Fast and Accurate Maximum Inner Product Recommendations on Map-Reduce

Rob Hall
Etsy Inc
55 Washington St, Brooklyn NY
rhall@etsy.com

Josh Attenberg
Etsy Inc
55 Washington St, Brooklyn NY
jattenberg@etsy.com

ABSTRACT

Personalization has become a predominant theme in online advertising; the internet allows advertisers to target only those users with the greatest chances of engagement, maximizing the probability of success and user happiness. However, a naïve approach to matching users with their most suitable content scales proportionally to the product of the cardinalities of the user and content sets. For advertisers with large portfolios, this quickly becomes intractable. In this work, we address this more general *top-k personalization* problem, giving a scalable method to produce recommendations based on personalization models where the affinity between a user and an item is captured by an inner product (i.e., most matrix factorization models). We first transform the problem into finding the k-nearest neighbors among the items for each user, then approximate the solution via a method which is particularly suited for use on a map-reduce cluster. We empirically show that our method is between 1 and 2 orders of magnitude faster than previous work, while maintaining excellent approximation quality. Additionally, we provide an open-source implementation of our proposed method, this implementation is used in production at Etsy for a number of large-scale personalization systems, and is the same code as used in the experiments below.

Categories and Subject Descriptors

I.5.4 [**Computing Methodologies**]: Pattern Recognition

Keywords

Recommender systems; maximum inner product search; map-reduce.

1. INTRODUCTION

Since the introduction of the internet, personalization has transformed the way that people consume content. Recommender systems have become ubiquitous on e-commerce sites like Amazon [13, 19], and have become almost a necessity for media curation, as exemplified but he success of sites like Neflix [3]. Personalization has also become a powerful tool of the advertiser [2, 20, 22], where personalized models attempt to improve ad relevance, thereby increasing the chances of engaging with the user.

Personalization technology has benefitted from the inputs of a vibrant research community motivated by clear business-driven use cases, an abundance of data, and data mining competitions like the million dollar Netflix prize [3]. This diverse research, along with the varying circumstances for applying personalization technology, has lead to a plethora of mathematical personalization techniques. Item-based, user-based, and collaborative filtering recommender systems have all enjoyed success both in practice and in the results reported in research literature. We encourage the reader to consult [18] for a detailed treatment on personalization and recommendation technology. In this paper, we focus on a especially successful family of techniques for performing collaborative filtering, techniques based on matrix factorization [10]. In particular, we deal with so-called *top-k personalization*, where we are interested with the k items we believe to be most relevant to a particular user. Typically, we seek to optimize some precision-like metric on the set of k candidates generated for a user.

Specifically, we deal with the problem of generating recommendations (e.g., recommending products to users) from models where the affinity between an item and a user is captured by an inner product between the vector representations for each. Examples of such models are given in e.g., [3, 4, 6, 7, 10, 11, 15, 16]. The vector representations may correspond to latent features of the users and items, or to "topics" among the items which the users are interested in, depending on how the model is fit.

Although the theory for the construction and fitting of these kind of models is now well established, little attention has been paid to the problem of actually finding the top k recommendations for a user from the model. Selecting the optimal content for a user requires finding those items with the greatest value of an affinity score to that user. To do this exactly requires a linear scan of all the available items, which is prohibitive for large scale systems, where there can be hundreds of millions of items [6]. Whats more, this procedure is required for each user, who can number in the billions [3]. Clearly, a brute force solution to the top-k personalization problem is intractable.

The first work to deal with the problem of efficiently generating the top k recommendations from a matrix factoriza-

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2742000.

tion recommender system is [17]. The authors describe a data structure conceptually similar to a ball tree, but which can be used to find points with maximum inner products to a query point, rather than the nearest neighbors. This method appears to suffer from the same curse of dimensionality which befalls the ball tree in high dimensions [8], and, for high dimensional models, does not provide a sufficient speedup over the brute force approach. However [17] may be useful when the dimension of the personalization models is small.

More recently [24] dealt with the problem via a reduction to k-nearest neighbors, which they then approximated by using Locality Sensitive Hashing [1, 21]. Our method is conceptually very similar to [24]. We give a slightly simpler reduction to k-nearest neighbors, and then approximate these on map-reduce using a method similar to [14]. We show that the approach works well for various kinds of recommender systems with different properties.

The method described below has been well tested in a production setting, being used to build personalized datasets for a number of recommender systems at Etsy[1], an online marketplace of handmade and vintage goods with tens of millions of users and items. Additionally, an open-source implementation of the method we describe is available as part of our machine learning package called "Conjecture" [2], implemented in the scala-based hadoop DSL, scalding.

2. SETUP

Suppose we have a model of n users and m items, in which the affinity between user i and item j is captured by:

$$\text{affinity}(i, j) = u_i' v_j, \quad u_i, v_j \in \mathbb{R}^d,$$

where $u_i'v_j$, denotes the inner product between u_i and v_j. The vectors describing the users $u_1, \ldots u_n$ and items $v_1, \ldots, v_m$ may have different properties and interpretations depending on the nature of the fitted model. For example they can be dense real vectors in the case of linear matrix factorization models [7, 15, 16, 23] sparse non-negative vectors in the case of Poisson non-negative matrix factorization [11], or dense non-negative vectors in the case of LDA or Bayesian Poisson models [4, 6]. In this setting, the goal of *top-k* personalization is to find the set of k items for each user which maximizes the sum of affinity scores:

$$R(i) = \arg \max_{\pi_1 \neq \pi_2 \neq \ldots \neq \pi_k} \sum_{j=1}^{k} \text{affinity}(i, \pi_j). \tag{1}$$

Here R returns the set of indices $\pi_1, \ldots, \pi_k$ corresponding to the items with the highest affinity scores. Note that the naive approach to computing $R(i)$ for all users takes time proportional to $n \cdot m$, since we must consider every user and item pair. Approaches which build data structures to prune the search space are typically not helpful when the goal is to provide an exact solution, as they run into the "curse of dimensionality" [8]. Similar to [24], we propose to approximate the solution to (1) in two stages: first we construct a transformation of the user and item vectors so that the euclidean distances to the items for a user are monotonically decreasing as the inner product between the original vectors increases. Then we use an approximate k-nearest neighbor method to find the nearest items to each user, which correspond to those items with the highest affinity score.

3. REDUCTION TO KNN

We can reduce the computation of $R(i)$ to a k-nearest neighbor computation. The basic idea is to map the users and items to an augmented representation in $\mathbb{R}^{d+1}$, in which all the vectors describing the items have equal norms, and where the inner product is preserved. In the below equations the norm $\| \cdot \|$ is the ℓ_2 norm wherever it appears.

Let

$$c = \max_j \|v_j\| \tag{2}$$

be the maximum norm among the item vectors. We define the vectors:

$$\tilde{u}_i = (u_{i,1} \cdot c, \ldots, u_{i,d} \cdot c, 0) \tag{3}$$

which is just the scaled original vector with a zero element added to the end, and

$$\tilde{v}_j = (\frac{v_{j,1}}{c}, \ldots, \frac{v_{j,d}}{c}, \sqrt{1 - \frac{\|v_j\|^2}{c^2}}). \tag{4}$$

These augmented item vectors all have unit length:

$$\|\tilde{v}_j\|^2 = \frac{\|v_j\|^2}{c^2} + (1 - \frac{\|v_j\|^2}{c^2}) = 1, \tag{5}$$

what's more, in the augmented representation, the inner products between each user and item vector is preserved:

$$\tilde{u}_i'\tilde{v}_j = \frac{c}{c} u_i' v_j + 0 \cdot \sqrt{1 - \frac{\|v_j\|^2}{c^2}} = u_i' v_j, \tag{6}$$

Therefore we have the following result

Proposition 3.1.

$$\arg \max_j u_i' v_j = \arg \min_j \|\tilde{u}_i - \tilde{v}_j\|^2, \tag{7}$$

Proof. Expanding the square norm, and substituting in (5) then (6) we have

$$\|\tilde{u}_i - \tilde{v}_j\|^2 = \|\tilde{u}_i\|^2 + 1 - 2\tilde{u}_i'\tilde{v}_j = (c^2\|u_i\|^2 + 1) - 2u_i'v_j,$$

the norm $\|u_i\|$ is a constant in the minimization. □

This leads to:

$$R(i) = \arg \min_{\pi_1 \neq \pi_2 \neq \ldots \neq \pi_k} \sum_{j=1}^{k} \|\tilde{u}_i - \tilde{v}_j\|^2,$$

thus the problem of finding recommendations is reduced to the problem of finding nearest neighbors in the augmented representation.

Note that the role of c from (2) is to ensure that all the item vectors are mapped to representations with unit length. An upper bound on the true maximum norm among the items would also work, and could potentially save having to perform a scan of the items to find this value, however the computation of c on a map-reduce cluster, the setting we primarily consider, is trivial. Also the role of c in (3) is not important, if we replace it with some other value d in that definition then we obtain:

$$\|\tilde{u}_i - \tilde{v}_j\|^2 = (d^2\|u_i\|^2 + 1) - 2\frac{d}{c} u_i' v_j,$$

[1]`www.etsy.com`
[2]`https://github.com/etsy/Conjecture`

which can also be used to compute the recommendations, since we have just scaled the affinity score by a constant.

We also note that a consequence of (6) is the possibility to perform a kind of constrained matrix factorization, in which the vectors describing the items are all constrained to have unit norm, and the quality of the fit is no worse than that of the unconstrained model in one fewer dimensions. This applies irrespective of the type of model being used. To see this, note that we can just fit a $d-1$-dimensional unconstrained model, then make it into a d-dimensional model which has the same goodness of fit, and obeys the constraint.

Compared to [24] this method requires fewer additional coordinates, although this is not a big concern since both methods add relatively few extra dimensions when compared to the size of the original vectors (which are typically on the order of hundreds of dimensions). We note that the method of [24] also requires the computation of (2). Finally another small advantage of our proposed method is that it preserves the inner products exactly, whereas the method of [24] is an approximation, although one which can be made arbitrarily tight at the cost of adding additional dimensions. Our construction is conceptually simple, in essence the extra dimension we add just pushes the items with small norms away from all the users, so that items with large norms (i.e., those that have a chance to form high inner products to the users) are more likely to be the nearest neighbors of the users.

We implement the transformation described in (3) and (4) as a pre-processing step, which requires two map-reduce stages: one to compute c from (2) and then a second one to produce the vectors. These take time proportional to the sum of the number of items and users and are likely to be trivial for even very large datasets.

4. APPROXIMATING THE KNN ON MAP-REDUCE

Having transformed the problem of computing recommendations to one of finding k-nearest neighbors, we now aim to approximate the solution to the latter, which will yield an approximate solution to (1). Like [24] we also use the approach of Locality Sensitive Hashing [1, 21], however we use a different method to hash the points, which adapts to the distribution of the points in space.

We note that a consequence of both the transformation given in Section 3 and [24] is that the items and users become separated in space, that is, the distance between users and items are much larger than those between pairs of points of the same type. To see why, observe that both methods force the items and users apart from each other along the new dimensions in the space. The result of this is that standard techniques for locality sensitive hashing can break down. There are two possible problems: first, a hash bucket contains a great quantity of items, meaning that the technique reduces to a brute force approach (see [24] Figure 3, which shows the number of dot products needed is between a quarter to half as much as the brute force method – which is intractable if we want to provide recommendations to more than a handful of users), and second, collisions between items and users are unlikely, so we have to use many different hash tables in order to have a good quality approximation. The problem is exacerbated since increasing the granularity of the hashing function to deal with problems of overly full bins will lead to an even smaller probability of collisions between the users and items. The theory developed in [24] rests on the assumption that the top-k recommendations for a user will have high score (bounded below by S_0), whereas we observe that many users' top recommendations may have low scores. This could be a consequence of the model uncertainty for the users who have not provided much implicit feedback.

We give a method which produces approximate recommendations for all the users irrespective of the scores of their true top-k recommendations. The hashing strategy utilized herein results in splitting the items into hash buckets of approximately equal size. The idea is to divide the space into buckets corresponding to the cells of a voronoi diagram generated from a subset of the item vectors. To begin we select $s \approx m^{1/2}$ items given by indices ρ_1, ρ_s uniformly at random. We then define the hash function:

$$h(x) = \arg\min_{i \in \{1,\ldots,s\}} \|x - \tilde{v}_{\rho_i}\|, \quad x \in \mathbb{R}^{d+1}. \tag{8}$$

which returns the index to the closest item from among the sampled set of items. This leads to buckets with approximately equal numbers of items in them. Buckets which are too large can be further subdivided by selecting more points from that bucket to be the centers of new cells.

We then map users and items to the same set of hash buckets via $h(\cdot)$. It is intuitive that these buckets are likely to contain the closest item vectors to each user vector, since the buckets correspond to sets of items which are close to each other in space, and where one of these items (the chosen center) is the closest center to the user. Some initial exploration of the theoretical validity of this approach is given by [9], however strong theoretical guarantees are not yet available to our knowledge, and are an area for future work.

We can improve the quality of the approximation by considering e.g., the items in the p closest buckets to a user, rather than just the closest one. We can implement this by hashing each user vector into multiple hash buckets by using:

$$h^p(x) = \arg\min_{i_1 \neq i_2 \neq \ldots \neq i_p \in \{1,\ldots,s\}} \sum_{j=1}^{p} \|x - \tilde{v}_{\rho_{i_j}}\|, \tag{9}$$

where p is the number of hash codes produced for each user. These are just the indices of the p closest items from the subsample.

To implement this approach on map-reduce we first stage the set of item vectors $\tilde{v}_{\rho_1}, \ldots, \tilde{v}_{\rho_s}$ to each mapper, where we take as input the item and user vectors in the representation given in (3) and (4). These are hashed using the above method, producing one hash code for each item and p hash codes for each user. The vectors are then directed to reducers depending on these hash codes, so that all the items and users within each bucket are available in memory in the reduce nodes. There we can just perform a brute force search, since the amount of data is small.

We may, however, be in a situation where e.g., all the users map to a single hash bucket, or where the distribution of users to hash buckets is highly skewed. In such a case if we implement the method described above then there is the possibility for a few reducers to become overwhelmed with work, whereas others do nothing. We can get around this by implementing an analog to a fragment-replicate join on map reduce (see e.g., [25]). To do this we count the number of users who are in each hash bucket, when we have more than some pre-defined limit (e.g., 20,000 users) then we split the users randomly into multiple "fragments" so that each fragment obeys the limit on the number of users. We then take the set of items associated with the hash bucket, and replicate it once for each fragment of users. These fragments of users and replicates of items are then distributed across different reduce tasks, so that we still compute the brute force solution for each user, but each mapper only has to deal with a tractable amount of users. This means that we avoid the problem of long-running reduce stages.

In summary the approach is:

- Compute (2) and create transformed user (3) and item vectors (4).
- Select $s \approx \sqrt{m}$ item vectors at random, and write those to a file which can be put on all the mappers.
- Count how many items are in each bucket under (8), if necessary add more items to the set of random ones.
- Count how many users are in each bucket under (9), if necessary replicate the item buckets and fragment the user buckets.
- Stage pairs of correpsonding user and item buckets to reducers, where a brute force search for the KNN is done.
- Finally combine the results from the different buckets for all users.

Therefore overall, our approach involves performing a number of user-item comparisons which is proportional to $nm^{1/2}$ first we compare each user to one of the $m^{1/2}$ selected centers, then in each bin we have approximately $m^{1/2}$ items which we compare each user to. We also have to compute approximately $m^{3/2}$ item-item distances to construct the item bins (8). When the number of items is are large, this presents a substantial saving over the brute force method, which take time proportional to nm.

One thing to note is that the feasability of this approach depends on being able to store the s chosen item vectors $\tilde{v}_{\rho_1}, \dots, \tilde{v}_{\rho_s}$ in memory on each map task for the first stage. We experiment on a cluser where each task is allowed up to 3GB of ram. When the data are on the orders of hundreds of millions, then we have tens of thousands of vectors, and for vectors of moderate dimension (200) or higher diensional sparse vectors, these fit in memory. To run this approach on an even larger data set could be done by another application of the replicate-fragment strategy, namely we could fragment the set of cell centers, so mappers would find the closest center to each point from among a subset of the centers. A reduce stage would then combine these for each input point and find the center which is the closest overall.

The above approach is similar to that considered in [14]. There, a similar procedure to form hash buckets is used, with some additional work to actually compute the exact k-nearest neighbors. Using their method in combination with the reduction from Section 3 would in fact lead to an exact method of producing recommendations, although at the cost of performing more computation. However, since most personalization tasks can tolerate some approximation, we proceed with a simplified, computationally tractable version.

An open-source implementation of this approach is provided as part of our machine learning package "Conjecture" (`https://github.com/etsy/Conjecture`)[3]. Conjecture provides efficient implementations of a number of common machine learning and data science tasks using the "Scalding" scala-based Hadoop MapReduce DSL. Note that Scalding itself is actually a wrapper around another flow-oriented Hadoop DSL, "Cascading", which provides most of the tooling for planning and executing large computations where the underlying primitive is the map-reduce job.

5. EXPERIMENTAL ANALYSIS

We experiment with two datasets, both of which generated from the implicit feedback of users "favoriting" items on Etsy (see [6] for more details). The first dataset we consider is called "SVD" and is generated by using the stochastic-SVD method [5] on the binary matrix of users and items, where a cell is one if the user favorited the item and zero otherwise. We restrict to items and users with sufficient favorites to produce a high quality model. This dataset covers approximately 27 million items, and 13 million users, and the vectors are 200-dimensional. The second dataset is a non-negative matrix factorization of the same matrix. We implemented the Poisson NMF model of [11], constructing a model with 1000 dimensions. We optimized this model using a projected gradient approach [12]. The model is sparse, hence we can fit these vectors in memory despite the larger dimension.

We implemented our approach with various values of p (from 1 to 9) to trade off the approximation quality with the size of the computation. While we ran the method for all 13 million users, in order to do a comparison with the true, best fit recommendations, we only considered 1000 of the users chosen uniformly at random, since for this number we can brute force the true recommendations without undue computational effort. Likewise we implemented the technique of [24] to compare to using our datasets. We found that these methods do not scale well enough to be able to perform the computation of recommendations for all the users at once, therefore in order to make a comparison we just computed the approximate set of recommendations for same set of users for whom we generated the true recommendations. We implemented that approach with various numbers of hash tables (25, 50, 100, 150), and various numbers of hashes per table (10 and 15). Using fewer hash codes leads to the method becoming too computationally demanding, and using more leads to a decrease in the approximation quality since the hash buckets become too sparse.

As a measure for quality of the recommendations we use the ratio of the score of the top approximate recommendation to the top exact recommendation. When we consider the top-k recommendations then we use the ratio of the sums of scores. We average this across the users. The reason we use this measure instead of e.g., the fraction of the exact recommendations that were returned, is because a user may have many good candidates for recommendations to which the model assigns similar scores. It is often more important to retrieve items with high scores than to necessarily retrieve the exact top scoring items. Results are summarized in Figures 1 and 2. We show for our method "Voronoi" and "LSH" of [24], the quality of the recommenda-

[3]The class which implements the method is called FastKNN.

tions as the computational load is increased. To measure the computational complexity we count the item-user dot products that we compute in each hash bucket for the methods, and for our method we also add the user-item comparisons that happened in the computation of (9) and the item-item comparisons of (8). We report the fraction of this number of comparisons compared to the brute force approach, which requires comparing every item and user. We do not report the total running time of either method, since other jobs which run on our computation cluster would potentially skew the results.

We find that on both datasets our method requires between one and two orders of magnitude fewer user-item comparisons in order to achieve high quality recommendations. Although the LSH method can produce precise results, we note it would only be approximately ten times faster than brute force, whereas our method is a over a thousand times faster. In addition to performing fewer comparisons, the amount of intermediate data between the map-reduce stages is also greatly reduced under our method. The reason is that we are far more conservative with how we replicate the item vectors – doing so only as part of the fragment-replicate strategy to ensure a fast computation. On the other hand the LSH method essentially replicates each item once for each hash table we use, which is significantly more. Likewise the user vectors are similarly replicated once for each hash table under the LSH technique, and p times under our method which we have shown can be much smaller.

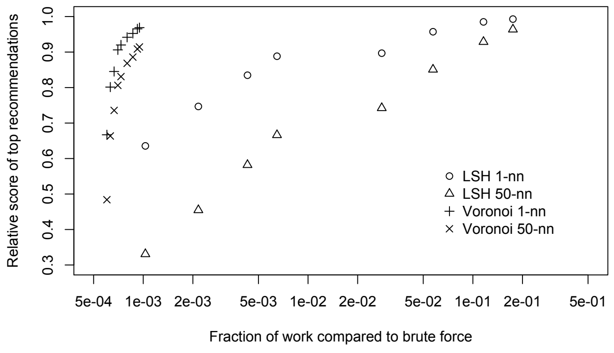

Figure 1: Recommendation quality of the top-1 and top-50 recommendation vs amount of computation required, for the SVD dataset. Note that the x-axis is on a log scale. The different points for each method correspond to different settings of the parameters. The proposed method outperforms prior work by 2 orders of magnitude.

It is not yet clear why the advantage of our approach is slightly diminished when operating on the Poisson dataset. It may be a consequence of the higher dimensionality of this data compared to the SVD model.

6. CONCLUSION

We presented a method for generating personalized content from any model which captures user-item affinities via

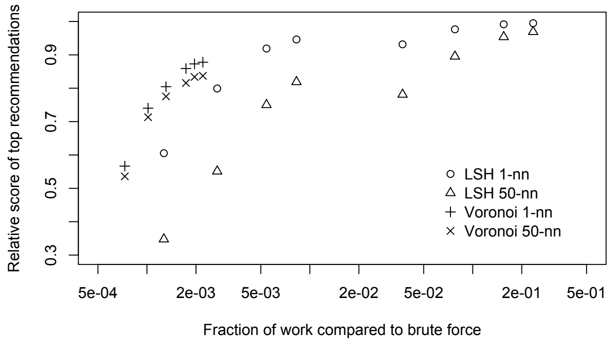

Figure 2: Recommendation quality of the top-1 and top-50 recommendation vs amount of computation required, for the Poisson dataset.

and inner product between a user vector and an item vector. We implemented our approach on map-reduce and showed that despite the increased cost of hashing compared to LSH, the overall computational burden of our method is much smaller, while still providing high quality recommendations.

While our approach outperforms prior work empirically on our datasets, it currently lacks the theoretical guarantees which come with the LSH method. However given the good performance of the method it is likely that some theoretical statement can be made about its approximation quality. For example, as we increase the number of buckets then we do better at retrieving the 1-nn (in the limit the method just reduces to brute force which is exact). The choice of $m^{1/2}$ buckets which we suggested here is just to balance the computational demand of the bucketing and the brute force search within each bucket. It may be that there exists a better choice for the number of buckets, which could depend on the characteristics of the data, its size and dimensionality etc.

It may also be possible to improve upon our bucketing method, by considering the bounding sphere of each bucket, and mapping users to the buckets for which they are closest to the surface of the bounding sphere, rather than the selected center. This would map users to the buckets where they have the possibility of finding a close item if it exists. We are actively researching the theory behind our model in addition to developing practical improvements in model efficiency. We hope to include this ongoing work in a future publication.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008.

[2] J. Attenberg, S. Pandey, and T. Suel. Modeling and predicting user behavior in sponsored search. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*,

KDD '09, pages 1067–1076, New York, NY, USA, 2009. ACM.

[3] J. Bennett and S. Lanning. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[4] P. Gopalan, J. M. Hofman, and D. M. Blei. Scalable recommendation with poisson factorization. *CoRR*, abs/1311.1704, 2013.

[5] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[6] D. J. Hu, R. Hall, and J. Attenberg. Style in the long tail: Discovering unique interests with latent variable models in large scale social e-commerce. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1640–1649, New York, NY, USA, 2014. ACM.

[7] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[8] P. Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press LLC, Boca Raton, FL, 2nd edition, April 2004.

[9] B. Kang and K. Jung. Robust and efficient locality sensitive hashing for nearest neighbor search in large data sets, 2012.

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

[11] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *In NIPS*, pages 556–562. MIT Press, 2000.

[12] C.-J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Comput.*, 19(10):2756–2779, Oct. 2007.

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

[14] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proc. VLDB Endow.*, 5(10):1016–1027, June 2012.

[15] D. Oard and J. Kim. Implicit feedback for recommender systems. In *in Proceedings of the AAAI Workshop on Recommender Systems*, pages 81–83, 1998.

[16] L. Peska and P. Vojtas. Negative implicit feedback in e-commerce recommender systems. In *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*, WIMS '13, pages 45:1–45:4, New York, NY, USA, 2013.

[17] P. Ram and A. G. Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 931–939, New York, NY, USA, 2012.

[18] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[19] J. B. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

[20] S. Schroed, A. Kesari, and L. Neumeyer. Personalized ad placement in web search. In *Proceedings of ADKDD'10*, ADKDD'10, 2010.

[21] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 2006.

[22] M. Shatnawi and N. Mohamed. Statistical techniques for online personalized advertising: A survey. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 680–687, New York, NY, USA, 2012. ACM.

[23] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. Climf: Learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 139–146, New York, NY, USA, 2012. ACM.

[24] A. Shrivastava and P. Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2321–2329. Curran Associates, Inc., 2014.

[25] J. W. Stamos and H. C. Young. A symmetric fragment and replicate algorithm for distributed joins. *IEEE Trans. Parallel Distrib. Syst.*, 4(12):1345–1354, 1993.