Journal of Machine Learning Research () Submitted ; Published

# DataRank: A Framework for Ranking Biomedical Datasets

**Editor:**


## Abstract

a


## 1. Introduction

Over the past decades, a wide range of datasets including clinical, gnomic, imaging, behavioural, etc. is created in biomedical research, and there is no unified and systematic way for retrieving them. Although several repositories(e.g. DBGap, TCGA, GEO, etc.) has been established NIH (2015), unfortunately like many cases of big-data applications with a "diminishing return", retrieval of datasets is getting harder as the number of repositories and active datasets increases. In fact, the current indexing and searching system suffers from a number of problems, including

(I) **Indexing** of datasets rely on human resources, which is a costly and noisy precess.

(II) **Searching** between and within the repositories is not well developed, and user should know the dataset and repository prior to search.

(III) **Integration** of the repositories is widely ignored and indexing, searching and maintenance of each repository is being done independently.

(IV) **Privacy**, only contain public datasets, which are only a small fraction of all the datasets is being generated. (Not sure, should check it out later)

(V) **Ranking** of search results of each repository is being done naively based matching of the query to the dataset's limited metadata.

(VI) **Recommendation** is now is a part of most of modern information retrieval systems with large amount of users and items, but it has not been considered for datasets and research scientists.

In this paper, we provide a solution for problems (I)-(VI), respectively by

(i) **Crawling** the PMC articles for patterns and rules provided by NIH to discover datasets directly from papers.

(ii) **Integrating** all the discovered datasets into one unified repository.

(iii) **Providing**, general information for both private and public datasets.

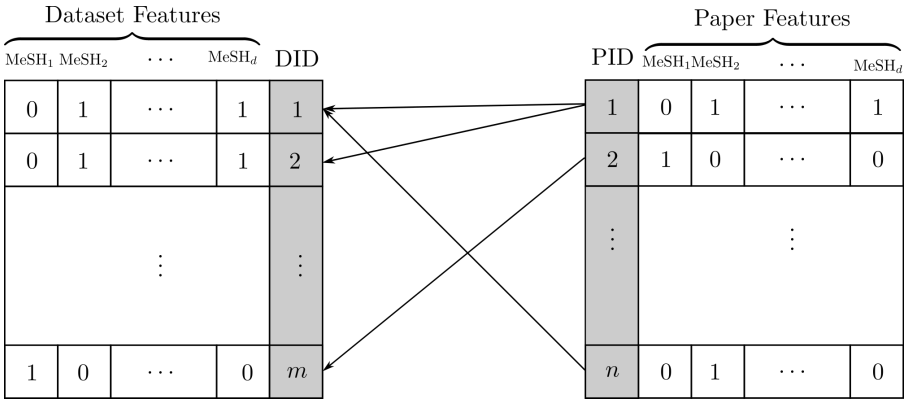


Figure 1: A bipartite graph between $m$ datasets and $n$ papers with MeSH terms as their features. PID and DID are Paper ID and Dataset ID respectively where existence of an edge between a paper and a dataset implies that dataset is used in that paper.

(iv) **Ranking** datasets using different informative features including MeSH, number of citations, etc.

(v) **Recommendation** of other datasets beyond current search results to make related datasets even more discoverable.

It should be noted that the task of crawling and indexing datasets is a non-trivial and independent task to others, so the methods and techniques for implementing it are outside of the scope of this manuscript. Therefore, in the rest of this paper we mainly focus on developing efficient methods for (ii)-(v) and for the crawling task we take an simple method as follows: By creating regular expressions for all the citation rules provided by NIH for each repository we discovered more than 20,000 datasets out of more than 1 million PMC full text articles and created a bipartite graph between dataset ids and paper ids. The product of crawling process is actually a bipartite graph, though incomplete, between articles and datasets. In the rest of this paper we describe our methodology to process this bipartite graph, Figure 1.

The rest of this paper is organized as follows: in section 2 we first review existing methods for enhancing information retrieval systems and then in section 3 we present out methods for the aforementioned problems. We outline implementation remarks and experimental study in section 5 and finally we make conclusions and state possible future works in section 6.

## 2. Background

Probably, information retrieval is the point when hypes about "Big Data" are came into realization, because as repositories of data are getting larger, effective information retrieval is getting much complicated. For example, in order to close the gaps between the largest information retrieval system in NIH, PubMeD, and the rate of growth of the repository, a number of techniques has been including

- relevance ranking Lu et al. (2009b), where documents are ranked according to Term-Frequency Inverse-Document-Frequency (TF-IDF) relevance to the free-text query.
- query expansion Lu et al. (2009a), which the (mappable) tokens in the query mapped to a MeSH term using Automatic Term Mapping (ATM) NLM (2015) and query is augmented with a set of MeSH terms,
- relevance feedback Yu et al. (2009), where user feedbacks in previous ranking results are used to improve ranking results,
- etc. Lu (2011)

developed to enhance retrieval of articles. By default, PubMed uses a hybrid approach to process a free-text query NCBI (2014) by expanding the query using Automatic Term Mapping (ATM) NLM (2015) and sorts the results in a reverse chronological order based on the document TF-IDF relevance to the query.

However, in the machine learning community tools and learning algorithms for ranking objects has been well developed Chapelle and Chang (2011). For example, given a labeled dataset, i.e. having the rankings for every document, RankSVM (Joachims, 2002) trains a nonlinear model to predicts future ranking based on training set. RefMed Yu et al. (2009) uses RankSVM to learn a model for each query by incorporating user feedbacks as "training dataset labels" into training process. Unfortunately, like other ranking algorithms, RankSVM only works well when enough labeled training samples (user feedbacks) are provided to the learning algorithm, which is very unlikely in real world situations, to expect a user to rate all the search results.

During past two decades a wide range of methods for creating recommender systems has been developed Adomavicius and Tuzhilin (2005) to address the problem of unknwon user ratings for majority of results. For instance, collaborative filtering is a well-known algorithm for estimating user ratings for set of items based on their similarities to set of rated items Koren et al. (2009).

Roughly speaking, Datarank implements, relevance ranking, query expansion and relevance feedback using a Bayesian approach without requiring a training process.

## 3. Methodology

The DataRank algorithm is an online algorithm which updated its predictive model after receiving explicit and implicit feedbacks from users. Specifically, we resort to incorporate user feedbacks explicitly you giving an option to the user to rate the search results, and we interpret user clicks as implicit feedbacks. Therefore, for every new user, the algorithm works in an offline manner and after receiving user feedbacks it updates the model to present user specific rankings. In the following parts we first describe the features and then explain our DataRank model offline ranking and finally extend offline DataRank to online setting.

### 3.1 Features

In this paper we use the corresponding set of MeSH terms as features for each articles and use the bag-of-words representation for representing documents. More precisely, the corpus

$n$ articles is represented by a $d \times n$ binary matrix, $M \in \{0,1\}^{d\times n}$. Similarly we define the matrix of $X \in \{0,1\}^{d\times m}$ for representing $m$ dataset via bag-of-words of MeSH terms, where $\mathbf{x}_i$ is the $i^{th}$ column of $X$, is $i^{th}$ document in the corpus. Also, for each dataset, the dataset label $\mathbf{y}_i$, is the dataset identifier which is a categorical variable, i.e., $\mathbf{y} \in \{1 \cdots m\}^m$, the bipartite graph is represented via the adjacency matrix $A \in \{0,1\}^{n\times m}$. Since we are considering binary features for both documents and datasets, the dataset features can be readily obtained from the document features and adjacency matrix:

$$X = \min(MA, 1) \tag{1}$$

where $\min(\cdot)$ is a elementwise operator over its matrix argument.

## 4. Ranking

### 4.1 General Ranking

### 4.2 Offline Ranking

For the problem of ranking we consider a probabilistic approach and propose a graphical model to specify dependencies between random variables in the model. Therefore, $\mathbf{x}, \mathbf{y}$ should henceforth be understood as realizations of the corresponding random variables. We also introduce another random variable $\mathbf{q}$ for search queries over the same sample space as $\mathbf{x}$, i.e., MeSH terms which $\mathbf{q} \in \{0,1\}^d$. Finally, $\mathbf{c}$ is an (observed) random variable which defines a prior over the labels $\mathbf{y}$. The graphical model shown in Figure 2-(a). It should be noted that, in any case variables $\mathbf{x}$ (dataset features) and $\mathbf{c}$ (datasets prior) are observed and we never need their marginal distributions and therefore we only consider the query $\mathbf{q}$ as evidence.

In this model, the problems of ranking for a given query $\mathbf{q}$ is to find the posterior distribution for the dataset labels given evidence. More precisely the posterior distribution

$$\Pr(\mathbf{y}|\mathbf{q}, \mathbf{c}, \mathbf{x}) \propto \Pr(\mathbf{y}|\mathbf{c}) \Pr(\mathbf{q}|\mathbf{x}) \tag{2}$$

full specifies the ranking, where the posterior distribution is expanded according to the graphical model Figure 2-(a). Since, $\Pr(\mathbf{y}|\mathbf{c})$ is not function of evidence, we can legitimately consider it as dataset-prior and consider $\Pr(\mathbf{q}|X)$ as query-likelihood.

In the standard statistical modelling procedures, the next step is to specify dataset-prior and query-likelihood distributions.

**Query-likelihood.** Since the binary feature vector $\mathbf{x}$ and query $\mathbf{q}$ are representation of sets, using Vann diagram, we can easily compute the likelihood

$$\Pr(\mathbf{q}|\mathbf{x}) = \frac{\Pr(\mathbf{q}, \mathbf{x})}{\Pr(\mathbf{x})} = \frac{|\mathbf{q} \cap \mathbf{x}|}{|\mathbf{x}|} \tag{3}$$

where set operations are applied to the set representation of the binary vectors.

However, this probability does not account for mismatches in partial matching of terms in the query and features of each dataset, and this leads to the phenomena that two queries

whit the same number of matches but different number of mismatches have the same probability. To remedy this crucial problem, we can add the number of mismatches to the denominator, which is equivalent to condition on $\mathbf{x} \cup \mathbf{q}$

$$\widehat{\Pr}(\mathbf{q}|\mathbf{x}) \propto \Pr(\mathbf{q}|\mathbf{x} \cup \mathbf{q}) = \frac{|\mathbf{q} \cap \mathbf{x}|}{|\mathbf{q} \cup \mathbf{x}|} \tag{4}$$

the value of (5) is known as Jaccard index or Tanimoto or Jaccard coefficient Manning et al. (2008). Thus for the query likelihood we have

$$\Pr(\mathbf{q}|\mathbf{x}) = \frac{\widehat{\Pr}(\mathbf{q}|\mathbf{x})}{\sum_{i=1}^{n} \widehat{\Pr}(\mathbf{q}|\mathbf{x}_i)} \tag{5}$$

**Dataset-prior.** Instead of learning a prior for datasets using empirical Bayesian methods, we simply chose to propose a reasonable subjective prior for the datasets. Basically, the better way to think about prior is to imagine the posterior distribution in a scenario where there is no evidence available, which is equivalent to say that what is the best ranking of datasets if do not have a query. There are many ways to answer this question, but a reasonable approach is to sort them based on their general popularity. Interestingly, we can quantitatively specify the popularity for the datasets by taking into account of how many citations they have in the training dataset, i.e., $\mathbf{c} = \mathbf{1}^T A$. More precisely, we can write prior as

$$\Pr(\mathbf{y}|\mathbf{c}) = \frac{\mathbf{c}}{\mathbf{c}^T \mathbf{1}} = \frac{\mathbf{1}^T A}{\mathbf{1}^T A \mathbf{1}} \tag{6}$$

where $\mathbf{1}$ is the vector of ones of corresponding dimension. So far, we have fully specified the prior and likelihood and hence posterior for the offline model and the ranking of datasets amounts to merely sort datasets decreasingly according to their posterior distribution.

### 4.3 Online Ranking

In this part we extend the offline-DataRank to the online setting by incorporating user feedback into ranking. For this porous, we propose a new model, Figure 2-(b), which introduces *incomplete* user ratings $\mathbf{r} \in \{0 \cdots k\}^m$, and the dataset online-labels $\mathbf{z}$, where $k$ is the number of different state of ratings for each result in the ranking and 0 is used to denote unknown values in the ratings. Thus, in the ranking process, ratings $\mathbf{r}$ are initialized with zero value and at each epoch users rates the search results and updates the $\mathbf{r}_t$. As shown in the graphical model, the online-labels depend on user feedback but, (offline) dataset labels $\mathbf{y}$ do not.

Similar to offline method, the task is to compute the posterior

$$\Pr(\mathbf{z}|\mathbf{r}, \mathbf{y}, \mathbf{c}, \mathbf{q}, \mathbf{x}) \propto \Pr(\mathbf{z}|\mathbf{r}) \Pr(\mathbf{y}|\mathbf{c}, \mathbf{q}, \mathbf{x}) \tag{7}$$

where the factorization induced by the graphical model Figure 2-(b) and in this model evidence is incomplete user rating $\mathbf{r}$. Interestingly the offline posterior $\Pr(\mathbf{y}|\mathbf{c}, \mathbf{q}, \mathbf{x})$ can be regarded as prior in this model, which implies that without any evidence, user ratings, the online posterior is exactly equal to its prior, i.e., offline posterior, which makes a perfect sense.

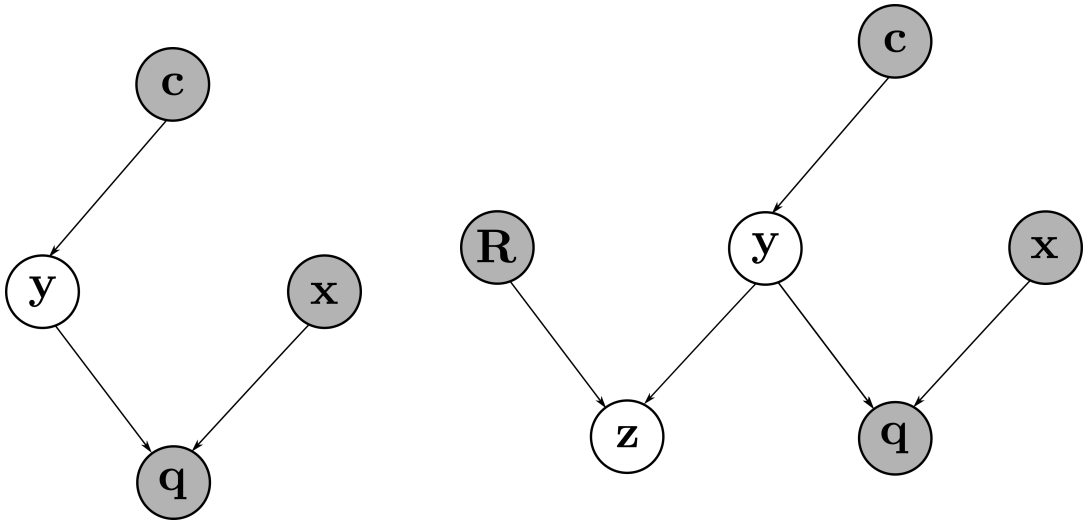


Figure 2: Probabilistic graphical models for offline (a) and online (b) methods for ranking datasets. The shaded nodes are observed variable, arrow implies conditional dependence and rectangles are short hand for replication of the inside nodes. For details see text.

However, the ratings $\mathbf{r}$ is incomplete, i.e. contain zero values, and in order to specify the likelihood $\Pr(\mathbf{z}|\mathbf{r})$ we first need to estimate the unknown values of user ratings. Once we estimate all the unknown values in the user rating vector $\mathbf{r}$ we can readily compute the likelihood $\Pr(\mathbf{z}|\mathbf{r})$ by normalizing $\mathbf{r}$

We use collaborative filtering Su and Khoshgoftaar (2009) to estimate unknown values of user ratings for the datasets that has not been rated yet. Collaborative filtering methods generally work by constructing a similarity matrix between items (dataset), and defining a method for finding a set of neighbours of each item, and then computes the rating of undated items as weighted linear combination its rated neighbours, where the weights are proportional to similarity between items.

Since the rating vectors is extremely sparse we choose to define the method for finding neighbours to return all the rated items, to use all the user rating information. Thus, the key step remains to perform collaborative filtering is to define a similarity measure between datasets. Regarding, the binary MeSH representation of datasets, we opt to use Tanimoto kernel Ralaivola et al. (2005) between datasets for measuring similarity between datasets

$$K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{k}_\cap(\mathbf{x}_i, \mathbf{x}_j)}{\mathbf{k}_\cap(\mathbf{x}_i, \mathbf{x}_i) + \mathbf{k}_\cap(\mathbf{x}_j, \mathbf{x}_j) - \mathbf{k}_\cap(\mathbf{x}_i, \mathbf{x}_j)} \tag{8}$$

where $K$ is a $m \times m$ symmetric positive definite matrix, $\mathbf{k}_\cap(\mathbf{x}_i, \mathbf{x}_j) = \frac{|\phi(\mathbf{x}_i) \cap \phi(\mathbf{x}_j)|}{|\phi(\mathbf{x}_i) \cup \phi(\mathbf{x}_j)|}$ is intersection kernel Gärtner et al. (2006) between $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\phi(\cdot)$ is a general nonlinear feature function.

Having a similarity matrix between datasets, using collaborative filtering we can readily fill the unknown values in the completed rating vector $\mathbf{R}$

$$\mathbf{R}_i = \begin{cases} \mathbf{r}_i, & \mathbf{r}_i \neq 0 \\ \mathbf{r}^T K_i, & \mathbf{r}_i = 0 \end{cases} \tag{9}$$

where $K_i$ is the $i^{th}$ column of the kernel matrix. Having computed the completed rating vector, to compute the likelihood, we only need to normalize $\mathbf{R}$

$$\Pr(\mathbf{z}|\mathbf{r}) = \frac{\mathbf{R}}{\mathbf{1}^T\mathbf{R}} \tag{10}$$

Using (5) and (10) we can fully specify the online prediction posterior (7) and therefore, online ranking for the DataRank algorithm.

## 5. Experimental Study

### 5.1 Implementation

DataEngine is deployed on an Ubuntu 13.04 system with Intel Xeon X5650 2.67GHz and 4 GB RAM. Nginx 1.4.6 is used for static request and django 1.7.4 is used for dynamic request. uwsgi is used as an application server hosting django models, which is also used as an interconnector between nginx and django. Nginx is chosen for a consideration of future high traffic. A django/python combination is more extensible for future recommendation algorithm research with assistance of numpy and scipy.

#### 5.1.1 System Architecture

A brief system architecture is shown below. The abstract architecture is a standard MVC model.

All user interaction with UI is passed to interface layer for futuer response. All static request is passed to nginx and will be responsed immediately. Dynamic request is passed to model layer. After processing and computation, a response will be given back to user via same path.

At system entry, there is a login management component. User could register, login or stay anonymous. When user either registered or logined, this user will be given a unique identifier. For identified user, all history action could be retrieved for recommendation and all new actions will be recorded as preferences. A search component works at next step. User insert one or a combination of keywords, a list of datasets will be shown on display via an offline ranking algorithm. Keywords will be passed to backend and be processed by a set of function and algorithms, and then a list of datasets is given with a keywords related order. The algorithm will be introduced on section **??**.

Display component plays as an important connector among different models and functions. User gives rates and comments on this step. When new user related information is added, a refresh component is activated. It will pass all new infomation to backend, and a backend online learning algorithm will collect all user historic and current preferrences and then give a new ranking based on updated information. This infomation will be shown on display component.

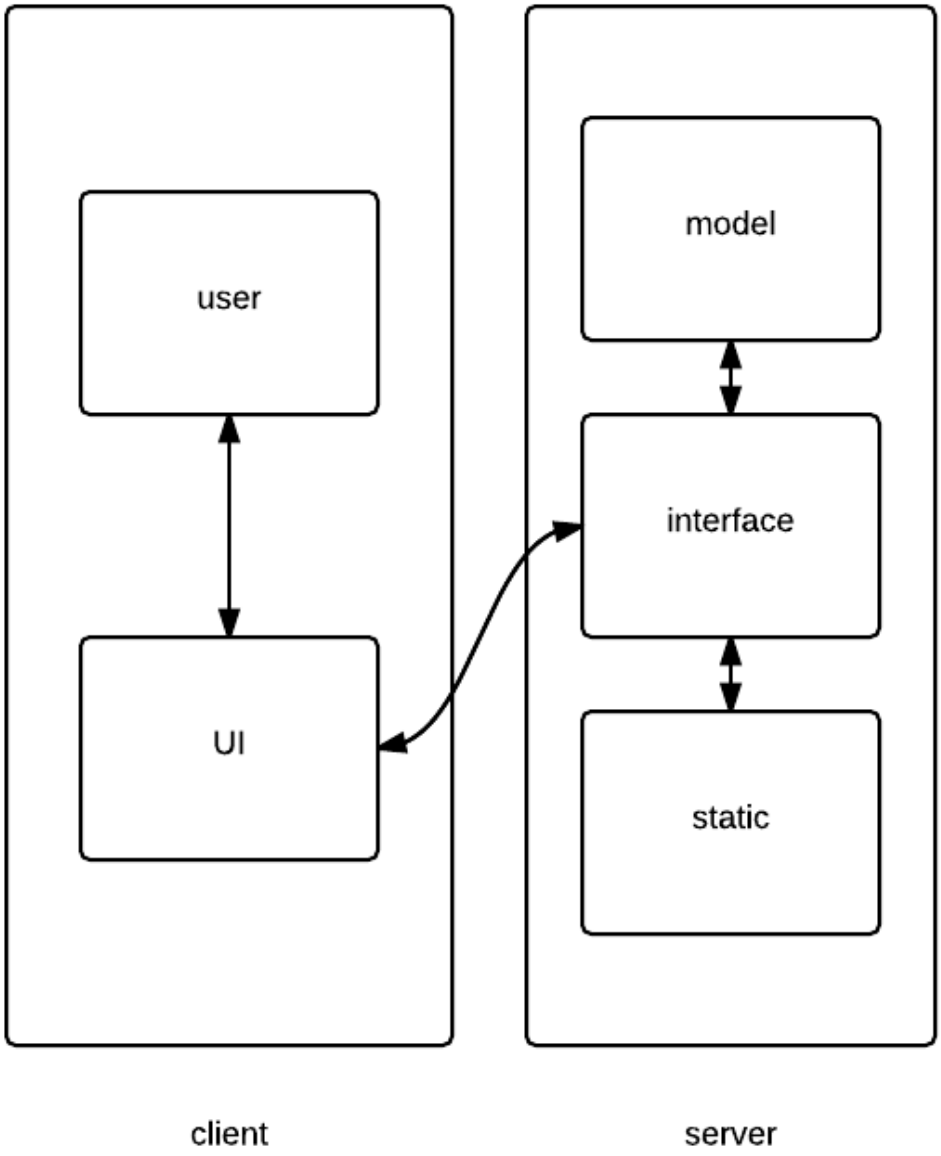


Figure 3: System Components

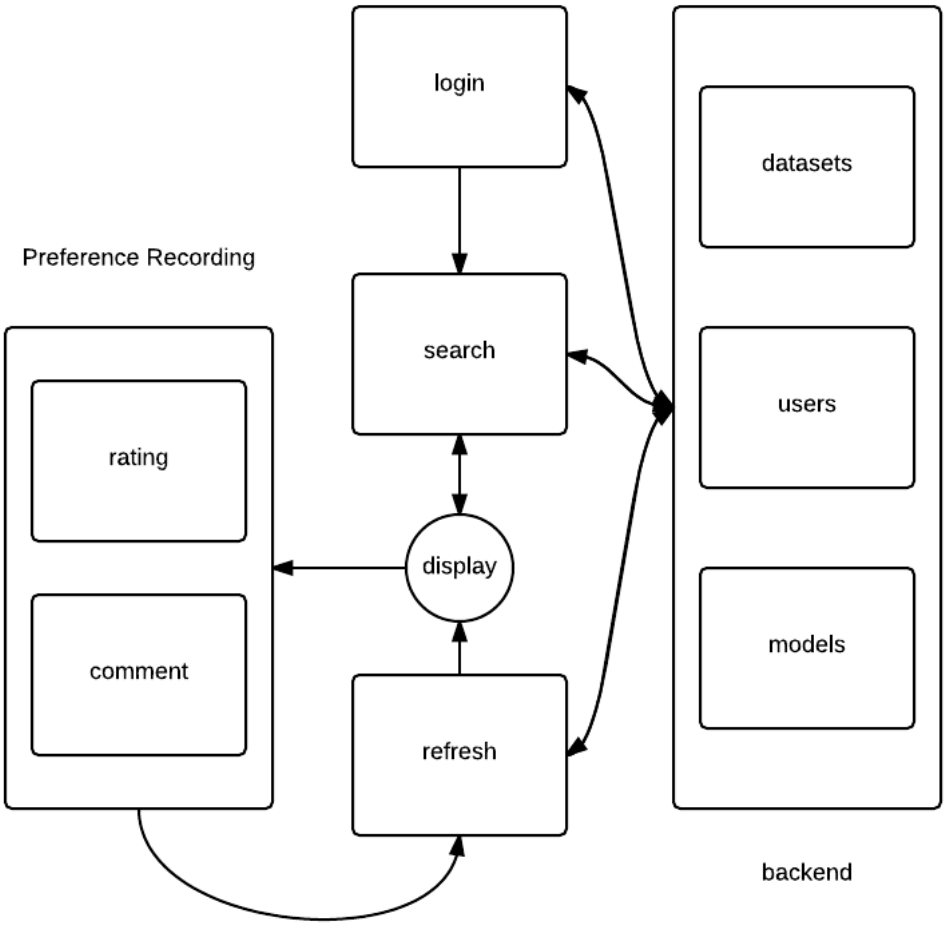
login
datasets
Preference Recording
search
users
rating
display
models
comment
refresh
backend

Figure 4:

#### 5.1.2 Searching Pattern

Searching bar is the first function component faced by user. It could support several basic pattern. Current preffered keywords are MeSH terms. However there are punctuations included in existing MeSH terms text. We choose semicolon as splitter. When a string of keywords is passed to backend component, a grammar parser component is activated. it first changes keywords to lower letters, splits keywords by semicolons, and then remove travial characters for each keyword. A set of cleaned keywords will be passed to next model for further processing.

Now, dataset specific searching is supported. User may have a destination data source. Thus supporting a data resource fixed searching pattern is necessary. We use an at sign (@) on the end of keywords, and dataset name is given after it. This dataset name is stored, and all future refreshing will only occur under this scale. A simple example is (DNA;Genes@GeneBank).

#### 5.1.3 Refresh

Refreshing is one of the most important feature in this system. Refreshing is majorly session based. A session is define as the a period of actions with one fixed query. If a new query is requested, a new session will be created. In a session, user could give ratings and update preferences unlimitedly. Everytime, when new information is given, an update of recommendation would be activated. A standard session work flow is: a user input keywords, and a list of datasets are given. After considering, user gives ratings to some of them and trigger refresh component. Relearning user preferences will be executed and a list of new ordered data will be given.

#### 5.1.4 Ratings and Comments

Ratings and comments are two important feature for online learning, which is introduced on section **??**. As users may change their ideas frequently, ratings are stored on browser session, which provide better security than traditional cookies. Those ratings will be passed to server side and stored when refresh action is triggered. Each rating is a pair of dataset ID and rating score. Comments are also passed to server side with dataset id, and user related information will be added later. All stored ratings and comments include time information and user information and even keywords. All this information could be used for future ranking algorithm development.

#### 5.1.5 Assistant Information

In order to help user evaluate and understanding datasets orders better, we offer several parameters, such as posterior likelihood value (online ranking results) and raw prior probability. Counts of citations are also provided. All these statistics could be used as an compliment of metadata in order to have a better understanding of datasets.

### 5.2 Experiments

Unfortunately for such a novel problem, to this time, there is no labaled dataset which ground truth ranking of datasets are known. The lack of labeled dataset is not only a

challenge for learning a predictive model, but also makes evaluation of the model more tricky. Yet, we set to perform two set of experiments to show that

1. by getting user feedbacks, DataRank actually learns about user preferences
2. DataRank provides user-specialized results for each user

In the first set of experiment we aim to show how well DataRank predicts user ratings, by comparing DataRank by gold-standard ranking, the ranking in which is presented by DataRank given current features and actual user ratings, i.e. the best possible ranking using DataRank and full labels Shalev-shwartz (2007). Since the ranking is linearly related to its user rating, we measure the discrepancy between DataRank ranking and gold-standard DataRank rankings by just measure the *loss* between the predicted value of rating and and the actual user rating. For instance, suppose at $t^{th}$ step user rates $j^{th}$ dataset, we can easily compute the loss

$$\mathbf{l}^{(t)} = |\hat{\mathbf{r}}_j^{(t-1)} - \mathbf{r}_j^{(t)}| \tag{11}$$

where $\hat{\mathbf{r}}$ is the predicted user rating in previous step. Since the loss is quite instantaneous we smooth it by defining *regret* Shalev-shwartz (2007) as average loss until know

$$\mathbf{R}^{(t)} = \frac{1}{t}\sum_{k}^{t} |\hat{\mathbf{r}}_j^{(k-1)} - \mathbf{r}_j^{(k)}| \tag{12}$$

we computed regret for each time for 5 different subjects and the rults for 20 feedbacks are shown in Figure 5. It can be seen that regret has a global decreasing meaning that throughout time, DataRank make more accurate predictions.

For the second set of experiments we set different subjects search for the same query and rate the results according to their preferences. Then we in this experiment we show that DataRank is able to present user special results. For this case user A and B search for the same query, e.g. "Mice", and $t = 1 \cdots T$ we compute the *Kappa statistic* Viera et al. (2005) for measuring disagreement between users. For example we cam compute the ratings between users $A$ and $B$ at each time

$$\kappa_{A,B}^{(t)} = \kappa(\mathbf{r}_A^{(t)}, \mathbf{r}_B^{(t)}) \tag{13}$$

We again run this experiment for 5 subjects using the same query and computed parwise kappa score for all the users and depicted their average throughout time in Figure 6. Figure 6 shows that even for the same query users tend to have disagreement which leads to different rating for each user.

## 6. Conclusions

(Xiaoqian)

## References

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

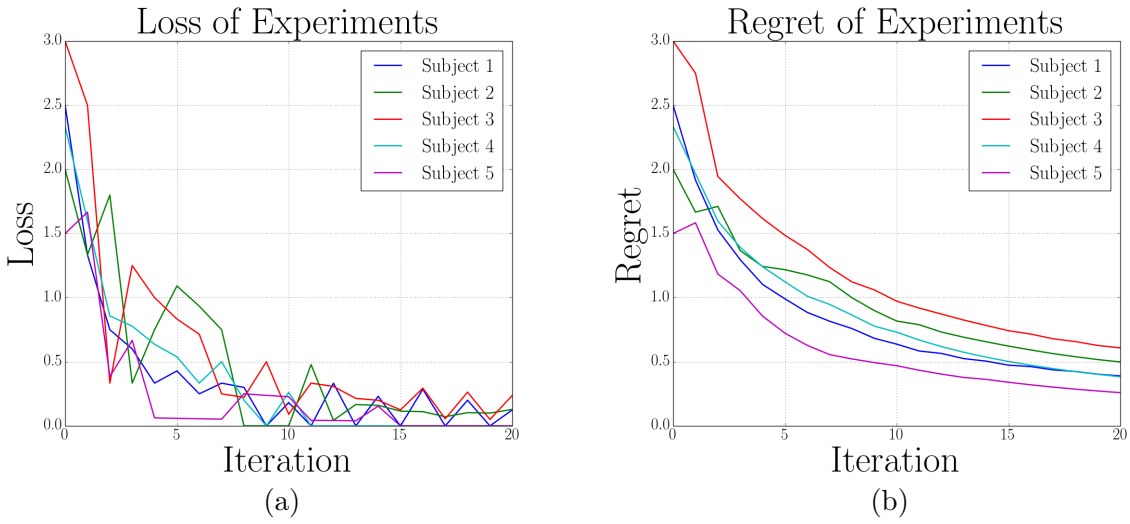


Figure 5: Loss and Regret of predicted ratings throughout time.

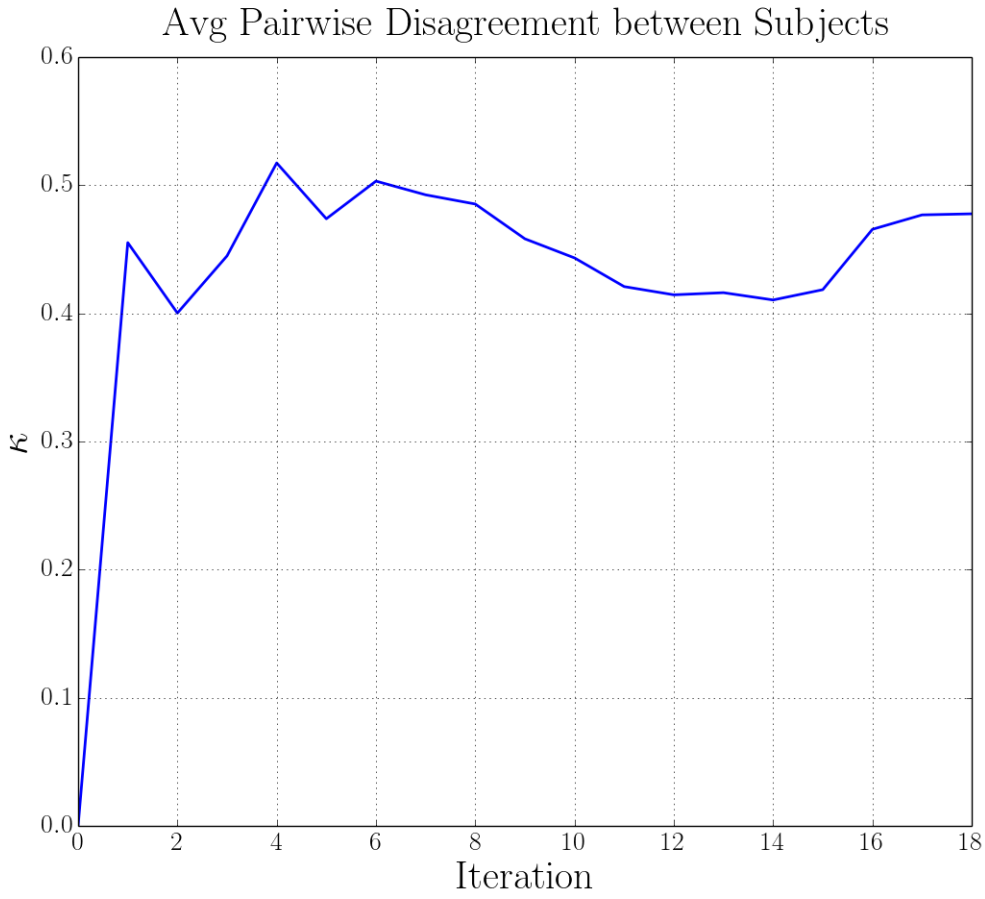


Figure 6: Average disagreement between users.

Olivier Chapelle and Yi Chang. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research - Proceedings Track*, 14:1–24, 2011.

Thomas Gärtner, Germany Quoc V Le, and Alex J Smola. A Short Tour of Kernel Methods for Graphs. Technical report, Stanford University, 2006.

T. Joachims. Optimizing Search Engines Using Clickthrough Data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 133–142, 2002.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

Zhiyong Lu. PubMed and beyond: a survey of web tools for searching biomedical literature. *Database*, 2011:baq036, 2011.

Zhiyong Lu, Won Kim, and W John Wilbur. Evaluation of Query Expansion Using MeSH in PubMed. *Information retrieval*, 12(1):69–80, 2009a.

Zhiyong Lu, Won Kim, and W John Wilbur. Evaluating Relevance Ranking Strategies for MEDLINE Retrieval. *Journal of the American Medical Informatics Association : JAMIA*, 16(1):32–36, July 2009b.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press, 2008.

NCBI. PubMed Help, 2014. URL http://www.ncbi.nlm.nih.gov/books/NBK3827/.

NIH. NIH Data Sharing Repositories, 2015. URL http://www.nlm.nih.gov/NIHbmic/nih_data_sharing_repositories.html.

NLM. Automatic Term Mapping, 2015. URL http://www.nlm.nih.gov/bsd/disted/pubmedtutorial/020_040.html.

Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

Shai Shalev-shwartz. *Online learning: Theory, algorithms, and applications*. PhD thesis, The Hebrew University of Jerusalem, 2007.

Xiaoyuan Su and Taghi M Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, 2009:4:2—-4:2, January 2009.

Anthony J Viera, Joanne M Garrett, and Others. Understanding interobserver agreement: the kappa statistic. *Fam Med*, 37(5):360–363, 2005.

Hwanjo Yu, Taehoon Kim, Jinoh Oh, Ilhwan Ko, and Sungchul Kim. RefMed: relevance feedback retrieval system fo PubMed. In *Proceedings of the 18th ACM conference on Information and knowledge Management*, pages 2099–2100. ACM, 2009.