exercises_2

Bas Machielsen

2022-07-12

Question 1

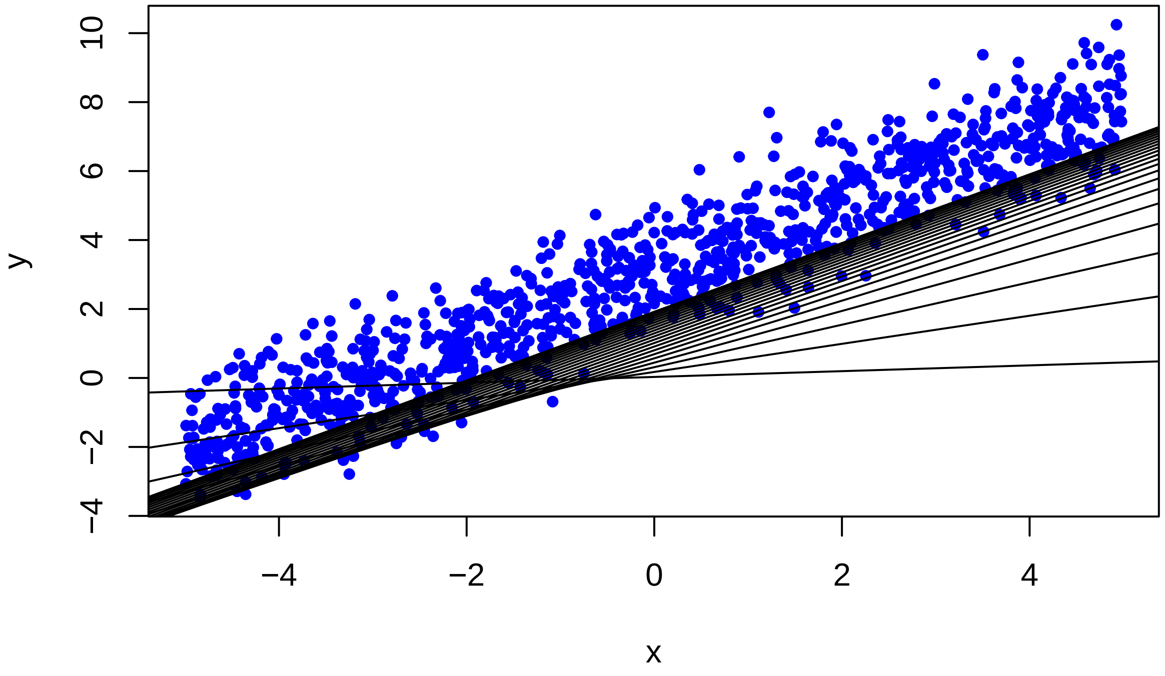
```r
set.seed(55)

x <- runif(1000, -5, 5)
y <- x + rnorm(1000) + 3
```

```r
cost <- function(actual, prediction){

  return(mean((actual-prediction)^2))

}
```

```r
gradient_desc <- function(x, y, learn_rate, conv_threshold, n, max_iter) {
  plot(x, y, col = "blue", pch = 20)
  m <- 0
  c <- 0
  yhat <- m * x + c
  MSE <- sum((y - yhat) ^ 2) / n

  converged = F
  iterations = 0
  while(converged == F) {
    ## Implement the gradient descent algorithm
    m_new <- m - learn_rate * ((1 / n) * (sum((yhat - y) * x)))
    c_new <- c - learn_rate * ((1 / n) * (sum(yhat - y)))
    m <- m_new
    c <- c_new
    yhat <- m * x + c
    MSE_new <- sum((y - yhat) ^ 2) / n

    if(iterations %% 5 == 0){
      abline(c,m)
    }
    if(MSE - MSE_new <= conv_threshold) {
      abline(c, m)
      converged = T
      return(paste("Optimal intercept:", c, "Optimal slope:", m))
    }
    iterations = iterations + 1
    if(iterations > max_iter) {
      abline(c, m)
      converged = T
      return(paste("Optimal intercept:", c, "Optimal slope:", m))
    }
  }
}
```

Set iteration number to 100 and the learning rate to 0.01. Initialize the intercept and slope to zero.

```
gradient_desc(x, y, learn_rate = 0.01, 0.01, 1000, 100)
```

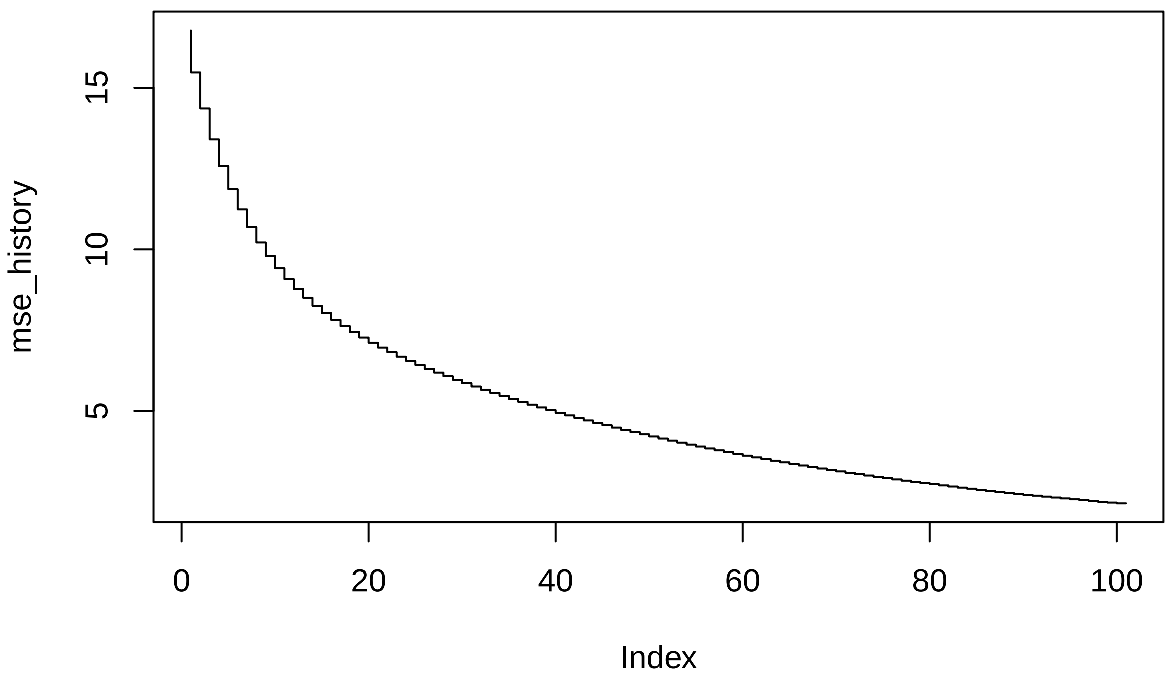


```
## [1] "Optimal intercept: 1.91835558791842 Optimal slope: 0.996133140819179"
```

Keep the history and plot the cost function over time.

```
plot_cost_function <- function(x, y, learn_rate, conv_threshold, n, max_iter) {

  m <- 0
  c <- 0
  yhat <- m * x + c
  MSE <- sum((y - yhat) ^ 2) / n

  mse_history <- vector(length = max_iter)
  mse_history[1] <- MSE

  converged = F
  iterations = 0
  while(converged == F) {
    ## Implement the gradient descent algorithm
    m_new <- m - learn_rate * ((1 / n) * (sum((yhat - y) * x)))
    c_new <- c - learn_rate * ((1 / n) * (sum(yhat - y)))
    m <- m_new
    c <- c_new
    yhat <- m * x + c
    MSE_new <- sum((y - yhat) ^ 2) / n
    if(MSE - MSE_new <= conv_threshold) {
      converged = T
      mse_history[iterations] <- MSE_new
      plot(mse_history, type = "S")
      return(paste("Optimal intercept:", c, "Optimal slope:", m))
    }
    iterations = iterations + 1
    mse_history[iterations] <- MSE_new

    if(iterations > max_iter) {
      converged = T
      mse_history[iterations] <- MSE_new
      plot(mse_history, type = "S")
      return(paste("Optimal intercept:", c, "Optimal slope:", m))
    }
  }

}
```

```
plot_cost_function(x, y, 0.01, 0.001, 1000, 100)
```



```
## [1] "Optimal intercept: 1.91835558791842 Optimal slope: 0.996133140819179"
```

First neural network

- Load the Boston data and split it using set.seed(55) again.

```
library(MASS); library(tidyverse)
boston <- MASS::Boston

boston <- boston %>%
  mutate(dummy = if_else(medv > 20, 1, 0))
```

- Scale the data and create a confusion matrix based on the standard logistic regression.

- Create predictions (remember to scale also the test set). Report the accuracy.

The remaining questions will be implemented in Python

- Use the neuralnet package to create a simple neural network with 1 hidden layer, backprop algorithm, SSE cost function and a 0.1 learning rate parameter.

```
import pandas as pd
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

from sklearn import preprocessing
from sklearn.model_selection import train_test_split

x_vars = pd.DataFrame(preprocessing.normalize(r.boston.iloc[:,0:13]))
y_var = r.boston['dummy'].copy()

X_train, X_test, y_train, y_test = train_test_split(
  x_vars, y_var, test_size=0.5,
  random_state=55)
```

build a neural network

```
model = keras.Sequential()

model.add(layers.Dense(
    4, # Amount of Neurons
    input_dim=13, # Define an input dimension because this is the first layer
    activation='linear' # Use relu activation function because all inputs are positive
))

model.add(layers.Dense(
    1, # Amount of Neurons. We want one output
    activation='sigmoid' # Use sigmoid because we want to output a binary classification
))

model.compile(
    loss='mean_squared_error', # The loss function that is being minimized
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.1),
    metrics=['binary_accuracy', tf.keras.metrics.AUC()]
)
```

```
model.fit(
    X_train, # Input training data
    y_train, # Output training data
    epochs=1000, # Amount of iterations we want to train for
    verbose=0 # Amount of detail you want shown in terminal while training
)
```

```
## <keras.callbacks.History object at 0x7f5ce068e750>
```

- Report the accuracy of this network after predicting the test set.

```
model.test_on_batch(X_test, y_test, return_dict=True)
```

```
## {'loss': 0.1496087908744812, 'binary_accuracy': 0.8023715615272522, 'auc': 0.9216560125350952}
```

- Now change the network architecture to c(8,2), which means 2 hidden layers with 8 and 2 neurons respectively. Is accuracy increased?
- Apply early stopping and check the accuracy again.
- Plot model architecture.

```
tf.keras.utils.plot_model(model)
```

```
## <IPython.core.display.Image object>
```