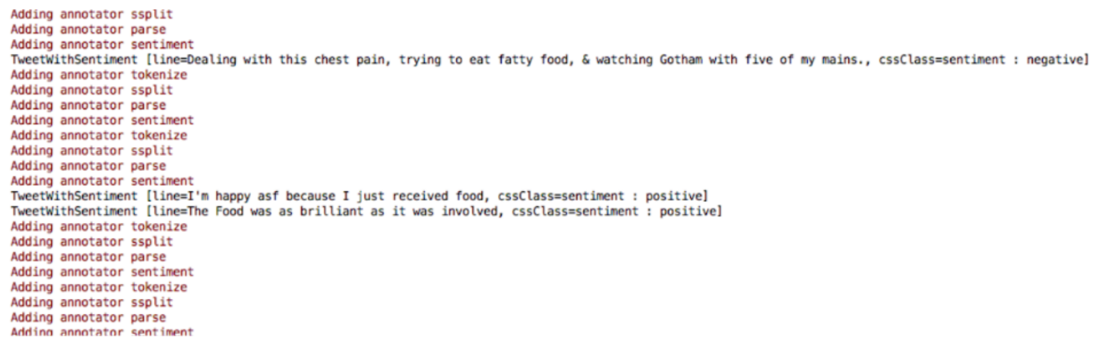
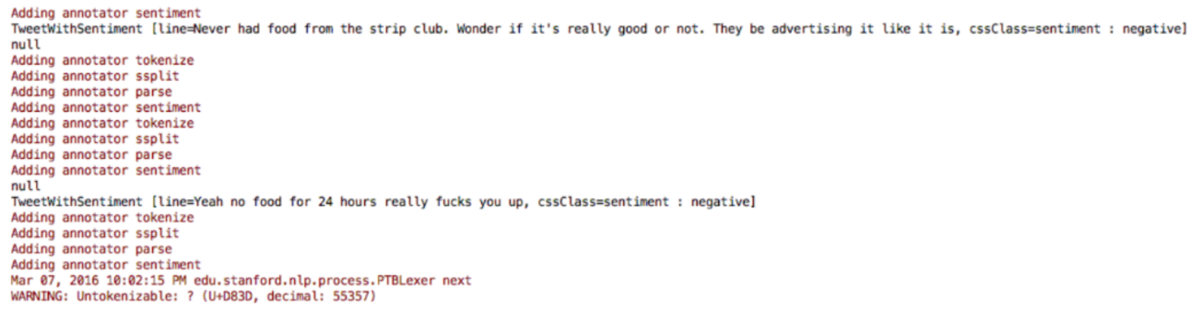
Mohamed Gharibi (7)

This code will collect the tweets depends on the hashtag specified, for example here we used the word “food”. **TwitterSentimentAnalysis** class will collect the tweets “20” and the **SentimentAnalyzer** will do the sentiment analysis for the tweets wich had been collected. The results will be printed in the log.



```
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
TweetWithSentiment [line=Dealing with this chest pain, trying to eat fatty food, & watching Gotham with five of my mains., cssClass=sentiment : negative]
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
TweetWithSentiment [line=I'm happy asf because I just received food, cssClass=sentiment : positive]
TweetWithSentiment [line=The Food was as brilliant as it was involved, cssClass=sentiment : positive]
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
```



```
Adding annotator sentiment
TweetWithSentiment [line=Never had food from the strip club. Wonder if it's really good or not. They be advertising it like it is, cssClass=sentiment : negative]
null
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
null
TweetWithSentiment [line=Yeah no food for 24 hours really fucks you up, cssClass=sentiment : negative]
Adding annotator tokenize
Adding annotator ssplit
Adding annotator parse
Adding annotator sentiment
Mar 07, 2016 10:02:15 PM edu.stanford.nlp.process.PTBLexer next
WARNING: Untokenizable: ? (U+D83D, decimal: 55357)
```

Question 2:

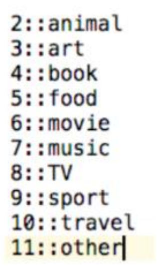
Training data: Twitter streaming/ categorized data (The categorization will be from lab 5&6).

Testing data: UserId, category, rating.

Rating data based on sentiment analysis, (retweet count).

Expected outcome is to make recommendation based on the user profile (e.g. location, gender and age).

Define 10 categories:



```
2::animal
3::art
4::book
5::food
6::movie
7::music
8::TV
9::sport
10::travel
11::other
```

Collect tweets as training data to categorize the tweets into these categories based on the keyword searching.

To do this we had to collect like this:

1. UserId: Tweets userID should converted to integer.
2. Category: It is analysed using TF-IDF.
3. Rating: using the sentiment analysis to provide the rating.
4. Timestamp: this time should be converted into integer as well.

Four items should be recorded in the test data:

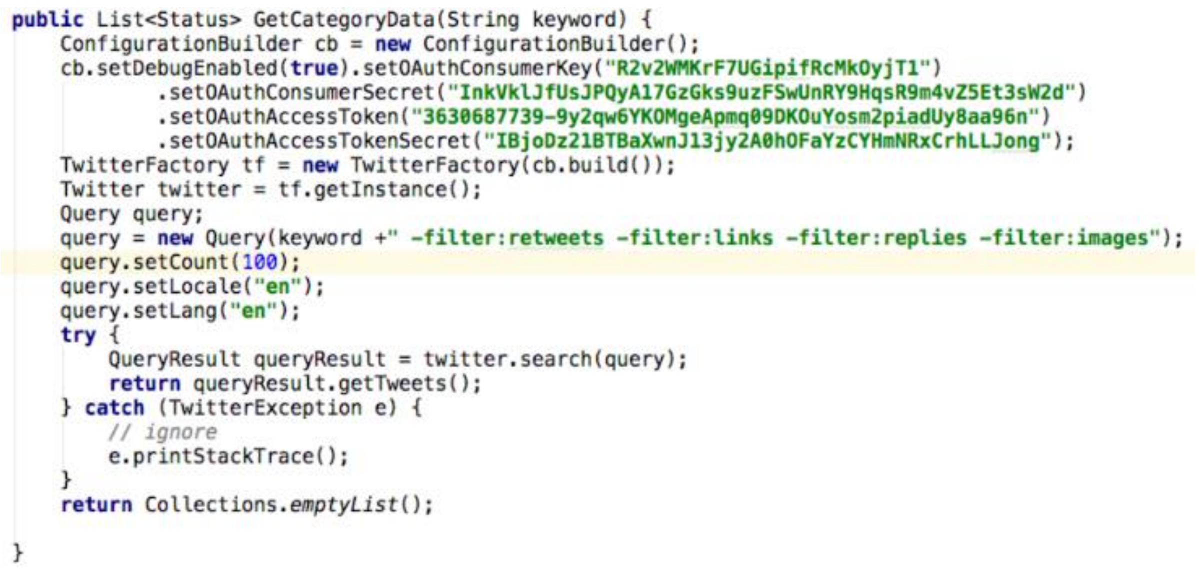
1. UserId: since the data is collected by a device, the userId specified as "8888".
2. Category: analysed by FT-IDF.
3. Rating: using the sentiment analysis.
4. Timestamp: Get the time user's input.

The four items should be written in a file called "rating.txt"

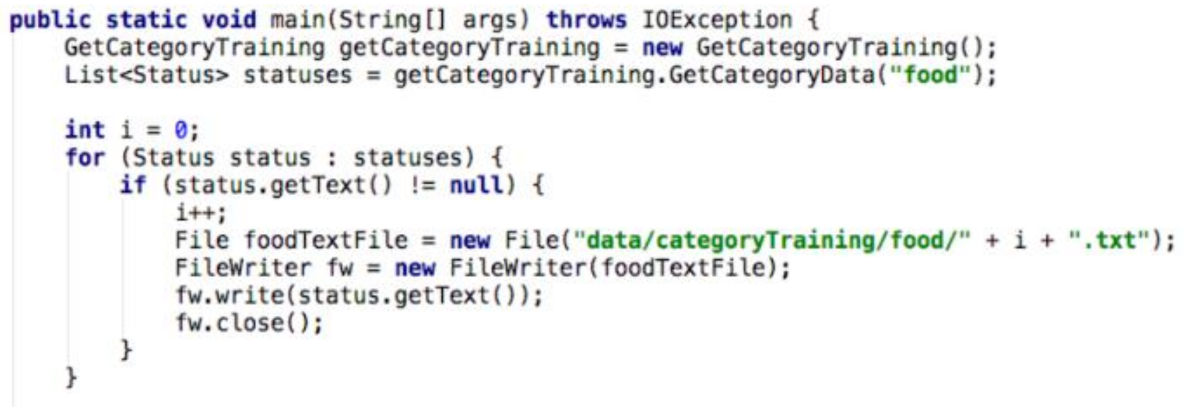
Then, we get the category mapping file which called "category.txt"

After that, we get the recommendation for a particular user.

Finally, send the results to the smartphone.

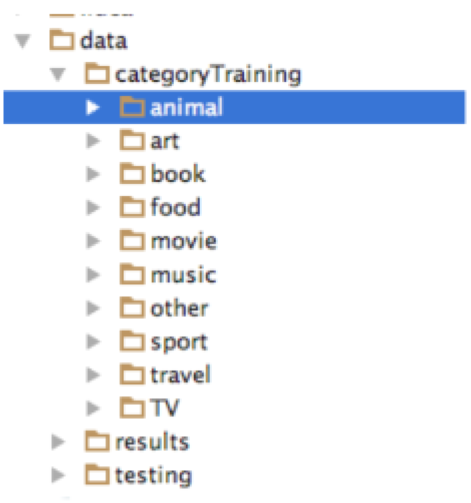


```java
public List<Status> GetCategoryData(String keyword) {
    ConfigurationBuilder cb = new ConfigurationBuilder();
    cb.setDebugEnabled(true).setOAuthConsumerKey("R2v2WMKrF7UGipifRcMkOyjT1")
            .setOAuthConsumerSecret("InkVklJfUsJPQyA17GzGks9uzFSwUnRY9HqsR9m4vZ5Et3sW2d")
            .setOAuthAccessToken("3630687739-9y2qw6YKOMgeApmq09DKOuYosm2piadUy8aa96n")
            .setOAuthAccessTokenSecret("IBjoDz21BTBaXwnJ13jy2A0hOFaYzCYHmNRxCrhLLJong");
    TwitterFactory tf = new TwitterFactory(cb.build());
    Twitter twitter = tf.getInstance();
    Query query;
    query = new Query(keyword +" -filter:retweets -filter:links -filter:replies -filter:images");
    query.setCount(100);
    query.setLocale("en");
    query.setLang("en");
    try {
        QueryResult queryResult = twitter.search(query);
        return queryResult.getTweets();
    } catch (TwitterException e) {
        // ignore
        e.printStackTrace();
    }
    return Collections.emptyList();

}
```



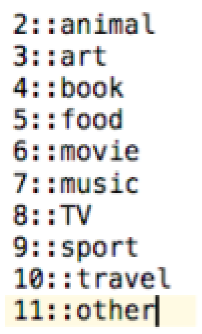
```java
public static void main(String[] args) throws IOException {
    GetCategoryTraining getCategoryTraining = new GetCategoryTraining();
    List<Status> statuses = getCategoryTraining.GetCategoryData("food");

    int i = 0;
    for (Status status : statuses) {
        if (status.getText() != null) {
            i++;
            File foodTextFile = new File("data/categoryTraining/food/" + i + ".txt");
            FileWriter fw = new FileWriter(foodTextFile);
            fw.write(status.getText());
            fw.close();
        }
    }
```

File Structure:



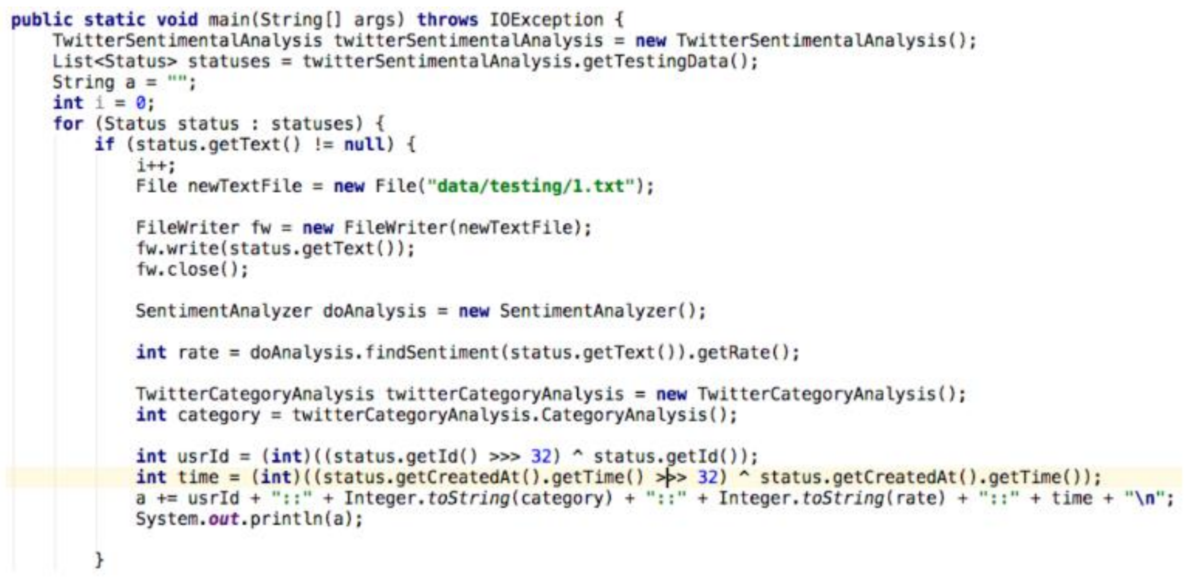
```
▼ data
    ▼ categoryTraining
        ▶ animal
        ▶ art
        ▶ book
        ▶ food
        ▶ movie
        ▶ music
        ▶ other
        ▶ sport
        ▶ travel
        ▶ TV
    ▶ results
    ▶ testing
```

Category mapping:

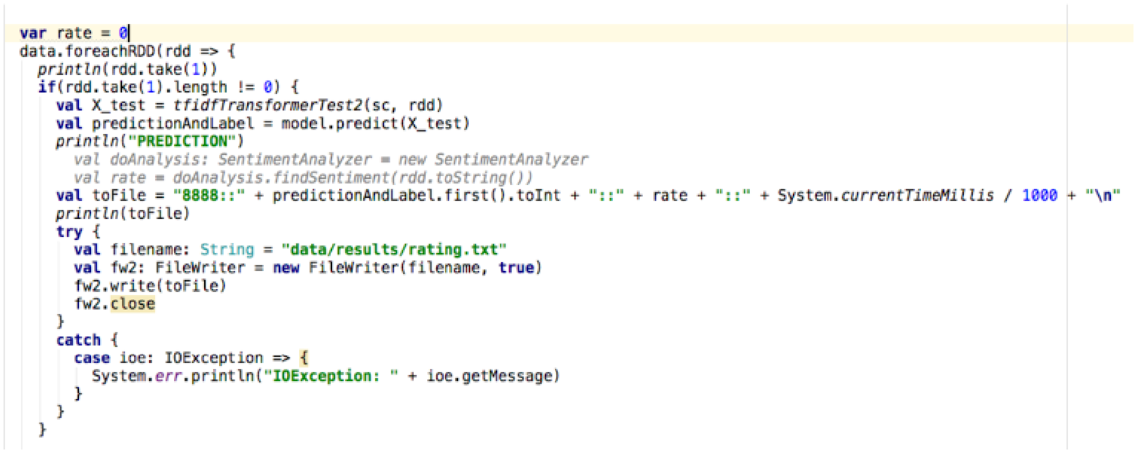


```
2::animal
3::art
4::book
5::food
6::movie
7::music
8::TV
9::sport
10::travel
11::other
```

Do the sentiment analysis and the category analysis from the collected tweets:

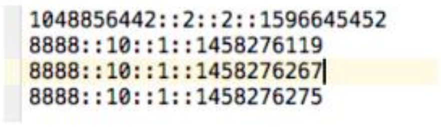


```java
public static void main(String[] args) throws IOException {
    TwitterSentimentalAnalysis twitterSentimentalAnalysis = new TwitterSentimentalAnalysis();
    List<Status> statuses = twitterSentimentalAnalysis.getTestingData();
    String a = "";
    int i = 0;
    for (Status status : statuses) {
        if (status.getText() != null) {
            i++;
            File newTextFile = new File("data/testing/1.txt");

            FileWriter fw = new FileWriter(newTextFile);
            fw.write(status.getText());
            fw.close();

            SentimentAnalyzer doAnalysis = new SentimentAnalyzer();

            int rate = doAnalysis.findSentiment(status.getText()).getRate();

            TwitterCategoryAnalysis twitterCategoryAnalysis = new TwitterCategoryAnalysis();
            int category = twitterCategoryAnalysis.CategoryAnalysis();

            int usrId = (int)((status.getId() >>> 32) ^ status.getId());
            int time = (int)((status.getCreatedAt().getTime() >>> 32) ^ status.getCreatedAt().getTime());
            a += usrId + "::" + Integer.toString(category) + "::" + Integer.toString(rate) + "::" + time + "\n";
            System.out.println(a);

        }
```



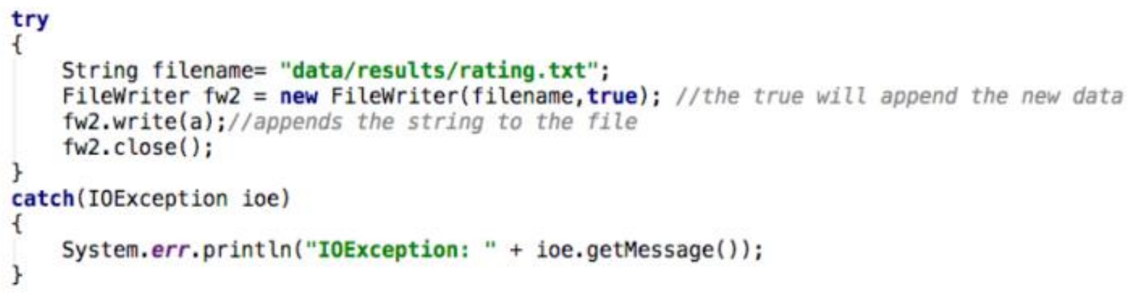
```
var rate = 0
data.foreachRDD(rdd => {
  println(rdd.take(1))
  if(rdd.take(1).length != 0) {
    val X_test = tfidfTransformerTest2(sc, rdd)
    val predictionAndLabel = model.predict(X_test)
    println("PREDICTION")
      val doAnalysis: SentimentAnalyzer = new SentimentAnalyzer
      val rate = doAnalysis.findSentiment(rdd.toString())
    val toFile = "8888::" + predictionAndLabel.first().toInt + "::" + rate + "::" + System.currentTimeMillis / 1000 + "\n"
    println(toFile)
    try {
      val filename: String = "data/results/rating.txt"
      val fw2: FileWriter = new FileWriter(filename, true)
      fw2.write(toFile)
      fw2.close
    }
    catch {
      case ioe: IOException => {
        System.err.println("IOException: " + ioe.getMessage)
      }
    }
  }
```

Data collected by device:



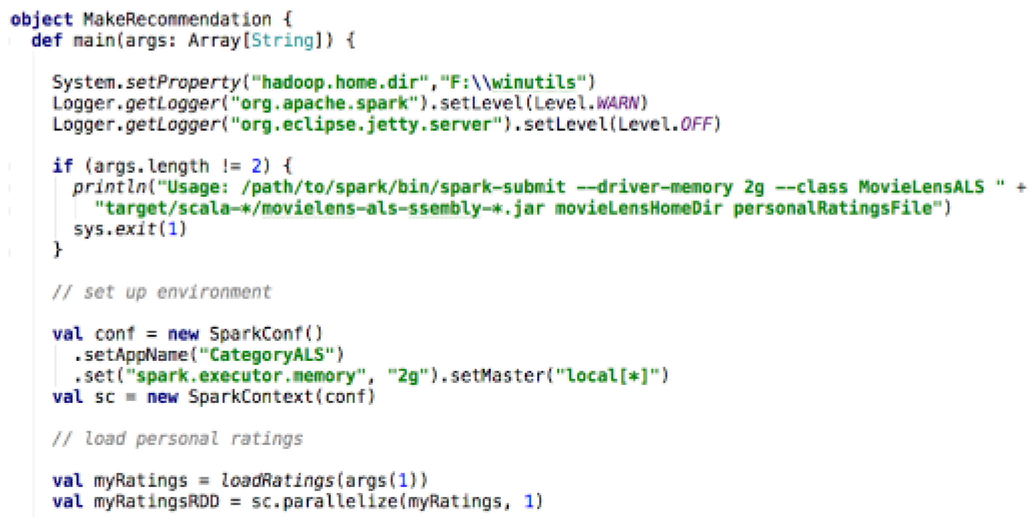
```
1048856442::2::2::1596645452
8888::10::1::1458276119
8888::10::1::1458276267
8888::10::1::1458276275
```

Write the collected data into file:

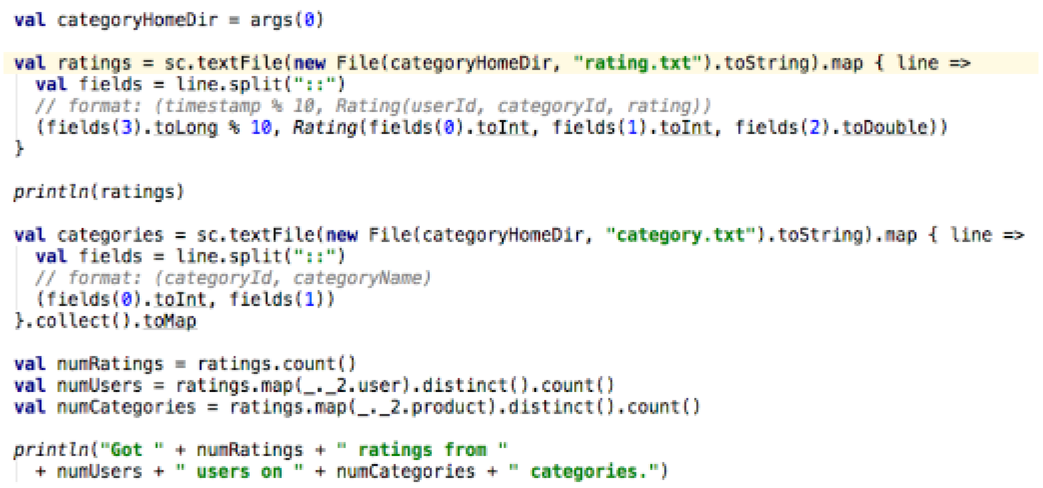


```
try
{
    String filename= "data/results/rating.txt";
    FileWriter fw2 = new FileWriter(filename,true); //the true will append the new data
    fw2.write(a);//appends the string to the file
    fw2.close();
}
catch(IOException ioe)
{
    System.err.println("IOException: " + ioe.getMessage());
}
```

Recommendations of user category:

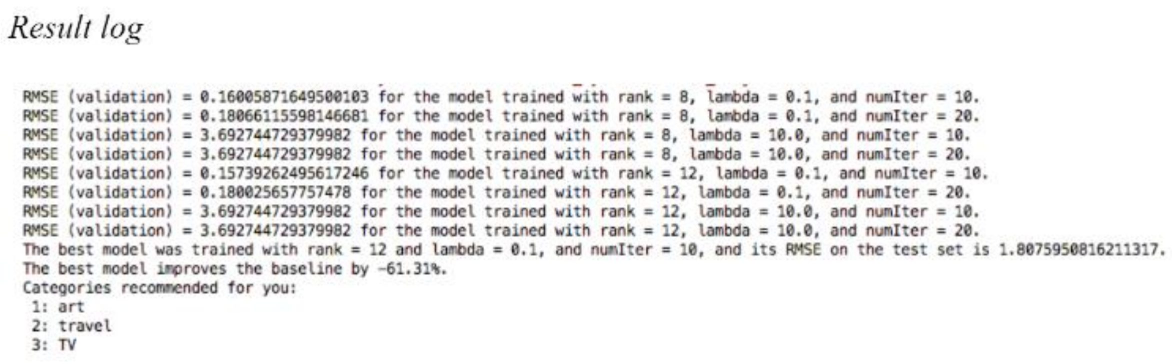


```
object MakeRecommendation {
  def main(args: Array[String]) {

    System.setProperty("hadoop.home.dir","F:\\winutils")
    Logger.getLogger("org.apache.spark").setLevel(Level.WARN)
    Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

    if (args.length != 2) {
      println("Usage: /path/to/spark/bin/spark-submit --driver-memory 2g --class MovieLensALS " +
        "target/scala-*/movielens-als-ssembly-*.jar movieLensHomeDir personalRatingsFile")
      sys.exit(1)
    }

    // set up environment

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

    // load personal ratings

    val myRatings = loadRatings(args(1))
    val myRatingsRDD = sc.parallelize(myRatings, 1)
```



```
val categoryHomeDir = args(0)

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

println(ratings)

val categories = sc.textFile(new File(categoryHomeDir, "category.txt").toString).map { line =>
  val fields = line.split("::")
  // format: (categoryId, categoryName)
  (fields(0).toInt, fields(1))
}.collect().toMap

val numRatings = ratings.count()
val numUsers = ratings.map(_._2.user).distinct().count()
val numCategories = ratings.map(_._2.product).distinct().count()

println("Got " + numRatings + " ratings from "
  + numUsers + " users on " + numCategories + " categories.")
```

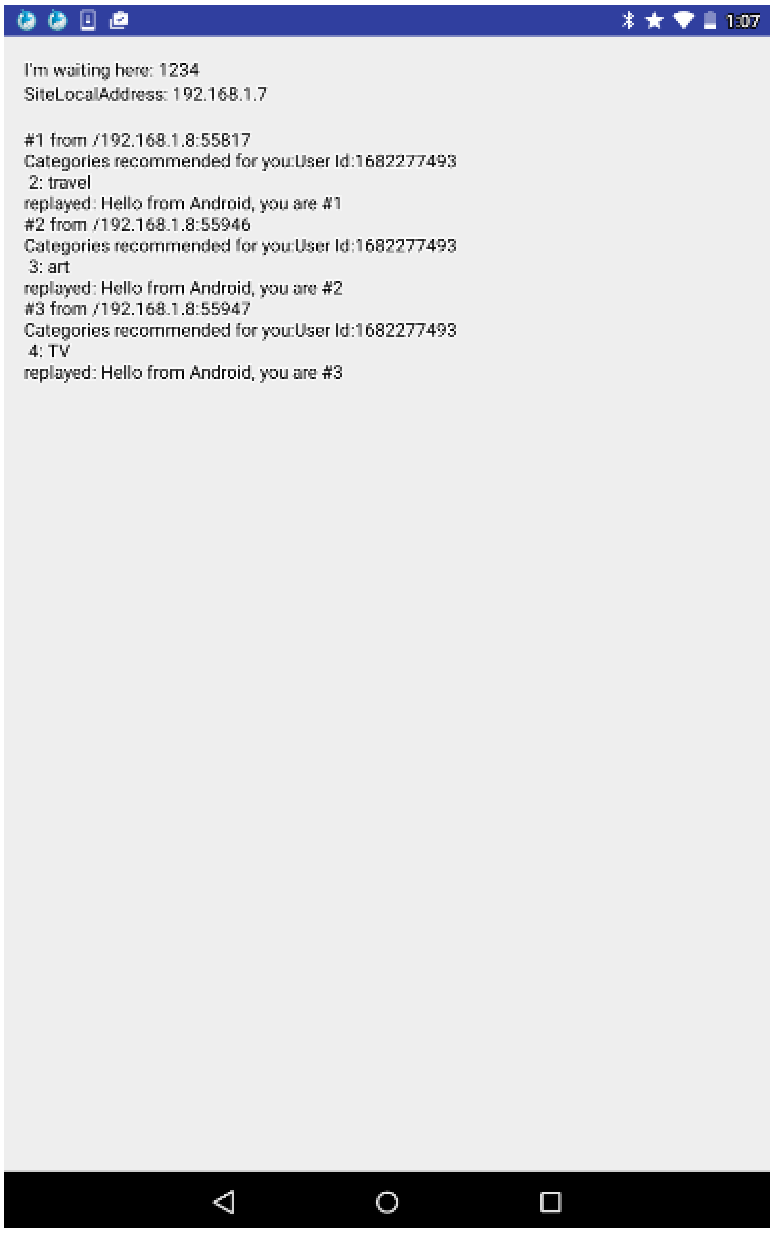
Log result



Result log

```
RMSE (validation) = 0.16005871649500103 for the model trained with rank = 8, lambda = 0.1, and numIter = 10.
RMSE (validation) = 0.18066115598146681 for the model trained with rank = 8, lambda = 0.1, and numIter = 20.
RMSE (validation) = 3.692744729379982 for the model trained with rank = 8, lambda = 10.0, and numIter = 10.
RMSE (validation) = 3.692744729379982 for the model trained with rank = 8, lambda = 10.0, and numIter = 20.
RMSE (validation) = 0.15739262495617246 for the model trained with rank = 12, lambda = 0.1, and numIter = 10.
RMSE (validation) = 0.180025657757478 for the model trained with rank = 12, lambda = 0.1, and numIter = 20.
RMSE (validation) = 3.692744729379982 for the model trained with rank = 12, lambda = 10.0, and numIter = 10.
RMSE (validation) = 3.692744729379982 for the model trained with rank = 12, lambda = 10.0, and numIter = 20.
The best model was trained with rank = 12 and lambda = 0.1, and numIter = 10, and its RMSE on the test set is 1.8075950816211317.
The best model improves the baseline by -61.31%.
Categories recommended for you:
 1: art
 2: travel
 3: TV
```

Result on the smartphone



```
I'm waiting here: 1234
SiteLocalAddress: 192.168.1.7

#1 from /192.168.1.8:55817
Categories recommended for you:User Id:1682277493
 2: travel
replayed: Hello from Android, you are #1
#2 from /192.168.1.8:55946
Categories recommended for you:User Id:1682277493
 3: art
replayed: Hello from Android, you are #2
#3 from /192.168.1.8:55947
Categories recommended for you:User Id:1682277493
 4: TV
replayed: Hello from Android, you are #3
```

Many thanks for Ting Xia for the help.