PROOF OF CONCEPT 1

FIT 5202 – GROUP ASSIGNMENT

Wongnaret Khantuwan
That Bao Thu Ton
Thi Bich Ngoc Hoang
Lam Wai Chun

Group 1

Semester 2 - 2019

Overview

When data arrive in a stream, we estimate clusters dynamically, updating them as new data. Spark MLlib provides support for streaming k-means clustering.

Code explanation

1. Choice of Spark API

The spark API will be imported to support the graph processing and result displaying. The usage of each API is listed as below

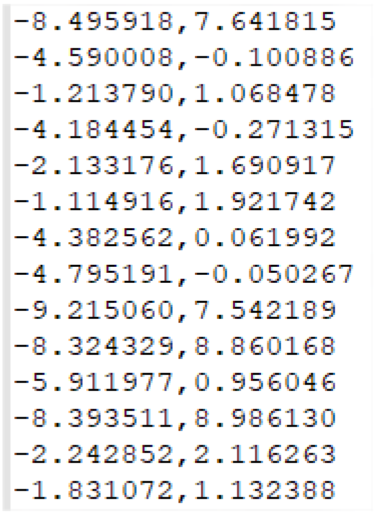
API	Usages
org.apache.spark.mllib.clustering.StreamingKMeans	ML class for k-means analysis
org.apache.spark.mllib.linalg.Vectors	class for vector creating.
org.apache.spark.mllib.regression.LabeledPoint	the features and labels of a data point.
org.apache.spark.streaming.{Seconds, StreamingContext}	This class is required for streaming task

Figure 1: Choice of SparkAPI table

2. Code flow

a. **Assumption:** This task is a Streaming task. Thus, the new data will stream to Training set directory and Test set directory and Streaming K-means will refresh these two directories every 3 seconds.

b. **Assumption:** The data is two-dimensional data, and the number of desired K-means clusters is 3.

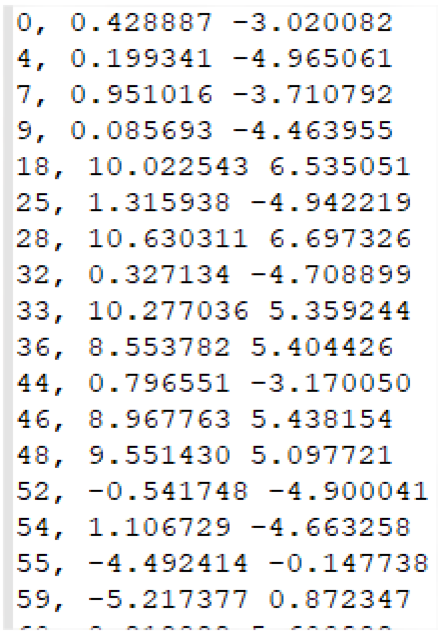


```
-8.495918,7.641815
-4.590008,-0.100886
-1.213790,1.068478
-4.184454,-0.271315
-2.133176,1.690917
-1.114916,1.921742
-4.382562,0.061992
-4.795191,-0.050267
-9.215060,7.542189
-8.324329,8.860168
-5.911977,0.956046
-8.393511,8.986130
-2.242852,2.116263
-1.831072,1.132388
```

c. **Input File format:**
This task consist of 2 types of the input file:

- Training data: the training data has the format “feature_1,feature_2” as the following sample

- Test data: the training data has the format
“row_id, feature_1 feature_2” as the following sample



```
0,  0.428887 -3.020082
4,  0.199341 -4.965061
7,  0.951016 -3.710792
9,  0.085693 -4.463955
18,  10.022543 6.535051
25,  1.315938 -4.942219
28,  10.630311 6.697326
32,  0.327134 -4.708899
33,  10.277036 5.359244
36,  8.553782 5.404426
44,  0.796551 -3.170050
46,  8.967763 5.438154
48,  9.551430 5.097721
52,  -0.541748 -4.900041
54,  1.106729 -4.663258
55,  -4.492414 -0.147738
59,  -5.217377 0.872347
```

Figure 2 illustrates the overview of code flow to doing streaming k-means clustering.

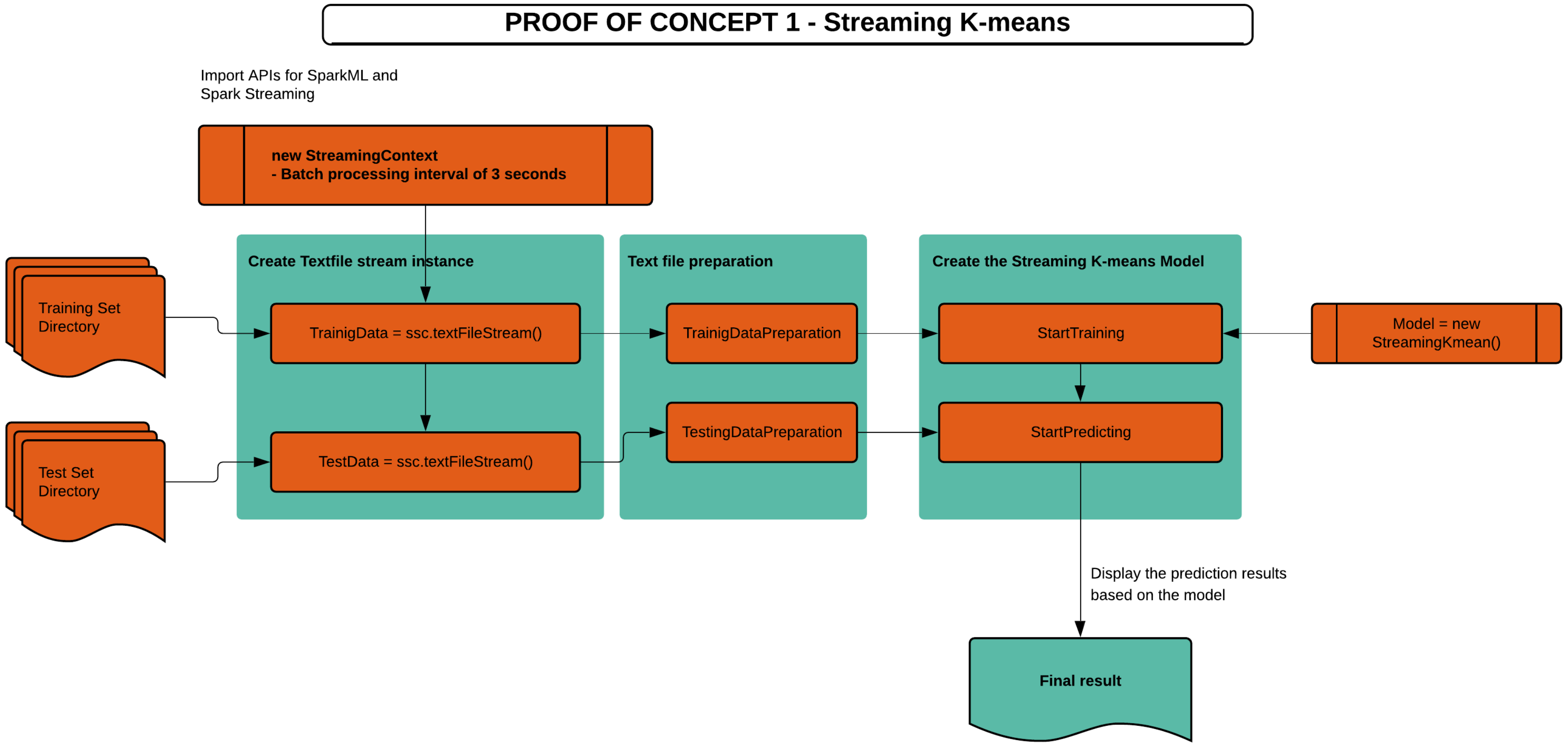

PROOF OF CONCEPT 1 - Streaming K-means
Import APIs for SparkML and Spark Streaming
new StreamingContext - Batch processing interval of 3 seconds
Create Textfile stream instance
Text file preparation
Create the Streaming K-means Model
Training Set Directory
TrainigData = ssc.textFileStream()
TrainigDataPreparation
StartTraining
Model = new StreamingKmean()
Test Set Directory
TestData = ssc.textFileStream()
TestingDataPreparation
StartPredicting
Display the prediction results based on the model
Final result

Figure 2: Code Flow (Overview)

Step 1: Read data and import necessary APIs

After importing all necessary APIs such as :

```
import org.apache.spark.mllib.clustering.StreamingKMeans
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.streaming._
```

The StreamingContext instance is created and set interval time for refreshing the data sources every 3 seconds. Then the data source for training data and testing data is assigned using textFileStream()

Next, the data will be prepared and convert into vectors for feeding to the Streaming K-means Model. The detail of the vector creating processing will be described in the next section

Step 2: Text preparation

Since the Streaming is read the input data as string. Thus we need to process the input string into to the correct based on Map function. Figure 3 illustrates the flow for processing the input.

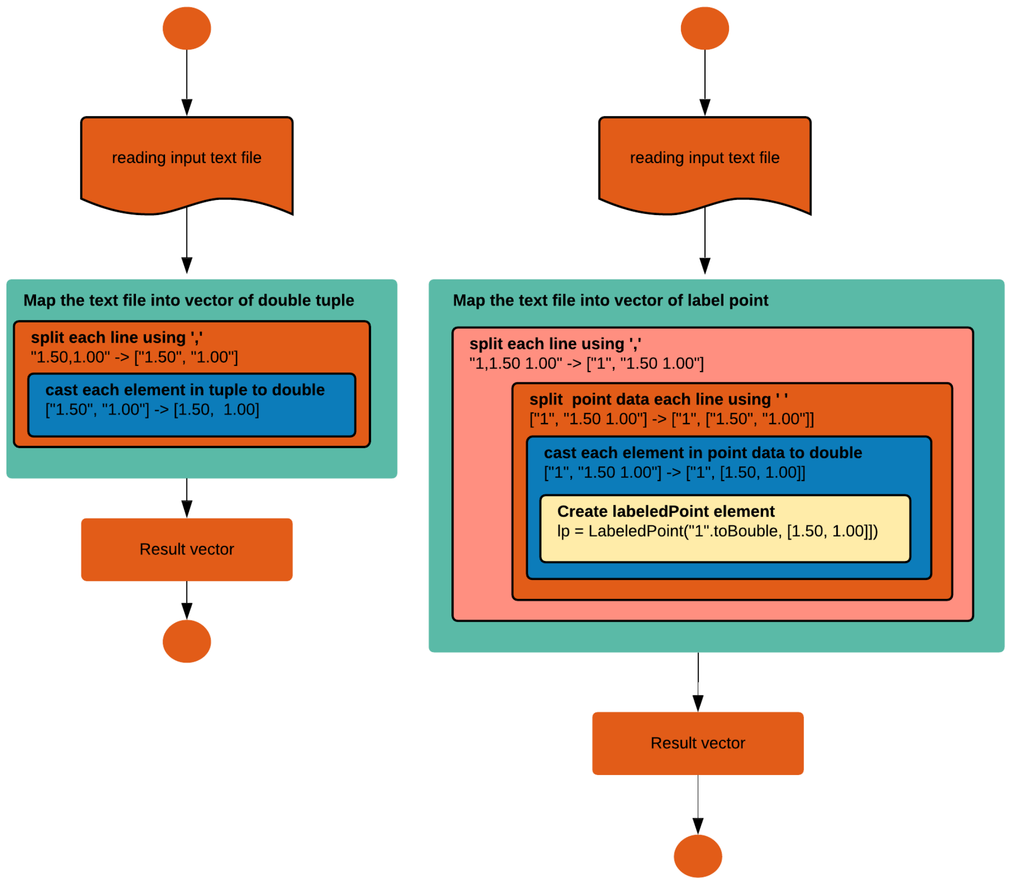

reading input text file
Map the text file into vector of double tuple
split each line using ','
"1.50,1.00" -> ["1.50", "1.00"]
cast each element in tuple to double
["1.50", "1.00"] -> [1.50, 1.00]
Result vector
reading input text file
Map the text file into vector of label point
split each line using ','
"1,1.50 1.00" -> ["1", "1.50 1.00"]
split point data each line using ' '
["1", "1.50 1.00"] -> ["1", ["1.50", "1.00"]]
cast each element in point data to double
["1", "1.50 1.00"] -> ["1", [1.50, 1.00]]
Create labeledPoint element
lp = LabeledPoint("1".toBouble, [1.50, 1.00]])
Result vector

Figure 3: Text file Preparation

Step 3: Create the Streaming K-means Model

In this step, we will create the Streaming K-means Model instance using StreamingKMeans class, The number of clusters is assigned to 3, decay factor is assigned to 1, and the dimension of the data is 2 dimensions as shown on the following code.

```
val model = new StreamingKMeans().setK(3).setDecayFactor(0.5).setRandomCenters(2,100.0)
```

For the decay factor, It is a parameter to control the decay (or "forgetfulness") of the algorithm.

3.1 Training

The training data vector is used for training the model as shown on the following code.

```
model.trainOn(trainingData)
```

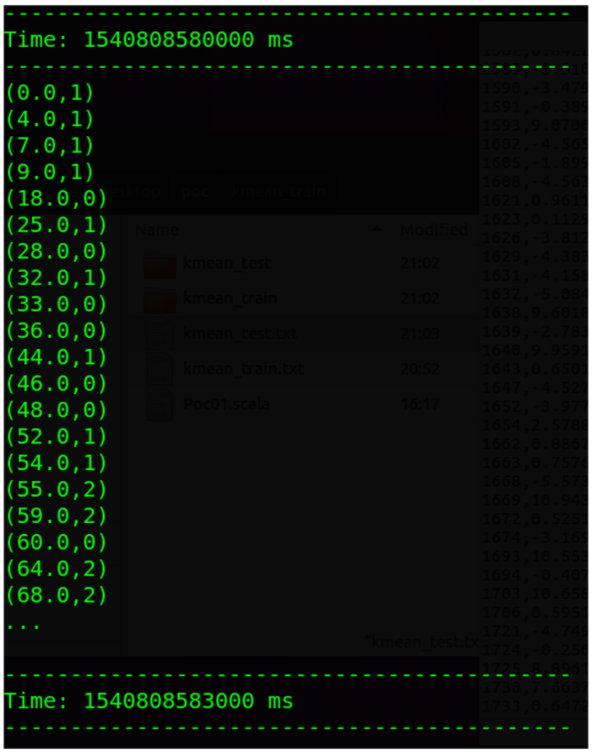
3.2 Testing

The testing data with the LabeledPoint vector format is used for testing based on the trained model from the previous step. The following code show how to predict the data.

```
model.predictOnValues(testData.map(lp => (lp.label, lp.features))).print()
```

Step 4: Displaying the Prediction result

The prediction result will be display on the screen with the format (row id, predictions values) as the following figure.



```
-------------------------------------------
Time: 1540808580000 ms
-------------------------------------------
(0.0,1)
(4.0,1)
(7.0,1)
(9.0,1)
(18.0,0)
(25.0,1)
(28.0,0)
(32.0,1)
(33.0,0)
(36.0,0)
(44.0,1)
(46.0,0)
(48.0,0)
(52.0,1)
(54.0,1)
(55.0,2)
(59.0,2)
(60.0,0)
(64.0,2)
(68.0,2)
...

-------------------------------------------
Time: 1540808583000 ms
-------------------------------------------
```

Figure 4: Prediction results