Decay values
h_distance <- 50
h_date <- 1/12
h_time <- 3

calc_spatial_distance <- function(coords_matrix, coords_target) {
  # Calculate absolute distance in km
  distances = apply(coords_matrix, 1, function(coord)
    distHaversine(coord, coords_target))
  abs_km_distances = abs(distances) / 1000
  return (abs_km_distances)
}

calc_date_distances <- function(date_list, date_target) {
  # Calculate absolute distance in years
  distances = lapply(date_list, function(date)
    difftime(date, date_target, units = "days"))
  num_abs_distances = abs(as.numeric(distances) / 365)
  return(num_abs_distances)
}

calc_time_distances <- function(time_list, time_target) {
  # Calculate absolute distance accounting for 24-hour day cycle (23 o'clock and 1 o'clock are close no
  distances = lapply(time_list, function(time)
    min(abs(
      difftime(time, time_target, units = "hours")
    ), 24 - abs(
      difftime(time, time_target, units = "hours")
    )))
  num_distances = as.numeric(distances)
  return(num_distances)
}

kernel_function <- function(distances, h) {
  weights <- lapply(distances, function(distance)
    exp(-(distance ^ 2 / (2 * h ^ 2))))
  return(unlist(weights))
}


predict_temp <- function(data,
                         coords,
                         date,
                         time,
                         weight_aggregation) {
 print(coords)
  data <- data[data$date < date, ] # filter for only prior data points
  coords_matrix <- cbind(data$latitude, data$longitude) # combine coordinates
  y <- data$air_temperature # extract target variable
  # Calculate distances & weights
  spatial_distances <- calc_spatial_distance(coords_matrix, coords)
  date_distances <- calc_date_distances(data$date, date)
  time_distances <- calc_time_distances(data$time, time)
  spatial_weights <- kernel_function(spatial_distances, h_distance)
  date_weights <- kernel_function(date_distances, h_date)
  time_weights <- kernel_function(time_distances, h_time)

  aggregated_weights <- NULL
  if (weight_aggregation == "sum") {
    aggregated_weights <- spatial_weights + date_weights + time_weights
  } else if (weight_aggregation == "multiply") {
    aggregated_weights <- spatial_weights * date_weights * time_weights
  }
  return(sum(aggregated_weights * y) / sum(aggregated_weights))
}
# Function to plot kernel decay
plot_kernel_decay <- function(min_value,
                              max_value,
                              h_values,
                              units,
                              title = "Spatial Distance") {
  # Define kernel function
  kernel_function <- function(distance, h) {
    exp(-(distance ^ 2) / (2 * h ^ 2))
  }

  # Generate distances based on the provided range
  distances <- seq(min_value, max_value, length.out = 500)

  # Plot setup
  plot(
    distances,
    kernel_function(distances, h_values[1]),
    type = "l",
    lwd = 2,
    col = "blue",
    ylim = c(0, 1),
    xlab = paste("Distance in", units),
    ylab = "Kernel Value",
    main = title
  )

  # Add additional curves for other h values
  if (length(h_values) > 1) {
    colors <- rainbow(length(h_values))
    for (i in seq_along(h_values)) {
      lines(
        distances,
        kernel_function(distances, h_values[i]),
        col = colors[i],
        lwd = 2
      )
    }
    legend(
      "topright",
      legend = paste0("h = ", h_values),
      col = colors,
      lwd = 2
    )
  } else {
    legend(
      "topright",
      legend = paste0("h = ", h_values[1]),
      col = "blue",
      lwd = 2
    )
  }
}

# Example Usage
plot_kernel_decay(
  min_value = 0,
  max_value = 1500,
  h_values = c(50, 100, 175, 300, 500, 1000),
  # Different h values to compare
  units = "kilometers",
  title = "Kernel Decay for Spatial Distance"
)

plot_kernel_decay(
  min_value = 0,
  max_value = 12,
  h_values = c(0.5, 1, 3, 6, 12),
  # Different h values to compare
  units = "hours",
  title = "Kernel Decay for Time Distance"
)

plot_kernel_decay(
  min_value = 0,
  max_value = 1,
  h_values = c(0.1, 0.25, 0.5, 1, 5),
  # Different h values to compare
  units = "years",
  title = "Kernel Decay for Date Distance"
)




dates = c('1999-01-08', # Bromma
         '2012-03-13', # Linköping
         '2001-07-13', # Arlanda
         '2004-09-04', # Gothenburg
         '2014-03-19', # Luleå
         '2011-05-29', # Malmö
         '2010-01-25', # Västerås
         '2015-02-03', # Dalarna
         '2012-12-31', # Kiruna
         '2011-09-09') # Jönköping

coordinates <- list(
  c(59.3544, 17.9415),  # Bromma Airport
  c(58.4062, 15.6805),  # Linköping Airport
  c(59.6519, 17.9186),  # Arlanda Airport
  c(57.6666, 12.2878),  # Gothenburg Airport
  c(65.5490, 22.1232),   # Luleå
  c(55.5366, 13.376),   # Malmö Airport
  c(59.5894, 16.6336),  # Västerås Airport
  c(60.4336, 15.5018),  # Dalarna Airport
  c(67.8210, 20.3368),  # Kiruna Airport
  c(57.7576, 14.0687)   # Jönköping Airport
)

locations <- c(
  "Bromma Airport",
  "Linköping Airport",
  "Arlanda Airport",
  "Gothenburg Airport",
  "Luleå Airport",
  "Malmö Airport",
  "Västerås Airport",
  "Dalarna Airport",
  "Kiruna Airport",
  "Jönköping Airport"
)

times <- c("05:00:00", "06:00:00", "04:00:00", "20:00:00", "02:00:00",
                   "08:00:00", "00:00:00", "02:00:00", "08:00:00", "10:00:00")

actual_temperatures <- c(-5, 5, 13.9, 15, -11, 12.2, -8.9, -7.8, -3.9, 15.5)

# Initialize a data frame to store results
results <- data.frame(
  Location = character(),
  Date = character(),
  Time = character(),
  Predicted_Temperature = numeric(),
  Actual_Temperature = numeric(),
  Deviation = numeric(),
  stringsAsFactors = FALSE
)

# Loop to calculate predictions and store results
for (i in seq_along(times)) {
  # Convert date and time
  date <- as.Date(dates[i])
  time <- strptime(times[i], format = "%H:%M:%S", tz = "UTC")
  # Validate coordinate length
  if (length(coordinates[[i]]) != 2) {
    stop(sprintf("Coordinates for %s are invalid: %s", locations[i], coordinates[[i]]))
  }

  # Predict temperature
  y_pred <- predict_temp(
    data = st,
    coords = coordinates[[i]],  # Use [[i]] to extract the vector
    time = time,
    date = date,
    weight_aggregation = "sum"
  )

  # Calculate deviation
  deviation <- actual_temperatures[i] - y_pred

  # Append to results data frame
  results <- rbind(results, data.frame(
    Location = locations[i],
    Date = dates[i],
    Time = times[i],
    Predicted_Temperature = round(y_pred, 2),
    Actual_Temperature = actual_temperatures[i],
    Deviation = round(deviation, 2),
    stringsAsFactors = FALSE
  ))
}

# Save the results to an RDS file for easier loading in R
write.csv(results, "predictions_kernel_sum.csv", row.names = FALSE)


## Assignment 3


library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
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=FALSE)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}


best_C <- which.min(err_va) * by
best_C

filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALS
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
cat("err0 is:", err0, "\n")

filter1 <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALS
mailtype <- predict(filter1,te[,-58])
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
cat("err1 is:", err1, "\n")

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)
cat("err2 is is:", err2, "\n")

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)
cat("err3 is is:", err3, "\n")




# 3

# Extract support vector indices, coefficients, and intercept
sv <- alphaindex(filter3)[[1]] # Support vector indices
co <- coef(filter3)[[1]]       # Coefficients for the support vectors
inte <- -b(filter3)            # Negative intercept

# Initialize a vector to store the decision values
k <- NULL

# Compute decision values for the first 10 points
for (i in 1:10) {
  k2 <- NULL # Initialize contributions for the current data point
  for (j in 1:length(sv)) {
    # Compute the RBF kernel value between point i and support vector j
    diff <- as.numeric(spam[i, -58] - spam[sv[j], -58]) # Difference between features
    rbf_kernel <- exp(-sum(diff^2) / (2 * 0.05^2))      # RBF kernel computation
    k2 <- c(k2, co[j] * rbf_kernel)                    # Contribution from support vector j
  }
  # Compute final decision value for data point i
```

```r
  k <- c(k, sum(k2) + inte)
}

# Output the computed decision values
k

# Compare the computed decision values with predict()
predict(filter3, spam[1:10, -58], type = "decision")


## Assignment 4


library("neuralnet")

### 1.

set.seed(1234567890)
# create data
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin=sin(Var))
# split data
train <- mydata[1:25,]
test <- mydata[26:500,]
# Random initialization of the weights in the interval [-1, 1]
set.seed(1234567890)
winit <- runif(10,-1,1)

# train data with one hidden layer and 10 hidden units
nn <- neuralnet(Sin~Var, train, hidden=10, startweights=winit)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train, cex=2)
points(test, col = "blue", cex=1)
points(test[,1],predict(nn,test), col="red", cex=1)
title("NN with 10 hidden units")
legend(x="bottomleft", legend=c("train", "test","prediction"), fill=c("black","blue", "red"))


### 2.

h1 <- function(x) x
h2 <- function(x) ifelse(x > 0, x, 0)
h3 <- function(x) log(1+exp(x))

nn_h1 <- neuralnet(Sin~Var, train, hidden=10, act.fct=h1, startweights=winit)
nn_h2 <- neuralnet(Sin~Var, train, hidden=10, act.fct=h2, startweights=winit)
nn_h3 <- neuralnet(Sin~Var, train, hidden=10, act.fct=h3, startweights=winit)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train, cex=2)
points(test, col = "blue", cex=1)
points(test[,1],predict(nn_h1,test), col="red", cex=1)
```

```r
title("Linear activation function")
legend(x="bottomleft", legend=c("train", "test","prediction"), fill=c("black","blue", "red"))
# Plot of the training data (black), test data (blue), and predictions (red)
plot(train, cex=2)
points(test, col = "blue", cex=1)
points(test[,1],predict(nn_h2,test), col="red", cex=1)
title("ReLU")
legend(x="bottomleft", legend=c("train", "test","prediction"), fill=c("black","blue", "red"))

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train, cex=2)
points(test, col = "blue", cex=1)
points(test[,1],predict(nn_h3,test), col="red", cex=1)
title("Softplus")
legend(x="bottomleft", legend=c("train", "test","prediction"), fill=c("black","blue", "red"))

### 3.

# create new data
set.seed(1234567890)
Var <- runif(500, 0, 50)
mydata_3 <- data.frame(Var, Sin=sin(Var))

# Plot of the actual data (blue) and the prediction

plot(mydata_3[,1],predict(nn,newdata=mydata_3),col="red",cex=0.5, xlab="Var", ylab="Sin")
points(mydata_3,cex=1, col="blue")
title("Prediction on bigger interval")
legend(x="bottomleft", legend=c("true","prediction"), fill=c("blue", "red"))

cat("Minimum value of prediction:", min(predict(nn,mydata_3)))

### 5.
set.seed(1234567890)
Var <- runif(500,0,10)
mydata_5 <- data.frame(Var, Sin=sin(Var))

nn5 <- neuralnet(Var~Sin, mydata_5, hidden=10, threshold=0.1, startweights=winit)

plot(mydata_5$Sin,predict(nn5,newdata=mydata_5),col="red",ylab="Var", xlab="Sin", ylim=c(0,10))
points(mydata_5$Sin,mydata_5$Var)
title("Predict x from sin(x)")
legend(x="bottomleft", legend=c("test","prediction"), fill=c("black", "red"))
```