Beyond the Third Dimension: Visualizing High-dimensional Data with Projections

Renato R.O. da Silva | University of Sao Paulo, Brazil
Paulo E. Rauber and Alexandru C. Telea | University of Groningen, The Netherlands

Many application fields produce large amounts of multidimensional data. Simply put, these are datasets where, for each measurement point (also called data point, record, sample, observation, or instance), we can measure many properties of the underlying phenomenon. The resulting measurement values for all data points are usually called variables, dimensions, or attributes. A multidimensional dataset can thus be described as an $n \times m$ data table having n rows (one per observation) and m columns (one per dimension). When n is larger than roughly 5, such data is called high-dimensional. Such datasets are common in engineering (think of manufacturing specifications, quality assurance, and simulation or process control); medical sciences and e-government (think of electronic patient dossiers [EPDs] or tax office records); and business intelligence (think of large tables in databases).

While *storing* multidimensional data is easy, *understanding* it is not. The challenge lies not so much in having a large number of observations but in having a large number of dimensions. Consider, for instance, two datasets A and B. Dataset A contains 1,000 samples of a single attribute, say, the birthdates of 1,000 patients in an EPD. Dataset B contains 100 samples of 10 attributes, say, the amounts of 10 different drugs distributed to 100 patients. The total number of measurements in the two datasets is the same (1,000). Yet, understanding dataset A is quite easy, and it typically involves displaying either a (sorted) bar chart of its single variable or a histogram showing the patients' age distribution. In contrast, understanding dataset B can be very hard—for example, it

might be necessary to examine the correlations of *any* pair of two dimensions of the 10 available ones.

In this article, we discuss *projections*, a particular type of tool that allows the efficient and effective visual analysis of multidimensional datasets. Projections have become increasingly interesting and important tools for the visual exploration of high-dimensional data. Compared to other techniques, they scale well in the number of observations and dimensions, are intuitive, and can be used with minimal effort. However, they need to be complemented by additional visual mechanisms to be of maximal added value. Also, as they've been originally developed in more formal communities, they're less known or accessible to mainstream scientists and engineers. We provide here a compact overview of how to use projections to understand high-dimensional data, present a classification of projection techniques, and discuss ways to visualize projections. We also comment on the advantages of projections as opposed to other visualization techniques for multidimensional data, and illustrate their added value in a complex visual analytics workflow for machine learning applications in medical science.

Exploring High-Dimensional Data

Before outlining *solutions* for exploring high-dimensional data, we need to outline typical *tasks* that must be performed during such exploration. These can be classified into observation-centric tasks (which address questions focusing on observations) and dimension-centric tasks (which address questions focusing on the dimensions). Observation-centric tasks include finding groups of similar observations and finding outliers (observations that are very different from the rest of the data). Dimension-centric tasks include finding sets of dimensions that are strongly correlated and dimensions that are mutually independent. There exist also tasks that combine observations and dimensions, such as finding which dimensions make a given group of observations different from the rest of the data. Several visual solutions exist to address (parts of) these tasks, as follows. More details on these and other visualization techniques for high-dimensional data appear elsewhere.[1,2]

Tables

Probably the simplest method is to display the entire dataset as a $n \times m$ table, as we do in a spreadsheet. Sorting rows on the values in a given column lets us find observations with minimal or maximal values for that column and then read all their dimensions horizontally in a row. Visually scanning a sorted column lets us see the distribution of values of a given dimension.

But while spreadsheet views are good for showing detailed information, they don't scale to datasets having thousands of observations and tens of dimensions or more. To address such scalability, table lenses refine the spreadsheet idea: they work much like zooming out of the drawing of a large table, thereby reducing every row to a row of pixels. Rather than showing the actual textual cell content, cell values are now drawn as horizontal pixel bars colored and scaled to reflect data values. As such, columns are effectively reduced to bar graphs. Using sorting, we can now view the variation of dimension values for much larger datasets. However, reasoning about the correlation of different dimensions isn't easy using table lenses.

Scatterplots

Another well-known visualization technique for multidimensional data is a scatterplot, which shows the distribution of all observations with respect to two chosen dimensions i and j. Finding correlations, correlation strengths, and the overall distribution of data values is now easy. To do this for m dimensions, a so-called $m \times m$ scatterplot matrix can be drawn, showing the correlation of each dimension i with each other dimension j. However, reasoning about observations is hard now—an observation is basically a set of m^2 points, one in each scatterplot in the matrix. Also, scatterplot matrices don't scale well for datasets having more than roughly 8 to 10 dimensions

Parallel Coordinates

A third solution for visualizing multidimensional data is parallel coordinates. Here, each dimension is shown as a vertical axis, thus the name *parallel coordinates*. Each observation is shown as a fractured line that connects the m points along these axes corresponding to its values in all the m dimensions. Correlations of dimensions (shown by adjacent axes) can now be spotted as bundles of parallel line segments; inverse correlations are shown by a typical x-shaped line-crossing pattern. Yet, parallel coordinates don't scale well beyond 10 to 15 dimensions. Also, they might require careful ordering of the axes to bring dimensions that one wants to compare close to each other in the plot.

Multidimensional Projections

Projections take a very different approach to visualizing high-dimensional data. Think of the n data

Data table 2D projection

a table row gets
mapped to a point

2D point distance reflects
nD row distance
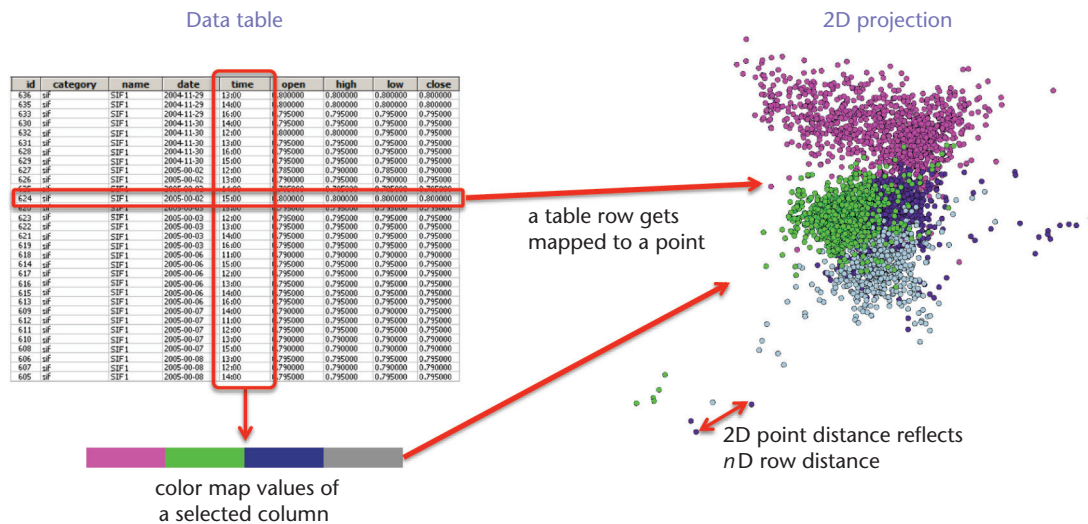
color map values of
a selected column

Figure 1. From a multivariate data table to a projection. Projections can be thought of as reducing the unnecessary dimensionality of the data (the original m dimensions) keeping the inherent dimensionality (that which encodes distances, or similarities, between points).

points in an m-dimensional space. The dataset can then be conceptually seen as a point cloud in this space. If we could see in m dimensions, we could then (easily) find outliers as those points that are far from all other points in the cloud and find important groups of similar observations as dense-and-compact regions in the point cloud.

However, we can't see in more than three dimensions. Note also that a key ingredient of performing the above-mentioned tasks is reasoning in terms of *distances* between the points in m dimensions. Hence, if we could somehow map, or project, our point cloud from m to two or three dimensions, keeping the distances between point-pairs, we could do the same tasks by looking at a 2D or 3D scatterplot. Projections perform precisely this operation, as illustrated by Figure 1. Intuitively, they can be thought of as reducing the unnecessary dimensionality of the data (the original m dimensions) keeping the inherent dimensionality (that which encodes distances, or similarities, between points). Additionally, we can color-code the projected points by the values of one dimension, to get extra insights.

There are two main use cases for projections. The first is to reduce the number of dimensions by keeping only one dimension from a set of dimensions which are strongly correlated, or by dropping dimensions along which the data has a very low variance. Essentially, this preserves patterns in the data (clusters, outliers) but makes its usage simpler,

as there are fewer dimensions to consider next. The simplified dataset can next be used instead of the original one in various processing or analysis tasks. The second use case involves reducing the number of dimensions to two or three, so that we can visually explore the reduced dataset. In contrast to the first case, this usually isn't done by dropping dimensions but by creating two or three synthetic dimensions along which the data structure is best preserved. We next focus on this latter use case.

Projection Techniques
Many different techniques exist to create a 2D or 3D projection, and they can be classified according to several criteria, as follows.

Dimension versus distance. The dimension versus distance classification looks at the type of information used to construct a projection. Distance-based methods use only the distances, or similarities, between m-dimensional observations. Typical distances here are Euclidean and cosine, thus, the projection algorithm's input is an $n \times n$ distance matrix between all observation pairs. Such methods are also known as multidimensional scaling (MDS) because they intuitively scale the m-dimensional distances to 2D distances. Technically, this is done by optimizing a function that minimizes the so-called *aggregated normalized stress*, or summed difference between the inter-point distances in m dimensions and 2D, respectively. The main advantage of MDS methods is

that they don't require the original dimensions—a dissimilarity matrix between observations is sufficient and extremely useful in cases where we can measure the similarities in some data collections but don't precisely know which attributes (dimensions) explain those similarities. The main disadvantage of MDS methods is that they require storing (and analyzing) an $n \times n$ distance matrix. For n being tens of thousands of observations, this can be very expensive.[3] Several MDS refinements have been proposed, such as ISOMAP,[4] Pivot MDS,[5] and Fastmap,[6] which can compute projections in (near) linear time to the number of observations.

In contrast, dimension-based methods use as input the actual m dimensions of all observations. For datasets having many more observations than dimensions (n much larger than n), this gives considerable savings. However, we now need to have access to the original dimension values. Arguably the best known method in this class is principal component analysis (PCA), whose variations are also known under the names of singular value decomposition (SVD) or Karhunen-Loève transform (KLT).[7] Intuitively put, the idea of 2D PCA is to find the plane, in m dimensions, on which the projections of the n observations have the largest spread. Visualizing these 2D projections will then give us a good way of understanding the actual variance of the data in m dimensions.[8] While simple and fast, PCA-based methods work well only if the observations are distributed close to a planar surface in m dimensions. To understand this, consider a set of observations uniformly distributed on the surface of the Earth (a ball in 3D). When projecting these, PCA will effectively squash the ball to a planar disk, projecting diametrically opposed observations on the ball's surface to the same location, meaning the projection won't preserve distances. What we actually want is a projection that acts much as a map construction process, where the Earth's surface is unfolded to a plane, with minimal distortions.

Global versus local. The global versus local classification looks at the type of operation used to construct a projection. Global methods define a single mapping, which is then applied for all observations. MDS and PCA methods fall in this class. The main disadvantage of global methods is that it can be very hard to find a single function that optimally preserves distances of a complex dataset when projecting it (as in the Earth projection example). Another disadvantage is that computing such a global mapping can be expensive (as in the case of classical MDS). Local methods address both these issues, selecting a (small) subset of observations, called representatives, from the initial dataset and then projecting these by using a high-accuracy method. This isn't expensive, as the number of representatives is small. Finally, the remaining observations close to each representative are fit around the position of the representative's projection. This is cheaper, simpler, and also more accurate than using a global technique. Intuitively, think of our Earth example as splitting the ball surface into several small patches and projecting these to 2D. When such patches have low curvature, fitting them to a 2D surface is easier than if we were to project the entire ball at once. Good local methods include PLMP[9] and LAMP.[10] Using representatives has another added value: users can arrange these as desired in 2D, thereby controlling the projection's overall shape with little effort.

Distance versus neighborhood preserving. A final classification looks into what a projection aims to preserve. When it's important to accurately assess the similarity of points, distance preservation is preferred. All projection techniques listed above fall into this class. However, as we've seen, getting a good distance preservation for all points can be hard. When the number of dimensions is very high, the Euclidean (straight-line) distances between all point-pairs in a dataset tend to become very similar, so accurately preserving such distances has less value. In such cases, it's often better to preserve *neighborhoods* in a projection—this way, the projection can still be used to reason about the groups and outliers existing in the high-dimensional dataset. Actually, the depiction of groups could get even clearer because the projection algorithm has more freedom to place observations in 2D, as long as the nearest-neighbors of a point in 2D are the same as those of the same point in m dimensions. The best-known method in this class is t-stochastic neighbor embedding (t-SNE), which is used in many applications in machine learning, pattern recognition, and data mining, and has a readily usable implementation (https://lvdmaaten.github.io/tsne).

Type of data. Most projection methods handle quantitative dimensions, whose values are typically continuously varying over some interval. Examples are temperature, time duration, speed, volume, or financial transaction values. However, projection techniques such as multiple correspondence analysis (MCA) can also handle categorical data (types) or mixed datasets of quantitative and categorical data. A good description of MCA and related techniques is given by Greenacre.[11]
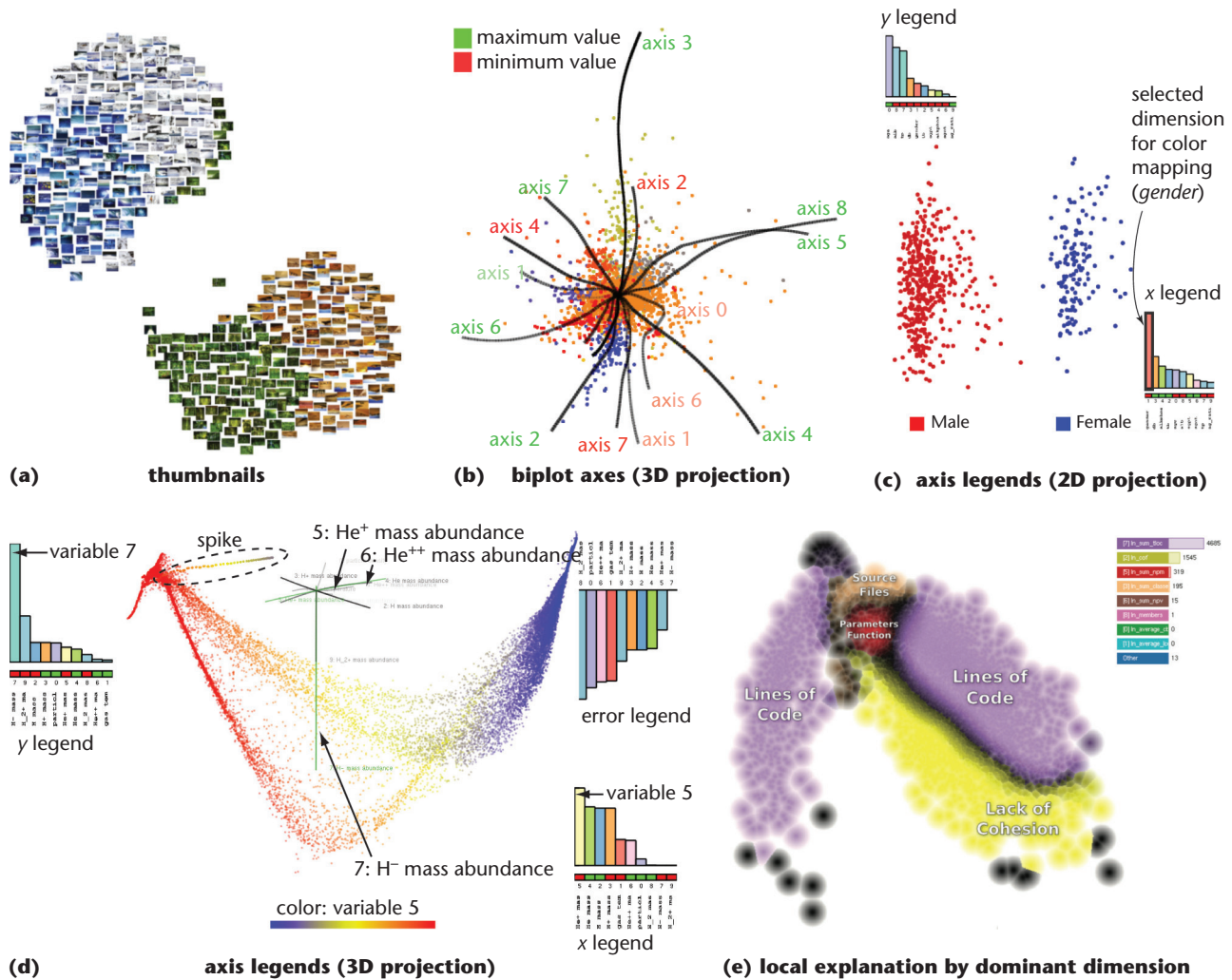
Figure 2. Projection visualizations with (a) thumbnails, (b) biplot axes, (c) and (d) axis legends, and (e) key local dimensions.

The Projection Explorer is a very good place to start working with projections in practice.[12] This tool implements a wide range of state-of-the-art projection techniques that can handle hundreds of thousands of observations with hundreds of dimensions and provides several visualizations to interactively customize and explore projections. The tool is freely downloadable from http://infoserver.lcad.icmc.usp.br/infovis2/Tools.

Visualizing Projections

The simplest and most widespread way to visualize a projection is to draw it as a scatterplot. Here, each point represents an observation, and the 2D distance between points reflects the similarities of the observations in m dimensions. Points can be also annotated with color, labels,

or even thumbnails to explain several of their dimensions.

Figure 2a shows this for a dataset where observations are images. The projection shows image thumbnails, organized by similarity. We can easily see here that our image collection is split into two large groups; we can get more insight into the composition of the groups by looking at the thumbnails.

However, in many cases, there's no easy way to draw a small thumbnail-like depiction of all the m attributes of an observation. Projections will then show us groups and outliers, but how do we explain *what* these mean? In other words, how do we put the dimension information back into the picture? Without this, the added value of a projection is limited.

There are several ways of explaining projections. By far the simplest, and most common, is to color code the projection points by the value of a user-chosen dimension. If we next see strong color correlations with different point groups in the projection, we can explain these in terms of the selected dimension's specific values or value ranges. However, if we have tens of dimensions, using each one to color code the projection is tedious at best. Moreover, it could be that no *single* dimension can explain why certain observations are similar. Tooltips can be shown at user-chosen points, which does a good job explaining a few outliers one by one, but it doesn't work if we want to explain a large number of points together.

One early way to explain projections is to draw so-called *biplot axes*.[13] For PCA projections and variants, lines indicate the directions of maximal variation in the 2D space of all m dimensions. Intuitively put, biplot axes generalize the concept of a scatterplot, where we can read the values of two dimensions along the x and y axes, to the case where we have m dimensions. Moreover, strongly correlated dimensions appear as nearly parallel axes, and independent dimensions appear as nearly orthogonal axes. Finally, the relative lengths of the axes indicate the relative variation of the respective dimensions. Biplots can also be easily constructed for any other projection, including 3D projections that generate a 3D point cloud rather than a 2D scatterplot.[14] In such cases, the biplot axes need not be straight lines. Figure 2b shows an example of biplot axes for a dataset containing 2,814 abstracts of scientific papers. Each observation (abstract) has nine dimensions, indicating the frequencies of the nine most used technical terms in all abstracts. The projection, created using a force-based technique, places points close to each other if the respective abstracts are similar. Labels can be added to the axes to tell their identity and also indicate their signs (extremities associated to minimum and maximum values). The curvature of the biplot axes tells us that the projection is highly nonlinear—intuitively, we can think that the nine-dimensional space gets distorted when squashed into the resulting 3D space. This is undesirable because reading the values of the dimensions along such curved axes is hard.

Still, interpreting biplot axes can be challenging, especially when we have 10 or more variables, as we get too many lines drawn in the plot. Moreover, most users are accustomed to interpreting a point cloud as a Cartesian scatterplot—that is, they want to know what the horizontal (x) and vertical (y) axes of the plot mean. For a projection, this isn't easy because these axes don't straightforwardly map to data dimensions but, rather, to *combinations* of dimensions. Luckily, we can compute the contribution of each of the original m data dimensions to the spread of points along the projection's x and y axes. Next, we can visualize these contributions by standard bar charts (see Figure 2c): for each dimension, the x and y axis legends show a bar indicating how much that dimension is visible on the x and y axes. Long bars, thus, indicate dimensions that strongly contribute to the spread of points along the horizontal and vertical directions. Figure 2c shows how this works: the dataset contains 583 patient records, each having 10 dimensions describing patients' gender, age, and eight blood measurements. The projection shows two clusters placed aside each other.

How do we explain these? In the x axis legend, we see a tall orange bar, which tells us that this dimension (gender) is strongly responsible for the points' horizontal spread. If we color the points by their gender value, we see that, indeed, gender explains the clusters. Axis legends can also be used for 3D projections, as in Figure 2d, which shows a 3D projection of a 200,000-sample dataset with 10 dimensions coming from a simulation describing the formation of the early universe.[14] As we rotate the 3D projection, the bars in the axis legends change lengths and are sorted from longest to shortest, indicating the best-visible dimensions from a given viewpoint (dimensions 5 and 7, in our case). A third legend (Figure 2d, top right) shows which dimensions we *can't* see well in the projection from the current viewpoint. These dimensions vary strongly along the viewing direction, so we shouldn't use the current viewpoint to reason about them.

Biplot axes can also be inspected to get more detail. For example, we see that the projection's saddle shape is mainly caused by variable 7 and that the spike outlier is caused by a combination of dimensions 5 and 6. This interactive viewpoint manipulation of 3D projections effectively lets us create an infinite set of 2D scatterplot-like visualizations on-the-fly. Both biplot axes and axis legends explain a projection globally. If well-separated groups of points are visible, we can't directly tell which variables are responsible for their appearance without visually correlating the groups' positions with annotations, which can be tedious. Local explanations address this by explicitly splitting the projection into groups of points that admit a single (simple) explanation, depicting this explanation atop

the groups. Figure 2e shows this for a dataset containing 6,773 open source software projects, each having 11 quality metrics, along with their download count.[15] The projection, constructed with LAMP, shows a concave shape but no clearly separated clusters.

Let's consider next every projected point and several of its close neighbors—that is, a small circular patch of projected points. Because these points are close in the projection, they should also be similar in m dimensions. We can analyze these points to find which dimension is most likely responsible for their similarity. By doing this for all points in turn, we can rank all m dimensions by the number of points whose neighborhoods they explain. If we color code points by their best-explaining dimension, the projection naturally splits into several clusters. We can next add labels with the names of their explaining dimensions. Finally, we can tune the points' brightness to show how much of a point's similarity is explained by the single selected dimension. In Figure 2e, we see, for instance, that the lines of code metric (purple) explains two clusters of points—by interactive brushing, we can find that one contains small software projects and the other has large software projects. The bright-to-dark color gradient shows how it's increasingly hard to explain a point's similarity with its neighbors once we approach the cluster border, that is, the place where another dimension becomes key to explaining local similarity. Doing this visual partitioning of the projection into groups explained by dimensions would have been hard using global methods only, such as biplot axes or axis legends. Besides explaining groups in a projection via single dimensions, we can also use tag clouds to show the names of several dimensions.[16]

Interpreting Projections

As already explained, projections can be used as visual proxies of high-dimensional spaces that enable reasoning about a dataset's structure. For this to work, however, a projection should faithfully preserve those elements of the data structure that are important for the task at hand. As such, before using a projection, it's essential to check its quality.

The easiest way to do this is to compute the aggregated normalized stress. Low values of this stress tell us that the projection preserves distances well. However, if this single figure indicates low quality, we don't know what that precisely means or which observations are affected. More insight can be obtained by showing scatterplots of the original distances in m dimensions versus distances in the

projection. Figure 3a illustrates this for several datasets and projection techniques.[10] The ideal projection behavior is shown with red diagonal lines; figures in each scatterplot show the aggregated normalized stress, telling us that LAMP is generally better than the other two studied projections. Yet, we don't know what this means precisely. Looking at the scatterplots' deviations from the red diagonals, we get more insight in the nature of the errors: points under the diagonal tell us that original distances are subestimated in the projection, that is, that the projection compressed the data. Note that this is quite a typical phenomenon: projections have to embed points in a much lower-dimensional space, so crowding occurs very likely. For the *isolet* dataset, we see, for example, that small 2D distances can mean a wider range of high-dimensional distances than large 2D distances, so close points in a projection may or may not be that close in m dimensions. For the *viscontest* dataset, we see that LAMP has a constant spread around the diagonal, indicating a uniform error distribution for all distance ranges. In contrast, Glimmer shows a much worse error distribution.

While useful to reason about *distances*, such scatterplots don't tell us *where* in the projection we have errors. For this, we can use observation-centric error metrics.[17] The aggregate error shows the normalized stress, aggregated per point rather than for all points. Figure 3b shows this for a projection created with LAMP. As we see, the projection overall is of good quality, with the exception of four small hot spots. Figure 3c shows errors created by false neighbors—that is, points close in 2D but far in m dimensions, or zones where the projection compressed the high-dimensional space. We see here only three hot spots, meaning that the fourth one in Figure 3b wasn't caused by false neighbors. Figure 3d shows errors created by missing neighbors—that is, points close in m dimensions but far in 2D. The missing neighbors of the selected point of interest are connected by lines, which are bundled to simplify the image. The discrepancy between the 2D and original distances is also color coded on the points themselves. In this image, we see that the missing neighbors of the selected point are quite well localized on the other side of the projection. This typically happens when a closed surface in m dimensions is split by the projection to be embedded in 2D. Finally, Figure 3e shows for a selected group of points all the points that are closer in m dimensions to a point in the group than to any other point but closer to points outside that
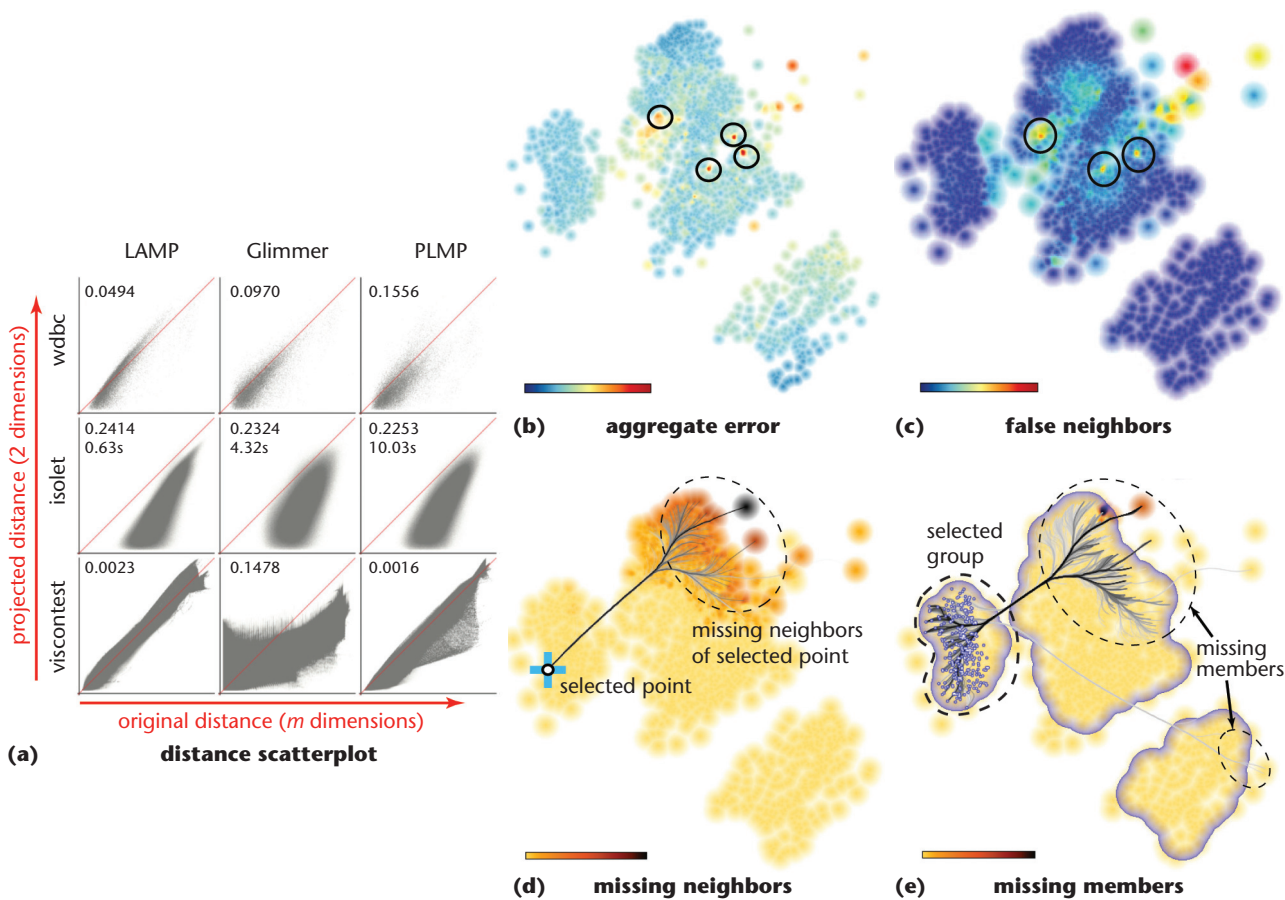
Figure 3. Projection visualized with (a) distance-centric methods and (b) through (e) observation-centric methods. The ideal projection behavior is shown with red diagonal lines. Figures in each scatterplot show the aggregated normalized stress, telling us that LAMP is generally better than the other two studied projections.

group in 2D. This lets us easily see if groups that appear in the projection are indeed complete or if they actually miss members.

Using Projections in Visual Analytics Workflows

So far, we've shown how we can construct projections, check their quality, and visually annotate them to explain the contained patterns. But how are projections used in complex visual analytics workflows? The most common way is to visually explore them while searching for groups, and when such groups appear, to use tools like the ones presented so far to explain them in terms of dimensions and dimension values.[2] This is often done in data mining and machine learning.

We illustrate this with a visual analytics workflow for building classifiers for medical diagnosis.[18] The advent of low-cost, high-accuracy imaging devices has enabled both doctors and the public to generate large collections of skin lesion images. Dermatologists want to automatically classify these into benign (moles) and potentially malignant (melanoma), so they can focus their precious time on analyzing the latter. For this, image classifiers can be used: each skin image is described in terms of several dimensions, or features, such as color histograms, edge densities and orientations, texture patterns, and pigmentation. Next, dermatologists manually label a training dataset of images as benign or malignant, using it to train a classifier so it becomes able to label new images. Other applications of machine learning include algorithm optimization, designing search engines, and predicting software quality.

Designing good classifiers is a long-standing problem in machine learning and is often referred to as the "black art" of classifier design.[19] The problem is multiple-fold: understanding discriminative features; understanding which observations are
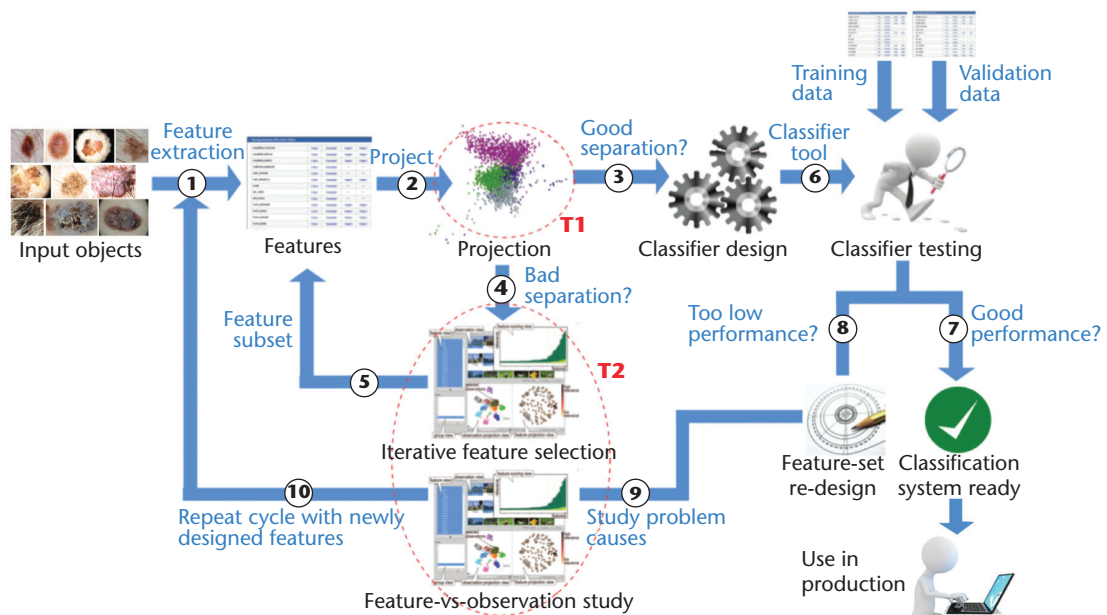
Figure 4. Using projections to build and refine classifiers in supervised machine learning.

hard to classify and why; and selecting and designing features to improve classification accuracy. Projections can help all these tasks, via the workflow in Figure 4. Given a set of input observations, we first extract features that are typically known to capture their essence (step 1). This yields a high-dimensional data table with observations as rows and features as columns. We also construct a small training set by manual labeling. Next, we want to determine how easy the classification problem ahead of us will be. For this, we project the training set and color observations by class labels (step 2). If the classes we wish to recognize are badly separated, it makes little sense to spend energy on designing and testing a classifier, since we seem to have a poor feature choice (step 4) We can then interactively select the desired class groups in the projection and see which features discriminate them best,[18] repeating the cycle with a different feature subset (step 5). If, however, classes are well separated in the projection (step 3), our features discriminate them well, so the classification task isn't too hard. We then proceed to design, train, and test the classifier (step 6). If the classifier yields a good performance, we're done: we have a production-ready system (step 7). If not, we can again use projections to see which are the badly classified observations (step 8), which features are responsible for this (step 9), and engineer new features that separate these better (step 10). In this workflow, projections serve two key tasks: *predicting* the ease of building a good classifier ahead of the actual construction (T1), thereby saving us from designing a classifier with unsuitable features, and *showing* which observations are misclassified and their feature values (T2), thereby helping us design better features in a targeted way.

Projections are the new emerging instrument for the visual exploration of large high-dimensional datasets. Complemented by suitable visual explanations, they're intuitive, easy to use, visually compact, and easy to learn for users familiar with scatterplots. Recent technical developments allow their automatic computation from large datasets in seconds, helping users avoid complex parameter settings or needing to understand the underlying technicalities. As such, they're part of the visual data scientist's kit of indispensable tools.

But as projections become increasingly more useful and usable, several new challenges have emerged. Users require new ways to manipulate a projection to improve its quality in specific areas, to obtain the best-tuned results for their datasets and problems. Developers require consolidated implementations of projections that would let them integrate them in commercial-grade applications such as Tableau. And last but not least, users and scientists require more examples of workflows showing how projections can be used in visual analytics sensemaking to solve problems in increasingly diverse application areas. ■

References

1. S. Liu et al., "Visualizing High-Dimensional Data: Advances in the Past Decade," *Proc. EuroVis–STARs*, 2015, pp. 127–147.
2. C. Sorzano, J. Vargas, and A. Pascual-Montano, "A Survey of Dimensionality Reduction Techniques," 2014; http://arxiv.org/pdf/1403.2877.
3. W.S. Torgeson, "Multidimensional Scaling of Similarity," *Psychometrika*, vol. 30, no. 4, 1965, pp. 379–393.
4. J.B. Tenenbaum, V. de Silva, and J.C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science*, vol. 290, no. 5500, 2000, pp. 2319–2323.
5. U. Brandes and C. Pich, "Eigensolver Methods for Progressive Multidimensional Scaling of Large Data," *Proc. Graph Drawing*, Springer, 2007, pp. 42–53.
6. C. Faloutsos and K.-I. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," *SIGMOD Record*, vol. 24, no. 2, 1995, pp. 163–174.
7. K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 1990.
8. I.T. Jolliffe, *Principal Component Analysis, ser. Statistics*, Springer, 2002, p. 487.
9. F.V. Paulovich, C.T. Silva, and L.G. Nonato, "Two-Phase Mapping for Projecting Massive Data Sets," *IEEE Trans. Visual Computer Graphics*, vol. 16, no. 6, 2010, pp. 1281–1290.
10. P. Joia et al., "Local Affine Multidimensional Projection," *IEEE Trans. Visual Computer Graphics*, vol. 17, no. 12, 2011, pp. 2563–2571.
11. M. Greenacre, *Correspondence Analysis in Practice*, 2nd ed., CRC Press, 2007.
12. P. Pagliosa et al., "Projection Inspector: Assessment and Synthesis of Multidimensional Projections," *Neurocomputing*, vol. 150, 2015, pp. 599–610.
13. M. Greenacre, *Biplots in Practice*, CRC Press, 2007.
14. D. Coimbra et al., "Explaining Three-Dimensional Dimensionality Reduction Plots," *Information Visualization*, vol. 15, no. 2, 2015, pp. 154–172.
15. R. da Silva et al., "Attribute-Based Visual Explanation of Multidimensional Projections," *Proc. EuroVA*, 2015, pp. 134–139.
16. F.V. Paulovich et al., "Semantic Wordification of Document Collections," *Computer Graphics Forum*, vol. 31, no. 3, 2012, pp. 1145–1153.
17. R.M. Martins et al., "Visual Analysis of Dimensionality Reduction Quality for Parameterized Projections," *Computers & Graphics*, vol. 41, 2014, pp. 26–42.
18. P.E. Rauber et al., "Interactive Image Feature Selection Aided by Dimensionality Reduction," *Proc. EuroVA*, 2015, pp. 54–61.
19. P. Domingos, "A Few Useful Things to Know about Machine Learning," *Comm. ACM*, vol. 10, no. 55, 2012, pp. 78–87.

Renato R.O. da Silva is a PhD student at the University of Sao Paulo, Brazil. His research interests include multidimensional projections, information visualization, and high-dimensional data analytics. Contact him at rros@icmc.usp.br.

Paulo E. Rauber is a PhD student at the University of Groningen, the Netherlands. His research interests include multidimensional projections, supervised classifier design, and visual analytics. Contact him at p.e.rauber@rug.nl.

Alexandru C. Telea is a full professor at the University of Groningen, the Netherlands. His research interests include multiscale visual analytics, graph visualization, and 3D shape processing. Telea received a PhD in computer science (data visualization) from the Eindhoven University of Technology, the Netherlands. Contact him at a.c.telea@rug.nl.