Recommender System as a Service based on the Alternating Least Squares algorithm

Gašper Slapničar
Jožef Stefan Institute,
Department of Intelligent Systems
Jamova cesta 39
1000 Ljubljana
+386 51 721 041
slapnicar.gasper@gmail.com

Boštjan Kaluža, Mitja Luštrek
Jožef Stefan Institute, Department of Intelligent Systems
Jamova cesta 39
1000 Ljubljana
+386 1 477 3944
{bostjan.kaluza, mitja.lustrek}@ijs.si

Zoran Bosnić
University of Ljubljana, Faculty of Computer and Information Science
Večna pot 113
1000 Ljubljana
+386 1 479 8237
zoran.bosnic@fri.uni-lj.si

ABSTRACT

In this paper, we describe a production-ready recommender system as a service for recommending eco-friendly tourist accommodations. It offers two main features: (1) it returns personalized recommendations for a user by creating a latent factor model through matrix factorization (Alternating Least Squares algorithm, ALS) and (2) it returns accommodations that are similar to a given accommodation by calculating content-based similarity using the Jaccard coefficient and the Euclidian distance. The system is evaluated on the collected data by using cross-validation and Precision@k as a performance measure. It achieves 19% Precision@k for personalized recommendations to a user based on his past interactions with accommodations. This score far surpasses a random recommender implementation that achieves 1% Precision@k.

Keywords

Recommender system, parallel computing, machine learning, big data, matrix factorization

1. INTRODUCTION

Due to significant growth and evolution of e-Commerce and digitalization in many fields (medicine, economics, etc.) in the past years, the size and complexity of collected data is growing rapidly. Data are being generated in real time and are mostly unstructured. Such data collections are thus being referred to as the "big data".

Alongside the development of big data technologies, which can successfully store and process large amounts of unstructured data in real time, companies want to use these technologies in machine learning and offer machine learning algorithms to users in a simple way, as a service. This led to recent development of many Machine Learning as a Service (MLaaS) providers.

MLaaS platforms can be used to develop recommender systems, which are an essential part of e-Commerce and can bring a significant competitive advantage. These systems focus on creating personalized recommendations which should include items that will most likely interest a potential customer. Since e-Commerce and data science are evolving rapidly, experts from different fields are required for development of a production grade recommender system. This can be addressed by developing the system in a cloud and thus ensuring separation of concerns.

We address the problem of developing such a recommender system by dividing in into two parts. The first part is data collection. We record the user-item interactions in the event-based style, where each interaction corresponds to an action of the user on the web portal and is an element of a predefined list. User is represented by a unique tracking id while items correspond to a list of unique accommodation ids.

The second part and core of the problem are the recommendation algorithms. For customized recommendations to a user based on past actions, we create a latent factor model using matrix factorization. The model learns by using Alternating Least Squares algorithm. We propose an algorithm that can be efficiently executed in parallel and is a good candidate to use with distributed big data technologies. For recommending similar accommodations based on their properties we use the Jaccard coefficient and normalized Euclidian distance merged together into a common similarity measure.

2. BACKGROUND

A large number of MLaaS platforms emerged recently. These support a wide variety of machine learning algorithms out of the box and can be used to develop a recommender system. In the following subsections we overview considered platforms, and describe the outline of our proposed approach.

2.1 MLaaS PLATFORM

First, we compared several MLaaS platforms, such as BigML, Google Prediction API, Azure ML, Amazon ML and Prediction.IO. Due to white-box design and open source access we chose Prediction.IO machine learning server. It is implemented as a distributed scalable stack based on latest big data technologies, such as Apache HBase, Apache Hadoop, Apache Spark and Elasticsearch [3].

Since it implements Apache Spark's MLlib, it comes with native support for many algorithms that are suitable for development of a recommender system. It also offers *templates* which are implementations of some machine learning algorithms on actual problem domains and are available at templates.prediction.io/ [3].

Prediction.IO machine learning server consists of three main parts [3]:

1. **Prediction.IO platform** – open source machine learning stack for creating ***engines*** with machine learning algorithms,
2. **Event Server** – API for collecting events from different sources and unifying the format,
3. **Template Gallery** – templates with implementations of machine learning algorithms on real problem domains.

Figure 1 shows how clients interact with the machine learning server and it also shows that the server can have several engines, each corresponding to one machine learning application.

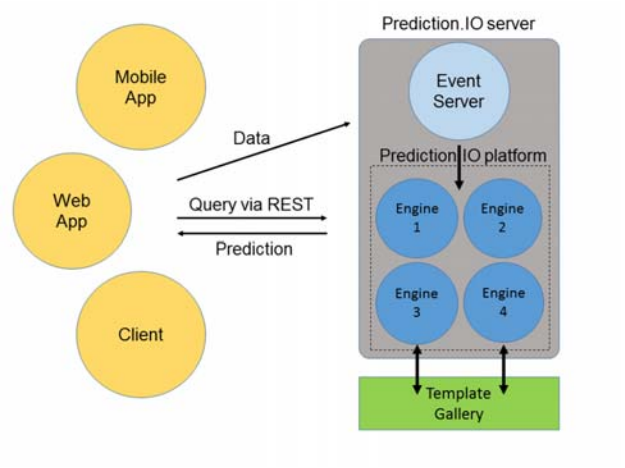

Figure 1: Conceptual architecture and interaction between Prediction.IO server and client.

MVC (model – view – controller) architectural pattern is well established in web development for years, since it helped to speed up the development process and lowered the learning curve.

It would make sense to implement such a pattern in data science, especially in machine learning. Prediction.IO represents one of the first attempts of this with their DASE engine architecture. Each engine follows this architectural pattern and must contain all the components with the exception of Evaluation, which is optional:

- **D**ata Source/**D**ata Preparator – reads data from an input source (database) and transforms it into a desired format,
- **A**lgorithm – the machine learning algorithm used to create the predictive model,
- **S**erving – takes client queries and returns prediction results,
- **E**valuation – quantifies prediction success with a numerical score.

2.2 ALGORITHMS

The "Netflix prize" competition has demonstrated that within collaborative filtering, latent factor models are the most successful method for recommending products. This method is gaining popularity, due to good scalability and better precision in comparison with neighborhood methods [1, 2].

Latent factor models are most successfully implemented by using matrix factorization. Two of the best known algorithms for solving the matrix factorization problem are Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS). A lot of effort has been put towards parallelization of the SGD algorithm. This has proven to be a difficult problem. ALS is an algorithm that is closely related to SGD and offers high level of potential parallelization [2].

3. DATA AND RECOMMENDATION TASK

The data are collected from a website, which offers eco-friendly accommodations. Any accommodation which meets 5 out of 10 required criteria is considered eco-friendly. Examples of these criteria are re-usage of water, usage of solar energy, waste recycling, etc.

Training data are being collected in real time in event-based style. This means that each action that a user does on the web portal, corresponds to a single user-item interaction.

User – action – item format was chosen to describe interactions between users and accommodations as it comprises all the required information. Each event on the web portal can be represented with this format. *User* corresponds to a unique user performing actions on the web portal and is traced by using a long lasting cookie id. *Action* is whatever this user does on the web portal and it corresponds to the user-item interaction. It is an element of a predefined list containing all relevant possible actions on the web portal. *Item* corresponds to unique identifier of the accessed object and is an element of a predefined set of all existing accommodation ids.

Example of this format is given as: user *U1 views* accommodation *I1* (*U1 – view – I1*).

A custom API was developed in order to implement basic authentication and authorization together with data sanity check. This API connects to the Event Server of Prediction.IO machine learning server and saves the collected data to HBase data store in real time. Each event corresponds to an HTTP POST request which contains parameters corresponding to *user – action – item* format. It also contains the timestamp of the event.

Different feedback mechanisms can be used to record user-item interactions. Recommender systems work best with data collected through explicit feedback mechanism such as ratings. Due to the implementation of the web portal, only implicit feedback mechanism is available (e.g. *views*, *inquiries*). A mapping was implemented to map the collected implicit data to explicit numerical ratings on the scale of 2 to 5. This is explained further in the following section.

Due to the design of the web portal, a user can either do a complete search from the landing page or he can access the page with accommodation details directly from a web search engine. In first case the web portal design allows us to implement recommendations for this specific user that can be displayed with the search results. In second case we are limited by design to recommend similar accommodations to the currently viewed accommodation. This also makes more sense since this type of access typically occurs for users without any past interactions with the web portal. Thus we develop two distinct recommending functionalities.

4. RECOMMENDATION OF ACCOMMODATIONS FOR A USER

First broad recommendation approach is known as *collaborative filtering* and it relies only on past user-item interactions. Based on these past actions it then finds similar users to other users. This approach usually produces better results but suffers from the *cold start* problem. This means that for a new user without past interactions (no history), the system will not be able to produce any meaningful recommendations. An important advantage is that the system can obtain the required data by observing and recording user history [2, 6].

To recommend accommodations to a given user, we used latent factor models approach, which is a method within collaborative filtering group.

Latent factor models try to explain the past ratings by characterizing users and items on hidden variables called factors, which are inferred from these past ratings patterns. The factors measure dimensions that are not obvious or easily explained. For users, each factor measures how much the user likes the item which scored high or low on the corresponding factor [2].

This method requires explicit preference values which we defined through the following mappings:

- view an accommodation ⇒ value 2.0,
- open inquiry window ⇒ value 4.0,
- send an inquiry ⇒ value 5.0,
- close inquiry window without sending ⇒ value 4.0.

The last action is important as an anomaly can happen where a user can access the inquiry window directly from a link and can therefore bypass other preceding actions.

In the case where a user does more than one action on the same accommodation, the highest preference value is kept.

4.1 Matrix factorization model

Latent factor models are most commonly created using matrix factorization. First a ratings matrix *R* of dimensions *m x n* is created from collected data. This means that the rows of matrix R represent *m* users and the columns represent *n* items. A specific value R_{ij} represents the rating given by user *i* to item *j*. Matrix *R* is then factorized into smaller matrices *U* and *V*. Matrix U represents *users* and is of dimensions *m x rank*, while matrix *V* represents *items* and is of dimensions *rank x n*. *Rank* is the parameter that tells us how many latent factors we want to use to describe a *user* or an *item* and is always much smaller than the dimensions of the original matrix which is very large (*rank* << *m, n*) [2].

Some elements of matrix *R* are not defined, meaning they cannot be interpreted as 0. This means that decomposition methods such as SVD (singular value decomposition) cannot be used.

4.2 Alternating Least Squares algorithm

Alternating Least Squares algorithm is used to modify latent factors and learn the model. It first initializes matrix *V* with small random values. When *V* is fixed, it iterates through matrix *U* and modifies the latent factors to best correspond to known ratings by minimizing the error. When this is done, it fixes *U* and does the same for *V*. It alternates doing this, solving the least squares problem defined by Equation (1) [2].

$$\min_{q,p} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\left|\left|q_i\right|\right|^2 + ||p_u||^2) \qquad \textbf{(1)}$$

In Equation 1, *K* is the set of *(u,i)* pairs for which rating r_{ui} is known. $(r_{ui} - q_i^T p_u)$ represents the error between known and predicted rating where q_i^T is the vector of latent factors for an item and p_u is the vector of latent factors for a user. $\lambda(\left|\left|q_i\right|\right|^2 + ||p_u||^2)$ represents the normalization term where λ is the normalization parameter which ensures that there is no data overfitting.

This algorithm is very suitable to be executed in parallel since many vectors of latent factors of *U* or *V* can be computed at the same time, considering one of the matrices is fixed. Potentially this allows the computation of all vectors of a single matrix in parallel.

The result of the algorithm is a model of latent factors which predicts preference values of any known user for any item.

5. SIMILAR ITEM RECOMMENDATION

Content-based filtering is the second broad recommendation approach and it focuses on creating a profile of each user or item to describe its nature. This profile usually contains the properties of an item. The system then finds similar items based on these profiles. This approach relies on obtaining data about items from an outside source, meaning it cannot obtain this data on its own [2, 6].

To recommend similar accommodations, an approach based on content-based filtering and the properties of accommodations was used. Based on our data we chose two similarity measures: Jaccard coefficient and Euclidian similarity.

5.1 Jaccard coefficient

Each accommodation has its corresponding attributes (e.g. pool, internet, bathroom, solar cells etc.). Each of these attributes is represented by an integer value and is either present or not. This was presented using a binary vector, where the index corresponds to the integer value of the attribute. Value 1 denotes presence of this attribute, while value 0 denotes its absence.

Since this data is binary, Jaccard coefficient is most suitable to measure attribute similarity between accommodations. It is defined by Equation (2), where $A \cap B$ represents the common attributes and $A \cup B$ represents the union of attributes of both accommodations:

$$Jaccard(A,B) = \frac{A \cap B}{A \cup B} \qquad \textbf{(2)}$$

5.2 Euclidian similarity

Each accommodation is also defined by its geographic position, consisting of longitude and latitude. Euclidian distance was used to first measure the distance between accommodations as shown by Equation (3).

$$Euclidian\_dist(a,b) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad \textbf{(3)}$$

A transformation of the distance into a similarity measure was then done by using Equation (4).

$$Euclidian\_sim = \frac{1}{1 + Euclidian\_dist(a,b)} \qquad \textbf{(4)}$$

Finally both similarity scores were merged into a unified similarity score. This was done by using Equation (5), where w_j and w_e correspond to weight parameters given to each similarity score:

$$Sim = \frac{w_j \cdot Jaccard(A,B) + w_e \cdot Euclidian\_sim}{w_j + w_e} \qquad \textbf{(5)}$$

6. EVALUATION

The system was evaluated using *Precision@k,* which is a standard measurement in information retrieval systems. Precision@k is defined by equation 6 [4].

$$Precision@k = \frac{Nr.of\ relevant\ among\ k\ recommended}{k}$$

In systems such as search engines and recommenders, the first *k* results are most important for their performance. It is highly important to show the *relevant* items among the first *k*. *Relevant* in the case of evaluation means the accommodations with which a user actually had an interaction in the past.

Recommender system was evaluated using *k-fold* cross validation, where the data is randomly split into *k* sets. The learning set always contains 80% of the data while the testing set contains 20%.

After the data was split, we chose evaluation parameters based on our problem. The following evaluation parameters were used:

- kFold = 5 – Chosen as a standard value.
- threshold = 2.0 – The threshold tells us which items are considered *relevant*. Threshold 2.0 means that any viewed or more strongly preferred is considered relevant.
- k = 3, 10 – Chosen as the user sees 3 accommodations on the site without scrolling and sees 10 accommodations on the first page of results.

The evaluation was executed for different parameters of algorithm:

- λ = 0.01 – Standard value which ensures that latent factor values do not overfit the learning data.
- numIterations = 5, 10 – Number of iterations during which latent factors are being modified.
- Rank = 5, 10, 20 – Number of latent factors which are used to describe user-item interactions.

Results are shown in Table 1.

rank	numIterations	Precision@3	Precision@10
5	5	1,74%	1,25%
5	10	5,38%	2,68%
10	5	8,90%	4,05%
10	10	17,66%	6,29%
20	5	**18,79%**	6,69%
20	10	16,69%	6,16%
Random recommender		0,97%	0,92%

Table 1: The evaluation results for different parameters of algorithm.

The results compare Precision@k of the developed recommender system with random recommender. The best result 19% was very superior to the 1% of random recommendation.

Better results are shown with low *k*. This is due to the fact that the average user only has three actions. This means that when the system recommends these items, it usually does so early (among the first three). Subsequently, by increasing the number of recommended items, the precision decreases, since there are not any relevant items left to recommend.

This measure proves to be pessimistic as Precision@k = 100% cannot be achieved when *k* is greater than the average number of actions per user. In case we had exactly one relevant item per user in the testing set, this measure could be normalized by dividing the scores with the maximum possible score that could be obtained. Example is shown by Equation 6 [5].

$$Precision@3_{norm} = \frac{18.79\%}{\frac{1}{3}} = 56.37\%$$
$$Precision@10_{norm} = \frac{6.69\%}{\frac{1}{10}} = 66.9\% \quad \textbf{(6)}$$

Comparison of our system with random recommender is shown on Figure 2.

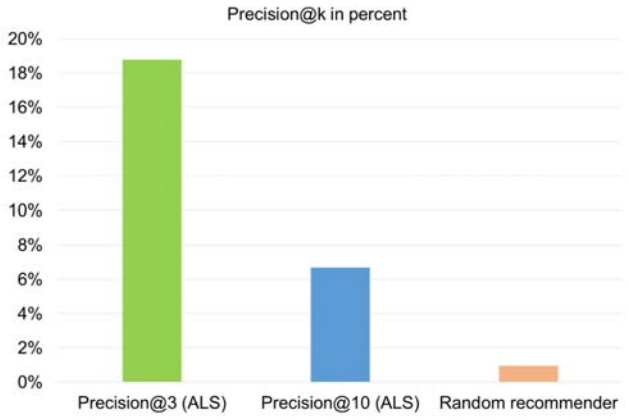

Figure 2: Precision@k comparison of the developed system with random recommender.

7. CONCLUSION

We developed a recommender system as a service for eco-friendly accommodations booking site. It was deployed in a cloud by using cloud-based machine learning server Prediction.IO. Data was collected in real time in user-action-item format by leveraging Prediction.IO built-in Event Server API. It offers recommendations for a user based on his past actions by building a latent factor model using matrix factorization with Alternating Least Squares learning algorithm. It also offers similar accommodations based on accommodation properties by computing Jaccard and Euclidian similarities.

The developed system was evaluated using the Precision@k performance measure and compared with a random recommender. It achieved 19% Precision@k which is far superior to 1% achieved by the random recommender.

MLaaS platform has shown to be an efficient tool for developing real-world machine learning applications, with a gentle learning curve. Due to good evaluation results we expect improved business results and improved user experience.

At the time of writing, a graphical representation of recommended accommodations is being implemented on the web portal. We further plan to use an algorithm developed for implicit datasets and to use other attributes about accommodations, especially price groups, since these have high influence on potential customers.

8. REFERENCES

[1] Bennett, J. and Lanning S. "The Netflix prize." *Proceedings of KDD cup and workshop*. 2007.

[2] Koren, Y., Bell R. and Volinsky C. "Matrix factorization techniques for recommender systems." *Computer* 8: 30-37, 2009.

[3] https://docs.prediction.io/. Accessed 20th September 2015.

[4] C. D. Manning, P. Raghavan, and H. Schutze. "Introduction to Information Retrieval." Cambridge University Press, New York, NY, USA, 2008.

[5] Slapničar, G. "Recommending accommodations using machine learning provider in a cloud." EngD thesis, University of Ljubljana, 2015.

[6] Adomavicius, G. and Tuzhilin, A. "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions." in Knowledge and Data Engineering, IEEE Transactions on, vol.17, no.6, pp.734-749, 2005