MindMiner: A Mixed-Initiative Interface for Interactive Distance Metric Learning

Xiangmin Fan[1], Youming Liu[1], Nan Cao[2], Jason Hong[3], and Jingtao Wang[1]

[1] Computer Science and LRDC, University of Pittsburgh, Pittsburgh, PA, USA
{xiangmin,youmingliu,jingtaow}@cs.pitt.edu
[2] IBM T.J. Watson Research, Yorktown Heights, NY, USA
nancao@us.ibm.com
[3] HCI Institute, Carnegie Mellon University, Pittsburgh, PA, USA
jasonh@cs.cmu.edu

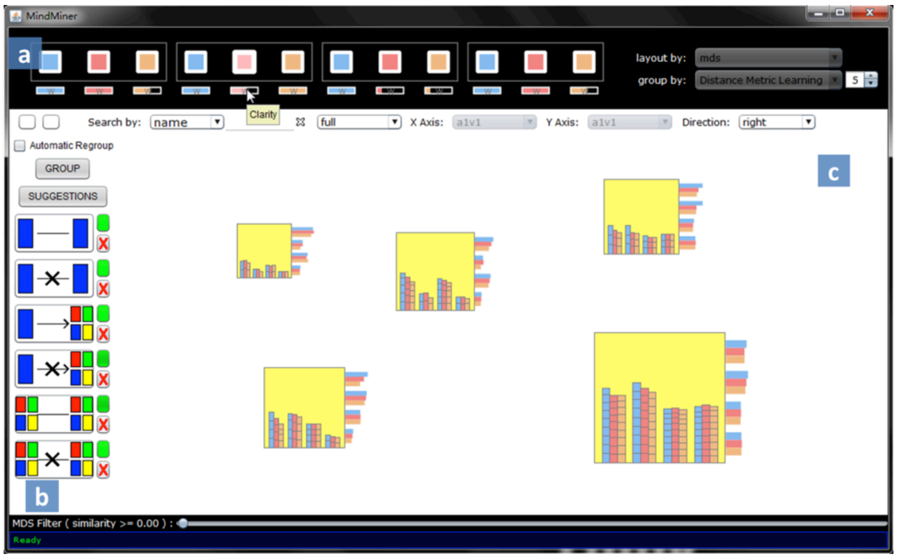
Abstract. We present MindMiner, a mixed-initiative interface for capturing subjective similarity measurements via a combination of new interaction techniques and machine learning algorithms. MindMiner collects qualitative, hard to express similarity measurements from users via *active polling with uncertainty* and *example based visual constraint creation*. MindMiner also formulates human prior knowledge into a set of inequalities and learns a quantitative similarity distance metric via *convex optimization*. In a 12-subject peer-review understanding task, we found MindMiner was easy to learn and use, and could capture users' implicit knowledge about writing performance and cluster target entities into groups that match subjects' mental models. We also found that MindMiner's constraint suggestions and uncertainty polling functions could improve both efficiency and the quality of clustering.

Keywords. Mixed-Initiative Interface, Clustering, Visualization, Convex Optimization, Intelligent User Interfaces, Machine Learning.

1 Introduction

Cluster analysis is a common task in exploratory data mining, and involves combining entities with similar properties into groups. Clustering is desirable in that it is unsupervised and can discover the underlying structure of data without *a priori* information. However, most clustering techniques face one key challenge when used in real world applications: clustering algorithms expect a quantitative, deterministic distance function to quantify the similarity between two entities. In most real world problems, such similarity measurements usually require subjective domain knowledge that can be hard for users to explain. For example, a human instructor may easily find that the writing styles of two students are very similar to each other by reviewing their writing samples. However, such perceived similarities may not be reflected accurately in the distance measurement between two corresponding feature vectors.

Previous efforts have been made by researchers to improve the quality of clustering using both algorithmic [8], [27, 28] and user interface [6], [10], [11] approaches. For example, various semi-supervised clustering algorithms have been proposed by researchers in the machine learning community, either by adapting a similarity measure via user specified constraints or by modifying the process of determining intermediate cluster centers. However, most existing work focuses on *theoretical feasibility*: they assume users can provide sufficient, unambiguous, and consistent information to facilitate clustering before the algorithms start.



layout by: mds
group by: Distance Metric Learning 5
Search by: name
full
X Axis: a1v1
Y Axis: a1v1
Direction: right
MDS Filter (similarity >= 0.00) :
Ready

Fig. 1. The primary UI of MindMiner, showing 23 students in a college-level philosophy class grouped into five clusters based on their performance (accuracy, clarity, and insight) in four writing assignments using six example constraints specified by an instructor. MindMiner consists of three parts: (a) The *Active Polling Panel* allows users to optionally indicate the importance for each measurement. Each colored square box represents one feature (4 assignments x 3 features). The rectangular bars beneath show real-time updates of the corresponding "weights"; (b) The *Constraints Management Sidebar* displays example-based constraints collected; (c) The *Interactive Visualization Workspace* lets a user see detailed information about entities, create example-based constraints, split and combine groups, examine and refine clustering results and examine personalized groups.

Researchers in HCI and Information Visualization have also explored the use of interactive applications for guided clustering [10], [16], [20], [24]. Some interfaces rely on real time feedback of clustering results to help users choose proper features, samples, and the number of clusters to use. Other systems, such as IVC [10], attempt to provide mechanisms to collect users' *a priori* knowledge, such as which samples should be in the same group, and which should not. However, most existing interactive clustering systems focus on *conceptual demonstration* and do not address important elements for making such systems practical, such as how to browse, how to manage users' collected *a priori* knowledge, and how to achieve better clustering results with more representative constraint examples.

To address these challenges, we created a mixed-initiative interface, MindMiner (Fig. 1), to capture users' subjective similarity measurements. MindMiner makes contributions in both interaction design and machine learning algorithms. MindMiner captures prior knowledge from users through *active polling with uncertainty* and *example based visual constraint creation*. *Active polling with uncertainty* enables users to specify their subjective opinion on the *global* importance of a feature (including the value "not sure") which improves the accuracy and speed of the clustering results. *Example based visual constraint creation* allows to directly express their *a priori* domain knowledge via six types of constraints on the data samples being visualized. The constraint management interface allows users to browse existing examples, investigate the impact of each constraint, and discover conflicting conditions.

MindMiner also provides interface level support that uses *active learning* to provide optional hints as to which examples might be more helpful for clustering. We also report how inequalities are formulated based on the collected *a priori* knowledge and how the inequalities are used in a convex optimization process to extract the "mental model" of entity similarity from users in the form of the Mahalanobis distance metric.

Specifically, this paper makes the following contributions:

- We propose two interaction techniques, *active polling with uncertainty* and *example-based constraints collection*, to collect, visualize, and manage implicit, subjective domain knowledge by scaffolding end-users incrementally. These techniques assume that users' domain knowledge may be *ambiguous* and *inconsistent*.
- We introduce an improved distance metric learning algorithm that takes into account input ambiguity and avoids trivial solutions[1] in existing algorithms.
- We present effective *active learning* heuristics and corresponding interface design to collect pairwise constraints at both entity and group levels. We show in a 12-subject controlled study that our design can significantly enhance the clustering relevance.
- We present an interactive data exploration and visualization system, MindMiner, to help end-users externalize domain knowledge and improve data exploration efficiency via distance metric learning. To our knowledge, this is the first interactive system that provides both algorithm and interface level support for handling *inconsistent*, *ambiguous* domain knowledge via distance metric learning.

2 MindMiner in Action

We present a scenario giving an overview of MindMiner. MindMiner was originally designed for computer assisted peer-review and grading scenarios, but can also be used for other interactive clustering tasks.

[1] When the number of constraints is small, e.g., less than 20, existing algorithms tend to generate trivial distance metrics that have only one or two non-zero dimensions.

Alice is an instructor for a philosophy course with 23 students. There are four writing assignments, and the essays submitted by students are graded via three features (accuracy, clarity, and insight). The grading is done by herself, the TA, and "double-blind" peer-review by students. Alice feels it is tedious and time consuming to get a clear picture of the overall performance of the whole class. Alice also wants to identify students with similar writing problems so that she can provide customized feedback to them. Alice can use MindMiner to achieve a balance between workload and feedback accuracy.

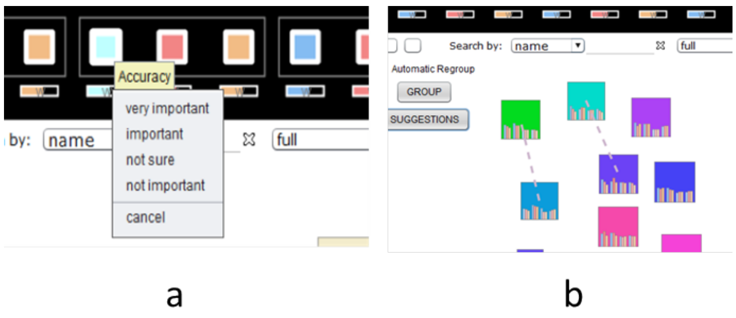


Fig. 2. Knowledge collection interfaces of MindMiner. a: Interface for *active polling with uncertainty*. b: Interface for *example-based constraints collection*.

After logging into MindMiner, Alice retrieves student performance data from a remote server. Alice believes that writing accuracy is the most important factor she cares about and clarity a close second. She is not sure about the importance of insight. Therefore, she uses the *Active Polling Panel* (Fig. 2.a) to make a choice for each feature. She chooses "very important" for accuracy, "important" for clarity and "not sure" for insight.

Then Alice teaches MindMiner her subjective judgments on performance similarity of students by labeling some example constraints. Alice reviews detailed information of the students by mousing over the nodes. MindMiner automatically selects the most potentially informative pairs and highlights the suggestions with dashed lines (Fig. 2.b). She examines two students involved in a constraint suggestion. After judging that they performed similarly, she drags them together, which creates a must-link constraint between the two students, telling MindMiner that these students should be grouped together. A corresponding symbol for this constraint then appears in the *Constraints Management Sidebar* (Fig. 1.b). She later creates a cannot-link between dissimilar students by right clicking and dragging from one to the other. Every time Alice adds a new constraint, the distance metric learning module runs a convex optimization algorithm to derive the optimized solution. The bars in the *Active Polling Panel* (Fig. 1.a) show the updated weights of corresponding feature dimensions in real-time.

MindMiner also checks if there are any conflicts caused by new constraints. If so, it gives a warning by highlighting the corresponding constraints in the *Constraints Management Sidebar* using a red background. Alice checks the conflicting constraints and finds that one of the previous example constraints she created is not correct so she deletes it. Each constraint item in the *Constraints Management Sidebar* is double-linked with corresponding students via mouse hovering, so it is easy for Alice to diagnose the cause when a conflict is reported by MindMiner.

Alice clicks the "group" button located on the top of the *Constraints Sidebar* to see whether the examples provided by her are sufficient for grouping students together in a useful manner. MindMiner applies the updated distance metric using a k-means clustering algorithm, and then displays the resulting groups. Alice then checks the results and finds that the groups are not as good as she expected. She adds a few more constraints and then she checks "automatic regroup". In this mode, once there is a new constraint, MindMiner's learning algorithm executes and the system automatically regroups the students based on the most updated distance metric. Alice continues this iterative process by adding new constraints, deleting existing constraints or adjusting importance levels of the features, until she gets satisfactory clustering results.

3 Related Work

We have organized related work into three categories: interactive machine learning, clustering interfaces, and semi-supervised clustering algorithms.

3.1 Interactive Machine Learning

Because of the inherent ambiguities in human activities and decision-making processes, many researchers believe that machine learning algorithms can never be perfect in replacing human experts [25]. As an alternative, researchers have investigated mixed-initiative interfaces [15] that keep humans in the loop, providing proper feedback and domain knowledge to machine learning algorithms [1], [2], [7], [12], [18], [21], [26].

For example, CueFlik [12] allows end-users to locate images on the web through a combination of keyword search and iterative example-based concept refinement activities. Although both CueFlik and MindMiner support distance metric learning and active learning, there are major differences between the two systems. First, CueFlik supports one-class information retrieval while MindMiner focuses on multi-group semi-supervised clustering that matches a user's mental model. Second, in CueFlik, users' feedback was provided in the form of positive and negative image examples and it is not necessary to handle conflicting examples due to the diversity of online images. In comparison, MindMiner collects both feature uncertainty information and pairwise relationship information from users to formulate a convex optimization problem. MindMiner also provides a dedicated constraint management interface to collect, browse user-specific knowledge and resolve prospective knowledge conflictions, which are common in multi-group clustering tasks.

Apolo [7] is an interactive sense making system intended to recommend new nodes in a large network by letting users specify exemplars for intended groups. Apolo is similar to MindMiner in that both systems accept examples for intended groups from end-users. However, Apolo focuses on suggesting new members for existing groups in a graph topology, whereas MindMiner aims to generalize examples from end-users, using existing groups to create new groups via distance metric learning. MindMiner also works on an unstructured, unlabeled sample space rather than a graph.

In addition to providing representative examples [7], [12], interactive machine learning also allows end-users to specify their preferences on system output and adapt to these preferences in model parameters [18], [21], [26]. ManiMatrix [18] lets users interactively indicate their preferences on classification results by refining parameters of a confusion matrix. In CAAD [21], users correct classification errors caused by the activity detection algorithm by manually moving documents to the correct category. The input matrix for the Nonnegative Matrix Factorization (NMF) algorithm in CAAD also gets updated by such changes.

3.2 Clustering Interfaces

Many interactive tools have been designed to facilitate cluster analysis and exploration by providing real time feedback to parameter and dataset changes. For example, Hierarchical Cluster Explorer (HCE) [24] is an interactive hierarchical clustering system that supports comparison and dynamic querying of clusters. DICON [6] uses a tree-map style, icon-based group visualizations and a combination of k-means clustering and manual grouping to facilitate cluster evaluation and comparison. NodeTrix [14] combines a matrix representation for graphs with traditional node-link graph visualization techniques. Users can select and group nodes to generate an adjacency matrix to visualize cluster patterns in a graph. Many researches also focus on finding clusters in multidimensional data based on parallel coordinate plots (PCPs) [17]. For example, Fua et al. [13] used hierarchical clustering in PCP to detect clusters. Novotny [19] use polygonal area in PCP to represent clusters. However none of these techniques incorporate end-user feedback to improve clustering.

IVC [10] supports incorporating user specified pairwise constraints in clustering. The authors leveraged the PCK-Means algorithm proposed by Basu, Banerjee and Mooney [3] for clustering, and the pairwise constraints were incorporated into the k-means algorithm as a penalty in the cluster assignment state. In contrast to MindMiner, IVC was only tested in simulations and there was no method to manage constraints.

Basu, Fisher et al. [4] implemented a document clustering system that combines user specified constraints and supervised classification. However, there was no constraint collection and management interface in their system.

3.3 Semi-Supervised Clustering Algorithms

Researchers in machine learning have explored the use of human knowledge in unsupervised clustering [8], [27, 28], i.e. semi-supervised clustering. The users' prior knowledge was leveraged in semi-supervised clustering algorithms by either adapting the similarity measure or modifying corresponding search-rules. The semi-supervised clustering algorithm in MindMiner was inspired by the one proposed by Xing et al. [28]. Three major revisions in Xing's algorithms were made in MindMiner to generate higher quality results on real-world data. First, we added the support for user specified feature uncertainty and two additional groups of pairwise constraints. Second, we incorporated an active learning algorithm and corresponding heuristics to improve the quality of constraints collected. Third, we added a regularization step to avoid trivial solutions derived from the convex optimization.

4 Design of MindMiner

In the following sections, we discuss these parts in more detail, including the visualization design, the knowledge collection interfaces in MindMiner and the underlying mathematical modeling and the convex optimization algorithm for learning the distance metric respectively.

4.1 Visualization Design

We use interactive stacked bar charts in MindMiner to visualize clusters of data with multivariate features. Fig. 3 illustrates an example of our design in which a student dataset is visualized. Each student, treated as an entity, is characterized by his/her performances in a writing course along different features, i.e. accuracy, clarity, and insight. These features are defined by the user, and are measured based on the peer-review scores of three writing assignments.

As shown in Fig. 3.a, we use different background colors to illustrate different assignments, and use different foreground colors to represent the different features. A student's feature vector is represented as a bar chart (Fig. 3.b) in which the sizes of the bars represent the corresponding review scores. Similarly, we represent a clustered group of students (Fig. 3.c) by packing all of the students' review scores together into a stacked bar chart, categorized by assignments (Fig. 3.d). We also represent the averaged student feature scores of each assignment as another grouped bar chart attached to the group. The position of the bar chart, i.e. left, right (default location, Fig. 3.d), bottom, and top, can be customized by users. The resulting visualization shows the overall distribution of data while keeping individual details easily visible.

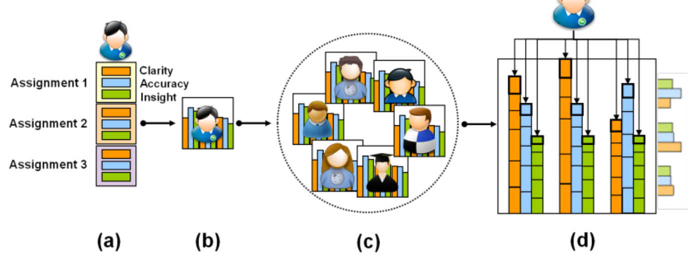


Fig. 3. MindMiner visualization design. a) Feature vector of a student based on three writing assignments and three different features. b) Student barchart icon. c) A group of similar students. d) Stacked bar chart icon for a cluster of students.

4.2 Knowledge Collection Interfaces

MindMiner offers two novel knowledge collection techniques, *active polling with uncertainty* and *example-based constraints collection*, to make it easier for end-users to externalize their implicit mental models of entity similarity. We also introduce an active learning [9] heuristic to help users provide similarity examples that are more informative to the follow-up learning algorithms.

Active Polling with Uncertainty. MindMiner lets users specify their perceived importance of each feature via Active polling with uncertainty (Fig. 2.a). Available choices are – "not important", "important", "very important" and "not sure". This step is optional and the default choice is "not sure". These choices correspond to different parameter search spaces in the convex optimization stage. As we illustrate later, expressing subjective certainty can reduce the number of examples needed in the next step and improve clustering quality.

Example-based Constraints Collection. MindMiner allows users to specify their knowledge on entity similarity via examples. This approach is supported by a psychology theory [23], which suggests that people represent categories through examples or prototypes. Instead of collecting absolute annotations or labels from end-users, which have been proven by many research findings to be unreliable, especially for subjective domain knowledge, we choose to collect *pairwise* knowledge instead.

Table 1. Symbols and descriptions of the six pairwise constraints supported by MindMiner. Collected constraints are shown in the *Constrains Management Sidebar* (Fig. 1.b).

Symbol	Name	Details
🟦—🟦	Must-link	Lets user specify that two entities should be grouped together. Leads to a new entry in equation (2)
🟦✕🟦	Cannot-link	Lets user specify that two entities should not be grouped together. Leads to a new entry in equation (3).
🟦→🟥🟩🟦🟨	Must-belong	Lets user specify that one entity should be included in a specific group. Leads to multiple must-links, and added as multiple entries in equation (2)
🟦✕→🟥🟩🟦🟨	Cannot-belong	Lets user specify that one entity should not be included in a specific group. Leads to multiple cannot-links, and added as multiple entries in equation (3)
🟥🟩🟦🟨—🟥🟩🟦🟨	Similar groups	Lets user specify that two existing groups should be put together. Leads to multiple must-links, and added as multiple entries in equation (2)
🟥🟩🟦🟨✕🟥🟩🟦🟨	Dissimilar groups	Lets user specify that no items in the two existing groups should be put into the other group. Leads to multiple cannot-links, and added as multiple entries in equation (3)

End-users can provide three types of constraints to represent their prior knowledge on entity similarity (Table 1): 1) pairwise entity similarity (must-link, cannot-link); 2) entity-group similarity (must-belong, cannot-belong); 3) pairwise group similarity (similar-groups, dissimilar-groups). The latter two types include new constraints that are never found in existing semi-supervised clustering algorithms. All six constraints can be specified by users in the primary interface via mouse-based direct manipulation operations (Table 1). Constraints created are shown in the *Constraint Management Sidebar* (Fig 1.b). This sidebar allows users to browse, remove, or check the impact of each constraint created. Conflicting constraints are also highlighted in red. In the clustering stage, the top-right corner of each constraint in the *Constraints Management Sidebar* shows whether this constraint is currently satisfied (green means satisfied while red means no). Using visual feedback, rather than directly enforcing them via heuristics, allows end-users to inspect the unsatisfied constraints and refine them if necessary.

Two challenges arise when collecting similarity samples. First, not all constraints are equally useful. A user could provide multiple examples, but it might not improve the convergence speed of the convex optimization algorithm or reliability of the distance metric learning process. Second, investigating the similarity between two entities or groups can be repetitive, tedious, and demanding on short-term memory.

To address these challenges, MindMiner uses an active learning approach to automatically select the entity/group pairs that could be most informative to the follow-up convex optimization algorithm, and then encourages users to specify their similarities. The detailed active learning algorithm and heuristics are described in detail in the next section.Distance Metric Learning Algorithms

Once MindMiner collects a new piece of information (feature uncertainty or pairwise sample similarity) from users, it converts such information into a set of inequalities, formulates a convex optimization problem [5], and learns a distance metric from user provided similarity information. This distance metric is then used for clustering entities into groups. There are four major steps in this distance metric learning process: 1) constraint conflict detection; 2) inequalities generation; 3) convex optimization; 4) results regularization. We explain details of each step in the rest of this section.

4.3 Mathematical Background

An entity in MindMiner is denoted by an n-dimensional feature vector. For example, entity s_i is represented by (s_{i1}, s_{i2},..., s_{in},) in which n is the dimension in the feature space. The similarity measurement $d(s_i, s_j)$ between entity s_i and entity s_j is defined as:

$$d(s_i, s_j) = \sqrt{(s_i - s_j)W(s_i - s_j)^T} \tag{1}$$

Here W is an $n*n$ distance metric matrix. Letting $W=I$ leads to Euclidean distance. In MindMiner, we restrict W to be diagonal for efficiency concerns, the same framework can be used to learn a complete W with sufficient user examples. Determining each non-zero element in the diagonal W corresponds to learning a metric in which the different measurement features are given different “weights”. Therefore, our goal here is to find W (weights vector) which best respects the information collected via the active polling process and interactive constraint creation process.

4.4 Constraint Conflict Detection

The information collected with *active polling with uncertainty* is used to define the lower and upper bound of the associated weight for each feature in the follow-up optimization process. The choice "Very important" corresponds to a weight of 1 (highest), "not important" corresponds to a weight of 0 (lowest), the weights of "important" features are set to be in a range of [0.6, 1] while "not sure" features are set to be within [0, 1]. In the end, we get a set of ranges for the weights of all features:

$$WeightBounds(WB) = \{[w_{1_{lb}}, w_{1_{ub}}], \dots, [w_{n_{lb}}, w_{n_{ub}}]\}$$

As shown in Table 1, depending on the constraint type, each constraint collected will be converted to one or multiple pairwise relationships and a Boolean flag. For must-link and cannot-link, the corresponding list only contains one pair, with a Boolean flag indicating the similarity relationship (true for similar and false for dissimilar) between the entities involved in the pair. For other types of constraints, they are first converted to multiple pairwise constraints such as must-links or cannot-links. Then these must-links or cannot-links are added to the pairs list of the corresponding constraint.

```
input  : Existing constraints in $C$ and the new added constraint $c$
output: All the conflicting constraint(s)
for each exsiting constraint $C_i \in C$ do
    if the similarity flags of $c$ and $C_i$ are different then
        //conflicts may exist;
        iterate through the pairs lists of $c$ and $C_i$ to see if there is a
        common pair. If yes, mark $C_i$ as a conflicting constraint.
    end
end
```

Algorithm 1. Constraint conflict detection.

By using this list based constraint representation, Algorithm 1 presents pseudo code to detect prospective conflicts in the constraints provided by end-users. If a constraint conflict is detected, corresponding constraints in the *Constraints Management Sidebar* (Fig. 1.b) will turn red. Also, hovering over a conflicting constraint will highlight the remaining constraint(s) in conflict, as well as the corresponding entities and groups.

4.5 Active Learning Heuristic

As noted earlier, not all user-specified examples are equally helpful in improving the results from convex optimization. Some examples could be repetitive and would not justify the time spend by users to specify them or the extra computer-time added to the optimization process. To address this dilemma, we adopted concept of active learning, which allows MindMiner to identify and suggest ambiguous entity relationships that are most informative in improving the quality of distance metric learning. For example, suppose that an entity S_1 is currently in cluster A and S_1 is on the boundary of A and B while student S_2 is the best exemplar of A; then the constraint suggestion < S_1, S_2> would be posed to end-users asking for whether they are similar or not. We designed a three-step active learning heuristic listed below to recommend informative constraint samples to end users. This active learning heuristic will be executed every time when a new constraint is provided by users. The informative entity pairs discovered via active learning are marked with dashed lines in the main interface.

- Within each cluster *c*, find the entity with minimum distance to the center of *c* as the exemplar for *c*.
- For each entity *s*, calculate the difference *d* between the distances from *s* to the nearest two clusters c_1 and c_2. If *d* is less than a threshold, we mark the entity *s* as ambiguous.
- For each entity s' that was marked as ambiguous, create a constraint suggestion between s' and the exemplar of cluster it currently belongs to.

4.6 Inequality Generation

We also keep two global sets: *S*, which is a set of pairs of entities to be "similar" and *D*, which is a set of pairs of entities to be "dissimilar". All the similar pairs are added to *S* while all the dissimilar pairs are added to *D* during the interactive constraint creation process.

After the constraint conflict detection step, we convert the user knowledge collected through *active polling with uncertainty* and *example-based constraints collection* to Weight Bounds which are a set of weight ranges for all features, and *S* and *D* which are sets of pairs of similar/dissimilar entities.

A straightforward way of defining a criterion for the meaningful distance metric is to demand that pairs of entities in *S* have small squared distance between them (eq.2). However, this is trivially solved with *W*=0 and is not informative. Our approach was primarily inspired by the method proposed by Xing et al. [28]. To avoid the above mentioned trivial solution, we add a new inequality constraint (eq.3) to ensure it takes dissimilar entities apart. In this framework, we transform the problem of learning meaningful distance metrics to a convex optimization problem:

$$\min_w \sum_{(s_i,s_j)\in S} d^2(s_i,s_j) \tag{2}$$

s.t.

$$\sum_{(s_i,s_j)\in D} d(s_i,s_j) \geq 1 \tag{3}$$

For each

$$w_k : w_k \geq 0 (1 \leq k \leq n) \tag{4}$$

Each sum item in eq.2 corresponds to a positive constraint collected, while each sum item in eq.3 corresponds to a negative constraint collected (Table 1).

It can be proven that the optimization problem defined by eq.2 – eq.4 is convex, and the distance metric W_{raw} can be solved by efficient, local-minima-free optimization algorithms.

Unfortunately, according to our early experiences on real world data, it is not desirable to use W_{raw} as the distance metric for the follow-up clustering tasks. According to our observations, when the number of constraints is very small, especially at the beginning of a task, convex optimization usually lead to a sparse distance metric where most values in the distance metric are close to zeros, i.e. only minimal features, e.g., 1 or 2 features, are taken into account in similarity measurement, implying a trivial solution that does not represent the real-world situation. We use an extra result regularization step and leverage the information collected in the *active polling with uncertainty* step to generate more meaningful distance metric that could be a better representation of a user's mental model.

4.7 Result Regularization

In order to make distance metrics respect both feature uncertainty information and the constraints collected by MindMiner, we regularize *Wraw* by using *Weight Bounds (WB)*. Detailed steps are described in Algorithm 2.

After finishing the result regularization step, we get a *W* that conforms to all the prior knowledge we collected from end-users. We apply *W* to the distance metric function and get the relevant distance metric. Then the distance metric *W* is used in k-means clustering algorithm to generate meaningful clusters.

```
input  : the raw weight $W_{raw_i}$ and the "lower bound" ($W_{i_{lb}}$) and "upper
         bound" ($W_{i_{ub}}$)
output: the regularized weight $W_i$ ($1 \leq i \leq n$)
Iterate through $W_{raw}$ and find the maximum and minimum values
$W_{raw-max}$, $W_{raw-min}$
for i from 1 to n do
    if $W_{i_{lb}} = W_{i_{ub}} = 1$ then
        set $W_i = 1$
    else if $W_{i_{lb}} = W_{i_{ub}} = 0$ then
        set $W_i = 0$
    else
        set $W_i = W_{i_{lb}} + (W_{i_{ub}} - W_{i_{lb}}) \cdot \frac{W_{raw_i} - W_{raw-min}}{W_{raw-max} - W_{raw-min}}$
end
```

Algorithm 2. Result Regularization.

4.8 Implementation

MindMiner is written in Java 1.6. The convex optimization algorithm is implemented in Matlab. We use Matlab Builder JA to generate a Java wrapper (i.e. a .jar file) around the actual Matlab programs. MindMiner can run as a desktop application or a web application via Java Web Start.

5 Evaluation

We conducted a 12-subject user study to understand the performance and usability of MindMiner. We had three basic goals for this study. One was to figure out whether or not the ideas behind MindMiner are easy to understand and if the current MindMiner implementation is easy to learn. The second goal was to evaluate the overall performance of MindMiner in capturing the similarity measurements in users' minds. The third goal was to verify the efficacy of each new component designed (i.e. *active learning* heuristics, *example based visual constraint creation*, and *active polling with uncertainty*). The data loaded in MindMiner in this study was anonymized real world data from a 23 student philosophy course in a local university with permission from the internal review board (IRB) and the instructor.

5.1 Experimental Design.

The study consisted of five parts:

Overview. We first gave participants a brief introduction and a live demo of MindMiner. We explained each task to them, and answered their questions. After the introduction, we let the participants explore the interface freely until they stated explicitly that they were ready to start the follow-up tasks.

Clustering and active learning. We used a within-subjects design in this session. There were two similar tasks: task 1 was clustering the students into four groups based on their performance in the first assignment; task 2 was the same as the previous task except that users were to only consider the "accuracy" features of the assignments. There were two conditions in this section: (A) providing constraint suggestions via active learning; (B) without active learning. Six participants performed task 1 with condition A and task 2 with condition B. The other six performed task 1 with condition B and task 2 with condition A. The order of the two tasks was counterbalanced. Each participant could provide up to ten example-based pairwise constraints (both positive examples and negative examples) for each task. The active polling with uncertainty feature was disabled in both conditions. We collected each participant's task completion time for each condition and the distance metrics derived by the learning algorithm.

Active polling with uncertainty. We used a between-subjects design in this session with two conditions: the constraints & active polling condition and the constraints-only condition. The active learning feature was enabled in both conditions. The task required users to find five students with similar performances to one student named "Indrek". We told the participants that the accuracy and clarity features of the first two assignments were very important to consider and asked them to define the importance of other features themselves. We hypothesized that given meaningful clustering results, one can find similar students easily just by going over each student in the target's group. Otherwise, if the clustering was not good, the participants would have to view groups besides the target's group to find similar students.

Free exploration. In this session, the participants were asked to group the students into three categories based on their own grouping criteria. Users were encouraged to think aloud and even write down their rules on a piece of paper. They were also encouraged to explore MindMiner as long as they wanted.

Qualitative feedback. After participants completed all the tasks, they were asked to complete a questionnaire and describe their general feeling towards our system.

5.2 Participants and Apparatus.

We recruited 12 participants (5 female) between 22 and 51 years of age (mean = 27) from a local university. Two were instructors from physics department and psychology department respectively. The other ten were graduate students who have teaching experience. Each study lasted for around 60 minutes (up to 90 minutes maximum), and each participant was given a $10 gift card for the time.

A Lenovo ThinkPad T530 laptop computer with Intel Core i5-3210 CPU, 4GB RAM, running Windows 7 was used. An external NEC 23 inch LCD monitor with a resolution of 1920*1080 was attached to the laptop to run MindMiner.

5.3 Evaluation Results

Clustering and Active Learning.

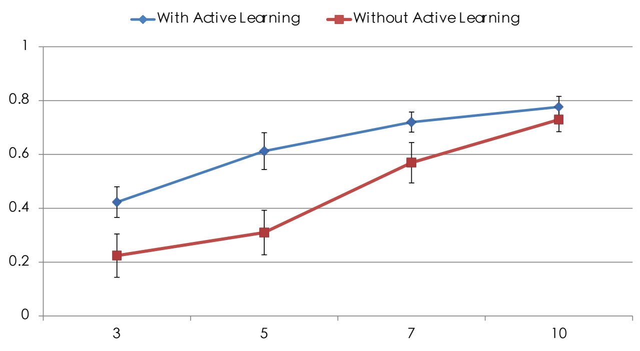


Fig. 4. Average cosine similarities between “gold standard” and distance metrics learned by different numbers of constraints (the higher the better).

The average task completion time in the “with active learning” condition is significantly shorter than that of the “without active learning” condition (266.4s vs. 357.4s, $F_{1,\ 11}=13.403$, $p<0.01$). We observed that with active learning suggestions enabled, participants tended to compare the students involved, instead of randomly picking several students to compare. MindMiner suggestions gave them clear “targets” to inspect; otherwise, they would look for “targets” themselves, which usually leads to more time. Furthermore, when there were no suggestions, participants had to compare multiple students, which required having to remember many students’ scores. In comparison when they had system suggestions, they only need to compare two students.

To evaluate the quality of distance metrics learned in the two conditions, we defined our "gold standard" to be a weight vector where the weights of predefined important features are 1s, and the weights of other features are 0s. We used cosine similarity between the standard weight vector and the weight vector learned from our algorithm to measure the quality of distance metric learned (Fig. 4).

Analysis of variance revealed that there was a significant difference ($F_{1, 11}$=7.42, p<0.05) in the quality of the distance metric learned. We found that there was a significant main effect ($F_{3, 9}$=19.30, p<0.05) in quality among different numbers of constraints collected. Pairwise mean comparison showed that more constraints led to significantly better quality distance metrics. With the same number of constraints, the quality of distance metrics learned with active learning was significantly higher than that without active learning for all four numbers of constraints in Fig. 4.

Active Polling with Uncertainty.
When active poling with uncertainty was enabled, the average completion time was 252.7 seconds (σ =19.6). When disabled, the average completion time was 304.8 seconds (σ =43.1). However, the difference was not statistically significant (*p*=0.297).

The active polling with uncertainty condition also led to significantly more similar students discovered (4.67 vs. 2.50, p<0.001) than the condition without active polling (Fig. 5). This finding showed that active polling with uncertainty could also facilitate users by helping them to learning process to derive more relevant entities.

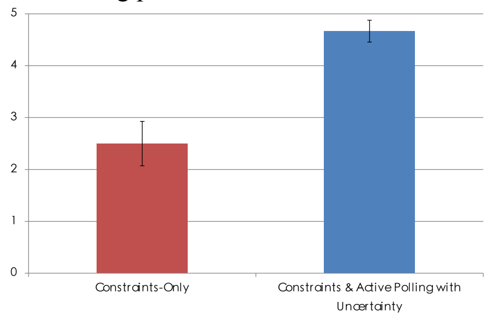


Fig. 5. Average number of similar students discovered by condition (the more the better).

Free Exploration.
A total of 458 interaction activities were recorded in the free exploration session (Fig. 6). Examining details panels was the most frequent activity (51.1%), followed by adding constraints (20.1%). Of all the 92 constraints added by participants in this session, 74 (80.4%) were from system suggestions. Other frequent activities included using the active polling with uncertainty feature (13.8%), grouping entities (8.3%), checking existing constraints (5.0%), and deleting constraints (1.7%). Among all the 8 constraint deletions, 6 were unsatisfied inappropriate constraints and 2 were constraint conflicts.

We observed that participants tended to add more positive examples (must-link, must-belong, and similar-groups) than negative examples (cannot-link, cannot-belong, and dissimilar-groups) (78.6% vs. 21.4%) when the active learning feature was disabled. Participants tend to not provide negative examples even when they were confident that two entities were very different; when the active learning feature was enabled, the ratio of negative examples almost doubled (40.8%) and the difference was statistically significant. This observation indicated that the current active learning interface and heuristics in MindMiner can increase users' awareness and contribution to negative examples.

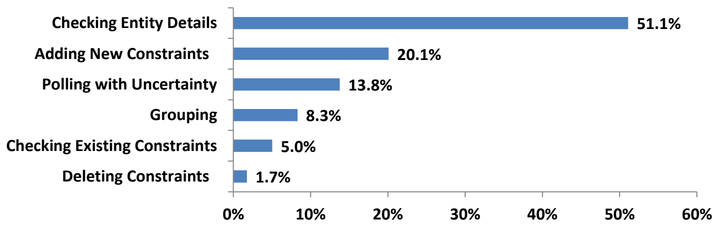


Fig. 6. Activity distribution of participants.

Although participants were encouraged to take a look at the suggested entity relationships first before searching for their own examples in the active learning condition, some subjects chose not to do so. When asked for why, the reasons were either that they didn't completely trust the computer or that they simply enjoyed the feeling of finding examples from scratch. In either case, the active learning suggestions provided hints for reducing their example finding efforts.

Subjective Feedback.

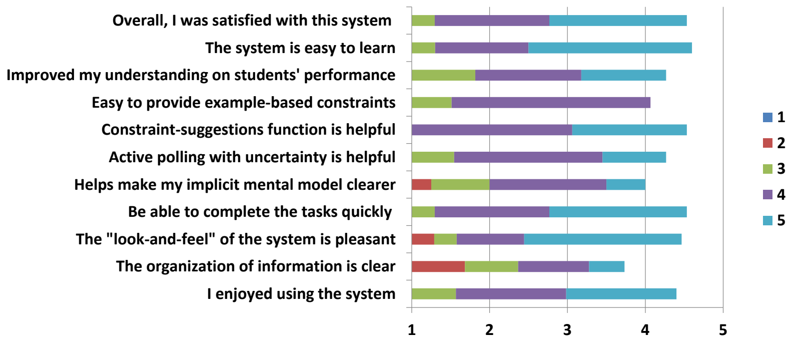


Fig. 7. Subjective ratings on a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree).

Overall, participants reported positive experiences with MindMiner. Participants felt that the system improved their understanding of students' performance through peer-review data (Fig. 7).

6 Conclusion

We presented MindMiner, a mixed-initiative interface to capture domain experts' subjective similarity measurements via a combination of new interaction techniques and machine learning algorithms. MindMiner collects qualitative, hard to express similarity measurements from users via *active polling with uncertainty*, *example based visual constraint creation* and *active learning*. MindMiner also formulates human prior knowledge into a set of inequalities and learns a quantitative similarity distance metric via convex optimization. In a 12-subject user study, we found that 1) MindMiner can capture the implicit similarity measurement from users via *examples collection* and *uncertainty polling*; 2) *active learning* could significantly improve the quality of distance metric learning when the same numbers of constraints were collected; 3) the *active polling with uncertainty* method could improve the task completion speed and result quality.

References

1. Amershi, S., Fogarty, J., Kapoor, A. and Tan, D.: Effective End-User Interaction with Machine Learning. In: Proceedings of the 25th AAAI Conference on Artificial Intelligence, pp. 1529-1532. AAAI, Menlo Park (2011)
2. Amershi, S., Fogarty, J., Kapoor, A. and Tan, D.: Overview Based Example Selection in End User Interactive Concept Learning. In: Proceedings of the 22nd annual ACM symposium on User interface software and technology, pp. 247-256. ACM, New York (2009)
3. Basu, S., Banerjee, A., Mooney, R. J.: Active Semi-Supervision for Pairwise Constrained Clustering. In: Proceedings of the 2004 SIAM International Conference on Data Mining, vol. 4, pp. 333-344. SIAM, Philadelphia (2004)
4. Basu, S., Fisher, D., Drucker, S. M., Lu, H.: Assisting Users with Clustering Tasks by Combining Metric Learning and Classification. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence, pp. 394-400. AAAI, Menlo Park (2010)
5. Boyd, S., Vandenberghe, L: Convex Optimization. Cambridge University Press, 2004.
6. Cao, N., Gotz, D., Sun, J., Qu, H.: DICON: Interactive Visual Analysis of Multidimensional Clusters. In: IEEE Transactions on Visualization and Computer Graphics, 17(12), pp. 2581-2590. IEEE Press, New York (2011)
7. Chau, D., Kittur, A., Hong, J. and Faloutsos, C.: Apolo: making sense of large network data by combining rich user interaction and machine learning. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 167-176. ACM, New York (2011)
8. Cohn, D., Caruana, R., McCallum, A.: Semi-supervised clustering with user feedback. Constrained Clustering: Advances in Algorithms, Theory, and Applications, 4(1), 17-32.
9. Cohn, D., Atlas, L., Ladner, R.: Improving generalization with active learning. Machine Learning, 15(2), pp. 201-221, 1994.
10. DesJardins, M., MacGlashan, J., Ferraioli, J.: Interactive visual clustering. In: Proceedings of the 12th international conference on Intelligent user interfaces, pp. 361-364. ACM, New York (2007)
11. Dy, J. and Brodley, C.: Visualization and interactive feature selection for unsupervised data. In: Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 360-364. ACM, New York (2000)

12. Fogarty, J., Tan, D., Kapoor, A. and Winder, S.: CueFlik: interactive concept learning in image search. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 29-38. ACM, New York (2008)
13. Fua, Y. H., Ward, M. O., Rundensteiner, E. A.: Hierarchical parallel coordinates for exploration of large datasets. In: Proceedings of the conference on Visualization'99: celebrating ten years, pp. 43-50. IEEE Computer Society Press, New York (1999)
14. Henry, N., Fekete, J. and McGuffin. NodeTrix: a hybrid visualization of social networks. In: IEEE Transactions on Visualization and Computer Graphics, 13(6), pp. 1302-1309. IEEE Press, New York (2007)
15. Horvitz, E.: Principles of Mixed-Initiative User Interfaces. In: Proceedings of the SIGCHI conference on Human Factors in Computing Systems, pp. 159-166. ACM, New York (1999)
16. Huang, Y., Mitchell, T.: Exploring Hierarchical User Feedback in Email Clustering. In: EMAIL'08: Proceedings of the Workshop on Enhanced Messaging-AAAI, pp. 36-41. AAAI, Menlo Park (2008)
17. Inselberg, A., Dimsdale, B. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In: Proceedings of the 1st conference on visualization, pp. 23-26. IEEE Press, Washington (1990)
18. Kapoor, A., Lee, B., Tan, D., Horvitz, E.: Interactive optimization for steering machine classification. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1343-1352. ACM, New York (2010)
19. Novotny, M.: Visually effective information visualization of large data. In Proceedings of the 8th Central European Seminar on Computer Graphics, pp. 41-48. (2004)
20. Peter, A., Shneiderman, B.: Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In: Proceedings of the SIGCHI conference on Human Factors in Computing Systems, pp. 265-274. ACM, New York (2008)
21. Rattenbury, T., Canny, J.: CAAD. An Automatic Task Support System. In: Proceedings of the SIGCHI conference on Human Factors in Computing Systems, pp. 687-696. ACM, New York (2007)
22. Robertson, J., Stats.: We're Doing It Wrong. http://cacm.acm.org/blogs/blog-cacm/107125-stats-were-doing-it-wrong/
23. Rosch, E., Mervis, C.: Family resemblances: Studies in the internal structure of categories. Cognitive Psychology, 7(4), pp. 573-605. Elsevier, Amsterdam (1975)
24. Seo, J., Shneiderman, B.: Interactively exploring hierarchical clustering results [gene identification]. Computer, 35(7), 80-86. IEEE, New York (2002)
25. Shneiderman, B., Maes, P.: Direct Manipulation vs. Interface Agents. Interactions, 4(6), 42-61. ACM, New York (1997)
26. Talbot, J., Lee, B., Kapoor, A., Tan, D.: EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1283-1292. ACM, New York (2009)
27. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S.: Constrained K-Means clustering with background knowledge. In: ICML, vol 1, pp. 577-584. 2001.
28. Xing, E., Ng, A., Jordan, M., Russell, S.: Distance Metric Learning with Application to Clustering with Side-Information. In Advances in neural information processing systems, pp. 505-512 (2002)