



NETFLIX RECOMMENDATION SYSTEM

CSYE7200 34322 BIG-DATA SYS ENGR USING SCALA SEC 01 – SPRING 2018

Team - 1

Sonali Chaudhari-001285029
Praneeth Reddy-001225762

OUR PROPOSAL

- **WITH THE HELP OF USER DATA AND RATINGS.**
- **GENERATE A RELEVANT SUGGESTION BASED ON PAST AVAILABLE DATASET.**
- **RECOMMENDING HIGHEST PREDICTED RATING TO A PARTICULAR USER.**
- **MEETING THE DEADLINE OF THE PROJECT.**

USE CASE/ ACTOR

ACTOR

- **USER WILL BE THE SOLE ACTOR OF THE SYSTEM**

USE CASE

- **USER WILL BELOW OPERATIONS:**
 - **PROVIDING RATING FOR MOVIES**
- **THE APPLICATION WILL PROVIDE LIST OF MOVIES WITH HIGHLY PREDICTED RATINGS**

DETAILS OF DATA

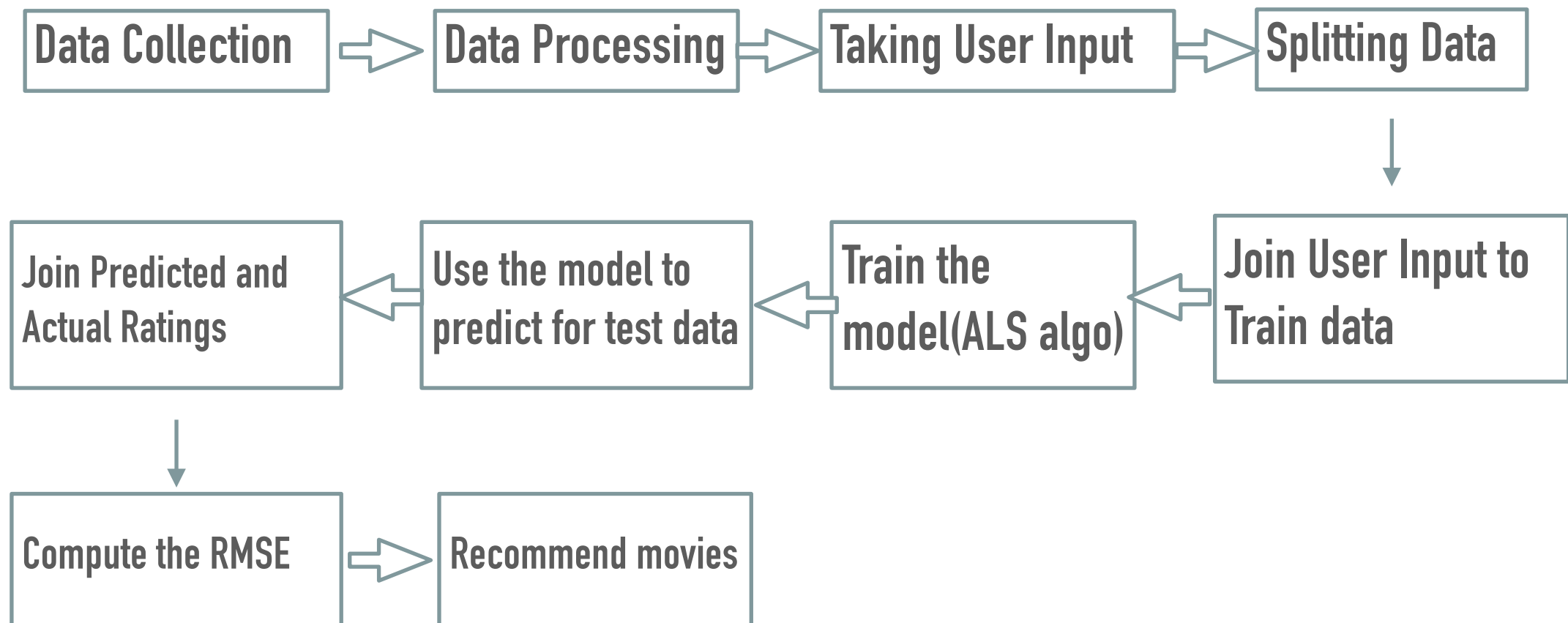
ACTUAL DATA:

- **RATINGS DATA : 10048050**
- **MOVIES : 17770**
- **USER: 480189**

DATA USED (HEAP SIZE ISSUE):

- **RATING DATA : 5010199**
- **MOVIES : 1000**
- **USERS: 404555**

WORKFLOW



PREDICTION ACCURACY

ROOT MEAN SQUARE ERROR (RMSE)

RMSE IS THE PARAMETER USED TO MEASURE THE DIFFERENCE BETWEEN THE PREDICTED VALUES AND THE ACTUAL VALUES.

$$\text{RMSE} = \sqrt{\sum \frac{(y_{pred} - y_{ref})^2}{N}}$$

RMSE FOR DIFFERENT RUNS

```
// Model training
val model = ALS.train(training,8,10,0.01)
```

```
// Implementing trained model on the test
```

```
DataProcessing > main(args: Array[String])
```

```
DataProcessing
```

```
18/04/18 18:23:16 INFO spark.SparkContext:
    .scala:111, took 3.795507228 s
```

```
RMSE: 1.141159286583431
```

```
18/04/18 18:23:16 INFO spark
```

```
MatrixFactorizationModel.s
```

```
18/04/18 18:23:16 INFO spark
```

```
shuffle 0 is 162 bytes
```

```
18/04/18 18:23:16 INFO spark
```

```
shuffle 22 is 176 bytes
```

```
18/04/18 18:23:16 INFO spark
```

```
// Model training
```

```
val model = ALS.train(training,8,5,0.01)
```

```
// Implementing trained model on the test
```

```
val prediction = model.predict(test.map(x => (x.user, x.item)))
```

```
// Joining predicted values and actual values
```

```
val predRatings = prediction.map(x => ((x.user, x.item), x.rating))
    .join[Double](test.map(x => ((x.user, x.item), x.rating)))
```

```
DataProcessing > main(args: Array[String])
```

```
DataProcessing
```

```
0.925 s
```

```
18/04/18 18:19:57 INFO spark.SparkContext:
```

```
Job finished: reduce at DataProcessing
```

```
.scala:111, took 3.414699773 s
```

```
RMSE: 1.1098127804418634
```

```
18/04/18 18:19:57 INFO spark.SparkContext:
```

```
Starting job: lookup at
```

```
// Model training
```

```
val model = ALS.train(training,8,5,0.099)
```

```
// Implementing trained model on the test
```

```
val prediction = model.predict(test.map(x => (x.user, x.item)))
```

```
// Joining predicted values and actual values
```

```
DataProcessing > main(args: Array[String])
```

```
DataProcessing
```

```
18/04/18 18:14:25 INFO executor.Executor:
    (TID 165). 945 bytes result sent to driver
```

```
18/04/18 18:14:25 INFO scheduler.TaskSetManager:
    stage 32.0 (TID 165) in 990 ms on localhost
```

```
18/04/18 18:14:25 INFO scheduler.TaskScheduler:
    whose tasks have all completed, from pool
```

```
18/04/18 18:14:25 INFO scheduler.DAGScheduler:
    DataProcessing.scala:111) finished in 0.0
```

```
18/04/18 18:14:25 INFO spark.SparkContext:
    DataProcessing.scala:111, took 3.6811863
```

```
RMSE: 0.9750691117807623
```

APPLICATION AND USER INTERFACE

KAFKA PRODUCER AND CONSUMER

The image displays a screenshot of an IDE (IntelliJ IDEA) with two Scala files open: `KafkaProd` and `KafkaCons`. The `KafkaProd` file implements a Kafka producer, and the `KafkaCons` file implements a Kafka consumer. Both files are part of the `edu.neu.netflix` package.

KafkaProd.scala

```
package edu.neu.netflix

import java.util.Properties

object KafkaProd extends App {

  override def main(args: Array[String]) = {

    val topic = "userratings"
    println(s"Connecting to $topic")
    val rnd = new Random()
    val props = new Properties()

    props.put("bootstrap.servers", "localhost:9092")
    props.put("client.id", "KafkaProducer")
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")

    val producer = new KafkaProducer[String, String](props)
    val t = System.currentTimeMillis()
    //userID, movieID, Rating, timestamp
    val data = new ProducerRecord[String, String](topic, "0.886,4.0,"+t)
    print("Message Published Successfully")
    producer.send(data)
    producer.close()
  }
}
```

KafkaCons.scala

```
import java.util.Properties

object KafkaCons {

  def main(args: Array[String]) = {

    val TOPIC = "userratings"

    val props = new Properties()
    props.put("bootstrap.servers", "localhost:9092")

    props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
    props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer")
    props.put("group.id", "something")

    val consumer = new KafkaConsumer[String, String](props)
    consumer.subscribe(util.Collections.singletonList(TOPIC))

    while (true) {
      val records = consumer.poll(100)
      for (record <- records.asScala) {
        println(record)
      }
    }
  }
}
```

The console output shows the following logs:

```
18/04/19 11:17:13 INFO utils.AppInfoParser: Kafka version : 1.1.0
18/04/19 11:17:13 INFO utils.AppInfoParser: Kafka commitId : fdcf75ea326b8e07
Message Published Successfully18/04/19 11:17:14 INFO clients.Metadata: Cluster ID: tob_huNaRX0lembNt5L_SA
18/04/19 11:17:14 INFO producer.KafkaProducer: [Producer clientId=KafkaProducer] Closing the Kafka producer with timeoutMillis = 9223372036854775807 ms.
Process finished with exit code 0
```

Movie Database

Movie	Rating
Ray	
Speed	
Reservoir Dogs	
Mean Girls	
Something's Gotta Give	
X-Men	
American Beaty	
Rush Hour	
Pay it forward	

<-----Please rate the movies (1(Low) to 5(High)) and Get Suggestions-->

NETFLIX

Show 10 entries

Search:

name	expectedRating
Loading...	

Showing 0 to 0 of 0 entries

Previous

Next

ACCEPTANCE CRITERIA

- **APPLICATION SHOULD BE ABLE TO HANDLE AT LEAST 2500 REQUESTS
SIMULTANEOUSLY AND THE MODEL SHOULD BE SCALABLE TO ADD NEW DATA
SOURCES AS AND WHEN REQUIRED.**
- **ACHIEVE >90% ACCURACY**

CHALLENGES FACED

- SPARK, SCALA, KAFKA AND PLAY FRAMEWORK COMPATIBILITY ISSUE
- RESTRUCTURING DATA TO USE INTO CORRECT FORMAT
- OVERFITTING OF THE MODEL
- TUNING THE MODEL
- HEAP SIZE ISSUE

USING PLAY FRAMEWORK

- IMPLEMENTED MVC
- INTEGRATING SPARK
- IMPLEMENTED MOCKITO FOR APPLICATION ,MOVIE
CONTROLLER SPEC TO TEST VARIOUS FEATURES

USING PLAY FRAMEWORK

- **TASK COMPLETED**
 - DATA PROCESSING
 - TAKING USER INPUT
 - PREDICTION GENERATING
- **TASK TO BE COMPLETED**
 - USER INTERFACE

GITHUB REPOSITORY LINK

<https://github.com/reddyse/Big-Data-Engineering-Using-Scala>

THANK YOU...