A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems

Ali Elkahky[*]
Department of Computer Science
Columbia University
New York, NY 10027, USA
ame2154@columbia.edu

Yang Song, Xiaodong He
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
{yangsong,xiaohe}@microsoft.com

ABSTRACT

Recent online services rely heavily on automatic personalization to recommend relevant content to a large number of users. This requires systems to scale promptly to accommodate the stream of new users visiting the online services for the first time. In this work, we propose a content-based recommendation system to address both the recommendation quality and the system scalability. We propose to use a rich feature set to represent users, according to their web browsing history and search queries. We use a Deep Learning approach to map users and items to a latent space where the similarity between users and their preferred items is maximized. We extend the model to jointly learn from features of items from different domains and user features by introducing a multi-view Deep Learning model. We show how to make this rich-feature based user representation scalable by reducing the dimension of the inputs and the amount of training data. The rich user feature representation allows the model to learn relevant user behavior patterns and give useful recommendations for users who do not have any interaction with the service, given that they have adequate search and browsing history. The combination of different domains into a single model for learning helps improve the recommendation quality across all the domains, as well as having a more compact and a semantically richer user latent feature vector. We experiment with our approach on three real-world recommendation systems acquired from different sources of Microsoft products: Windows Apps recommendation, News recommendation, and Movie/TV recommendation. Results indicate that our approach is significantly better than the state-of-the-art algorithms (up to 49% enhancement on existing users and 115% enhancement on new users). In addition, experiments on a publicly open data set also indicate the superiority of our method in comparison with transitional generative topic models, for modeling cross-domain recommender systems. Scalability analysis show that our multi-view DNN model can easily scale to encompass millions of users and billions of item entries. Experimental results also confirm that combining features from all domains produces much better performance than building separate models for each domain.

[*]Work done at Microsoft Research during the author's summer internship.

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741667.

General Terms

User Modeling

Keywords

User Modeling; Recommendation System; Multi-View Learning; Deep Learning

1. INTRODUCTION

Recommendation systems and content personalization play increasingly important role in modern online web services. Many recent web services work on finding the most relevant content to their users in order to maximize the engagement with the site and minimize the time for finding the relevant content. A major approach to this task is called Collaborative Filtering (CF) [3, 19, 22, 21, 23, 6], which uses user's previous history of interaction in the web site to predict the most relevant content for recommendation. Another common approach is content-based recommendation [14, 15], which uses features about items and/or users to recommend new items to users based on the similarity between features. While both approaches work well in many practical applications [6], they usually face certain limitations and challenges especially with the increasing demand of personalization and recommendation quality.

Specifically, CF needs a considerable amount of previous history of interaction with the site before it can give high quality recommendation. This problem is known as user cold start problem [24]. In a newly established online service, the problem becomes more severe since users have little or no history of interaction with the site. Therefore, traditional CF methods often fail to produce high quality recommendation for new users. On the other hand, content-based recommendation methods extract features from each user and/or item and use those features to make recommendations. For example, if two News articles N_i and N_j share the same topic and a user liked article N_i, the system may recommend article N_j to the user. Similarly, if two users U_i and U_j share certain similarities like their location, age or gender, the system may recommend to user U_j the items that user U_i liked before. In practice, research has shown that content-based approach can handle the cold start problem for new items well [32]. However, its effectiveness is questionable when applying to recommendations for new users since user-level features are usually more difficult to acquire and often generated from a limited information in user online profiles, which failed to accurately capture the actual user's interests.

To address these limitations, we propose a recommendation system that leverages both user and item features. To build user features, unlike many user profile-based approaches, we propose to extract rich features from user's browsing and search histories to model user's interests. The underlying assumption is that, users' historical online activities reflect a lot about user's background and preference, and therefore provide a precise insight of what items and topics users might be interested in. For example, a user with many infant-related queries and website visits (e.g., toysrus.com) may suggest that she is a mom for a new-born baby. With these abundantly available user online activities, recommending relevant items can be achieved more efficiently and effectively.

In our work we propose a novel deep learning approach extended from the Deep Structured Semantic Models (DSSM) [9] to map users and items to a shared semantic space and recommend items that have maximum similarity with users in the mapped space. To achieve this, our model projects the users and items, each of which is represented by a rich feature set, through non-linear transformation layer(s) to a compact shared latent semantic space where the similarity between the mapping of the user and mappings of items liked by the user is maximized. This allows the model to learn interesting mappings such as people who visited *fifa.com* will like to read News articles about the *World Cup* and play *soccer games* on PC or Xbox. The rich features in the user side enable modeling the user's behavior and thus overcome many limitations in the content-based recommendation. It also addresses the user cold start problem effectively since the model allows us to capture user interests from queries and recommend related items (say music) even if they do not have any history on using music services. Our deep learning model has a ranking based objective which aims at ranking positive examples (items that users like) higher than negative examples. This ranking based objective has shown to be better for recommendation systems [9].

Furthermore, we extend the original DSSM model, which is referred to as *single-view* DNN in this paper since it learns from user features and items that come from a single domain, to jointly learn features of items from different domains. We name the new model *Multi-View* Deep Neural Network (MV-DNN). In literature, multi-view learning is a well-studied area which learns from data that do not share common feature space [27]. We consider MV-DNN as a general Deep learning approach in the multi-view learning setup. Specifically, in our data sets with News, Apps and Movie/TV logs, instead of building separate models for each of the domain that naively maps the user features to item features within the domain, we build a novel multi-view model that discovers a single mapping for user features in the latent space such that it is jointly optimized with features of items from all domains. MV-DNN allows us to learn a better user representation that leverages more data across domains and tackles the data sparsity problem in a principled way by leveraging user preference data from all domains. We show in our experiments that this multi-view extension improves the recommendation quality across all the domains at the same time. In addition, it is worth mentioning that the non-linear mapping in the deep learning models enables us to find a compact representation of users in the latent space which makes it much easier to store the learnt user mapping and share the information among different tasks.

One challenge in using deep learning to model rich user features is the high dimension of the feature space which makes the learning inefficient and may impact the generalization ability of the model. We propose several effective and scalable dimensionality reduction techniques that reduce the dimension to a reasonable size without the loss of much information.

To summarize, the contributions in this work are: (1) use rich user features to build a general-purpose recommendation system, (2) propose a deep learning approach for content-based recommendation systems and study different techniques to scale-up the system, (3) introduce the novel Multi-View Deep learning model to build recommendation systems by combining data sets from multiple domains, (4) address the user cold start issue which is not well-studied in literature by leveraging the semantic feature mapping learnt from the multi-view DNN model, and (5) perform rigorous experiments using four real-world large-scale data set and show the effectiveness of the proposed system over the state-of-the-art methods by a significantly large margin.

The rest of this paper is organized as following, first we review major approaches in recommendation systems including papers that focus on the cold start problem in Section 2; in Section 3, we describe the data sets we work with and detail the type of features we use to model the user and the items in each domain, respectively. We then review the basic DSSM model and discuss how it could be extended for our setting in Section 4; in Section 5, we introduce the multi-view deep learning model in details and discuss its advantages; in Section 6, we discuss the dimension reduction methods to scale-up the model; in Section 7, 8, 9 & 10, we present a comprehensive empirical study; we finally conclude in Section 11 and suggest several future work.

2. RELATED WORK

There has been extensive study on recommendation systems with a myriad of publications. In this section, we aim at reviewing a representative set of approaches that are mostly related to our proposed approach.

In general, recommendation systems can be divided into collaborative recommendation and content based recommendation. Collaborative Recommendation systems recommend an item to a user if similar users liked this item. Examples of this technique include nearest neighbor modeling [3], Matrix Completion [19], Restricted Boltzmann machine [22], Bayesian matrix factorization [21] etc. Essentially, these approaches are user collaborative filtering, item collaborative filtering or both item and user collaborative filtering. In user collaborative filtering such as [3], the algorithm computes the similarity between users based on items they liked. Then, the scores of user-item pairs are computed by combining scores of this item given by similar users. Item based collaborative filtering [23], computes similarity between items based on users who like both items, then recommend the user items similar to the ones she liked before. User-item based collaborative filtering finds a common space for items and users based on user-item matrix and combines the item and user representation to find a recommendation. All matrix factorization approaches like [19] and [21] are examples of this technique. CF can be extended to large-scale setups like in [6]. However, CF is generally unable to handle new users and new items, a problem which is often referred to as cold-start issue.

The second approach for recommendation systems is content-based recommendation. This approach extracts features from item's and/or user's profile and recommend items to users according to these features. The underlying assumption is that similar users tend to like items similar to the items they liked previously. In [14], a method is proposed to construct a search query with some features of items the user liked before to find other relevant items to recommend. Another example is presented in [15] where each user is modeled by a distribution over News topics that is constructed from articles she liked with a prior distribution of topic preference computed using all users who share the same location. This approach can handle new items (News articles) but for new users the system used location feature only which implies that new users are expected to see most frequent topics in their location. This might be a good features to recommend News but in other domains, for example Apps recommendation, using only location information may not work as a good prior over user's preferences.

Recently, researchers have developed approaches that combine both collaborative recommendation and content based recommendation. In [16], the author used item features to smooth user data before using collaborative filtering. In [7], the authors used Restricted Boltzmann Machine to learn similarity between items, and then combined this with collaborative filtering. A Bayesian approach was developed in [32] to jointly learn the distribution of items, research papers in their case, over different components (topics) and the factorization of the rating matrix.

Handling the cold start issue in recommendation systems is studied mainly for new items (items that have no rating by any user). As we mentioned before, all content based filtering can handle cold start for item, and there are some methods that were developed and evaluated specifically for this issue like in [24] and [7]. The work in [18] studied how to learn user preferences for new users incrementally by recommending items that give the most information about user preferences while minimizing the probability of recommending irrelevant content. User modeling via rich features have been studied a lot recently. For example, it has been shown that user search queries can be used to discover the similarities between users [25]. Rich features from user search history has also been used for personalized web search [26]. For recommendation systems, the authors in [2] leveraged the user's historical search queries to build personalized taxonomy for recommending Ads. On the other hand, researchers have discovered that a user's social behaviors can also be used to build the profile of the user. In [1], the authors used user's tweets in Twitter data to recommend News articles.

Most traditional recommendation system research focused on data within a single domain. Recently, there has been an increasing interest in cross domain recommendation. There are different approaches for addressing cross domain recommendation. One approach is to assume that different domains share similar set of users but not the items, as illustrated in [20]. In their work, the authors augmented data from rating of movies and books from datasets that have common users. The augmented data set was then used to perform collaborative filtering. They showed that this in particular helped the cases where users with little profile information in one of the domains (cold-start users). The second approach addressed the scenarios where the same set of items shared different types of feedbacks in different domains like user clicks or user explicit rating. As shown in [17], the authors introduced a coordinate system transfer method for cross domain matrix factorization. In [12], the authors studied the cross domain recommendation in the case where there existed no shared users or items between domains. They developed a generative model to discover common clusters between different domains. However, a challenge in their approach is its ability to scale beyond medium datasets due to the computational cost. A different approach was introduce in [28] for author collaboration recommendation where they built a topic model to recommend authors to collaborate from different research fields.

For many approaches in recommendation systems the objective function is to minimize the root mean squared error on the user-item matrix reconstruction. Recently, ranking based objective function has shown to be more effective in giving better recommendation as shown in [11].

Deep learning has recently been proposed for building recommendation systems for both collaborative and content based approaches. In [22], an RBM model was used for collaborative filtering. Deep learning for content based recommendation has been done for example in [30] where deep learning was applied to learn embedding for music features. This embedding was then used to regularize matrix factorization in collaborative filtering.

3. DESCRIPTION OF THE DATA SETS

In this section introduces the data sets. We describe the data collection process and the feature representations for each data set, as well as some basic statistics of the data.

The four data sets used in this study were collected from user logs of several Microsoft products, including (1) Search engine logs from Bing Web vertical, (2) News article browsing history from Bing News vertical, (3) App download logs from Windows AppStore, and (4) Movie/TV view logs from Xbox. All the logs were collected between December 2013 and June 2014, with primary focus on English-speaking markets including United States, Canada and Great Britain.

(**User Features**) We collected users' search queries and their clicked URLs from Bing to form user features. Queries were first normalized, stemmed and then split into unigram features and URLs were shorten into domain-level only (e.g., www.linkedin.com) to reduce the feature dimension. We then used TF-IDF scores to keep only the most popular and non-trivial features. Overall, we selected 3 million unigram features and 500K domain features, leading to a total length of 3.5-million user feature vector.

(**News Features**) We collected news article clicks from Bing News vertical. Each News item is represented by three parts of features. The first part is the title features encoded using letter tri-gram representation as we will describe in the next section. Secondly, the top-level category of each News (e.g., Entertainment) is encoded as binary features. Finally, the Named Entities in each article, extracted using

Type	DataSet	UserCnt	Feature Size	Joint Users
User View	Search	20M	3.5M	/
Item View	News	5M	100K	1.5M
	Apps	1M	50K	210K
	Movie/TV	60K	50K	16K

Table 1: Statistics of the four data sets used in this paper. The *Joint Users* column indicates the number of common users between each item view and the user view.

an internal proprietary NLP parser, is encoded using letter tri-gram as well. This results in a 100K feature vector.

(**App Features**) Users' App download histories were collected from Windows AppStore logs. The title of each App is represented using letter tri-gram, combining with its category (e.g., Game) features in the binary format. Do to the nature of constant change of App descriptions, we decided not to include that as part of the feature space. This results in a 50K feature vector for Apps.

(**Movie/TV Features**) From Xbox logs, we collected the Movie/TV view history for each Xbox user. The title and description of each item were combined into text features and then encoded using letter tri-gram. The genre is also used as binary features. This results in a 50K feature vector for Movie/TV.

In our neural network framework, user features are mapped into the user view and the rest is mapped into different item views. For training purpose, each user view is matched with an item view which contains the exact set of users. To achieve this, we sub-sampled logged-in users (i.e., users with uniquely annonymized and hashed Microsoft User IDs) from each user-item view pair and performed inner join based on their IDs. This results in different number of users for each user-item view pair. Table 1 depicts some basic statistics of the data used in this paper.

4. DSSM FOR USER MODELING IN RECOMMENDATION SYSTEMS

A deep structured semantic model (DSSM) is introduced in [9] to enhance query document matching in the web search context. Given the close relationship to our proposed multi-view deep neural network, we briefly review DSSM here.

The typical architecture of the DSSM is shown in Fig. 1.The input (raw text features) to the DNN is a high dimensional term vector, e.g., raw counts of terms in a query or a document without normalization. Then the DSSM passes its input through two neural networks, one for each of the two different inputs, respectively, and maps them into semantic vectors in a shared semantic space. For Web document ranking, DSSM computes the relevance score between a query and a document as the cosine similarity of their corresponding semantic vectors, and ranks documents by their similarity scores to the query.

More formally, if we denote x as the input term vector, y as the output vector, $l_i, i = 1, ..., N-1$, as the intermediate hidden layers, W_i as the i-th weight matrix, and b_i as the i-th bias term, we have

$$\begin{aligned} l_1 &= W_1 x \\ l_i &= f(W_i l_{i-1} + b_i), i = 2, ..., N-1 \\ y &= f(W_N l_{N-1} + b_N) \end{aligned} \tag{1}$$

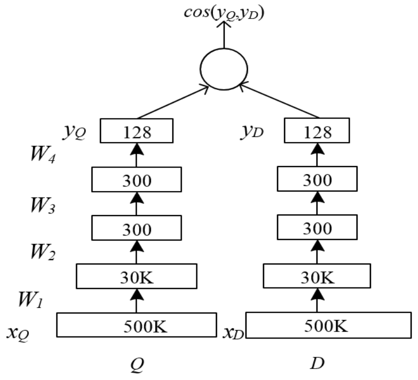

Figure 1: The illustration of the deep structured semantic model (DSSM).

where we use the *tanh* function as the activation function at the output layer and the hidden layers $l_i, i = 2, ..., N-1$:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2}$$

The semantic relevance score between a query Q and a document D is then measured as:

$$R(Q, D) = cosine(y_Q, y_D) = \frac{{y_Q}^T y_D}{||y_Q|| \cdot ||y_D||} \tag{3}$$

where y_Q and y_D are the semantic vectors of the query and the document, respectively. In Web search, given the query, the documents are sorted by their semantic relevance scores.

Conventionally, each word w is represented by a one-hot word vector where the dimensionality of the vector is the size of the vocabulary. However, the vocabulary size is often very large in real-world Web search tasks, and the one-hot vector word representation makes model learning very expensive. Therefore, the DSSM uses a *word hashing* layer to represent a word by a letter-trigram vector. For example, given a word (e.g. *web*), after adding word boundary symbols (e.g. *#web#*), the word is segmented into a sequence of letter-n-grams (e.g. letter-tri-grams: *#-w-e, w-e-b, e-b-#*). Then, the word is represented as a count vector of letter-tri-grams. For example, the letter-trigram representation of *web* is:

In Figure 1, the first layer matrix W_1 denotes the letter-trigram matrix transforming from a term-vector to its letter-trigram count vector, which requires no learning. Even though the total number of English words may grow to be extremely large, the total number of distinct letter-trigrams in English (or other similar languages) is often limited. Therefore, it can generalize to new words unseen in the training data.

In training, it is assumed that a query is relevant to the documents that are clicked on for that query, and the parameters of the DSSM, i.e., the weight matrix W_i, are trained using this signal. I.e., first the posterior probability of a document given a query is estimated from the semantic relevance score between them through a softmax function

$$P(D|Q) = \frac{exp(\gamma R(Q, D))}{\sum_{D' \in \mathbf{D}} exp(\gamma R(Q, D'))} \tag{4}$$

where γ is a smoothing factor in the softmax function, which is usually set empirically on a held-out data set in our experiment. **D** denotes the set of candidate documents to be ranked. Ideally, **D** should contain all possible documents. In practice, for each (query, clicked-document) pair, denoted by (Q, D^+) where Q is a query and D^+ is the clicked document, we approximate **D** by including D^+ and N randomly selected unclicked documents, denote by $\{D_j^-; j = 1,, N\}$.

In training, the model parameters are estimated to maximize the likelihood of the clicked documents given the queries across the training set.

$$L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q) \tag{5}$$

where Λ denotes the parameter set of the neural networks.

5. MULTI-VIEW DEEP NEURAL NETWORK

The DSSM can be viewed as a multi-learning framework where it maps two different views of the data into a shared view. In that sense, it can be viewed in a more general setting to learn a shared mapping between two different views.

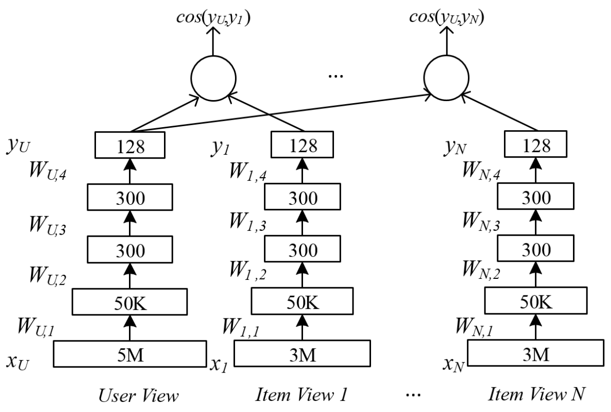

Figure 2: Multi-view DNN for multiple domain recommendation. It uses a DNN to map high-dimensional sparse features (e.g., raw features of users, News, App) into low-dimensional dense features in a joint semantic space.. The first hidden layer, with 50k units, accomplishes word hashing. The word-hashed features are then projected through multiple layers of non-linear projections. The final layer's neural activities in this DNN form the feature in the semantic space. Note that the input feature dimension x (5M and 3M) in this figure is hypothetical as in practice each view can have arbitrary number of features. See text for details.

In this work, we propose an extension to the DSSM where we have more than two views of the data and we call this Multi-view DNN (MV-DNN). In this setting, we have $v + 1$ views, one pivot view called X_u and other v auxiliary views X_1 through X_v and each X_i has its own input domain $X_i \in R^{d_i}$. Each view also has its own non-linear mapping layer(s) $f_i(X_i, W_i)$ which transforms X_i into shared semantic space Y_i. The training data contains a set of samples. The j^{th} sample has an instance of the pivot view $X_{u,j}$ and one active auxiliary view $X_{a,j}$ where a is the index of the active view in sample j. All other views input $X_{i:i \neq a}$ are set to 0 vectors. The objective of is finding a non-linear mapping for each view such that the sum of similarities, in the semantic space between mapping of the pivot view Y_u and mappings of all

Algorithm 1 Training Multi-View DNN

```
 1: Input: $N$ = # of view pairs, $M$ = # of training iterations,
         $U_A$ = user view architecture,
         $I_A = \{I_{A1}, ...I_{AN}\}$ item view architecture,
         $U_D = \{U_{D1}, ...UDN\}$ user input files,
         $I_D = \{I_{D1}, ...I_{DN}\}$ item input files,
         $\mathbf{W_U}$ = user view weight matrix,
         $\mathbf{W_I} = \{W_{I1}, ...W_{IN}\}$ item view weight matrices
 2: Initialization
 3: Initialize $\mathbf{W_U}$ and $\mathbf{W_I}$ using $U_A$ and $I_A$
 4: for $m = 1$ to $M$
 5:   for $v = 1$ to $N$
 6:     $T_U \leftarrow U_{Dv}$
 7:     $T_I \leftarrow I_{Dv}$
 8:     train $\mathbf{W_U}$ and $\mathbf{W_I}$ using $T_U$ and $T_I$
 9:   end for
10: end for
11: Output: $\mathbf{W_U}$ = final user weight matrix,
           $\mathbf{W_I}$ = final set of item view weight matrices
```

other views $Y_1, ...Y_v$ is maximized. Formally, we have:

$$p = \arg \max_{W_u, W_1, ..W_v} \sum_{j=1}^{N} \frac{e^{\alpha_a \cos(Y_u, Y_{a,j})}}{\sum_{X' \in R^{d_a}} e^{\alpha \cos(Y_u, f_a(X', W_a))}} \tag{6}$$

The architecture of MV-DNN is shown in figure 2. In our recommendation systems setup, we set the pivot view X_u to user features and create auxiliary view for each different type of items we aim to recommend.

The intuition for having this objective function is to try to find a single mapping for user's features, namely W_u, that can transform users features into a space that matches all different items the user liked in different views/domains. This way of sharing parameters allows the domains that do not have enough information to learn good mapping through other domains which have more data. It should work well if the assumption that users who have similar News articles taste also have similar taste in other domains which means those domains can benefit from user mapping learned by News domain. If this assumption is valid, then samples from any domain will help in grouping similar users more accurately in all domains. The empirical results showed that the assumption is reasonable in the domains we experimented on, which we elaborate more in the experiment sections.

5.1 Training MV-DNN

MV-DNN can be trained using Stochastic Gradient Decent (SGD). In practice, each training example contains a pairs of inputs, one for the user view and one for the data views. Therefore, despite the fact that there exists only one user view in our model, it is often more convenient to have N user feature files, each of which corresponds to an item feature file, where N is the total number of user-item view pairs. In Algorithm 1, we sketch the high-level process of training MV-DNN. When taking the derivative with respect to all $W_i \in \{W_u, W_1, ..W_v\}$ we end up with only two non-zero derivatives $\frac{\partial p}{\partial W_u}$ and $\frac{\partial p}{\partial W_a}$ which allow us to apply the same update rules for DSSM as in [9] substituting q by X_u and d by X_a.

5.2 Advantages of MV-DNN

Although MV-DNN is extended from the original DSSM framework, it has several unique characteristics that makes it superior than its predecessor. First of all, the original DSSM model uses for both the query view and document view the same size of feature dimensions, and preprocessed using the same representation (letter tri-gram for example). This poses a huge restrictions in the feature composition step. Due to the heterogeneity nature of recommendation systems, it is quite likely that the user view and item view pose different input features. Also, many types of features cannot be optimally represented using letter trigram. For example, the URL domain feature often contains prefix and suffix like www, com, org which will be mapping to the same feature if letter tri-gram is applied. In practice, we have discovered that the letter tri-gram representation works ideally in the case that the input raw text is short (e.g., query text and document title in the original DSSM model), but becomes inappropriate to model user level features which often contain a large collection of queries and URL domains. By removing this constraint, the new MV-DNN model can incorporate categorical features such as movie genre and app category, geospatial features such as country and region, as well as raw text features represented using uni-grams or bi-grams from user inputs.

Secondly, MV-DNN has the capability of scaling up to many different domains which the original DSSM framework is unable to. By performing pair-wise training between each user-item view pair as described in Algorithm 1, our model can easily adopt new sets of view pairs which may contain completely independent sets of users and items at any stage of the training process, e.g., adding a new data set from Xbox games. By alternating user-view pairs during each training iteration, our model can eventually converge to an optimal embedding of user view that is trained through all item views. Note that although we can have different user sets in different item views in theory, during our experiment, we choose to keep the same set of users across all views with the consideration of both convenience and the ease of feature normalization.

6. DIMENSION AND DATA REDUCTION

In practice, the proposed deep learning approach often needs to handle a huge amount of training examples in high dimensional feature spaces for the user view. In order to scale the system up, we propose several dimensionality reduction techniques to reduce the number of features in the user view. Then, we propose an idea to compact and summarize user training examples which reduces the number of training data to be linear to the number of users.

6.1 Top Features

One simple dimensionality reduction approach for user features is to select the top-K most frequent features. We select all features that have a probability of 0.001 or greater to be picked by a user. The main underlying hypothesis is that users can be described well using a relatively small set of frequent features that explains the users' common online behavior. Note that as described in Section 3, the user's raw features are preprocessed using TF-IDF scores so that the top features we select no longer contains common stop words in search queries. This resulted in 83K user features.

6.2 K-means

K-means [8] is a well-known clustering technique aims at creating a number of clusters such that the sum of distances between each point and its nearest cluster is minimized:

$$\arg\min_{C_1,..C_k} \sum_{i=1}^{N} \min_{C_j \in \{C_1,..C_k\}} distance(X_i, C_j), \tag{7}$$

Where X_i is a data point and C_j indicates a cluster centroid. K-means has shown good performance as a way to learn unsupervised features in vision [5]. The basic idea is to group similar features together into one cluster and generate a new feature vector Y which has the size equal to number of clusters K and has number of appearances of features in cluster i as value of its i^{th} component. To do this we represent each feature using a vector f_i of length U where U is the number users in the training data and $f_i(j)$ equals to the number of times user i has feature j. f_i are then normalized to have length 1. One important aspect in running K-means is that the relevant distance here is the angle between different feature vectors which can be computed using cosine similarity. Then the reduced dimensional vector Y_i is created for each user vector f_i as follows: denote $1 \le Cls(a) \le K$ the cluster assigned to feature a, we can compute Y_i using:

$$Y_i(j) = \sum_{a:X_i(a)>0 \& Cls(a)=j} f_i(a) \tag{8}$$

To be able to extract a reasonable number of features using K-means we need to have a relatively large number of clusters, since a small number of clusters (say 100) will results in large chunk of features to be in the same cluster considering the size of user features to be 3.5M. This will hence generate features that are difficult to learn useful pattern from. In order to mitigate this issue, we set the number of clusters to be $10k$. This means that on average there are 350 features per cluster. The large size of clusters and raw features makes K-means computationally expensive to run. In our experiment, we use a cloud-based Map-Reduce implementation for K-means [4].

6.3 Local sensitive Hashing

Local Sensitive Hashing (LSH) [10] works by projecting the data using a random projection matrix into a much lower dimensional space such that the pairwise cosine distance in the original space is still preserved in the new space. LSH requires a transformation matrix $A \in R^{d\times k}$, where d is the number of features in the original space and k is the number of random projections used. This means That A contains k different projection, indicated by A_i, each of which takes X vector and output a single hash value Y_i. The output vector $Y \in R^k$ of LSH can be found by concatenating all different hash values of Y_i. Specifically, to compute each Y_i we use the following equation:

$$Y_i = \begin{cases} 1 & \text{if } A_i X \ge 0 \\ 0 & else. \end{cases} \tag{9}$$

The cosine similarity between two vectors $X_1, X_2 \in R^d$ can be approximated by $cos\left(\frac{H(Y_1,Y_2)}{k}\pi\right)$ where $H(Y_1, Y_2)$ is the hamming distance between the LSH of input vectors. In order to preserve cosine similarity with higher accuracy we need to increase the number of projections used k. We used $k = 10,000$, the same as used for K-means clusters. While LSH can be applied to each vector independently and all projection can be computed independently which makes the algorithm highly distributable under the

Map-Reduce framework, the algorithm requires the generation of the transformation matrix A which is our case has $3.5M \times 10^4$ entries generated from $N(0,1)$, requiring about 300GB storage. Also, A must be stored on each node during the computation of LSH vectors. These issues make LSH prohibitively expensive in the Map-Reduce framework.

Many solutions have been proposed to address this problem. A lot of them are based on generating sparse A [13]. Here, we used the pooling trick introduced in [31]. The idea is to keep a pool B of size m of random numbers generated from $N(0,1)$ where m is usually significantly smaller than the size of a transformation matrix A. To get the entry in A_{ij} one simply applies consistent hash function of i, j to get an index in B and look up the value. In our experiments we set $m = 1,000,000$ which is more than $10,000$ times reduction in memory requirement and can be easily stored in each node in during Map-Reduce using only 10M storage.

6.4 Reduce the Number of Training Examples

The training data for each view contains a set of pairs $(User_i, Item_j)$ such that $User_i$ liked $Item_j$. In practice, a user may like many items which sometimes cause the training data to be very large. For example, in our News recommendation data set, the number of pairs is well over 1 billion and causes the training to be very slow even when using an optimized GPU implementation.

To alleviate this problem, we compress the training data so that it includes a *single* training example per user per each view. Specifically, the compressed training example includes the user features paired with the features with an *average* score of all items the user liked in that view. This reduces the number of training examples per view to the number of users in the training data, which greatly reduce the training data size. Note that a concern of this technique is that the objective function now becomes maximizing the similarity between user features and the average features of items the user liked. There is a slight difference when it comes to evaluation since during test time each user is given only one single item. However, this approximation is necessary to make the system scale up well. Also, experimental results will show that this approximation still generates very promising results in practice.

7. EXPERIMENTAL SETUP

In this section, we explain the process of our empirical study and briefly review several recommendation algorithms we compared to as baselines.

For each data set, we aim at evaluating the system performance on users who already have previous interaction in this domain (old users) and users who did not have any previous interaction but have some search and browsing history that can be encoded into user view (new users). For evaluation, the data sets were split into training and test sets using the following criteria:

First, each user is randomly assigned a label of either "train" or "test" with the probability ratio of 0.9:0.1. Then, for each user with a "test" label, we further label them as "old" or "new" user with the probability ratio of 0.8:0.2. For those labeled as "old" users, 50% of their items are used for training and the rest for test. On the other hand, for "new" users, their items are only used for testing as the user-item pairs are guaranteed to never appear in the training process. This way, these users are indeed completely *new* to the system. The details of the dataset breakdown is in table 2.

In terms of performance evaluation, for each $(user_i, item_j)$ pair in the training data we select 9 other random items $item_{r1}, ...item_{r9}$, where $r1$ though $r9$ are random indices, and create 9 testing pairs $(user_i, item_{rk}), 1 \leq k \leq 9$ and added to the testing data set. The evaluation criteria is to measure of how well the system ranks the correct pair $(user_i, item_j)$ against the other random items for the same user $(user_i, item_{rk})$. Therefore, we employed two metrics for this: (1) the Mean Reciprocal Rank (MRR), which computes the inverse of the rank of the correct item among other items and average the score across the whole testing data, and (2) Precision@1 (P@1) which computes the percentage of times the system ranks the correct item as the top item.

We compared to several baseline systems described below:

Standard SVD Matrix decomposition: In this baseline, we construct the user/item matrix and perform matrix decomposition using SVD. This is a standard baseline for collaborative filtering techniques that does not use any features from items or users. In practice, this baseline is computationally feasible only on relatively small data set (in our case, the Apps data). In addition, this approach cannot give recommendation on new users since they do not appear in the user-item matrix.

Most Frequent Items: Since SVD is unable to handle new user recommendation, this approach is used as trivial baseline for new users. It works by first computing the frequency of each item in the training data, and then for each testing sample it ranks $(user_i, item_j)$ with respect to other random items according to their frequency in training set.

Canonical Correlation Analysis (CCA): CCA [29] is a traditional multi-view learning technique aims at finding two pairs of linear transformations, one for each input view, such that the correlation between transformed data is maximized. It is similar to DSSM with two main differences: (1) CCA often uses linear transformation, despite there exists non-linear transformations through kernel versions of CCA, which is often computationally prohibited in large-scale experiments, and (2) CCA maximizes the correlation under certain fixed variance constraints while DSSM maximizes the rank of the correct pairs. The ranking objective has shown be to a better objective recommendation systems. In our experiment for CCA, we only used top-k users features since the other two dimension reduction techniques, K-means and LSH, produce a non-sparse feature vectors which make the correlation matrix too dense to be efficiently computed.

Collaborative Topic Regression (CTR): CTR [32] is a recently-developed recommendation system that combines both Bayesian matrix factorization and items features to create a recommendation for items. It has shown to be successful in academic paper recommendations. In the CTR model, two inputs are given: a collaborative matrix and items features (represented using bag-of-words). The model matches users to items by maximizing the reconstruction error of the collaborative matrix and leveraging item features as an extra signal. This helps to model new items that did not appear before in the training data. For new user recommendation in our scenario, we take the transpose of the collaborative matrix A as input and supply user features instead of items features.

For our proposed approach, for both Apps and News data sets, we first run three sets of experiments to train single-view DNN models, each of which corresponds to a dimension reduction method in Section 6 (SV-TopK,SV-Kmeans and SV-LSH). Then we run another three sets of experiments for MV-DNN. The first two combine Apps and News data using TopK and Kmeans user features (MV-TopK and MV-Kmeans). The third set of experiments train a joint model between Apps, News and Movie/TV features with TopK user features (MV-TopK w/Xbox).

8. RESULTS AND DISCUSSION

Table 3 and 4 shows results obtained from different approaches in App and News data respectively[1]. We separate the algorithms into three types for clarification: Type *I* are baseline algorithms; Type *II* are our single-view models and Type *III* are multi-view DNN models. We see that the naive most-frequent-item baseline performs very poorly which confirms that the simple solutions to new users will not work well in our cases. It is also shown that standard SVD matrix factorization is not good enough in this task even for the current users that have entries in the collaborative filtering matrix. Surprisingly, the CCA model performed no better than random guess in Apps data which shows that using the non-linear mapping in DSSM plus the ranking based objective are important to the system. The CTR model [32] performed reasonably well for current users but not as good for the new users.

For the single-view DNN (shown as Type *II*), the results indicated that the performance is dependent on the dimensionality reduction approach used. It can be seen that the best one for both Apps and News data was the top-K feature dimension method which outperformed other two by a large margin. This can be viewed as a confirmation of the hypothesis that users can be model with a relatively small set of informative features. It also indicates that K-means and LSH are less effective to capture the semantics of the user behavior correctly. As a head-to-head comparison between our single-view models (Type *II*) and traditional recommendation methods (Type *I*), our best model (SV-TopK) outperformed the best baselines CTR [32], which also leveraged item features for recommendation, by 11% for all users (0.497 vs. 0.448 MRR score), and 36.7 % for new users (0.436 vs. 0.319 MRR score), relatively. For P@1, we see a much larger improvement: 13% for all users and <u>88.7%</u>for new users, which shows the effectiveness of our system on recommending top-rated items.

Furthermore, for MV-DNN, results showed that adding more domains indeed helps to improve all domains at the same time. Specifically, by combining both News view and App view for training, we see a significant improvement on both News and App data sets in both metrics. Specifically, for the App data shown in Table 3, the MRR score increased from 0.497 to 0.517 for all users when comparing to the best single-view model, which is a 4% relative increase comparing to the single-view model. More importantly, we see a much bigger improvement on new users where one view's lack of new user data can be compensated by the data from other views. This is demonstrated by the 7% MRR (from 0.436 to 0.466) and 11% P@1 (from 0.268 to 0.297) relative improvement on the new users of App data set. Therefore, we are eager to know: can we safely conclude that more views can indeed improve the performance of the system? To find the answer to this hypothesis, we further added the Xbox data into the framework and trained a MV-DNN model with three user-item view pairs. The results are quite encouraging: MRR scores are further improved by 6% for all users and 8% for new users in App data, relatively. On the other hand, by comparing to the state-of-the-art algorithm, our best MV-DNN (with Xbox view) with top-K features performed 25.2% better than the CTR model for all users (from 0.277 to 0.347) , and <u>115%</u> better for new users (from 0.142 to 0.306), for P@1.

Similar results can also be observed for the News data in Table 4, where MV-DNN scored 49% better for all users and 101% better for new users than the CTR model, relatively. Note that in this table, the results for CCA and SVD are missing. Due to the extreme size of the training data which contains 1.5 Million users with over 1 Billion entries, these two traditional algorithms failed miserably to handle such big data. We will detail our discussion on the scalability in the next section but it is quite evident here that our DNN based approach can easily scale up to billion-entry computing capacity while yield excellent recommendation results.

In order to explore the effectiveness of the learned pattern from the system, we perform the following experiment to test the recommendation performance with *single-feature inputs*. Specifically, we took the best performing system (MV-DNN with top-k features) and construct user features with only one domain feature turned on. The resulting user feature thus has only one value which is the ID of the domain. We then run our model for prediction against other views in order to find the top-matched News and Apps among all existing items. Table 5 shows some of those results. It can be seen that the learned recommendation system is indeed quite effective. In the first example, we assume a user has only visited *brackobama.com*. The top matched news show all related information regarding *President Obama* and *Obamacare health*, which are all relevant to the website. On the other hand, the top matched Apps are also related to health in this case. In the second example, we have a user who visited *www.spiegel.de*, a major online German News Website, which is not telling a lot about users except they can read German. The system matches articles for them on *FIFA world cup 2014* which seems to be a common interest for Germans in this time span. In last example where the user seems to have interests in *baby-related* information, both the top matched News and Apps are quite relevant to *baby, pregnancy* and etc.

9. EXPERIMENTS ON PUBLIC DATA

To further show the strength of our approach in modeling cross-domain users, we perform a set of experiments on the public data[2] by the authors in [28]. The data set contains authors from different research fields and the objective is to recommend authors from another field for cross-domain collaboration. The data contains 33,739 authors from five domains (data mining, theory etc.), where each data entry specifies the name of the research field, the title and abstract of a paper, the list of authors and the year the paper got published. We use a single-view DNN to model this cross-domain collaboration (e.g., collaboration between data

[1]Due to the sensitivity of Xbox data, we are unable to show the individual results for Movie/TV recommendation performance. However, in the tables, we still show how Xbox view can be used to enhance the performance of other views.

[2]Available at http://arnetminer.org/billboard/collaboration

Data Set	Training			Testing		
	Number Of unique users	Number of unique items	Number of training pairs	Number of new users	Number of test pairs for old users	Number of test pairs for new users
Apps Data	200K	55k	2.5M	1K	11K	2K
News Data	1.5M	5M	> 1B	5K	50K	10K
Xbox Data	16K	10K	45K	1K	10K	3K

Table 2: Details of the Train/Test breakdown for the three data sets used to evaluate our proposed approach.

User View with Single Domain ID Feature	**Top Matched News**	**Top Matched Apps**
barackobama.com	Obama to Delay Obamacare Again to Help Democrats Froma Harrop: Democrats should not run away from Obamacare Democratic Senator: I am willing to defy Obama Governor Jindal proposes Republican alternative to Obamacare	7 Minutes Fitter Relax Meditate Escape Sleep Sleep Tracker U.S. Constitution
spiegel.de	Nazi-Era Jerseys on View in World Cup Exhibit 2014 World Cup Day 3 Lessons: Colombia Fun In The Sun... Belgium Vs. Algeria World Cup 2014: Live Stream... Colombia vs. Ivory Coast: Tactical Preview ...	ESPN Cricinfo Golf News RSS Pulse News Dinamalar - Tamil News Paper
linkedin.com	RectorSeal, ... Acquires Assets of Resource Conservation... Berkshire Partners Teams With Glen T. Senk To Co-Invest ... TF Financial: National Penn Bancshares, Inc. to Acquire ... H.I.G. Capital Portfolio Company Surgery Partners to Acquire ...	LinkedIn App LinkedIn Touch The Economist on Windows The Wall Street Journal
babycenter.com	Jenelle Evans' Baby Name: What We Know Catelynn Lowell ... Are Reportedly Pregnant With Baby #2! Jenelle Evans Can Take Drugs During Pregnancy If She Wants Pregnant Jenelle Evans: What Should She Name Her Baby?	Parents Pregnancy & Baby Guide ANIMALS FOR KIDS GAME Minecraft Fan Hub GS Preschool Games

Table 5: Examples of learned mapping between URL domains, News articles and Apps. For the domains, only their feature IDs are used for training. The underlying domain names are unknown to the target applications.

	Algorithm	**All Users**		**New Users**	
		MRR	P@1	MRR	P@1
I	Most Frequent	0.298	0.103	0.303	0.119
	CF	0.337	0.142	/	/
	CCA (TopK) [29]	0.295	0.105	0.295	0.104
	CTR [32]	0.448	0.277	0.319	0.142
II	SV- Kmeans	0.359	0.159	0.336	0.154
	SV-LSH	0.372	0.169	0.339	0.158
	SV-TopK	**0.497**	**0.315**	**0.436**	**0.268**
III	MV-Kmeans	0.362	0.16	0.339	0.156
	MV-TopK	0.517	0.335	0.466	0.297
	MV-TopK w/ Xbox	**0.527**	**0.347**	**0.473**	**0.306**

Table 3: Results for different algorithms on Windows Apps Data Set. Type *I* algorithms are baseline methods we compare with. Type *II* are single user-item view methods trained using the original DSSM framework. Type *III* are multi-view DNN models we proposed. The best performance is achieved by training a MV-DNN on all three user-item views with TopK as feature selection method.

	Algorithm	**All Users**		**New Users**	
		MRR	P@1	MRR	P@1
I	Most Frequent	0.301	0.111	0.305	0.111
	CTR [32]	0.427	0.215	0.276	0.123
II	SV-Kmeans	0.386	0.192	0.294	0.143
	SV-LSH	0.45	0.247	0.34	0.186
	SV-TopK	**0.486**	**0.286**	**0.358**	**0.208**
III	MV-Kmeans	0.391	0.194	0.296	0.145
	MV-TopK	0.494	0.303	0.368	0.222
	MV-TopK w/ Xbox	**0.503**	**0.321**	**0.398**	**0.245**

Table 4: Results for the News Data Set. Similarly, the best performance is achieved by our multi-view models. Note that due to the extreme big size of this data set (> 1B entries), traditional algorithms like CF (SVD) and CCA failed to handle it due to memory constraint.

mining and theory researchers). In this case, both the user view and item view share the same feature representations. Specifically, we use the unigram words from the titles and abstracts of the papers that an authors published during the training period (1990 to 2001) as features, which results in a feature dimension of 31,932. Similar to the original authors evaluation approach, we randomly selected a set of authors who have already had cross-domain collaboration during the training period and have at least five cross-domain collaborations during testing period to be our evaluation set. For each set of collaboration between different fields, we train a separate single-view DNN model in 100 iterations.

Table 6 shows the results. Overall, our method performs significantly better than the CTL method in all four cross-domain data sets except for the metric of P@20. In particular, we achieve a much higher recall at 100, with the best improvement of 96% in the DM to theory recommendation performance. The results indicate that using rich user features with non-linear deep neural models can indeed capture lots of semantics that cannot be accurately modeled

Cross Domain	**ALG**	**P@10**	**P@20**	**MAP**	**R@100**
DM to Theory	CTL	37.7	**36.4**	40.6	35.6
	SV-DNN	**40.0**	24.2	**41.2**	**69.8**
MI to DB	CTL	32.5	**30.0**	36.9	59.8
	SV-DNN	**35.7**	20.6	**39.7**	**70.9**
MI to DM	CTL	30.0	**24.0**	35.6	49.6
	SV-DNN	**37.2**	21.5	**41.8**	**63.3**
Visual to DM	CTL	28.3	**26.0**	32.8	36.3
	SV-DNN	**35.2**	20.8	**37.8**	**64.4**

Table 6: Recommendation performance on the cross-domain collaboration public data [28].

Algorithm	Data Set	
	Apps	News
CF	4 hours	OutOfMemory
CCA [29]	4 hours	OutOfMemory
CTR [32]	40 hours	120 hours
SV-TopK	33 hours	50 hours
MV-TopK	60 hours	
MV-TopK w/ Xbox	62 hours	

Table 7: Training time for all algorithms in comparison. The CTR model using Bayesian approach takes twice longer to finish than our most expensive DNN model.

using traditional word-based co-occurrence models such as generative topic models. We believe the performance can be further improved using multi-view DNN models but we leave that to future investigation.

In terms of efficiency, the authors reported 12-15 hours training time for the CTL method for the entire data set. Our algorithm runs much fast in that each model finished 100 iterations training within only 5-7 minutes on the same amount of data on a GPU machine. We will detail the scalability of our framework in the next section.

10. ALGORITHM SCALABILITY

This section we compare the performance of varies algorithms in terms of their training time. Recall that in the previous section, we mentioned that (in Table 4) for the News data, SVD (CF) and CCA were unable to handle the user-item matrix which has 1 Billion entries. This shows one of the strengths of our deep learning framework which is trained using SGD and thus able to handle massive amount of data using distributed training. The detail of the performance is shown in Table 7. We can observe that for the relatively small Apps data set, SVD and CCA finished relatively fast (around 4 hours, but yielded quite poor recommendation performance). The single-view DNN model (SV-TopK) finished 100 training iterations in 33 hours. The content-based CTR model, however, took a long time to train. The reason is that CTR requires an initial seed of topic proportion (θ) and topic distribution (β) trained using an LDA model. CTR then took these files and optimized the correlation between users and item features. Therefore, it turned out to be more expensive to train CTR than our deep learning model, for both data sets. On the other hand, we saw that both SV-TopK and MV-TopK exhibited (sub)linear training time to the data size, since usually SGD runs less epochs to converge when more data become available.

Figure 3 shows the training error during each iteration of the MV-TopK model for both News and Apps views. In our experiment, we manually specify the training iterations to be 100. One reason is that we continue to see improved performance for all views even though the improvement becomes smaller and smaller over time. On the other hand, we discovered that in practice the convergence of some views is faster than others. For example, for that particular model in Figure 3, the News view converged quickly after 20 iterations, while it took around 70 iterations for the Apps view to reach convergence. Due to the process of alternating user-item view pairs during training as well as the different convergence speed for different views, performing early stopping to further improve the scalability of our model becomes a critical future work.

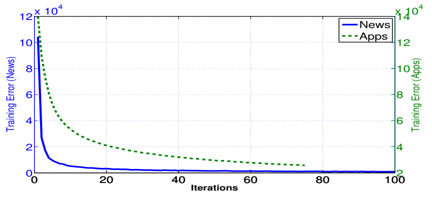

Figure 3: Training errors for two views in MV-TopK model.

11. CONCLUSION AND FUTURE WORK

In this work, we presented a general recommendation framework that uses deep learning to match rich user features to items features. We also showed how to extend this framework to combine data from different domains to further improve the recommendation quality. We then discussed different ways to make this approach scalable to large data sets through dimensionality reduction. The proposed model can handle both existing user and new user recommendation. Experiments on several large-scale real-world data sets indicated that the proposed approach worked much better than other systems by large margin.

As a pilot study, we believe that this work has opened a new door to recommendation systems using deep learning from multiple data sources. Despite the fact that most of the evaluation in this paper used proprietary data, the framework should be able to generalize to other data sources without much additional effort as shown in Section 9 using a small public data set. For example, recommending music to users based on their tweets, recommending restaurants according to Facebook status updates, or recommending images and videos based on user search queries.

For future work, we aim at incorporating more user features into the user view. We want to make our DNN learning more scalable so that eventually the entire set of user features can be used for training without dimension reduction. We also aim at adding more domains into our multi-view framework and further analysis its performance in details. Another important direction is to investigate how to incorporate collaborative filtering with our approach which currently running only as content-based filtering approach.

12. ACKNOWLEDGEMENTS

We greatly appreciate the authors in the KDD'12 paper [28] for making their data public. In particular, Sen Wu for his great support in discussion of the data set. We also want to thank anonymous reviewers for their invaluable feedback.

13. REFERENCES

[1] Fabian Abel, Qi Gao, Geert-Jan Houben, and Ke Tao. Twitter-based user modeling for news recommendations. In *IJCAI'13*, pages 2962–2966.

[2] Amr Ahmed, Abhimanyu Das, and Alexander J Smola. Scalable hierarchical multitask learning algorithms for conversion optimization in display advertising. In *WSDM'14*, pages 153–162.

[3] Robert M Bell and Yehuda Koren. Improved neighborhood-based collaborative filtering. In *KDD'13 CUP*, 2007.

[4] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. 2007.

[5] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS'11*, pages 215–223.

[6] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW'07*, pages 271–280.

[7] Asela Gunawardana and Christopher Meek. Tied boltzmann machines for cold start recommendations. In *RECSYS'08*, pages 19–26.

[8] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979.

[9] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM'13*, pages 2333–2338.

[10] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC'98*, pages 604–613.

[11] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local collaborative ranking. In *WWW'14*, pages 85–96.

[12] Bin Li, Qiang Yang, and Xiangyang Xue. Transfer learning for collaborative filtering via a rating-matrix generative model. In *ICML'09*, pages 617–624.

[13] Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *KDD'06*, pages 287–296.

[14] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[15] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. Personalized news recommendation based on click behavior. In *IUI'10*, pages 31–40.

[16] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI'02*.

[17] Weike Pan, Evan Wei Xiang, Nathan Nan Liu, and Qiang Yang. Transfer learning in collaborative filtering for sparsity reduction. In *AAAI'10*, 2010.

[18] Al Mamunur Rashid, George Karypis, and John Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. *ACM SIGKDD Explorations Newsletter*, 10(2):90–100, 2008.

[19] Jasson DM Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML'05*, pages 713–719.

[20] Shaghayegh Sahebi and Peter Brusilovsky. Cross-domain collaborative recommendation in a cold-start context: The impact of user profile size on the quality of recommendation. In *User Modeling, Adaptation, and Personalization*, pages 289–295. Springer, 2013.

[21] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *ICML'08*, pages 880–887.

[22] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML'07*, pages 791–798.

[23] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW'01*, pages 285–295. ACM, 2001.

[24] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *SIGIR'02*, pages 253–260.

[25] Yang Song, Weiwei Cui, Shixia Liu, and Kuansan Wang. Online behavioral genome sequencing from usage logs: decoding the search behaviors. In *WWW'14*, pages 91–94.

[26] Yang Song, Hongning Wang, and Xiaodong He. Adapting deep ranknet for personalized search. In *WSDM'14*, pages 83–92.

[27] Shiliang Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, 23(7-8):2031–2038, 2013.

[28] Jie Tang, Sen Wu, Jimeng Sun, and Hang Su. Cross-domain collaboration recommendation. In *KDD'12*, pages 1285–1293.

[29] Bruce Thompson. Canonical correlation analysis. *Encyclopedia of statistics in behavioral science*, 2005.

[30] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *NIPS'13*, pages 2643–2651.

[31] Benjamin Van Durme and Ashwin Lall. Online generation of locality sensitive hash signatures. In *ACL'10 Short Papers*, pages 231–235.

[32] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *KDD'11*, pages 448–456.