

# CS5542 Big Data Apps and Analytics

## LAB ASSIGNMENT #7

### REPORT and SCREEN SHOTS

*Submitted by Rakesh Reddy Bandi(02), Mark J Schultz(26)*

#### 1) Add Sentimental Analysis using Twitter Streaming (related to project)

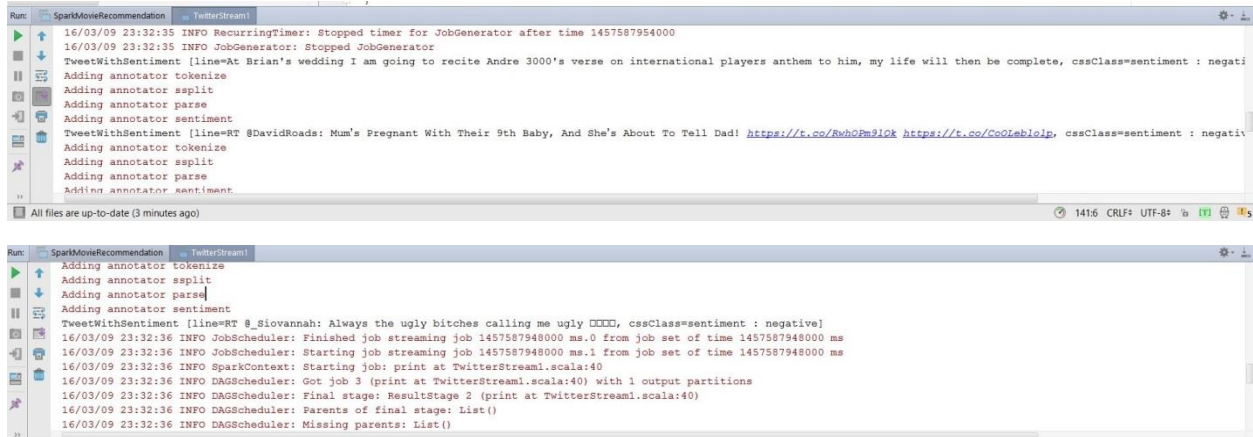
We begin by streaming data from Twitter. We use the filter feature in spark to only include Tweets in English. We then use the sentiment analyzer to classify sentiment into negative and positive connotations.

We've provided an example here.



```
Run: SparkMovieRecommendation TwitterStreaml
Adding annotator parse
Adding annotator sentiment
TweetWithSentiment [line=RT @5808: 1 Night Only // Hong Kong, Asia World Expo // SLFL xx https://t.co/igqgqd2ch, cssClass=sentiment : negative]
16/03/09 23:32:34 INFO TwitterReceiver: Twitter receiver stopped
16/03/09 23:32:34 INFO ReceiverSupervisorImpl: Called receiver onStop
16/03/09 23:32:34 INFO ReceiverSupervisorImpl: Deregistering receiver 0
16/03/09 23:32:34 ERROR ReceiverTracker: Deregistered receiver for stream 0: Stopped by driver
16/03/09 23:32:34 INFO ReceiverSupervisorImpl: Stopped receiver 0
16/03/09 23:32:34 INFO BlockGenerator: Stopping BlockGenerator
TweetWithSentiment [line=RT @FijiBoiDio: I'm trying to be successful and make sure my family all straight and I change the world positively, cssClass=sentiment : negative]
Adding annotator tokenize
Adding annotator sentiment
```

We show several more examples.



```
Run: SparkMovieRecommendation TwitterStreaml
16/03/09 23:32:35 INFO RecurringTimer: Stopped timer for JobGenerator after time 1457587954000
16/03/09 23:32:35 INFO JobGenerator: Stopped JobGenerator
TweetWithSentiment [line=At Brian's wedding I am going to recite Andre 3000's verse on international players anthem to him, my life will then be complete, cssClass=sentiment : negati
Adding annotator tokenize
Adding annotator split
Adding annotator parse
Adding annotator sentiment
TweetWithSentiment [line=RT @DavidRoads: Mum's Pregnant With Their 9th Baby, And She's About To Tell Dad! https://t.co/RvhQm9lOk https://t.co/Ce0Leblolp, cssClass=sentiment : negati
Adding annotator tokenize
Adding annotator split
Adding annotator parse
Adding annotator sentiment

All files are up-to-date (3 minutes ago) 141:6 CRLF UTF-8 15

Run: SparkMovieRecommendation TwitterStreaml
Adding annotator tokenize
Adding annotator split
Adding annotator parse
Adding annotator sentiment
TweetWithSentiment [line=RT @Siovannah: Always the ugly bitches calling me ugly [REDACTED], cssClass=sentiment : negative]
16/03/09 23:32:36 INFO JobScheduler: Finished job streaming job 1457587948000 ms.0 from job set of time 1457587948000 ms
16/03/09 23:32:36 INFO JobScheduler: Starting job streaming job 1457587948000 ms.1 from job set of time 1457587948000 ms
16/03/09 23:32:36 INFO SparkContext: Starting job: print at TwitterStreaml.scala:40
16/03/09 23:32:36 INFO DAGScheduler: Got job 3 (print at TwitterStreaml.scala:40) with 1 output partitions
16/03/09 23:32:36 INFO DAGScheduler: Final stage: ResultStage 2 (print at TwitterStreaml.scala:40)
16/03/09 23:32:36 INFO DAGScheduler: Parents of final stage: List()
16/03/09 23:32:36 INFO DAGScheduler: Missing parents: List()
```

## 2) Make Recommendations (related to project)

For this section, we utilized the SparkMovieReccomendation.scala code to recommend restaurants to the user. Using 25 handpicked restaurants, we picked the top 10 restaurants recommended based on our user's preferences.

We partitioned the data into training (60%), testing (20%), and validation (20%). The training set uses the last digit of the time step to provide a random number between 1 and 9.

```
val numPartitions = 4
val training = ratings.filter(x => x._1 < 6)
    .values
    .union(myRatingsRDD)
    .repartition(numPartitions)
    .cache()
val validation = ratings.filter(x => x._1 >= 6 && x._1 < 8)
    .values
    .repartition(numPartitions)
    .cache()
val test = ratings.filter(x => x._1 >= 8).values.cache()

val numTraining = training.count()
val numValidation = validation.count()
val numTest = test.count()

println("Training: " + numTraining + ", validation: " + numValidation + ", test: " + numTest)
```

We then train the model and optimize to find the best training model:

```
// train models and evaluate them on the validation set

val ranks = List(8, 12)
val lambdas = List(0.1, 10.0)
val numIters = List(10, 20)
var bestModel: Option[MatrixFactorizationModel] = None
var bestValidationRmse = Double.MaxValue
var bestRank = 0
var bestLambda = -1.0
var bestNumIter = -1
for (rank <- ranks; lambda <- lambdas; numIter <- numIters) {
    val model = ALS.train(training, rank, numIter, lambda)
    val validationRmse = computeRmse(model, validation, numValidation)
    println("RMSE (validation) = " + validationRmse + " for the model trained with rank = "
        + rank + ", lambda = " + lambda + ", and numIter = " + numIter + ".")
    if (validationRmse < bestValidationRmse) {
        bestModel = Some(model)
        bestValidationRmse = validationRmse
        bestRank = rank
        bestLambda = lambda
        bestNumIter = numIter
    }
}
```

This screenshot shows the iterations of training:

```
RMSE (validation) = 5.29230621162741 for the model trained with rank = 8, lambda = 0.1, and numIter = 10.  
RMSE (validation) = 5.463701808903616 for the model trained with rank = 8, lambda = 0.1, and numIter = 20.  
RMSE (validation) = 6.475417707203035 for the model trained with rank = 8, lambda = 10.0, and numIter = 10.  
RMSE (validation) = 6.475417707203036 for the model trained with rank = 8, lambda = 10.0, and numIter = 20.  
RMSE (validation) = 4.353967647451537 for the model trained with rank = 12, lambda = 0.1, and numIter = 10.  
RMSE (validation) = 4.6815966390435815 for the model trained with rank = 12, lambda = 0.1, and numIter = 20.  
RMSE (validation) = 6.475417707203036 for the model trained with rank = 12, lambda = 10.0, and numIter = 10.  
RMSE (validation) = 6.475417707203036 for the model trained with rank = 12, lambda = 10.0, and numIter = 20.
```

The final recommendations:

```
The best model was trained with rank = 12 and lambda = 0.1, and numIter = 10, and its RMSE on the test set is 5.420072919734174.  
The best model improves the baseline by -392.67%.  
Restaurants recommended for you:  
1: Ruby Tuesday  
2: Panera  
3: IHOP  
4: Pizza Hut  
5: Taco Bell  
6: KFC  
7: Red Lobster  
8: Jack Stack  
9: Five Guys  
10: Burger King
```

### 3) Recommendation sent to smartphone

After creating the recommendations above, our goal is to send these recommendations to the User. Therefore, we have taken this recommendation output, and sent it as a notification to the smartphone. We show the screenshot below:

