Meetup Data Analysis

By,

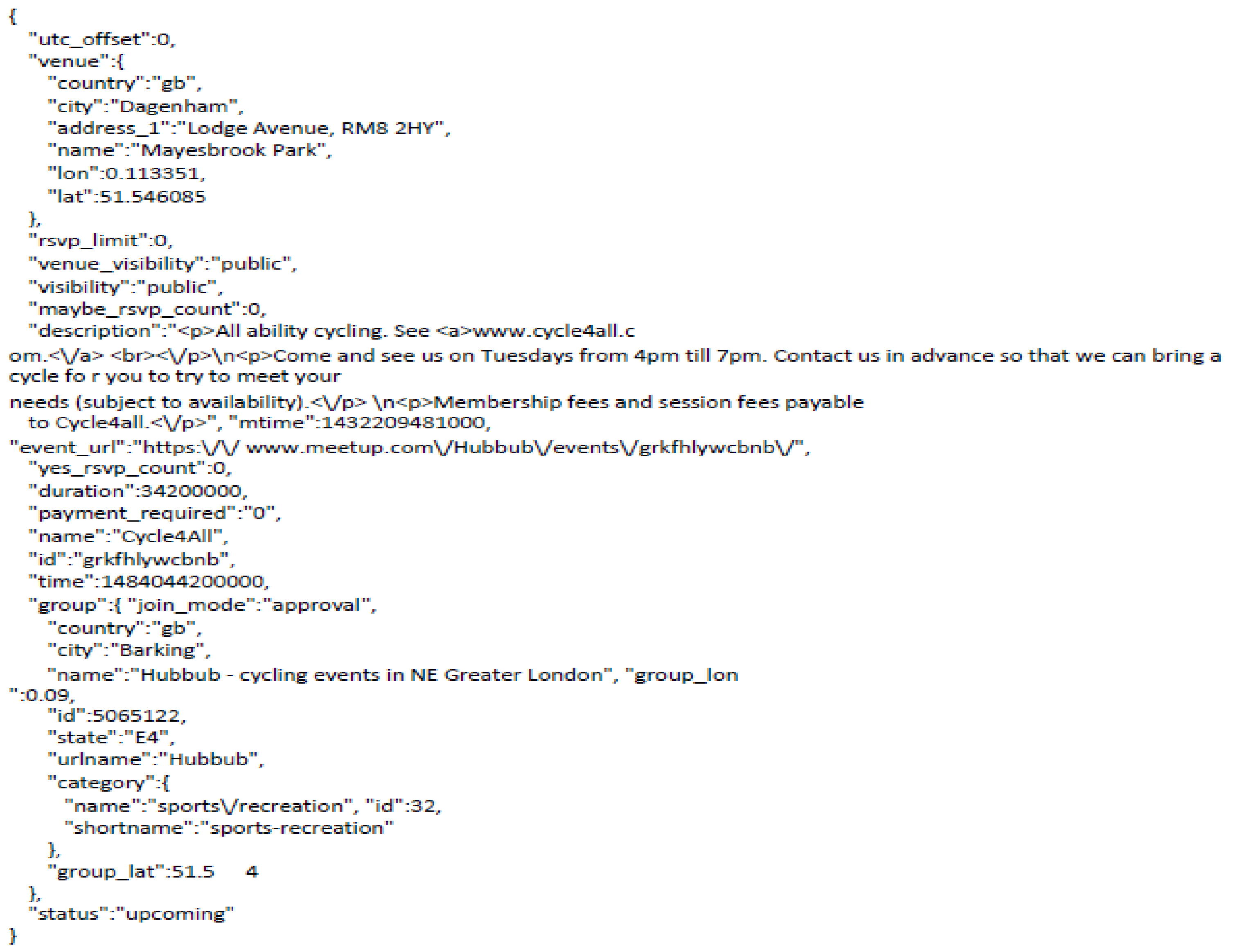
Sushmanth Sagala

Spark Project – UpX Academy

Project Information

- **Domain:** Social
- **Technology use:** Spark streaming, Spark, Spark MLlib
- **Dataset:** http://stream.meetup.com/2/open_events
- Meetup is an online social networking portal that facilitates offline group meetings in various localities around the world. Meetup allows members to find and join groups unified by a common interest, such as politics, books and games.

Sample Events Dataset



```
{
  "utc_offset":0,
  "venue":{
    "country":"gb",
    "city":"Dagenham",
    "address_1":"Lodge Avenue, RM8 2HY",
    "name":"Mayesbrook Park",
    "lon":0.113351,
    "lat":51.546085
  },
  "rsvp_limit":0,
  "venue_visibility":"public",
  "visibility":"public",
  "maybe_rsvp_count":0,
  "description":"<p>All ability cycling. See <a>www.cycle4all.c
om.<\/a> <br><\/p>\n<p>Come and see us on Tuesdays from 4pm till 7pm. Contact us in advance so that we can bring a
cycle fo r you to try to meet your
needs (subject to availability).<\/p> \n<p>Membership fees and session fees payable
  to Cycle4all.<\/p>", "mtime":1432209481000,
"event_url":"https:\/\/ www.meetup.com\/Hubbub\/events\/grkfhlywcbnb\/",
  "yes_rsvp_count":0,
  "duration":34200000,
  "payment_required":"0",
  "name":"Cycle4All",
  "id":"grkfhlywcbnb",
  "time":1484044200000,
  "group":{ "join_mode":"approval",
    "country":"gb",
    "city":"Barking",
    "name":"Hubbub - cycling events in NE Greater London", "group_lon
":0.09,
    "id":5065122,
    "state":"E4",
    "urlname":"Hubbub",
    "category":{
      "name":"sports\/recreation", "id":32,
      "shortname":"sports-recreation"
    },
    "group_lat":51.5     4
  },
  "status":"upcoming"
}
```

Business Questions

- **Streaming/Spark Sql**
  - Load the streaming data
  - Count the number of events happening in a city eg. Hyderabad
  - Count the number of free events
  - Count the events in Technology category
  - Count the number of Big data events happening in US
  - Find the average duration of Technology events
- **Spark MLLIB**
  - Group the events by their category (k-means clustering)

Q1. Load the streaming data

- Custom receiver to load data from external URL.
- Asynchronous HTTP request to read the data from streaming URL.
- def onStart() {

```
        val cf = new AsyncHttpClientConfig.Builder()
        cf.setRequestTimeout(Integer.MAX_VALUE)
        cf.setReadTimeout(Integer.MAX_VALUE)
        cf.setPooledConnectionIdleTimeout(Integer.MAX_VALUE)
        client= new AsyncHttpClient(cf.build())
        inputPipe = new PipedInputStream(1024 * 1024)
        outputPipe = new PipedOutputStream(inputPipe)
        val producerThread = new Thread(new DataConsumer(inputPipe))
            producerThread.start()
            client.prepareGet(url).execute(new AsyncHandler[Unit]{
            def onBodyPartReceived(bodyPart: HttpResponseBodyPart) = {
                bodyPart.writeTo(outputPipe)
                AsyncHandler.STATE.CONTINUE
            }
            ....
        })
  }
```

Q1. Load the streaming data

- Class DataConsumer extends Runnable to read the stream data and store.

```
        val bufferedReader = new BufferedReader( new InputStreamReader( inputStream ))
        var input=bufferedReader.readLine()
        while(input!=null){
            store(input)
            input=bufferedReader.readLine() }
```

- Defining the case classes to extract respective data.

```
    case class EventDetails(id: String, name: String, city: String, country: String, payment_required: Int, cat_id: Int, cat_name: String, duration: Long)

    case class Venue(name: Option[String], address1: Option[String], city: Option[String], state: Option[String], zip: Option[String], country: Option[String], lon: Option[Float], lat: Option[Float])

    case class Event(id: String, name: Option[String], eventUrl: Option[String], description: Option[String], duration: Option[Long], rsvpLimit: Option[Int], paymentRequired: Option[Int], status: Option[String])

    case class Group(id: Option[String], name: Option[String], city: Option[String], state: Option[String], country: Option[String])

    case class Category(name: Option[String], id: Option[Int], shortname: Option[String])
```

Q1. Load the streaming data

- parseEvent method uses Json4s lib to extract the json data and define the EventDetails type.

```
val json=parse(eventJson).camelizeKeys
val event=json.extract[Event]
val venue=(json \ "venue").extract[Venue]
val group=(json \ "group").extract[Group]
val category=(json \ "group" \ "category").extract[Category]
EventDetails(event.id, event.name.getOrElse(""), venue.city.getOrElse(""), venue.country.getOrElse(""), event.paymentRequired.getOrElse(0), category.id.getOrElse(0), category.shortname.getOrElse(""), event.duration.getOrElse(10800000L))
```

- Starting the event stream with Batch Interval of 2 secs,

```
val ssc=new StreamingContext(conf, Seconds(2))
val eventStream = ssc.receiverStream(new MeetupReceiver("http://stream.meetup.com/2/open_events")).flatMap(parseEvent)
```

Stateful Stream

- Using Window stream to do aggregations across Intervals of stream.
- Window and Slide interval = 10 sec
- Batch interval = 2 sec
- val windowEventStream = eventStream.window(Seconds(10),Seconds(10)) windowEventStream.cache()
- Custom Functions to sum aggregations while using updateStateByKey.
- def updateSumFunc(values: Seq[Int], state: Option[Int]): Option[Int] = { val currentCount = values.sum val previousCount = state.getOrElse(0) Some(currentCount + previousCount) }
- def updateSumFunc2f(values: Seq[Double], state: Option[Double]): Option[Double] = { val currentCount = values.sum val previousCount = state.getOrElse(0.0) Some(currentCount + previousCount) }

Q2. Count the number of events happening in a city eg. Hyderabad

- Filtering the list of events happening in a city say “New York”.
- Reducing the events to get the number of events happening in this city for the current Window computation.
- Aggregating the events count across the Window intervals using updateStateByKey.

-
  ```
  val cityEventsStream = windowEventStream.filter{event => event.city ==
  "New York"}.map{event =>
  (event.city,1)}.reduceByKey(_+_).updateStateByKey(updateSumFunc _)
  ```
- Printing the count of number of events happening in “New York” during each Window interval.

-
  ```
  cityEventsStream.foreachRDD(rdd => {rdd.foreach{case (city, count) =>
  println("No. of Events happening in %s city::%s".format(city, count))}})
  ```

Q3. Count the number of free events

- Filtering the list of free events happening by using condition when ever payment_required value is 0.
- Reducing the events to get the number of free events happening for the current Window computation.
- Aggregating the events count across the Window intervals using updateStateByKey.
-
  ```
  val freeEventsStream = windowEventStream.filter{event =>
  event.payment_required == 0}.map{event =>
  ("Free",1)}.reduceByKey(_+_).updateStateByKey(updateSumFunc _)
  ```
- Printing the count of number of free events happening during each Window interval.
-
  ```
  freeEventsStream.foreachRDD(rdd => {rdd.foreach{case (free, count) =>
  println("No. of Free Events happening::%s".format(count))}})
  ```

Q4. Count the events in Technology category

- Filtering the list of Technology events happening.
- Reducing the events to get the number of Technology events happening for the current Window computation.
- Aggregating the events count across the Window intervals using updateStateByKey.
- Reusing the Technology category events for another question by storing the count in a stateless variable.
- val techEventsStream = windowEventStream.filter{event => event.cat_name == "tech"}
- var techCount = 0
- val countTexhEventsStream = techEventsStream.map{event => (event.cat_name,1)}.reduceByKey(_+_).updateStateByKey(updateSumFunc _)
- Printing the count of number of Technology events happening during each Window interval.
- countTexhEventsStream.foreachRDD(rdd => {rdd.foreach{case (cat_name, count) => techCount = count; println("No. of %s Events happening::%s".format(cat_name,count))}})

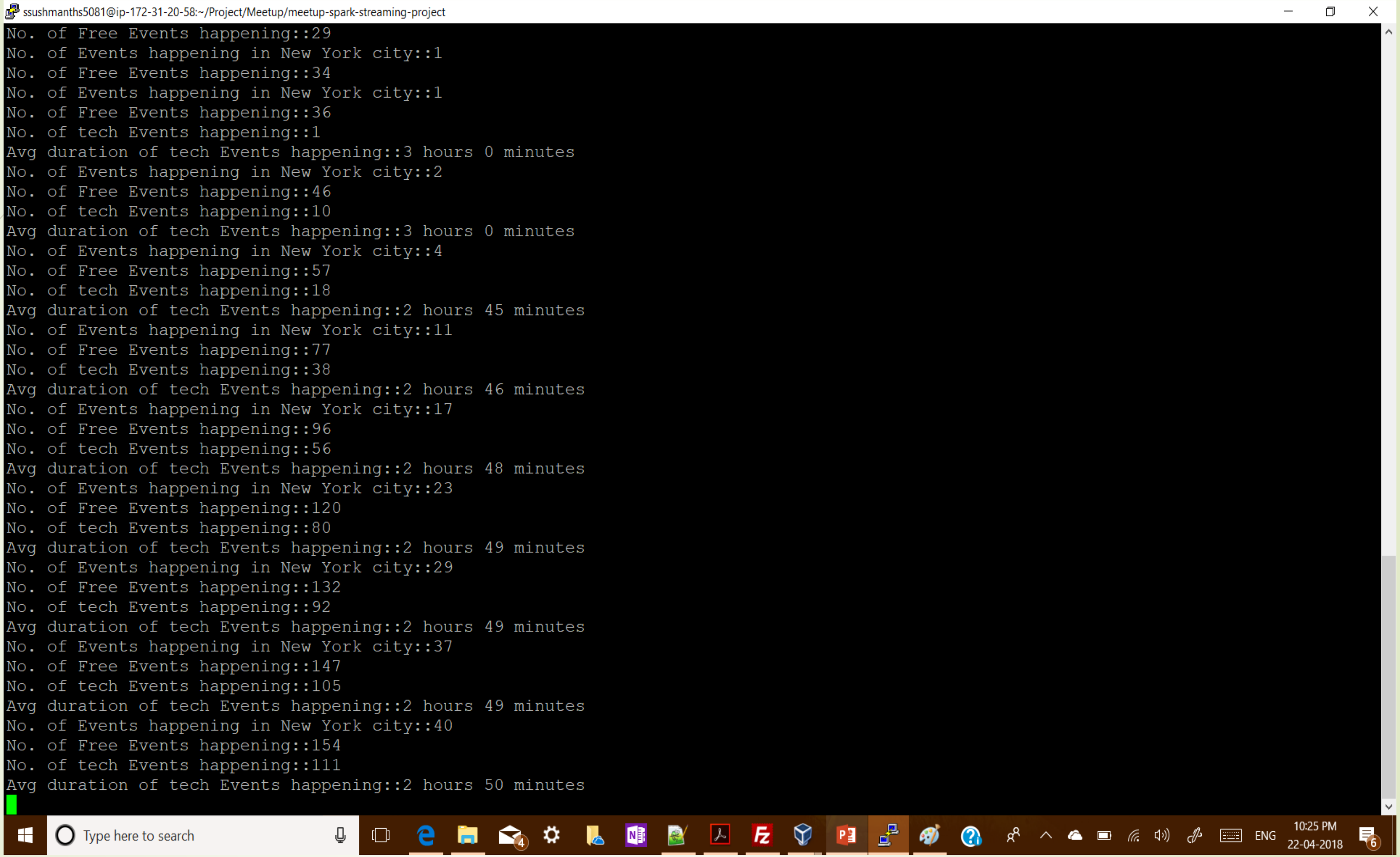
Q5. Count the number of Big data events happening in US

- Filtering the list of Big data events happening in “US”.
- Reducing the events to get the number of Big data events happening in US for the current Window computation.
- Aggregating the events count across the Window intervals using updateStateByKey.
- val bigDataUSEventsStream = windowEventStream.filter{event => event.country == "us" && event.name.toLowerCase.indexOf("big data") >= 0}.map{event => ("Big Data",1)}.reduceByKey(_+_).updateStateByKey(updateSumFunc _)
- Printing the count of number of Big data events happening in “US” during each Window interval.
- bigDataUSEventsStream.foreachRDD(rdd => {rdd.foreach{case (name, count) => println("No. of %s Events happening in US::%s".format(name,count))}})

Q6. Find the average duration of Technology events

- Reducing the Technology events to get the event duration for the current Window computation.
- Aggregating the events duration across the Window intervals using updateStateByKey.
- Computing the Average duration and Printing the Average duration for Technology events happening during each Window interval.
- val sumDurTechEventsStream = techEventsStream.map{event => (event.cat_name + " Events", event.duration.toDouble / 60000.0)}.reduceByKey(_+_).updateStateByKey(updateSumFunc2f _)
- sumDurTechEventsStream.foreachRDD(rdd => {

```
            rdd.map{case(x:String, y:Double) => (x, y / techCount.toDouble)}.foreach{case (cat_name:String, avg:Double) => {
            val hrs = (avg / 60.0).toInt
            val min = (avg % 60).toInt
            println("Avg duration of %s happening::%d hours %d minutes".format(cat_name,hrs,min))      }
    } })
```



ssushmanths5081@ip-172-31-20-58:~/Project/Meetup/meetup-spark-streaming-project
Type here to search
ENG
10:25 PM
22-04-2018

```
No. of Free Events happening::29
No. of Events happening in New York city::1
No. of Free Events happening::34
No. of Events happening in New York city::1
No. of Free Events happening::36
No. of tech Events happening::1
Avg duration of tech Events happening::3 hours 0 minutes
No. of Events happening in New York city::2
No. of Free Events happening::46
No. of tech Events happening::10
Avg duration of tech Events happening::3 hours 0 minutes
No. of Events happening in New York city::4
No. of Free Events happening::57
No. of tech Events happening::18
Avg duration of tech Events happening::2 hours 45 minutes
No. of Events happening in New York city::11
No. of Free Events happening::77
No. of tech Events happening::38
Avg duration of tech Events happening::2 hours 46 minutes
No. of Events happening in New York city::17
No. of Free Events happening::96
No. of tech Events happening::56
Avg duration of tech Events happening::2 hours 48 minutes
No. of Events happening in New York city::23
No. of Free Events happening::120
No. of tech Events happening::80
Avg duration of tech Events happening::2 hours 49 minutes
No. of Events happening in New York city::29
No. of Free Events happening::132
No. of tech Events happening::92
Avg duration of tech Events happening::2 hours 49 minutes
No. of Events happening in New York city::37
No. of Free Events happening::147
No. of tech Events happening::105
Avg duration of tech Events happening::2 hours 49 minutes
No. of Events happening in New York city::40
No. of Free Events happening::154
No. of tech Events happening::111
Avg duration of tech Events happening::2 hours 50 minutes
```

Sample output screenshot

Q7. Group the events by their category (k-means clustering)

- Building a recommendation model by using k-means clustering on events.
- Recommendation of group members is done based on clustering the event categories and rsvp's responses respect to events.
- Parsing history Events.
- val eventsHistory = ssc.sparkContext.textFile("data/events/events.json", 1).flatMap(parseHisEvent)
- Parsing history Rsvps.
- case class Member(memberName: Option[String], memberId: Option[String])
- case class MemberEvent(eventId: Option[String], eventName: Option[String], eventUrl: Option[String], time: Option[Long])
  - val json=parse(rsvpJson).camelizeKeys
  - val member=(json \ "member").extract[Member]
  - val event=(json \ "event").extract[MemberEvent]
  - val response=(json \ "response").extract[String]
  - (member, event, response)
- val rsvpHistory = ssc.sparkContext.textFile("data/rsvps/rsvps.json", 1).flatMap(parseRsvp)

Q7. Group the events by their category (k-means clustering)

- Broadcasting Dictionary to load list of English dictionary words.
- val localDictionary = Source.fromURL(getClass.getResource("/wordsEn.txt")).getLines.zipWithIndex.toMap
- val dictionary= ssc.sparkContext.broadcast(localDictionary)
- Feature Extraction to get the 10 most popular words from the event description, to form the event category vectors for each event.
- def eventToVector(dictionary: Map[String, Int], description: String): Option[Vector]={

```
        val wordsIterator = breakToWords(description)
        val topWords=popularWords(wordsIterator)
        if (topWords.size==10) Some(Vectors.sparse(dictionary.size,topWords))
else None }
```

- val eventVectors=eventsHistory.flatMap{ event=>eventToVector(dictionary.value,event.description.getOrElse("")) }

Q7. Group the events by their category (k-means clustering)

- Training the history events based on k-means clustering model.
- val eventClusters = KMeans.train(eventVectors, 10, 2)
- Creating the Event History Ids and RSVP Member Event Id to join based on the Event ID.
- val eventHistoryById=eventsHistory.map{event=>(event.id, event.description.getOrElse(""))}.reduceByKey{(first: String, second: String)=>first}
- val membersByEventId=rsvpHistory.flatMap{ case(member, memberEvent, response) => memberEvent.eventId.map{id=>(id,(member, response))} }
- val rsvpEventInfo=membersByEventId.join(eventHistoryById)
- Example: (eventId, ((member, response), description))
- (221069430, ((Member(Some(Susan Beck),Some(101089292)), yes), '…'))
- (221149038, ((Member(Some(Tracy Ramey),Some(153724262), no), '…'))

Q7. Group the events by their category (k-means clustering)

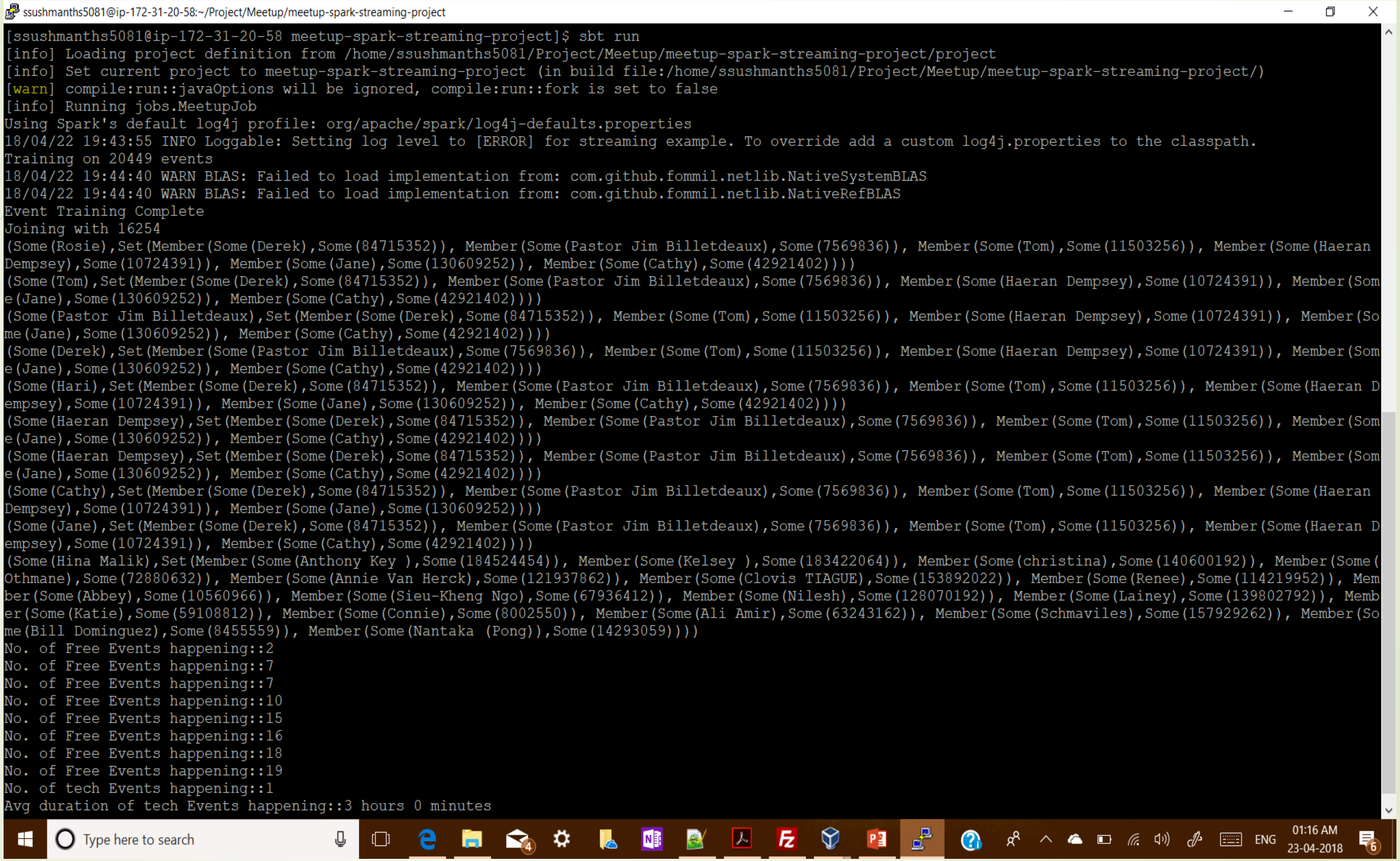
- Predicting the Event cluster based on the trained model.
- val memberEventInfo = rsvpEventInfo.flatMap{ case(eventId, ((member, response), description)) => {eventToVector(dictionary.value,description).map{ eventVector=> val eventCluster=eventClusters.predict(eventVector) (eventCluster,(member, response))   }   } }

- Clustering members into groups based on the predictions.
- val memberGroups = memberEventInfo.filter{case(cluster, (member, memberResponse)) => memberResponse == "yes"}.map{case(cluster, (member, memberResponse)) => (cluster,member)}.groupByKey().map{case(cluster,memberItr) => (cluster,memberItr.toSet)}

Q7. Group the events by their category (k-means clustering)

- Member Recommendations based on the clustering.
- val recommendations = memberEventInfo.join(memberGroups).map{case(cluster, ((member, memberResponse), members)) => (member.memberName, members-member)}

- Example: (member.memberName, members)
- (Some(Rosie),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Some(Haeran Dempsey),Some(10724391)), Member(Some(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))



ssushmanths5081@ip-172-31-20-58:~/Project/Meetup/meetup-spark-streaming-project

```
[ssushmanths5081@ip-172-31-20-58 meetup-spark-streaming-project]$ sbt run
[info] Loading project definition from /home/ssushmanths5081/Project/Meetup/meetup-spark-streaming-project/project
[info] Set current project to meetup-spark-streaming-project (in build file:/home/ssushmanths5081/Project/Meetup/meetup-spark-streaming-project/)
[warn] compile:run::javaOptions will be ignored, compile:run::fork is set to false
[info] Running jobs.MeetupJob
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/04/22 19:43:55 INFO Loggable: Setting log level to [ERROR] for streaming example. To override add a custom log4j.properties to the classpath.
Training on 20449 events
18/04/22 19:44:40 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
18/04/22 19:44:40 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
Event Training Complete
Joining with 16254
(Some(Rosie),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Some(Haeran
Dempsey),Some(10724391)), Member(Some(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))
(Some(Tom),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Haeran Dempsey),Some(10724391)), Member(Som
e(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))
(Some(Pastor Jim Billetdeaux),Set(Member(Some(Derek),Some(84715352)), Member(Some(Tom),Some(11503256)), Member(Some(Haeran Dempsey),Some(10724391)), Member(So
me(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))
(Some(Derek),Set(Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Some(Haeran Dempsey),Some(10724391)), Member(Som
e(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))
(Some(Hari),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Some(Haeran D
empsey),Some(10724391)), Member(Some(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))
(Some(Haeran Dempsey),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Som
e(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))
(Some(Haeran Dempsey),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Som
e(Jane),Some(130609252)), Member(Some(Cathy),Some(42921402))))
(Some(Cathy),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Some(Haeran
Dempsey),Some(10724391)), Member(Some(Jane),Some(130609252))))
(Some(Jane),Set(Member(Some(Derek),Some(84715352)), Member(Some(Pastor Jim Billetdeaux),Some(7569836)), Member(Some(Tom),Some(11503256)), Member(Some(Haeran D
empsey),Some(10724391)), Member(Some(Cathy),Some(42921402))))
(Some(Hina Malik),Set(Member(Some(Anthony Key ),Some(184524454)), Member(Some(Kelsey ),Some(183422064)), Member(Some(christina),Some(140600192)), Member(Some(
Othmane),Some(72880632)), Member(Some(Annie Van Herck),Some(121937862)), Member(Some(Clovis TIAGUE),Some(153892022)), Member(Some(Renee),Some(114219952)), Mem
ber(Some(Abbey),Some(10560966)), Member(Some(Sieu-Kheng Ngo),Some(67936412)), Member(Some(Nilesh),Some(128070192)), Member(Some(Lainey),Some(139802792)), Memb
er(Some(Katie),Some(59108812)), Member(Some(Connie),Some(8002550)), Member(Some(Ali Amir),Some(63243162)), Member(Some(Schmaviles),Some(157929262)), Member(So
me(Bill Dominguez),Some(8455559)), Member(Some(Nantaka (Pong)),Some(14293059))))
No. of Free Events happening::2
No. of Free Events happening::7
No. of Free Events happening::7
No. of Free Events happening::10
No. of Free Events happening::15
No. of Free Events happening::16
No. of Free Events happening::18
No. of Free Events happening::19
No. of tech Events happening::1
Avg duration of tech Events happening::3 hours 0 minutes
```

Sample output screenshot

Conclusion

- Meetup Streaming data loaded and analysed successfully.
- Streaming events data loaded through Spark Streaming using Custom Receivers and handled as Asynchronous HTTP requests.
- History Events and Rsvp data analysed through Spark MLlib to build an Group member recommendations based on K-means clustering model.
- Code: https://github.com/ssushmanth/meetup-stream