COEN 242 Big Data

Project 2

Movie Review Analysis and Twitter Emerging Topic Detector using Apache Spark

Immanuel Amirtharaj
Jackson Beadle

June 13, 2018

Commands to Run

Part 1

Both Scala applications for Part 1 were run on the Hadoop cluster in client mode. Although the JAR files execute in both client and cluster mode, we specifically chose client mode as it gave more detailed information about CPU time. This information was important for our performance analysis discussed later. The JAR files can be executed on the Hadoop cluster with the following commands.

For Query 1:

```
spark-submit --class Popularity --master yarn-client spark1.jar
<movie_path> <reviews_path> <output_path>
```

For Query 2:

```
spark-submit --class Reviews --master yarn-client spark2.jar
<movie_path> <reviews_path> <output_path>
```

Part 2

We ran into issues executing the JAR file for the Twitter emerging topics detector on the Hadoop cluster. When trying both client and cluster mode, YARN was unable to open the class. Thus, we executed the Spark program locally in the Intellij IDE. Our project has the same settings as the example described in the Part 2 help document on semantic analysis. All Intellij project files are included in the submission.

Execution on the Hadoop cluster can be called with the following command.

```
spark-submit --class TwitterAnalyzer --master yarn-client twitter.jar
<consumer_key> <consumer_secret> <access_token> <access_token_secret>
```

Methodology

Movie Reviews

For both queries we used the Spark RDD approach. We verified that both of these queries produced the same results as Project 1.

For Query 1 we first mapped the movie data as a collection of tuples in the form (movieId, movieTitle). We then mapped the review data as a collection in the form of (movieId, 1). We called reduceByKey to aggregate the keys and calculate a reviewCount for output in the form (movieId, reviewCount). To get the movie name in addition to the review count, we joined the mapped reviews and movieIds on the movieId and sorted it in ascending order.

For Query 2 we first mapped the reviews as a tuple (movieId, rating). We then grouped by the key movieId which returns a tuple where the first element is movieId and the second is an array of movie reviews associated with the movieId. Then we mapped that result in the format of (movieId, (averageRating, reviewCount)). We were able to then filter the RDD to remove any movies with 10 or fewer ratings or an average rating before 4.0.

Twitter Semantic Analysis

For the second part of the project, we used a DStream of tweets and stateful streaming to detect emerging topics. Topics are defined as unique hashtags that may appear in tweets. We used sliding windows in order to determine the frequency of topics in different subsets of the Twitter data stream. We used a window duration of 45 seconds and a sliding duration of 15 seconds.

To determine the emerging topic, all the hashtags in the window of tweets were read, mapped, and reduced to calculate the count of unique topics. The data stream is then sorted and the topic with the largest positive difference between states was defined as the emerging topic. To calculate the difference between two windows, we used mapWithState in order to keep a record of the topics counts in the previous window. For a given Twitter topic, the previous count is stored as the state. However, our StateSpec function to update our state also returns a value, the difference in counts. This output from our mapWithState function gives us the data stream to sort and pick the most emergent topic.

Tweets not written in English, as determined by checking the language field of the tweet metadata, or not containing hashtags were not included in our emerging topic windows. Semantic analysis of non-English tweets is less accurate and topics cannot be determined without any hashtags by our given definition.

Code Snippet

The code snippet below provides our implementation of the StateSpec function used for mapWithState and how the topics are mapped and reduced to determine emerging topics. The final statement extracts the emergent topic from the topics data stream.

```
// This function updates the state with the new count
// and returns the difference between the windows in terms of topic frequency
def updateTopicCounts(key: String,
                      value: Option[Int],
                      state: State[Int]): (String, Int) =
{
 val existingCount: Int =
   state
     .getOption()
     .getOrElse(0)

 val newCount = value.getOrElse(0)

 state.update(newCount)
 (key, newCount - existingCount)
}

// Define our StateSpec function to be called by mapWithState
val stateSpec = StateSpec.function(updateTopicCounts _)

// map and count the frequency of topics in the hashtags, sort in descending order by count
val topics = window.flatMap(t => t("hashtags").asInstanceOf[Array[String]])
    .map(h => (h, 1))
    .reduceByKey(_+_)
    .mapWithState(stateSpec)
    .map{case (topic, count) => (count, topic)}
    .transform(_.sortByKey(false))


var emergingTopic = ""
// use rdd.take to extract the first topic which is the one with the highest difference
topics.foreachRDD(rdd => {
 val top = rdd.take(1)
 top.foreach{case (count, topic) => emergingTopic = topic}
})
```

Results

Query 1 Output

```
39058   Monty Python and the Holy Grail (1975)
39227   Good Will Hunting (1997)
39600   "Dark Knight, The (2008)"
39725   Indiana Jones and the Last Crusade (1989)
40103   One Flew Over the Cuckoo's Nest (1975)
40250   "Terminator, The (1984)"
40436   Die Hard: With a Vengeance (1995)
40706   Memento (2000)
40931   "Princess Bride, The (1987)"
41283   Beauty and the Beast (1991)
42193   Men in Black (a.k.a. MIB) (1997)
42558   Titanic (1997)
43660   Shrek (2001)
44121   Mission: Impossible (1996)
44742   Ace Ventura: Pet Detective (1994)
45303   "Lion King, The (1994)"
45413   Gladiator (2000)
45544   Speed (1994)
49643   "Sixth Sense, The (1999)"
50168   True Lies (1994)
50375   Aladdin (1992)
50809   Saving Private Ryan (1998)
51338   Dances with Wolves (1990)
51837   "Lord of the Rings: The Return of the King, The (2003)"
51882   "Lord of the Rings: The Two Towers, The (2002)"
52474   Fargo (1996)
52658   Seven (a.k.a. Se7en) (1995)
53398   Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
53717   Batman (1989)
54783   Back to the Future (1985)
56820   "Fugitive, The (1993)"
56827   "Lord of the Rings: The Fellowship of the Ring, The (2001)"
57070   "Godfather, The (1972)"
57232   Independence Day (a.k.a. ID4) (1996)
57416   Apollo 13 (1995)
57879   American Beauty (1999)
59271   "Usual Suspects, The (1995)"
59693   Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
60024   Fight Club (1999)
61672   Star Wars: Episode V - The Empire Strikes Back (1980)
61836   Terminator 2: Judgment Day (1991)
62714   Star Wars: Episode VI - Return of the Jedi (1983)
66008   Toy Story (1995)
66512   Braveheart (1995)
67662   Schindler's List (1993)
74355   Jurassic Park (1993)
77045   Star Wars: Episode IV - A New Hope (1977)
77960   "Matrix, The (1999)"
84078   "Silence of the Lambs, The (1991)"
87901   Pulp Fiction (1994)
91082   "Shawshank Redemption, The (1994)"
91921   Forrest Gump (1994)
```

Query 2 Output

```
"Matrix, The (1999)"    4.154098255515649   77960
Monty Python and the Holy Grail (1975)  4.155153873726253   39058
Memento (2000)  4.1570775807006335  40706
Touch of Evil (1958)    4.157241777264859   5199
To Kill a Mockingbird (1962)    4.157649361114308   17374
Chinatown (1974)    4.157715931945426   18397
Inception (2010)    4.161755956596878   35297
M (1931)    4.163554278678432   4873
"Big Sleep, The (1946)" 4.163652229097255   6303
A Song of Lisbon (1933) 4.166666666666667   15
Life Is Beautiful (La Vita è bella) (1997)  4.167062784709843   25245
Notorious (1946)    4.167152752460809   5486
Pulp Fiction (1994) 4.169975313136369   87901
All About Eve (1950)    4.174582798459563   5453
Black Mirror: White Christmas (2014)    4.174603174603175   63
Goodfellas (1990)   4.17828875746609    33987
Yojimbo (1961)  4.180264741275572   4155
"Guten Tag, Ramón (2013)"   4.181818181818182   11
"Dark Knight, The (2008)"   4.182070707070707   39600
Still Bill (2009)   4.1875  16
City of God (Cidade de Deus) (2002) 4.187872863087181   19947
"O Auto da Compadecida (Dog's Will, A) (2000)"  4.191558441558442   154
"Lives of Others, The (Das leben der Anderen) (2006)"   4.199038891372374   8948
A Silent Voice (2016)   4.2 20
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)   4.200819672131147   7930
Spirited Away (Sen to Chihiro no kamikakushi) (2001)    4.202589307120594   20855
Double Indemnity (1944) 4.202603887997147   5607
Paths of Glory (1957)   4.20264575040974    4271
North by Northwest (1959)   4.205228001893441   19013
"Third Man, The (1949)" 4.209418968212611   7676
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964) 4.213030410183875   28280
Casablanca (1942)   4.2143927037912325  30043
The Blue Planet (2001)  4.217948717948718   273
Over the Garden Wall (2013) 4.219745222929936   157
Whiplash (2013) 4.226775956284153   183
One Flew Over the Cuckoo's Nest (1975)  4.22913497743311    40103
Fight Club (1999)   4.2307160469145675  60024
12 Angry Men (1957) 4.231208570075758   16896
Rear Window (1954)  4.232552144363722   21335
Seven Samurai (Shichinin no samurai) (1954) 4.255073602972702   13994
"Godfather: Part II, The (1974)"    4.263475012950189   36679
Human (2015)    4.264705882352941   34
Schindler's List (1993) 4.266530696698294   67662
Human Planet (2011) 4.271573604060913   197
"Usual Suspects, The (1995)"    4.300188962561792   59271
O Pátio das Cantigas (1942) 4.3076923076923075  13
Welfare (1975)  4.318181818181818   11
"Godfather, The (1972)" 4.339810758717364   57070
Planet Earth II (2016)  4.352631578947369   95
Band of Brothers (2001) 4.394366197183099   284
"Shawshank Redemption, The (1994)"  4.429014514393623   91082
Planet Earth (2006) 4.478779840848806   754
```

Twitter Emerging Topic Output

```
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/oQxjQ9f7HE, hashtags -> [Ljava.lang.String;@62ae99fa, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/zWgFjNoZgh, hashtags -> [Ljava.lang.String;@19162025, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/4sfMu2VmJJ, hashtags -> [Ljava.lang.String;@f1d252e, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/XLzliC9PWY, hashtags -> [Ljava.lang.String;@4c04db2d, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/7Qwa3guifs, hashtags -> [Ljava.lang.String;@5a73fa1e, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/6h8CMIOSHT, hashtags -> [Ljava.lang.String;@72ccee7d, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/rXAJwNRqFp, hashtags -> [Ljava.lang.String;@149e915f, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/XGaEekDFqI, hashtags -> [Ljava.lang.String;@7e29ed52, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/9sIR5Vzn5h, hashtags -> [Ljava.lang.String;@3b4f5b83, language -> en, sentiment -> NEGATIVE)
Map(text -> I just pulled off this play in front of millions of people worldwide.

#FortniteE3 https://t.co/67wI8s8lSg, hashtags -> [Ljava.lang.String;@2e65e591, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/EtFyrLKkul, hashtags -> [Ljava.lang.String;@4e118d6b, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/G4a5am8XWu, hashtags -> [Ljava.lang.String;@7fe3aa86, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/31rCMwOuAp, hashtags -> [Ljava.lang.String;@3e3175ba, language -> en, sentiment -> NEGATIVE)
Map(text -> RT @Valkyrae: Happy to be here competing for a good cause! Such an honor.
@FortniteGame #FortniteE3 https://t.co/w056mBJveF, hashtags -> [Ljava.lang.String;@3ff95680, language -> en, sentiment -> NEUTRAL)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/j87mdxBYyr, hashtags -> [Ljava.lang.String;@7dc1eec1, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/DnwJwUiouI, hashtags -> [Ljava.lang.String;@1f14dc65, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/c9KyvP4scy, hashtags -> [Ljava.lang.String;@2a3df698, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/ZRl18FIub4, hashtags -> [Ljava.lang.String;@5f05e809, language -> en, sentiment -> NEGATIVE)
Map(text -> RT @NoahJ456: I just pulled off this play in front of millions of people worldwide.

#FortniteE3 https://t.co/67wI8s8lSg, hashtags -> [Ljava.lang.String;@2d40b2ce, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/BBABA059DV, hashtags -> [Ljava.lang.String;@148ea4f7, language -> en, sentiment -> NEGATIVE)
Map(text -> RT @NoahJ456: I just pulled off this play in front of millions of people worldwide.

#FortniteE3 https://t.co/67wI8s8lSg, hashtags -> [Ljava.lang.String;@51c209c8, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/NthYUMC2k7, hashtags -> [Ljava.lang.String;@21f7e659, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/WrxQvW2K90, hashtags -> [Ljava.lang.String;@989f66b, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/vdmYiEDuvw, hashtags -> [Ljava.lang.String;@68c152b1, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/DhEu6K7jtX, hashtags -> [Ljava.lang.String;@290c9a28, language -> en, sentiment -> NEGATIVE)
Map(text -> GOOOO 2eme game ! #FortniteE3, hashtags -> [Ljava.lang.String;@3757747e, language -> en, sentiment -> NEUTRAL)
Map(text -> RT @NoahJ456: I just pulled off this play in front of millions of people worldwide.

#FortniteE3 https://t.co/67wI8s8lSg, hashtags -> [Ljava.lang.String;@71abf651, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/cjZosJFLco, hashtags -> [Ljava.lang.String;@139d760f, language -> en, sentiment -> NEGATIVE)
Map(text -> I entered the #FortniteE3 VIP swag bag giveaway! https://t.co/vmFQftnxI9, hashtags -> [Ljava.lang.String;@5e0d738e, language -> en, sentiment -> NEGATIVE)
```

As evidenced in this output for #ForniteE3, many of the tweets share the same content. It appears there was a sweepstakes that required a tweet for entry, with an easily available auto-generated message. However, you will notice that since the hashtags array of strings is not printed out, they're memory locations are printed instead, showing these are in fact unique tweets even though they share the same content. To make this more obvious, we reintroduced the user field to our mapped tweets so it is more obvious. An example of the new option can be found below for topic #BTSxCorden.

```
Map(sentiment -> NEUTRAL, text -> RT @aegyotaetae: baby boy culture 🥺 #BTSxCorden https://t.co/8UJe4PcNEf, language -> en, user -> BTSmusicNsujin, hashtags ->
[Ljava.lang.String;@6e757c79)
Map(sentiment -> NEGATIVE, text -> Jimin didn't open the door to green room HE OPENED IT TO HELL #BTSxCorden, language -> en, user -> jinddicted, hashtags ->
[Ljava.lang.String;@269a250f)
Map(sentiment -> NEGATIVE, text -> RT @latelateshow: BTS HAVE ARRIVED TO THE #LATELATESHOW! #BTSxCorden https://t.co/9qqck1Z00Z, language -> en, user -> kymmb11, hashtags ->
[Ljava.lang.String;@307cf944)
Map(sentiment -> NEUTRAL, text -> #BTSxCorden MY BABES LOOK AMAZINGGG, language -> en, user -> Isabellaxvy, hashtags -> [Ljava.lang.String;@13125ab3)
Map(sentiment -> NEUTRAL, text -> RT @btsvotingteam: #BTSxCorden in few minutes! 🔔
@BTS_twt, language -> en, user -> linhtae_1230, hashtags -> [Ljava.lang.String;@56417a1c)
Map(sentiment -> NEGATIVE, text -> RT @ilsanspjm: WHY DID NAMJOON SAY WELCOME BACK TO JAMES CORDEN ON HIS OWN SHOW MY LUNGS

#BTSxCorden (@BTS_twt)

https://t.co/c4hCP0T6o9, language -> en, user -> SummerLoveFSK, hashtags -> [Ljava.lang.String;@21048f86)
Map(sentiment -> POSITIVE, text -> RT @babyjincafe: seokjin's cute little "hello" and wave,, baby:( #BTSxCorden https://t.co/kUrsIiysUv, language -> en, user -> chillsunicorn, hashtags
-> [Ljava.lang.String;@7665e43b)
```

Performance Analysis

Below are the times recorded to run both queries from Part 1 in our MapReduce implementation, as well as our Spark RDD implementation. The programs were run on the large dataset. We used real time as a comparison because it is the most obvious measure of performance experienced by the programmer. Real time measurements for the MapReduce jobs were extracted from the counters Hadoop tracks. For Spark, real time was calculated using the Linux `time` command.

Framework	Query	Real Time
MapReduce	Query 1	343.130s
	Query 2	305.740s
Spark	Query 1	55.496s
	Query 2	46.015s

It is clear, the Spark implementations greatly outperform the MapReduce jobs. This difference in execution time is due largely to Spark storing intermediate data representations in memory and limiting the amount of I/O calls. This allows Spark to finish jobs much faster since I/O is the largest bottleneck.

To further document the Spark behavior, we also recorded the amount of CPU time spent on all tasks for the Spark programs. These times are divided into time spent on computations in tasks, time spent sorting intermediate results, and total execution time.

Query	Computations	Sorting	Total
Query 1	51.305s	50.665s	101.970s
Query 2	70.002s	40.125s	110.127s

From this table, we can see significant time was spent on sorting the results. In query 1 for instance, almost half of the total execution time is spent sorting results. This highlights just how intensive sorting results is for a given program. Query 2 uses a smaller portion of its total execution time for sorting. This result conforms to expectations since there are significantly fewer records produced for query 2 than query 1; 387 records compared to 45,115 records.