

# Variational Bayesian PCA versus $k$ -NN on a Very Sparse Reddit Voting Dataset

Jussa Klapuri, Ilari Nieminen, Tapani Raiko, and Krista Lagus

Department of Information and Computer Science,  
Aalto University, FI-00076 AALTO, Finland

{jussa.klapuri, ilari.nieminen, tapani.raiko, krista.lagus}@aalto.fi

**Abstract.** We present vote estimation results on the largely unexplored Reddit voting dataset that contains 23M votes from 43k users on 3.4M links. This problem is approached using Variational Bayesian Principal Component Analysis (VBPCA) and a novel algorithm for  $k$ -Nearest Neighbors ( $k$ -NN) optimized for high dimensional sparse datasets without using any approximations. We also explore the scalability of the algorithms for extremely sparse problems with up to 99.99% missing values. Our experiments show that  $k$ -NN works well for the standard Reddit vote prediction. The performance of VBPCA with the full preprocessed Reddit data was not as good as  $k$ -NN's, but it was more resilient to further sparsification of the problem.

**Keywords:** Collaborative filtering, nearest neighbours, principal component analysis, Reddit, missing values, scalability, sparse data

## 1 Introduction

Recommender systems are software tools and techniques providing suggestions for items or objects that are assumed to be useful to the user [11]. These suggestions can relate to different decision-making processes, such as which books might be interesting, which songs you might like, or people you may know in a social network. The particular interest of an RS is that of reddit.com [10], which is a type of online community where *users* can vote *links* either up or down, i.e. *upvote* or *downvote*. Reddit currently has a global Alexa rank of 118 and 52 in the US [2]. Our objective is to study whether the user will like or dislike some content based on a priori knowledge of the user's voting history.

This kind of problem can generally be approached efficiently by using *collaborative filtering* (CF) methods. In short, collaborative filtering methods produce user specific recommendations of links based on voting patterns without any need of exogenous information about either users or links [11]. CF systems need two different entities in order to establish recommendations: *items* and *users*. In this paper items are referred to as *links*. With users and links, conventional techniques model the data as a sparse user-link matrix, which has a row for each user and a column for each link. The nonzero elements in this matrix are the votes.

The two main techniques of CF to relate users and links are *the neighborhood approach* and *latent factor models*. The neighborhood methods are based on finding similar neighbors to either links or users and computing the prediction based on these neighbors' votes, for example, finding  $k$  nearest neighbors and choosing the majority vote. Latent factor models approach this problem in a different way by attempting to explain the observed ratings by uncovering some latent features from the data. These models include neural networks, Latent Dirichlet Allocation, probabilistic Latent Semantic Analysis and SVD-based models [11].

### 1.1 Related Work

This paper is heavily based on a master's thesis [7], and the preprocessed datasets are the same. The approach of estimating votes from the Reddit dataset in this paper is similar to [9] although they preprocessed the dataset in a different way and used root-mean-square error as the error measure while this paper uses classification error and class average accuracy. There has also been some research on comparing a  $k$ -NN classifier to SVM with several datasets of different sparsity [3], but the datasets they used were lower-dimensional, less sparse and not all users were evaluated in order to speed up the process. Still, their conclusion that  $k$ -NN starts failing at a certain sparsity level compared to a non-neighborhood model is the same as in this paper.

## 2 Dataset

Reddit dataset originated from a topic in social news website Reddit [6]. It was posted by a software developer working for Reddit in early 2011 in hopes of improving their own recommendation system. The users in the dataset represent almost 17% of the total votes on the site in 2008, which means that the users represented in the dataset are very in the whole Reddit community.

The original dataset consists of  $N = 23\,091\,688$  votes from  $n = 43\,976$  users over  $d = 3\,436\,063$  links in 11 675 subreddits. Subreddits are similar to subforums, containing links that have some similarity but this data was not used in the experiments. The dataset does not contain any additional knowledge on the users or the content of the links, only user-link pairs with a given vote that is either -1 (a downvote) or 1 (an upvote).

Compared to the Netflix Prize dataset [8], Reddit dataset is around 70 times sparser and has only two classes. For the missing votes, there is no information on whether a user has seen a link and decided not to vote or simply not having seen the link at all. In this paper, the missing votes are assumed to be missing at random (MAR) [12]. This is a reasonable assumption due to high dimensionality of the data and low median number of votes per user.

### 2.1 Preprocessing

The dataset can be visualized as a bipartite undirected graph (Figure 1). Even though no graph-theoretic approaches were used in solving the classification

problem, this visualization is particularly useful in explaining the preprocessing and the concept of core of the data. In graph theory, the degree of a vertex  $v$  equals to the number of edges incident to  $v$ . If the degree of any vertex is very low, there may not be enough data about the vertex to infer the value of an edge, meaning the vote, where this vertex is in the either end. An example of this can be found in Figure 1 between  $link_4$  and  $user_7$ , where the degree of  $user_7$  is 1 and the degree  $link_4$  is 3. Clearly this is a manifestation of the *new user problem* [1], meaning that the user would have to vote more links and the link would have to get more votes in order to accurately estimate the edge.

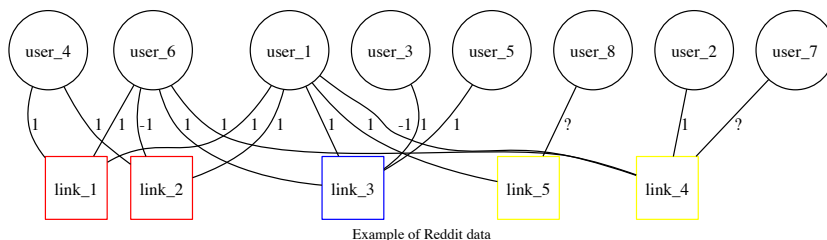


Fig. 1: Bipartite graph representation of Reddit data. Subreddits are visualized as colors on the links (square vertices). The vote values -1,1 are represented as numbers on the arcs, where “?” means no vote has been given.

These kinds of new users and unpopular links make the estimation task very difficult and thus should be pruned out of the data. This can be done by using *cutoff* values such that all the corresponding vertices having a degree below the cutoff value are pruned out of the data. With higher cutoff values, only a subset of the data called *core of the data* remains, which is similar to the idea of  $k$ -cores [13]. This subset contains the most active users who have voted a large part of all the remaining links, and the most popular links which are voted by almost every user. The full Reddit dataset was pruned into two smaller datasets, namely, the big dataset using 4 as the cutoff value for all vertices (users and links), and the small dataset with a stricter cutoff value of 135. See [7] for more details on these parameters.

The resulting  $n \times d$  data matrices  $M$  were then split randomly into a training set  $M_{\text{train}}$  and a test set  $M_{\text{test}}$ , such that the training set contained 90% of the votes and the test set the remaining 10%, respectively. The splitting algorithm worked user-wise, i.e., it randomly divided a user’s votes between the training set and test set for all users such that at least one vote was put into the test set, even if the user had given less than 10 votes.

Training set properties are described in Table 1 where it is apparent that the ratio of upvotes gets higher as the pruning process gets closer to the core

of the data. In general the estimation of downvotes is a lot more difficult than upvotes. This is partly due to the fact that downvotes are rarer and thus the prior probability for upvotes is around six to nine times higher than for the downvotes. This also means that the dataset is sparser when using only the downvotes. Summary statistics of the nonzero values in the training sets are given in Table 2. Small test set contained 661,903 votes and big test set 1,915,124 votes.

Table 1: Properties of votes given by users and votes for links for the preprocessed datasets.

Dataset	Users ( $n$ )	Links ( $d$ )	Votes ( $N$ )	Density ( $\frac{N}{nd}$ )	Ratio of upvotes
Small	7,973	22,907	5,990,745	0.0328	0.9015
Big	26,315	853,009	17,349,026	0.000773	0.8688
Full	43,976	3,436,063	23,079,454	0.000153	0.8426

Table 2: Summary statistics of the training sets on all given votes in general. Downvotes and upvotes are considered the same here.

	Mean	Median	Std	Max
Small dataset: Users	751.4	354	1156.2	15820
Links	261.5	204	159.49	2334
Big dataset: Users	659.3	101	2335.7	82011
Links	20.3	7	55.6	3229

### 3 Methods

This section describes the two methods used for the vote classification problem: the  $k$ -nearest neighbors ( $k$ -NN) classifier and Variational Bayesian Principal Component Analysis (VBPCA). The novel  $k$ -NN algorithm is introduced in Section 3.2.

#### 3.1 $k$ -Nearest Neighbors

Nearest neighbors approach estimates the behavior of the active user  $u$  based on users that are the most similar to  $u$  or likewise find links that are the most similar to the voted links. For the Reddit dataset in general, link-wise  $k$ -NN

seems to perform better than user-wise  $k$ -NN [7] so every  $k$ -NN experiment was run link-wise.

For the  $k$ -NN model, *vector cosine-based similarity* and *weighted sum of others' ratings* were used, as in [14]. More formally, when  $\mathcal{I}$  denotes the set of all links,  $i, j \in \mathcal{I}$  and  $\mathbf{i}, \mathbf{j}$  are the corresponding vectors containing votes for the particular links from all users, the similarity between links  $i$  and  $j$  is defined as

$$w_{ij} = \cos(\mathbf{i}, \mathbf{j}) = \frac{\mathbf{i} \cdot \mathbf{j}}{\|\mathbf{i}\| \|\mathbf{j}\|}. \quad (1)$$

Now, let  $\mathcal{N}_u(i)$  denote the set of closest neighbors to link  $i$  that have been rated by user  $u$ . The classification of  $\hat{r}_{ui}$  can then be performed link-wise by choosing an odd number of  $k$  neighboring users from the set  $\mathcal{N}_u(i)$  and classifying  $\hat{r}_{ui}$  to the class that contains more votes. Because the classification depends on how many neighbors  $k$  are chosen in total, all the experiments were run for 4 different values of  $k$ . For better results, the value of parameter  $k$  can be estimated through  $K$ -fold cross validation, for example.

However, this kind of simple neighborhood model does not take into account that users have different kinds of voting behavior. Some users might give downvotes often while some other users might give only upvotes. For this reason it is good to introduce rating normalization into the final  $k$ -NN method [14]:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in \mathcal{N}_u(i)} w_{ij}(r_{uj} - \bar{r}_j)}{\sum_{j \in \mathcal{N}_u(i)} |w_{ij}|}. \quad (2)$$

Here, the term  $\bar{r}_i$  denotes the average rating by all user to the link in  $i$  and is called the mean-centering term. Mean-centering is also included in the nominator for the neighbors of  $i$ . The denominator is simply a normalizing term for the similarity weights  $w_{ij}$ .

The algorithm used for computing link-wise  $k$ -NN using the similarity matrix can generally be described as:

1. Compute the full  $d \times d$  similarity matrix  $S$  from  $M_{\text{test}}$  using cosine similarity. For a  $n \times d$  matrix  $M$  with normalized columns,  $S = M^T M$ .
2. To estimate  $\hat{r}_{ui}$  using Eq. (2), find  $k$  related links  $j$  for which  $r_{uj}$  is observed, with highest weights in the column vector  $S_i$ .

For the experiments, this algorithm is hereafter referred to as “ $k$ -NN full”.

### 3.2 Fast sparse $k$ -NN Implementation

For high-dimensional datasets, computing the similarity matrix  $S$  may be infeasible. For example, the big Reddit dataset would require computing half of a  $853,009 \times 853,009$  similarity matrix, which would require some hundreds of gigabytes of memory. The algorithm used for computing fast sparse  $k$ -NN is as follows:

1. Normalize link feature vectors (columns) of  $M_{\text{train}}$  and  $M_{\text{test}}$ .
2. FOR each user  $u$  (row) of  $M_{\text{test}}$  DO
  - (a) Find all the links  $j$  for which  $r_{uj}$  is observed and collect the corresponding columns of  $M_{\text{train}}$  to a matrix  $A_u$ .
  - (b) Find all the links  $j$  for which  $\hat{r}_{uj}$  is to be estimated and collect the corresponding columns of  $M_{\text{test}}$  to a matrix  $B_u$ .
  - (c) Compute the pseudosimilarity matrix  $S_u = A_u^T B_u$  which corresponds to measuring cosine similarity for each link between the training and test sets for user  $u$ .
  - (d) Find the  $k$  highest values (weights) for each column of  $S_u$  and use Eq. (2) for classification.
3. END

This algorithm is referred to as “ $k$ -NN sparse” in Sec. 4.

Matrix  $S_u$  is called a pseudosimilarity matrix since it is not a symmetric matrix and most likely not even a square matrix in general. This is because the rows correspond to the training set feature vectors and the columns to the test set feature vectors. The reason this algorithm is fast is due to the high sparsity and high dimensions of  $M$ . Also, the fact that only the necessary feature vectors from training and test are multiplied as well as parallelizing this in matrix operation, makes this operation much faster. On the contrary, for less sparse datasets this is clearly slower than computing the full similarity matrix  $S$  from the start. If the sparsifying process happens to remove all votes from a user from the training set, a naive classifier is used which always gives an upvote. This model could probably be improved by attempting to use user-wise  $k$ -NN in such a case. With low mean votes per user, the matrices  $S_u$  stay generally small, but for the few extremely active users,  $S_u$  can still be large enough not to fit into memory (over 16 GB). In these cases,  $S_u$  can easily be computed in parts without having to compute the same part twice in order to classify the votes.

It is very important to note the difference to a similar algorithm that would classify the votes in  $M_{\text{test}}$  one by one by computing the similarity only between the relevant links voted in  $M_{\text{train}}$ . While the number of similarity computations would stay the same, the neighbors  $\mathcal{N}_u(i)$  would have to be retrieved from  $M_{\text{train}}$  again for each user-link pair  $(u, i)$ , which may take a significant amount of time for a large sparse matrix though the memory usage would be much lower.

### 3.3 Variational Bayesian Principal Component Analysis

Principal Component Analysis (PCA) is a technique that can be used to compress high dimensional vectors into lower dimensional ones and has been extensively covered in literature, e.g. [5].

Assume we have  $n$  data vectors of dimension  $d$  represented by  $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_n$  that are modeled as

$$\mathbf{r}_u \approx \mathbf{W}\mathbf{x}_u + \mathbf{m}, \quad (3)$$

where  $\mathbf{W}$  is a  $d \times c$  matrix,  $\mathbf{x}_j$  are  $c \times 1$  vectors of principal components and  $\mathbf{m}$  is a  $d \times 1$  bias vector.

PCA estimates  $\mathbf{W}$ ,  $\mathbf{x}_u$  and  $\mathbf{m}$  iteratively based the observed data  $\mathbf{r}_u$ . The solution for PCA is the unique principal subspace such that the column vectors of  $\mathbf{W}$  are mutually orthonormal and, furthermore, for each  $k = 1, \dots, c$ , the first  $k$  vectors form the  $k$ -dimensional principal subspace. The principal components can be determined in many ways, including singular value decomposition, least-square technique, gradient descent algorithm and alternating W-X algorithm. All of these can also be modified to work with missing values. More details are discussed in [4].

Variational Bayesian Principal Component Analysis (VBPCA) is based on PCA but includes several advanced features, including regularization, adding the noise term into the model in order to use Bayesian inference methods and introducing prior distributions over the model parameters. VBPCA also includes *automatic relevance determination* (ARD), resulting in less relevant components tending to zero when the evidence for the corresponding principal component for reliable data modeling is weak. In practise this means that the decision of the number of components to choose is not so critical for the overall performance of the model, if there is not too few chosen. The computational time complexity of VBPCA using the gradient descent algorithm per one iteration is  $O((N+n+d)c)$ . More details on the actual model and implementation can be found in [4].

In the context of the link voting prediction, data vectors  $\mathbf{r}_u$  contain the votes of user  $u$  and the bias vector  $\mathbf{m}$  corresponds to the average ratings  $\bar{r}_i$  in Eq. (2). The  $c$  principal components can be interpreted as features of the links and people. A feature might describe how technical or funny a particular link  $i$  is (numbers in the  $1 \times c$  row vector of  $\mathbf{W}_i$ ), and how much a person enjoys technical or funny links (the corresponding numbers in the  $c \times 1$  vector  $\mathbf{x}_u$ ). Note that we do not analyse or label different features here, but they are automatically learned from the actual votes in an unsupervised fashion.

## 4 Experiments

The experiments were performed on the small and big Reddit datasets as described in Section 2. Also, we further sparsified the data such that for each step, 10% of the remaining non-zero values in the training set were removed completely at random. Then the models were taught with the remaining training set and error measures were calculated for the original test set. In total there were 45 steps such that during the last step the training set is around 100 times sparser than the original. Two error measures were utilized, namely classification error and class average accuracy. The number of components for VBPCA was chosen to be the same as in [7], meaning 14 for the small dataset and 4 for the big dataset. In short, the heuristic behind choosing these values was based on the best number of components for SVD after 5-fold cross validation and doubling it, since VBPCA uses ARD. Too few components would make VBPCA underperform and too many would make it overfit the training set, leading to poorer performance.

The  $k$ -NN experiments were run for all 4 different  $k$  values simultaneously, which means that the running times would be slightly lower if using only a single value for  $k$  during the whole run.

Classification error was used as the error measure, namely  $1/N \#\{\hat{r}_{ui} \neq r_{ui}\}$ . Class average accuracy is defined as the average between the proportions of correctly estimated downvotes and correctly estimated upvotes. For classification error, lower is better while for the value of class average accuracy higher is better.

VBPCA was given 1000 steps to converge while also using the rmsstop criterion, which stops the algorithm if either the absolute difference  $|\text{RMSE}_{t-50} - \text{RMSE}_t| < 0.0001$  or relative difference  $|\text{RMSE}_{t-50} - \text{RMSE}_t|/\text{RMSE}_t < 0.001$ . Since VBPCA parameters are initialized randomly, the results and running times fluctuate, so VBPCA was ran 5 times per each step and the mean of all these runs was visualized.

In addition to  $k$ -NN and VBPCA, naive upvote and naive random models were also implemented. Naive upvote model estimates all  $\hat{r}_{ui}$  as upvotes and naive random model gives an estimate of upvote with the corresponding probability of upvotes from the training set, e.g.  $p = 0.9015$  for small dataset, and a downvote with probability  $1 - p$ .

All of the algorithms and experiments were implemented in Matlab running on Linux using an 8-core 3.30 GHz Intel Xeon and 16 GB of memory. However, there were no explicit multicore optimizations implemented and thus  $k$ -NN algorithms were practically running on one core. VBPCA was able to utilize more than one core.

#### 4.1 Results for Small Reddit Dataset

The results for the methods on the original small Reddit dataset are seen in Table 3, which includes the user-wise  $k$ -NN before sparsifying the dataset further. The naive model gives a classification error of 0.0982, the dummy baseline for the small dataset. The running time of  $k$ -NN full is slightly lower than  $k$ -NN sparse in Figure 2a during the first step, but after sparsifying the training set,  $k$ -NN sparse performs much faster. It can be seen from Figure 2b that the  $k$ -NN classifier performs better to a certain point of sparsity, after which VBPCA is still able to perform below dummy baseline. This behavior may partly be explained by the increasing number of naive classifications by the  $k$ -NN algorithm caused by the increasing number of users in  $M_{\text{train}}$  with zero votes given. Figure 2c indicates that the higher the number of neighbors  $k$ , the better. Downvote estimation is consistently higher with VBPCA than with  $k$ -NN (Figure 2d).

#### 4.2 Results for Big Reddit Dataset

Classification error for the dummy baseline is 0.1313 for the big dataset. Figure 3b indicates that the fast  $k$ -NN classifier loses its edge against VBPCA quite early on, around the same sparsity level as for the small dataset. However, VBPCA seems to take more time on converging (Figure 3a). Higher  $k$  values lead to better performance, as indicated in Figure 3c. Downvote estimation with



Table 3: Metrics for different methods on the small Reddit dataset.

Method	Class				
	Accuracy	error	accuracy average	Downvotes	Upvotes
Naive	0.9018	0.0982	0.5000	0.0000	<b>1.0000</b>
Random	0.8225	0.1775	0.4990	0.0965	0.9016
$k$ -NN_User	0.9176	0.0824	0.6256	0.2621	0.9890
$k$ -NN_Link	<b>0.9237</b>	<b>0.0763</b>	0.6668	0.3470	0.9865
VBPCA	0.9222	0.0778	<b>0.6837</b>	<b>0.3870</b>	0.9805

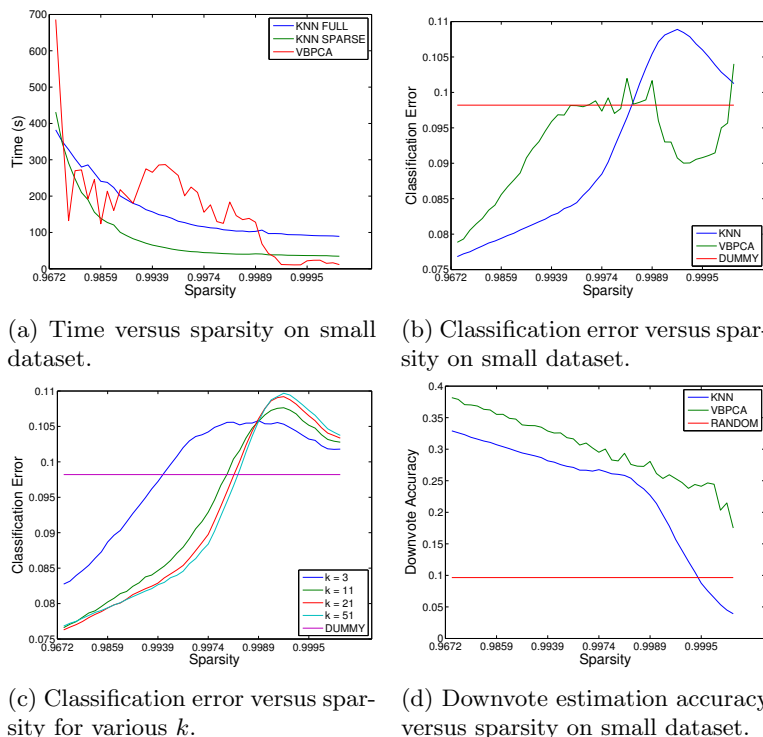


Fig. 2: Figures of the experiments on small dataset.

$k$ -NN seems to suffer a lot from sparsifying the training set (Figure 3d), while VBPCA is only slightly affected.

## 5 Conclusions

In our experimental setting we preprocessed the original Reddit dataset into two smaller subsets containing some of the original structure and artificially sparsified the datasets even further. It may be problematic or even infeasible to use

Table 4: Metrics for different methods on the big Reddit dataset.

Method	Class				
	Accuracy	error	accuracy average	Downvotes	Upvotes
Naive	0.8687	0.1313	0.5000	0.0000	<b>1.0000</b>
Random	0.7720	0.2280	0.5002	0.1316	0.8689
$k$ -NN_User	0.8930	0.1070	0.6738	0.3766	0.9711
$k$ -NN_Link	<b>0.9048</b>	<b>0.0952</b>	<b>0.7091</b>	<b>0.4438</b>	0.9745
VBPCA	0.8991	0.1009	0.6929	0.4132	0.9726

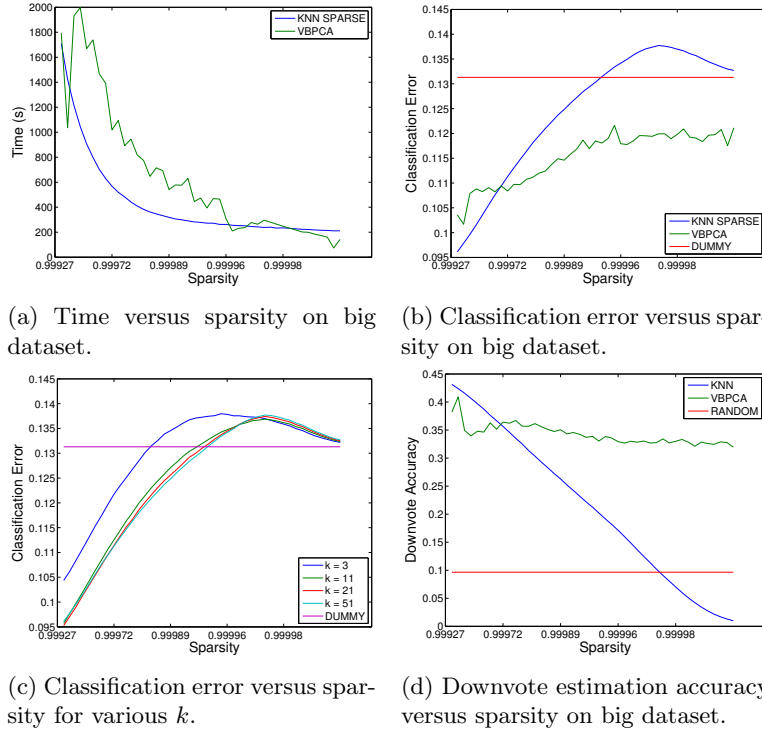


Fig. 3: Figures of the experiments on big dataset.

standard implementations of  $k$ -NN classifier for the high-dimensional very sparse dataset such as the Reddit dataset, but this problem was avoided using the fast sparse  $k$ -NN presented in Sec. 3. While initially  $k$ -NN classifier seems to perform better, VBPCA starts performing better when the sparsity of the datasets grow beyond approximately 0.99990. VBPCA is especially more unaffected by the increasing sparsity for the downvote estimation, which is generally much harder.

There are many ways to further improve the accuracy of  $k$ -NN predictions. The fact that the best results were obtained with the highest number of neighbors

( $k=51$ ) hints that the cosine-based similarity weighting is more important to the accuracy than a limited number of the neighbors. One could, for instance, define a tunable distance metric such as  $\cos(\mathbf{i}, \mathbf{j})^\alpha$ , and find the best  $\alpha$  by cross-validation. The number of effective neighbors (sum of weights compared to the weight of the nearest neighbor) could then be adjusted by changing  $\alpha$  while keeping  $k$  fixed to a large value such as 51.

## References

- [1] Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17(6), 734–749 (2005)
- [2] Alexa: Alexa - reddit.com site info. <http://www.alexa.com/siteinfo/reddit.com>, accessed May 2, 2013
- [3] Grčar, M., Fortuna, B., Mladenič, D., Grobelnik, M.: knn versus svm in the collaborative filtering framework. In: *Data Science and Classification*, pp. 251–260. Springer (2006)
- [4] Ilin, A., Raiko, T.: Practical approaches to principal component analysis in the presence of missing values. *Journal of Machine Learning Research (JMLR)* 11, 1957–2000 (July 2010)
- [5] Jolliffe, I.T.: *Principal Component Analysis*. Springer, second edn. (2002)
- [6] King, D.: Want to help reddit build a recommender? – a public dump of voting data that our users have donated for research. <http://redd.it/dtg4j> (2010), accessed January 17, 2013
- [7] Klapuri, J.: *Collaborative Filtering Methods on a Very Sparse Reddit Recommendation Dataset*. Master’s thesis, Aalto University School of Science (2013)
- [8] Netflix: Netflix prize webpage. <http://www.netflixprize.com/> (2009), accessed January 5, 2012
- [9] Poon, D., Wu, Y., Zhang, D.Q.: Reddit recommendation system. <http://cs229.stanford.edu/proj2011/PoonWuZhang-RedditRecommendationSystem.pdf> (2011), accessed January 17, 2013
- [10] Reddit: reddit: the front page of the internet. <http://www.reddit.com/about/>, accessed January 17, 2013
- [11] Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.): *Recommender Systems Handbook*. Springer (2011)
- [12] Rubin, D.B.: *Multiple Imputation for Nonresponse in Surveys*. Wiley (1987)
- [13] Seidman, S.: Network structure and minimum degree. *Social networks* 5(3), 269–287 (1983)
- [14] Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. *Adv. in Artif. Intell.* 2009, 4:2–4:2 (January 2009), <http://dx.doi.org/10.1155/2009/421425>