Lab 3

johmo870, nisra674, alebj452

2024-12-18

Statement of Contribution

- Nisal Amashan(nisra674) - Assignment4, Report
- John Möller (johmo870) - Assignment2, Report
- Alexander Björhag (alebj452) - Assignment1, Assignment3, Report

Assignment 1: Theory

What is the kernel trick?

The kernel trick is that we can choose a kernal $k(x, x´)$ rather than designing a d-dimensional transformation $\phi(x)$ since $\phi(x)$ only appears in the model model through inner products like $\phi(x)^T\phi(x´)$ This allows us to work with kernels $k(x, x´)$ instead of chosing $\phi(x)$ Found on page 194 in the book

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 in Support Vector Machines (SVMs) controls the trade-off between minimizing the training error and maximizing the margin for generalization. A larger C places more emphasis on minimizing the training error, which can lead to smaller margins and potential overfitting. Conversely, a smaller C allows for a larger margin by tolerating some misclassifications, which can improve generalization. This trade-off makes C a crucial regularization hyperparameter.

Found on page 211 in the book.

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

A mini-batch is a small subset of the training data used to compute the gradient during training, typically containing 10, 100, or 1,000 data points. An epoch refers to one complete pass through the entire training dataset, which consists of $\frac{n}{n_b}$ iterations where n is the total number of data points and n_b is the size of a mini-batch.

Found on page 125 in the book

Assignment 2: Kernel Methods

Assignment 3: Support Vector Machines

```
## Filter0 error: 0.165
```

```
## Filter1 error: 0.1672909

## Filter2 error: 0.1498127

## Filter3 error: 0.01373283
```

Here is the generalization error rate for each model created from the code we get.

Questions to answer

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

The filter that we should return is filter1 because the other filter models have some things that can be problematic. Before every filter model a for loop was created where we were using the training dataset and the validation set to chose the hyperparameter C. So in filter0 we are reusing the validation set when we evaluate the model again that can give biased error.

In filter3 we are using the entire dataset were all the data sets (Train, validation and test are divided) so we are reusing the validation set when choosing hyperparamter c and when we later evaluate the model were we are reusing the test set. That is the reason we get the best error rate in that model.

Between filter1 and filter2 the error rate is pretty close and filer2 is actually better, but in this model we have combined the training and validation set and the hyperparameter c was tuned with the help of the validation set there is a risk of data leakage even if we in the end have the test data set to evaluate the model.

So that is the reason why filter1 should be chosen. That filter gives the most fair estimate because we use the validation set when tuning the hyperparamter c and later uses the unseen test data set when evaluating the model.

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

Like i mentioned before the filter1 should be return with the generalization error err1 (0.1672909) because that is the most fair were we have used the unseen test set in the end when we evaluate the model and should therefore reflect how well the model perform on "real data".

(3) Once a SVM has been fitted to the training data, a new point is essentially classified according to the sign of a linear combination of the kernel function values between the support vectors and the new point. You are asked to implement this linear combination for filter3. You should make use of the functions alphaindex, coef and b that return the indexes of the support vectors, the linear coefficients for the support vectors, and the negative intercept of the linear combination. See the help file of the kernlab package for more information. You can check if your results are correct by comparing them with the output of the function predict where you set type = "decision". Do so for the first 10 points in the spam dataset. Feel free to use the template provided in the Lab3Block1 2021 SVMs St.R file.

```
rbf_kernal <- rbfdot(0.05) # the radial basis function kernel from the package
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.
  k2<-NULL
  for(j in 1:length(sv)){
    k2<- c(k2, rbf_kernal(as.vector(unlist(spam[i, -58])), as.vector(unlist(spam[sv[j], -58]))))
  }
  k<-c(k, sum(co*k2) + inte)
}
```

After the implementation of some code to the template we can see that we get the same values as the predict function from the model

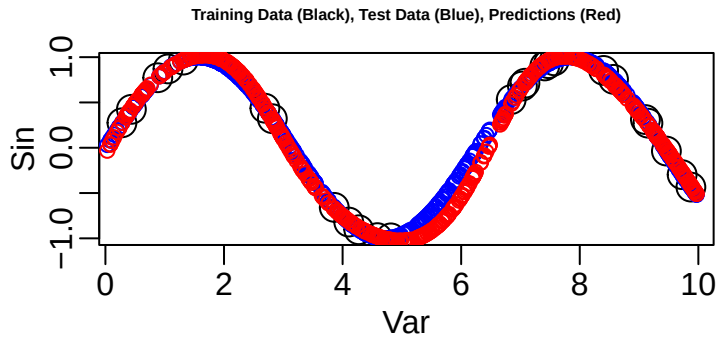
Assignment 4: Neural Networks

(1) Learning the Sine Function

We trained a neural network to learn the sine function using 500 data points sampled uniformly in the interval $[0, 10]$. Out of these, 25 points were used for training, and the rest were used for testing.

Results

- The predictions from the neural network closely matched the sine function on the test data.
- **Plot**: The training data, test data, and neural network predictions are shown in the figure below.



Comments

- The neural network with a single hidden layer containing 10 units performed well in approximating the sine function.
- The smoothness of the sine curve was captured accurately, especially for the test data within the training range.

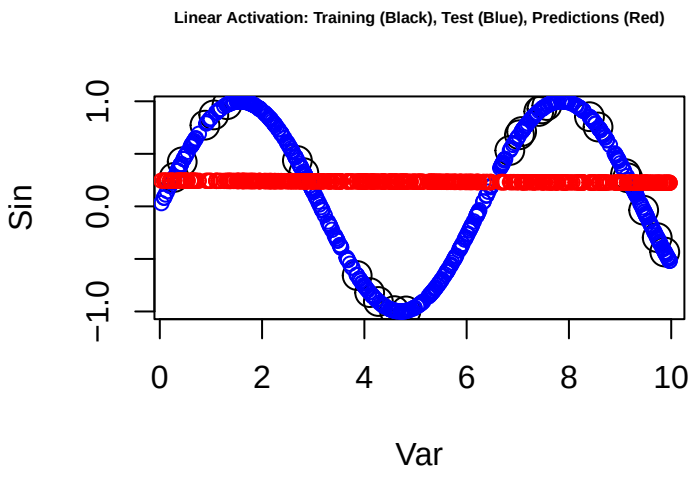
(2) Custom Activation Functions

We repeated the task with three custom activation functions:

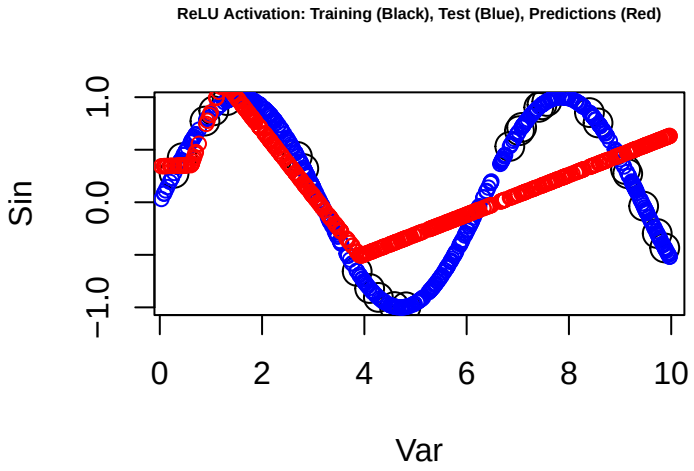
- **Linear Activation**: $h_1(x) = x$
- **ReLU Activation**: $h_2(x) = \max(0, x)$
- **Softplus Activation**: $h_3(x) = \ln(1 + \exp(x))$

Results

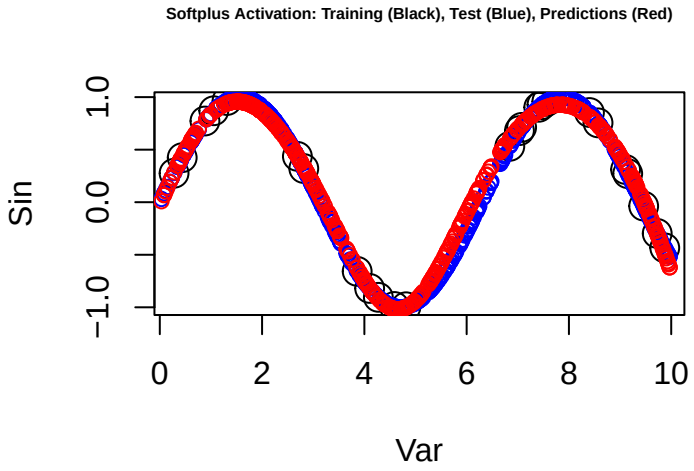
- **Linear Activation**: The neural network predicted a linear trend that deviated from the sine function.


Linear Activation: Training (Black), Test (Blue), Predictions (Red)
Sin
1.0
0.0
−1.0
0
2
4
6
8
10
Var

- **ReLU Activation**: The predictions exhibited sharp changes but failed to approximate the smooth sine curve.



- **Softplus Activation**: The predictions were smoother compared to ReLU but still diverged from the sine function in some regions.



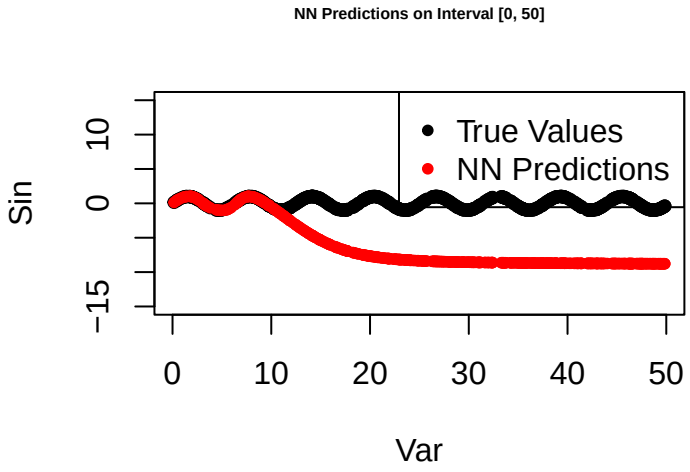
Comments

- The default logistic activation performed better
- **Linear** activation resulted in poor approximation due to its inability to model non-linearity.
- **ReLU** captured some non-linear behavior but introduced sharp transitions.
- **Softplus** performed better

(3) Predictions on a New Interval [0, 50]

Using the neural network trained in part (1), we predicted the sine function values for 500 points sampled uniformly in $[0, 50]$.

Results



Comments

- The predictions did not generalize well outside the original interval $[0, 10]$.
- The neural network failed to capture the periodic behavior of the sine function for larger values of x.
- This is because the network was trained only on a limited range of inputs.

(4) Explanation of Prediction Convergence

The predictions from part (3) converge to a constant value because of how the network's weights influence the output:

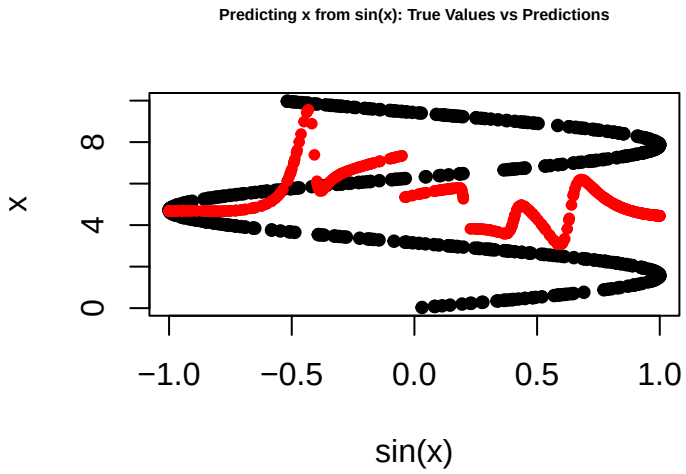
- **Saturation of Hidden Neurons**:
  The large weights between the input and hidden layer cause the hidden neurons to saturate (output close to 0 or 1) when the inputs are outside the training range `[0, 10]`.
- **Bias Term Dominance**:
  With the hidden neurons producing nearly constant outputs, the final predictions are dominated by the bias term and output layer weights, resulting in flattened predictions.

This happens because the network was trained only on the `[0, 10]` range and cannot generalize beyond it.

(5) Predicting x from $\sin(x)$

We trained a neural network to predict x from $\sin(x)$ using 500 points sampled in the interval $[0, 10]$.

Results



Comments

- The neural network performed poorly in this task.
- The sine function is not one-to-one in the interval $[0, 10]$, leading to ambiguity in mapping $\sin(x)$ back to x.
- The convergence error was resolved by setting a higher threshold (threshold $= 0.1$).
- The resulting predictions were inconsistent and scattered.

Appendix

Code for Assignment 1

Code for Assignment 2

Code for Assignment 3

```
# Lab 3 block 1 of 732A99/TDDE01/732A68 Machine Learning
# Author: jose.m.pena@liu.se
# Made for teaching purposes
#install.packages("kernlab")
library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
```

```r
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))
}

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


rbf_kernal <- rbfdot(0.05) # the radial basis function kernel
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.
  k2<-NULL
  for(j in 1:length(sv)){
    k2<- c(k2, rbf_kernal(as.vector(unlist(spam[i, -58])), as.vector(unlist(spam[sv[j], -58]))))
  }
  k<-c(k, sum(co*k2) + inte)
}
k
predict(filter3,spam[1:10,-58], type = "decision")
temp <- data.frame(
  "Manual Calculations" = k,
```

```
  "Model Calculations" = as.vector(predict(filter3, spam[1:10, -58], type = "decision"))
)
```

Code for Assignment 4

```
library(neuralnet)

# --------------------------------
# Question 1: Learning the Sine Function
# --------------------------------

set.seed(1234567890)

# Generate 500 points uniformly in [0, 10]
Var <- runif(500, 0, 10)
mydata <- data.frame(Var, Sin = sin(Var))

# Split data into training and testing sets
tr <- mydata[1:25, ]   # Training: first 25 points
te <- mydata[26:500, ] # Testing: remaining points

# Train neural network with 1 hidden layer and 10 units
nn <- neuralnet(Sin ~ Var,
                data = tr,
                hidden = 10)

# Plot Training and Test Data with Predictions
par(mar = c(3, 3, 2, 1), mgp = c(1.5, 0.5, 0))
plot(tr, cex = 2, main = "Training Data (Black), Test Data (Blue), Predictions (Red)")
points(te, col = "blue", cex = 1)
points(te[, 1], predict(nn, te), col = "red", cex = 1)

# --------------------------------
# Question 2: Custom Activation Functions
# --------------------------------

# Define custom activation functions
softplus <- function(x) log(1 + exp(x))
relu <- function(x) ifelse(x > 0, x, 0)
linear <- function(x) x

# ---- Linear Activation ----
nn_linear <- neuralnet(Sin ~ Var,
                       data = tr,
                       hidden = 10,
                       act.fct = linear)

# Plot Linear Activation Results
plot(tr, cex = 2, main = "Linear Activation: Training (Black), Test (Blue), Predictions (Red)")
points(te, col = "blue", cex = 1)
points(te[, 1], predict(nn_linear, te), col = "red", cex = 1)

# ---- ReLU Activation ----
nn_relu <- neuralnet(Sin ~ Var,
```

```
                    data = tr,
                    hidden = 10,
                    act.fct = relu)

# Plot ReLU Activation Results
plot(tr, cex = 2, main = "ReLU Activation: Training (Black), Test (Blue), Predictions (Red)")
points(te, col = "blue", cex = 1)
points(te[, 1], predict(nn_relu, te), col = "red", cex = 1)

# ---- Softplus Activation ----
nn_softplus <- neuralnet(Sin ~ Var,
                         data = tr,
                         hidden = 10,
                         act.fct = softplus)

# Plot Softplus Activation Results
plot(tr, cex = 2, main = "Softplus Activation: Training (Black), Test (Blue), Predictions (Red)")
points(te, col = "blue", cex = 1)
points(te[, 1], predict(nn_softplus, te), col = "red", cex = 1)

# ------------------------------
# Question 3: Predictions on Interval [0, 50]
# ------------------------------

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

# Predict using the trained NN from Question 1
predictions <- predict(nn, newdata)

# Plot True Values vs NN Predictions
plot(newdata$Var, newdata$Sin,
     col = "black",
     main = "NN Predictions on Interval [0, 50]",
     xlab = "Var",
     ylab = "Sin",
     ylim = c(-15, 15),
     pch = 20, cex = 1.2)

points(newdata$Var, predictions,
       col = "red", pch = 20, cex = 1)

legend("topright", legend = c("True Values", "NN Predictions"),
       col = c("black", "red"), pch = 20)

# ------------------------------
# Question 5: Predicting x from sin(x)
# ------------------------------

# Generate data: sin(x) as input, x as output
set.seed(1234567890)
Var <- runif(500, 0, 10)
Sin <- sin(Var)
data <- data.frame(Sin, Var)
```

```r
# Train a neural network to predict x from sin(x)
nn_inverse <- neuralnet(Var ~ Sin,
                        data = data,
                        hidden = 10,
                        threshold = 0.1)  # Higher threshold to avoid convergence error

# Predict values
predictions_inverse <- predict(nn_inverse, data)

# Plot True x Values vs Predictions
plot(data$Sin, data$Var,
     col = "black",
     main = "Predicting x from sin(x): True Values vs Predictions",
     xlab = "sin(x)",
     ylab = "x",
     pch = 20, cex = 1.2)

points(data$Sin, predictions_inverse,
       col = "red", pch = 20, cex = 1)
```