CS6350: BIG DATA ANALYTICS and MANAGEMENT

Fall 2016

HW #3

Related to: Data Analytics and Stream Framework using Spark

Due: November 18, 2016 by 11.55 p.m.

This homework consists of two parts. Here, we focus on K-means clustering (data analytics), classification, recommendation (collaborative filtering) and steam framework with Spark and Kafka.

Part A:

Q1

Using spark machine learning library spark-mlib, **use kmeans to cluster the movies using the ratings given by the user,** that is, use the item-user matrix from **itemusermat File provided** as input to your program.

Dataset description.

Dataset: Itemusermat File.

The **itemusermat file contains** the ratings given to each movie by the users in **Matrix format.** The file contains the ratings by users for 1000 movies.

Each line contains the movies id and the list of ratings given by the users.

A rating of 0 is used for entries where the user did not rate a movie.

From the sample below, user1 did not rate movie 2, so we use a rating of 0.

A sample **Itemusermat file** with the item-user matrix is shown below.

	user1	user2
movie1	4	3
movies2	0	2

Set the number of clusters (**k**) to 10

Your Scala/python code should produce the following output:

- For each cluster, **print any 5 movies in the cluster. Your output should contain the movie_id, movie title, genre and the corresponding cluster** it belongs to. **Note:** Use the **movies.dat** file to obtain the movie title and genre.

For example
cluster: 1
123,Star wars, sci-fi

Q2 Classification

Using spark MLlib, use the supervised learning (decision tree and Naive Bayes) algorithms to classify types of glass based on the dataset "glass.data"

The dataset comprises of the following attributes.

Attribute Information:

1. Id number: 1 to 214
2. RI: refractive index
3. Na: Sodium (unit measurement: weight percent in corresponding oxide, as

   are attributes 4-10)
4. Mg: Magnesium
5. Al: Aluminum
6. Si: Silicon
7. K: Potassium
8. Ca: Calcium
9. Ba: Barium
10. Fe: Iron

11. Type of glass: (class attribute)

-- 1 building_windows_float_processed

-- 2 building_windows_non_float_processed

-- 3 vehicle_windows_float_processed

-- 4 vehicle_windows_non_float_processed (none in this database)

-- 5 containers

-- 6 tableware

-- 7 headlamps

Please use 60% of the data for training and 40% for testing and give the accuracy of the classifiers.

Q3. Use Collaborative filtering find the accuracy of ALS model accuracy. Use **ratings.dat** file. It contains

User id :: movie id :: ratings :: timestamp. Your program should report the accuracy of the model.

For details follow the link: http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

Please use 60% of the data for training and 40% for testing and report the accuracy of the model.

Part B: Spark Streaming framework with Kafka

Q4. Write a scala/Python/Java program using Apache Spark to find Top N bi-gram (https://en.wikipedia.org/wiki/N-gram) words (with its frequencies) from a document whose frequencies are above a certain number in real-time. **You have to design a real-time streaming framework with Apache Spark and Apache Kafka.** The Spark Streaming module will find Top N bi-gram periodically if it gets data. Use Kafka Console producer to give input. You can type a sentence/paragraph in Kafka Console producer and you will see corresponding output to Spark Streaming module. The output will be generated periodically for input data. Set Spark micro-batch period to 10/15 sec.

For example, for the following raw text, we want to apply bi-gram analysis.

“Alice is testing spark application. Testing spark is fun”.

Applying bi-gram will generate token like this:

[(Alice, 'is'), ('is', 'testing'), ('testing', ‘spark’), (‘spark’, 'application.'), ('testing', ‘spark’), (‘spark’, 'is'), (‘is’, 'fun')].

So, if we want to find top 2 frequent bi-gram tokens whose frequencies are above 1, the output will be:

('testing', ‘spark’) 2

In this program, you have also remove the stop words (**http://www.textfixer.com/resources/common-english-words.txt**) and stemming (use nltk/porter stemmer) (**http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html**)

before applying your bi-gram logic for your document. For example,

“Testing spark is fun” will look like “test spark be fun”.

After stemming, your output of bi-gram will be like following:

Document:

“Alice is testing spark application. Testing spark is fun”.

After removing stop words, stemming and applying bi-gram will generate tokens like this:

[(Alice, 'be'), ('be', 'test'), ('test', ‘spark’), (‘spark’, 'application.'), ('test', ‘spark’), (‘spark’, 'be'), (‘be’, 'fun')].

So, if we want to find top 2 frequent bi-gram tokens whose frequencies are above 1, the output will be:
('test', ‘spark’) 2