Parallel Taxonomy Discovery

Shalin Shah {shalin.shah@target.com}
Ramasubbu Venkatesh {ramasubbu.venkatesh@target.com}
Jinghe Zhang {jinghe.zhang@target.com}
Venkataramana Kini {venkataramana.kini@target.com}
Sayon Majumdar {sayon.majumdar@target.com}
Srikanth Ryali {srikanthryali7@gmail.com}

AI Sciences
Target Corporation
Sunnyvale, CA 94086, USA

February 7, 2020

Abstract

Recommender systems aim to personalize the shopping experience of a user by suggesting related products, or products that are found to be in the general interests of the user. The information available for users and products is heterogenous, and many systems use one or some of the information. The information available include the user’s interactions history with the products and categories, textual information of the products, a hierarchical classification of the products into a taxonomy, user interests based on a questionnaire, the demographics of a user, inferred interests based on product reviews given by a user, interests based on the physical location of a user and so on. Taxonomy discovery for personalized recommendation is work published in 2014 which uses the first three information sources – the user’s interaction history, textual information of the products and optionally, an existing taxonomy of the products. In this paper, we describe a parallel implementation of this approach on Apache Spark and discuss the modifications to the algorithm in order to scale it to several hundreds of thousands of users with a large inventory of products at Target corporation. We run experiments on a sample of users and provide results including some sample recommendations generated by our parallel algorithm.

1 Introduction

A recommender system that uses several sources of information to generate items that are in the interests of a user might be more successful than using

a single source of information. Recommender systems can use the user’s interaction history in a matrix factorization approach for matrix completion. A recommender system that generates item similarities could use contextual information about items, or use textual information to generate items similar to the item in consideration [1]. Embeddings could be generated for products using graph-based approaches [2][3] or using word2vec [4]. The drawback of these approaches is that in a graph-based recommender system, it is difficult to scale them to millions of users and items as it would make the parameter space of a graph or a neural network too large in a shared memory system, especially if the dimensionality is large. Taxonomy discovery for personalized recommendation [5] uses the user purchases and item descriptions to jointly learn the latent factors of users and products and a taxonomy over the products, and this work describes the hurdles and solutions to scaling this algorithm to a potentially millions of users and items.

2 Taxonomy Discovery for Personalized Recommendation

We first describe the approach in [5].

A latent factor is a numerical vector representation of a user or a product. The advantage of learning latent factors is that vector operations can be used on them, like inner products and distances. An inner product between a latent factor of a user and a product produces a real number that represents the affinity of the user towards that product. Sorting the affinities can give a ranked list of products that the user has interests in.

A path is a list of nodes from the root to the item in a taxonomy. A path has a probability associated with it based on the latent factors and a language model. A path looks like the following:

Clothing $\rightarrow$ Kid’s clothing $\rightarrow$ Kid’s Jackets
 $\rightarrow$ Hooded Kid’s Jackets $\rightarrow$ (Item) Nike Blue Hooded Kid’s Jacket

The taxonomy discovery paper [5] alternates between latent factor learning and path sampling. Each iteration, it first learns the latent factors and then, samples the path of the product using a Gaussian hierarchy. For details, we refer the reader to the paper [5].

Each node is modeled as a normal over its parent. And each item is also modeled as a normal over its parent node (i.e. the leaf node). So, items under the same leaf node share a common mean which means that less frequently bought items i.e. tail items have an advantage as they share some information with other items under the same leaf node (and with the entire path).

The probability of preferring item i against item j is then:

$$P(i > j|u) = \frac{\exp(x_{ui})}{\exp(x_{ui}) + \exp(x_{uj})}$$

Where, $x_{ui} = \langle v_u, v_i \rangle + q_i$

Where v_u is the user's latent factor and v_i and v_j are the item latent factors, where the user prefers item i to item j [6][7].

Each node and item share information with other nodes and items under the same parent in the hierarchy:

$$v_i \sim N(v_{pi}, \sigma^2 I)$$

Where v_{pi} is the parent of item i in the taxonomy.

q_i is a bias term which is modeled as a zero mean normal.

$$q_i = N(0, \tau^2)$$

The cost function to learn the latent factors is as follows:

$$L_{uij} = \log \sigma(x_{ui} - x_{uj}) - \sum_{k=1}^{L_i} \frac{\|v_{ik} - v_{i_{k+1}}\|^2}{2\sigma^2} - \sum_{k=1}^{L_j} \frac{\|v_{jk} - v_{j_{k+1}}\|^2}{2\sigma^2} - \frac{q_i}{\tau^2} - \frac{q_j}{\tau^2}$$

The first term is the difference between the dot products of a bought and a not bought item. The second and fourth terms enforce that child nodes (including items) are as close as possible to the parent nodes (in the entire path of $0, 1, 2, \dots, L_i$).

In path sampling, the language model of the item descriptions, the latent factors of the nodes and the products, and a nested Chinese restaurant process are used. The item descriptions are used in a language model to compute the probability of choosing the child node of a parent (starting from the root). In addition to that, the latent factors of the nodes are used to generate a probability of choosing a particular child node:

$$v_i \sim N(v_{pi}, \sigma^2 I)$$

Where the mean of the normal is the factor of the parent node. A Chinese restaurant process (CRP) is used to decide on whether a new node should be created in the taxonomy. The CRP prior prefers nodes that have more children as compared to the other nodes:

$$P(y \rightarrow z) = \begin{cases} \frac{n_z}{n_y + \alpha} & \text{for an existing node} \\ \frac{\alpha}{n_y + \alpha} & \text{for a new node} \end{cases}$$

3 Parallel Taxonomy Discovery

We implemented the taxonomy discovery [5] algorithm on Apache Spark. We had to make some modifications and approximations to the algorithm to scale it to millions of users and items. We describe the modifications as under.

3.1 Path Sampling

We use mini-batch Gibbs sampling. In the path sampling stage of an algorithm, the latent factors are not updated (the path sampling and latent factor updates are alternated). So, the only thing that changes while path sampling is the language model when a product samples a different path as compared to its original path. This happens mostly in the leaf nodes of the taxonomy. If a product in a very different category changes its path, it is most likely not changing any of the textual information that is useful when other items in a different leaf category condition on the changes. So, we group the products by the level 4 leaf category (Hooded Kid’s Jackets) and do a mini-batch Gibbs sampling.

Also, we have an existing taxonomy that is quite accurate. So, we don’t want the taxonomy to change excessively. We don’t want the new path to vary in a very significant way from a path that is known to be useful and accurate, unless the probabilities indicate. Exploration is a less important goal than correction. So, we use a Metropolis Hasting acceptance rule in addition to Gibbs sampling to accept the new path.

It is known that asynchronous Gibbs sampling without shared memory has difficulty with convergence. The approach in [8] says that, if an acceptance rule is used, there is better convergence. We use the Barker’s rule [9] to accept new paths. The Barker’s rule has the following acceptance probability:

$$P_{accept} = \frac{P_{new}}{P_{new} + P_{old}}$$

This ensures that since we already have an accurate taxonomy, vastly different and low probability paths are not accepted frequently and that the path sampling iterations do not need a very large burn in period.

3.2 Latent Factor Updates

In our implementation, each user is associated with a randomly chosen mini-batch (of 1000 users) and a mini-batch gradient descent is performed. Each iteration, the mini-batch of a user is changed randomly. So, each iteration, a

user might be in a different mini-batch. We found this to improve convergence and the results.

Also, because of the exponential function in the partial derivatives we faced the problem of exploding gradients which caused the latent factors to become very large and biased towards very popular products. The result was that the diversity in the recommendations was unacceptably low. We had to convert each latent factor to a unit vector each iteration of a mini-batch. Also, after an iteration has completed and the latent factors updated, we convert each latent factor into a unit vector [10].

Moreover, we reduce the learning rate after each iteration. We start with a learning rate of 0.1 and then reduce the learning rate using an exponential decay. This was shown to improve convergence and the hit rate after each iteration. Also, we do not use early stopping. Based on our experience, we found it useful to adjust the learning rate such that the results don't deteriorate as iterations progress, and run the algorithm for the full specified number of iterations. We use 10 iterations.

4 Results

We ran our algorithm of a parallel implementation of [5] on a sample of 250,000 users and 40,000 items of the furniture category. We use a dimensionality of 200. We filter out users who don't have more than 10 interactions. There are several other filtering techniques that [5] uses, like remove users who buy only the most frequent items. We do not use any of these techniques, and we found that the diversity of the recommendations is still quite good.

The hit rate @10 of a recommender system is the rate with which we find an item that a user interacted with in the top 10 recommendations, in the test set. We generate recommendations for a user using all 40,000 items and then calculate the hit rate on a randomly held out test set of items which a user interacted with. We do not differentiate between item views and purchases. While generating recommendations, we remove products which the user has already interacted with.

In our sample of 250k users and 40k items, we found the hit rate@10 to be 3.1%, hit rate@50 to be 8% and hit rate@100 to be 14% (out of a possible 40k items). We find that the quality of recommendations, as seen on a random sample of users is quite good. Please see the examples below in tables 1,2,3,4.

Table 1: Example Recommendations in the furniture category

User's Interactions
Recommended Items

Figure 1: Recommendations (Furniture)

Table 2: Example Recommendations in the furniture category

User's Interactions
Recommended Items

Figure 2: Recommendations (Furniture)

5 Conclusion

In this paper, we presented a parallel implementation of [5] which is a really interesting attempt at using several heterogeneous sources for generating personalized recommendations. Our approach can be easily implemented on an Apache Spark cluster. For latent factor learning, we use mini-batch gradient descent and for path sampling, we use mini-batch Gibbs's sampling with an acceptance probability of new paths. We achieved reasonably good hit rates, considering the sparsity of interactions. The example recommendations in the furniture category show that the algorithm is capable of learning from the interactions of a user and generate useful recommendations.

References

- [1] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
- [2] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele

Table 3: Example Recommendations in the furniture category

User's Interactions
Recommended Items

Figure 3: Recommendations (Furniture)

Table 4: Example Recommendations in the furniture category

User's Interactions
Recommended Items

Figure 4: Recommendations (Furniture)

Osella, and Enrico Ferro. Knowledge graph embeddings with node2vec for item recommendation. In *European Semantic Web Conference*, pages 117–120. Springer, 2018.

- [3] Shalin Shah and Venkataramana Kini. Hebbian graph embeddings. *arXiv preprint arXiv:1908.08037*, 2019.
- [4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [5] Yuchen Zhang, Amr Ahmed, Vanja Josifovski, and Alexander Smola. Taxonomy discovery for personalized recommendation. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 243–252, 2014.
- [6] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

- [7] Alfredo Láinez Rodrigo and Luke de Oliveira. Distributed bayesian personalized ranking in spark.
- [8] Alexander Terenin, Daniel Simpson, and David Draper. Asynchronous gibbs sampling. *arXiv preprint arXiv:1509.08999*, 2015.
- [9] Av A Barker. Monte carlo calculations of the radial distribution functions for a proton? electron plasma. *Australian Journal of Physics*, 18(2):119–134, 1965.
- [10] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016.

6 Appendix

Detailed balance is a desirable condition that ensures that the stationary distribution of a Markov chain is the distribution of interest (if you interpret the path sampling procedure as a way of learning a distribution over taxonomies). Barker’s rule that was mentioned earlier in the paper has detailed balance and here is a simple proof that if the proposal density is normal, then it has detailed balance in a Metropolis Hastings sampler.

Proof:

The detailed balance condition states the following:

$$P_{old}P(new \rightarrow old) = P_{new}P(old \rightarrow new)$$

$$\text{Let } A = P_{old}P(new \rightarrow old)$$

$$\text{And } B = P_{new}P(old \rightarrow new)$$

$$\text{So } A = P_{old} * g(new|old) * P_{accept_{new}} \text{ and } B = P_{new} * g(old|new) * P_{accept_{old}}$$

Where g is the proposal density. Now if g is symmetric,

$$g(new|old) = g(old|new) = N(old - new|0, \sigma^2 I)$$

$$\text{Then, } A = B$$

$$\text{Thus, } P_{old}P(new \rightarrow old) = P_{new}P(old \rightarrow new)$$

■