“Tell Me More”: Finding Related Items from User Provided Feedback.

Jeroen De Knijf[1], Anthony Liekens[2], and Bart Goethals[1]

[1] Department of Mathematics and Computer Science, Antwerp University
[2] VIB Department of Molecular Genetics, Antwerp University

Abstract. The results returned by a search, datamining or database engine often contains an overload of potentially interesting information. A daunting and challenging problem for a user is to pick out the useful information. In this paper we propose an interactive framework to efficiently explore and (re)rank the objects retrieved by such an engine, according to feedback provided on part of the initially retrieved objects. In particular, given a set of objects, a similarity measure applicable to the objects and an initial set of objects that are of interest to the user, our algorithm computes the k most similar objects. This problem, previously coined as ’cluster on demand’ [10], is solved by transforming the data into a weighted graph. On this weighted graph we compute a relevance score between the initial set of nodes and the remaining nodes based upon random walks with restart in graphs. We apply our algorithm “Tell Me More” (TMM) on text, numerical and zero/one data. The results show that TMM for almost every experiment significantly outperforms a k-nearest neighbor approach.

1 Introduction

The increasing usage of information technology to store, process and exchange data the last decades, has resulted in the availability in enormous amounts of data. One of the major challenges in modern society is to efficiently find and retrieve the desired information. Although a vast amount of efficient and powerful tools are available to assist in the daily information need, the number of potentially interesting objects retrieved is in general tremendous and unrelated to the user needs. For a user it is an challenging task, if the goal can be accomplished at all, to filter out all or a great deal of the useful objects. It is our belief that the main problem is not the expressiveness of the query language, but that it is the case that user does not have enough apriori knowledge of what is interesting and what is not. In order to enhance the retrieval capabilities of modern search, mining and database engines, we argue that an interactive framework that allows for simple and intuitive user directed exploration of the relevant objects is needed. A particularly intuitive and appealing approach is where a user wants to find objects that are similar to some manually inspected objects selected for his specific interest.

For example, consider a text retrieval engine. The initial results of the user query contains about 10,000 relevant documents. After manually inspecting the ten highest ranked documents, the user marked three of them as relevant. In the framework we propose, the initial list of 10,000 documents is reordered, such that the most similar documents to the three interesting documents are ranked higher. In this reordered list, the user can again manually inspect the top ten documents and mark the interesting ones. As a second example consider a binary database, where each record consists of an unique customers id and all the goods purchased by the customer over the past year. Suppose that a marketeer is interested in specific groups of customers, and tries to distill interesting groups based on their purchasing behavior. A common approach to achieve this, is by using a frequent itemset mining algorithm [1]. However, once some interesting patterns are discovered and the initial group of interesting customers is identified, the marketeer wants to find similar customers. That is, clients that purchased more or less the same products. However, given an initial frequent itemset, the number of frequent itemsets with a few different items can be overwhelming. Our approach can efficiently compute the k most similar customers (i.e. transactions) with respect to an initial customer group (i.e. set of transactions) of interest. The previous examples contained situations where we proposed to improve the results of a search or datamining engine by using user feedback, however one can also think off different situations where the most related objects on itself might be of interest. For instance, consider a clinical trial where some of the patients are given a drug to cure a certain disease. A natural question is, which of the patients that did not participate at the trial are alike the patients that responded positive to the drug.

In this paper we introduce an efficient and interactive framework Tell Me More (TMM) to solve the aforementioned problems. We transform the problem into a weighted graph, where the weight of the link between object i and j is the similarity between i and j. On this graph we compute a relevance score between a set of nodes and the remaining nodes based upon random walks and random walks with restart in graphs. The main contribution of the paper are as follows:

– We propose a framework to rerank the results of a search, database or datamining engine based upon feedback provided by the user.
– We propose an algorithm to find the most related nodes based upon an non-empty initial set of nodes.
– We propose a relatedness score that corrects for objects that are apriori highly related to many objects (hubs).
– TMM is independent of any specific data type, i.e. we demonstrate the usability on binary, text and numerical data.
– Thorough experimental evaluation on real-life data shows that TMM significantly outperforms a k-nearest neighbor approach. Moreover, the experimental results shows that the adjustment of the score in order to compensate for objects that are apriori similar to many objects in the dataset, leads, in a substantial number of settings, in significant better results.

The remainder of this paper is organized as follows. In the next section we formally define the problem. In section 3 we discuss related work. In the following section we discuss random walks and random walks with restart and present our algorithm: Tell Me More. In section 5 we discuss the experimental setup and similarity measures used. Moreover, we report on the results obtained by TMM on various benchmark datasets and compare these with the results obtained by the k-nearest neighbor approach and by a ranking approach which is solely based upon random walk with restarts. Finally, in the last section we draw conclusions and give directions for further research.

2 Problem Statement

Let D denote the data collection, that is D is the initial set of objects. The problem addressed in this paper is the following: given a (small) subset $S \subset D$ of the dataset, find the k most related objects to S in D, with k a user specified parameter. In order to conduct a proper evaluation and to avoid philosophical discussion of what is related, we assume that there exists a set of m class labels $C = \{c_1, \ldots, c_m\}$, such that every object in the dataset belongs to exactly one class. Moreover, every object $x \in S$ belongs to the same class. In this setting the most related objects, becomes the objects with the same class as the class of the objects in S.

Given a dataset, an initial set of interest S and a similarity function, the k-nearest neighbor approach selects the k objects of D that are most similar to the objects in S. In case S contains a single object, KNN returns the k objects with the highest similarity score. In case that S consists of multiple objects, then the similarity score for an object $j \in D$ is the average similarity with respect to all objects in S. Hence, KNN return the k objects that have the largest average similarity score with the objects in S. We will experimentally compare our approach with the KNN approach in section 5.

3 Related Work

The problem previously described can be seen as a special instance of clustering, that is, some part of an unknown cluster is given and the task is to complete the cluster. Because of this resemblance, the problem was previously coined by Ghahramani and Heller [10] as 'clustering on demand'. In this paper we adopt this terminology. However, in general the goal of clustering is to divide the data into multiple parts, where our goal is to partially complete one cluster. Moreover, clustering is generally performed unsupervised where in the clustering on demand setting we get some examples that provides hints of the class membership.

Under the assumption that the objects consists of binary features, Ghahramani and Heller [10] take a probabilistic Bayesian viewpoint to solve the clustering on demand problem. In particular, for each object, every feature is assumed to be drawn from a fixed Bernoulli distribution with parameter θ_j, where θ_j equals the probability that feature j is equal to one. Moreover, it is assumed that the objects are generated independently and identically. In this setting, the relevance score for an object $\mathbf{x}$ boils down to $: \frac{\Pr(\mathbf{x}|S)}{\Pr(\mathbf{x})}$. Ghahramani and Heller [10] show that the log score can be computed efficiently with a conjugate Beta prior on the Bernoulli model parameters. The main difference with our approach is that we do pose no restriction on the type of data.

Very similar to our clustering on demand problem, is the problem of finding similar web pages of the current web page [5, 12]. However, the problem mainly focus on finding good similarity measures for web pages. Once an appropriate measure is found, a standard k-nearest neighbor search is performed. A further difference with our approach is that their problem is defined to find only related pages to one web page, where we generalize to sets of objects.

Different types of random walks are commonly used in graph mining to measure the relatedness between nodes in a graph. For example, in the work by Tong and Faloutsos [22], where in a social network the goal is to find subgraphs that are 'best' connected with some given query nodes. In Pang et al. [18] the aim is, given a multimedia collection, to automatically assign keywords to multimedia objects. In Sun et al. [21] random walks are used to derive the most similar nodes and the most abnormal nodes on bipartite graphs. Also in the Information Retrieval field are random walks commonly used. For example, Craswell and Szummer [4] use a random walk on the bipartite clickgraph to adjust the ranking of the search engine. In order to generate a set of appropriate keywords for an online advertiser, Fuxman et al. [9] uses an approach where a bipartite query-click graph is constructed. Given a set of URLs that are of interest to the advertiser, their method suggest keyword for the URL based upon performing a random walk with absorbing states on the bipartite click graph.

This overview of related work is far from complete, however to summarize all work that is somewhat related demands a survey paper on itself. Especially in the area of Information Retrieval and Web Mining, there are many methods bases upon random walk with restarts to find related objects. Although TMM also uses random walks with restarts to rank objects, there are some major differences with the different methods that make use of random walks with restart. First, TMM find related objects based upon user provided feedback. Second, our framework is suitable for any type of data, as long as their is a similarity measure applicable between the objects. Finally, instead of solely using the score obtained by performing random walks with restarts, we also adjust the score for the hubinnes of the nodes. In section 5, we show that this adjustment for hubiness does result, in a substantial number of settings, in significant improvement.

4 Algorithm

In this section we will present our main algorithm. First we describe how our problem can be transformed to a weighted graph. Then we discuss random walks and random walks with restarts on graphs. Next we present our TMM algorithm. Finally, we give a straightforward k-nearest neighbor algorithm.

4.1 Graph Construction

The first step is to transform the problem into a graph based setting. A graph $G = \{V, E, \lambda\}$ is defined as an undirected weighted graph, where V is a set of vertices, E a set of edges (unordered pair of vertices) and λ a labeling function that assigns a weight from the interval $[0, 1]$ to each edge, i.e. $\lambda : E \rightarrow [0, 1]$. Given our dataset D and a similarity function $Sim(i, j)$ that gives for every pair of objects from D a similarity score between i and j in the $[0, 1]$ interval. A score of 1 meaning that the objects are identical, while a score of 0 implies that the objects are completely dissimilar. The graph G is constructed as follows: First, all objects i in the dataset D are added as nodes in the graph G. Second, for every unordered pair of vertices i and j, with $i \neq j$, and edge is added if $Sim(i, j) > 0$. Furthermore, the weight of the edge is equal to the similarity between i and j, that is $\lambda\{i,j\} = Sim(i,j)$. Note that, in general this graph will be dense. Specifically , this graph is often an (almost) complete graph. This is caused by the particular setting, where the dataset is the initial answer set from an search or datamining engine. Hence, the objects in the dataset are apriori related with each other.

4.2 Random Walk with Restarts

Informally, a random walk on a graph G can be described as follows: consider a random particle currently at node v in G, at each iteration the particle moves to a neighbor with a probability that is proportional to its edge weight. The steady state probability of a node w, denoted as $\mathbf{u}(w)$ states the probability that the particle visits this node in the long term. Intuitively, the steady state probability of node w states the importance of w in the graph. In our setting, $\mathbf{u}(w)$ defines how central w is in the graph. Under the assumption that a random walk on G is ergodic, it is well known that the steady state probability of a random walk on G satisfies Equation 1, with M_G the column normalized adjacency matrix of G.

$$\boldsymbol{u_{k+1}} = M_G \times \boldsymbol{u_k} \tag{1}$$

Like a random walk, a random walk with restart can informally be described as a particle currently at node v in G. Let $\boldsymbol{q}_S$ be the normalized vector where for each node in S the corresponding value in $\boldsymbol{q}_S$ is set to 1, while the remaining values of $\boldsymbol{q}_S$ are set to 0. Informally, $\boldsymbol{q}_S$ contains the restart nodes, with an uniform distribution over the nodes in S. Additional to moving at each iteration to one of its neighbors with a probability that is proportional to its edge weight, the particle has a probability c of moving uniformly at random to one of the restart nodes. The relevance score of node w with the set of nodes S ($\boldsymbol{u}_S(w)$) is then defined as the steady state probability of the random walk with restart. Under the assumption that a random walk on G is ergodic, the random walk with restart on a graph G satisfies Equation 2. With M_G the column normalized adjacency matrix of G, c the restart probability and $\boldsymbol{q}_S$ the restart vector.

$$\boldsymbol{u}_S = (1-c) \times M_G \times \boldsymbol{u}_S + c \times \boldsymbol{q}_S \tag{2}$$

In our specific setting, we work with undirected, non-bipartite and connected graphs. It trivially follows that a random walk on these graph is ergodic. Consequently, in the limit, random walks and random walks with restart converges to a steady state probability.

The steady state probability of a random walk with restart, can be easily computed by iteratively applying Equation 2, this method is also known as the power iteration method. The pseudo code to compute the steady state probability of a random walk with restart is given in Algorithm 1. Note that, Algorithm 1 can be trivially used to compute the steady state probability of a random walk. In this case the input parameter c should be set to zero. Also note that in the actual implementation, we do not have to construct the graph. Instead, a normalized version of the similarity matrix is sufficient to obtain the desired output. Further worth mentioning, is that different optimizations techniques are available to compute the steady state probability of a random walk on

Algorithm 1 **Random Walk with Restart**

```
Input: adjacency matrix $M_G$, restart probability $c$, set of restart nodes $S$
Output: $\boldsymbol{u}_S$
 1: initialize $\boldsymbol{q}_S \leftarrow 0$
 2: for all $i \in S$ do
 3:     $\boldsymbol{q}_S \leftarrow 1$
 4: end for
 5: normalize $\boldsymbol{q}_S$
 6: column normalize $M_G$
 7: while not converged $\boldsymbol{u}_S$ do
 8:     $\boldsymbol{u}_S \leftarrow (1-c) \times M_G \times \boldsymbol{u}_S + c \times \boldsymbol{q}_S$
 9: end while
10: return $\boldsymbol{u}_S$
```

graphs. In particular, Tong et al. [23] report significant speedups compared with the power iteration method.

4.3 Tell Me More

Most graph mining algorithms based upon random walks (for example [18]) solely uses the steady state probability as the relevance score between nodes. However, consider the following example: Given two nodes w and v, and a set of restart nodes S. Suppose that $\boldsymbol{u}_S(w) = 0.2$ and $\boldsymbol{u}_S(v) = 0.3$, moreover $\boldsymbol{u}(w) = 0.1$ and $\boldsymbol{u}(v) = 0.6$. Hence, the apriori relevance of node v is much higher than the relevance score of node v with respect to the set of nodes S. In fact the initial set S harms the importance of node v, while node w becomes far more important due to the initial set S. But nevertheless, when only using the steady state probability of the random walk with restart, v is preferred above w. In our Algorithm, we take the prior importance of a node into account to adjust the score of the steady state probability. In particular, for an initial set S the score of a node v is determined by

$$\boldsymbol{u}_S(v) \times \frac{\boldsymbol{u}_S(v)}{\boldsymbol{u}(v)}.$$

Intuitively, this adjustment lowers the probability of objects that are similar to most other objects, however the prior importance of a node is not completely neglected. The restart probability parameter (needed in Algorithm 1) was determined experimentally. We found that a broad range of values (between 0.1 and 0.99) delivered almost equally good results. However, the optimal setting for all experiments was when the restart probability was set to 0.99. Note that, in this setting the influence of the nodes in the restart vector is maximal. Moreover, a restart probability of 1 is pointless, because then the steady state probabilities of nodes that are not in the restart vector is equal to zero. The TMM algorithm is given in Algorithm 2. The input parameters are the dataset D, the initial objects of interest S and the number of related objects (i.e. k) a user is interested in. The first step is to create the adjacency matrix M_G. That is, the adjacency matrix representation of the graph G, constructed from D. Note that, in this step we assume that the similarity function between objects in the dataset is available. In the next steps the steady state probability of the random walk with restart and the random walk is computed. This step is performed by Algorithm 1. Then the final relevance score for every node in the dataset is computed. Finally, the k objects with the highest relevance score are returned.

Algorithm 2 Tell Me More

```
Input: Dataset $D$, set of initial objects of interest $S$, $k$
Output: list of $k$ most interesting objects given $S$
1: $M_G \leftarrow$ construct the normalized adjacency matrix
2: $\boldsymbol{u}_S \leftarrow$ RWR($M_G, 0.99, S$)
3: $\boldsymbol{u} \leftarrow$RWR($M_G, 0, S$)
4: for all $d \in D$ do
5:     $LK(d) \leftarrow \boldsymbol{u}_S(d) \times \frac{\boldsymbol{u}_S(d)}{\boldsymbol{u}(d)}$
6: end for
7: return the $k$ objects with the highest score in $LK$
```

5 Experiments

In this section we describe the experiments conducted on different datasets. First, we discuss the general experimental setup. Then, for each of the different types of data, we discuss in detail the similarity measure and the pre-processing step performed. Additionally, we characterize the adopted datasets and report about the obtained results.

The goal of the experiments is to evaluate the effectiveness of TMM, that is how well TMM performs in finding similar objects. We compare the results of TMM with the results obtained by KNN. Additionally, to investigate the effect of the score adjustment, we compare TMM with a pure random walk with restart based ranking. That is, we only use Algorithm 1 (RWR) to compute the ranking of the nodes. Note that, the goal of the experiments is not to find the best similarity function. We used off the shelf similarity measures that are available for the particular types of objects.

In order to conduct a proper evaluation, we assume that every object in the dataset belongs to exactly one class. Similarity between objects is then defined as belonging to the same class. The primary performance measure is precision at a certain cut-off value, denoted as $@k$. That is, given that the objects in S are of class c, the number of objects of class c in the k highest ranked objects divided by k. More formally, let c be the class label of the documents in the initial set S, and let L be the list of the k highest ranked documents, then

$$@k = \frac{\#\text{ of documents of class } c \text{ in } L}{|L|}.$$

Note that, in our experiments we did not count objects of S that were in the top k of most related objects. The reason for this, is because TMM always returned the objects of S as most relevant objects. However, this would lead to a strong bias in favor of TMM. In our experiments we used different values for k, namely $k = 10, 20, 50$ and 100. For a dataset D with $|C|$ classes, we randomly selected for each class $|S| = 1, 2, 3, 4$ objects as initial input. For each of these sets we run our TMM algorithm, the k-nearest neighbor algorithm and the ranking method solely based upon RWR. Next, we computed the score at the different cutoff points. Depending on the number of elements in the dataset, we repeated this procedure either 100 or 1000 times. We combine the results of the different random initial sets of the same size and the different classes, and reported the average precision@k over these combined results. Hence, for each initial set size and each k value, we report the mean value of $|C| \times n$ precision@k scores, with $|C|$ the number of classes and $n = 100$ or $n = 1000$ depending on the dataset. In order to test whether the scores obtained for TMM are significantly different from the scores obtained by the other approaches, we performed a paired t test over the results at a 0.95 confidence level. Whenever the scores obtained for a certain batch are significantly higher than the scores obtained for a corresponding batch with another algorithm, we printed the results in boldface.

All experiments where conducted on a quad core 2.1Ghz desktop computer, containing $4GB$ of main memory. Over all used datasets, the response time of TMM was reasonable good. At worst it took TMM around 4.1 second to provide the k most related objects, while in the best case the results were completed in less than 1 second. However, the run time can easily be improved by using more sophisticated methods (for example the work described in [23]) to compute random walks on graphs. Moreover, another feature of their algorithm is that it does not require that the complete matrix is loaded into main memory, which makes it feasible to run random walks on extreme large datasets.

5.1 Text Documents

In order to apply our method to search engines for text documents, we represent each object (document) $d \in D$ as a bag of words, i.e. a vector where each dimension corresponds to a separate word. In case a word occurs in the document, then its corresponding value in the vector will be non zero. The dimension of the vectors is equal to the number of different words in the dataset. We used the so called tf-idf [19] weighting scheme to calculate a value for a word in the document. That is, the weight vector $\boldsymbol{v}_d$ for document $d \in D$ is: $[w_{1,d}, \ldots, w_{n,d}]$. Where $w_{t,d} =$

$$tf(t, d) \times log \frac{|D|}{|\{d \in D : t \in d\}|}.$$

With $tf(t, d)$ the term frequency of t in d, i.e. the number of different occurrences of the term t in document d. And $log \frac{|D|}{|\{d \in D : t \in d\}|}$ the inverse document frequency, that is the logarithm of the total number of documents in D divided by the number of documents in which t occurs. The similarity between two text documents is then defined as the cosine similarity between the two vectors that represent the document. That is, let $d, e \in D$, and let there be n different terms in the collection then $Sim(d,e) =$

$$\frac{d\dot{}e}{||d||\ ||e||} = 1 - \frac{\sum_{i=1}^{n} w_{i,d} \times w_{i,e}}{\sum_{i=1}^{n} w_{i,d}^2 \times \sum_{i=1}^{n} w_{i,e}^2}.$$

Clearly, the similarity takes as maximal value 1 whenever two documents are identical. Likewise, when there is no term in common between the documents the similarity equals 0. Hence, this similarity function can be used in our framework.

Test Collection We used two different document collections, namely the Wikipedia XML dataset and the Reuters 21578 text categorization dataset. The Wikipedia XML dataset [6] was provided for the document mining track at INEX 2006 [7]. The collection consists of $150,094$ XML documents with 60 different class labels. The collection was constructed by converting web pages from the Wikipedia project to XML documents. The class labels correspond to the Wikipedia portal categories of the web pages. For the document mining track a subset of the total Wikipedia collection was selected such that each document belonged to exactly one class. We did not use any structural features of the XML documents, only the content of XML documents. We used the tf-idf vector for all documents, that was provided by the organizers of the XML document mining track. In order to reduce the number of objects in the dataset, such that the matrix would fit into main memory, we selected from the complete dataset the documents that belong to one of the four largest classes. In total our dataset consists of $13,146$ documents divided over four different classes.

The Reuters-21578 text categorization dataset, is a collection of documents that appeared on Reuters news wire in 1987. These documents were manually assembled and categorized. From this collection we selected only the documents that belong to exactly one category. From the resulting set, we filtered out all documents that belonged to a category with less than hundred documents. The resulting dataset contains $7,783$ documents divided over ten classes.

In order to use the dataset in our experiments, we used the Lemur Toolkit [3] to pre-process the data. We first performed stemming and stop word removal, then we computed over the remaining data the tf-idf score for the words in the documents.

Results The results obtained over the text collection are displayed in Table 1. The results reported for the Wikipedia collection are the average values over $4,000$ runs, i.e. $1,000$ randomly selected initial sets S per class. For all setting, the results obtained by TMM where significantly better than the corresponding results with the KNN algorithm. The difference is maximal when the ten most similar documents are considered. For example, for $k = 10$ and $|S| = 4$ the score for the TMM approach is more than 7 percent points higher than the score obtained by the KNN algorithm. But also in case that more than ten similar documents are required, the difference is still considerable; the score for TMM is between 1 and 7 percent points higher than the score obtained

[3] http://www.lemurproject.org/

		TMM				RWR				KNN			
	$\|S\|$	@10	@20	@50	@100	@10	@20	@50	@100	@10	@20	@50	@100
W	1	53.10^*	51.31^*	48.34^*	46.07^*	53.10	51.31	48.34	46.07	46.97	45.20	43.15	41.34
	2	55.02^*	53.03^*	50.10^*	47.73^*	55.02	53.03	50.10	47.73	48.41	46.91	44.62	42.72
	3	56.40^*	54.48^*	51.72^*	49.52^*	56.40	54.48	51.72	49.52	49.72	48.06	45.97	43.81
	4	56.92^*	54.85^*	51.96^*	49.61^*	56.92	54.85	51.96	49.61	49.81	48.38	46.25	44.19
R	1	70.23^*	**67.66**	**63.88**	**57.87**	69.92	67.26	62.74	56.24	69.78	67.10	62.41	55.69
	2	74.38^*	71.89^*	**68.37**	**62.45**	74.21	71.85	67.39	60.07	72.85	70.43	65.95	59.33
	3	78.10^*	76.02^*	**72.57**	**66.43**	78.16	75.85	71.23	64.33	76.11	73.98	69.24	62.31
	4	79.60^*	77.41^*	**74.12**	**68.00**	79.40	77.34	73.06	66.00	78.15	75.58	70.95	63.88

Table 1. Results obtained on the Wikipedia collection (top, W) and the Reuters text categorization dataset (bottom, R). Boldfont means that the results for TMM are significantly better (at 95% confidence level), than the corresponding results for both the RWR and the KNN approach. An asterisk indicated that the results are significantly better over the KNN approach only.

for the KNN approach. Further noteworthy, is that the ranking for both TMM and KNN keeps improving when more documents are added to the initial set of interesting documents S.

Comparing TMM with the RWR approach, we have the remarkable result that TMM and RWR obtain exactly the same score on the Wikipedia dataset. Further investigation revealed that also the ranking of the documents was identical between the two approaches. These results indicate, with the similarity measure and pre-processing steps used, that there were no hub nodes in the Wikipedia collection.

Because some classes of the Reuters collection contain less than $1,000$ documents, the reported presicion@k results are the average values over $1,000$ runs. That is, for each of the ten classes 100 randomly selected initial sets S were generated. Also in this case are the results obtained by TMM significantly better than the results of the KNN approach. However, the difference in accuracy scores is lower then the difference obtained over the Wikipedia collection.

The differences reaches its maximum value, of more than 4 percent points, when more initial documents of interest are considered and when the cutoff value is large enough. In the other settings is the difference between 0.5 and 2 percent points. Further noteworthy, is that for both methods the addition of extra documents in the initial set S increases the predictive performance.

Comparing TMM with the RWR approach, it can be observed that there is a significant improvement over the accuracy scores when the RWR score is adjusted to compensate for hub nodes. However, this only occurs for larger cutoff values, i.e. @50, @100. Note that, the difference is increasing when more documents are considered. There is one setting were the RWR algorithm obtains better results than the TMM approach, namely with @10 and $|S| = 3$. However, this difference is minimal and neglectable.

5.2 Binary Data

Besides the wide availability of binary datasets, many data can be represented as binary strings. For example, the work by Cilibrasi et al. [2] transforms music files to binary

		TMM				RWR				KNN			
	$\|S\|$	@10	@20	@50	@100	@10	@20	@50	@100	@10	@20	@50	@100
C	1	**72.41**	**68.58**	**63.49**	**60.19**	66.64	63.28	59.14	56.73	63.78	59.69	53.78	43.78
	2	**69.15**	**65.94**	**61.84**	**58.83**	62.14	59.70	56.92	55.34	60.95	58.36	52.16	42.95
	3	**67.87**	**65.05**	**61.17**	**61.46**	60.23	58.55	56.24	54.98	59.01	57.55	50.84	42.89
	4	**65.46**	**63.37**	**59.92**	**57.97**	58.19	56.40	54.92	54.01	57.31	55.71	50.89	42.92
M	1	99.71*	99.43*	98.35*	**97.03**	99.67	99.47	98.33	96.94	99.56	98.26	91.76	76.94
	2	99.84	99.63*	98.97*	**97.98**	99.83	98.65	98.83	97.59	99.75	99.02	94.94	85.22
	3	99.89*	99.75*	99.35*	98.78*	99.86	99.28	99.11	98.71	99.75	99.28	96.30	89.15
	4	99.85	99.75*	99.16*	98.58*	99.88	99.73	99.13	98.55	99.84	99.37	97.10	91.83

Table 2. Results obtained on the Chess dataset (top, C) and the Mushroom dataset (bottom, M). Bold font means that these results are significantly better (at 95% confidence level), than the corresponding results for both the RWR and the KNN approach. An asterisk indicated that the results are significantly better over the KNN approach only.

strings. Due to the wide availability of binary data and the universal applicability, we include some experiments with binary data. In this setting, each object $x \in D$ is a binary vector of length n, i.e. $x = [i_1, \ldots, i_n]$. With $i_j \in \{0, 1\}$, and n the fixed number of features for the objects in the dataset.

Recently, there has been a growing interest in the data mining community in using compression to extract valuable knowledge, see for instance [25, 8, 26, 13]. This is because, as stated by Faloutsos and Megalooikonomou [8], several core aspect of datamining are essentially related with compression. In order to define a similarity score for binary data, we use a compression based similarity method, namely: normalized compression distance [14]. The intuition behind this method is that two objects are deemed close if we can significantly 'compress' one given the information in the other. Here compression is based on the ideal mathematical notion of Kolmogorov complexity, which is unfortunately not effectively computable. However, this similarity measure is approximated by using any available off the shelf compressor, e.g. gzip, zlib. More formally, $NCD(x, y) =$

$$\frac{C(x, y) - \min(C(x), C(y))}{\max(C(x), C(y))}.$$

Where $C(x, y)$ is the compressed size of the concatenation of x and y. Likewise, $C(x)$ and $C(y)$ is the length of the compressed version of x and y respectively. The NCD value is minimal, that is $NCD(x, y) = 0$, when $x = y$. Moreover, when there is no common information between x and y the NCD value equals one. Hence, in order to fit in our framework the similarity between two items x and y ($Sim(x, y)$) is defined as:

$$1 - NCD(x, y).$$

The NCD between objects was computed with a tool [4] provided by the authors of [14].

[4] available at http://www.complearn.org/

Testsets We used binary versions of the Chess(kr-kp) and the Mushroom UCI dataset. These binary version where made available by [3]. The Chess dataset contains $3,196$ objects, divided over two classes. The Mushroom dataset contains $8,124$ objects and also consists of two classes.

Results The results on the binary datasets are shown in Table 2. The results for the Chess dataset are obtained by computing the average values over $2,000$ runs, i.e. $1,000$ randomly selected initial sets S per class. For this dataset are all the results of the TMM approach significantly better than results for both the KNN as well as the RWR approach. The advantage of TMM is considerable: it varies between 16 percent points and 6 percent point over the KNN approach, and between 3 and 6 percent points over the RWR approach. Notice, that for all algorithms the addition of objects to the initial set S, especially for lower cutoff values, leads to remarkably worse results. Although this observation seems counter intuitive to the principle of using feedback to improve the ranking, a possible explanation lies in the experimental setup. In our experimental setting, we randomly select objects from a given class to be used as initial set. When the objects of a given class are widely spread, it is likely that by selecting at random objects from this class, these objects can be quite dissimilar from each other.

First worth noticing about the results on the Mushroom dataset, is that all algorithms obtained a remarkably good ranking. This suggests that the NCD is indeed a good similarity measure to use with binary data. The second remarkable observation is that the difference of the scores between TMM and KNN is largely dependent on the cutoff value. For example, when only the ten most similar objects are considered the difference between TMM and KNN varies between 0.15 and 0.04 percent points. However, if we consider the parameter setting @100, then the difference ranges from 7 to 20 percent points. The difference between TMM and the RWR method, is for all cases minimal. Solely in the setting, where the number of initial objects is low and the cutoff value is large, resulted in significantly better performance for TMM. In all other settings were the two methods equally good. That is, most of the times TMM obtained better performance and sometimes RWR obtained a higher precision score, but all these differences in obtained precision@k are minimal and neglectable.

5.3 Numerical Data

The last type of data considered in our experiments is numerical data. That is, every object x in the dataset is a point in a d-dimensional space $\mathcal{R}^d$. Hence, each object x in the dataset is a numerical vector of length d, i.e. $x = [i_1, \ldots, i_d]$ with $i_j \in \mathcal{R}$ for $1 \leq j \leq d$.

For numerical data we use a similarity measure based upon the Euclidean distance. The similarity between points x and y in the dataset, equals one minus the Euclidean distance between x and y, divided by the maximal distance between any point two points in the dataset. That is, with $\Delta(x, y)$ the Euclidean distance between x and y, $Sim(x, y) =$

$$1 - \frac{\Delta(x, y)}{\max\{\Delta(i, j) | i, j \in D\}}.$$

		TMM				RWR				KNN			
	$\|S\|$	@10	@20	@50	@100	@10	@20	@50	@100	@10	@20	@50	@100
I	1	**84.45**	**79.25**	**71.22**	**63.99**	74.30	69.15	63.05	59.43	69.85	66.30	61.33	58.22
	2	**81.05**	**76.65**	**70.09**	**64.00**	72.55	67.83	62.29	59.23	70.07	67.00	62.14	59.32
	3	**85.90**	**81.33**	**72.99**	**64.23**	73.40	69.03	63.56	59.32	70.07	67.00	62.14	59.32
	4	**84.10**	**79.00**	**71.31**	**65.75**	71.45	67.65	62.60	60.31	68.90	65.70	61.33	59.11
B	1	**95.25**	**94.75**	**93.47**	**91.71**	94.25	93.25	91.36	88.21	94.25	93.30	91.23	88.29
	2	**97.75**	**97.18**	**96.27**	**94.56**	96.00	95.33	93.79	90.87	96.00	95.35	93.78	90.91
	3	97.80^*	**97.55**	**96.47**	**95.13**	96.70	95.88	94.14	91.21	96.60	95.98	94.13	91.21
	4	**98.70**	**98.40**	**97.56**	**95.95**	97.30	96.80	95.67	92.86	97.70	97.03	95.57	92.83
S	1	84.80^*	**83.48**	**80.59**	**79.15**	83.25	81.03	77.76	75.28	82.00	80.48	77.68	74.89
	2	**90.80**	**88.98**	**86.07**	**83.53**	87.55	85.28	82.33	78.83	85.25	83.15	81.33	78.13
	3	**89.60**	**88.60**	**84.80**	**82.07**	84.60	83.05	79.90	76.40	84.75	83.18	80.00	76.84
	4	**90.65**	**89.18**	**86.75**	**84.59**	86.80	84.35	80.40	77.75	83.34	81.88	79.03	76.02

Table 3. Results obtained for the Ionospehere dataset (I, top), the Breast cancer dataset (B, middle) and the Spambase dataset (S, bottom). Boldfont means that the result is significantly better (at 95% confidence level), than the corresponding result for both the RWR and the KNN approach. Likewise, an asterisk means that the score is significantly better than the score for the KNN approach only.

Hence, $Sim(x, y)$ gets the value of 1 if x is identical to y and a value of 0 if x and y are the two points with the largest distance in the dataset.

Testsets We used three different datasets available at the UCI repository. Two of these datasets contain a relatively low number of objects, namely 351 for the Ionosphere and 561 for the Breast Cancer dataset. The Spambase dataset contains $4,601$ objects. In all three datasets are the objects divided over two classes. Due to the relatively low number of objects in the Ionosphere and Breast Cancer dataset, we only conducted 200 runs per initial set S, i.e. 100 runs for each class.

Results The results obtained over the numerical datasets are displayed in Table 3. The first remarkable observation, which only holds for the numerical datasets, is that the scores obtained for KNN and RWR are quite similar. That is, most of the times RWR obtains a slightly higher score than KNN. This in contrast with earlier experiments, where the scores obtained by RWR was, in general, considerably better than the scores obtained with the KNN approach. The results in Table 3 show that TMM performs—over all numerical datasets considered—significantly better in all but two settings than RWR. Moreover, TMM obtained significantly higher scores in all settings over KNN. The largest difference (more than 15 percent points) is obtained on the Ionosphere dataset. Likewise, the smallest significant difference (1 percent point) is obtained on the Breast Cancer dataset.

6 Discussion & Conclusion

In this paper we introduced an interactive framework to effectively explore and rerank objects retrieved by a search or datamining engine, based upon user provided feedback. We argued that such an framework can be a valuable tool for a user to find the desired information. The TMM approach can be used for very different data types and is in spite of its naive implementation relatively efficient.

We thoroughly experimentally evaluated TMM, and for every dataset considered for almost every setting, the results show that TMM performs significantly better at a 95 percent confidence level than a straightforward k-nearest neighbor approach. Also the effect of the score adjustment in TMM, in order to compensate for objects that are apriori similar to many objects, has a positive effect on the ranking results. For a substantial number of settings, this adjustment results in a significantly better ranking at a 95 percent confidence level. However, the improvement of TMM over a purely RWR methods is dependent on the dataset. For the Wikipedia dataset there was no improvement, which is likely caused because of the omission of hub nodes in the dataset. For the Reuters dataset and the Mushroom dataset, the improvement of TMM resulted only in a limited number of settings in significant better results. Nevertheless, in none of the experiments conducted it was the case that the score adjustment resulted in significantly worse results. Concluding, the proposed adjustment to compensate for hub nodes is, in general, extremely valuable.

Another issue worth reflecting one, is the assumption we made that the objects are similar if they belong to the same class. Obviously, this is a coarse level of similarity. However, the alternative is to let a domain expert decide what is most related to an object. For many data types, this is however an unrealistic option. In the IR field, where it is common practice to let experts decide which documents are relevant for a given query, it is an issue of debate [24]. In our case, we need a domain experts to judge the relevance of document given a set of documents, with is even more cumbersome than deciding whether a document is relevant given a query. For the other data types, it is even more problematic for a domain expert to decide whether two objects are similar or not. Hence, in order to perform an extensive proper evaluation the assumption that two objects are related if they are from the same class is the best option that is achievable for a wide range of different data types.

Concluding, the proposed framework Tell Me More offers and an interactive tool that allows for simple and intuitive user directed exploration of the relevant objects. The flexibility of TMM allows it to be used with different types of data in a search, database or datamining engine. Moreover, because of its relatively quick response time, TMM can be used in an online setting.

Interesting directions for further research includes the exploration of different similarity functions between objects, especially appealing is a similarity function between web pages that takes both the structure of the web (i.e. the link information) and the content of the web page into account. Another direction is to let the user specify not only the relevant objects but also the objects that are not of his interest.

References

1. R. Agrawal, H. Mannila, R. Srikant, H.Toivonen, and A. Verkamo. Fast discovery of association rules. In *ADMA*, pages 307–328, 1996.
2. R. Cilibrasi, P. Vitányi, and R. Wolf. Algorithmic clustering of music. In *4th International Conference on WEB Delivering of Music*, pages 110–117, 2004.
3. F. Coenen. The lucs-kdd discretised/normalised arm and carm data library.
4. N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR*, pages 239–246, 2007.
5. J. Dean and M. Henzinger. Finding related pages in the world wide web. *Computer Networks*, 31(11-16):1467–1479, 1999.
6. L. Denoyer and P. Gallinari. The Wikipedia XML Corpus. *SIGIR Forum*, 40(1):64–69, 2006.
7. L. Denoyer and P. Gallinari. Report on the XML mining track at inex 2005 and inex 2006: categorization and clustering of XML documents. *SIGIR Forum*, 41(1):79–90, 2007.
8. C. Faloutsos and V. Megalooikonomou. On data mining, compression, and Kolmogorov complexity. *Data Mining and Knowledge Discovery*, 15(1):3–20, 2007.
9. A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal. Using the wisdom of the crowds for keyword generation. In *WWW*, 2008.
10. Z. Ghahramani and K. Heller. Bayesian sets. In *Advances in Neural Information Processing Systems*, 2005.
11. T. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.
12. T. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating strategies for similarity search on the web. In *WWW*, pages 432–442, 2002.
13. J. De Knijf. Mining tree patterns with almost smallest supertrees. In *SIAM International Conference on Data Mining*. SIAM, 2008.
14. M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, 2004.
15. D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.
16. K. Onuma, H. Tong, and C. Faloutsos. Tangent: a novel, 'surprise me', recommendation algorithm. In *KDD*, pages 657–666, 2009.
17. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
18. J. Pan, H. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.
19. G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.
20. A. Siebes, J. Vreeken, and M van Leeuwen. Item sets that compress. In *SIAM International Conference on Data Mining*, 2006.
21. J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *IEE Intl. Conf. on Data Mining*, pages 418–425, 2005.
22. H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.
23. H. Tong, C. Faloutsos, and J. Pan. Random walk with restart: fast solutions and applications. *Knowl. Inf. Syst.*, 14(3):327–346, 2008.
24. E. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. In *SIGIR*, pages 315–323, 1998.
25. J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214, 2011.
26. D. Xin, J. Han, X. Yan, and H. Cheng. On compressing frequent patterns. *Data & Knowledge Engeneering*, 60(1):5–29, 2007.