Learning Global Term Weights for Content-based Recommender Systems

Yupeng Gu[*]
Northeastern University
Boston, MA, USA
ypgu@ccs.neu.edu

Bo Zhao
LinkedIn Corp
Sunnyvale, CA, USA
bozhao@linkedin.com

David Hardtke
LinkedIn Corp
Sunnyvale, CA, USA
dhardtke@linkedin.com

Yizhou Sun
Northeastern University
Boston, MA, USA
yzsun@ccs.neu.edu

ABSTRACT

Recommender systems typically leverage two types of signals to effectively recommend items to users: user activities and content matching between user and item profiles, and recommendation models in literature are usually categorized into collaborative filtering models, content-based models and hybrid models. In practice, when rich profiles about users and items are available, and user activities are sparse (cold-start), effective content matching signals become much more important in the relevance of the recommendation.

The de-facto method to measure similarity between two pieces of text is computing the cosine similarity of the two bags of words, and each word is weighted by TF (term frequency within the document) × IDF (inverted document frequency of the word within the corpus). In general sense, TF can represent any local weighting scheme of the word within each document, and IDF can represent any global weighting scheme of the word across the corpus. In this paper, we focus on the latter, i.e., optimizing the global term weights, for a particular recommendation domain by leveraging supervised approaches. The intuition is that some frequent words (lower IDF, e.g. \database") can be essential and predictive for relevant recommendation, while some rare words (higher IDF, e.g. the name of a small company) could have less predictive power. Given plenty of observed activities between users and items as training data, we should be able to learn better domain-specific global term weights, which can further improve the relevance of recommendation.

We propose a unified method that can simultaneously learn the weights of multiple content matching signals, as well as global term weights for specific recommendation tasks. Our method is efficient to handle large-scale training data generated by production recommender systems. And experiments on LinkedIn job recommendation data justify the effectiveness of our approach.

Keywords

Term weighting, recommender systems, feature selection

1. INTRODUCTION

Recommendations are ubiquitous on the web in all kinds of areas, including product recommendation, movie/music recommendation, job recommendation, etc. Recommender systems typically leverage two types of signals to effectively recommend items to users: user activities and content matching between user and item profiles. Recommendation models in literature are usually categorized into collaborative filtering models, content-based models and hybrid models. The effectiveness of different approaches varies depending on the characteristics of data and user interactions of the specific domain. For example, if user activities are sparse and a lot of users do not have much interaction with the system (among all LinkedIn users who we show job recommendation results, few people have actually applied the recommended jobs), collaborative filtering would suffer from the cold start issues and therefore we should rely more on content-based signals. Moreover, in scenarios where rich profiles about users and items are massively available (most LinkedIn users have rich profiles about their past work experience, titles and skills, etc.; jobs on LinkedIn also have complete profiles), effective content matching signals become even more important in the relevance of the recommendation.

When we talk about user profiles in recommendation context, we typically refer to the profile of user preferences. We could specifically ask for user preferences (in LinkedIn job recommendation we allow users to specify their preferred location, seniority level, etc.), infer user preferences from past user interactions [30], or assume other types of user profiles are proxies of user preferences. We focus on the last case in the scope of this paper. Specifically, we can reasonably assume users' LinkedIn profiles which include their past work experience, skills, titles indicate their preferences for future jobs, and therefore, we could rely on content matching signals between user profiles and job profiles to compute the relevance scores between users and jobs.

[*] This work was conducted during an internship at LinkedIn.

WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883069 .

Since we are using user profiles as proxies for user preferences for jobs, and both user profiles and job profiles are rich in text, it becomes obvious that more effective content analysis methods and text similarity measures are crucial for improving the relevance of recommendation. The de-facto method to measure similarity between two pieces of text is computing the cosine similarity of the two bags of words, and each word is weighted by TF (term frequency within the document) × IDF (inverted document frequency of the word within the corpus). If we go beyond the narrow definition of TF and IDF, in general sense, TF can represent any local weighting scheme of terms within each document, and IDF can represent any global weighting scheme of terms across the corpus. Numerous previous work on content analysis can be applied to improve the calculation of TF, e.g., we could use term weighting schemes in IR such as BM25, and we could apply NLP techniques such as keyword extraction, topic analysis and salience detection to select important and topical keywords from the document. In this paper, we focus on improving IDF, or the global term weights across corpus, and it is clear that any content analysis techniques for improving TF are orthogonal and can be easily integrated with our approach.

Inverse document frequency (IDF) [28] of a term is proportional to an inverse function of the number of document it occurs in. The idea is simple: the more documents a term appears in, the less likely it can distinguish relevant and non-relevant documents. This simple yet effective method has been popular over the decades with wide applications in various areas. However, sometimes IDF is not optimal or even not reliable. In some scenarios, relatively frequent terms could be very predictive. For example, "machine learning" [1] is a very predictive term in job recommendation, but its IDF is decreasing since we are having more machine learning jobs in the corpus. In some other circumstances, we should down-weight some rare and high IDF terms in a recommendation task. Consider the scenario when some rare but non-essential terms happen to match between user and job profiles, which could result in absurd job recommendation to the user.

In a production recommender system, tremendous amount of past user activities can be collected and serve as training data for improving the recommendation model. Intuitively, from the massive training data, we should be able to learn domain-specific global term weights, which are optimized for the particular prediction task comparing with unsupervised generic scheme like IDF. For example, if we observe users who list machine learning as their skills are more likely to apply for machine learning jobs, we can infer "machine learning" is a more important term. With optimized global term weights, the content matching signals and relevance of recommendation can be improved as a result.

Ideally, the learning of global term weights and the learning of final relevance score between users and items should be seamlessly integrated in a unified framework, with the same objective function. For example, if cosine function is used for computing the similarity between text, then the term weight learning should target on directly optimizing the cosine similarity; if there are multiple cosine similarity scores between different sections of user and item profiles (e.g. *user title* section vs. *job skills* section; and *user skills* section vs. *job skills* section), then the global term weights should be optimized holistically across all matching sections. The unified framework should also easily allow other features not based on content matching and cosine similarity in the overall relevance model. These aspects are all satisfied in our proposed method.

Learning global term weights has other applications as well. For example, when we construct inverted index of jobs for job search and recommendation, we could potentially ignore terms with low weights so that index size and query performance could be improved.

Overall, in this paper we investigate the problem of automatically learning global term weights for content-based recommender systems. More specifically, we propose a unified supervised learning framework that can simultaneously learn term weights as well as the weights of text similarity features (cosine similarity scores) in the final relevance model. Our proposed method is efficient to handle large scale training data generated by production recommender systems. We conduct experiments on real data from LinkedIn job recommendation system to demonstrate the effectiveness of our approach. Based on our knowledge, we are the first to propose such method.

2. PROBLEM DEFINITION

In this section, we formally define the problem we target to solve in this paper. First of all, we use the following examples to motivate our work.

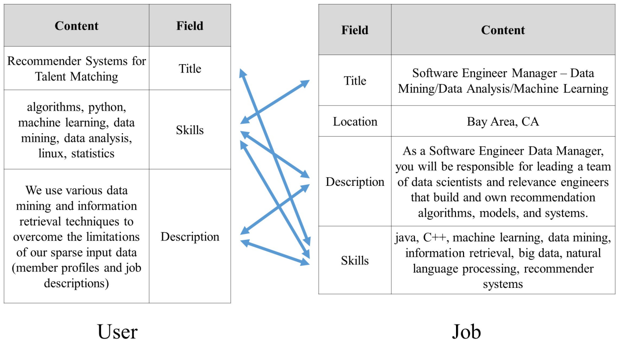
Case 1. In some cases, relatively frequent terms could be more predictive of relevant recommendation results. Consider a user with description "*I have enrolled in a project which provides users with a visualization of federal government financial statistics using machine learning techniques*". And we compute cosine similarity between user description with the following two jobs description with equal length: the first job says "*We are a managed services provider for the federal government*", and the second job says "*You will apply machine learning algorithms to analyze big data*". Both job descriptions share one phrase with the user ("federal government" vs. "machine learning"), but if we have more machine learning jobs than government-related jobs in the job database, "machine learning" will be associated with a lower IDF score than "federal government". As a result, the government job will have a higher similarity score than the machine learning job, but it is clearly less relevant.

Case 2. Consider the scenario when some rare but non-essential terms happen to match between user and job profiles, which could result in absurd job recommendation to the user. For example, a user has worked at a company that is funded by a venture capital firm V, and the user mentioned V in his/her profile. A job is posted by another company that is also funded by venture capital V, and V is also mentioned in the job description. The job is not related to the user's expertise, but the term V happens to match between the member and job profile. Since V is quite rare in the job corpus, it has very high IDF and therefore it can artificially boost the similarity score between member and job profile, although the job is not relevant to the user.

In this paper, we propose a supervised learning approach that can simultaneously learn global term weights as well as multiple text similarity features between user and item profiles. Formally, given a user and an item as in Figure 1, we aim to solve the following questions:

[1] In this work we also treat n-grams as terms.

- Predict the relevance score of the item to the user.
- Learn the weights of multiple content matching features between user and item profiles (e.g., user skills against job skills, user titles against job skills)
- Learn the optimal global term weights for each user text section and item text section (e.g., importance of \machine learning" in job skills)



Content	Field
Recommender Systems for Talent Matching	Title
algorithms, python, machine learning, data mining, data analysis, linux, statistics	Skills
We use various data mining and information retrieval techniques to overcome the limitations of our sparse input data (member profiles and job descriptions)	Description

Field	Content
Title	Software Engineer Manager – Data Mining/Data Analysis/Machine Learning
Location	Bay Area, CA
Description	As a Software Engineer Data Manager, you will be responsible for leading a team of data scientists and relevance engineers that build and own recommendation algorithms, models, and systems.
Skills	java, C++, machine learning, data mining, information retrieval, big data, natural language processing, recommender systems

Figure 1: An example of content matching signals between user and job profiles

We will propose a model that is able to address the issues mentioned in the two scenarios above since the global term weights are optimized for the particular recommendation domain.

We will use the notations in Table 1 in our paper. We refer recommendation items as jobs in this paper, but the approach can work with other types of items with rich text information (e.g., products, movies).

Table 1: Table of Notations

N_U	Number of users
N_V	Number of jobs
N_W	Number of terms
F_U	Number of fields for user
F_V	Number of fields for job
$\boldsymbol{u}^{i,s}$	User i's N_W dimensional bag-of-words representation in field s
$\boldsymbol{v}^{j,t}$	Job j's N_W dimensional bag-of-words representation in field t
$W^{(1)}_{sw}$	The weight for term w in field s
$W^{(2)}_{st}$	The weight for the pair of user field s and job field t
$\boldsymbol{W}^{(1)}$	Weight matrix in the first layer
$\boldsymbol{W}^{(2)}$	Weight matrix in the second layer
y_{ij}	Binary response from user i to job j
S_p	Set of field pairs that are used in the model
S^+	Positive label set
S^-	Negative label set
λ_1	Regularization coefficient for $\boldsymbol{W}^{(1)}$
λ_2	Regularization coefficient for $\boldsymbol{W}^{(2)}$

3. METHOD

In this section, we describe our approach in details, we first describe the overall model that predicts relevance of items to users, then we describe how we can incorporate the learning of global term weights into the model.

3.1 Logistic Regression Model

First we briefly review the logistic regression model, which is a commonly used model because of its efficiency and effectiveness. Logistic regression estimates the probability of a binary response based on a series of input features. In our task we will design features based on text similarity between user and job profiles and feed them into the logistic regression model to predict the probability that the job is a good recommendation for the user. To incorporate our data into the logistic regression model, we must carefully design the feature for each data instance, namely a pair of user and job.

Intuitively there are many reasons why the user is interested in a job: the user possesses the skills required by the job, the seniority of the job is a good match, they are in the same location, etc. This corresponds to the concept of fields in our framework: both users and jobs have various text fields such as title, skills, location and so on. An example is given in Figure 1 where the user and job profile have some matching fields.

For a pair of user text field and job text field, we calculate the similarity between the two pieces of text as a feature of the overall recommendation model. The classic text similarity function is cosine similarity, formally, the similarity between user i's field s and job j's field t is,

$$s^{(i,j)}_{s,t} = \frac{\boldsymbol{u}^{i,s} \cdot \boldsymbol{v}^{j,t}}{||\boldsymbol{u}^{i,s}||_2 \cdot ||\boldsymbol{v}^{j,t}||_2} . \quad (1)$$

Here $||\cdot||_2$ is the l_2 norm of a vector, and $\boldsymbol{u}^{i,s}$ represents the term vector, each dimension represents a term, and the value in each dimension represents the weight of the term. $\boldsymbol{u}^{i,s}$ is often decomposed as $\boldsymbol{tf}_s \circ \boldsymbol{idf}_s$, where $\boldsymbol{tf}_s$ represents local term weights. Each element of $\boldsymbol{tf}_s$ could simply be the frequencies of terms in the document, or it could be estimated by more advanced methods, such as IR weighting scheme (e.g., BM25), or other NLP and content analysis approaches. $\boldsymbol{idf}_s$ represents the inverted document frequencies of terms or other global term weights for all terms in field s, and $\circ$ is the element-wise product between two vectors: $\boldsymbol{x} \circ \boldsymbol{y} = (x_1 y_1, \cdots, x_n y_n)$.

Not every pair of fields is meaningful, for example, it does not make too much sense to compute the similarity between user's skills and job's location. Therefore we will only compute similarity scores in a subset S_p of all possible pair of fields as the features for the recommendation model. The selection of meaningful pairs could be done offline and by domain experts.

We assign weights for each pair of fields in S_p to indicate the importance of the feature. The probability that job j is a good recommendation for user i is determined by the weighted sum of input feature scores plus a threshold, namely

$$p(y_{ij} = 1|\{\boldsymbol{u}^{i,s}\}^{F_U}_{s=1}, \{\boldsymbol{v}^{j,t}\}^{F_V}_{t=1}, \boldsymbol{W}) = \sigma\Big(\sum_{s,t \in S_p} w_{st} s^{(i,j)}_{s,t} + w_0\Big) \quad (2)$$

where $\sigma(\cdot)$ is the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ and $s^{(i,j)}_{s,t}$ is defined as in Equation (1).

The logistic regression model in our framework is depicted in Figure 2. Note that it is flexible to include other features that are not based on content matching between users and jobs into this model, and the learning of the weights for those features are the same, except we do not perform global term weight optimization for those features, which we will describe in the next section.

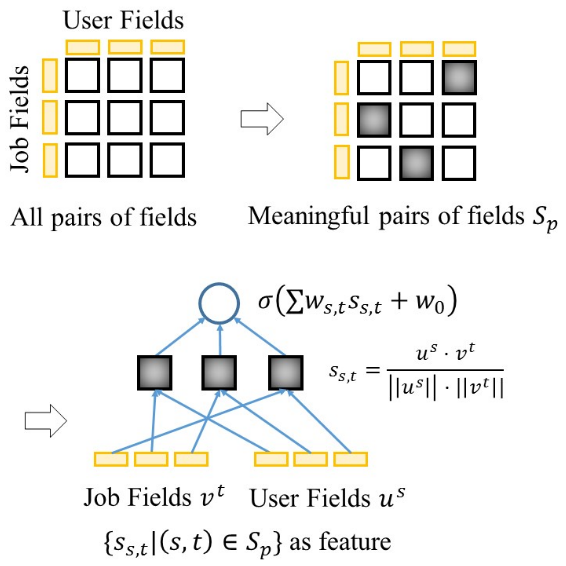

User Fields
Job Fields
All pairs of fields
Meaningful pairs of fields S_p
$\sigma(\sum w_{s,t} s_{s,t} + w_0)$
$s_{s,t} = \frac{u^s \cdot v^t}{||u^s|| \cdot ||v^t||}$
Job Fields v^t
User Fields u^s
$\{s_{s,t} | (s,t) \in S_p\}$ as feature

Figure 2: Logistic Regression Model

3.2 Multi-layer Logistic Regression Model

In this section, we describe how we can learn global term weights. In order to simultaneously learn term weights ($W^{(1)}_s$, $W^{(1)}_t$) and weights of text similarity features ($W^{(2)}_{s,t}$) in a unified framework, we design a two-layer neural network, where terms are in the bottom layer and text similarity features are in the top layer.

In the term layer of the neural network, we associate weights with each term. Those weights are considered as model parameters and the gradients can back-propagate to the first layer to learn them. More specifically, the raw feature of each word w in field s is adjusted from $u^{i,s}_w$ to $W^{(1)}_{sw} \cdot u^{i,s}_w$. Note that existing global weighting scheme such as IDF can still be incorporated in $u^{i,s}_w$.

On one hand, if a term is predictive in the recommendation task (a user with machine learning skill is more likely to apply for machine learning jobs), we would expect the corresponding global term weight to be large so that the term will be more important when cosine similarity is calculated. On the other hand, if some term is meaningless or even misleading, the weight for the corresponding term should be close to zero in order to reduce the effect of the term.

It is noteworthy that the same term in different fields will be assigned with different weights, since they may carry different meanings and have different importance scores. For instance, the term \California" is essential in field \location", but may not be that meaningful if it appears in another field. Therefore, we assign weight $W^{(1)}_{sw}$ to each term w and field s where the term appears in. We use a $(F_U + F_V) \times N_W$ matrix $\boldsymbol{W}^{(1)}$ to represent the term weights in all fields, where the entry at s^{th} row and w^{th} column represents the weight for term w in field s. Another design choice would be creating a set of term weights for each matching pair of user text field and job text field, but it would result in too many parameters to estimate.

Now user i's raw feature in field s ($\boldsymbol{u}^{i,s}$) is mapped to the transformed feature $\tilde{\boldsymbol{u}}^{i,s} = W^{(1)}_s \circ \boldsymbol{u}^{i,s}$, where $W^{(1)}_s$ denotes the s^{th} row of matrix $W^{(1)}$. The field pair similarity score is now calculated based on the transformed feature values. An example of the first layer is shown in Figure 3. Formally, the similarity score between user i's field s and job j's field t is changed to

$$s^{(i,j)}_{s,t} = \frac{\tilde{\boldsymbol{u}}^{i,s} \cdot \tilde{\boldsymbol{v}}^{j,t}}{||\tilde{\boldsymbol{u}}^{i,s}||_2 \cdot ||\tilde{\boldsymbol{v}}^{j,t}||_2}. \tag{3}$$

After the construction of first layer, each input feature (similarity score) is assigned a weight in the overall relevance model. Naturally these weights appear in the second layer of the neural network and we represent the weight as a sparse $F_U \times F_V$ matrix $\boldsymbol{W}^{(2)}$, where the element at s^{th} row and t^{th} column represents the weight for user-job field pair (s, t). A toy example of the neural network model is depicted in Figure 4. Note that in this framework, we could easily add features in the second layer that are not based on content similarity, and we can learn the weights of these features together with the text similarity features, the only difference is that we do not have term layer for those non-text features.

The probability that user i will apply for the job j is given by

$$p_{ij} = p(y_{ij} = 1) = \sigma\Big(\sum_{s,t \in S_p} W^{(2)}_{s,t} s^{(i,j)}_{s,t} + w^{(2)}_0\Big) \tag{4}$$

where σ is the logistic function and y_{ij} is a binary value that indicates the label of the user-job pair. Basically $y_{ij} = 1$ denotes a successful recommendation and $y_{ij} = -1$ means otherwise. We will use σ_{ij} to denote $\sum_{s,t \in S_p} W^{(2)}_{s,t} s^{(i,j)}_{s,t} + w^{(2)}_0$ in the remaining part.

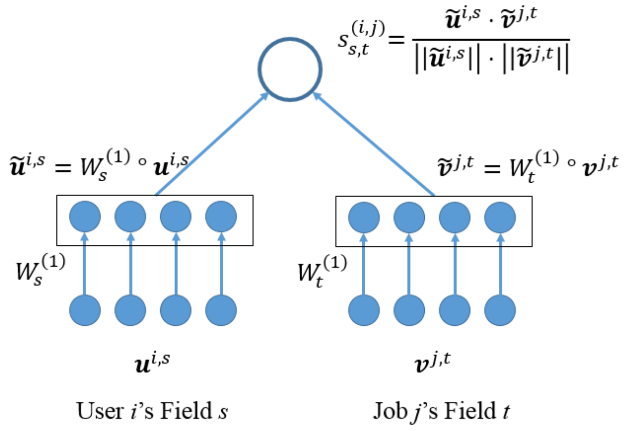

$s^{(i,j)}_{s,t} = \frac{\tilde{\boldsymbol{u}}^{i,s} \cdot \tilde{\boldsymbol{v}}^{j,t}}{||\tilde{\boldsymbol{u}}^{i,s}|| \cdot ||\tilde{\boldsymbol{v}}^{j,t}||}$
$\tilde{\boldsymbol{u}}^{i,s} = W^{(1)}_s \circ \boldsymbol{u}^{i,s}$
$\tilde{\boldsymbol{v}}^{j,t} = W^{(1)}_t \circ \boldsymbol{v}^{j,t}$
$W^{(1)}_s$
$W^{(1)}_t$
$\boldsymbol{u}^{i,s}$
$\boldsymbol{v}^{j,t}$
User i's Field s
Job j's Field t

Figure 3: First Layer of the Neural Network Model

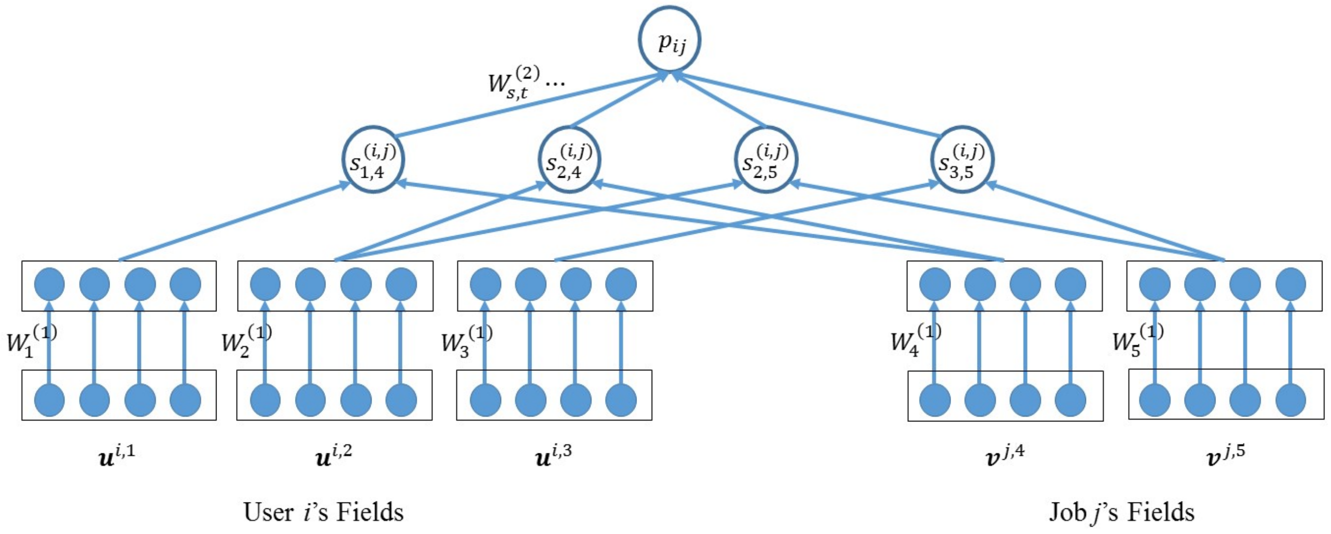


Figure 4: The Neural Network Model - A Toy Example. Each filled circle denotes a term and each box represents a field. Each hollow circle denotes a neuron in the neural network. Parameters are $W^{(1)}$ and $W^{(2)}$.

We use log-loss to denote the prediction error, therefore the objective function is

$$\mathcal{J}(\boldsymbol{W^{(1)}}, \boldsymbol{W^{(2)}}) = \sum_{i=1}^{N_U} \sum_{j=1}^{N_V} \log\left(1 + e^{-y_{ij}\sigma_{ij}}\right) .$$

We will minimize the objective with respect to model parameters $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$.

The summation above contains $N_U \cdot N_V$ terms, namely all possible pairs of users and jobs in the dataset. However, in real world the actual user-job interaction matrix is very sparse. Therefore we only consider the set of good recommendations S^+, and sample the set of bad recommendations S^- from the remaining pairs. The union of the two disjoint sets will be used to approximate the two summations in the formula above. We will specify the criterion for good/bad recommendations in our dataset and how they are generated in section 4.1.

3.3 Regularization

The parameter of our model is $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$. l_2 regularization is added on logistic weights $\boldsymbol{W}^{(2)}$ to avoid overfitting. For the term weights $\boldsymbol{W}^{(1)}$, as the feature dimension is very large and majority of terms should receive small weights, we add an l_1 regularization on $\boldsymbol{W}^{(1)}$ to encourage sparsity in term-level weights. So the final objective function we will minimize is

$$\begin{aligned} J(\boldsymbol{W^{(1)}},\boldsymbol{W^{(2)}}) = & \sum_{(i,j)\in S^+ \cup S^-} \log\left(1 + e^{-y_{ij}\sigma_{ij}}\right) \\ & + \lambda_1 \cdot \sum_{s=1}^{F_U+F_V} \sum_{w=1}^{N_W} |W_{sw}^{(1)}| + \frac{\lambda_2}{2} \cdot ||\boldsymbol{W}^{(2)}||_F^2 \end{aligned} \quad (5)$$

where $||\cdot||_F$ is the Frobenius norm of a matrix: $||\boldsymbol{A}||_F = \left(\sum_{m,n} A_{mn}^2\right)^{1/2}$.

3.4 Optimization

Since the number of parameters is large and there are tremendous amount of training data, we use stochastic gradient descent (SGD) to learn the model, since it is proven to be scalable and effective. For learning term weights in the bottom layer, we use ideas similar to the common backpropagation approach [23], where the error is propagated backwards from the top label layer down to the first layer. To handle the optimization for l_1 norm, we use the subgradient strategy proposed in [24].

The gradients w.r.t parameters can be calculated as follows. First, we look at the top layer weights (for field pairs):

$$\frac{\partial J}{\partial W_{st}^{(2)}} = \sum_{i,j:y_{ij}\neq 0} c_{ij} \cdot \frac{\boldsymbol{u}^{i,s} \cdot \boldsymbol{v}^{j,t}}{||\boldsymbol{u}^{i,s}|| \cdot ||\boldsymbol{v}^{j,t}||} + \lambda_2 \cdot W_{st}^{(2)} \quad (6)$$

where

$$c_{ij} = -y_{ij} \cdot \frac{e^{-y_{ij}\sigma_{ij}}}{1 + e^{-y_{ij}\sigma_{ij}}}.$$

Then, the gradients for first layer weights in user fields:

$$\frac{\partial J}{\partial W_{sw}^{(1)}} = \begin{cases} \frac{\partial \tilde{J}}{\partial W_{sw}^{(1)}} + \lambda_1 \frac{W_{sw}^{(1)}}{|W_{sw}^{(1)}|} & \text{if } |W_{sw}^{(1)}| > 0 \\ \frac{\partial \tilde{J}}{\partial W_{sw}^{(1)}} + \lambda_1 & \text{if } W_{sw}^{(1)} = 0,\ \frac{\partial \tilde{J}}{\partial W_{sw}^{(1)}} < -\lambda_1 \\ \frac{\partial \tilde{J}}{\partial W_{sw}^{(1)}} - \lambda_1 & \text{if } W_{sw}^{(1)} = 0,\ \frac{\partial \tilde{J}}{\partial W_{sw}^{(1)}} > \lambda_1 \\ \frac{\partial \tilde{J}}{\partial W_{sw}^{(1)}} & \text{if } W_{sw}^{(1)} = 0, -\lambda_1 \leq \frac{\partial \tilde{J}}{\partial W_{sw}^{(1)}} \leq \lambda_1 \end{cases}$$

where

$$\begin{aligned} \frac{\partial \tilde{\mathcal{J}}}{\partial W_{sw}^{(1)}} = & \sum_{i,j:y_{ij}\neq 0} \sum_{t:(s,t)\in S_p} \frac{c_{ij} \cdot W_{st}^{(2)}}{||\tilde{\boldsymbol{u}}^{i,s}|| \cdot ||\tilde{\boldsymbol{v}}^{j,t}||} \cdot \Big(u_w^{i,s} v_w^{j,t} W_{tw}^{(1)} \\ & - \frac{W_{sw}^{(1)} \cdot (u_w^{i,s})^2 \cdot (\tilde{\boldsymbol{u}}^{i,s} \cdot \tilde{\boldsymbol{v}}^{j,t})}{||\tilde{\boldsymbol{u}}^{i,s}||^2}\Big) . \end{aligned} \quad (7)$$

The gradients for first layer weights in job fields are similar:

$$\frac{\partial J}{\partial W_{tw}^{(1)}} = \begin{cases} \frac{\partial \tilde{J}}{\partial W_{tw}^{(1)}} + \lambda_1 \frac{W_{tw}^{(1)}}{|W_{tw}^{(1)}|} & \text{if } |W_{tw}^{(1)}| > 0 \\ \frac{\partial \tilde{J}}{\partial W_{tw}^{(1)}} + \lambda_1 & \text{if } W_{tw}^{(1)} = 0,\ \frac{\partial \tilde{J}}{\partial W_{tw}^{(1)}} < -\lambda_1 \\ \frac{\partial \tilde{J}}{\partial W_{tw}^{(1)}} - \lambda_1 & \text{if } W_{tw}^{(1)} = 0,\ \frac{\partial \tilde{J}}{\partial W_{tw}^{(1)}} > \lambda_1 \\ \frac{\partial \tilde{J}}{\partial W_{tw}^{(1)}} & \text{if } W_{tw}^{(1)} = 0, -\lambda_1 \leq \frac{\partial \tilde{J}}{\partial W_{tw}^{(1)}} \leq \lambda_1 \end{cases}$$

$$\frac{\partial \mathcal{J}}{\partial W_{tw}^{(1)}} = \sum_{i,j:y_{ij}\neq 0} \sum_{s:(s,t)\in S_p} \frac{c_{ij} \cdot W_{st}^{(2)}}{||\tilde{\boldsymbol{u}}^{i,s}|| \cdot ||\tilde{\boldsymbol{v}}^{j,t}||} \cdot (u_w^{i,s} v_w^{j,t} W_{sw}^{(1)} - \frac{W_{tw}^{(1)} \cdot (v_w^{j,t})^2 \cdot (\tilde{\boldsymbol{u}}^{i,s} \cdot \tilde{\boldsymbol{v}}^{j,t})}{||\tilde{\boldsymbol{v}}^{j,t}||^2}) \,. \tag{8}$$

Since the terms in the first layer are very sparse, we need to consider such sparsity in deciding the learning rate in SGD. Here, we use the method of adaptive step-size described in [22] and [10] to update the learning rate for each feature dynamically. The intuition is that for a sparse feature that appear very few times, the step size of such feature should be larger. Specifically, we keep track of the gradient applied on a parameter in every iteration and decrease its step-size accordingly. In short, the more a parameter has been updated, the smaller its step-size becomes. The updating rule of any parameter θ is given by

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \frac{g_{t+1}}{\sqrt{\sum_{t'=1}^{t+1} g_{t'}^2}} \tag{9}$$

where $\theta^{(t)}$ is the value of θ for the t^{th} times θ appears, η is the learning rate and g_t is the gradient of θ for the t^{th} times that θ is updated.

4. EXPERIMENTS

In this section we will demonstrate the advantage of our model compared to baseline models. We will first describe the dataset and then evaluate our model with other baselines. We also conduct several case studies to show which terms are the most predictive in our recommendation task, as well as which pairs of fields are important. Those case studies show the alignment of our model with the intuition.

4.1 Dataset

We use a real world dataset from LinkedIn[2] to evaluate our model. LinkedIn has a feature called “Jobs You May Be Interested In" (JYMBII), which provides job recommendations to its members that match the member's profile in some way. When a user logs in, he/she is able to see several recommendations under the JYMBII panel in the timeline as in Figure 5, and the user can click the job to see details, apply for it or simply ignore them.

We used job recommendation data from May 2015. Each record contains information such as the user ID, job ID, whether the user applied/viewed for the job, time stamp and so on. We further divide them into two sets according to the interaction between the user and the job. We consider the label of a user-job pair as *positive* if the member *applied for* the job, and one as *feedback negative* if the member has seen the job recommendation but *did not click* it. The reason for the different criterion is to distinguish the two labels as much as we can, as applying for the job is a much stronger behavior than simply clicking it. The collection of positive pairs constitutes the positive label set S^+. In consideration of balancing positive and negative samples, we sample a subset of negative pairs from our negative label set S^-. For the negative label set, half is chosen from the feedback negatives (the user did not click the job), the other half is called random negatives, which are generated by randomly sampling pairs of users and jobs. We need the random negatives because there is bias if we only use feedback negatives as negative training data.

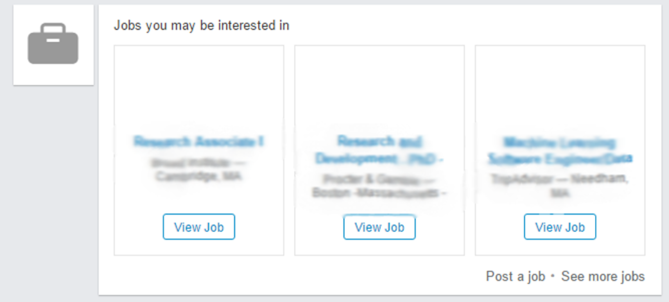


Figure 5: JYMBII (“Jobs you may be interested in”) panel

In total, our sample data contain about 3.1 million user-job pairs. 90% of the data are used for training and the remaining 10% as test. The dictionary contains 490,089 distinct terms, where stop words have already been removed and meaningful phrases are also extracted (such as “machine learning"), we simply treat these phrases the same as other uni-gram terms. Users have 51 fields and jobs have 24 fields. As mentioned before, we manually scanned the possible field pairs between users and jobs and keep 79 field pairs in the logistic regression.

4.2 Baseline

In our experiments, we compare with a baseline and several variations of our method. In all the methods, TF in certain short text fields are simply term frequency, while in longer text fields are BM25 scores. Standard IDF scores are also used in our approach as mentioned in Section 3.2, so we are essentially learning an adjustment of the standard IDF scores.

- The basic logistic regression model as described in 3.1 with TF-IDF weighting scheme.

- One variation of our multi-layer logistic regression model where only term weights in the job's fields are learned, while on member side, heuristic TF-IDF is used. Since we reduce the number of parameters, the training of this model is more efficient.

- One variation of our multi-layer logistic regression model where only top portion of terms are kept in every field after the training process. The remaining terms are dropped as if they never exist. This mainly tries to test

[2] http://www.linkedin.com/

if we can effectively reduce our index size by learning term weights.

By comparing our model with the first baseline, we are able to show the advantage of using automatically learned term weights in a specific task rather than using a heuristic one. Improvement of the first variation demonstrates the significance in constructing entity feature using learned parameters as well. The comparison to the second variation illustrates our ability to achieve a good recommendation result effectively using only a portion of the terms. For fair comparison, we use the same coefficient for the l_2 regularization and apply adaptive learning rates in all methods above.

4.3 Evaluation of the Multi-layer Logistic Regression Model

We use the area under the ROC (receiver operating characteristic) curve (AUC) as well as the area under precision-recall curve (AUPRC) to evaluate the results. AUC and precision/recall are important measures in terms of recommendations. ROC curve illustrates the performance of a binary classifier as the threshold changes. Two axes of the curve is true positive rate $TPR = \frac{\sum \text{True Positive}}{\sum \text{Condition Positive}}$ and false positive rate $FPR = \frac{\sum \text{False Positive}}{\sum \text{Condition Negative}}$. AUC is the area under the ROC curve, and basically the score will be higher if the probability of more positive instances ranks higher than negative ones. Precision-recall curve is also a plot that presents the results of a binary classifier and it is created by plotting precision against recall as the threshold varies. Both of them are popular evaluation measures for a binary classifier.

Table 2: Effectiveness of MLRM

Method	AUC	AUPRC
Baseline	0.692	0.671
MLRM	**0.811 (+17.2%)**	**0.793 (+18.2%)**
MLRM (jobs only)	0.792 (+14.5%)	0.771 (+14.9%)

Table 2 shows the AUC and AUPRC of baseline, our method MLRM and MLRM (jobs only). The ROC curve and the precision-recall curve of our methods and the baseline are shown in Figure 6. As we can see, MLRM can improve both measures by more than 17% over the baseline, which clearly justifies the effectiveness of learning global term weights. Even if we only learn term weights for jobs, we could still improve the relevance by 14%.

Table 3: Effectiveness of selecting top terms

Method	AUC	AUPRC
MLRM	**0.811 (+17.2%)**	**0.793 (+18.2%)**
MLRM (top 90%)	0.786 (+13.6%)	0.764 (+13.9%)
MLRM (top 80%)	0.760 (+9.8%)	0.768 (+14.4%)
MLRM (top 50%)	0.756 (+9.3%)	0.761 (+13.4%)
MLRM (top 10%)	0.744 (+7.5%)	0.780 (+16.2%)

In consideration of efficiency, after learning the parameters for terms in all fields, we could use the terms that have highest weights in each field as the field's representation. Those terms which have a weight lower than a threshold are discarded. If we build inverted index on jobs to allow search and recommendation, we could only select these top terms so that index size can be reduced and query time can be improved, since query terms that are not top terms will not hit any inverted index. The results of the variations are shown in Table 3, in general if we drop terms, results will be worse, but they are still better than the baseline. Note that we already perform L1 normalization in learning the term weights, so there are already quite a few terms that have zero weights in the full model. We can see that even if we only use the top 10% terms, AUC is still 7.5% better than the baseline.

In sum, our full model has the best performance among all the trials. We are able to achieve an outstanding performance as well even if we are only allowed to manipulate one type of entity in a recommendation task. In particular, our approach still has considerable improvement over the baseline even if we use only half of the terms in every field to do recommendation, which indicates less storage requirement and better computing efficiency. The results of keeping fewer important keywords show a trade-off between performance and even higher space and computational efficiency.

Table 4: Sensitivity of regularization parameter

Coefficient of l_1 Regularization	AUC	AUPRC
$\lambda_1 = 10^{-8}$	0.804 (+16.2%)	**0.789 (+17.6%)**
$\lambda_1 = 10^{-7}$	0.806 (+16.4%)	**0.789 (+17.6%)**
$\lambda_1 = 10^{-6}$	**0.808 (+16.8%)**	**0.789 (+17.6%)**
$\lambda_1 = 10^{-5}$	0.780 (+12.7%)	0.750 (+11.8%)

We set the regularization coefficient to be $\lambda_1 = 10^{-6}$ and $\lambda_2 = 10^{-5}$ in all of the results and figures above. We try different values of λ_1 in our model and the comparison with baseline is shown in Table 4. We can observe that our model is not sensitive to the choice of regularization coefficient.

We also study the effect of adaptive learning rate and l_1 regularization in Table 5. In short, regularization and optimizing tricks do improve our model's performance.

Table 5: Effect of Adaptive Learning Rate and l_1 Regularization

Method	AUC	AUPRC
MLRM without l_1 and adaptive	0.793 (+14.6%)	0.774 (+15.4%)
MLRM with l_1	0.808 (+16.8%)	0.789 (+17.6%)
MLRM with adaptive	**0.811 (+17.2%)**	**0.793 (+18.2%)**

4.4 Case Studies

In addition to the improvement in terms of AUC/AUPRC as shown above, we also conduct a few case studies on the parameters which can tell some interesting stories behind our model.

4.4.1 First Layer Weights $W^{(1)}$

These weights are also known as term weights in different fields. Recall that $W^{(1)}_{sw}$ is large if term w is discriminating

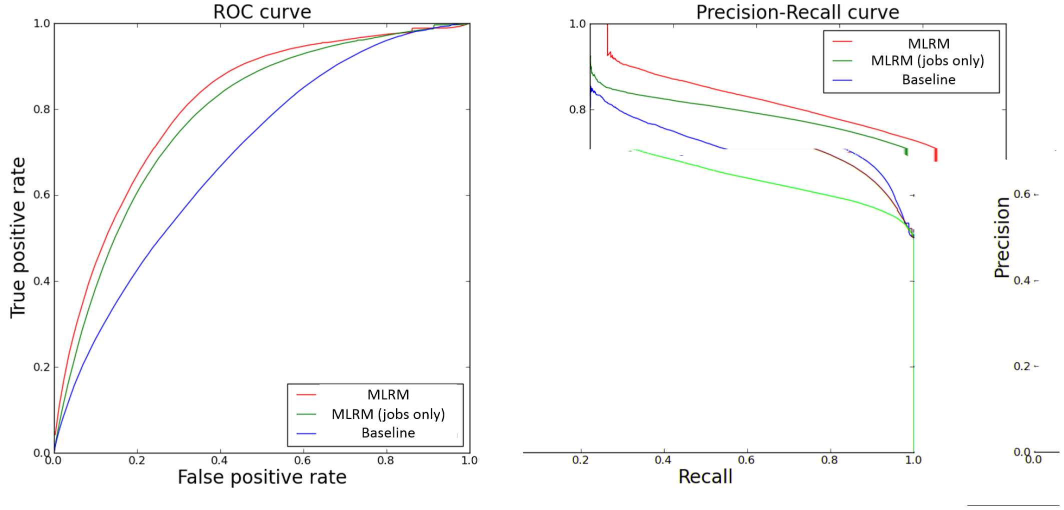


Figure 6: Comparison on Test Dataset

and predictive in field s, whereas $W^{(1)}_{sw'}$ is close to zero if term w' acts like a stop word. We plot a histogram of $W^{(1)}_{sw}$ for some of job fields in Figure 7, and histograms on term weights in user fields are similar. The weight distribution basically follows the Zipf's Law.

There are a few terms in each field with large weights. The importance of these keywords is straightforward: if they appear in the particular fields of both user and job, the user is more likely to apply for the job. In other words, these keywords are predictive in our job recommendation task. From the results we can also recognize which are the most predictive skills, locations, etc. when people are looking for jobs. For instance, the term \machine learning" has a large weight in both user and job's skill field. It can be inferred that the chance of a machine learning person applying for a machine learning related job is higher than another user-job pair with different skills.

Table 6: Top Field Pairs

Rank	User Field	Job Field
1	Skill Id	Skill Id
2	Summary	Description
3	Skill Terms	Skill Terms
4	Past Position Summary	Description
5	Past Title	Title
...	...	...
79	Past Title	Skill

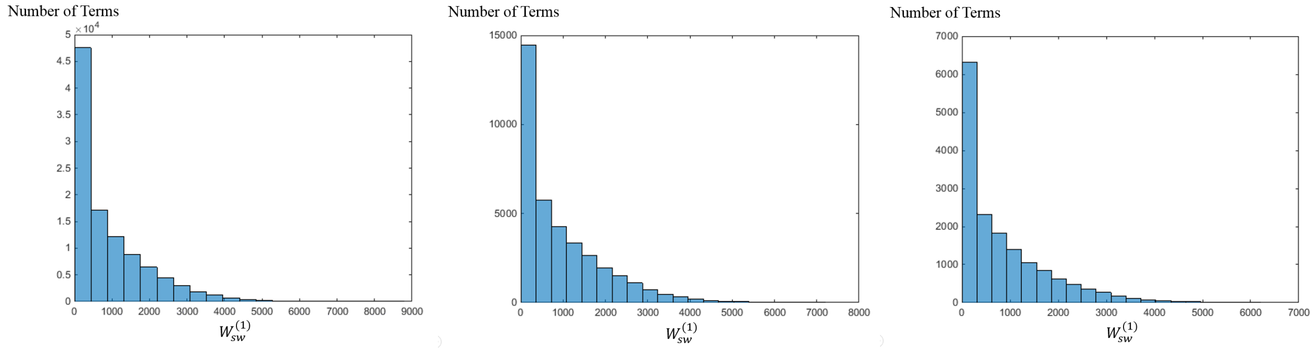
4.4.2 *Second Layer Weights* $W^{(2)}$

These field-pair weights indicate which pairs of fields come first in determining the overall relevance of jobs to users. A larger weight $W^{(2)}_{st}$ indicates that the interaction between user field s and job field t is more essential. We sort those field pairs according to $W^{(2)}_{st}$ and summarize the results in Table 6. We can see from the table that the most important factors in job application are, the matching of user and job skills (ranking 1^{st} and 3^{rd}), and user's summary with job's description (ranking 2^{nd}); whereas the matching between user's past title and job's skills is the least important.These field-pair weights agree with both the weights learned by the baseline model and our intuition.

5. RELATED WORK

Designing text representations from the content has attracted interests from researchers in various fields. The defacto method for weighting terms is the TF × IDF scheme, and in general sense TF can represent any local weighting scheme of the word within each document, and IDF can represent any global weighting scheme of the word across the corpus. Numerous content analysis approaches can be applied to determine TF, including IR based measures (e.g., BM25), NLP based methods such as topic analysis, keyword extraction, salience detection, etc. As we have mentioned, these methods are orthogonal to the focus of this paper, and they can be easily integrated with our approach.

In this paper we focus on learning the global term weights, so far the most successful approach for global term weights has been inverse document frequency (IDF), which was first introduced as \term specificity" in [28]. Some approaches have been proposed to optimize term weights for the purpose of document categorization, including some supervised approaches [6, 8, 27, 15, 16, 4, 19, 7], which exploit the category label of the documents to provide some guidance on term weighting. Those methods build classifiers that estimate the probability that each document belongs to certain category using term weights as parameters. Soucy and Mineau [27] utilize statistical confidence intervals to estimate the proportion of documents containing each term, thus define the strength of each term. Their method favors the terms that are proportionally more frequent in the positive class. Deng et al. [7] propose a weighting scheme that consists of two parts: the importance of a term in a document as well as the importance of the term for expressing sentiment. Those measures are learned based on statistical functions of the supervised document label information. Lan et al. [15] propose a new factor called \relevant frequency"



Description **Skills** **Standardized Skills**

Figure 7: Job Fields

which takes category information into account to improve term's discriminating power. Unsupervised text representations are mostly based on statistical information of terms in the corpus. These measures contain document frequency, χ^2 statistic [31], information gain, mutual information, odds ratio and so on.

The settings of these previous methods are mainly document categorization, which is very different from the setting of this paper on content-based recommendation, where cosine similarity is leveraged to calculate the similarity between texts, and therefore the learning of term weights should directly optimize the cosine similarity. Moreover, there are multiple matching fields between users and items, and they should be considered holistically for learning term weights.

For recommendation systems where entities are associated with text information, there are various context-aware methods [2, 13] that try to incorporate user profile information with the system in order to achieve a better recommendation performance. Specifically, many models have been proposed to utilize text information. Text can be used as a pre-filter [1], post-filter, or integrated with the recommendation model. Among the integrated models, some approaches [20, 9, 5, 18] use text to do user classification or profile inference, and apply the learned label as either filtering or modification to the rating scores. Some approaches [11, 3, 29, 21, 12] use trained textual labels or sentiments for latent rating dimension. They try to correspond the topics inferred from the text with the latent factors in content-based recommendation models. For example, Agarwal and Chen [3] build a topic model on both user and item side, and use the topic distribution to match the latent factor in matrix factorization. Other methods consider text as an auxiliary feature besides the ratings. Li et al. [17] consider text as an additional dimension of input feature for the recommendation model. Apart from directly using text as an additional feature for a recommendation task, several latent semantic models have been developed in order to obtain the similarity of two documents at *topic* level. This is inspired by the fact that sometimes two relevant documents may share few terms in common because of their language discrepancy. In this setting, a deep structure is usually built to generate a highly non-linear concept vector for a text string. Some of the studies have been applied to web search ranking and relevance tasks [14, 26, 25]. Although these approaches present a more sophisticated framework to utilize textual knowledges, our simple yet effective method has a very clear explanation of the role of each term in the recommendation system. In addition, our algorithm is more scalable towards real-world tasks. All of these methods rely on the representation of text without optimizing the text representation at *term* level. In this paper we propose a general framework that can simultaneously learn domain specific term weights as well as relevance between users and items for recommendation.

6. CONCLUSION

In this paper, we propose a method to learn global term weights for improving content-based recommendation. Our method can simultaneously learn term weights and the final relevance score between users and items. Text similarity function (cosine) is directly optimized and multiple cosine similarity scores between different sections of user and item profiles are considered holistically. The unified framework also easily allows other features not based on content matching and cosine similarity in the overall relevance model. Our proposed method is efficient to handle large scale training data generated by production recommender systems. We conduct experiments on real data from LinkedIn job recommendation system to demonstrate the effectiveness of our approach, we could improve AUC by over 17%. Moreover, we demonstrate that learning global term weights has potential to improve the efficiency of recommender systems.

Acknowledgements

This work is partially supported by NSF Career Award #1453800.

7. REFERENCES

[1] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. on Information Systems (TOIS)*, 23(1):103{145, 2005.

[2] G. Adomavicius and A. Tuzhilin. Context-aware recommender systems. In *Recommender Systems Handbook*, pages 217{253. Springer, 2011.

[3] D. Agarwal and B.-C. Chen. fLDA: matrix factorization through latent dirichlet allocation. In *Proc. of the third ACM Int. Conf. on Web Search and Data Mining*, pages 91{100, 2010.

[4] L. Barak, I. Dagan, and E. Shnarch. Text categorization from category name via lexical

reference. In *Proc. of Human Language Technologies: The 2009 Annual Conf. of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 33{36, 2009.

[5] M. De Gemmis, P. Lops, G. Semeraro, and P. Basile. Integrating tags in a semantic content-based recommender. In *Proc. of the 2008 ACM Conf. on Recommender Systems*, pages 163{170, 2008.

[6] F. Debole and F. Sebastiani. Supervised term weighting for automated text categorization. In *Text Mining and its Applications*, pages 81{97. Springer, 2004.

[7] Z.-H. Deng, K.-H. Luo, and H.-L. Yu. A study of supervised term weighting scheme for sentiment analysis. *Expert Systems with Applications*, 41(7):3506{3513, 2014.

[8] Z.-H. Deng, S.-W. Tang, D.-Q. Yang, M. Z. L.-Y. Li, and K.-Q. Xie. A comparative study on feature weight in text categorization. In *Advanced Web Technologies and Applications*, pages 588{597. Springer, 2004.

[9] J. Diederich and T. Iofciu. Finding communities of practice from user profiles based on folksonomies. In *Proc. of the 1st Int. Workshop on Building Technology Enhanced Learning solutions for Communities of Practice (TEL-CoPs'06)*, pages 288{297, 2006.

[10] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121{2159, 2011.

[11] G. Ganu, N. Elhadad, and A. Marian. Beyond the stars: Improving rating predictions using review text content. In *Proc. of the 12th Int. Workshop on the Web and Databases*, volume 9, pages 1{6, 2009.

[12] Y. Gu, Y. Sun, N. Jiang, B. Wang, and T. Chen. Topic-factorized ideal point estimation model for legislative voting network. In *Proc. of the 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 183{192, 2014.

[13] N. Hariri, B. Mobasher, and R. Burke. Query-driven context aware recommendation. In *Proc. of the 7th ACM Conf. on Recommender Systems*, pages 9{16, 2013.

[14] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proc. of the 22nd ACM Int. Conf. on Information and Knowledge Management*, pages 2333{2338, 2013.

[15] M. Lan, C. L. Tan, and H.-B. Low. Proposing a new term weighting scheme for text categorization. In *Proc. 2006 AAAI Conf. on Artificial Intelligence*, volume 6, pages 763{768, 2006.

[16] M. Lan, C. L. Tan, J. Su, and Y. Lu. Supervised and traditional term weighting methods for automatic text categorization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 31(4):721{735, 2009.

[17] Y. Li, J. Nie, Y. Zhang, B. Wang, B. Yan, and F. Weng. Contextual recommendation based on text mining. In *Proc. of the 23rd Int. Conf. on Computational Linguistics: Posters*, pages 692{700, 2010.

[18] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73{105. Springer, 2011.

[19] Q. Luo, E. Chen, and H. Xiong. A semantic term weighting scheme for text categorization. *Expert Systems with Applications*, 38(10):12708{12716, 2011.

[20] H. Mak, I. Koprinska, and J. Poon. Intimate: A web-based movie recommender using text categorization. In *IEEE/WIC Int. Conf. on Web Intelligence*, pages 602{605, 2003.

[21] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proc. of the 7th ACM Conf. on Recommender Systems*, pages 165{172, 2013.

[22] H. B. McMahan and M. Streeter. Adaptive bound optimization for online convex optimization. In *Proc. of the 23rd Annual Conf. on Learning Theory (COLT)*, 2010.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[24] M. Schmidt, G. Fung, and R. Rosales. Optimization methods for l1-regularization. *University of British Columbia, Technical Report TR-2009*, 19, 2009.

[25] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proc. of the 23rd ACM Int. Conf. on Information and Knowledge Management*, pages 101{110, 2014.

[26] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proc. of the Companion Publication of the 23rd Int. Conf. on World Wide Web Companion*, pages 373{374, 2014.

[27] P. Soucy and G. W. Mineau. Beyond tfidf weighting for text categorization in the vector space model. In *Proc. 19th Joint Int. Conf. Artificial Intelligence*, volume 5, pages 1130{1135, 2005.

[28] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11{21, 1972.

[29] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proc. of the 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 448{456, 2011.

[30] J. Wang and D. Hardtke. User latent preference model for better downside management in recommender systems. In *Proc. of the 24th Int. Conf. on World Wide Web*, pages 1209{1219, 2015.

[31] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. 1997 Int. Conf. Machine Learning*, volume 97, pages 412{420, 1997.