Spark_HW2_KNN

106065503 蔡仲庭

(一)

輸入三個參數去給 Spark 跑

vim run_yarn.sh

圖一，最後一段 knn_test/inpu 567832 6

分別是 hdfs 上"SUSY_pred.csv"的位置，567832 是 user 查詢的 id，6 是 k 個 near。

```
spark-submit --class KNN \
--num-executors 32 \
--deploy-mode client \
target/scala-2.10/knn-application_2.10-1.0.jar knn_test/input 567832 6
```

圖(一)

(二)

圖二，csv 檔案讀進來後有 20 欄，第一欄是 id，第二到第十九是特徵值，第 20 欄是 label，依照這個格式 map 並形成這個結構的 RDD:

data: org.apache.spark.rdd.RDD[(Int, Array[Double], Double)]

備註 ：.slice 的 function 是把第二欄到第十九欄的值合併成一個 array

```
      val data = sc.textFile(files).map(line =>line.split(',')).map(elems => (ele
ms(0).toInt,elems.slice(1,19).map(_.toDouble),elems(19).toDouble))
```

圖(二)

(三)

圖三，算兩個 node 的距離，u 跟 v 是兩個 node:

distance(u,v) = [sum_{i=1….n}(v_i - u_i)^2] ^ {1/2}

```
def distance(xs: Array[Double], ys: Array[Double]) = {
    sqrt((xs zip ys).map { case (x,y) => pow(y - x, 2) }.sum)
  }
```

圖(三)

圖四，val input 存放的是使用者輸入 id 後得到值，因為我們只會用到 tuple 中的第二個，所以要._2。之後用 input 去跟資料集中的每個點去算距離並將結果 map 而形成 Point_distance，它是一個 RDD 結構 ：

Point_distance: org.apache.spark.rdd.RDD[(Int, Double, Double)]

```
val input = data.take(id+1)(id)._2

println("Your Input Id is : "+ id + "\n Your Input K is :" + k )

println("")

val Point_distance = data.map(d=>(d._1,distance(d._2,input),d._3))
```

圖(四)

(四)

圖五，目的是去找到 k 個最近距離並顯示出來

Point_distance.filter(_._1 !=id)是因爲要過濾自己跟自己去算距離的結果

.sortBy(_._2).take(k) 排序並找出 k 個最近的 node。

Find_K_near 是一個陣列存放排序後並找出 k 個後的結果。

```
val Find_K_near = Point_distance.filter(_._1 != id).sortBy(_._2).take(k)
Find_K_near.foreach(println)
```

圖(五)

Find_K_near.foreach(println)後結果如圖六所示。

```
(388549,0.45567682941273047,0.0)
(199834,0.6125625036507685,0.0)
(4449577,0.636307246429779,0.0)
(4821478,0.6497774336040087,0.0)
(2444950,0.6550362683335391,1.0)
(834631,0.6557941620032456,0.0)
```

圖(六)

(五)

用平行的方式去計算這 Find_K_near 這個陣列

.map(points =>(points._3,1)).reduceByKey(_+_)利用 map 及 reduce 的概念去計算 label 為 0 時有幾個，label 為 1 時有幾個。

.sortBy(_._2,false).first 則是 label 最多的結果。

Ex : labelCount: (Double, Int) = (0.0,3)

```
        val labelCount = sc.parallelize(Find_K_near).map(points =>(points._3,1)).re
duceByKey(_+_).sortBy(_._2,false).first
```

圖(七)

(六)

圖八，顯示 Predict Result 它的結果(labelCount._1)及它的機率 (labelCount._2.toDouble / k)。Real Result 則是一開始輸入 id 的真實結果。

```
println("Predict Result : " + labelCount._1)
println("Probability : " + labelCount._2.toDouble / k)

println("")

println("Real result : "+ data.take(id+1)(id)._3.toInt.toString )
```

圖(八)

(七)

如果今天 id 是 100，k 是 6，最後的結果圖如圖九表示。

```
 Main function !!!!!!

Your Input Id is : 100
 Your Input K is :6

(2234142,0.40360034701006786,1.0)
(4611754,0.46493642992557477,1.0)
(3892895,0.46614369224762736,0.0)
(959809,0.46803571039782704,1.0)
(2650781,0.4809535118204027,0.0)
(375028,0.49814619140585364,0.0)
-----------------------------------
Predict Result : 0.0
Probability : 0.5

Real result : 0
```

圖(九)