Nobody cares if you liked Star Wars: KNN graph construction on the cheap

Anne-Marie Kermarrec, Olivier Ruas, François Taïani

To cite this version:

Anne-Marie Kermarrec, Olivier Ruas, François Taïani. Nobody cares if you liked Star Wars: KNN graph construction on the cheap. Europar 2018, Aug 2018, Turin, Italy. pp.419-431, 10.1007/978-3-319-96983-1_30 . hal-01867230

HAL Id: hal-01867230
https://hal.inria.fr/hal-01867230

Submitted on 4 Sep 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Nobody cares if you liked Star Wars: KNN graph construction on the cheap.

Anne-Marie Kermarrec[1,2], Olivier Ruas[0000−0002−6862−9046][3], and François Taïani[3][0000−0002−9692−5678]

[1] Mediego, Rennes, France
[2] EPFL, Lausanne, Switzerland
[3] Univ Rennes, Inria, CNRS, IRISA, France
anne-marie.kermarrec@mediego.com, olivier.ruas@inria.fr,
francois.taiani@irisa.fr

Abstract. K-Nearest-Neighbors (KNN) graphs play a key role in a large range of applications. A KNN graph typically connects entities characterized by a set of features so that each entity becomes linked to its k most similar counterparts according to some similarity function. As datasets grow, KNN graphs are unfortunately becoming increasingly costly to construct, and the general approach, which consists in reducing the number of comparisons between entities, seems to have reached its full potential. In this paper we propose to overcome this limit with a simple yet powerful strategy that samples the set of features of each entity and only keeps the least popular features. We show that this strategy outperforms other more straightforward policies on a range of four representative datasets: for instance, keeping the 25 least popular items reduces computational time by up to 63%, while producing a KNN graph close to the ideal one.

1 Introduction

K-Nearest-Neighbors (KNN) graphs play a crucial role in a large number of applications, ranging from classification [22] to recommender systems [4,15,17]. In a KNN graph, every entity (or node) is linked to its k closest counterparts, based on a given similarity metric. Despite being one of the simplest model of machine learning, computing an exact KNN graph[4] is unfortunately a highly time consuming task. A simple brute force approach for instance has a quadratic complexity in the number of entities. For applications for which data freshness is more valuable than the exactness of the results, such as news recommenders, such computation time is prohibitive. To overcome these costs, most applications therefore compute an approximate KNN graph by using pre-indexing mechanisms [5,11] or by exploiting greedy incremental strategies [4,10] to reduce the number of similarity computations. However, it seems hard to lower even further that number.

[4] We focus here on the computation of the whole graph, which is different from the related but distinct problem of answering KNN queries.

In this paper we focus on an orthogonal approach, and leverage *sampling* as a preliminary pruning step to accelerate the time to compute similarities between two entities. Our proposal stems from the observation that many KNN graphs computations are performed on entities (users, documents, molecules) linked to items (e.g. the web pages an user has viewed, the terms of a document, the properties of a molecule). In these KNN graphs, the similarity function is expressed as a set similarity between bags of items (possibly weighted), such as Jaccard's coefficient or cosine similarity. The goal of sampling is to limit the size of these bags of items and thus the time to compute the similarity.

Sampling might however degrade the resulting approximated KNN graph to a point where it becomes unusable, and must therefore be performed with care. In this paper we propose to sample the bags of items associated with each entity to a common fixed size s, by keeping their *s least popular* items. Our intuition is that less popular items are more discriminant when comparing entities than more popular or random items. For instance, the fact that Alice enjoys the original 1977 *Star Wars* movie tells us less about her tastes than the fact she also loves the 9 hour version of Abel Gance's 1927 *Napoléon* movie.

We compare this policy against three other sampling policies: *(i)* keeping the s most popular items of each entity, (*ii*) keeping s random items of each entity, and (*iii*) sampling the universe of items, independently of the entities. We evaluate these four sampling policies on four representative datasets. As a case study, we finally assess the effects of these strategies on recommendation, an emblematic application of KNN graphs. Our evaluation shows that our sampling policy clearly outperforms the other policies in terms of computation time and resulting quality: keeping the 25 least popular items reduces the computational time by up to 63%, while producing a KNN graph close to the ideal one. The recommendations done by using the resulting KNN graphs are moreover as good as the one relying on the exact KNN graph on all datasets.

The rest of this paper is organized as follows. In Section 2 we formally define the context of our work and our approach. The evaluation procedure is described in Section 3. Section 4 presents our experimental results. The related work is discussed in Section 5 and we conclude in Section 6.

2 Problem Statement: reduce KNN computation time.

2.1 System model and problem

For ease of exposition, we will speak about *users* rather than *entities*, but our approach remains applicable to any entity-item dataset. We consider a set of users $U = \{u_1, .., u_n\}$ in which each user u is associated with a set of items (the movies this user has liked, the pages she has viewed), termed her *profile*, and noted P_u. We note I the universe of all items: $I = \cup_{u \in U} P_u$.

A *k-nearest neighbor* (KNN) graph associates each user u with the set of k other users $knn(u) \subseteq U$ which are closest to u according to a given similarity metric on profiles:

$$\begin{aligned} sim : U \times U \to & \qquad \mathbb{R} \\ (u, v) \quad & \quad sim(u, v) = f_{sim}(P_u, P_v). \end{aligned}$$

Thus computing the KNN graph results in finding $knn(u)$ for each u such that

$$knn(u) \in \underset{S \in \mathcal{P}(U \setminus \{u\}) : |S| = k}{\text{argmax}} \sum_{v \in S} sim(u, v), \tag{1}$$

where $\mathcal{P}(X)$ is the powerset of a set X. We focus in this work on Jaccard similarity, a commonly used similarity metric, but our work can be applied to others. The Jaccard similarity between two users u and v is expressed as the size of the intersection of their profiles divided by the size of the union of their profiles:

$$f_{sim}(P_u, P_v) = J(P_u, P_v) = \frac{|P_u \cap P_v|}{|P_u \cup P_v|} \tag{2}$$

Since $|P_u \cup P_v| = |P_u| + |P_v| - |P_u \cap P_v|$, and since we can store $|P_u|$ for every user, computing the size of the intersection is the only non-trivial operation required to compute the Jaccard similarity.

2.2 Gance's Napoléon tells us more than Lucas's Star Wars

Computing the intersection $P_u \cap P_v$ is time consuming for large sets and is the main bottleneck of Jaccard's similarity. To reduce the complexity of this operation, we propose to sample each profile P_u into a subset $\widehat{P}_u$ in a preparatory phase applied when the dataset is loaded into memory, and to compute an approximated KNN graph on the sampled profiles.

Although simple, this idea has surprisingly never been applied to the computation of KNN graphs on entity-item datasets. Sampling carries however its own risks: if the items that are most characteristic of a user's profile get deleted, the KNN neighborhood of this user might become irremediably degraded. To avoid this situation, we adopt a *constant-size sampling* that strives to retain the *least popular items* in a profile.

The intuition is that unpopular items carry more information about a user's tastes than other items: if Alice and Bob have both enjoyed Abel Gance's *Napoléon*—a 1927 silent movie about Napoléon's early years—they are more likely to have similar tastes, than if they have both liked *Star Wars: A New Hope*—the 1977 first installment of the series, enjoyed by 96% of users[5].

2.3 Our approach: Constant-Size Least Popular Sampling (LP)

More formally, if the size of the profile of an user u is larger than a parameter s, we only keep its s least popular items

$$\widehat{P}_u \in \underset{S \in \mathcal{P}_u^s}{\text{argmin}} \sum_{i \in S} pop(i), \tag{3}$$

[5] `https://www.rottentomatoes.com/m/star_wars`, accessed 21 Feb. 2018

where $\mathcal{P}_u^s$ is the set of subsets of P_u of a given size s, i.e. $\mathcal{P}_u^s = \{S \in \mathcal{P}(I) : |S| = s \wedge S \subseteq P_u\}$, and $pop(i)$ is the popularity of item $i \in I$ over the entire dataset:

$$pop(i) = |\{u \in U : i \in P_u\}|. \tag{4}$$

If the profile's size is below s, the profile remains the same: $\widehat{P}_u = P_u$.

In terms of implementation, we compute the popularity of every item when reading the dataset from disk. We then use Equ. (3) to sample the profile of every user in a second iteration. The sampled profiles are finally used to estimate Jaccard's similarity between users when the KNN graph is constructed:

$$\widehat{J}(P_u, P_v) = J(\widehat{P}_u, \widehat{P}_v) = \frac{|\widehat{P}_u \cap \widehat{P}_v|}{|\widehat{P}_u| + |\widehat{P}_v| - |\widehat{P}_u \cap \widehat{P}_v|} \tag{5}$$

3 Experimental Setup

3.1 Baseline algorithms and competitors

Our Constant-Size Least Popular sampling policy (*LP* for short) can be applied to any KNN graph construction algorithm [4,5,10]. For simplicity, we apply it to a brute force approach that compares each pair of users and keeps the k most similar for each user. This choice helps focusing on the raw impact of sampling on the computation time and KNN quality, without any other interfering mechanism.

We use full profiles for our baseline, and compare our approach with three alternative sampling strategies: *constant-size most popular*, *constant-size random*, and *item sampling*.

Baseline: no sampling We use our brute force algorithm without sampling as our baseline. This approach yields an exact result, which we use to assess the approximation introduced by sampling, and provide a reference computing time.

Constant-size most popular sampling (MP) Similarly to LP, MP only keeps the s most popular items of each profile P_u:

$$\widehat{P}_u \in \underset{\mathcal{S} \in \mathcal{P}_u^s}{\operatorname{argmax}} \sum_{i \in S} pop(i). \tag{6}$$

As with LP, we do not sample the profile if its size is lower than s.

Constant-size random sampling (CS) This sampling policy randomly selects s items from P_u, with a uniform probability. As above, there is no sampling if the size of the profile is lower than s. In terms of implementation, this policy only requires one iteration over the data.

Dataset	**Users**	**Items**	**Scale**	**Ratings** > 3	$\lvert P_u \rvert$	$\lvert P_i \rvert$	**Density**
ml1M [13]	6,038	3,533	1-5	575,281	95.28	162.83	2.697%
ml10M [13]	69,816	10,472	0.5-5	5,885,448	84.30	562.02	0.805%
ml20M [13]	138,362	22,884	0.5-5	12,195,566	88.14	532.93	0.385%
AM [20]	57,430	171,356	1-5	3,263,050	56,82	19.04	0.033%

Table 1. The datasets used in our experiments

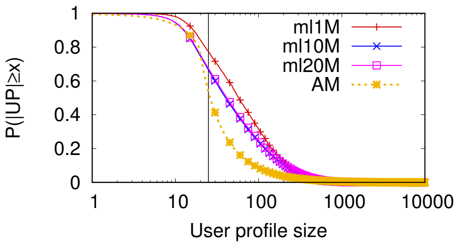


Fig. 1. CCDF of user profile sizes on the datasets used in the evaluation (positive ratings only). Between 77% (movielens1M) and 53% (AmazonMovies) of profiles are larger than the default cut-off value 25 (marked as a vertical bar).

Item Sampling (IS) This last policy uniformly removes items from the complete dataset. More precisely, each item $i \in I$ is kept with a uniform probability p to construct a reduced item universe $\hat{I}$ (i.e. $\forall i \in I : \mathbb{P}(i \in \hat{I}) = p$). The sampled profiles are then obtained by keeping the items of each profile that are also in $\hat{I}$: $\widehat{P}_u = P_u \cap \hat{I}$. On average, the profile of all users is reduced by a factor of $\frac{1}{p}$, but this policy does not adapt to the characteristics of individual profiles: small profiles run the risk of losing too much of their content to maintain good quality results.

3.2 Datasets

We use four publicly available datasets containing movie ratings (Table 1): 3 datasets from the MovieLens project, and one from Amazon. Ratings range from disliking (0.5 or 1) to liking (5). To apply Jaccard similarity, we binarize the datasets by keeping only ratings that reflect a positive opinion (i.e. > 3), *before* performing any sampling. Figure 1 shows the resulting Complementary Cumulative Distribution Functions (CCDF) of profile sizes for each dataset. For instance, more than 66% of users have profiles larger than 25 in movielens10M (ml10M). This means that a constant-size sampling with $s = 25$ on movielens10M removes more than 3 millions ratings (-69.23%).

The three Movielens datasets movielens1M (ml1M for short), movielens10M (ml10M) and movielens20M (ml20M) originate from GroupLens Research [13].

They contain movie reviews by on-line users from 1995 to 2015, and only consider users with more than 20 ratings.

The AmazonMovies dataset (AM) [20] aggregates movie reviews received by Amazon from 1997 to 2012. To avoid users with very few ratings (the so-called *cold start problem*), we only consider users with at least 20 ratings.

3.3 Evaluation metrics

We measure the effect of sampling along two main metrics: *(i)* their computation *time*, and *(ii)* the *quality* ratio of the resulting KNN graph.

The time is measured from the beginning of the execution of the algorithm, until the KNN graph is computed. It does not take into account the preprocessing of the dataset, which is evaluated separately in Section 4.2.

When applying sampling, the resulting KNN graph is an approximation of the exact one. In many applications such as recommender systems, this approximation should provide neighborhoods of high quality, even if those do not overlap with the exact KNN. To gauge this quality, we introduce the notion of *similarity ratio*, which measures how well the average similarity of an approximated graph compares against that of an exact KNN graph. Formally we define the *average similarity* of an approximate KNN graph $\widehat{G}_{\text{KNN}}$ as

$$avg\_sim(\widehat{G}_{\text{KNN}}) = \mathop{\mathbb{E}}_{(u,v)\in U^2 : v\in\widehat{knn}(u)} f_{sim}(P_u, P_v), \tag{7}$$

i.e. as the average similarity of the edges of $\widehat{G}_{\text{KNN}}$, and we define the *quality* of $\widehat{G}_{\text{KNN}}$ as its *normalized* average similarity

$$quality(\widehat{G}_{\text{KNN}}) = \frac{avg\_sim(\widehat{G}_{\text{KNN}})}{avg\_sim(G_{\text{KNN}})}, \tag{8}$$

where G_{KNN} is an ideal KNN graph, obtain without sampling.

A quality close to 1 indicates that the approximate neighborhoods of $\widehat{G}_{\text{KNN}}$ present a similarity that is very close to that of ideal neighborhoods, and can replace them with little loss in most applications, as we will show in the case of recommendations in our evaluation.

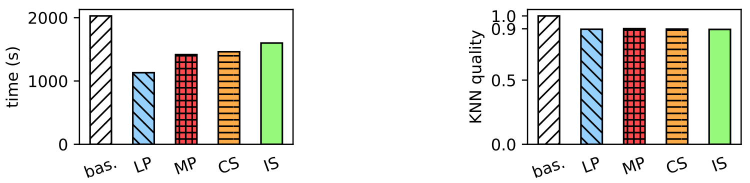
Throughout our experiments, we use a 5-fold cross-validation procedure which creates 5 training sets composed of 80% of the ratings. The remaining 20%, i.e. the training sets, are used for recommendations in Section 4.4. Our results are the average on the 5 resulting runs.

3.4 Experimental setup

We have implemented the sampling policies in Java 1.8. We ran our experiments on a 64-bit Linux server with two Intel Xeon E5420@2.50GHz, totaling 8 hardware threads, 32GB of memory, and a HHD of 750GB. We use all 8 threads.

Dataset	**Base.**	**LP**	*Δ* **(%)**	**MP**	*Δ* **(%)**	**CS**	*Δ* **(%)**	**IS**	*Δ* **(%)**
ml1M	19	**11**	**-40.5**	14.3	-24.7	14.2	-25.3	12.9	-32.1
ml10M	2028	**1131**	**-44.2**	1416.6	-30.1	1461.6	-27.9	1599.8	-21.1
ml20M	8393	**4865**	**-42.0**	5766.0	-31.3	5965.0	-28.9	6535.3	-22.1
AM	1862	**687**	**-63.1**	817.8	-56.1	748.1	-59,8	850.0	-54.4

Table 2. Computation time (s) of the baseline and the 4 sampling policies. The parameters were chosen to have a quality equal to 0.9. LP reduces computation time by 40% (ml1M) to 63% (AM), and outperforms other sampling policies on all datasets.



(a) Computation time (lower is better)

(b) KNN quality (higher is better)

Fig. 2. Computation time and KNN quality of the baseline and the sampling policies on movielens10M, when quality is set to 0.9. LP yields a reduction of 44.2% in computation time, outperforming other sampling policies.

Our code is available online[6]. In our experiments, we compute KNN graphs with k set to 30, which is a standard value.

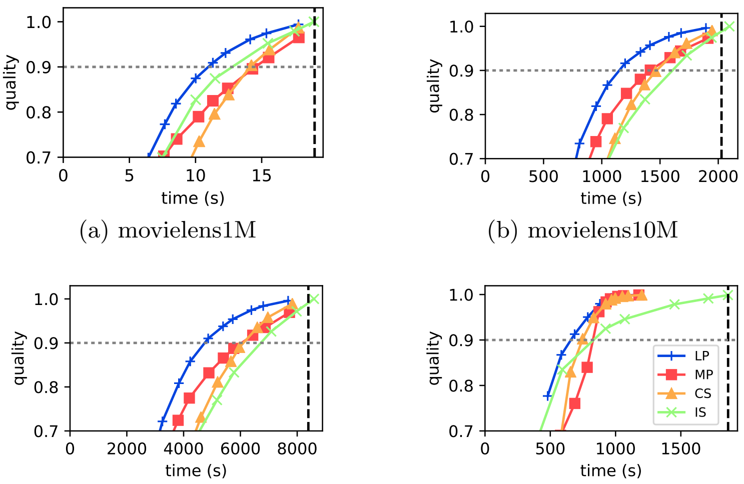
4 Experimentations

4.1 Reduction in computing time, and quality/speed trade-off

The baseline algorithm (without sampling) produces an exact KNN graph, with a quality of 1. To compare the different sampling policies (LP, MP, CS and IS) on an equal footing, we configure each of them on each dataset to achieve a quality of 0.9. The resulting parameter s ranges from 15 (LP on AM) to 75 (MP on movielens1M), while p (for IS) varies between 0.35 (on AmazonMovies) and 0.68 (on movielens20M). Table 2 summarizes the computation times measured on the four datasets with the percentage time reduction obtained against the baseline ($\varDelta$ columns), while Figure 2 shows the results on movielens10M. LP outperforms all other policies on all datasets, reaching a reduction of up to 63%.

Because they reduce the size of profiles, sampling policies exchange quality for speed. To better understand this trade-off, Figure 3 plots the evolution of the computation time and the resulting quality when s ranges from 5 to 200 for LP, MP, and CS ($s \in \{5, 10, 15, 20, 30, 40, 50, 75, 100, 200\}$), and p ranges from 0.1 to 1.0 for IS ($p \in \{0.1, 0.2, 0.4, 0.5, 0.75, 0.9, 1.0\}$).

[6] https://gitlab.inria.fr/oruas/SamplingKNN



(a) movielens1M

(b) movielens10M

(c) movielens20M

(d) AmazonMovies

Fig. 3. Trade-off between computation time and quality. Closer to the top-left corner is better. LP clearly outperforms all other sampling policies on all datasets.

Dataset	**Base.**	**LP**	Δ **(s)**	**MP**	Δ **(s)**	**CS**	Δ **(s)**	**IS**	Δ **(s)**
ml1M	0.36	0.50	+0.14	0.49	+0.13	0.46	+0.10	0.33	-0.03
ml10M	4.03	5.49	+1.46	5.67	+1.64	4.99	+0.96	3.98	-0.05
ml20M	8.55	11.95	+3.40	12.35	+3.80	11.05	+2.50	8.71	+0.16
AM	3.42	4.90	+1.48	4.70	+1.28	4.32	+0.90	2.41	-1.01

Table 3. Preprocessing time (seconds) for each dataset, and each sampling policy, with parameters set so that the resulting KNN quality is 0.9. The preprocessing times are negligible compared to the computation times.

For clarity, we only display points with a quality above 0.7, corresponding to the upper values of s and p. The dashed vertical line on the right shows the computation time of the baseline (producing a quality of 1), while the dotted horizontal line shows the quality threshold of 0.9 used in Table 2 and Figure 2.

Lines closer to the top-left corner are better. The figures confirm that our contribution, LP, outperforms other sampling policies on all datasets. There is however no clear winner among the remaining policies: IS performs well on movielens1M, but arrives last on the other datasets, and MP and CS show no clear order, which depends on the dataset and the quality considered.

4.2 Preprocessing overhead

As is common with KNN graph algorithms [5,10], the previous measurements do not include the loading and preprocessing time of the datasets, which is typically dominated by I/O rather than CPU costs. Sampling adds some overhead to this preprocessing, but Table 3 shows that this extra cost (Δ columns) remains negligible compared to the computation times of Table 2. For instance, LP adds 3.4s to the preprocessing of movielens20M, which only represents 0.07% of the complete execution time of the algorithm (4865s + 11.95s = 4877s). IS even decreases the preprocessing time on 3 datasets out of 4, by starkly reducing the bookkeeping costs of profiles while introducing only a low extra complexity.

4.3 Influence of LP at the user's level

Constant size sampling has a different influence on each user, depending on this user's profile's size. Profiles whose sizes are below the parameter s remain unchanged while larger profiles are truncated, thus losing information.

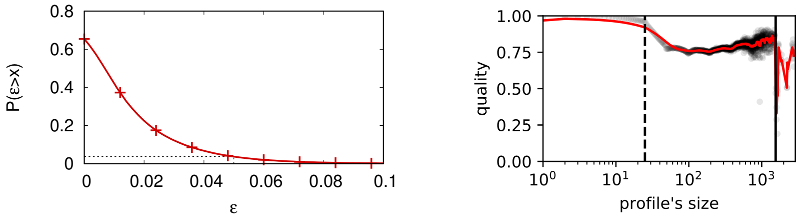
Figure 4 investigates the impact of this loss with our approach, LP, on movielens10M with $s = 25$ (corresponding to a quality of 0.9). Figure 4a plots the distribution of the similarity error $\epsilon = |J(P_u, P_v) - J(\widehat{P}_u, \widehat{P}_v)|$ introduced by sampling when ϵ is computed for each pair of users (u, v). The figure shows that 35% of pairs experience no error ($\epsilon = 0$), and that 96% have an error below 0.05 (dotted vertical line), confirming that our sampling only introduces a limited distortion of similarities.

Figure 4b represents the impact of LP on the quality of users' neighborhoods, according to the initial profile size of users. For every user u with an initial profile size of $|P_u|$, we compute the average similarity of u's approximated neighborhood $\widehat{knn}(u)$, and normalize this similarity with that of u's exact neighborhood $knn(u)$. The closest to 1 the better. We then average this normalized similarity for users with the same profile size $\{u \in U : |P_u| = P\}$. These points are displayed as a scatter plot (in black, note the log scale on the x axis), and using a moving average of width 50 (red curve). The first dashed vertical line is the value of the truncation parameter s ($x = 25$). The points after the second vertical line (at $x = 1553$) represent 24 users (out of 69816) and thus are not statistically significant. As expected, there is a clear threshold affect around the truncation value $s = 25$, yet even users with much larger profiles retain a high neighborhood quality, that remains on average above 0.75.

4.4 Recommendations

We want to evaluate the impact of the loss in quality on a practical use of the KNN graphs. To do so we perform item recommendations using the exact KNN graphs and the approximated graphs produced with LP. We recommend the items that an user u is more likely to like. This likelihood is expressed as a weighted average of the ratings the items received by the neighbors of u, weighted by the similarity of u with them. We use the real profiles, without sampling nor binarization, to compute these predicted ratings. After computing the score of every item, we recommend to u a set R_u composed by 30 items with the highest scores:

$$R_u \in \underset{R \subseteq I \setminus P_u : |R| = 30}{\operatorname{argmax}} \sum_{i \in S} \sum_{v \in knn_u} sim(u, v) * r_{v,i}, \tag{9}$$



(a) CCDF of the similarity's error. Only 4% of the users have their similarities changed by more than 0.05.

(b) Quality per user as a function of a user's profile size (note the log scale for x).

Fig. 4. Influence on the similarity and the quality of sampling with LP with $s = 25$ on movielens10M (total KNN quality equal to 0.9).

Dataset	**Base.**	**LP**	Δ
movielens1M	0.218	0.220	+0.002
movielens10M	0.273	0.275	+0.002
movielens20M	0.256	0.258	+0.002
AmazonMovies	0.595	0.596	+0.001

Table 4. Recommendation recall without sampling (*Base.*) and using the *Least Popular* (LP) policy (total KNN quality set to 0.9).

where $r_{v,i}$ is the rating made by the user v on the item i. We use the same 5-fold cross-validation as used for the KNN graph computation. We consider a recommendation successful when a recommended item is found within the 20% removed ratings (the testing set) with a rating above 3 ($r_{u,i} > 3$). The quality of the recommendation is measured using *recall*, the proportion of successful recommendations among all recommendations.

Table 4 shows the recall we obtain by using the exact KNN graphs obtained with the baseline and with LP using when the KNN quality is set to 0.9. In spite of its approximation, LP introduces no loss in recall, and even achieves slightly better scores than the baseline, which shows that our sampling approach can be used with little impact in concrete applications.

5 Related work

For small datasets, some specific data structures can be used to compute the KNN graphs very efficiently [3,18,21]. On the other hand, these solutions do not scale and computing efficiently exact KNN graphs with large datasets remains an open problem.

For large datasets, an approximation of the KNN graph, called approximate nearest-neighbor (ANN) graph, is computed instead, by decreasing the number of comparisons between users. Locally Sensitive Hashing [11,14] hashes users into buckets and only users within the same buckets are compared. Depending on the chosen similarity, different hashing functions are used [6,7,8]. Despite being very efficient for KNN queries, the preprocessing is too expensive to compete with other ANN graph algorithms. KIFF [5] first assigns to every user the users with which she shares at least one item. Since the Jaccard similarity is null if two users do not share any item, the neighbors research is limited to these ones. This algorithm performs particularly well on sparse datasets. Hyrec [4] and NNDescent [10] rely on the assumption that the neighbors of the neighbors are more likely to be also neighbors than random users to decrease drastically number of similarity computed.

However it seems that lowering even further the number of similarities is no longer possible. An orthogonal strategy is to speed-up the similarity computation itself by compacting the users' profiles. b-bit minwise hashing [2,16] relies on a similar approach than LSH to compact users' profiles in order to approximate the Jaccard similarity. It is space efficient but at the expense of a high preprocessing time. In [9] the profiles are compacted by using bit arrays: each bit represents a feature, which value has been rounded. This does not scale and cannot be used in our case where the items are the features. To avoid such a problem [12] uses constant-sized Bloom filters to encode the profiles. Then the Jaccard's similarity is approximated by a bitwise AND operation. Despite its privacy properties and its speed-up, there is a substantial loss in precision.

As far as we know, sampling has never been used to compact the users' profiles, even though it is used in information filtering systems such as collaborative filtering. It can be used to find association rules [1], to reduce the size of the items' universe to recommend [17] and to change the distribution of the training points [19,23]. The popularity is used to solve the cold-start problem [24] by finding items the new user is likely to rate, but not to represent its profile in a compact manner.

6 Conclusion

In this paper, we have proposed *Constant-Size Least Popular Sampling* (LP) to speed up the construction of KNN graphs on entity-item datasets. By keeping only the least popular items of users' profiles, we make them shorter and thus faster to compare. Our extensive evaluation on four realistic datasets shows that LP outperforms more straightforward sampling policies. More precisely, LP is able to decrease the computation time of KNN graphs by up to 63%, while providing a KNN graph close to the ideal one, with no observable loss when used to compute recommendations.

In the future, we plan to investigate more advanced sampling policies, and to explore how sampling could be combined with orthogonal greedy techniques to accelerate KNN graph computations [4,5,10].

Acknowledgments

This work was partially funded byt the PAMELA project of the French National Research Agency (ANR-16-CE23-0016), the Web-Alter-Ego Google Focused Award, the ANR-DFG joint project DISCMAT (ANR-14-CE35-0010) and the DeSceNt project granted by the Labex CominLabs excellence laboratory of the French Agence Nationale de la Recherche (ANR-10-LABX-07-01).

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *VLDB'94*, 1994.
2. Y. Bachrach and E. Porat. Sketching for big data recommender systems using fast pseudo-random fingerprints. In *ICALP*, 2013.
3. A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.
4. A. Boutet, D. Frey, R. Guerraoui, A.-M. Kermarrec, and R. Patra. Hyrec: leveraging browsers for scalable recommenders. In *Middleware*, 2014.
5. A. Boutet, A.-M. Kermarrec, N. Mittal, and F. Taïani. Being prepared in a sparse world: the case of knn graph construction. In *ICDE*, 2016.
6. A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997.*, 1997.
7. A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 1997.
8. M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC'2002*, 2002.
9. B. Cui, H. T. Shen, J. Shen, and K.-L. Tan. Exploring bit-difference for approximate knn search in high-dimensional databases. In *ADC*, 2005.
10. W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, 2011.
11. A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, 1999.
12. M. Gorai, K. Sridharan, T. Aditya, R. Mukkamala, and S. Nukavarapu. Employing bloom filters for privacy preserving distributed collaborative knn classification. In *WICT*, 2011.
13. F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *TIIS*, 2015.
14. P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.
15. J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. Lars: A location-aware recommender system. In *ICDE*, 2012.
16. P. Li and A. C. König. Theory and applications of b-bit minwise hashing. *Communications of the ACM*, 2011.
17. G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 2003.
18. T. Liu, A. W. Moore, K. Yang, and A. G. Gray. An investigation of practical approximate nearest neighbor algorithms. In *NIPS*, 2004.

19. I. Mani and I. Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Workshop on learning from imbalanced datasets, ICML*, 2003.
20. J. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *WWW*, 2013.
21. A. W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *UAI*, 2000.
22. N. Nodarakis, S. Sioutas, D. Tsoumakos, G. Tzimas, and E. Pitoura. Rapid aknn query processing for fast classification of multidimensional data in the cloud. *CoRR*, 2014.
23. R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *ICDM*, 2008.
24. A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *IUI*, 2002.