Developing fast and scalable recommendation system on large datasets

Gopi Krishnan Nambiar   Parminder Bhatia
Revant Kumar

Advanced Internet Computing CS 6675
Group: 17
April 24, 2015

Abstract

A common task of recommender systems is to improve customer experience through personalized recommendations based on prior implicit feedback. These systems passively track different sorts of user behavior, such as purchase history, watching habits and browsing activity, in order to model user preferences. Unlike the much more extensively researched explicit feedback, we do not have any direct input from the users regarding their preferences.Our project aims to capture two important aspects in the domain of recommendation systems. First, we try to incorporate implicit and explicit features of Recommendation systems and how they help in achieving better representation for User Recommendations System . Second, we plan to address the problem of scalability of recommendation system by using distributed frameworks and analyze its the performance compared to earlier implementation.

Contents

1 Introduction

Customers have a ton of options to choose from in today's market. For instance, in order to buy an appliance or clothes for any occasion, there are multiple websites competing with each other in order to give the customer a good deal. In addition to this, once the customer makes a purchase, the website or retailer retains this information and uses it to predict and inform the customer about new releases or other products that the customer would be interested in buying.

This is where the role of recommender systems comes in. These systems make use of existing data such as profiles, page views, favorites, genres and other related information to come up. They may also be used to predict the rating or opinion a user may have about a particular product, and make use of this information to surface relevant products or personalized recommendations for their users.

Today, most e-commerce websites, social networks and music streaming services make use of complex recommendation systems. Some prominent examples would be Amazon, eBay, NetFlix, Spotify, last.fm, Facebook, MySpace and LinkedIn. In fact, NetFlix threw an open challenge called the NetFlix Prize to the academic community which was basically to build a recommender system on the company's dataset and return recommendations that were more than 10% accurate compared to the company's existing algorithm. The contest prize of $1,000,000 was awarded to BellKor's Pragmatic Chaos team using tie breaking rules. In this project, we are implementing a few of the algorithms that had been suggested by one of the members of the winning team, Yehuda Koren.

Recent research has proven that combining both collaborative filtering and content based filtering techniques lead to imtproved results in the case of recommendation algorithms

2 Literature Survey

The authors in [1] have introduced a new idea to formulate topic based social influence analysis using a Graph - G(V,E) where V represents users or entities and E represents undirected edges. The authors attempt to design a model which takes into consideration both topic distribution and network information to get a sense of the social influence, which is very different from the existing works on social network analysis conducted till that point of time. These two items are used as inputs for the program. In order to make the system conform to Map-Reduce framework, the authors implement a modified message passing algorithm which is compatible for the same. First, a Topic Factor Graph (TFG) is constructed which is a probabilistic model that includes all the information in a combined model. This is followed by the construction of the TAP model and parallelization procedure for the Map Reduce implementation. The TFG model consists of observed variables and latent variables as well which are the nodes in the network. They have run the algorithm in a distributed manner which contributes to a speedup of about 15X for large data sets. As expected with parallel

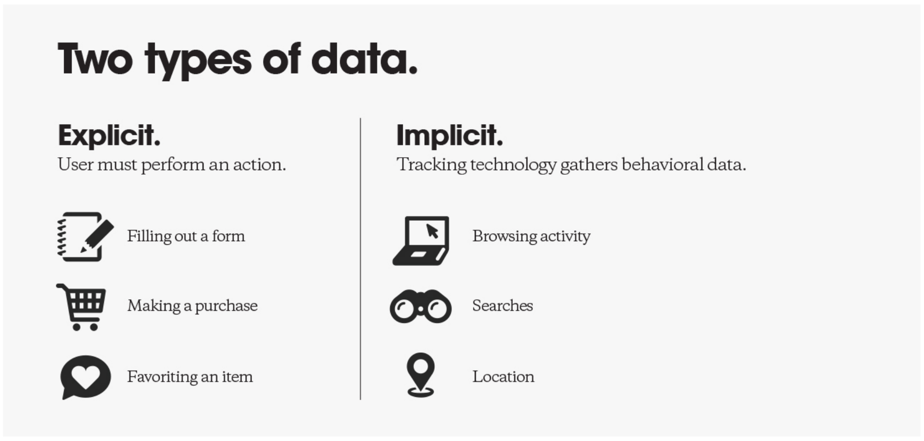


Figure 1: Implicit Explicit Information

processing, this method is not efficient for smaller data sets. [7] The cold-start recommendation task in an online retail setting for users who have not yet purchased (or interacted in a meaningful way with) any available items but who have granted access to limited side information, such as basic demographic data (gender, age, location) or social network information (Facebook friends or page likes). In this paper[7] , they formalize neighborhood-based methods for cold-start collaborative filtering in a generalized matrix algebra framework that does not require purchase data for target users when their side information is available. In real-data experiments with 30,000 users who purchased 80,000+ books and had 9,000,000+ Facebook friends and 6,000,000+ page likes, they show that using Facebook page likes for cold-start recommendation yields up to a 3-fold improvement in mean average precision (MAP) and up to 6-fold improvements in Precision and Recall compared to most-popular-item, demographic, and Facebook friend cold start recommenders. These results demonstrate the substantial predictive power of social network content, and its significant utility in a challenging problem ? recommendation for cold-start users.

3 Objectives

- Implementation of CF Recommendation Systems.
- Incorporate Implicit and Explicit Information.
- Implementation and Evaluation for Large Scale Datasets using Spark.
- Evaluation of Hadoop vs Spark type of architectures.

4 Frameworks

4.1 Spark

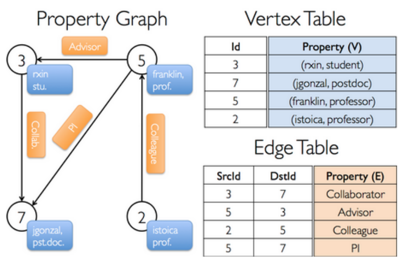
Apache Spark is a cluster computing platform designed to be fast and generalpurpose. On the speed side, Spark extends the popular MapReduce model

Figure 2: Large Scale Recommendation Requirement

to efficiently support more types of computations, including interactive queries and stream processing. Speed is important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours.
One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also more efficient than MapReduce for complex applications running on disk. On the generality side, Spark is designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to combine different processing types, which is often necessary in production data analysis pipelines.
In addition, it reduces the management burden of maintaining separate tools. Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries. It also integrates closely with other Big Data tools. In particular, Spark can run in Hadoop clusters and access any Hadoop data source, including Cassandra. Spark is an open source cluster computing environment similar to Hadoop, but it has some useful differences that make it superior in certain workloads. Spark enables in-memory distributed datasets that optimize iterative workloads in addition to interactive queries.

Spark is implemented in the Scala language and uses Scala as its application framework. Unlike Hadoop, Spark and Scala create a tight integration, where Scala can easily manipulate distributed datasets as locally collective objects.

Although Spark was created to support iterative jobs on distributed datasets, it's actually complementary to Hadoop and can run side by side over the Hadoop file system. Spark was developed at the University of California, Berkeley, Algorithms, Machines, and People Lab to build large-scale and low-latency data analytics applications.



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Figure 3: Spark GraphX Graphs

4.1.1 RDD

A Resilient Distributed Dataset (RDD), the basic abstraction in Spark. Represents an immutable, partitioned collection of elements that can be operated on in parallel. This class contains the basic operations available on all RDDs, such as map, filter, and persist.

4.1.2 GraphX

GraphX is a component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and aggregateMessages) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

4.1.3 Scala Build Tool

The Scala Built Tool (SBT) is a very popular build tool for Scala that supports both Scala and Java code. Building Spark projects with sbt is one of the easiest options because Spark itself is built with sbt. It makes it easy to bring in dependencies, which are very helpful for Spark. SBT also helps in packaging everything into a single deployable/JAR file. Since the JAR file would be shipped to the other machines, the developer must ensure that all the dependencies are included in them. To achieve this, either a bunch of JAR files could be added or an sbt plugin called sbt-assembly could be used in order to group everything into a single JAR file. If the dependencies are not transient, the developer can opt to skip using the plugin and instead could choose to run sbt/sbt assembly task in the Spark project and then add the resulting JAR file core/target/spark-core-assembly-0.X.0.jar to the classpath. The sbt-assembly package is a great tool to avoid having to manually manage a large number of JAR files. Suppose we want to construct a property graph consisting of the various collaborators on the GraphX project. The vertex property might contain the username and occupation. We could annotate edges with a string describing the relationships between collaborators:

5 Methodology

The prominent methods of building recommender systems are listed in the next few sections, and an additional section is devoted to the explanation of the integrated model which we have implemented in this project.

5.1 Collaborative Filtering

Collaborative filtering is a widely used technique for building recommender systems. These methods are primarily based on collecting and analyzing a large amount of information on user behavior, preference and predicting what users would like based on their similarity to other users.

A key advantage of the collaborative filtering approach is that it does not rely purely on machine analyzable content and hence capable of accurately recommending complex items such as movies without requiring an understanding of the item itself. Many algorithms have been used in measuring user similarity or item similarity in recommender systems like K-Nearest Neighbor techniques, Jaccard Similarity and Cosine Similarity to name a few. Collaborative Filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past.

When building a model from a user's profile, a distinction is should be made between explicit and implicit forms of data collection. These are the two main forms of data collection in case of collaborative filtering and can be explained as follows:

- Explicit: Where the system asks the user for input about preferences, favorites and other similar items
- Implicit: Where the system infers user preferences based on click through data, page views and browsing behavior.

5.1.1 Memory based

Memory based algorithms approach the collaborative filtering problem by utilizing the entire database. This method tries to find users which are similar to the existing user and predict ratings accordingly using the information of similar users. In this case, we need to measure the similarity between any 2 users. Hence, we use correlation measures such as Pearson correlation or cosine similarity measure. Finally, in order to predict a rating for a user, we need to find all the weights between this user and all other users. Next, we need to look at all non-zero weights and have every other user vote on the possible prediction for this user. Then all these votes are considered along with the weights and we end up with a final prediction for the rating.

5.1.2 Model based

Model based collaborative filtering techniques involve building a model based on the dataset of ratings. In other words, we extract some information from the dataset, and use that as a model to make recommendations without having to use the complete dataset every time. This approach potentially offers the benefits of both speed and scalability. Some of the popular model based techniques include Singular Value Decomposition, probabilistic latent semantic analysis (pLSI), Multiple Multiplicative Factor, Latent Dirichlet allocation and Markov Decision process based models.

SVD is basically a matrix factorization technique, where a matrix X with dimensions m by n is split into 3 different matrices as shown in the equation below:

$$A_{mxn} = U_{mxr} S_{rxr} V^T_{rxn}$$

The matrix S is a diagonal matrix containing the singular values of the matrix A. There are exactly r singular values, where r is the rank of A.

This approach has a more holistic goal of uncovering the latent factors that impact the ratings and hence provides a convincing explanation to the user as to why he was recommended a particular item. This method also handles data sparsity more elegantly compared to other methods.

5.1.3 Hybrid Methods

This method involves combining both memory and model based collaborative filtering techniques to come up with a hybrid model incorporating the best of both worlds. It has been reported in the latest research that the prediction accuracy of hybrid models is better in comparison to the native CF approaches. The tradeoff in this case is that the model is more expensive to construct and is much more complex than the native methods.

5.2 Content based methods

Content based systems focus on properties of items. Similarity of items is determined by measuring the similarity in their properties, whereas Collaborative Filtering systems focus on the relationship between users and items. In case of content based systems, we construct a profile for each item and user. Then predictions are made using these item and user vectors by estimating the degree to which a user would prefer a certain item by comparing the distance (cosine) between the particular item and user vectors.

5.3 Hybrid Methods

The hybrid method involves a combination of both collaborative filtering approaches as well as content based methods. These methods have been proven to perform better than native collaborative filtering or content based approaches. Basically, the hybrid method involves combining the predictions that are obtained from collaborative filtering methods and content based approaches.

5.4 Multifaceted Collaborative Filtering Model

The two successful approaches to Collaborative Filtering are latent factor models, which directly profile both users and products, and neighborhood models, which analyze similarities between products or users. In the Multifaceted Collaborative Filtering Model, the factor and neighborhood models can be smoothly merged, thereby building a more accurate combined model. Further accuracy improvements are achieved by extending the models to exploit both explicit and implicit feedback by the users.

The Multifaceted Collaborative Filtering Model is sum of baseline, neighbourhood and latent factor models.

5.4.1 Baseline Model

Typical CF data exhibit large user and item effects i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. It is customary to adjust the data by accounting for these effects, which we encapsulate within the baseline estimates.

Denote by μ the overall average rating. A baseline estimate for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i$$

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average.

5.4.2 Neighborhood Model

Previous models were centered around user-specific interpolation weights $\frac{s_{ij}}{\sum_{j \in S(i,u)} s_{ij}}$ relating item i to the items in a user-specific neighborhood $S(i,u)$. In order to facilitate global optimization, we would abandon such user-specific weights in favor of global weights independent of a specific user. The weight from j to i is denoted by w_{ij} and will be learnt from the data through optimization. The model describes each rating r_{ui} by the equation:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj}) w_{ij}$$

However, experience shows that the current model somewhat overemphasizes the dichotomy between heavy raters and those that rarely rate. Better results were obtained when we moderated this behavior, replacing the prediction rule with:

$$\hat{r}_{ui} = b_{ui} + \frac{1}{|R(u)|} \sum_{j \in R(u)} (r_{uj} - b_{uj}) w_{ij}$$

Complexity of the model can be reduced by pruning parameters corresponding to unlikely item-item relations.

5.4.3 Latent Factor Model

A popular approach to latent factor models is induced by an SVD-like lower rank decomposition of the ratings matrix. Each user u is associated with a user-factors vector $p_u \in R^f$, and each item i with an item-factors vector $q_i \in R^f$. Prediction is done by the rule:

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i$$

Parameters are estimated by minimizing the associated squared error function.

5.4.4 Integrated Model

The new neighborhood model is based on a formal model, whose parameters are learnt by solving a least squares problem. An advantage of this approach is allowing easy integration with other methods that are based on similarly structured global cost functions. As explained, latent factor models and neighborhood models nicely complement each other. Accordingly, in this section we will integrate the neighborhood model with our most accurate factor model – SVD++. A combined model will sum the predictions of neighborhood and factor models to enrich each other:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i + \frac{1}{|R(u)|} \sum_{j \in R(u)} (r_{uj} - b_{uj}) w_{ij}$$

In a sense, this provides a 3-tier model for recommendations. The first tier, $\mu + b_u + b_i$ describes general properties of the item and the user, without accounting for any involved interactions. The next tier, $p_u^T q_i$ provides the interaction between the user profile and the item profile. The final "neighborhood tier" contributes fine grained adjustments that are hard to profile. Model parameters are determined by minimizing the associated regularized squared error function through gradient descent. We loop over all known ratings in K. For a given training case r_{ui}, we modify the parameters by moving in the opposite direction of the gradient.

6 Data

For the evaluation, we have basically used Movie Lens Datasets. Currently, we have the following datasets for evaluation

- **80,000 Dataset**
- **1 million Dataset**
- **10 million Dataset**

7 Approach

7.1 Recommendations Systems - Implicit and Explicit

We have implemented the following Algorithms for Recommendation Systems on the Movie Lens data set.

1. Matrix Factorization - Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space.
2. SVD - Approach which makes use of dimensionality reduction to improve performance for a new class of data analysis.
3. Memory Based Collaborative Filtering
4. Alternating Least Square
5. Multifaceted Collaborative Filtering Model - Combines the advantages of Matrix Factorization (SVD++) and Collaborative Technique using explicit features .

For the implementation for the above algorithms, we used python as the language and used various libraries for Matrix - Vector operations like Numpy, Scipy, Gensim. The main objective of this part of our project is to implement and compare the performance of the above mentioned algorithms and the impact on their results by fine tuning the hyper parameters.

7.2 Recommendations Systems - Scale Systems

With the data associated with recommendation systems becoming huge, we basically wanted to make the above algorithms scalable. For this , we implemented some of the above implemented algorithms on Fast Distributed Framework. We used Spark as the Framework which has libraries which make Matrix Computations easier and faster.

We implemented SVD as well as SVD++ and ALS using GraphX on spark. For the implementation for the above algorithms, we used Scala as the language .

8 Technology Stack

1. Spark
2. GraphX
3. MlLib
4. Numpy
5. Scipy
6. Scikit Learn
7. Scala
8. Python

9 Evaluation and Testing Method

We will review and evaluate our recommender system using Statistical Accuracy Metrics as follows:

- Root Mean Square Error (RMSE)
- Mean Absolute Error (MAE)

For each of the algorithms implemented, we have applied 10-fold cross validation on the data-set to choose the value of learning rate, regularization parameter and number of latent factors. The best model obtained after cross validation is used to obtain the final results.

10 Results & Observations

10.1 Memory Based Collaborative Filtering

For the memory based collaborative filtering, we have implemented user-based, item-based and hybrid-based collaborative filtering. For each of the three methods, three similarity functions (cosine, pearson, jaccard) are implemented. 10-fold cross validation is performed to obtain the best similarity measure to be used in each of three models. The results obtained can be seen in the Figure 4.

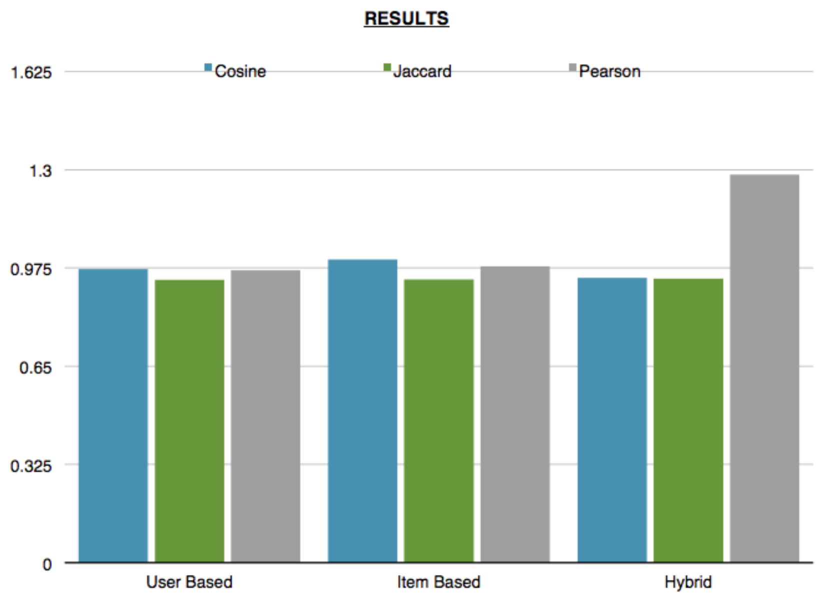


Fig : Bar Chart for Mean Squared Error for all the three types of similarities and for each of the user, item and hybrid based methods.

Figure 4: Bar Chart for Root Mean Squared Error for all the three types of similarities and for each of the user, item and hybrid based methods.

The bar chart shows that the Jaccard similarity will give us the least Root mean square error for all the three types : Item-Based, User-Based and Hybrid. Also, we also see from the bar chart above that cosine similarity performs the least for User based and Item based whereas Pearson performs the worst for Hybrid-based recommender.

For hybrid-based collaborative filtering, we have used a hybrid of user-based method and item-based method. First, we will calculate all the three similarities between the users using the users.csv file and all the three similarities between the items using the movies.csv. Now, we will apply cross validation to predict which similarity results in the lowest rmse. We found that the lowest rmse was obtained using jaccard similarity. Now, to predict a rating, we used Jaccard similarity to predict the rating for user and item based method, and then we took mean of the two ratings to obtain the final rating. This method ensures that we take into account the best rating obtainable from the user-based method and item based method. Lowest RMSE obtained in case of Hybrid is 0.9398.

10.2 Model Based Collaborative Filtering

For the model based collaborative filtering, we have considered the matrices of users and ratings available to us and performed Singular Value Decomposition (SVD). The results are obtained after applying 10-fold cross validation for various values of learning rate, reguralization parameter and low-rank. Refer the below table.

RMSE $r = 1$	$\lambda = 0.05$	$\lambda = 0.1$	$\lambda = 0.5$
$\mu = 0.0001$	0.9187	0.9184	0.9199
$\mu = 0.0005$	0.9967	0.9958	0.9841
$\mu = 0.001$	3.7921	3.9826	4.0196

RMSE $r = 3$	$\lambda = 0.05$	$\lambda = 0.1$	$\lambda = 0.5$
$\mu = 0.0001$	0.9167	0.9183	0.9176
$\mu = 0.0005$	0.9334	0.9346	0.9190
$\mu = 0.001$	4.5996	4.6174	4.5064

RMSE $r = 5$	$\lambda = 0.05$	$\lambda = 0.1$	$\lambda = 0.5$
$\mu = 0.0001$	0.9202	0.9211	0.9210
$\mu = 0.0005$	0.9485	0.9430	0.9295
$\mu = 0.001$	3.1634	3.2539	3.0749

Now, some of the other observations are:

1. When we vary r, we observe the following:
   - As we increase the value of r from 1 to 3, we observe that the RMSE decreases. However, when we increase the value of r from 3 to 5, we observe that the RMSE increases.
   - The time taken to run the program increases as we increase the value of r. This is due to the increased dimensions of the user matrix and movies matrix.

- As we increase r, the best RMSEs are obtained when μ is 0.0001 and the worst RMSEs when μ is 0.001. In case of μ being 0.001, the learning step is too big, and as a result gradient descent is not able to converge to the global minima.
- Also, in general, we will observe that by increasing the value of r above 5, the value of RMSE increases for a fixed value of μ, λ and number of iterations i.e. the stopping criteria.

2. Best Model is given by the following table:

Best Model	Values
μ	0.0001
λ	0.05
r	3
RMSE	0.9167

We choose this model as the best model because it gave the minimum average RMSE after preforming cross validation for various values of r, μ and λ.

3. In real systems, when we are using regularized Matrix Factorization, we will use the technique of cross-validation to choose parameters. We will use the 10-fold cross validation on the given training set for the various values of r, μ and λ. The parameters which will give the minimum RMSE will be chosen for the final model to predict the new ratings. Taking the parameters corresponding to the least RMSE will ensure that our final model is more accurate in predicting new ratings and thus produces less error. Also, this will choose the model which will not over-fit the given data.

10.3 Multifaceted Collaborative Filtering Model

For the multifaceted collaborative filtering model, we have integrated baseline, item-based neighborhood and SVD++ models in a single combined model. Then, later we applied 10-fold cross validation to choose the rank (number of latent factors), learning rate and regularization parameter. We choose learning rate to be 0.00001 and regularization parameter to be 0.05. The results obtained are shown in the following tables.

rank	1	3	5
RMSE	0.9572	0.9437	0.9573

Table 1: Baseline Model

rank	1	3	5
RMSE	0.9427	0.9357	0.9491

Table 2: Neighborhood Model

rank	1	3	5
RMSE	0.9371	0.9281	0.9323

Table 3: Latent Factor Model

rank	1	3	5
RMSE	0.9213	0.9147	0.9284

Table 4: Integrated Model

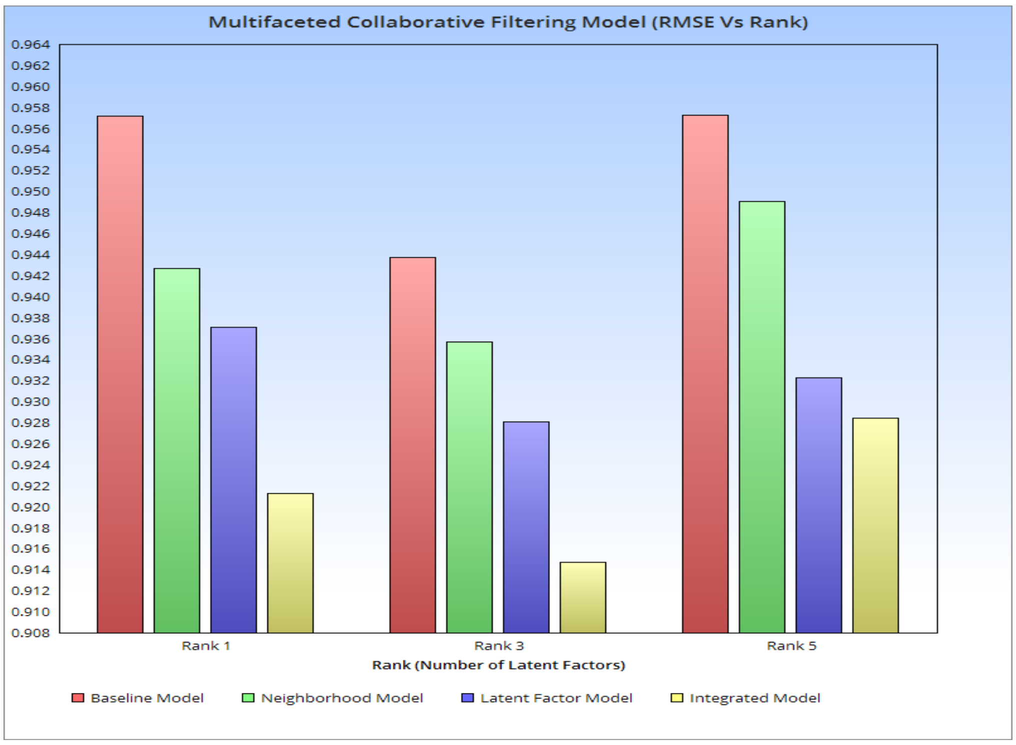


Figure 5: Results for multifaceted collaborative filtering model

These results can be further improved by incorporating implicit data in the integrated model.

10.4 Large Scale Recommendation Using Spark

Here , as mentioned , we have implemented ALS, SVD/PCA with Dimensionality reduction and SVD++ . Our emphasis here is on computing and evaluating the time complexity with increase in the size of dataset as well as with increase in the number of iterations.

First, we have plotted the graph(Figure 6) for using ALS with increase in the number of iterations on spark. As, we can see that both of them scale well, however for bigger dataset there is a spike in the end indicating higher communication cost.

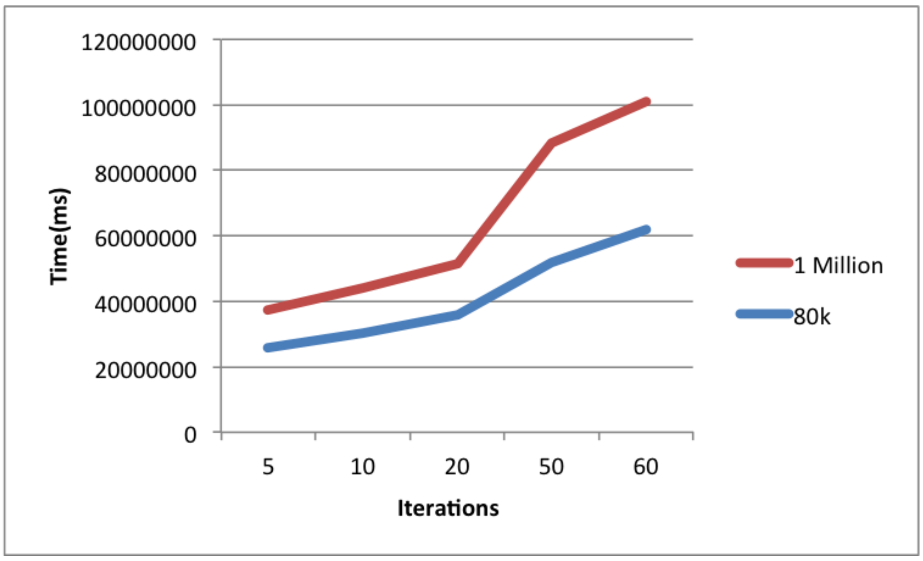


Figure 6: Time(micro sec) to execute with increase in number of iterations

Next, we plot the RMSE values along with increasing rank. This basically indicates that with increase in rank and varying the regularization parameter accordingly, the RMSE value keeps decreasing as shown in the Figure 7 till we reach an optimal number of latent features (rank represents the number of latent features). Beyond this threshold of optimal latent value, the RMSE starts increasing. In our example this value is 20. We also attempt to show the impact of increasing the rank on the time taken to complete running the ALS algorithm. As shown in the Figure 8, we observe that the run time increases with the rank.

10.5 Evaluation - Hadoop vs Spark

For Spark's comparison with Hadoop we did the following experiment. For the evaluation, we considered the EDI files, which is a health care data file and ran the experiment using Hadoop, Hive for finding Regular Expression in an EDI File. The results are as mentioned in the table. Comparison was done by using 4 Worker nodes of 2 GB compared to single machine having 8GB ramp with a single node Hadoop which can be considered as a Spark Node . We did not use the Spark because we didnt have access to setup up spark on that Cluster. As we can see as the data becomes huge, Hadoop tends to outperform single machine architecture. In the graphs we basically analyze the time required to analyze a single file or claim.

Initially , time is quite high for 5 node system as most of the time is lost in Communication and communication cost outweighs computation cost, which gradually decreases with more amount of data. As we can observe from the

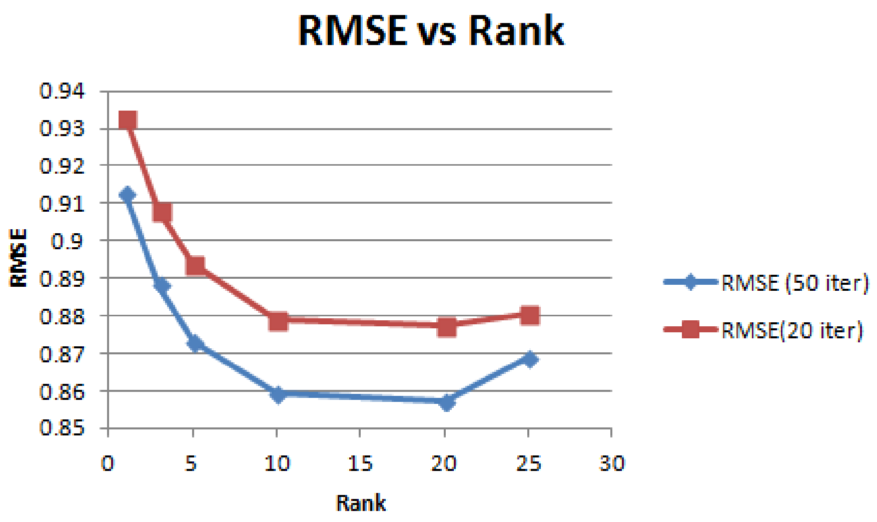


Figure 7: RMSE plotted against rank

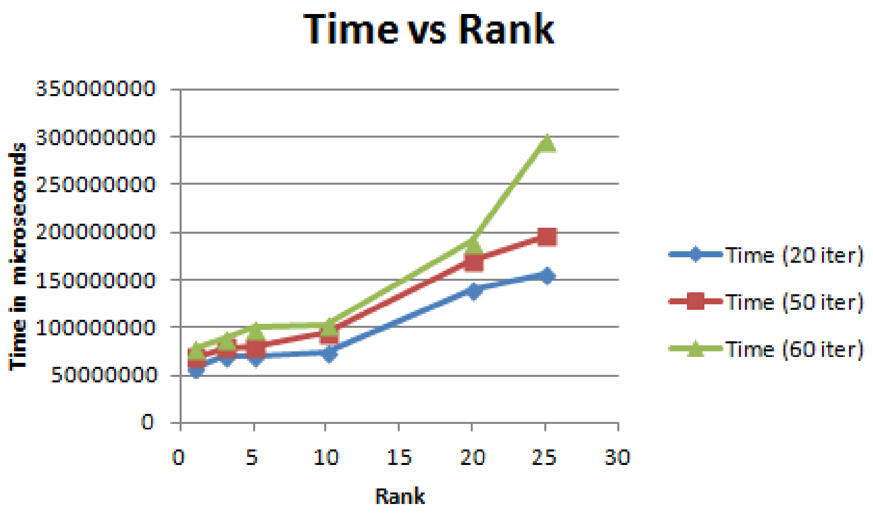
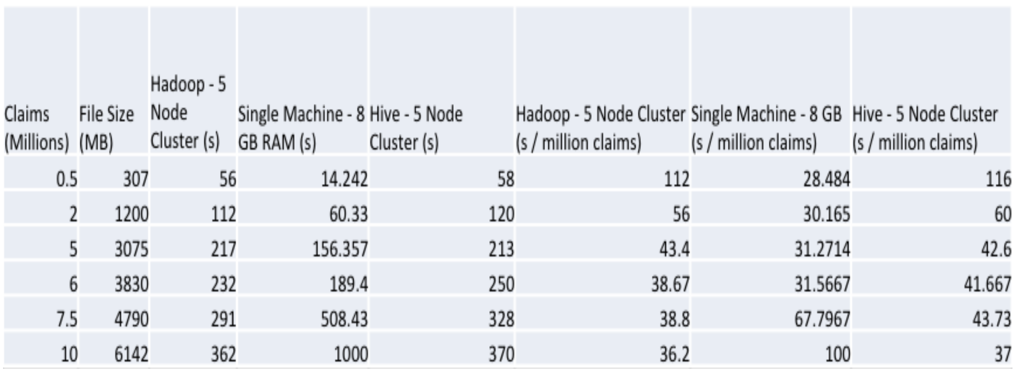


Figure 8: Time taken plotted against rank

graphs, time per claim is very good for a powerful machine but it then explodes after a certain threshold when it no longer can hold data in memory .



Claims (Millions)	File Size (MB)	Hadoop - 5 Node Cluster (s)	Single Machine - 8 GB RAM (s)	Hive - 5 Node Cluster (s)	Hadoop - 5 Node Cluster (s / million claims)	Single Machine - 8 GB (s / million claims)	Hive - 5 Node Cluster (s / million claims)
0.5	307	56	14.242	58	112	28.484	116
2	1200	112	60.33	120	56	30.165	60
5	3075	217	156.357	213	43.4	31.2714	42.6
6	3830	232	189.4	250	38.67	31.5667	41.667
7.5	4790	291	508.43	328	38.8	67.7967	43.73
10	6142	362	1000	370	36.2	100	37

Figure 9: Table for Results on Increasing Size Problem

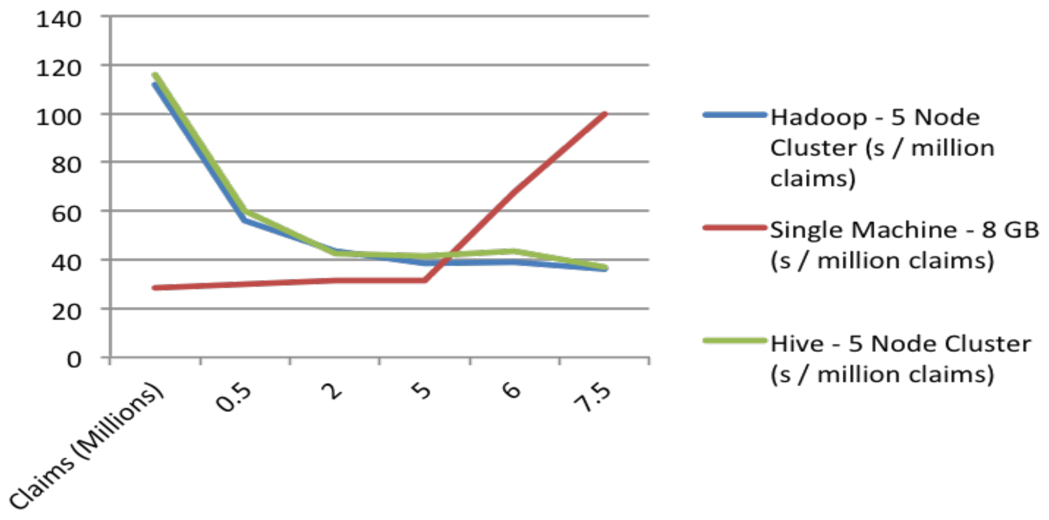


Figure 10: Graph for Results on Increasing Size Problem

10.6 Code

Code for all the models implemented in this project is available on Github – https://github.com/gkrishnan/largeScaleML.

11 Future Work

We plan to incorporate more implicit Information, for instance data such as the movie watched count as a boolean field. We achieved similar results as the earlier models without much improvement. We believe this is primarily due to lack of enough implicit information provided as an input to the framework. Thus, given more information such as user lists, activities which is available in the industry, our system would be able to perform even better than the earlier versions. We also plan to incorporate social network graph information from data sources like Facebook and Twitter to further strengthen the model.

12 Conclusion

We implemented various collaborative filtering techniques in this project, ranging from memory based techniques like user and item based recommendations to model based techniques like matrix factorization and SVD. Explicit information has been extensively used in this project for recommender systems, but we need to incorporate more implicit data for predictions. It is difficult to obtain implicit data because of the proprietary nature of this data. We also did a detailed evaluation on different dataset sizes of the performance of different number of nodes using the Hadoop family. We could clearly observe the performance benefit of a multiple node cluster as the size of the dataset increases, whereas in case of small datasets it could be observed that the single node implementation performs better. We have also evaluated algorithms such as Alternating Least Squares and SVD++ on Apache Spark. We could clearly observe a significant performance gain while using Spark as compared to Python based implementations.

References

1. Yehuda Koren. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

2. J. Tang, J. Sun, C. Wang, and Z. Yang. Social influence analysis in large-scale networks. In KDD, 2009.

3. User-based Collaborative-Filtering Recommendation Algorithms on Hadoop

4. Guy Shani and Asela Gunawardana. Evaluating Recommendation System

5. Jonathan Herlocker, Joseph Konstan, Loren Terveen, and John Riedl. Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems, Vol. 22, No. 1, January 2004, Pages 5-53

6. Social collaborative filtering for cold-start recommendations. S Sedhain, S Sanner, D Braziunas, L Xie - Proceedings of the 8th, 2014 - dl.acm.org

7. Feng, Qinyuan, et al. "Enhancing personalized ranking quality through multidimensional modeling of inter-item competition." Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on. IEEE, 2010.

8. Herlocker, Jonathan L., et al. "Evaluating collaborative filtering recommender systems." ACM Transactions on Information Systems (TOIS) 22.1 (2004): 5-53.