The identifying hidden data features problem solution

To cite this article: S Y Petrova and M A Boikova 2019 *J. Phys.: Conf. Ser.* **1352** 012039

The identifying hidden data features problem solution

S Y Petrova and M A Boikova

Yaroslav-the-Wise Novgorod State University, ul. B. St. Petersburgskaya, 41
173003 Veliky Novgorod, Russia

E-mail: svetayp@list.ru

Abstract. In the article, we considered recommender models based on matrix factorization demonstrate excellent performance in collaborative filtering. The standard Matrix Factorization approach in MLlib deals with clear ratings. To work with implicit data, we used the trainImplicit method. To simulate the processing of real-time data streams, we used the Spark Streaming library, which is responsible for receiving data from the input source and converting the raw data into a discretized stream discretized stream (DStream) consisting of Spark RDD. The rank parameter determines the number of hidden features in the low rank approximation matrices. As a rule, the greater the number of factors, the better, but for a large number of users or elements, it will directly affect the memory usage of the computing system and the amount of data required for training. Therefore, in our problem it was a compromise solution.

1. Introduction

The global development of Internet commerce has led to the active development of various types of recommendation systems that help users find products of interest to them, without spending much time on it. There are a huge number of services that recommend the most diverse elements to users: music [1], [2], films [3], books [4], [5], [6], video [7], news [8] and [9]. The most famous examples of such recommender systems are Amazon [10], Ebay [11] and AliExpress.

The principle of making recommendations of these sites is based on the user's search history and the history of past purchases. Both users and providers of advisory services benefit from the use of recommender systems, since an effective recommender system can increase company revenues and facilitate user interaction in online communities.

The overall objective of recommender systems is to improve the quality of customer service with the help of personalized recommendations based on preliminary explicit and implicit evaluations. The numerical value of the recommendation with explicit feedback indicates user preference, while the numerical value of the recommendation with implicit feedback indicates the validity of the recommendation.

Recommender systems based on explicit ratings often use the average rating of a product or service, which was obtained from evaluations of products supplied by users. This approach has several important practical advantages. It gives you the opportunity to work with relatively simple models of source data. For example, in content algorithms, it is perfectly acceptable to be limited to tables of the type "attribute-value of a characteristic". In the framework of this approach, it is assumed that each object is identified with a certain point of the multidimensional attribute space, the class of objects is represented by a correspondingly compact set of such points, and the recognition problem is reduced to constructing a deterministic or probabilistic decision rule relating new objects (points) to one of the classes of nonintersecting domains of indicative space. The main feature of this approach from the point of view of the description of filtering objects is that the formal descriptions - models of objects - use the sets formed by the features of the objects; relations between signs, even if they are known, are usually not part of these sets. Also, explicit feedback is not always available, and in such systems there is an insoluble problem of a "cold start" - this is when a new product does not have a rating.

Thus, the preferred method is to develop recommendations based on implicit estimates. These systems passively track various types of user actions, such as viewing a product, working with a basket, placing an order, and purchasing a product. Shopping history and repeated user behavior when working with the site allows you to develop a model of user preferences. The analysis of data obtained using the implicit approach is rather complicated, since the behavior of users is not deterministic. So if the user often performs the same action with certain goods, this does not indicate his higher preference. A one-time event can be caused by various reasons that have nothing to do with user preferences, for example, a user can watch a movie once, or watch a TV show every week, while evaluating the film and TV show equally well. However, the numerical value of an implicit rating is definitely useful, since it tells us about the confidence that we have in a certain observation.

2. Results and Discussion

The main component of the recommender system is the filtering algorithm. The algorithm converts a pre-structured input data set into a rating matrix. Latent factor models include: Latent factor model (LFM), neural networks, latent Dirichlet allocation, matrix factorization.

Since Spark recommender models based on matrix factorization demonstrate excellent performance in collaborative filtering [12], we focused our attention on this class of models. The goal of collaborative filtering based on factorization is to identify the hidden features of the data that explain the ratings.

In collaborative filtering systems, the utility $u\ (i;\ j)$ of element j for user i is estimated based on the utilities $u\ (i';\ j)$ assigned to element j by these users $i' \in \mathcal{S}$, which are "similar" to user i, where $\mathcal{S}$ represents a set of users [16].

When we deal with data that consists of explicit user preferences, such as ratings, approvals, likes, etc., we can use Explicit matrix factorization. For our case, this approach is not applicable, since we do not have explicit estimates, so we applied Implicit Matrix Factorization.

There are many different approaches to working with implicit data. MLlib implements a special approach that maps users i' 'and elements into a space of hidden factors in d dimensions, where d represents the dimension of space. In the space of hidden factors, the interaction between the user and the element is modeled as internal ratings: the preference matrix $\mathcal{P}$ and the trust weight matrix $\mathcal{C}$.

More formally, each element is represented by a vector $q_j \in \Re^d$, and each user is represented by a vector $p_i \in \Re^d$. Each rating is predicted by the product of the vectors given in equation 1, and the learning error measured for the $(i;\ j)$-th rating, which is assigned to element j by user i, is defined in equation 2 [13], [14] :

$$\hat{r}_{ij} = q_j^T p_i\ , \tag{1}$$

$$e_{ij} = r_{ij} - \hat{r}_{ij}\ \ , \tag{2}$$

where $\hat{r}_{ij}$ denotes the rating of the i-th user, which he would give to the j-th element, and e_{ij} denotes the learning error for this pair of user element.

The key to the model is the search for these vectors q_j and p_i.

To find out the factor vectors p_i and q_j, the model must minimize the regularized error on the set of known $\Re$ equation 3 [14]:

$$\text{ratings, as indicated in } \underset{q^*,p^*}{\text{Emin}} \sum\nolimits_{(i,j)\in R}\left(r_{ij} - q_j^T p_i\right)^2 + \lambda\left(\left\|q_j\right\|^2 + \left\|p_j\right\|^2\right). \tag{3}$$

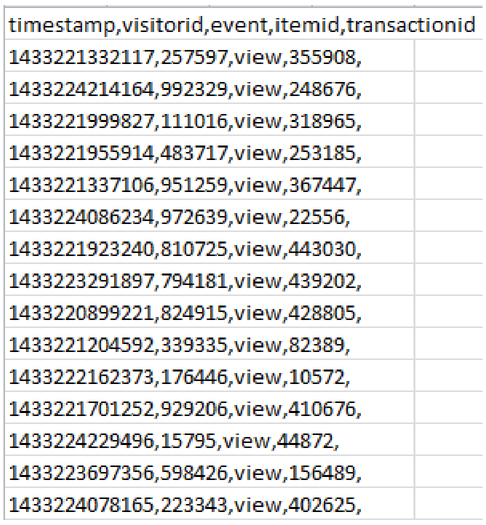
In Equation 3, R denotes a known set of ratings. The system studies the model by selecting previously defined ratings. However, it is very important to be able to make predictions for new products that do not have a rating; so the system should avoid retrofitting by regularizing the parameters studied. The constant λ in this equation controls the degree of regularization. Suppose that the product rating is actually the amount of purchase of this product by each user. To design a model, we need to reveal hidden relationships between users and products.

The implicit model creates two matrices: the matrix $\mathcal{P}$ informs us that the product was purchased by the user, and the matrix $\mathcal{C}$ represents the weighting factor of reliability in the form of the number of purchases (as a rule, the more users bought this product, the higher the reliability of what they really like it). The matrix that the model is trying to approximate is the preference matrix $\mathcal{P}$.

As a result, calculating the dot product of the user vector and the factor element will result in a user preference estimate for the element. The closer these scores are to zero, the higher the accuracy of the user's preference estimate. The high accuracy of the assessment of user preferences or the correspondence between user and product factors leads to recommendations.

To illustrate the concepts described above, we used a set of data describing user behavior on the e-commerce site, in the form of page views and product purchases. An example of the content of the source file is shown in figure 1. It contains a raw data set, in which each record is a string, separated by commas, representing an event on an e-commerce site:

– Timestamp – time stamp. Part of the timestamp is in the Unix-era (timestamp) format, for example, 1548835990 will be converted to Wednesday, January 30, 2019 8:13:10 GMT.
– Visitorid (visitor id) – unique user browsing a website.
– Event – user action, which may include: viewing a product, adding a product to the cart, buying.
– Itemid – a unique product identifier for a specific e-commerce site.
– Transactionid - transaction ID that will matter only if the user has made a purchase: 1433193915008, 552148, transaction, 81345, 5444.



```
timestamp,visitorid,event,itemid,transactionid
1433221332117,257597,view,355908,
1433224214164,992329,view,248676,
1433221999827,111016,view,318965,
1433221955914,483717,view,253185,
1433221337106,951259,view,367447,
1433224086234,972639,view,22556,
1433221923240,810725,view,443030,
1433223291897,794181,view,439202,
1433220899221,824915,view,428805,
1433221204592,339335,view,82389,
1433222162373,176446,view,10572,
1433221701252,929206,view,410676,
1433224229496,15795,view,44872,
1433223697356,598426,view,156489,
1433224078165,223343,view,402625,
```

Figure 1. Fragment of the source CSV file.

The data set for processing in the recommender system is taken from the Kaggle Datasets [15] public data platform. Data was collected from a real e-commerce website. This is the raw data, that is, without any content conversion. To ensure confidentiality, all values are presented in a hashed form.

The data set contains data on user behavior, that is, events such as clicks, adding to the cart, and transactions that were collected during the 4.5 months of 2015. For example:

– «1439694000000, 1, view, 100» means the user with visitorId = 1, selected item with id = 100 at 1439694000000 (time stamp Unix).

– «1439694000000, 2, transaction, 1000, 234» means the user with visitorId = 2 purchased item with id = 1000 in a transaction with id = 234 at 1439694000000.

The visitor can make three types of events, namely: viewing the product, adding to the cart or buying a product. In total, the data set includes: 1,048,575 events, including 1,013,165 product views, 26,756 additions to the cart and 8,654 purchases made by 556,528 unique visitors to the e-commerce site, assuming that there are no similar users with different visitor identifiers. The data set contains 157,270 unique products.

To simulate the processing of real-time data streams, we used the Spark Streaming library, which is responsible for receiving data from the input source and converting the raw data into a discretized stream discretized stream (DStream) consisting of Spark RDD. We then created a custom Spark Streaming application that would handle this stream of events. We first created a StreamingContext and a socket stream, and then applied a map transformation to break up the raw text and create a tuple (user, product, event). Next, we used foreachRDD to apply arbitrary processing to each RDD in the stream to calculate the desired metrics and output them to the console.

To reduce the number of variables for which machine learning will be conducted, we will create a low-dimensional representation of the rating matrix. That is, to get q and p, we reduced the matrix S to size k using the singular value decomposition (SVD) of the matrix M: $m\ x\ n$ (real or complex) is factorization of the form $U \sum V^*$, where U is the matrix $m \times R$. Σ is a rectangular diagonal $R \times R$ matrix with non-negative real numbers on the diagonal, and V is the $n \times r$ identity matrix. r is equal to the rank of the matrix M.

After the raw data was transformed, the model was trained. In our experiment to train the model, the data set was split into two parts: the training data set 80%, and the correction data set 20%. The training data is used in the algorithm to study the characteristics of the data set, and the test and test data are not fully visible to the algorithm. The test data set is then used to make improvements and is used as a buffer, but here the test data is still not visible.

Dividing a data set into a training data set and a testing data set with a ratio of 80:20 is shown in the following lines of code:

```
val Array(training, test) = ratings.randomSplit(Array(0.8, 0.2))

training.cache()

test.cache()
```

At the output of RDD, we had the following scheme: userID - Int, itemID - Int, rating - Double. Product rating is calculated based on user interaction histories with this product.

The standard Matrix Factorization approach in MLlib deals with clear ratings. To work with implicit data, we used the trainImplicit method with the following parameters:

```
val rank = 10 // the number of hidden factors in the ALS model

val numIterations = 10 // number of iterations of the ALS algorithm

val alpha = 0.01 // basic confidence in observing preferences

val lambda = 0.01 // regularization parameter

val model = ALS.trainImplicit(ratings, rank, numIterations, lambda, alpha)
```

The rank parameter determines the number of hidden features in the low rank approximation matrices. As a rule, the greater the number of factors, the better, but for a large number of users or elements, it will directly affect the memory usage of the computing system and the amount of data required for training. Therefore, in our problem it was a compromise solution.

Usually, the convergence of the ALS model to a fairly good solution occurs after a relatively small number of iterations. And although each iteration in ALS is guaranteed to reduce the recovery error of the score matrix, the parameter numIterations = 10 is a good default value.

An additional parameter, alpha, was also set, controlling the basic confidence level by applying weighting. A higher alpha level makes the model more confident that the missing data is equivalent to the lack of preferences for the corresponding user-element pair.

The value of the lambda regularization parameter was chosen using test methods and cross-validation.

Teaching such a model, we tried to find the best rank approximation d for the observed target matrix R of size $N \times M,$ where d is the number of hidden vector dimensions, N and M are the number of users and elements in the system, respectively. As a result of the work of the model, the recommendation takes the form of a list of top-k, that is, a set of k-elements that are most likely to be liked by the user. This is done by calculating the predicted score for each item and ranking the list based on that score.

3. Conclusion

To test the performance of the recommender system, the following accuracy metrics were used: MSE (Mean Square Error) - root-mean-square error; RMSE (Root Mean Square Error) - standard deviation.

These metrics allow us to estimate the difference between the real rating and the rating predicted by the recommender system. The calculation of errors occurred on the data set from the test set.

We define the vector of predictions as n, and the vector of the observed values of the predicted variable Y, then the MSE metric is calculated by formula 4 [17]:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \acute{Y}_i)^2). \tag{4}$$

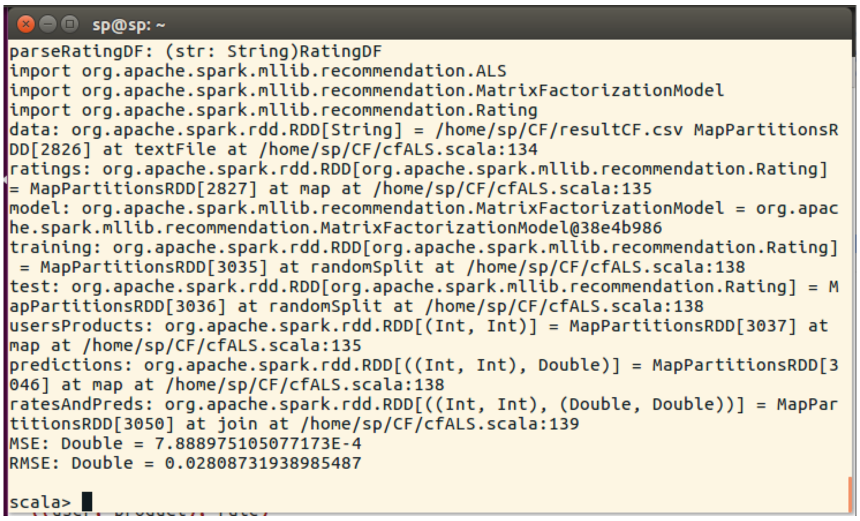
The MSE value tends to zero. If MSE = 0, then the system predicts the parameter with perfect accuracy, which is usually impossible.

RMSE is calculated by extracting the square root of the mean square difference between the prediction and the actual value using the formula 5 [17]:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (Y_i - \acute{Y}_i)^2)}. \tag{5}$$

This metric is one of the popular metrics, since the errors identified by this metric can have a significant impact on the user's decision. The value of RMSE is always non-negative and the closer to zero, the more perfect it corresponds to the data. Low RMSE values are preferred.

As a result of the cross-validation and selection of optimal model hyper-parameters for the collaborative filtering algorithm, the following values of accuracy metrics were obtained: MSE = $7.89*10^{-4}$, RMSE = 0.028, as shown in figure 2.



```
parseRatingDF: (str: String)RatingDF
import org.apache.spark.mllib.recommendation.ALS
import org.apache.spark.mllib.recommendation.MatrixFactorizationModel
import org.apache.spark.mllib.recommendation.Rating
data: org.apache.spark.rdd.RDD[String] = /home/sp/CF/resultCF.csv MapPartitionsR
DD[2826] at textFile at /home/sp/CF/cfALS.scala:134
ratings: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating]
= MapPartitionsRDD[2827] at map at /home/sp/CF/cfALS.scala:135
model: org.apache.spark.mllib.recommendation.MatrixFactorizationModel = org.apac
he.spark.mllib.recommendation.MatrixFactorizationModel@38e4b986
training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating]
 = MapPartitionsRDD[3035] at randomSplit at /home/sp/CF/cfALS.scala:138
test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.recommendation.Rating] = M
apPartitionsRDD[3036] at randomSplit at /home/sp/CF/cfALS.scala:138
usersProducts: org.apache.spark.rdd.RDD[(Int, Int)] = MapPartitionsRDD[3037] at
map at /home/sp/CF/cfALS.scala:135
predictions: org.apache.spark.rdd.RDD[((Int, Int), Double)] = MapPartitionsRDD[3
046] at map at /home/sp/CF/cfALS.scala:138
ratesAndPreds: org.apache.spark.rdd.RDD[((Int, Int), (Double, Double))] = MapPar
titionsRDD[3050] at join at /home/sp/CF/cfALS.scala:139
MSE: Double = 7.888975105077173E-4
RMSE: Double = 0.02808731938985487

scala>
```

Figure 2. Metrics of accuracy of the recommendation system.

Implementing a machine learning simulation model using implicit data has shown that users tend to choose products that are next to each other in a hidden vector space. Small values of the MSE and RMSE parameters indicate a high accuracy of the forecast and that this algorithm can be better scaled compared to other approaches, thanks to the flexibility in solving real problems.

References

[1] How it works? Recommendations in Yandex.Music. [Official web site Yandex] 2019 Available at: https://yandex.ru/blog/company/92883. (accessed 26.06.2019)

[2] Bischoff K 2012 We love rock'n'roll: analyzing and predicting friendship links in Last. fm, in *Proceedings of the 3rd Annual ACM Web Science Conference*. ACM 47–56

[3] Bennet J and Lanning S 2007 "The Netflix Prize" [KDD Cup and Workshop] Available at: www.netflixprize. com. (accessed 26.06.2019)

[4] ReadRate service - new features for PocketBook users 2019 Available at: http://www.pocketbook-int.com/ua/uk/node/7531.(accessed 25.06.2019)

[5] Savenkov K 2019 The recommendation system as a means of achieving business goals [Bookmate] Available at: https://events.yandex.ru/lib/talks/2939/ (accessed 25.06.2019)

[6] Koltsov S Draft 2019 MyBook, designed to increase interest in reading Available at: http://rastudent.ru/articles/internet_and_advertising/v_moskve_predstavili_proekt_mybook_prizvannyy_povysit_interes_k_chteniyu (accessed 25.06.2019)

[7] Covington P, Adams J and Sargin E 2016 Deep Neural Networks for YouTube Recommendations *RecSys '16 Proceedings of the 10th ACM Conference on Recommender Systems* 191–198

[8] Liu J, Dolan P and Pedersen E R 2010 Personalized News Recommendation Based on Click Behaviour *IUI '10 Proceedings of the 15th international conference on Intelligent user interfaces* 31–40

[9] Das A, Datar M, Garg A and Rajaram S 2007 Google news personalization: Scalable online collaborative filtering I*n Proc. of WWW'07: the 16th International Conference on World Wide Web* Banff Alberta Canada 271–280

[10] Linden G, Smith B and York J 2003 Amazon.com recommendations: Item to item collaborative filtering *IEEE Internet Computing* Vol **7** **1** 76–80

[11] Sundaresan N 2011 Recommender systems at the long tail *RecSys '11 Proceedings of the fifth ACM conference on Recommender systems* 1–6

[12] Koren Y, Bell R and Volinsky C 2009 Matrix factorization techniques for recommender systems *IEEE Computer* **42 (8)** 30–37

[13] Koren Y 2010 Collaborative filtering with temporal dynamics *Communications of the ACM* **53 (4)** 89–97
[14] Takacs G, Pilaszy I, Nemeth B and Tikk D 2008 Matrix factorization and neighbor based algorithms for the netflix prize problem *In Proceedings of the 2008 ACM conference on Recommender systems* 267–274
[15] Kaggle Datasets [Official web site] 2019 Available at: https://www.kaggle.com/retailrocket/ecommerce-dataset (accessed 26.06.2019)
[16] Adomavicius G and Tuzhilin A 2005 Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions *IEEE transactions on knowledge and data engineering* 734–749
[17] Dua R, Ghotra M S and Pentreath N 2017 Machine Learning with Spark, Second Edition *Develop intelligent machine learning systems with Spark 2.x* Packt Publishing Ltd. UK