**H2O**

**CLASS Hands on Exercise:**

***Purpose: Use an increasingly popular open source tool H2O for Deep Learning – speeds up programming from traditional Python / R but must be integrated with one. We chose R.***

***Goal: Verify we accurately predict our future trading strategies (based on ‘actionable insight’ from prior ‘trained’ data)***

**Our Use Case:** Classify (Categorize) our trading strategies as:

SB – Strong Buy (predict 10%+ gain in 1 month)

B – Buy (3% - 10%)

N – Neutral (-3% to 3%)

S – Sell Short (-3% to -10%)

SS – Strong Sell Short (over 10%)

6 input nodes (variables) will determine each category. The categories (all 5 above) will be the output nodes (dependent variables).

6 input nodes:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| fund | tech | RPearn | RPind | RPearnRel | RPindRel |

‘fund’ or fundamental analysis of a stock’s (rate top 5 analysts from 0-100); SB will be score 80-100;

B: 60-80; N: 40-60; S: 20-40; SS: 20 or less;

***Installation of R/H2O***

* Java JRE must be installed – recommend Java 8. Choose the JDK & install
* Windows - go to http://cran.rstudio.com/bin/windows/
* Or -🡪 **https://www.rstudio.com/products/rstudio/download2/**
* download, and install R and R Studio
* **Start R (run RStudio), and type install.packages(" h2o").**

**Also install the file I will send to you: groups-input-csv-original.csv and groups-input-csv. Can you install the files in C:\R?**

Next: (Ted will explain each step ….)

**library( h2o)**

**h2o.init( nthreads = -1)**

**1st RUN**

**> data <-h2o.importFile("C:\\R\\groups-input-csv-original.csv")**

**2nd RUN**

**> data <-h2o.importFile("C:\\R\\groups-input-csv.csv")**

**> y <-"group"**

**> x<- setdiff(names(data),y)**

**> parts <- h2o.splitFrame(data, 0.7)**

**> train <- parts[[ 1]]**

**> test <- parts[[ 2]]**

**> m <- h2o.deeplearning( x, y, train)**

**> p <- h2o.predict( m, test)**

**> p**

***Here is our Output 🡪***

predict B N S SB SS

1 SB 9.755072e-04 8.424343e-08 3.856824e-14 **9.990244e-01** 1.894005e-16

2 SS 1.672041e-13 7.191833e-08 2.560162e-03 6.774200e-13 **9.974398e-01**

3 B **9.867068e-01** 6.182144e-03 7.920686e-08 7.110992e-03 3.762406e-12

4 SB 1.126703e-03 2.230613e-07 1.387223e-12 **9.988731e-01** 1.893214e-14

5 N 1.154476e-03 9.757761e-01 2.306447e-02 2.063789e-06 2.929290e-06

6 N 3.196863e-04 9.986640e-01 1.016343e-03 8.909465e-11 2.274998e-10

**> h2o.confusionMatrix(m)**

[57 rows x 6 columns]

=========================================================================

Confusion Matrix: vertical: actual; across: predicted

B N S SB SS Error Rate

B 19 0 0 0 0 0.0000 = 0 / 19

N 0 64 0 0 0 0.0000 = 0 / 64

S 0 0 24 0 0 0.0000 = 0 / 24

SB 0 0 0 20 0 0.0000 = 0 / 20

SS 0 0 0 0 16 0.0000 = 0 / 16

Totals 19 64 24 20 16 0.0000 **= 0 / 143**

**> p$predict**

predict

1 SB

2 SS

3 B

4 SB

5 N

6 N

**> as.data.frame(p**)

predict B N S. SB SS

1 SB 9.755072e-04 8.424343e-08 3.856824e-14 9.990244e-01 1.894005e-16

2 SS 1.672041e-13 7.191833e-08 2.560162e-03 6.774200e-13 9.974398e-01

3 B 9.867068e-01 6.182144e-03 7.920686e-08 7.110992e-03 3.762406e-12

4 SB 1.126703e-03 2.230613e-07 1.387223e-12 9.988731e-01 1.893214e-14

5 N 1.154476e-03 9.757761e-01 2.306447e-02 2.063789e-06 2.929290e-06

6 N 3.196863e-04 9.986640e-01 1.016343e-03 8.909465e-11 2.274998e-10

7 N 7.312873e-05 9.998145e-01 1.123734e-04 2.243012e-15 2.208243e-14

8 SB 2.732036e-02 4.633872e-06 7.255322e-12 9.726750e-01 9.095348e-15

9 SB 1.126703e-03 2.230613e-07 1.387223e-12 9.988731e-01 1.893214e-14

10 SS 6.486652e-14 2.971261e-08 8.344041e-04 1.808259e-12 9.991656e-01

11 N 2.987399e-04 9.994842e-01 2.170970e-04 1.372817e-10 7.060468e-11

12 N 7.215496e-04 9.981465e-01 1.131950e-03 1.513080e-10 1.192840e-10

13 N 1.115478e-05 9.999594e-01 2.941696e-05 1.601000e-19 2.533715e-17

14 N 4.097711e-04 9.994997e-01 9.050652e-05 1.997488e-19 1.132346e-17

15 SB 1.238738e-03 1.165237e-07 1.458103e-13 9.987611e-01 4.199577e-16

16 SS 1.672041e-13 7.191833e-08 2.560162e-03 6.774200e-13 9.974398e-01

17 B 9.798642e-01 1.654524e-02 1.547230e-06 3.588990e-03 9.323031e-11

18 SS 1.799050e-12 3.870013e-07 5.752833e-03 1.256745e-11 9.942468e-01

19 N 1.154476e-03 9.757761e-01 2.306447e-02 2.063789e-06 2.929290e-06

20 N 7.312873e-05 9.998145e-01 1.123734e-04 2.243012e-15 2.208243e-14

21 N 1.115478e-05 9.999594e-01 2.941696e-05 1.601000e-19 2.533715e-17

22 N 1.015854e-04 9.998165e-01 8.186494e-05 4.992762e-19 3.250622e-17

23 S 4.023253e-08 2.393425e-03 9.941349e-01 2.151755e-10 3.471660e-03

24 SS 1.672041e-13 7.191833e-08 2.560162e-03 6.774200e-13 9.974398e-01

25 B 9.798642e-01 1.654524e-02 1.547230e-06 3.588990e-03 9.323031e-11

26 SS 6.486652e-14 2.971261e-08 8.344041e-04 1.808259e-12 9.991656e-01

27 N 1.426038e-05 9.999456e-01 4.015656e-05 6.565538e-18 6.265492e-16

28 N 7.645227e-05 9.999102e-01 1.338490e-05 5.275651e-15 3.389424e-15

29 N 1.115478e-05 9.999594e-01 2.941696e-05 1.601000e-19 2.533715e-17

30 N 4.097711e-04 9.994997e-01 9.050652e-05 1.997488e-19 1.132346e-17

31 B 9.867068e-01 6.182144e-03 7.920686e-08 7.110992e-03 3.762406e-12

32 N 1.154476e-03 9.757761e-01 2.306447e-02 2.063789e-06 2.929290e-06

33 N 3.544523e-04 9.991044e-01 5.411661e-04 1.145299e-10 1.327113e-10

34 N 3.196863e-04 9.986640e-01 1.016343e-03 8.909465e-11 2.274998e-10

35 N 5.618263e-05 9.999117e-01 3.214116e-05 2.826224e-15 9.313570e-15

36 N 7.312873e-05 9.998145e-01 1.123734e-04 2.243012e-15 2.208243e-14

37 N 1.015854e-04 9.998165e-01 8.186494e-05 4.992762e-19 3.250622e-17

38 SS 1.672041e-13 7.191833e-08 2.560162e-03 6.774200e-13 9.974398e-01

39 SB 8.348513e-04 2.224845e-07 8.049780e-13 9.991649e-01 1.112631e-14

40 B 9.798642e-01 1.654524e-02 1.547230e-06 3.588990e-03 9.323031e-11

41 S 3.766160e-08 4.430838e-03 9.638648e-01 1.198844e-09 3.170428e-02

42 N 2.987399e-04 9.994842e-01 2.170970e-04 1.372817e-10 7.060468e-11

43 N 3.544523e-04 9.991044e-01 5.411661e-04 1.145299e-10 1.327113e-10

44 N 1.115478e-05 9.999594e-01 2.941696e-05 1.601000e-19 2.533715e-17

45 B 8.290510e-01 1.503993e-01 7.627065e-07 2.054890e-02 1.080927e-10

46 SS 1.799050e-12 3.870013e-07 5.752833e-03 1.256745e-11 9.942468e-01

47 N 9.797917e-03 9.899141e-01 2.234244e-04 6.451131e-05 5.380773e-08

48 N 3.544523e-04 9.991044e-01 5.411661e-04 1.145299e-10 1.327113e-10

49 N 5.618263e-05 9.999117e-01 3.214116e-05 2.826224e-15 9.313570e-15

50 B 9.792264e-01 2.064622e-03 3.041374e-09 1.870895e-02 8.397225e-14

51 SS 1.672041e-13 7.191833e-08 2.560162e-03 6.774200e-13 9.974398e-01

52 SS 6.486652e-14 2.971261e-08 8.344041e-04 1.808259e-12 9.991656e-01

53 N 5.948543e-04 9.972433e-01 2.161819e-03 2.411978e-08 1.142070e-08

54 N 2.987399e-04 9.994842e-01 2.170970e-04 1.372817e-10 7.060468e-11

55 N 1.426038e-05 9.999456e-01 4.015656e-05 6.565538e-18 6.265492e-16

56 N 1.277972e-05 9.999550e-01 3.223691e-05 6.331860e-18 5.343621e-16

57 N 3.673299e-04 9.991115e-01 5.211423e-04 1.159279e-19 8.673570e-17

**> as.data.frame( h2o.cbind(p$predict,test$group) )**

predict group

1 B S

2 SS SS

3 N N

4 N N

5 N N

6 N N

7 N N

8 S S

9 B B

10 N N

\*\* I printed 1st 10 only above

\*\*\*\* next run:

**as.data.frame(h2o.cbind(p$predict,test$group)**

**\*\*\* what is the output; can you interpret? Ted will explain**

**\*\*\*\*STOP HERE – unless we are in class ……**

**Next: (NOT with original data)**

**h2o.shutdown();**

**Restart & run with new data-set; interpret all results:**

Copy and paste into RStudio

**library( h2o)**

**h2o.init( nthreads = -1)**

**data <-h2o.importFile("C:\\R\\groups-input-csv.csv")**

**y <-"group"**

**x<- setdiff(names(data),y)**

**parts <- h2o.splitFrame(data, 0.7)**

**train <- parts[[ 1]]**

**test <- parts[[ 2]]**

**m <- h2o.deeplearning( x, y, train)**

**p <- h2o.predict( m, test)**

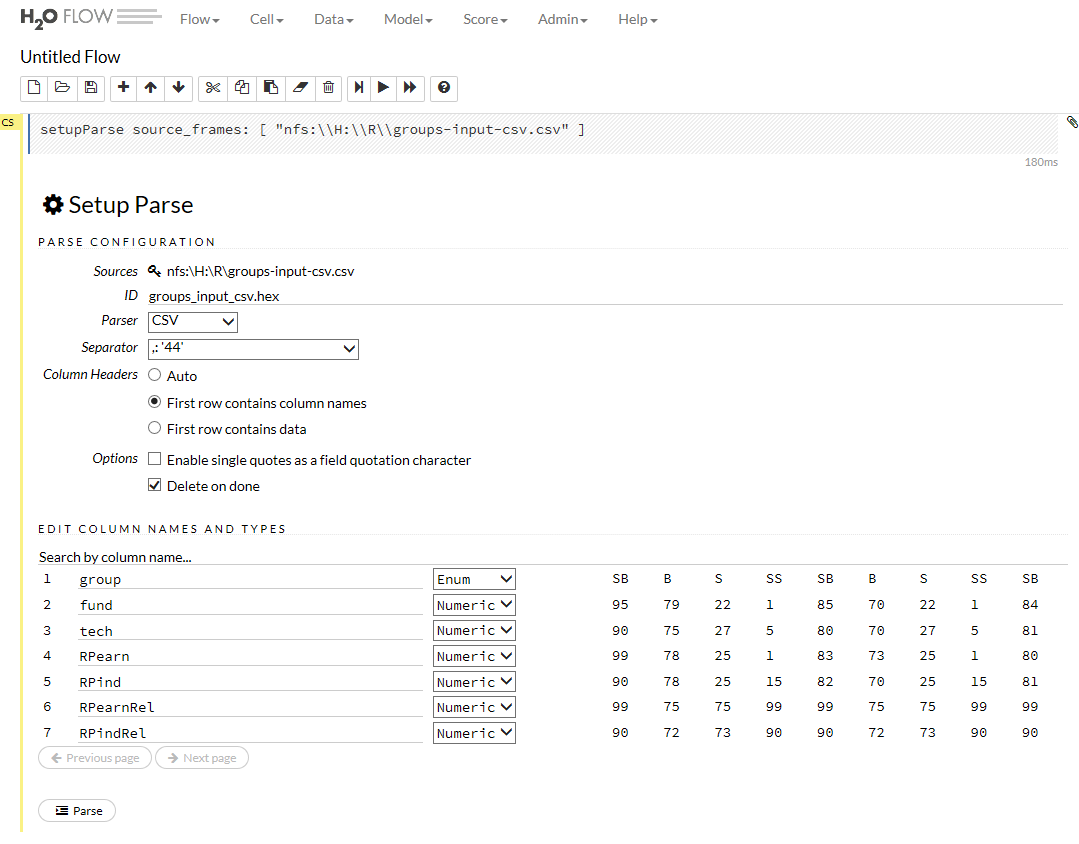
**as.data.frame(p**)

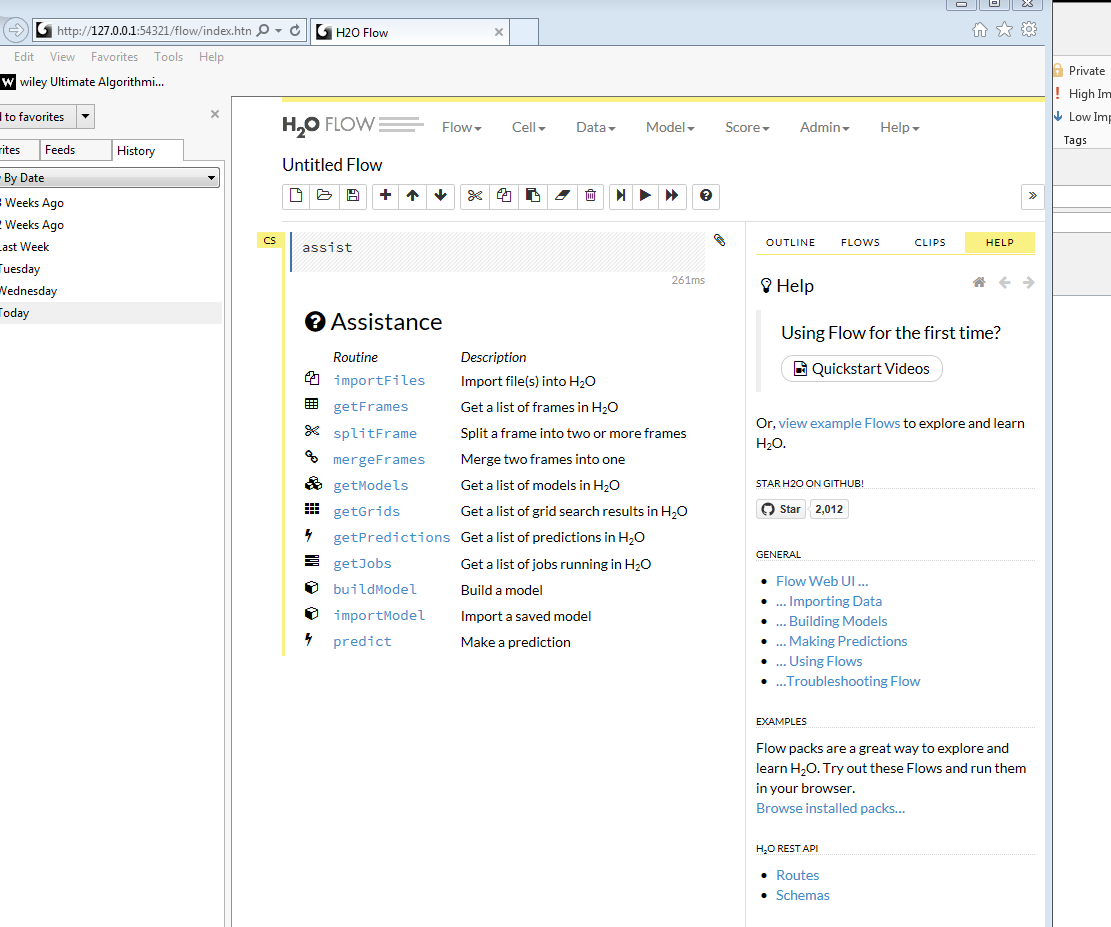
**h2o.confusionMatrix(m)**

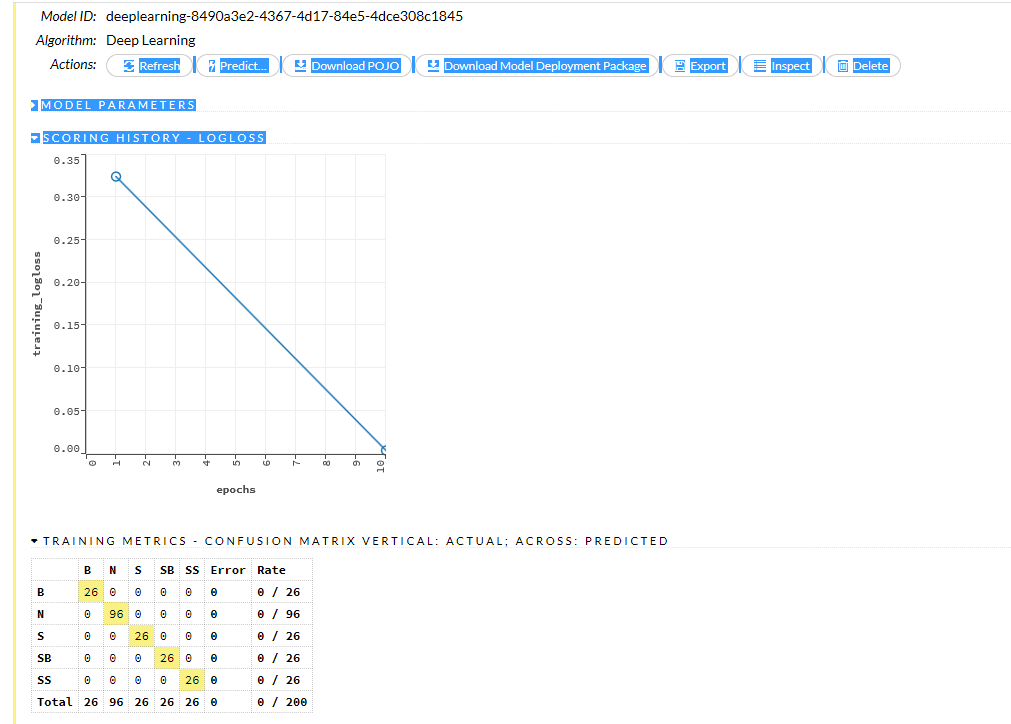
**as.data.frame( h2o.cbind(p$predict,test$group) )**

TRY THIS – simply type link; Ted will lead from there.

<http://127.0.0.1:54321/flow/index.html>







**With original file:**

Copy and paste into RStudio

**library( h2o)**

**h2o.init( nthreads = -1)**

**data <-h2o.importFile("C:\\R\\groups-input-csv-original.csv")**

**y <-"group"**

**x<- setdiff(names(data),y)**

**parts <- h2o.splitFrame(data, 0.7)**

**train <- parts[[ 1]]**

**test <- parts[[ 2]]**

**m <- h2o.deeplearning( x, y, train)**

**p <- h2o.predict( m, test)**

**as.data.frame(p**)

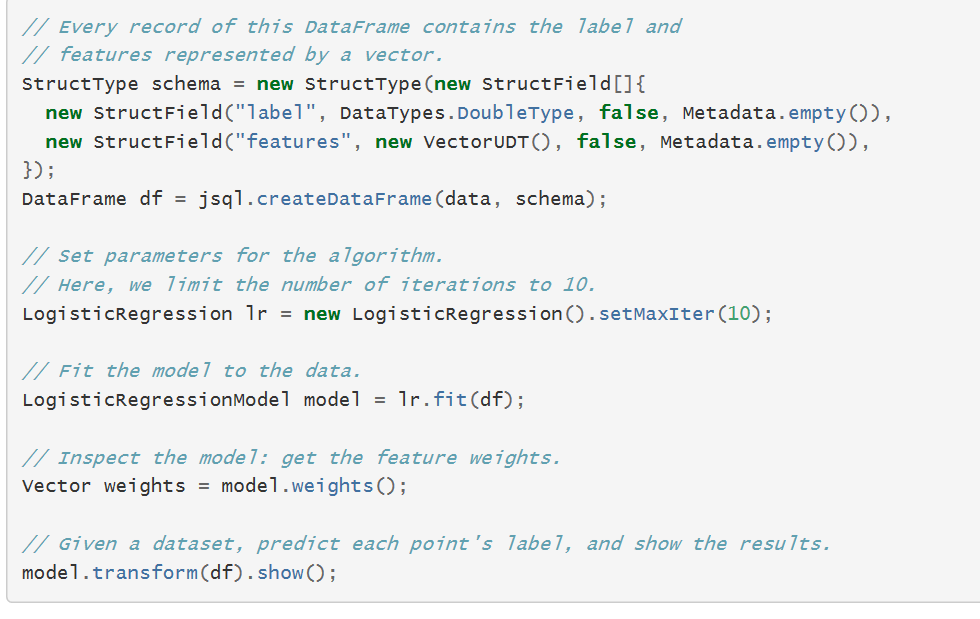
**h2o.confusionMatrix(m)**

**as.data.frame( h2o.cbind(p$predict,test$group) )**

\*\*\* ALL BELOW – is EXTRA, may NOT be covered …

MLlib is Spark’s scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives, as outlined below:

* [Data types](https://spark.apache.org/docs/1.2.0/mllib-data-types.html)
* [Basic statistics](https://spark.apache.org/docs/1.2.0/mllib-statistics.html)
  + summary statistics
  + correlations
  + stratified sampling
  + hypothesis testing
  + random data generation
* [Classification and regression](https://spark.apache.org/docs/1.2.0/mllib-classification-regression.html)
  + [linear models (SVMs, logistic regression, linear regression)](https://spark.apache.org/docs/1.2.0/mllib-linear-methods.html)
  + [naive Bayes](https://spark.apache.org/docs/1.2.0/mllib-naive-bayes.html)
  + [decision trees](https://spark.apache.org/docs/1.2.0/mllib-decision-tree.html)
  + [ensembles of trees](https://spark.apache.org/docs/1.2.0/mllib-ensembles.html) (Random Forests and Gradient-Boosted Trees)
* [Collaborative filtering](https://spark.apache.org/docs/1.2.0/mllib-collaborative-filtering.html)
  + alternating least squares (ALS)
* [Clustering](https://spark.apache.org/docs/1.2.0/mllib-clustering.html)
  + k-means
* [Dimensionality reduction](https://spark.apache.org/docs/1.2.0/mllib-dimensionality-reduction.html)
  + singular value decomposition (SVD)
  + principal component analysis (PCA)
* [Feature extraction and transformation](https://spark.apache.org/docs/1.2.0/mllib-feature-extraction.html)
* [Optimization (developer)](https://spark.apache.org/docs/1.2.0/mllib-optimization.html)
  + stochastic gradient descent
  + limited-memory BFGS (L-BFGS)



|  |
| --- |
| public final class JavaQueueStream { |
|  | private JavaQueueStream() { |
|  | } |
|  |  |
|  | public static void main(String[] args) throws Exception { |
|  |  |
|  | StreamingExamples.setStreamingLogLevels(); |
|  | SparkConf sparkConf = new SparkConf().setAppName("JavaQueueStream"); |
|  |  |
|  | // Create the context |
|  | JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, new Duration(1000)); |
|  |  |
|  | // Create the queue through which RDDs can be pushed to |
|  | // a QueueInputDStream |
|  |  |
|  | // Create and push some RDDs into the queue |
|  | List<Integer> list = new ArrayList<>(); |
|  | for (int i = 0; i < 1000; i++) { |
|  | list.add(i); |
|  | } |
|  |  |
|  | Queue<JavaRDD<Integer>> rddQueue = new LinkedList<>(); |
|  | for (int i = 0; i < 30; i++) { |
|  | rddQueue.add(ssc.sparkContext().parallelize(list)); |
|  | } |
|  |  |
|  | // Create the QueueInputDStream and use it do some processing |
|  | JavaDStream<Integer> inputStream = ssc.queueStream(rddQueue); |
|  | JavaPairDStream<Integer, Integer> mappedStream = inputStream.mapToPair( |
|  | i -> new Tuple2<>(i % 10, 1)); |
|  | JavaPairDStream<Integer, Integer> reducedStream = mappedStream.reduceByKey( |
|  | (i1, i2) -> i1 + i2); |
|  |  |
|  | reducedStream.print(); |
|  | ssc.start(); |
|  | ssc.awaitTermination(); |
|  | } |
|  | } |