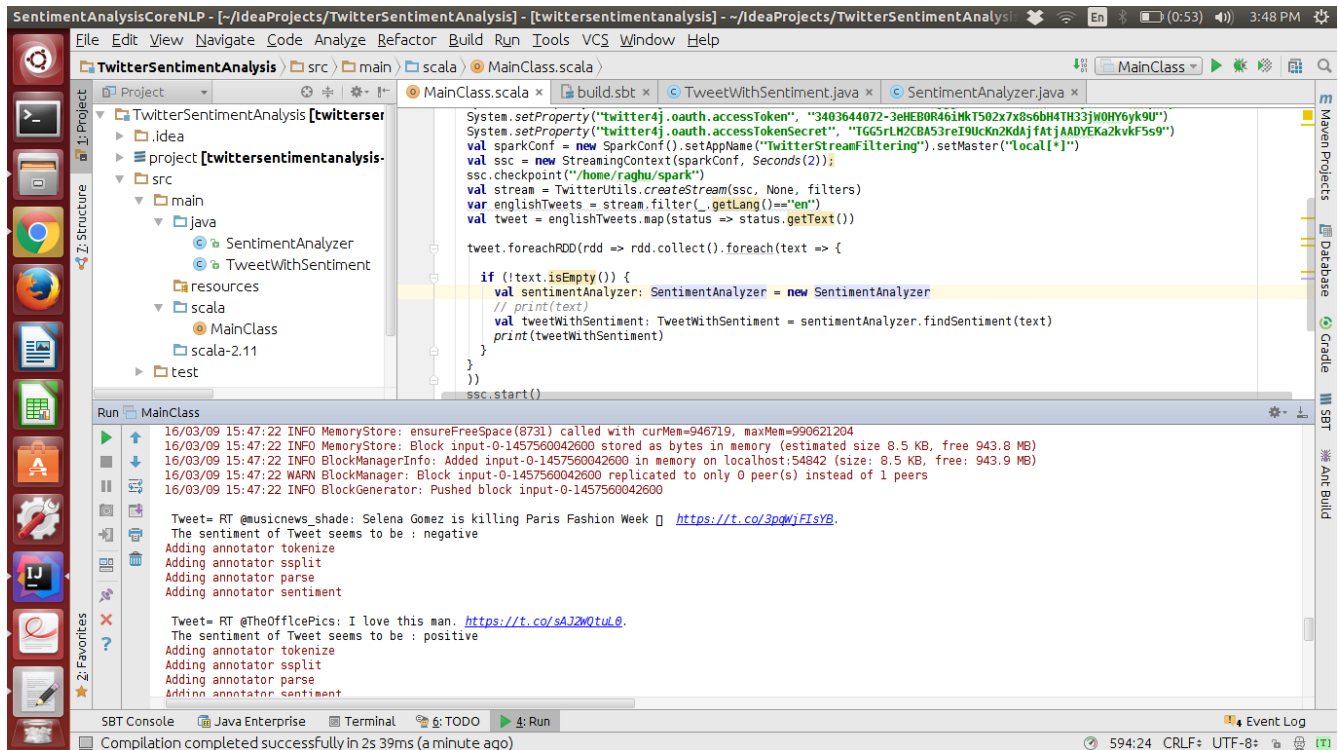


Lab 7 Assignment

Task 1:

The task here was to perform sentiment analysis on Twitter streaming data. I performed the following steps to complete the task.

1. Create Twitter stream using Twitter4j package.
2. Filter the Tweets where the language is English.
3. Extract only the Tweet text from the Tweet.
4. For every Tweet text perform sentiment analysis using the Stanford NLP class.
5. Output the Tweet Text along with the sentiment of the Tweet text.



The screenshot displays an IDE window for a project named "SentimentAnalysisCoreNLP". The project structure on the left includes a "src" directory with "main" and "test" subdirectories. The "main" directory contains "SentimentAnalyzer", "TweetWithSentiment", "resources", "scala", and "MainClass". The "scala" directory contains "MainClass.scala" and "scala-2.11". The "MainClass.scala" file is open in the editor, showing the following code:

```
System.setProperty("twitter4j.oauth.accessToken", "3403644072-3eHEB0R46iMkT502x7x8s6bH4TH33jW0HY6yk9U")
System.setProperty("twitter4j.oauth.accessTokenSecret", "TG65rLH2CBA53reI9Uckn2KdJfAtjAADYEXa2kvkF5s9")
val sparkConf = new SparkConf().setAppName("TwitterStreamFiltering").setMaster("local[*]")
val ssc = new StreamingContext(sparkConf, Seconds(2))
ssc.checkpoint("/home/raghu/spark")
val stream = TwitterUtils.createStream(ssc, None, filters)
val englishTweets = stream.filter(_.getLang() == "en")
val tweet = englishTweets.map(status => status.getText())

tweet.foreachRDD(rdd => rdd.collect().foreach(text => {
    if (!text.isEmpty()) {
        val sentimentAnalyzer: SentimentAnalyzer = new SentimentAnalyzer
        // print(text)
        val tweetWithSentiment: TweetWithSentiment = sentimentAnalyzer.findSentiment(text)
        print(tweetWithSentiment)
    }
}))
ssc.start()
```

The "Run" button is highlighted, and the "Run" console at the bottom shows the following output:

```
16/03/09 15:47:22 INFO MemoryStore: ensureFreeSpace(8731) called with curMem=946719, maxMem=990621204
16/03/09 15:47:22 INFO MemoryStore: Block input-0-1457560042600 stored as bytes in memory (estimated size 8.5 KB, free 943.8 MB)
16/03/09 15:47:22 INFO BlockManagerInfo: Added input-0-1457560042600 in memory on localhost:54842 (size: 8.5 KB, free: 943.9 MB)
16/03/09 15:47:22 WARN BlockManager: Block input-0-1457560042600 replicated to only 0 peer(s) instead of 1 peers
16/03/09 15:47:22 INFO BlockGenerator: Pushed block input-0-1457560042600

Tweet= RT @musicnews_shade: Selena Gomez is killing Paris Fashion Week https://t.co/3pQWjFt5YB.
The sentiment of Tweet seems to be : negative
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment

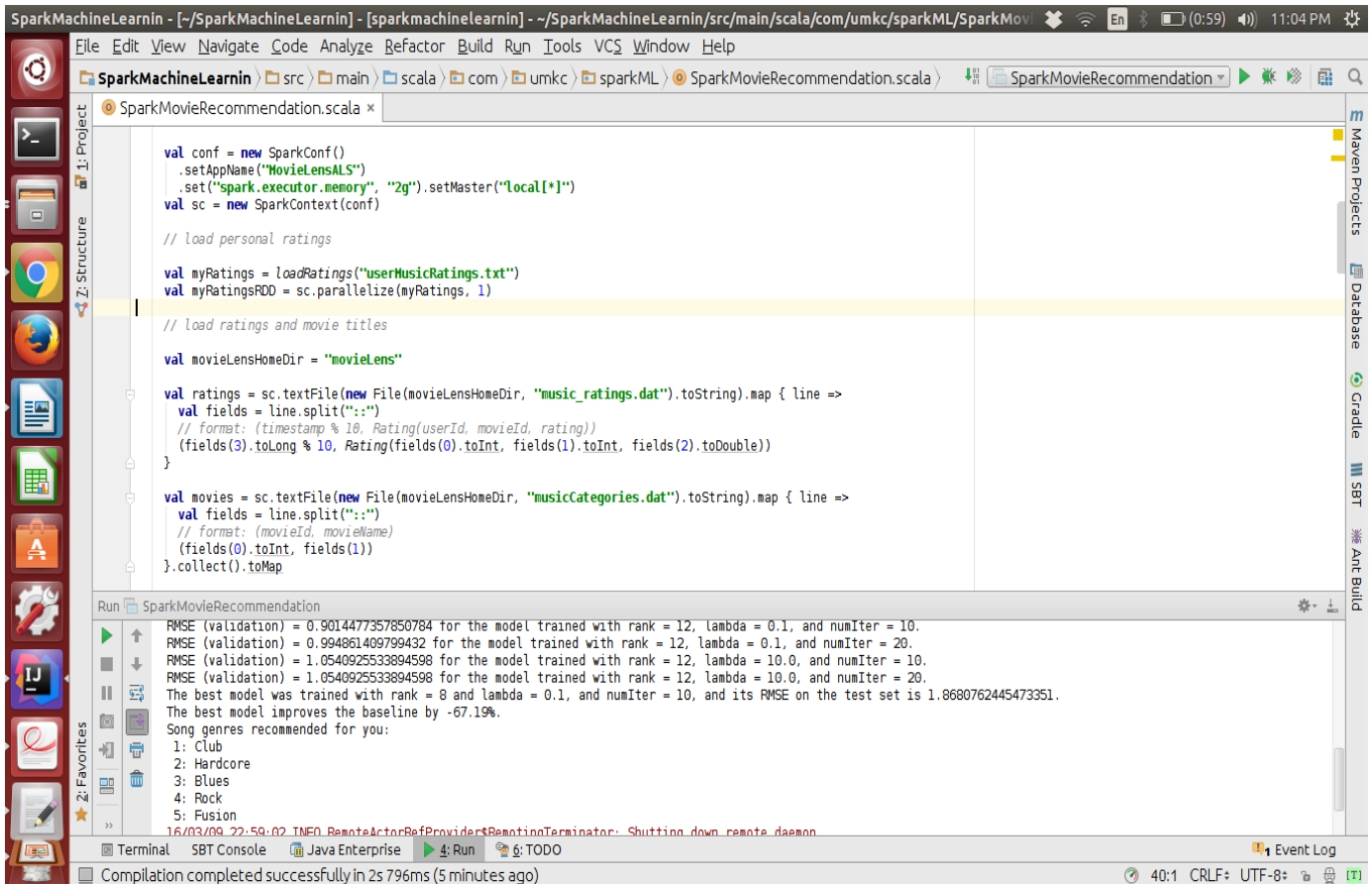
Tweet= RT @TheOfficePics: I love this man. https://t.co/SA72W0tUd0.
The sentiment of Tweet seems to be : positive
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
```

The status bar at the bottom indicates "Compilation completed successfully in 2s 39ms (a minute ago)" and "594/24 CRLF: UTF-8".

Task 2:

This task was to suggest recommendations to the user based on his ratings for music genres. The following steps were performed to complete the task.

1. Create training data through Twitter stream filtering and classification of data.
2. Manually formatting the collected data into the desired format.
3. Performing collaborative filtering on the training data to train the model.
4. Get sample data from Twitter to serve as test data.
5. Perform recommendation for the test data



The screenshot shows an IDE window titled "SparkMachineLearnin" with a file explorer on the left and a run console at the bottom. The main editor displays the file "SparkMovieRecommendation.scala". The code defines a SparkConf, loads ratings and movie titles from local files, and performs collaborative filtering. The run console shows the results of the training and testing process, including RMSE values and recommended song genres.

```
val conf = new SparkConf()
    .setAppName("MovieLensALS")
    .set("spark.executor.memory", "2g").setMaster("local[*]")
val sc = new SparkContext(conf)

// load personal ratings
val myRatings = loadRatings("userMusicRatings.txt")
val myRatingsRDD = sc.parallelize(myRatings, 1)

// load ratings and movie titles
val movieLensHomeDir = "movieLens"

val ratings = sc.textFile(new File(movieLensHomeDir, "music_ratings.dat")).map { line =>
    val fields = line.split(":::")
    // format: (timestamp % 10, Rating(userId, movieId, rating))
    (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble))
}

val movies = sc.textFile(new File(movieLensHomeDir, "musicCategories.dat")).map { line =>
    val fields = line.split(":::")
    // format: (movieId, movieName)
    (fields(0).toInt, fields(1))
}.collect().toMap
```

Run SparkMovieRecommendation

RMSE (validation) = 0.9014477357850784 for the model trained with rank = 12, lambda = 0.1, and numIter = 10.
RMSE (validation) = 0.994861409799432 for the model trained with rank = 12, lambda = 0.1, and numIter = 20.
RMSE (validation) = 1.0540925533894598 for the model trained with rank = 12, lambda = 10.0, and numIter = 10.
RMSE (validation) = 1.0540925533894598 for the model trained with rank = 12, lambda = 10.0, and numIter = 20.
The best model was trained with rank = 8 and lambda = 0.1, and numIter = 10, and its RMSE on the test set is 1.0680762445473351.
The best model improves the baseline by -67.19%.

Song genres recommended for you:

- 1: Club
- 2: Hardcore
- 3: Blues
- 4: Rock
- 5: Fusion

16/03/09 22:59:02 INFO RemoteActorRefProvider\$RemoteTerminator: Shutting down remote daemon

Terminal SBT Console Java Enterprise Run TODO

Compilation completed successfully in 2s 796ms (5 minutes ago)

40:1 CRLF UTF-8

Task 3:

The task here was to send the recommendations to phone through socket connection. The following steps were performed to accomplish the task.

1. Create a socket server at Android phone side.
2. Create a client socket for the recommendation process that provides recommendations.
3. Send the recommendations as string over the socket from the client to the server.
4. Display the received string on the UI and the Android client end.

