Under consideration for publication in Knowledge and Information Systems

Two Collaborative Filtering Recommender Systems Based on Sparse Dictionary Coding

Ismail E. Kartoglu[1], Michael W. Spratling[1]

[1]Department of Informatics, King's College London, London, UK

Abstract. This paper proposes two types of recommender systems based on sparse dictionary coding. Firstly, a novel predictive recommender system that attempts to predict a user's future rating of a specific item. Secondly, a top-n recommender system which finds a list of items predicted to be most relevant for a given user. The proposed methods are assessed using a variety of different metrics and are shown to be competitive with existing collaborative filtering recommender systems. Specifically, the sparse dictionary-based predictive recommender has advantages over existing methods in terms of a lower computational cost and not requiring parameter tuning. The sparse dictionary-based top-n recommender system has advantages over existing methods in terms of the accuracy of the predictions it makes and not requiring parameter tuning. An open-source software implemented and used for the evaluation in this paper is also provided for reproducibility.

Keywords: Recommender systems; algorithms; sparse coding; evaluation

1. Introduction

Recommender systems allow people to find products and services. They are becoming increasingly important in a range of applications, such as e-commerce, music, film and book recommendation, web search, e-learning, health and finding legal precedents (Bobadilla et al., 2013). Sparse dictionary coding is used to represent a signal using a sparse set of basis vectors selected from an overcomplete dictionary. It has applications in data compression, image and signal restoration, and in pattern classification (Wright et al., 2010). Previous work has shown that

Received xxx
Revised xxx
Accepted xxx

sparse dictionary coding can also be used to implement recommender systems (Ning and Karypis, 2011; Szabó et al., 2012).

There are different types of recommender systems that serve slightly different purposes. These include predictive recommender systems and top-n recommender systems (Herlocker et al., 2004). Predictive recommender systems predict user ratings, and they are not necessarily responsible for making recommendations (Herlocker et al., 2004). Making recommendations is the responsibility of top-n recommendation algorithms. These are responsible for ranking items so the more relevant items are presented to the user before other items. Unlike predictive algorithms, these algorithms do not necessarily make rating predictions (Herlocker et al., 2004).

2. Related Work

Póczos et al. (Szabó et al., 2012) proposed a predictive recommender system that is partly based on sparse coding, but used additional mechanisms for neighbour correction. In contrast, the method proposed here uses sparse coding without additional mechanisms and is thus simpler than the one in (Szabó et al., 2012). Another difference is that (Szabó et al., 2012) implemented a model-based recommender system, whereas the system proposed here is a memory-based (or neighbourhood-based) method (Breese et al., 1998; Ning and Karypis, 2011). Memory-based methods use a user-item matrix to find similar items or users and make recommendations based on those similarities. Model-based methods, on the other hand, employ the user-item matrix to train a model, and recommendations are then made using the trained model.

Ning et al. (Ning and Karypis, 2011) proposed a state-of-the-art top-n recommender system based on sparse coding that they named SLIM. The SLIM top-n recommender performs better than many other methods in the field in terms of recall and hit-rate. The method proposed here is very similar. However, there are two notable differences. Firstly, whereas Ning et al. solve the sparse coding problem using a regularised linear optimisation method, we explore other methods of sparse coding. Specifically, i) a greedy approach, using the PFP algorithm (Plumbley, 2006) which finds a single column of the dictionary at a time that reduces the error maximally and which can switch out the bases in the partial solution, ii) and a neural network approach, using the DIM algorithm (Spratling, 2014) which makes use of competing neurones to represent a given input and the neurone that wins the competition suppresses other neurones and this results in sparse solutions. Secondly, whereas the method in (Ning and Karypis, 2011) is a model-based approach, as such it requires parameter tuning, the methods described in this paper do not require parameter tuning. This paper shows that the top-n recommendation performance results of the approaches described in this paper are competitive with those of the state-of-the-art top-n recommendation algorithms in terms of some commonly used top-n recommendation metrics. This paper evaluates performance using datasets with different characteristics. Our results suggest that a personalisation metric should be used along with top-n recommendation quality metrics for a better evaluation of top-n recommenders. In this paper we also show that the SLIM recommender produces more personalised recommendations than other methods in the literature.

The proposed predictive recommender system is found to be competitive with an item-based k-nearest neighbours (k-NN) algorithm and with a matrix

factorisation (MF) algorithm (Koren et al., 2009) in terms of predictive accuracy. However, it has advantages over these algorithms in terms of execution speed and not requiring any parameter tuning. The proposed predictive algorithm is similar to the proposed top-n algorithm, however there are two important differences without which the predictive algorithm does not perform well: i) the proposed predictive algorithm removes some columns from the user-item matrix before it calculates sparse representations, ii) it normalises predicted ratings to be more accurate.

3. Methods

Software which implements the sparse dictionary based recommender systems described in this section, and which can be used to reproduce all the results, can be downloaded from `https://github.com/iemre/MRSR`. This software provides flexibility that allows researchers to plug in their own recommendation algorithm. Evaluation is handled by the software.

Formally, the problem of sparse coding can be expressed as follows:

$$\min ||\mathbf{x}||_0 \text{ subject to } \mathbf{b} = \mathbf{A}\mathbf{x}, \tag{1}$$

where the zero pseudo-norm $||\mathbf{x}||_0$ counts the number of non-zero elements in $\mathbf{x}$, $\mathbf{b}$ is a given vector for which a sparse representation is sought, and $\mathbf{A}$ is a given matrix (or dictionary) whose columns are linearly combined by the sparse representation $\mathbf{x}$ to reconstruct $\mathbf{b}$. In our application, $\mathbf{A}$ is the user-item matrix, $\mathbf{b}$ is a vector of ratings with missing value(s) that need to be predicted, and $\mathbf{x}$ is a sparse set of coefficients that select columns of the user-item matrix in order to reconstruct $\mathbf{b}$. The selected columns of $\mathbf{A}$ are not necessarily similar to $\mathbf{b}$, but in combination can accurately reconstruct $\mathbf{b}$. This is in contrast to typical collaborative filtering recommender systems, e.g. those implemented using k-NN, that select columns of $\mathbf{A}$ based on similarity to $\mathbf{b}$. In our application, the ratings, and hence the values in $\mathbf{b}$ and $\mathbf{A}$ are non-negative. The sparse coefficients, $\mathbf{x}$, are also constrained to be non-negative as it makes little intuitive sense to reconstruct one rating by subtracting another. This additional constraint was also found to result in superior performance.

Ning et al. (Ning and Karypis, 2011) model the sparse coding problem as a regularised linear optimisation problem by using $||\mathbf{x}||_1$ instead of $||\mathbf{x}||_0$, which is the correct mathematical formulation of the problem "find a vector with as few non-zero coefficients as possible". $||\mathbf{x}||_1$ is found to yield sparse solutions (Elad, 2010)(Bruckstein et al., 2009) and is also used to make finding solutions more tractable. Determining which (if either) is better is an empirical issue that we are contributing to answering, but is not a focus of the paper. Ning et al. (Ning and Karypis, 2011) use coordinate descent and soft thresholding to solve the optimisation problem. To solve the sparse coding problem in Equation 1, we used the DIM sparse solver (Spratling, 2014), and the PFP sparse solver (Plumbley, 2006).

The DIM sparse solver is a neural network based approach in which a set of neurones compete with each other to represent the input signal $\mathbf{b}$. This competition results in sparse solutions. The PFP sparse solver is a greedy approach that makes use of the geometry of polytopes to find columns of $\mathbf{A}$ to represent the input $\mathbf{b}$. Whereas the model proposed by (Ning and Karypis, 2011) requires

some parameters to be tuned, DIM and PFP sparse solvers do not require any parameter tuning.

By making $\mathbf{A}$ equal to the user-item matrix we are performing item-based collaborative filtering. User-based collaborative filtering could be performed by simply using the transpose of $\mathbf{A}$. However, here we report results only for item-based collaborative filtering as it is the more common approach due to its improved scalability (Sarwar et al., 2001a; Ning and Karypis, 2011).

3.1. A Predictive Recommender

This section describes the approach used to implement a predictive recommender based on sparse dictionary coding. The input to the system is an array of ratings with missing values. An illustrative example is:

$$\mathbf{\Lambda} = \begin{array}{c} \\ U1 \\ U2 \\ U3 \end{array} \begin{array}{c} \begin{array}{cccccc} I1 & I2 & I3 & I4 & I5 & I6 \end{array} \\ \begin{pmatrix} ? & ? & 1 & 5 & ? & ? \\ 5 & 4 & ? & ? & ? & 1 \\ ? & 3 & ? & 1 & ? & 4 \end{pmatrix} \end{array}$$

Where the question marks indicate the missing ratings. The aim is to predict the values of missing ratings. For example, assume that we want to predict the first user's rating for item 2. The column of $\mathbf{\Lambda}$ corresponding to item 2 is removed and becomes $\mathbf{b}$ in equation 1. In addition, since a prediction is being made for the first user, all columns corresponding to items that the first user has not rated are also removed from the dictionary. This results in the matrix $\mathbf{A}$ that is substituted into equation 1:

$$\mathbf{A} = \begin{array}{c} \\ U1 \\ U2 \\ U3 \end{array} \begin{array}{c} \begin{array}{cc} I3 & I4 \end{array} \\ \begin{pmatrix} 1 & 5 \\ ? & ? \\ ? & 1 \end{pmatrix} \end{array}$$

The reason for removing columns corresponding to items that the user has not rated is that these items can not help to make predictions about that user's preferences. Our experiments have shown that removing columns corresponding to items that the user has not rated results in superior predictive accuracy across different datasets.

Missing values in $\mathbf{A}$ and $\mathbf{b}$ are set to zero and equation 1 is solved to find $\mathbf{x}$. These sparse coefficients can be used to calculate the predicted rating that user 1 will give to item 2, as:

$$\tilde{b}_{1,2} = \frac{\mathbf{a}_1 \mathbf{x}}{||\mathbf{x}||_1}, \tag{2}$$

where $\mathbf{a}_1$ is the first row of $\mathbf{A}$. Normalising by the sum of the coefficients is necessary to produce ratings that fall within the rating scale.

3.2. A Top-n Recommender

This section describes the approach used to implement a top-n recommender based on sparse dictionary coding. Here the aim is not to predict the rating a user would give to a specific item, but to generate a list of items predicted

to be most relevant for a given user. This list is essentially those items, that are currently not rated, which are predicted to have the highest ratings. The implementation is given by Algorithm 1.

Algorithm 1: Generating top-n lists of recommended items

Input: Dictionary $\mathbf{\Lambda} \in \mathbb{R}^{l \times m}$, and n
Output: Top-n list of recommended items for each user
foreach *Column* $\boldsymbol{a}_j$ *of* $\mathbf{\Lambda}$ **do**
  $\mathbf{b} = \mathbf{a}_j \in \mathbb{R}^{l \times 1}$;
  $\mathbf{A} = \mathbf{\Lambda}$;
  Remove the j^{th} column from $\mathbf{A}$, hence $\mathbf{A} \in \mathbb{R}^{l \times (m-1)}$;
  Solve the sparse coding problem $\min ||\mathbf{x}||_0$ subject to $\mathbf{b} = \mathbf{A}\mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^{(m-1) \times 1}$;
  Set j^{th} column of $\mathbf{X}$ equal to $[\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(j-1), 0, \mathbf{x}(j), \mathbf{x}(j+1), \ldots, \mathbf{x}(m-1)]$ where $\mathbf{X}$ is initially an empty matrix;
end
Reconstruct $\tilde{\mathbf{B}} = \mathbf{\Lambda}\mathbf{X}$, where $\tilde{\mathbf{B}} \in \mathbb{R}^{l \times m}$ is the reconstruction of all ratings;
foreach *User (Row)* i *in* $\tilde{\boldsymbol{B}}$ **do**
  Sort the i^{th} row of $\tilde{\mathbf{B}}$ in descending order.;
  From the sorted row, take the first n items that are not rated in the i^{th} row of the original $\mathbf{A}$, and recommend those n items to user i;
end

Algorithm 1 removes the j^{th} column from the dictionary $\mathbf{\Lambda}$ to form the matrix $\mathbf{A}$, just like the method used for the Predictive Recommender described in section 3.1. This avoids the trivial solution where $\mathbf{b}$ is reconstructed using $\mathbf{a}_j$ via a sparse vector $\mathbf{x}$ with a single non-zero element.The algorithm then places the sparse solution vector $\mathbf{x}$ in the sparse solution matrix $\mathbf{X}$, and places 0 in the j^{th} position in $\mathbf{x}$ for two reasons: i) to make the number of rows in $\mathbf{X}$ equal to the number of columns in $\mathbf{\Lambda}$ to enable the multiplication (reconstruction) $\mathbf{\Lambda}\mathbf{X}$ ii) to avoid using the column j^{th} of the dictionary $\mathbf{\Lambda}$ in the reconstruction $\tilde{\mathbf{B}} = \mathbf{\Lambda}\mathbf{X}$, that is, to avoid the trivial solution (using the j^{th} column to reconstruct itself).

Algorithm 1 can be implemented in a parallel or distributed computing environment. Specifically, the sparse representations for each item (column of $\mathbf{\Lambda}$) can be calculated in parallel (a naive way of doing this is to replicate the data on multiple computers and calculate sparse representations of different columns using these computers and/or multiple CPU cores). Similarly, the sorting of the rows in $\tilde{\mathbf{B}}$ can also be parallelised. It is also important to note that once the sparse representations have been computed (the first for loop in algorithm 1), then these can be effectively re-used many times. Therefore, one does not need to re-compute the first and the most costly step of the algorithm (finding a sparse representation for each item) every time. Finally, since $\mathbf{X}$ is a sparse representation of the user-item matrix $\mathbf{\Lambda}$, it will not be costly to store. For example, using the sparsity metric in (Hoyer, 2004), the sparsity of PFP and DIM algorithms are respectively 0.9725 and 0.9279 on the ML-100K dataset.

For Algorithm 1 to work for a given user, the algorithm requires that the user has rated at least one item, if the user has no ratings then the matrix row

corresponding to the user will be zeroes, and the reconstruction of the matrix (linearly combining columns) will also result in zeroes (no recommendations).

4. Results

The small (ML-100K) and medium (ML-1M) MovieLens movie datasets[1], and the three Jester joke datasets[2] were used to evaluate the performance of the proposed algorithms. The MovieLens-100K dataset contains 100,000 ratings from 943 users on 1682 items (movies). The ML-1M dataset contains about 1,000,209 ratings from 6040 users on approximately 3900 items. Finally, each of the three Jester datasets contains 100 items (jokes), and approximately 25000 users. The first, the second and the third Jester datasets contain approximately 1800000, 1700000 and 600000 ratings respectively.

In addition to the small and medium MovieLens movie datasets, some results on a large MovieLens dataset (ML-10M) are presented. The ML10M dataset contains 10 million ratings from 72000 users and 10000 items.

The MovieLens datasets and the Jester datasets have different characteristics. The first difference is the rating scale used in each dataset. While the ML datasets use a rating scale of 1-5 (inclusive), the Jester datasets use a rating scale of -10 to 10. The ratings in Jester datasets were rescaled to be between $0 < r \leq 5$. The second difference between the Jester and ML datasets is that the ratings in the ML dataset are discrete values (natural numbers), whereas the ratings in the Jester dataset are continuous values. This is due to the user interface employed to gather data from users (Goldberg et al., 2001). Finally, the Jester datasets are denser (higher $\frac{\text{ratings}}{\text{users} * \text{items}}$ rate). For example many users in the Jester datasets have rated all 100 items.

4.1. Predictive Accuracy Results

For all experiments, each dataset was split into two subsets: a training set, and a test set. For the MovieLens datasets, the test set contained 10 random ratings taken from each user, and the training set contained the rest of the ratings. This is because many other works also use 10 ratings in their test sets.(Sarwar et al., 2001a) (Sarwar et al., 2001b) (Deshpande and Karypis, 2004). For the three Jester datasets, 1 random rating taken from each user was put in the test set, and the training set contained the rest of the ratings. This is because the Jester dataset has relatively many users (about 25000 users) and we believe that evaluating 25000 predictions is sufficient evidence for the performance of the algorithms here. The implemented recommender systems were used to predict the ratings in a test set by using the corresponding training set.

The Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) metrics (Sarwar et al., 2001a) were used to assess predictive accuracy.

The results of the proposed, sparse, recommender systems are compared to those of the item-based k-NN recommender system. Two similarity measures were tested: Pearson correlation and cosine similarity. A range of values for the

[1] `http://grouplens.org/datasets/movielens/`
[2] `http://www.ieor.berkeley.edu/~goldberg/jester-data/`

Table 1. Comparison of Mean Absolute Error (MAE)

Dataset	Sparse (DIM)	Sparse (PFP)	k-NN (Pearson)	k-NN (Cosine)	MF
Jester 1	0.8187	**0.7999**	0.8164 (k=10)	0.8697 (k=15)	0.8464
Jester 2	0.8119	**0.7950**	0.8063 (k=10)	0.8566 (k=15)	0.8557
Jester 3	0.9056	0.9029	**0.8941** (k=10)	0.9263 (k=5)	0.9405
ML100K	0.7878	0.7780	**0.7657** (k=10)	0.7798 (k=10)	0.7678
ML1M	0.7536	0.7448	0.7536 (k=10)	0.8020 (k=10)	**0.7425**
Total:	4.0776	**4.0206**	4.0361	4.2516	4.1529

Table 2. Comparison of Root Mean Square Error (RMSE)

Dataset	Sparse (DIM)	Sparse (PFP)	k-NN (Pearson)	k-NN (Cosine)	MF
Jester 1	1.0429	**1.0264**	1.0624 (k=10)	1.1451 (k=15)	1.1458
Jester 2	1.0469	**1.0313**	1.0610 (k=10)	1.1354 (k=15)	1.1779
Jester 3	1.1413	**1.1406**	1.1451 (k=10)	1.1997 (k=5)	1.4675
ML100K	0.9998	0.9901	0.9875 (k=10)	1.0060 (k=10)	**0.9526**
ML1M	0.9485	0.9359	0.9485 (k=10)	1.0362 (k=10)	**0.8879**
Total:	5.1794	**5.1243**	5.2045	5.5224	5.6317

parameter k were tested ($5-30$, inclusive) for each dataset, and for each dataset we report results for the k that produced the best result on that dataset. Note that the results reported here for k-NN applied to the MovieLens dataset are similar to the best reported in the literature: an MAE of approximately 0.73 reported by (Sarwar et al., 2001a). The performance of the k-NN and MF algorithms is sensitive to the value of some parameters, and it is necessary to tune these parameters to obtain good performance on different datasets. An advantage of the proposed sparse-coding method is that no such parameter tuning is necessary.

Table 1 shows the MAE results, and table 2 shows the RMSE results, for all five different datasets. It can be seen that the proposed predictive recommender implemented using sparse dictionary coding (using either the DIM or PFP sparse solvers) is very competitive with the k-NN recommender and with the Matrix Factorisation (MF) method in (Koren et al., 2009). The parameter k was tuned for the k-NN recommender, and the λ regularisation parameter and the number of features were tuned for the MF recommender by using a subset of each training set.

We were able to evaluate the DIM predictive recommender using the large ML-10M dataset, as its implementation code supported sparse matrices, which enabled us to load large datasets into random-access memory. We created 4 test datasets from the large ML-10M dataset, ratings picked randomly without replacement. We ran 4 separate experiments on these datasets using the Sparse (DIM) predictive recommender. The Sparse (DIM) recommender achieved an average MAE of 0.7080 and an average RMSE of 0.9. These results are competitive with those in the literature, for example with the best MAE of about $0.72-0.73$ achieved in (Sarwar et al., 2001b) using a subset of the same dataset.

The proposed recommender system implemented using the PFP sparse solver

Table 3. Comparison of execution times (in seconds) for making 50 predictions

Dataset	Sparse (DIM)	Sparse (PFP)	k-NN (k=10, Pearson)	k-NN (k=10, Cosine)	MF
Jester-1	45.0	9.1	4.5	**3.5**	21.2
Jester-2	39.4	7.9	3.7	**3.4**	19.2
Jester-3	11.5	**3.2**	3.9	3.4	17.5
ML100K	1.2	**0.6**	3.7	3.0	11.8
ML1M	27.8	**10.8**	41.1	35.5	165.4
Total:	124.8	**31.6**	56.9	48.8	223.7

also displays a shorter execution time than an item-based k-NN recommender. Table 3 shows the execution times of the sparse coding based and k-NN recommender systems for making 50 (arbitrarily selected) rating predictions. Each recommender system made predictions on the same 50 randomly picked items. All algorithms were implemented in MATLAB and executed on a personal computer with an Intel Core i7 processor running at 2.8 GHz, and with 8 GB DDR3 (1333 MHz) RAM. Our experiments show that the time it takes for MF and k-NN algorithms to finish processing grows faster than the time it takes for the proposed predictive recommendation algorithms based on sparse dictionary coding. This can be explained by the fact that unlike k-NN and MF algorithms, the proposed algorithm, before predicting a rating for a given user, removes all the items that are not rated by the user from the user-item matrix before processing the matrix for prediction. This dramatically reduces the amount of data to be processed. More formally, let us denote the number of users and the number of items in the original dataset as $|U|$ and $|I|$ respectively. While the k-NN and MF algorithms take an input of size $|U| * |I|$, the proposed algorithm takes an input of size $|U| * |I^u|$ where $|I^u|$ is the number of items that the user u has rated. It is obvious that $|I^u|$ is necessarily much smaller than $|I|$ and we believe this justifies the execution time growth shown in Table 3, specifically the execution time differences using the ML100K and ML1M datasets.

We also made an attempt to compare our results to those by (Szabó et al., 2012). (Szabó et al., 2012) claim an RMSE of around 4.07 using the Jester dataset as one dataset rather than using the 3 subsets of it as we did in this paper. We merged all three Jester training sets and all three Jester tests sets, as a result we had three ratings for each user in the resulting test set, and the rest of the ratings were in the training set. We scaled the ratings from the range 0-5 to the range 0-20 to be able to compare our results to those by (Szabó et al., 2012). The best RMSE we achieved using the proposed sparse coding algorithm based on DIM was 4.57 (using the rating range 0-20). However this comparison is limited as the code for the algorithm by (Szabó et al., 2012) is not shared in their paper (e.g. via a web link), and it is likely that our datasets are not exactly the same. A better comparison would first reproduce the results by (Szabó et al., 2012) using their implementation code, and then it would produce results for the proposed algorithms using the same datasets exactly.

Table 4. Correctly Placed Pairs (CPP) results

Dataset	Sparse (DIM)	Sparse (PFP)	k-NN (Pearson)	k-NN (Cosine)	MaxF	SLIM
Jester-1	0.8918	0.8833	0.6017	0.8753	0.8827	**0.9444**
Jester-2	0.8919	0.8835	0.6033	0.8768	0.8835	**0.9810**
Jester-3	0.8370	0.8335	0.7876	0.8265	0.8269	**0.9335**
ML100K	0.8883	0.8806	0.8511	0.8650	0.8469	**0.9612**
ML1M	0.9098	0.8265	0.8720	0.8892	0.8533	**0.9390**
Average:	**0.8838**	0.8615	0.7431	0.8665	0.8587	**0.9518**

4.2. Top-n Recommendation Results

The proposed top-n recommender was evaluated using the methodologies described in (Fouss et al., 2005; Cooper et al., 2014; Sarwar et al., 2000). The top-n recommendation metrics used in this research are concerned with a simple question: what is the utility of a list of recommended items to a user?

Correctly Placed Pairs (CPP) is a measure of top-n recommendation quality (Cooper et al., 2014). Intuitively, CPP checks if a top-n list of recommended items for a user ranks the most relevant items higher in the list than less relevant items. Formally, CPP is defined as follows (Cooper et al., 2014):

$$\frac{|\{\langle m', m''\rangle : m' \in I_T^u, m'' \in I \setminus I^u, m' <_u m''\}|}{t_u(q - i_u)} \tag{3}$$

where I_T^u is the set of all test items rated by user u in the test set, I is the set of all items, I^u is the set of all items rated by user u, $q = |I|$ is the number of all items, t_u is the number of items rated by user u in the test set, and i_u is the number of all items (in both sets) rated by user u. The numerator checks each pair of items $\langle m', m''\rangle$ and increments a counter if $m' <_u m''$ where the operator $<_u$ indicates that the left operand is ranked higher than the right operand in the list of recommended items generated for user u. Equation 3 is run for every top-n list of items generated for each user, then the average CPP is calculated over all users. An algorithm that recommends random items would produce a CPP of 0.5 (Fouss et al., 2005).

Table 4 shows the CPP results on the Jester and MovieLens datasets. For comparison we reproduced the CPP result of the *MaxF* algorithm (Fouss et al., 2005). The proposed algorithms based on sparse coding produce a better performance than the MaxF algorithm and the item-based k-NN recommender in terms of CPP. Furthermore, the CPP results of the sparse coding based recommenders are competitive with those of the best graphical algorithms in (Cooper et al., 2014) and in (Fouss et al., 2005). The best CPP results these graphical algorithms achieve are 0.9099 and 0.8962 on ML100K and ML1M respectively. The SLIM recommender had never been evaluated using the CPP metric. Our results suggest that the SLIM recommender show better CPP performance than other recommenders tested. For the results shown here, we tuned the parameters of the SLIM recommender for each dataset. The proposed methods based on PFP and DIM did not require any parameter tuning.

Recall is one of the most popular metrics for evaluating recommender systems (Herlocker et al., 2004).

Table 5. Hit-rate (N=10)

Dataset	Sparse (DIM)	Sparse (PFP)	k-NN (Pearson)	k-NN (Cosine)	MaxF	SLIM
Jester 1	0.9670	0.9611	0.3435	0.9485	0.9675	**0.9685**
Jester 2	0.9683	0.9617	0.3584	0.9481	0.9684	**0.9698**
Jester 3	**0.8241**	0.8188	0.6231	0.8197	0.8206	0.8235
ML100K	0.2879	0.2927	0.1253	0.1389	0.1156	**0.3171**
ML1M	0.2331	0.1772	0.0829	0.0987	0.0825	**0.2523**
Total	3.2804	3.2115	1.5332	2.9539	2.8721	**3.3312**

Table 6. Personalisation (N=10)

Dataset	Sparse (DIM)	Sparse (PFP)	k-NN (Pearson)	MaxF	SLIM
Jester-1	0.5279	0.6132	0.5547	0	**0.8411**
Jester-2	0.5556	0.6448	0.5576	0	**0.9310**
Jester-3	0.4343	0.4475	0.2799	0	**0.6715**
ML100K	0.9724	0.8883	**0.9784**	0	0.9072
ML1M	0.9194	0.9057	0.9176	0	**0.9561**
Average:	0.6819	0.6999	0.6576	0	**0.8614**

Formally, the recall metric is defined as follows:

$$\text{Recall}_u = \frac{|\text{test}_u \cap \text{topN}_u|}{|\text{test}_u|}$$

where test_u is the set of items rated by user u in the test set, topN_u is the top-n recommendation list generated for user u, and N is the number of items recommended.

The hit-rate metric (Ning and Karypis, 2011) is a special case of recall where the test set contains only 1 rating for each user. Formally,

$$\text{Hit-rate} = \frac{\sum_{u \in U} |\text{test}_u \cap \text{topN}_u|}{|U|}$$

where U is the set of all users, test_u is the set of items rated by the user u in the test set, $|\text{test}_u| = 1$ for each user, and topN_u is the list of items recommended for user u. Table 5 compares the hit-rate performance of the implemented algorithms. The hit-rate of Sparse (DIM) in table 5 is competitive with the best hit-rates in (Ning and Karypis, 2011) on the same dataset.

The personalisation metric is defined as follows (Zhou et al., 2010):

$$h_{ij}(N) = 1 - \frac{q_{ij}(N)}{N} \tag{4}$$

where $q_{ij}(N)$ is the number of common items in the top N places of the lists of recommended products of user i and user j. After calculating h_{ij} for each pair of users, the average h_{ij} is calculated as the overall personalisation. For the results presented here, 10 recommendations were made for each user.

Table 6 presents the personalisation results of the same algorithms and datasets tested in table 4. It can be seen that the MaxF method has zero personalisation

since it recommends the same most frequently rated items to every user. Hence, although a simple algorithm such as MaxF may perform well in metrics such as CPP, it does not make personalised recommendations, and thus, its utility will be low. On the other hand, simply making random recommendations can increase personalisation. Hence, personalisation alone is not a good indicator of the utility of a recommender system. A good top-n recommender should produce high scores for both personalisation and quality metrics such a CPP and hit-rate. This is achieved by SLIM as well as by the proposed recommender system.

The SLIM implementation is written in C and the proposed top-n recommendation method in this paper in MATLAB so it is difficult to compare their execution times, and as a result we make no attempts to make this comparison.

5. Conclusions

The evidence from this research suggests that sparse coding can be effectively used for both predictive recommenders and top-n recommenders. The proposed predictive recommender makes predictions that have similar accuracy to k-NN and matrix-factorisation based methods, but it is faster and requires no parameter tuning. The top-n recommenders based on sparse coding produce more accurate predictions than other methods such as k-NN, and are competitive with state-of-the-art top-n recommendation algorithms such as SLIM in terms of the hit-rate metric, although we showed in this paper that SLIM produces more personalised recommendations. However, unlike SLIM, the proposed top-n recommendation methods using DIM and PFP sparse coding algorithms require no parameter tuning. Finally, the proposed top-n and predictive recommendation algorithms can be run in a distributed or a parallel computing environment, making it scalable to practical applications.

References

Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems*, 46:109–132.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bruckstein, A., Donoho, D., and Elad, M. (2009). From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM Rev.*, 51(1):34–81.

Cooper, C., Lee, S., Radzik, T., and Siantos, Y. (2014). Random walks in recommender systems: Exact computation and simulations. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, WWW Companion '14, pages 811–816, Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Deshpande, M. and Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177.

Elad, M. (2010). *Sparse and redundant representations: from theory to applications in signal and image processing.* Springer.

Fouss, F., Pirotte, A., and Saerens, M. (2005). A novel way of computing similarities between nodes of a graph, with application to collaborative recommendation. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pages 550–556.

Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr.*, 4(2):133–151.

Herlocker, J., Konstan, J., Terveen, L., and Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53.

Hoyer, P. (2004). Non-negative matrix factorization with sparseness constraints. *The Journal of Machine Learning Research*, 5:1457–1469.

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

Ning, X. and Karypis, G. (2011). Slim: Sparse linear methods for top-n recommender systems. In *Data Mining (ICDM), 2011 IEEE 11th International Conference*, pages 497–506.

Plumbley, M. (2006). Recovery of sparse representations by polytope faces pursuit. In *Independent Component Analysis and Blind Signal Separation*, pages 206–213. Springer.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000). Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001a). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA. ACM.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001b). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM.

Spratling, M. (2014). Classification using sparse representations: a biologically plausible approach. *Biological Cybernetics*, 108(1):61–73.

Szabó, Z., Póczos, B., and Lőrincz, A. (2012). Collaborative filtering via group-structured dictionary learning. In *Latent Variable Analysis and Signal Separation*, volume 7191 of *Lecture Notes in Computer Science*, pages 247–254. Springer Berlin Heidelberg.

Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T., and Yan, S. (2010). Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, 98(6):1031–1044.

Zhou, T., Kuscsik, Z., Liu, J., Medo, M., Wakeling, J., and Zhang, Y. (2010). Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515.