Cloud Computing and Cyber Security HW4

生機碩一 R09631007 吳乙澤

簡要：

利用 bitnami-docker-spark 安裝 spark 在 Macbook 中，之後進入 pyspark shell 跑放在下方網址的程式碼 https://github.com/tailer954/Cloud-Computing-and-Cyber-Security/tree/main/Spark-Logistic%20Regression 。

步驟及討論：

本作業有兩個部分，第一部分是根據老師所給網站的敘述，以 Spark 運行 Logistic Regression；第二部分則是撰寫自己的 SGD 函數進行訓練。我以三個步驟完成該作業，分別為「設定 Spark 環境」、「運行 Logistic Regression 範例程式」、「撰寫自己的 SGD 並運行」。在設定 Spark 環境時，遇到蠻多問題。譬如，bitnami 的 spark 沒有 nano、vi、vim 等文字編輯器，要下載又沒有權限。折騰了一會才發現要在 docker-compose.yml 動點手腳，使用者才能下載套件。

1. 設定 Spark 環境

我以 bitnami 的 docker-compose.yml 進行安裝，Spark 的版本為 3.0.1

(1) 以 curl 從網路下載 Spark 的 docker-compose.yml

```
curl -LO https://raw.githubusercontent.com/bitnami/bitnami-docker-spark/master/docker-compose.yml
```

(2) 在 docker-compose.yml 中的每個 image 加入 user: root，讓使用者進入 docker image 後，有 root 的權限安裝套件，如 numpy、nano、git 等

```
services:
  spark:
    image: docker.io/bitnami/spark:3-debian-10
+   user: root
    environment:
      - SPARK_MODE=master
      - SPARK_RPC_AUTHENTICATION_ENABLED=no
```

(3) 以 docker-compose 安裝 Spark

```
docker-compose up
```

(4) 進入 Spark Docker Container 後，建置 Spark

```
docker exec -it 38b46a7741ac bash
./build/mvn -DskipTests clean package
```

2. 運行 Logistic Regression 範例程式

logistic regression 的程式碼主要參照 https://github.com/samarthbhargav/spark-example/blob/master/run-spark-ex.py 及 https://spark.apache.org/docs/latest/mllib-linear-methods.html 兩個網站的介紹撰寫而成

(1) 進入 Docker 後，以 curl 從網路抓訓練用的資料集下來

```
curl -o dataset.txt https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt
```

(2) 進入 pyspark shell

```
pyspark
```

```
root@38b46a7741ac:/opt/bitnami/spark# pyspark
Python 3.6.12 (default, Oct 25 2020, 12:15:12)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
20/11/24 10:06:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Python version 3.6.12 (default, Oct 25 2020 12:15:12)
SparkSession available as 'spark'.
```

(3) 由於版本關係，老師所附網站的程式碼有 bug 存在。看過課堂助教的 Demo，並且查過 Google 後，我對老師所給的程式碼做了些調整。譬如，mapper 函數 return 的內容不能是 ndarray，最後 lambda 函數也要更改，否則會有 error 出現

```
def mapper(line):
    """
    Mapper that converts an input line to a feature vector
    """
    feats = line.strip().split(",")
    # labels must be at the beginning for LRSGD
    label = feats[len(feats) - 1]
    feats = feats[: len(feats) - 1]
    feats.insert(0,label)
    features = [ float(feature) for feature in feats ] # need floats
    return LabeledPoint(label, features)
```

```
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
trainErr = labelsAndPreds.filter(lambda lp: lp[0] != lp[1]).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```

(4) 在 pyspark shell 中，輸入上方的程式碼，便能印出 Training Error。見下圖所示，該 Error 為 0.007288629737609329

```
>>> labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
>>> trainErr = labelsAndPreds.filter(lambda lp: lp[0] != lp[1]).count() / float(parsedData.count())
>>> print("Training Error = " + str(trainErr))
Training Error = 0.007288629737609329
```

3. 撰寫自己的 SGD 並運行

該部分的程式碼我沒有用到 Spark，而是透過 numpy、random、math 等套件，自己手刻一個 Logistic Regression 的程式。將程式撰寫完畢後，我有以交叉熵(cross entropy)把 error 印出來，並跟第二部分「運行 Logistic Regression 範例程式」所得到的結果作比較

(1) 前處理。下載資料集後，將之以逗號作分割，並加到適當的資料結構中，方便後續訓練

```
# Setup Variable
dataset = []
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt"
r = requests.get(url)
open('dataset.txt', 'wb').write(r.content)
f = open('.\\dataset.txt')

# Make Dataset
for line in f:
    oneLineData = []
    arr = line.strip()
    arr = line.split(",")
    for element in arr:
        oneLineData.append(float(element))
    dataset.append(oneLineData)
```

(2) SGD。撰寫自己的 SGD，我以 while 做 iteration，for 把資料從資料結構中取出，以 math、numpy、加減乘除法更新權重(weight)及偏差值(bias)

```
# Iteration
while (True):
    # Set break condition and Reset
    iteration -= 1
    if iteration < 0 :
        break
    predict = 0
    loss = 0
    weightChange = [0, 0, 0, 0]
    biasChange = 0

    # Train
    dataset = np.array(dataset)        # Tranform list to nparray for computation
    weight  = np.array(weight)
    for i in datasetOrder[0:trainIndex]:
        data = dataset[i]
        ax = sum(data[0:4]*weight)
        predict = sigmoid(ax + bias)
        loss = -(data[4]*math.log(predict) + (1-data[4])*math.log(1-predict)) + loss
        biasChange    = biasChange + (predict - data[4])
        for i in range(4):
            weightChange[i] = weightChange[i]  + (predict - data[4])*data[i]
    # Renew the parameters
    weight = weight - lr/len(dataset) * np.array(weightChange)
    bias   = bias   - lr/len(dataset) * biasChange
```

(3) 將結果印出，包括 Algorithm Spending Time、Error、Accuracy 等等。見下圖所示，該 Error 為 0.07937029610999298，是 pyspark error 的 10 倍左右。我以 200 個 iteration、0.1 的學習率做訓練。見下方的 Error 圖，Error 一直都是下降的趨勢，表 0.1 學習率對該資料集是恰當的

```
# Get and print the result
correct = 0
for i in datasetOrder[trainIndex:testIndex]:
    data = dataset[i]
    ax = sum(data[0:4]*weight)
    predict = sigmoid(ax + bias)
    if data[4] == round(predict):
        correct += 1
print("Training Dataset Length : " + str(trainIndex))
print("Testing  Dataset Length : " + str(testIndex-trainIndex))
print("Weights is : " + str(weight))
print("Bias    is : " + str(bias) + "\n")
print("Error     : " + str(loss/trainIndex))
print("Accuracy : " + str(correct/(testIndex-trainIndex)))

# Plot
plt.plot(loss_history)
plt.title('Train Error')
plt.xlabel('Iteration')
plt.ylabel('Error')
plt.show()
```

```
My logistic regression algorithm spends 2.7266578674316406 s

Training Dataset Length : 892
Testing  Dataset Length : 480
Weights is : [-1.27422271 -0.70540891 -0.75306067 -0.22072343]
Bias    is : 0.8735332595275244

Error     : 0.07937029610999298
Accuracy : 0.9770833333333333
```