Master Thesis
Recommender Systems Comparison

Vasileios Symeonidis

27-05-2017

Contents

This page was intentionally left blank.

Part I
Master Thesis

1 Intro

This is the intro suction for this master thesis. Why we need recommendation systems? Retailers can propose the right product to the right target group. User get advertisements the may be interested in.[4]

2 Related Work

Below it is listed the related work to this thesis.

TODO: Update the above.

2.1 RecBench: Benchmarks for Evaluating Performance of Recommender System Architectures [1]

University of Minnesota, published in 2011 a paper stating a comparison between a recommender framework and a DBMS-based recommender. In that paper they used the Movie Lens dataset 100k, from the Netflix Challenge. The benchmark had five areas of comparison. Those areas were initialization, pure recommendation, filtered recommendation, blended recommendation, item recommendation and item update.

The initialization task was about the preparation needed for the system to go live. The next area was pure recommendation. By pure recommendation the author mean the home page recommendation, meaning the items that are going to be in the home page. Moving forward, we find the filtered recommendation. This recommendation is constrained by variables specific to the item, like movie genre etc. Another area of this evaluation contains the blended recommendation. Those recommendations are based on free text provided by the use in order to search. Item prediction is another are of the evaluation, in this prediction the user is navigated to the items page and the system is trying to predict the user's rating on the item. Last but not least, the paper examines the case of a new item being added to the system and how this is going to be incorporated to it.

As a result, of those experiments the paper conclude that "hand-build recommenders exhibit superior performance in model building and pure recommendation tasks, while DBMS-based recommenders are superior to more complex recommendations such as providing filtered recommendations and blending text-search with recommendation prediction issues.

2.2 Recommender Systems Evaluation: A 3D Benchmark [2]

In this paper the authors recognize the need for a common benchmark formula for recommender systems. This need lead them to propose one. They named it the 3D recommendation evaluation because they evaluate a system in three axis. These axis are business models, user requirements, and technical constrains. In business model axis they state that a recommender system must be evaluated on how well it serves the business case it is used for. In their paper they give the example of a video on demand service and evaluate it versus the pay per view business model and pay per subscription.

In the user requirements axis, the evaluate the system based on what need it covers for the users. Is it, for example, going to reduce search time or decision making time.

Last but not least is the technical constrains axis. In this axis the system is being evaluated based on data or hardware constrains, scalability and robustness.

2.3 RiVal: A New Benchmarking Toolkit for Recommender Systems [3]

RiVal, is an open source tool kit implemented in Java programming language. Rival is available via maven repositories. It is used in order to measure the evaluate recommender systems. Its evaluation is based on three points. Those point are data spiting, item recommendation, candidate item generation and performance measurement.

3 Collaborative filtering

Collaborative filtering is the process of filtering items based on others items with similar attributes.

needs more explanation

What is collaborative filtering. [4]

3.1 Content based

In content based recommender systems we try to make recommendations based on features we know. There are two types of content based recommender systems.

On the one hand we have the user based recommendation. This recommendation is done by trying to match users profiles, in order to find which item the user i might like. But in real world we don't have the needed information to make the recommendation to the user.

On the other hand we have the item-product based recommendation, in this case we are trying to find user that might like the given product. This is match easier due to the fact that you know more about a product than a user, and you can classify them easily.

In this case we have a matrix R that contains the rates given by users to items. This matrix most of the times will be low in density, this is because each user does not rate each product. The second matrix we come across is the M. This matrix contains all the movies with the their genres. Each characteristic is binary. For example, the movie with id i is both action and comedy and none of the other genres.

$$w = R^{-1}M^T \tag{1}$$

In order add an normalization factor to the above equation, we need to get it to the form below.

$$w = (\lambda I + R^T R)^{-1} R^T M \tag{2}$$

3.2 Latent Factors

Latent factors techniques are used to find attributes that are no clear in the dataset. This means this set of algorithms is trying to find the best metric, which may not be a clear one.

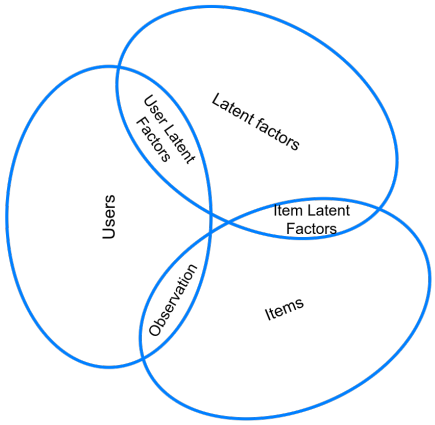

Figure 1: **LatentFactors**

In latent factors recommender systems we follow a similar approach but, in case of ALS(Alternating least squares), we are trying to find metrics that may lead us to the correct recommendation. Those metrics are not distinct, and may change in a number of iterations. Those metrics are inducted from the R matrix as we define it above. This makes this approach more tolerant to missing values, or wrong quality measures. Thus this metric as will be presented bellow is more efficient on prediction and time. [5]

$$\min_{X,Y} \sum_{r_{ui} observed} (r_{ui} - x_u^T y_i)^2 \tag{3}$$

$$\min_{X,Y} \sum_{r_{ui} observed} (r_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2) \tag{4}$$

ALS explanation. ALS algorithm is based on the latent factors theory.As mentioned before, this means that it is not going to use the attributes given by the dataset for the movies or the users. The algorithm is going to train it self based on the rating set only.

Algorithm 1 ALS for Matrix Completion

1: Initialize X,Y
2: **repeat**
3: **for** `u=1...n` **do**
4: $x_u = (\sum_{r_{ui}} y_i y_i^T + \lambda I_k)^{-1} \sum_{r_{ui}} r_{ui} y_i, \in r_{u*}$
5: **for** `i=1...m` **do**
6: $y_i = (\sum_{r_{ui}} x_u x_u^T + \lambda I_k)^{-1} \sum_{r_{ui}} r_{ui} x_u, \in r_{*i}$
7: **until** convergence

4 Our Experiment

4.1 Infrastructure

As the experiment's infrastructure we will describe the frameworks used during the implementation. The framework that has been used to implement the item-based algorithm was Apache Spark. The ALS algorithm was used from the Apache Spark's MLLib library. The framework is common to infrastructure because Apache Spark can orchestrate the work on multiple machine as well as in one. So the framework is as close as we can get to the infrastructure.

4.1.1 Apache Spark

The last decade, analyzing big data is at its peek. Lots of data are produced on daily basis. This means that the need of extracting information from them is raised. Lots of frameworks has been used in order to manage and analyze this amount of data. One of the analysis reasons is the need for accurate item recommendations to users. Those items could be movies (e.g. Netflix), music (e.g. Spotify) or products in general(e.g. Amazon). The one of the most popular frameworks that could enable this in a distributed way was Apache's Hadoop MapReduce.

Apache Hadoop has discrete jobs of processing data. The most common jobs are map and reduce but it has two more jobs, combine and partition. Hadoop has a master node and N worker nodes. The first is responsible to distribute the work, and the second for the work to be done. Each worker usually is called after the job is executing. Hence we have the mapper, the reducer, the partitioner and the combiner. In order to put this to a schema, you can see the figure 2 below.

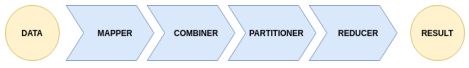

Figure 2: **Hadoop Jobs Order**

Hadoop map reduce, is a distributed map-reduce system, this means that it has a mechanism to distribute work on nodes and a common interface for handling data. In Hadoop's case this was able to happen due to Apache Hadoop Yarn and the Hadoop Distributed File System or as commonly used HDFS. When a job was scheduled, data were loaded by the HDFS to a worker, then the worker was done, he was putting the result back to the HDFS.

As mentioned in [6], "The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job."

So Hadoop has two basic processes, Map which is responsible for turning the data into key value pairs, and Reduce which takes those pairs and turns them into valuable data.

If we would like to see where in the DIKW (Data Information Knowledge Wisdom) stack. The Map process would with data and the reduce will end up with information. Of course this in not always the case, lots of algorithms require lots of cycles in order to complete.

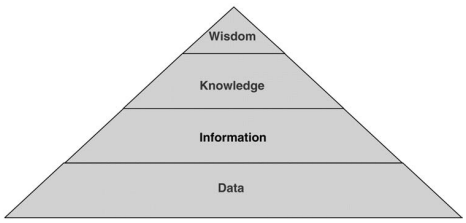

Figure 3: **Data Information Knowledge Wisdom Pyramid [7]**

But lets make a step back and take a look at Hadoop's architecture. As it is described in its official website [8], and shown in the figure 4 Hadoop uses Hadoop yarn in order to coordinate which process will run on which machine. Also it uses, its file system, the HDFS in order to have a common reference for the files over the network. Last but not least, Hadoop ecosystem is supported by the Hadoop Commons library.

In 2009, University of California, Berkley, proposed a new framework for cluster computing in their paper, Spark: Cluster Computing with Working Sets [9]. They wanted to tackle two major Hadoop issues.

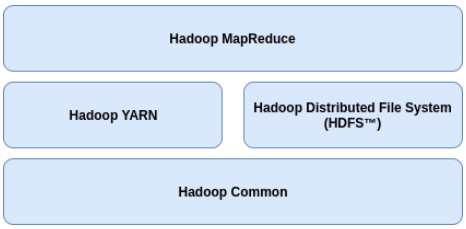

Figure 4: **Hadoup Software Stack**

The first was the iterative jobs. Each Hadoop job reads from the disk to load data. This means that having iterative jobs, on any given algorithm, you were going to get a large time penalty on reading and of course writing to the disk.

The second issue was the interactive analytics. Each Hadoop SQL interface was running as a seperete job, and as we mentioned previously we have a big impact on execution time.

In order to break the acyclic nature of Hadoop, they introduced the Spark's major abstraction, the RDDs. The name RDD stands for Resilient Distributed Datasets. Those datasets are a read-only collection of objects distributed across machines. If a machine fail, the lost part of the RDD can be recalculated. This notion is called lineage.

Spark is implemented in Scala. Scala is a high-level statically typed programming language. At the time that paper was published it was believed that Spark was the only system available in a general purpose programming language to make clusters process large amount of data. As it was mentioned in [9] "We believe that Spark is the first system to allow an efficient, general purpose programming language to be used interactively to process large datasets on clusters"

Back to RDDs, an RDD can be created with four different operations as it is described in [9]. The first operation is loading data from any shared file system. That file system could be HDFS or even an Amazon S3. The second way to create a RDD is by parallelizing any Scala collection. Spark will slice the collection into pieces and distribute it among the nodes. The third way is via transforming an RDD to another one. Because RDDs are immutable, any transformation operation on a RDD, filter, map, flatmap, will generate a new RDD. The last but not least method is by changing an RDDs persistence using save or cache operations.

Spark also give us the power to do a lot of different distributed operations. Some of them was mentioned before, but we also have operations that would return data to the driver program like collect or reduce.

Another important feature of Sparks spine are the shared variables. Spark at its first appearance introduced two of them. The first shared variable is the broadcasted variables. Those variable, RDDs or not, are variables that are commonly used in an algorithm, like a look-up table. By broadcasting a variable, each node gets a copy of the variable in order to access it quickly. The second shared variable that was introduced in that paper was the Accumulators. Those variable live on the spark context, but they can only be increased by any worker and be read from the driver program only. That paper concludes that Spark can outperform Hadoop in some machine learning algorithms and more specific on logistic regression.

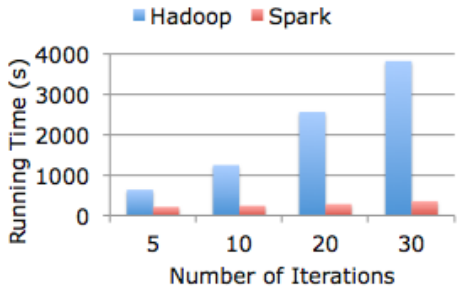

Figure 5: **Logistic regression, Hadoop vs Spark [10]**

Coming back to today, Spark's current architecture is depicted below in fig. 6. Spark nowadays has an sql interface in order to search in RDDs with in a query language. Also Spark support a streaming api to make available real-time analytics. Most of the core machine learning algorithms like ALS, which I used in order to complete this thesis, are inside the Spark's MLlib component. Finally spark has the component GraphX that is used for handling graphs and graph computation.

Apache Spark was by design meant to work within Hadoop ecosystem, and most importantly with the HDFS. Apache Spark does not has a file system by itself. You can load data from almost any database, cloud-based or not, even from a local file system. But most will agree that Hadoop and Spark work together just perfect.

To conclude, Spark has dominated the big data field the last years, Amazon and other cloud providers gives you the option to deploy an Apache Spark cluster on their infrastructure. Also large companies like Google and Intel are actively contributing to projects like Apache Spark On Kubernetes which can be found at the following Github repository (https://github.com/apache-spark-on-k8s/spark)

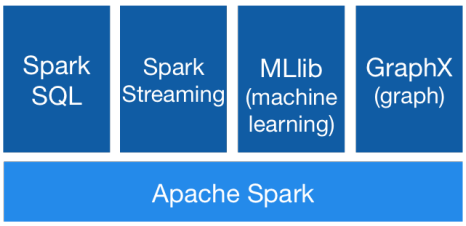

Figure 6: **Apache spark stack [11]**

4.2 Dataset

What is the dataset about. This dataset contains users, movies and the rating user made about the movies. This dataset is splited to multiple subsets of 80000 training sets and respective 20000 reviews. This dataset has been used by the related work on ..... [12]

4.3 Implementation and assumptions

During the implementation I had to make some assumptions and choices. The first of choices was the framework and the programming language that the implementation would take place. The framework that has been chosen, as you may had already figured out, is apache spark due to its trend and the high scalability it offers. The language of choice was scala, due to its functional nature.

4.4 Metrics

After the implementation I had to make the choice of the metrics I was going to use.

4.4.1 Mean Absolute Error

As metrics are commonly used the MSE, RMSE and MAE. Due to the fact that the author prefers the last one, MAE was used in this experiment.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} \sqrt{(y_i - x_i)^2}}{n} \tag{5}$$

4.4.2 Execution Time

Time is measured in milliseconds. Execution time is always a measure when we are comparing algorithms. Even more if those algorithms execution time is heavily dependent to their complexity and not their resources.

5 Results

Gathering the results of the above experiment, we can see roughly a very big difference between the two algorithms on both metrics, MAE and Execution Time.

Table 1: **Content Based Algorithm Results**

Training Dataset Testing Dataset	**Mean Absolute Error**	**Root Mean Square Error**	**Execution Time (ms)**
u1.base — u1.test	1.6467431428213226	2.1040777757932614	30514
u2.base — u2.test	1.6055222166704628	2.059448482243628	27714
u3.base — u3.test	1.608925907479106	2.0914574229083644	27164
u4.base — u4.test	1.6259192043203685	2.086253134349873	26687
u5.base — u5.test	1.6284658627202895	2.099015094620632	27124
ua.base — ua.test	1.6425364580036836	2.1224499666494605	26640
ub.base — ub.test	1.6357196576385744	2.099455301804093	26861

NOTE: RMSE gives greater error with peaks, meaning that MAE/RMSE

TODO: Break tables to more

TODO: Add rmse and mae difference diagrams

TODO: Add rmse/mae diagrams

Table 2: **Latent Factors Algorithm Results**

Training Dataset Testing Dataset	**Mean Absolute Error**	**Root Mean Square Error**	**Execution time (ms)**
u1.base — u1.test	1.1818684937209607	1.379344822269489	10195
u2.base — u2.test	1.1800652808093945	1.3784149024890802	6517
u3.base — u3.test	1.1783366748334452	1.3754645935161673	5377
u4.base — u4.test	1.1730543877181654	1.3706400992617636	5433
u5.base — u5.test	1.1686585291940668	1.3668078008537445	5217
ua.base — ua.test	1.2008035300836668	1.3968580905671946	5214
ub.base — ub.test	1.2134460078406009	1.411978048105032	5083

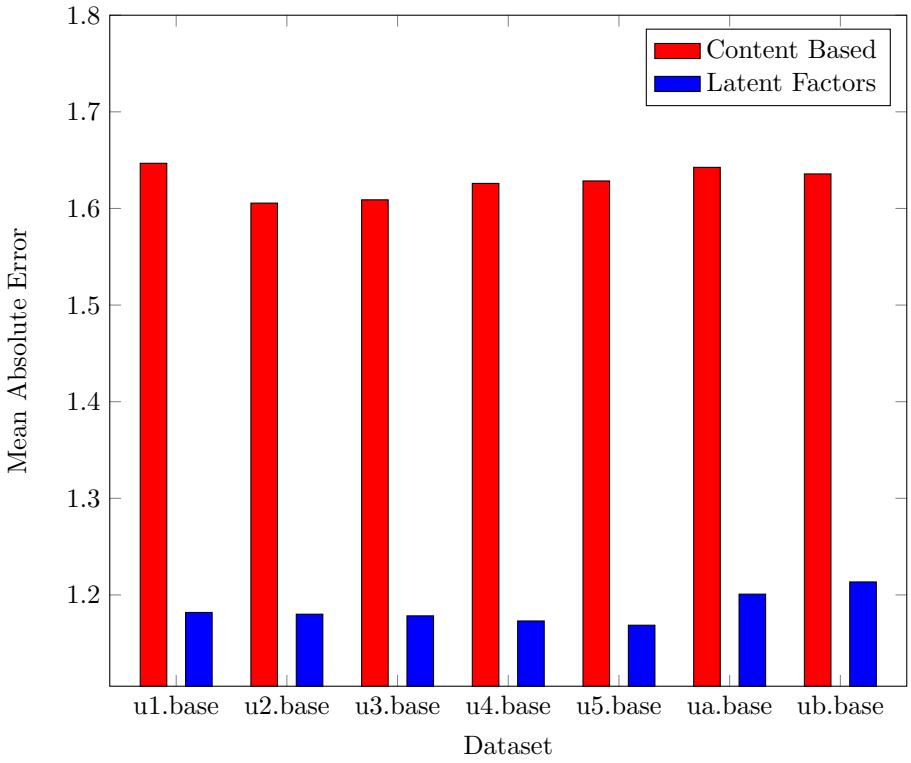

Figure 7: **Latent Factors vs Content Based on Mean Absolute Value**

6 Conclusion

As a conclusion we can see that als is better on both metrics from the content based.

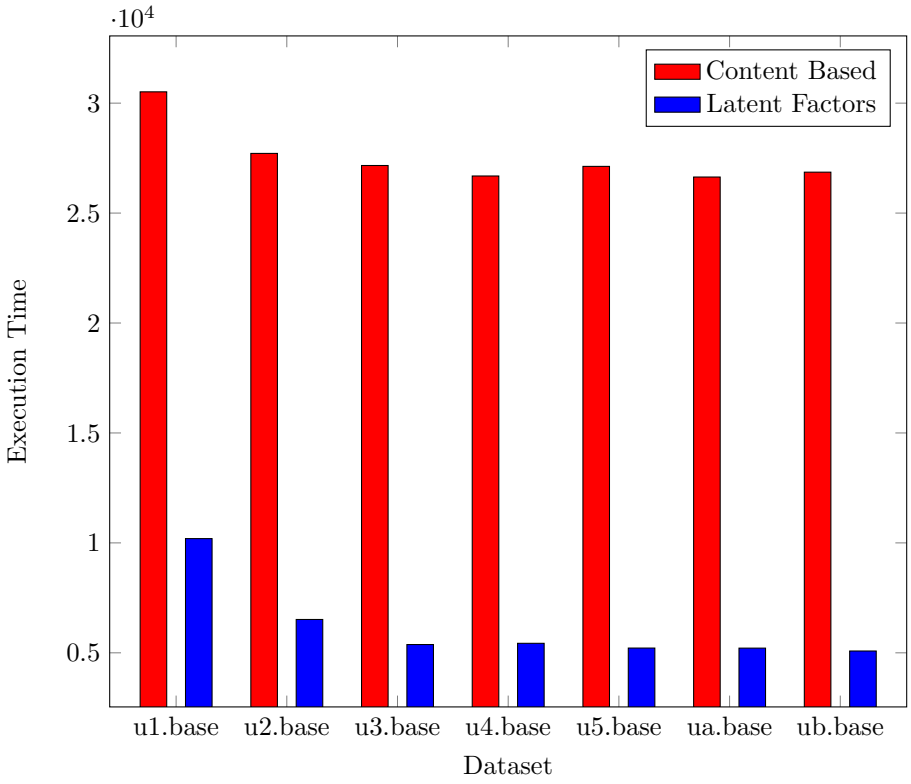

Figure 8: **Latent Factors vs Content Based on Execution Time**

References

[1] J. J. Levandoski, M. D. Ekstrand, M. J. Ludwig, A. Eldawy, M. F. Mokbel, and J. T. Riedl, "Recbench: benchmarks for evaluating performance of recommender system architectures," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, 2011.

[2] A. Said, D. Tikk, K. Stumpf, Y. Shi, M. Larson, and P. Cremonesi, "Recommender systems evaluation: A 3d benchmark.," in *RUE@ RecSys*, pp. 21–23, 2012.

[3] A. Said and A. Bellogín, "Rival: a toolkit to foster reproducibility in recommender system evaluation," in *Proceedings of the 8th ACM Conference on Recommender systems*, pp. 371–372, ACM, 2014.

[4] P. Melville and V. Sindhwani, "Recommender systems," *Encyclopedia of Machine Learning and Data Mining*, pp. 1056–1066, 2017.

[5] B. H. Haoming Li, M. Lublin, and Y. Perez, "Cme 323: Distributed algorithms and optimization, spring 2015." http://stanford.edu/~rezab/dao, 2015. Lecture 14, 5/13/2015.

[6] "IBM what is map-reduce." https://www.ibm.com/analytics/us/en/technology/hadoop/mapreduce/. Accessed: 2017-06-24.

[7] J. Rowley, "The wisdom hierarchy: representations of the dikw hierarchy jennifer rowley," *Journal of Information Science*, pp. 163–180, 2007.

[8] "Apache Hadoop." http://hadoop.apache.org/. Accessed: 2017-06-24.

[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2010.

[10] "Hadoop Vs Spark." https://amplab.cs.berkeley.edu/projects/spark-lightning-fast-cluster-computing/. Accessed: 2017-06-24.

[11] "Apache Spark lightning-fast cluster computing." https://spark.apache.org/. Accessed: 2017-05-21.

[12] "MovieLens grouplens." https://grouplens.org/datasets/movielens/. Accessed: 2017-05-22.

Part II
Appendices

In this part you will find the code used for this master thesis.

A Code used

Below is the code used in order to produce the results. It has been written for Apache Spark [11] using scala. The library breeze has been used for matrix processing.

A.1 Item Based Collaborative Filtering

```
import java.util

import breeze.linalg.{Axis, DenseMatrix, pinv}
import breeze.optimize.linear.PowerMethod.BDM
import org.apache.spark.SparkContext
import org.apache.spark.mllib.linalg.Matrix
import org.apache.spark.mllib.linalg.distributed.{
    ↪ CoordinateMatrix, MatrixEntry}
import org.apache.spark.mllib.recommendation.Rating
import org.apache.spark.rdd.RDD




object ContentBased {

  val sparkContext: SparkContext = Infrastructure.sparkContext

  def main(args: Array[String]) {

    val bestNormalizationFactor = Infrastructure.
        ↪ normalizationFactorsList.map { v =>
      val sum = Infrastructure.dataSetList.map(dataSet =>
          ↪ getMetricsForDataset(v, dataSet._1, dataSet._2)).map(
          ↪ u => u._5).sum
      val mean = sum/Infrastructure.dataSetList.size
      (v, mean)
    }.maxBy(v=> v._2)

    Infrastructure.dataSetList
```

```
    .map(dataSet => getMetricsForDataset(bestNormalizationFactor
        ↪ ._2, dataSet._1, dataSet._2))
    .foreach(metric => println(metric))
  println("training␣set", "testing␣set", "MSE", "RMSE", "MAE", "
      ↪ Execution␣Time")

}


private def getMetricsForDataset(normalizationFactor: Double,
    ↪ trainingSet: String, testingSet: String) = {
  val startingTime = System.currentTimeMillis()

  val itemsMatrixEntries: RDD[MatrixEntry] =
      ↪ generateItemMatrixEntries

  val itemMatrix: Matrix = new CoordinateMatrix(
      ↪ itemsMatrixEntries).toBlockMatrix().toLocalMatrix()
  val itemMatrixBreeze = toBreeze(itemMatrix).copy

  val ratings = sparkContext.textFile(trainingSet)
    .map(_.split("\t") match {
      case Array(user, item, rate, timestamp) => Rating(user.
          ↪ toInt, item.toInt, rate.toDouble)
    }).cache()

  val usersRatings = ratings.groupBy(r => r.user)
    .map(v => (v._1, generateUserMatrix(v._2)))

  val refinedMatrices = usersRatings
    .map(v => (v._1, getRefinedMatrices(v._2, itemMatrixBreeze))
        ↪ )

  val userWeights = refinedMatrices.map(v => Pair(v._1,
      ↪ generateWeight(v, normalizationFactor)))
  val testRatings = sparkContext.textFile(testingSet)
    .map(_.split("\t") match {
      case Array(user, item, rate, timestamp) => Rating(user.
          ↪ toInt, item.toInt, rate.toDouble)
    }).cache()

  // remove rating from dataset
  val usersProducts = testRatings.map {
    case Rating(user, product, rate) => (user, product)
  }
```

```
  // predict
  val b = sparkContext.broadcast(userWeights.collect())

  val predictions = usersProducts.map(v =>
    ((v._1, v._2),
     predict(
       b.value.apply(v._1 - 1)._2,
       getRow(itemMatrixBreeze, v._2 - 1))
    ))

  val ratesAndPredictions = testRatings.map {
    case Rating(user, product, rate) => ((user, product), rate)
  }.join(predictions)

  ///// Metrics ////

  // calculate MSE (Mean Square Error)
  val MSE = Metrics.getMSE(ratesAndPredictions)

  // calculate RMSE (Root Mean Square Error)
  val RMSE = Math.sqrt(MSE)

  // calculate MAE (Mean Absolute Error)
  val MAE = Metrics.getMAE(ratesAndPredictions)

  val endingTime = System.currentTimeMillis()

  val executionTime = endingTime - startingTime

  (trainingSet, testingSet, MSE, RMSE, MAE, executionTime,
      ↪ normalizationFactor)
}

private def predict(weight: DenseMatrix[Double], item:
    ↪ DenseMatrix[Double]): Double = {
  val result = item.t * weight
  if (result.data.length > 1) {
    println("something␣went␣wrong␣on␣prediction")
    0
  }
  else result.data.apply(0)

}

private def generateWeight(v: (Int, (DenseMatrix[Double],
    ↪ DenseMatrix[Double])), normalizationFactor: Double):
```

```
    ↪ DenseMatrix[Double] = {
  calculateWeightsWithNormalizationFactor(v._2._2, v._2._1,
      ↪ normalizationFactor)
}

private def calculateWeightsWithNormalizationFactor(
    ↪ ratingMatrix :DenseMatrix[Double], itemMatrix:
    ↪ DenseMatrix[Double], normalizationFactor: Double):
    ↪ DenseMatrix[Double] = {
  val lambdaIdentity = DenseMatrix.eye[Double](ratingMatrix.cols
      ↪ ) :* normalizationFactor
  pinv(
    lambdaIdentity
      +
      (ratingMatrix.t * ratingMatrix)
  ) * (ratingMatrix.t * itemMatrix)
}


private def calculateWeightsWithoutNormalizationFactor(
    ↪ ratingMatrix :DenseMatrix[Double], itemsMatrix:
    ↪ DenseMatrix[Double]): DenseMatrix[Double] = {
  pinv(ratingMatrix) * itemsMatrix
}

def getRefinedMatrices(userMatrix: DenseMatrix[Double],
    ↪ itemMatrix:DenseMatrix[Double]): (DenseMatrix[Double],
    ↪ DenseMatrix[Double]) = {
  var sequence = Seq[Int]()
  userMatrix.foreachKey { v =>
    if (userMatrix(v._1,v._2) == 0) {
      sequence = sequence :+ v._1
    }
  }
  val localItemMatrix = itemMatrix.delete(sequence, Axis._0)
  val localUserMatrix = userMatrix.delete(sequence, Axis._0)
  (localUserMatrix, localItemMatrix)
}


def getRow(matrix: DenseMatrix[Double], row: Int): DenseMatrix[
    ↪ Double] = {
  val numberOfColumns = matrix.cols
  val array = new Array[Double](numberOfColumns)
  for (i <- 0 until numberOfColumns){
    array(i)=matrix(row,i)
```

```
  }
  new DenseMatrix(numberOfColumns ,1, array)
}

def generateUserMatrix(userRatings: Iterable[Rating]):
    ↪ DenseMatrix[Double] = {

  val numberOfItems = Infrastructure.items.count().toInt
  val array = new Array[Double](numberOfItems)
  util.Arrays.fill(array, 0)
  userRatings.foreach(r => array(r.product - 1) = r.rating)
  new DenseMatrix(numberOfItems ,1, array)
}

private def toBreeze(matrix: Matrix): DenseMatrix[Double] = {
  val breezeMatrix = new BDM(matrix.numRows, matrix.numCols,
      ↪ matrix.toArray)
  if (!matrix.isTransposed) {
    breezeMatrix
  } else {
    breezeMatrix.t
  }
}

private def generateItemMatrixEntries: RDD[MatrixEntry] = {
  Infrastructure.items.flatMap(a => Array(
  MatrixEntry(a(0).toLong - 1, 0, a(4).toInt),
  MatrixEntry(a(0).toLong - 1, 1, a(5).toInt),
  MatrixEntry(a(0).toLong - 1, 2, a(6).toInt),
  MatrixEntry(a(0).toLong - 1, 3, a(7).toInt),
  MatrixEntry(a(0).toLong - 1, 4, a(8).toInt),
  MatrixEntry(a(0).toLong - 1, 5, a(9).toInt),
  MatrixEntry(a(0).toLong - 1, 6, a(10).toInt),
  MatrixEntry(a(0).toLong - 1, 7, a(11).toInt),
  MatrixEntry(a(0).toLong - 1, 8, a(12).toInt),
  MatrixEntry(a(0).toLong - 1, 9, a(13).toInt),
  MatrixEntry(a(0).toLong - 1, 10, a(14).toInt),
  MatrixEntry(a(0).toLong - 1, 11, a(15).toInt),
  MatrixEntry(a(0).toLong - 1, 12, a(16).toInt),
  MatrixEntry(a(0).toLong - 1, 13, a(17).toInt),
  MatrixEntry(a(0).toLong - 1, 14, a(18).toInt),
  MatrixEntry(a(0).toLong - 1, 15, a(19).toInt),
  MatrixEntry(a(0).toLong - 1, 16, a(20).toInt),
  MatrixEntry(a(0).toLong - 1, 17, a(21).toInt),
  MatrixEntry(a(0).toLong - 1, 18, a(22).toInt))
  )
```

```
  }
}
```

A.2 Latent Factors

```
import org.apache.spark.SparkContext
import org.apache.spark.mllib.recommendation.{ALS, Rating}
import org.apache.spark.rdd.RDD

object LatentFactors {

  val sparkContext: SparkContext = Infrastructure.sparkContext

  def main(args: Array[String]) {

    val bestNormalizationFactor = Infrastructure.
        ↪ normalizationFactorsList.map { v =>
      val sum = Infrastructure.dataSetList.map(dataSet =>
          ↪ getMetricsForDataset(dataSet._1, dataSet._2, v)).map(
          ↪ u => u._5).sum
      val mean = sum/Infrastructure.dataSetList.size
      (v, mean)
    }.maxBy(v=> v._2)

    Infrastructure.dataSetList
      .map(dataSet => getMetricsForDataset(dataSet._1, dataSet._2,
          ↪  bestNormalizationFactor._2))
      .foreach(metric => println(metric))
    println("training␣set", "testing␣set", "MSE", "RMSE", "MAE", "
        ↪ Execution␣Time")

  }

  private def getMetricsForDataset(trainingSet:String, testingSet
      ↪ :String, normalizationFactor: Double) = {

    val startingTime = System.currentTimeMillis()

    val ratings = sparkContext.textFile(trainingSet).map(_.split("
        ↪ \t") match { case Array(user, item, rate, timestamp) =>
      Rating(user.toInt, item.toInt, rate.toDouble)
    }).cache()

    //// Build the recommendation model using ALS
    val rank = 15 // 10 - 20
    val numIterations = 75 // 50 - 100
```

```
    val model = ALS.train(ratings, rank, numIterations,
        ↪ normalizationFactor)

    //import test dataset
    val testRatings = sparkContext.textFile(testingSet).map(_.
        ↪ split("\t") match { case Array(user, item, rate,
        ↪ timestamp) =>
      Rating(user.toInt, item.toInt, rate.toDouble)
    }).cache()

    // remove rating from dataset
    val usersProducts = testRatings.map {
      case Rating(user, product, rate) => (user, product)
    }

    // predict the rating
    val predictions = model.predict(usersProducts).map {
      case Rating(user, product, rate) => ((user, product), rate)
    }

    // join rdd to get the rating and the prediction value for
        ↪ each combination
    val ratesAndPredictions: RDD[((Int, Int), (Double, Double))] =
        ↪  testRatings.map {
      case Rating(user, product, rate) => ((user, product), rate)
    }.join(predictions)

    ///// Metrics ////

    // calculate MSE (Mean Square Error)
    val MSE = Metrics.getMSE(ratesAndPredictions)

    // calculate RMSE (Root Mean Square Error)
    val RMSE = Math.sqrt(MSE)

    // calculate MAE (Mean Absolute Error)
    val MAE = Metrics.getMAE(ratesAndPredictions)

    val endingTime = System.currentTimeMillis()

    val executionTime = endingTime - startingTime

    (trainingSet, testingSet, MSE, RMSE, MAE, executionTime)
  }
}
```

A.3 Infrastructure code

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.rdd.RDD

import scala.collection.immutable

object Infrastructure {
  val sparkConfiguration: SparkConf = new SparkConf()
    .setMaster("local[*]")
    .setAppName("RecommenderSystemsComparison")
  val sparkContext: SparkContext = {
    val sc = new SparkContext(sparkConfiguration)
    sc.setCheckpointDir("checkpoint/") // set checkpoint dir to
        ↪ avoid stack overflow
    sc
  }


  //import data to rdds
  val users: RDD[Array[String]] = sparkContext.textFile("ml-100k/
      ↪ u.user").map(u => u.trim.split("\\|")).cache()
  val genres: RDD[Array[String]] = sparkContext.textFile("ml-100k
      ↪ /u.genre").map(u => u.trim.split("\\|")).cache()
  val items: RDD[Array[String]] = sparkContext.textFile("ml-100k/
      ↪ u.item").map(u => u.trim.replace("||", "|").split("\\|"))
      ↪ .cache()
  val occupations: RDD[String] = sparkContext.textFile("ml-100k/u
      ↪ .occupation").cache()
  val dataSetList = List(
    ("ml-100k/u1.base", "ml-100k/u1.test"),
    ("ml-100k/u2.base", "ml-100k/u2.test"),
    ("ml-100k/u3.base", "ml-100k/u3.test"),
    ("ml-100k/u4.base", "ml-100k/u4.test"),
    ("ml-100k/u5.base", "ml-100k/u5.test"),
    ("ml-100k/ua.base", "ml-100k/ua.test"),
    ("ml-100k/ub.base", "ml-100k/ub.test")
  )

  val normalizationFactorsList: immutable.Seq[Double] = List
      ↪ (0.01,0.03,0.06,0.09,0.12,0.15,0.18,1)
}

import org.apache.spark.rdd.RDD

object Metrics {
```

```
  def getMSE (ratesAndPredictions: RDD[((Int, Int), (Double,
      ↪ Double))] ): Double = {
    ratesAndPredictions.map { case ((user, product), (r1, r2)) =>
      val err = r1 - r2
      err * err
    }.mean()
  }

  def getMAE (ratesAndPredictions: RDD[((Int, Int), (Double,
      ↪ Double))] ): Double = {
    ratesAndPredictions.map { case ((user, product), (r1, r2)) =>
      val err = r1 - r2
      Math.abs(err)
    }.mean()
  }
}
```

List of Tables

List of Figures