Dis-Function: Learning Distance Functions Interactively

Eli T. Brown* Jingjing Liu† Carla E. Brodley‡ Remco Chang§

Department of Computer Science
Tufts University

Abstract

The world's corpora of data grow in size and complexity every day, making it increasingly difficult for experts to make sense out of their data. Although machine learning offers algorithms for finding patterns in data automatically, they often require algorithm-specific parameters, such as an appropriate distance function, which are outside the purview of a domain expert. We present a system that allows an expert to interact directly with a visual representation of the data to define an appropriate distance function, thus avoiding direct manipulation of obtuse model parameters. Adopting an iterative approach, our system first assumes a uniformly weighted Euclidean distance function and projects the data into a two-dimensional scatterplot view. The user can then move incorrectly-positioned data points to locations that reflect his or her understanding of the similarity of those data points relative to the other data points. Based on this input, the system performs an optimization to learn a new distance function and then re-projects the data to redraw the scatterplot. We illustrate empirically that with only a few iterations of interaction and optimization, a user can achieve a scatterplot view and its corresponding distance function that reflect the user's knowledge of the data. In addition, we evaluate our system to assess scalability in data size and data dimension, and show that our system is computationally efficient and can provide an interactive or near-interactive user experience.

1 Introduction

The total body of collected data in the world is enormous and growing. Analyzing it is as valuable and important as it is difficult. Although powerful statistical analysis and machine learning tools exist for making sense of data, they are often complicated and require understanding outside the realm of a researcher's expertise to set model parameters. In particular, many data analysis methods such as clustering, retrieval and classification require the definition of a distance metric to define the distances/similarities among data points, which may not be intuitive for most domain experts to construct explicitly.

What is needed, then, is a system to bridge the space between the experts and the tools. In this paper we introduce an approach and prototype implementation, which we name Dis-Function, that allows experts to leverage their knowledge about data to define a distance metric. Using our system, an expert interacts directly with a visual representation of the data to define an appropriate distance function, thus avoiding direct manipulation of obtuse model parameters. The system first presents the user with a scatterplot of a projection of the data using an initial distance function. During each subsequent iteration, the expert finds points that are not positioned in accordance with his or her understanding of the data, and moves them interactively. Dis-Function learns a new distance function which incorporates the new interaction and the previous interactions, and then redisplays the data using the updated distance function.

In the remainder of this paper we first review the related work in learning a distance metric in the machine learning and visualization research literature. We then present the proposed approach which allows a user to implicitly describe a distance function over high-dimensional data by interacting with a visualization of the data. We present the results of experiments on a machine learning benchmark dataset with our prototype system to assess Dis-Function's ability to learn a distance metric for classification. In addition, we evaluate the system's ability to provide interactive or near-interactive speed and conclude that performance scales linearly in the number of dimensions and quadratically in the number of data points. We finish with a discussion of the potential of Dis-Function and future directions of research.

2 Related Work

This work leverages previous efforts in both machine learning and visualization. We begin with the machine learning work in distance functions, and then discuss interactive visualizations for high-dimensional data.

2.1 Machine Learning

In the last decade, increasing attention has been paid to learning a distance metric from data [2, 3, 9, 15, 31, 32, 33, 34, 35, 36, 39, 40]. These methods have been successfully applied to many real-world application domains including information retrieval, face verification and image recognition [8, 17, 21]. Methods in the machine learning literature assume that the learning method is given some "side information," most often in the form of pairwise constraints between instances. They assume that a domain expert can easily provide pairs of similar data points and pairs of dissimilar data points. An approximation to this information can be collected from the label information in supervised training datasets (by defining instances in the same class to be similar, and from distinct classes to be dissimilar).

Using this side information, existing methods seek to learn a distance metric such that the distance between similar examples should be relatively smaller than that between dissimilar examples. Although the distance metric can be a general function, the most prevalent one is the Mahalanobis metric defined by

$$D_A(x_i,x_j) = \sqrt{(x_i - x_j)^T A (x_i - x_j)}$$

where A is a positive semi-definite matrix and x_i and x_j are two instances in the data [37]. While existing methods have been proven to be effective, what they fail to adequately address is how such side information is obtained in the absence of class label information. Indeed, the majority of methods found in the machine learning literature are not truly interactive, but instead simulate user interaction. In contrast, our work provides an interactive visualization method for observing which instances are considered similar based on the current distance metric, and a way to directly manipulate the visualization to redefine similarity.

*e-mail: ebrown@cs.tufts.edu
†e-mail: jingjing.liu@tufts.edu
‡e-mail: brodley@cs.tufts.edu
§e-mail: remco@cs.tufts.edu

2.2 Visual Analytics and Machine Learning

In the visualization community, machine learning techniques have been used to project high-dimensional data into 2D information visualization for data exploration. Jeong, et al. [23] created a tool with a coordinated view between a projection of the data using principal component analysis (PCA) and parallel coordinates. The user can change the parameters of the projection interactively to explore the data. Similarly, Buja, et al. [6] created a tool with which a user can look at the data in a multi-dimensional scaling (MDS) projection and manipulate parameters directly to change the visualization. Dust and Magnets [38] and RadViz [20] layout high-dimensional points in a 2D visualization where the dimensions are anchored and their positions can be manipulated by the user to affect the display. These efforts demonstrate the effectiveness of combining interactive visualization with machine learning techniques. However, in these systems, the user's interaction is limited to modifying the parameters of the projection algorithm.

Several methods have been proposed that couple machine learning techniques with visualization to cluster or classify data. Nam, et al. [28] introduced ClusterSculptor, which allows the user to iteratively and interactively apply different clustering criteria to different parts of a dataset. Garg, et al. [14] use Inductive Logic Programming to learn rules based on user inputs. These rules can be stored and reused in other parts of the data to identify repeating trends and patterns. Andrienko, et al. [1] allow expert users to build classifiers of trajectories from sampled data, and interactively modify the parameters of the classifier at different stages in the analysis. Broekens, et al. [5] propose a system that allows a user to explore data in an MDS projection by dragging points around to affect clustering and layout. DesJardins, et al. [11] visualize data via a spring layout in which the user can interact with the visualization by pinning points in place. The pinned points are interpreted as constraints, and the constraints are used in a clustering analysis that results in a regenerated visualization that attempts to satisfy the constraints. Similarly, Endert, et al. [12] developed a spring-based system specific to text analysis, and developed a variety of interaction paradigms for affecting the layout. Choo, et al. [7] present iVisClassifier, which is a system based on supervised linear discriminant analysis that allows the user to iteratively label data and recompute clusters and projections. In all these systems, the user works closely with an automated machine learning algorithm through a visual interface to explore and better understand the data, but none of these systems explicitly addresses learning a distance function.

There have been some methods designed specifically to learn a distance function and select features. The interactive tool proposed by Okabe and Yamada [29] learns a distance function by allowing a user to interact with a 2D projection of the data. However this tool is restricted to clustering, and supports only pairwise constraints that are formed by requiring users to select pairs of points and specify whether or not they are in the same cluster. Thus the user is forced to make these decisions purely based on the 2D projection. In contrast, our method as described in Section 4 provides several coordinated views of the data and does not restrict the user to formulate only pairwise constraints. May, et al. [27] present the SmartStripes system which assists the user in feature subset selection by visualizing the dependencies and interdependencies between different features and entity subsets. This work is similar to ours in that both methods seek to identify relevant dimensions in a high-dimensional dataset. However, unlike the SmartStripes system that directly represents low-level statistical information of each feature, our approach hides the complex mathematical relationships in the features and allows the user to interact directly with the visual interface.

Perhaps most conceptually similar to our work is that by Endert, et al. [13], which presents variations of three projection techniques, including MDS, that can be updated based on user interaction. While their techniques are similar to ours, our approach emphasizes the externalization of a user's interactions to produce a useful, *exportable* distance function. Unlike the formulations of [13], our system produces distance functions that are simple enough that the user can observe the relative importance of features while interacting with the software.

3 Learning a Distance Function Interactively

Our approach to learning a distance function is both interactive and iterative. The user follows the procedure below until satisfied with the visualization, and thus with the *learned* underlying distance function.

1. Based on the current distance metric, we provide a two-dimensional scatterplot visualization of the data as well as other coordinated views (see Section 4).

2. The expert user observes and explores the provided visualizations and finds inconsistencies between the visualizations and his or her knowledge of the data. The user interacts with the scatterplot visualization via drag/drop and selection operations on data points with the mouse.

3. Dis-Function calculates a new distance function based on the feedback from the previous step. The new distance function is used to re-start the process at Step 1.

Figure 1 illustrates the process of iterating these three steps starting with the data as input, then making updates to the distance function until the user is satisfied with the 2D projection. In this section we describe our approach to each of these steps. We leave the details of the visualizations to the following section.

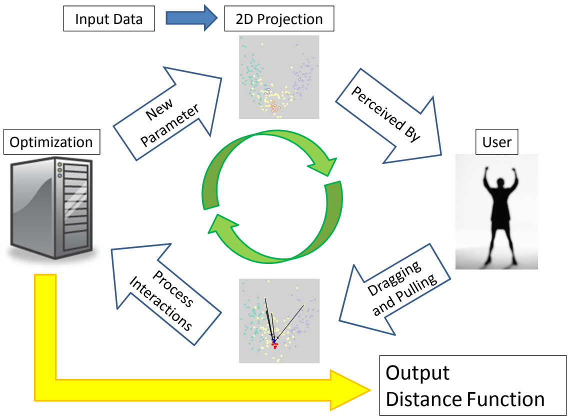


Figure 1: Flow chart showing the interactive process of using Dis-Function.

3.1 Producing a 2-D Scatterplot of the Data

To produce the two-dimensional scatterplot, we project the original (potentially high-dimensional) data to two dimensions via Multi-Dimensional Scaling (MDS) [4]. MDS has the property that when mapping from a high- to low-dimensional space, it preserves the relative distances between points. Thus, when a user looks to see if two points are the correct distance apart relative to others in the 2D projection, the relative distances between pairs of points observed in the projection correspond to their relative distance in the full-dimensional space as defined by the current distance metric. The

MDS projection is dependent on a distance function. The input is an $N \times N$ matrix D where each i, j entry contains the distance between points x_i and x_j from the set of all N data points in $\mathbb{R}^M$. (All notation used in this section is shown in Table 1).

Specifically, the projection algorithm accepts a pairwise distance matrix D covering all points, calculated with the "current" distance function. Note that in our experiments we set the initial distance function to be a uniformly-weighted Euclidean distance function across all possible features. Given the matrix D, we apply PCA to perform an eigenvector decomposition and compute the *principal components*, a ranked set of orthogonal vectors.[1] The top-ranked vector is the direction of highest variance in the distance matrix, and the second-ranked is the orthogonal vector that describes the next-most amount of variance. The data points, represented as vectors, are projected onto those two top-ranking principal components [24]. The final result is a set of N vectors in $\mathbb{R}^2$, one for each original data point. Using these new vectors we display the data as a scatterplot visualization.

3.2 User Input

In Section 4 we describe the expert's interaction with the data in more detail after we have presented the details of the visualization system. For now, we ask the reader to assume that the user is able to directly manipulate the scatterplot to define sets of points that should be nearer to one another or further apart. To this end, let us define two sets of data points Y_1 and Y_2, selected by the user, as sets of points which should be moved relative to each other. We then calculate a matrix U that will represent the user input when calculating an updated distance function as described in Section 3.3, where U is defined as follows:

$$U_{ij} = \begin{cases} \frac{intended\_distance}{original\_projected\_distance} & \text{if } (x_i, x_j) \in Y_1 \times Y_2, \\ 1 & \text{otherwise.} \end{cases} \tag{1}$$

where $original\_projected\_distance$ is computed as the Euclidean distance between points x_i and x_j in the scatterplot before the user moved them, and $intended\_distance$ is their Euclidean distance in the scatterplot after. Thus dragging data points x_i and x_j *closer* results in $U_{ij} < 1$, whereas dragging them *further* apart would result in $U_{ij} > 1$. These values will be used to compute a new distance function as described in the next section. Note that the majority of the values for $U_{i,j}$ will be equal to 1 because the interaction paradigm is that the user wants to change the relative distances between points in Y_1 and Y_2 *only*, wishing to maintain the relative distances of all other data points.

3.3 Updating the Distance Function

We incorporate user input to create a new distance function by solving an optimization problem over the space of possible distance functions. We use a weighted Euclidean distance function, i.e., Euclidean distance with each dimension of the data weighted by a coefficient. Although there are many other possibilities, we chose weighted Euclidean distance because it is easy for a human to map the magnitude of the weight of each feature to its relative importance. We describe in Section 4 how we present a visualization of the weights of the distance function to further help the user understand the data.

The distance between two points x_i and x_j is given by:

$$D(x_i, x_j|\Theta) = \sum_{k=1}^{M} \theta_k (x_{ik} - x_{jk})^2 \tag{2}$$

where M is the number of original dimensions in the data, Θ is the vector of feature weights, and θ_k is the weight for feature k. We initialize with all weights equal, i.e., $\theta_k = 1/M$.

Definitions used in describing our methods

N, M	Number of points, number of dimensions
$x_i \in \mathbb{R}^M$	Point i of the data
x_{ik}	Value of feature k of data point x_i
Θ	Vector in $\mathbb{R}^M$ containing the weight of each dimension for a distance function
θ_k	Weight of feature k in Θ
Θ^t and Θ^{t-1}	Indicate Θ values from before $(t-1)$ and after an optimization step
$D(x_i, x_j\|\Theta)$	Distance between x_i and x_j given parameters (dimension weight vector) Θ
δ_{ijk}	Abbreviation used in the gradient of the objective function as a stand-in for $(x_{ik} - x_{jk})^2$
$\mathbb{O}_{ijt}$	Abbreviation used in the gradient of the objective function for the square root of a term of the full objective function
L_{ij}	The impact coefficient in the objective function
U_{ij}	Entry in matrix U containing the user feedback information for the pair (x_i, x_j)

Table 1: Definitions of the symbols described in our methods.

To update Θ after a user interaction at time t, we seek to find the Θ^t that maintains the relative distances of points the user did not select while encouraging changes that affect the selected points in the desired direction. We formalize this intuition with the following optimization criterion:

$$\Theta^t = \arg\min_{\Theta^t} \sum_{i<j\leq N} L^t_{ij} \left(D(x_i, x_j|\Theta^t) - U^t_{ij} \cdot D(x_i, x_j|\Theta^{t-1}) \right)^2 \tag{3}$$

where $U^t_{i,j}$ is defined in Equation 1 and is the result of the user's interactions at round t based on the projection using the distance function defined by Θ^{t-1}. The term L^t_{ij}, defined in Equation 4, is a scalar weight that is greater than one when the points x_i and x_j are in Y^t_1 and Y^t_2, and one otherwise. In the summation over all points in the objective function of Equation 3, this increases the value of terms corresponding to points the user moved. We define L_{ij} at time t as:

$$L^t_{ij} = \begin{cases} \frac{N(N-1)}{|Y^t_1||Y^t_2|} - 1 & \text{if } (x_i, x_j) \in Y^t_1 \times Y^t_2, \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

where Y^t_1 and Y^t_2 are the sets of points in each user interaction set at iteration t. The value of the coefficient is the ratio of the number of unchanged pairs of points to the number of changed pairs. This heuristic and somewhat ad hoc weight is to ensure that the points the user selected have impact in the overall value of the objective function, even though the function is a sum over all points in the dataset, and Y^t_1 and Y^t_2 could be relatively small.

Our objective is to incorporate new user feedback at iteration t, while preserving the user's previous interactions. Previous iterations of feedback are not explicity represented. Instead, Equation 3 minimizes the difference, over all pairs of points, between the new distance and a multiple (U_{ij} from the user input) of the old distance. By including the old distance in the function and summing over all points, we provide some inertia against the user's updates. This was an important design decision, as machine learning methods for finding distance functions generally focus on a single set of constraints from the user and optimize once (with the exception of [2], which has an online version of the RCA algorithm).

[1] That is, we calculate an MDS projection by applying PCA to the pairwise distance matrix [16].

To find a solution to this optimization problem we use the method of conjugate gradient descent [19], which is an optimization method similar to hill-climbing: starting from an initial guess, the solver moves in steps toward a minimum of the objective function by walking along the gradient. At each step, the gradient is evaluated at the current guess, and a new guess is generated by moving in the direction of the gradient some small amount. This process continues until it converges. Although versions of the algorithm exist that determine step directions without the gradient, we provided the following gradient function to the solver for efficiency:

$$\nabla objective(\Theta) = \begin{bmatrix} \frac{\partial\Theta}{\partial\Theta_1} \\ \vdots \\ \frac{\partial\Theta}{\partial\Theta_M} \end{bmatrix} = \begin{bmatrix} 2\sum_{i<j\leq N}\delta_{ij1}\mathbb{O}_{ijt} \\ \vdots \\ 2\sum_{i<j\leq N}\delta_{ijM}\mathbb{O}_{ijt} \end{bmatrix} \tag{5}$$

where

$$\delta_{ijk} = (x_{ik} - x_{jk})^2$$

and

$$\mathbb{O}_{ijt} = L_{ij}\left(D(x_i,x_j|\Theta^t) - U^t_{ij}\cdot D(x_i,x_j|\Theta^{t-1})\right).$$

4 Visualization and User Interaction

Figure 2 shows Dis-Function, the prototype system introduced in this work. Dis-Function presents the user with three coordinated views of the data to aid in data exploration. Along the bottom of the window, seen in Figure 2E, user can see the raw data in a table with column labels. In Figure 2A, the interactive scatterplot visualization both displays data and captures user interaction. In 2C, a view we call *parallel bars* shows the user how the values of all the points in the dataset are distributed in each dimension. It appears as a bar graph with one bar for each dimension. The bars are each colored with a heat map to show how common each range of values along the bar is.

The three views are coordinated, which facilitates exploration [30]: selecting a point on the scatterplot causes the point to be moved into view in the data table and highlighted, as well as highlighted on the parallel bars view. The point is highlighted by a black line across each bar at the height corresponding to that point's value in the bar's dimension. Placing the mouse over an element in the data table causes the point to be highlighted in the scatterplot and parallel bars.

Together, these views allow a user to explore the data in order to provide more useful feedback to Dis-Function. Aside from just the relative distances among the points as shown in the scatterplot of the projection, the user can see the actual data in the original data space. Assuming the user has some domain knowledge, he or she will likely understand the implications of certain ranges of values in certain dimensions. The user can also observe from the parallel bars visualization how any data point fits into the scheme of the data on a dimensional basis. If a given point is an outlier in one or all dimensions, for example, that will be clear from the parallel bars visualization.

In addition to the views of the data, we provide two views of the distance function and the user's progress toward finding it. Figure 2D shows two tabs. The one visible in the figure shows a bar graph representation of the current distance function. Each bar represents a dimension, and the bar height encodes the weight of that dimension. Using the bar graph, the user can watch the distance function change after each feedback iteration. This allows the user to observe the relative importance of the different dimensions in the current distance function used to display the data in the scatterplot to the left. The hidden tab in Figure 2D contains a data table version of the same information, but includes history, and makes it easy to export the distance function from any iteration.

Having described how the data is visualized we now turn to how the user can interact with the data through this interface. Recall that the goal of the interaction is to define two sets of points that should be closer to one another, or further apart. To this end, the user can select points and drag-and-drop points to mark them as members of either set and to move them some amount closer together or further apart. The points in the two sets are marked by different colors in the scatterplot visualization, and they correspond to using the left or right mouse button when clicking or dragging points. These two sets of points, which we indicate by red and blue in the visualization, correspond to the two sets, Y^t_1 and Y^t_2 respectively. During the feedback step of each iteration, the user can select and unselect points, and repeatedly move points around. To signal completing one round of interaction, the expert clicks the *Moved Points* button (see 2B). At this point a new distance metric is learned from the feedback and the data is then reprojected using the new metric. Currently, the scatterplot and bar graph update as quickly as possible, without animation or optimizing for rotation. To provide context between iterations, after each iteration the user can see where the points in his or her input sets have been placed in the new projection via highlighting with colored rings (we illustrate this process in detail in the next section).

In the next section, we present empirical results of ten subjects interacting with Dis-Function and we provide preliminary experiments to assess its interactive speed. Our results show that our system is interactive or near-interactive for a standard machine leaning testing dataset.

5 Experiments and Results

In this section, we describe our evaluation of the effectiveness of Dis-Function at finding distance functions, the quality of distance functions learned by Dis-Function, and the time taken to perform each update as a function of the input. We begin with a presentation of the empirical results that demonstrate the efficacy of the proposed interaction method for defining sets of points to learn a distance metric.

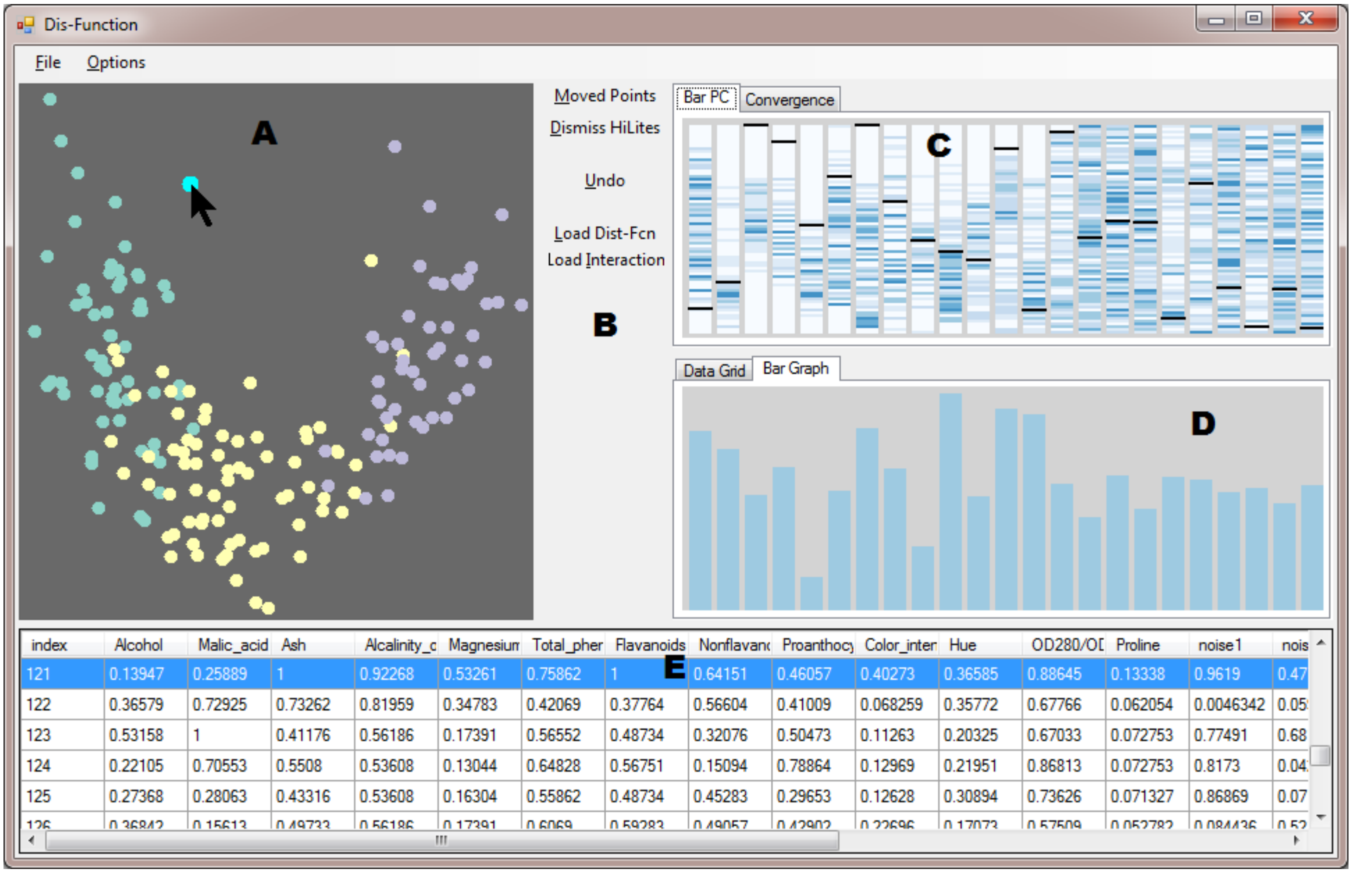
5.1 Empirical Results

We had ten subjects from Tufts University evaluate our software. In order to test software meant for experts in the absence of experts, we simulate the experience by coloring the data points in the scatter plot based on the known classes of the points; i.e., when the correct class membership of each point is visible, any user is an "expert" on the data.[2] We showed each participant how to use the software and let each perform as many iterations as desired. We performed our experiments on a modified version of the Wine dataset from the UCI Machine Learning repository [25] as this has been used in prior studies of defining a distance metric. The original Wine dataset has thirteen features and 178 instances, each labeled as one of three classes. We modified the Wine dataset as follows: we added ten noise features, each of which we generated by randomly choosing values from a uniform distribution over the range [0,1], matching the range of the data itself, which is normalized. We introduced these features in order to know exactly which features in the data were uninformative. We hypothesized that the user's interactions would result in a distance function giving these "useless" features a weight close to zero.

Because our users were given instant expertise in the form of data colored with class labels, we instructed them to provide feedback by moving points closer together that are in the same class (i.e., of the same color). In our experiments, we observed that all users quickly figured out that moving only a few points at a time did not result in significant changes to the distance function and further that moving points from class x that are far away from a class x cluster[3] to its

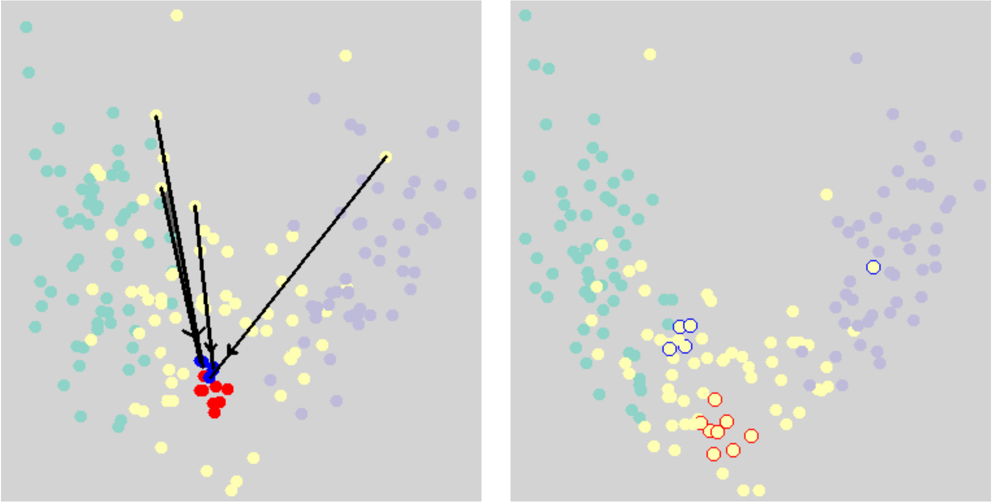
[2] Note that because the subjects interact based on class information, our experiments do not explicitly evaluate the efficacy of the coordinated visualizations.

[3] Note that there may be more than one cluster per class.



index	Alcohol	Malic_acid	Ash	Alcalinity_c	Magnesium	Total_pher	Flavanoids	Nonflavanc	Proanthocy	Color_inter	Hue	OD280/OE	Proline	noise1	nois
121	0.13947	0.25889	1	0.92268	0.53261	0.75862	1	0.64151	0.46057	0.40273	0.36585	0.88645	0.13338	0.9619	0.47
122	0.36579	0.72925	0.73262	0.81959	0.34783	0.42069	0.37764	0.56604	0.41009	0.068259	0.35772	0.67766	0.062054	0.0046342	0.05
123	0.53158	1	0.41176	0.56186	0.17391	0.56552	0.48734	0.32076	0.50473	0.11263	0.20325	0.67033	0.072753	0.77491	0.68
124	0.22105	0.70553	0.5508	0.53608	0.13044	0.64828	0.56751	0.15094	0.78864	0.12969	0.21951	0.86813	0.072753	0.8173	0.04
125	0.27368	0.28063	0.43316	0.53608	0.16304	0.55862	0.48734	0.45283	0.29653	0.12628	0.30894	0.73626	0.071327	0.86869	0.07
126	0.36842	0.15613	0.49733	0.56186	0.17391	0.6069	0.59283	0.49057	0.42902	0.22696	0.17073	0.57509	0.052782	0.084436	0.52

Figure 2: This screenshot shows Dis-Function comprising A) the MDS scatterplot visualization of the data; B) the buttons for recalculating the projection, undoing unsatisfying input, loading custom distance functions and user input data, etc.; C) the Parallel Bars visualization described in Section 4; D) a bar graph of the current distance function (obscured 'Data Grid' tab shows a tabular version); and E) the original data. All these views are tightly coordinated such that interactions with one view are immediately reflected on the others. For example, in the figure above, the mouse cursor is over a point in the scatterplot, and thus the corresponding point in the data table at the bottom is highlighted and the black lines on (C) highlight the values of the data point in each dimension as they relate to other data points.



(a)

(b)

Figure 3: These images show an example of how a user manipulates the visualization. A handful of points have been marked in blue and dragged closer to another set of points, marked in red. After the update (on the right), the points in those groups are closer together, and the clustering with respect to different colors is more compact. The same red and blue points marked on the left are indicated in their new positions on the right with red and blue halos.

center allows the system to converge more quickly. An example of a typical interaction is shown in Figure 3. The left side shows the original positions of data points with arrows indicating the user interaction; the user dragged the points from the start to the end of the arrow. The red and blue circles show the two sets of selected points. The right side shows the result of the reprojection of the data using the new distance function. The selected points have moved closer together and the clusters are more cohesive.

Our user study found all users were satisfied with the separation of different classes after 4–12 ($\mu = 7.3, \sigma = 2.5$) feedback updates. Figure 5 shows a sequence of updates where the augmented Wine dataset transitions from scattered to compact. Each step shown is after feedback (we do not show the user feedback step explicitly). The figure illustrates how the visualization changes with more user input. Note that the bar graph accompanying each scatterplot shows the weights of the dimensions in the distance function associated with the plot. Figure 4 shows the values of the dimension weights changing with each iteration for the same user as was used to generate Figure 5. Each sub-graph in Figure 4 shows the weight of a different dimension; the x-axis gives the iteration number and the y-axis shows the magnitude of the weight. Notice that the weights of the noisy features (the bottom ten) plunge steadily downward as was hypothesized; recall that these features were generated uniformly at random and thus provide no information about the classes in the data. In our experiment, all ten participants generated distance functions with low weights on these noisy features.

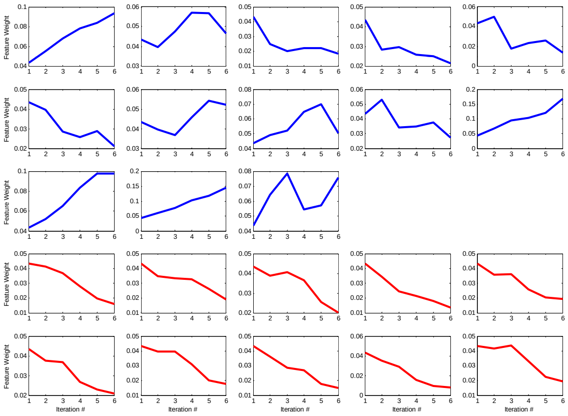


Figure 4: This figure shows the weight of each feature after each of User 10's five interactions. Each sub-graph shows a single feature. The x-axis gives the iteration number and the y-axis, the weight. The top thirteen correspond to the features in the original wine data and the bottom ten show the weights for the added noise features. Note that the weights of the added noise features quickly decrease and approach zero within a few iterations.

We evaluated the users' learned distance functions using a k-nearest-neighbor (k-NN) classifier. Recall that a k-NN classifier classifies a previously unseen (test) instance by taking the majority vote of the instance's k nearest neighbors, where "nearest" is calculated using a (weighted) Euclidean distance function. Thus we can evaluate the quality of the learned distance function using a leave-one-out cross-validation (LOOCV)[4] over the training data.

	k-NN Accuracy			
User	1	3	5	7
Even Weight	0.89	0.91	0.91	0.91
1	0.97	0.97	0.97	0.97
2	0.92	0.93	0.95	0.96
3	0.92	0.93	0.95	0.96
4	0.94	0.97	0.98	0.97
5	0.96	0.97	0.97	0.97
6	0.95	0.96	0.98	0.96
7	0.95	0.95	0.97	0.97
8	0.94	0.96	0.96	0.97
9	0.94	0.96	0.96	0.97
10	0.94	0.97	0.98	0.98

Table 2: Results of a leave-one-out cross-validation (LOOCV) for the Wine data using k-NN for $k = 1,3,5,7$. "Even Weight" is the baseline condition, i.e., an evenly-weighted Euclidean distance function without user interaction.

We show the results for $k = 1,3,5$ and 7 in Table 2. We note three observations from these results. First, all user-guided distance functions perform better than using the original unweighted Euclidean distance function. Second, performance is also a function of the user's ability as can be seen by the fact that users 2 and 3 performed worse than everyone else despite having the same directions. Finally, the Wine dataset is a relatively "easy" classification task in that our baseline accuracy is already 90%. We anticipate that for "harder" classification tasks we will see even more of a performance increase after user interaction.

5.2 Interactive Speed Performance

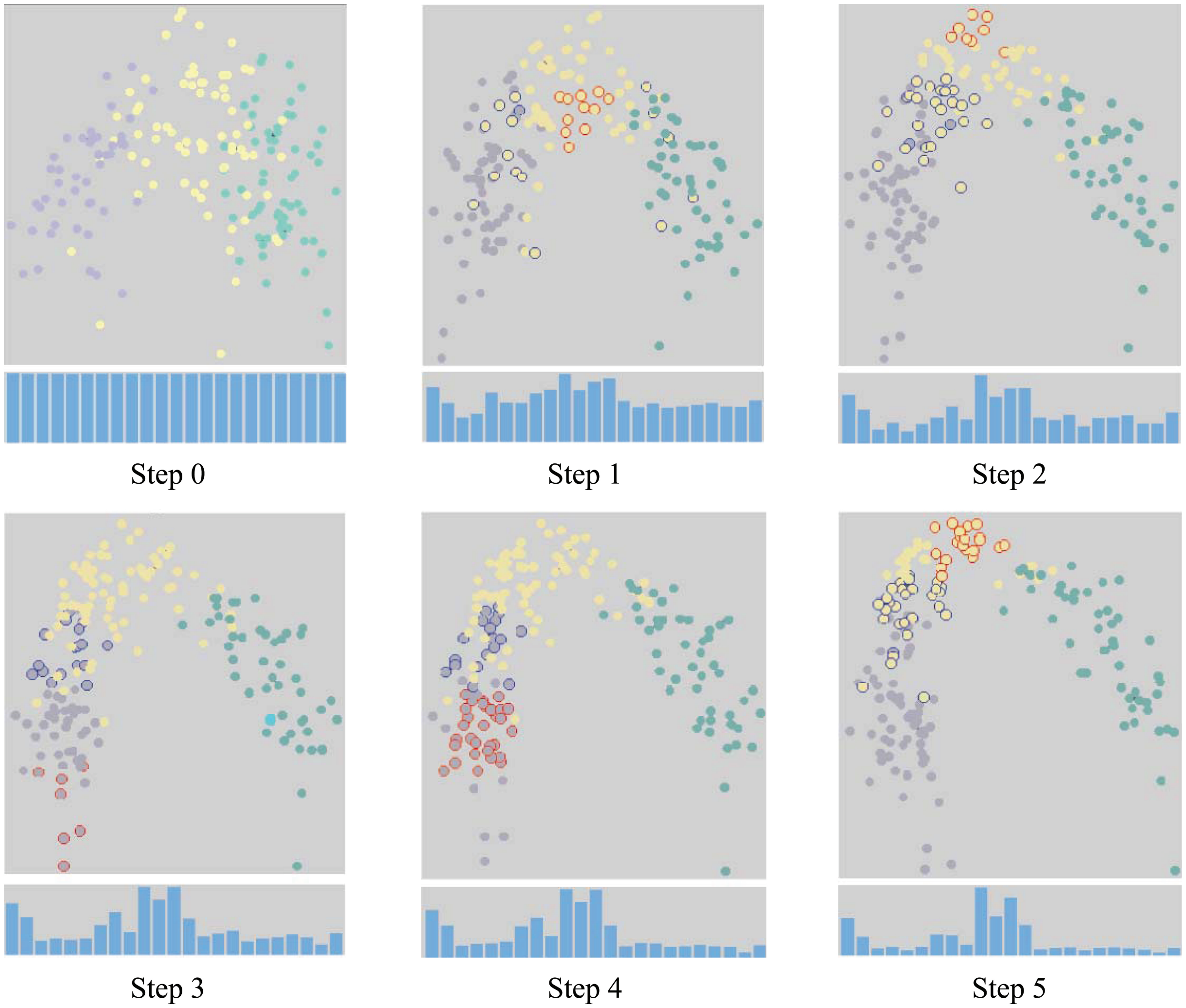
Using the Wine dataset, we find that user interactions with the visualization are fluid, and that updates based on user feedback take on the order of a second. In this section, we describe additional experiments to evaluate the scalability of Dis-Function in a controlled manner. Specifically, we examine the performance of Dis-Function as the dataset grows in size (in terms of number of rows) and in complexity (in number of dimensions) independently. Our experiment was conducted on a desktop computer with an AMD Phenom X3 processor and eight gigabytes of memory, running Windows 7 Home Premium. Our implementation of Dis-Function is in C#, using Windows Forms. The rendering is done in software using GDI+ (without using GPU hardware support), the PCA computation is done using the *PricipalComponentAnalysis.Compute* function in the Accord.NET Framework library,[5] and conjugate gradient is done using the *mincgoptimize* function from the C# ALGLIB library version 3.5.0.[6] At the time of the experiment, no other applications were running on the computer except for basic Windows services running in the background. In the remainder of this discussion, the reported performance is based on the amount of time required for Dis-Function to perform the optimization and re-projection, independent of the interface.

In the Dis-Function prototype, we include a stand-alone command-line executable that links against Dis-Function. This program allows us to write scripts that test different types of input and collect performance data. To test the dependence on data dimensionality, we extended the Wine dataset, which has 178 data points and 13 dimensions, up to 2000 dimensions. Those extra dimensions were filled with random numbers drawn from a uniform distribution over the range $[0,1]$, the same range as the original, normalized data. We ran our performance test repeatedly with all the data points, starting with only the real data dimensions (13), and cumulatively growing to the full 2000 dimensions. Figure 6 shows the

[4] In an LOOCV we hold out each instance one at a time, and use the rest of the data to form our k-NN classifier.

[5] http://code.google.com/p/accord/

[6] www.alglib.net



Step 0

Step 1

Step 2

Step 3

Step 4

Step 5

Figure 5: While Figure 3 demonstrates one step of feedback, this figure shows how the scatterplot visualization improves through a number of iterations of feedback (matching those of Figure 4). Each scatterplot shows the visualization after a round of feedback. The bar graph below each plot shows the distance function used to create the projection shown above it. Each bar represents a different dimension, and collectively they show the relative weights of the dimensions in the distance function. In each frame, the sets Y_1 and Y_2 from the previous interaction are highlighted with red and blue halos.

results of this experiment: the dependence of the *optimization* time on the number of dimensions (Figure 6 (a)), and the dependence of the *re-projection* time on the number of dimensions (Figure 6 (b)).

To evaluate the performance in data size, we randomly generated a 2000-element dataset with two dimensions, and used sequential subsets of it to create datasets of different sizes. Figure 7 (a) shows the time taken by the *optimization* as a function of the number of data points, and Figure 7 (b) shows the time taken by the *re-projection* as a function of the number of data points.

Both optimization and projection scale the same way: linearly in the number of dimensions and quadratically in the number of data points. The graphs include trend lines fit by Microsoft Excel, and in all cases the correlation is high, as seen in the figures. These results are aligned with our expectations because the conjugate gradient method can be expected to converge in as many steps as there are dimensions. In terms of number of data points, the calculations are dependent on pairwise distances, which number $O(N^2)$.

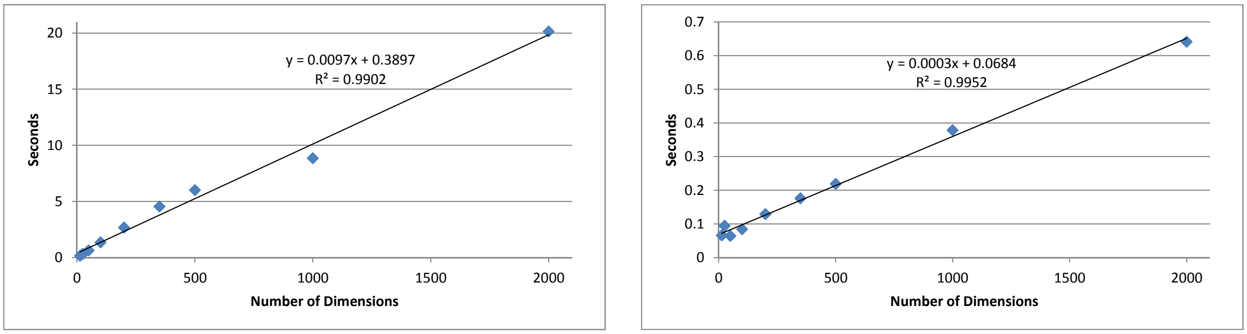
Although the performance as it stands makes Dis-Function comfortable to use, we believe the performance of the re-projection step can be improved substantially by introducing online singular value decomposition (SVD) into our PCA calculation, similar to the approach of Jeong, et al. [23]. Using online SVD would allow us to calculate the eigenvalues at each projection step incrementally. Another option for fast eigenvalue calculation is power iteration [18]. Separately, we could improve the performance of the optimization step by stopping it early: empirically we have noticed a good solution is reached in only a few steps. Truncating the number of steps the optimizer is allowed would sacrifice only a small amount of precision and speed up the software's response to user input.

6 Discussion

In this section we discuss Dis-Function as a general purpose data analytics tool, propose future work, and provide some usage guidelines.

6.1 Broad and Flexible Use

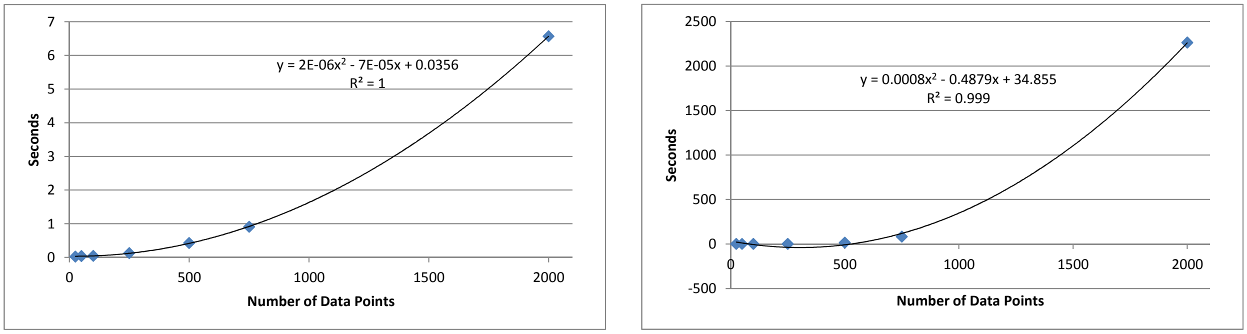
What we have presented in Dis-Function is a prototype for a widely-applicable data analytics tool. The distance functions pro-



(a)

(b)

Figure 6: Performance, as affected by *data complexity* (number of dimensions), of processing user feedback for one iteration by (a) running optimization to find a new distance function and (b) re-projecting data for the scatterplot. Notice that both operations scale linearly in data dimensionality.



(a)

(b)

Figure 7: Performance, as affected by *data size* (number of points), of processing user feedback for one iteration by (a) running optimization to find a new distance function and (b) re-projecting data for the scatterplot. Notice that both operations scale quadratically in data size.

duced by Dis-Function provide a form of knowledge externalization that quantifies expert notions of a data space. By assigning numerical weights to each dimension indicating relative importance, the learned distance function can also serve the purpose of feature selection. A user may discard features with a relatively low weight, thereby reducing the dimensionality of a large and complex dataset in order to make it easier for a user to explore and analyze.

Because a distance function is a representation of an expert's intention, if the expert has more than one intention, he or she can use Dis-Function to create multiple distance functions, each reflecting a different analysis hypothesis. For example, if a researcher wants to study subjects in two different contexts such as socioeconomic similarity and physiological similarity, he or she can run Dis-Function twice to produce two distance functions. The first time, the researcher moves points with similar socioeconomic background closer; the second time, the researcher drag points with similar physiological makeup together. Both resulting distance functions can be used in additional computational analysis, perhaps comparing how each clusters the data. (Recall that one can use the learned distance function with clustering algorithms such as *k*-means [26] or EM [10]).

6.2 Possible Extensions

Thinking of Dis-Function as a framework instead of just a prototype opens some exciting possibilities for capturing different types of expertise and exploring ways to express knowledge by interacting directly with a visualization. We have provided only one simple mechanism for capturing user input.

More techniques for incorporating user input will be tied to introducing different visualizations amenable to similar “semantic interactions” [12]. The goal is to find visualizations where we can discover a mapping between some manipulation of the view and a semantic meaning for the user, and where that meaning can be translated into mathematics for adjusting the generation of the visualization. Not only could we offer different types of projections, but we can learn distance functions for other types of data. For example, when representing hierarchical data using a phylogenetic tree, the framework of Dis-Function can be immediately applied because a phylogenetic tree is also generated from pairwise distance data.

We can experiment with completely different classes of visualization like parallel coordinates [22], RadViz [20], and Dust and Magnets [38], for which tools exist for exploring data by manipulating the parameters. Dis-Function could allow an expert to use those tools to discover similar data points, and then model that feedback to stretch dimensions for improved visualization.

6.3 Usage Tips

Our own informal experimentation revealed some best-practice ways of interacting with Dis-Function. While the semantic meaning of separating dissimilar points is clear, the optimization we have used to learn a distance function is not designed for such feedback. As an example, consider moving two points together: they can only move in one direction: toward each other. On the other hand, when specifying that two points should be further apart, the two points can be moved in any direction. Indeed, when separating groups of points, Dis-Function occasionally introduces re-orientation of all data points in a way that is difficult to correlate to the previous layout. In some cases, this behavior is desirable – for example to separate tightly overlapping clusters. However, in most cases, it makes sense to perform the transformation "move set A further from set B" as two iterations of feedback by moving points closer: move A closer to points far from B, then B closer to points far from A. This way it is clearer in which direction to spread sets A and B.

7 Conclusion

In this paper we presented a prototype implementation, named Dis-Function, that allows a user to interact with a visualization to define a custom distance function. In particular, through a set of coordinated views, the user can explore data and find points to drag closer together. Based on a series of these interactions, the system learns a weighted Euclidean distance function that can be used in any data analysis algorithm requiring the definition of a distance function. The weights are human-readable as importance ratings of each dimension, giving the user a way to understand what facets of the data are most relevant. We demonstrated the scalability of Dis-Function in both data size and complexity, and illustrated empirically by using a well-known dataset that an expert user could use Dis-Function to build a distance function that can be used to improve classification or clustering.

References

[1] G. Andrienko, N. Andrienko, S. Rinzivillo, M. Nanni, D. Pedreschi, and F. Giannotti. Interactive visual clustering of large collections of trajectories. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 3–10. IEEE, 2009.

[2] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, pages 937–965, 2005.

[3] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine Learning*, pages 81–88, 2004.

[4] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer Verlag, 2005.

[5] J. Broekens, T. Cocx, and W. Kosters. Object-centered interactive multi-dimensional scaling: Ask the expert. In *Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, pages 59–66, 2006.

[6] A. Buja, D. Swayne, M. Littman, N. Dean, and H. Hofmann. Interactive data visualization with multidimensional scaling. *Report, University of Pennsylvania*, 2004.

[7] J. Choo, H. Lee, J. Kihm, and H. Park. iVisClassifier: An interactive visual analytics system for classification based on supervised dimension reduction. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 27–34. IEEE, 2010.

[8] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 539–546, 2005.

[9] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon. Information-theoretic metric learning. In *Proceedings of International Conference on Machine Learning*, pages 209–216, 2007.

[10] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, pages 1–38, 1977.

[11] M. Desjardins, J. MacGlashan, and J. Ferraioli. Interactive visual clustering. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, pages 361–364. ACM, 2007.

[12] A. Endert, P. Fiaux, and C. North. Semantic interaction for visual text analytics. In *Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems (CHI)*, pages 473–482. ACM, 2012.

[13] A. Endert, C. Han, D. Maiti, L. House, and C. North. Observation-level interaction with statistical models for visual analytics. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 121–130. IEEE, 2011.

[14] S. Garg, J. Nam, I. Ramakrishnan, and K. Mueller. Model-driven visual analytics. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 19–26. IEEE, 2008.

[15] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, pages 513–520, 2004.

[16] A. N. Gorban, B. Kgl, D. C. Wunsch, and A. Zinovyev. *Principal Manifolds for Data Visualization and Dimension Reduction*. Springer Publishing Company, Incorporated, 2007.

[17] M. Guillaumin, J. Verbeek, and C. Schmid. Is that you? Metric learning approaches for face identification. In *Proceedings of the International Conference on Computer Vision*, pages 498–505, 2009.

[18] M. Heath. *Scientific Computing*. The McGraw-Hill Companies, Incorporated, 2001.

[19] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards (NBS)*, 1952.

[20] P. Hoffman, G. Grinstein, and D. Pinkney. Dimensional anchors: A graphic primitive for multidimensional multivariate information visualizations. In *Proceedings of the 1999 Workshop on New Paradigms in Information Visualization and Manipulation in Conjunction With the Eighth ACM International Conference on Information and Knowledge Management*, pages 9–16. ACM, 1999.

[21] S. Hoi, W. Liu, M. Lyu, and W. Ma. Learning distance metrics with contextual constraints for image retrieval. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2072–2078, 2006.

[22] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Proceedings of the 1st Conference on Visualization*, pages 361–378. IEEE, 1990.

[23] D. Jeong, C. Ziemkiewicz, B. Fisher, W. Ribarsky, and R. Chang. iPCA: An interactive system for PCA-based visual analytics. *Computer Graphics Forum*, pages 767–774, 2009.

[24] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[25] C. A. M. Forina, R. Leardi and S. Lanteri. PARVUS: An extendable package of programs for data exploration, classification and correlation. *Journal of Chemometrics*, pages 191–193, 1988.

[26] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[27] T. May, A. Bannach, J. Davey, T. Ruppert, and J. Kohlhammer. Guiding feature subset selection with an interactive visualization. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 111–120. IEEE, 2011.

[28] E. Nam, Y. Han, K. Mueller, A. Zelenyuk, and D. Imre. ClusterSculptor: A visual analytics tool for high-dimensional data. In *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 75–82. IEEE, 2007.

[29] M. Okabe and S. Yamada. An interactive tool for human active learn-

ing in constrained clustering. *Journal of Emerging Technologies in Web Intelligence*, 3(1):20–27, Feb. 2011.

[30] J. Roberts. State of the art: Coordinated multiple views in exploratory visualization. In *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV)*, pages 61–71, July 2007.

[31] R. Rosales and G. Fung. Learning sparse metrics via linear programming. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 367–373, 2006.

[32] C. Shen, J. Kim, L. Wang, and A. Hengel. Positive semidefinite metric learning with boosting. In *Advances in Neural Information Processing Systems 22*, pages 1651–1659, 2009.

[33] L. Torresani and K. C. Lee. Large margin component analysis. In *Advances in Neural Information Processing Systems*, pages 1385–1392, 2007.

[34] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems*, pages 10:207–244, 2006.

[35] K. Weinberger and L. Saul. Fast solvers and efficient implementations for distance metric learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1160–1167, 2008.

[36] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pages 505–512, 2002.

[37] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. *Michigan State Universiy*, pages 1–51, 2006.

[38] J. S. Yi, R. Melton, J. Stasko, and J. A. Jacko. Dust and Magnet: Multivariate information visualization using a magnet metaphor. *Information Visualization*, pages 239–256, 2005.

[39] Y. Ying, K. Huang, and C. Campbell. Sparse metric learning via smooth optimization. In *Advances in Neural Information Processing Systems 22*, pages 2214–2222, 2009.

[40] Y. Ying and P. Li. Distance metric learning with eigenvalue optimization. In *Journal of Machine Learning Research*, pages 1–26, 2012.