Submitted to the Annals of Applied Statistics

VISUALIZING STATISTICAL MODELS: REMOVING THE BLINDFOLD

By Hadley Wickham, Dianne Cook and Heike Hofmann

Rice University
Iowa State University
Iowa State University

Visualization can help in model building, diagnosis, and in developing an understanding about how a model summarizes data. This paper proposes three strategies for visualizing statistical models: (1) display the model in the data space, (2) look at all members of a collection, and (3) explore the process of model fitting, not just the end result. Each strategy is accompanied by examples, including MANOVA, classification algorithms, hierarchical clustering, ensembles of linear models, projection pursuit, self organizing maps and neural networks.

1. Introduction. Visual methods for high-dimensional data are well developed and understood. Our toolbox contains static graphics, such as scatterplot matrices, parallel coordinate plots and glyphs, interactive tools like brushing and linking, and dynamic methods, such as tours. We can also use these tools to visualize our models, and when we have done so, we have often been surprised: fitted models can be quite different from what we expect!

Visual model descriptions are a particularly important adjunct to numerical summaries because they help answer different types of questions:

- What does the model look like? How does the model change when its parameters change? How do the parameters change when the data is changed?
- How well does the model fit the data? How does the shape of the model compare to the shape of the data? Is the model fitting uniformly good, or good in some regions but poor in other regions? Where might the fit be improved?

If we can't easily ask and answer these questions, our ability to understand and criticize models is constrained, and our view of the underlying phenomenon flawed. This may not matter if we only care about accurate predictions, but better understanding of the underlying science will usually enhance generalization.

As well as facilitating good science, model visualization (model-vis) can also be used to enhance teaching and research. Pedagogically, it gives students another way of understanding newly encountered methods. Using modern interactive methods students can become experimental scientists studying a model; systematically varying the input and observing the model output. Model-vis can also be useful in the development of new theory; seeing where a model or a class of models performs poorly may suggest avenues for new research.

Converting data-vis methods to model-vis methods is not always straightforward. This paper summarizes our experiences, and provides three overarching strategies:

- Display the model in data space (m-in-ds), as well as the data in the model space. It is common to show the data in the model space, for example, predicted vs observed plots for regression, linear discriminant plots, and principal components. By displaying the model in

AMS 2000 subject classifications: Primary 62-09; secondary 62A01
Keywords and phrases: model visualization, exploratory model analysis

the high-d data space, rather than low-d summaries of the data produced by the model, we expect to better understand the fit.
- Look at all members of a collection, not just one. Exploring multiple models will usually give more insight into the data and the relative merits of different models. This is analogous to the way that studying many local optima may give more complete insight than a single global optimum, or the way that multiple summary statistics is more informative than one alone.
- Explore the process of fitting, not just the end result. Understanding how the algorithm works allows us to better understand how the data contributes to the final model.

The importance of a strong connection between statistical models and visual exploration has been recognized and incorporated into statistical software: Lisp-Stat (**?**), `rggobi` and its ancestors of (**?**), Datadesk (**?**), ViSta (**?**), ARC (**?**), Mondrian (**?**), iplots (**?**), Klimt (**?**) and Gauguin (**?**). This paper builds upon these ideas to shape a consistent strategy for the design of model visualizations.

The paper is structured into sections corresponding to these three principles. Each section contains examples of the principles for particular models. Section 3 describes the technique of displaying the model-in-data space, illustrated with examples from MANOVA, classification models and hierarchical clustering (**?**), and contrasts m-in-ds against the more common approach of displaying the data-in-model space. Section 4 explains the benefit of visualizing collections of multiple models, rather than just a single "best" model, illustrated with an ensemble of linear models. Section 5 shows how we can visualize an iterative model-fitting process, using examples from projection pursuit (**??**) and self-organizing maps (**?**).

2. Background. The following sections introduce our terminology for models, and describes the particular visual methods that we will use repeatedly in this paper.

2.1. *The different pieces of a model.* "Model" is an overloaded term. In this paper we use three terms to refer to three different levels of specificity: model family, model form and fitted model. These are defined as follows:

- The **model family** describes, at the broadest possible level, the connection between the variables of interest. For example, the linear model predicts a single continuous response with a linear combination of predictor variables, and assumes normally distributed error: $Y|X \sim \text{Normal}(\mu = AX, \sigma)$. The model family is almost always specified by the statistical problem (i.e., you can't use a test to determine whether you should use a linear model or model-based clustering).
- The **model form** specifies exactly how the variables of interest are connected within the framework of the model family. For example, the model form of a linear model specifies which variable is the response, and which variables are the predictors, e.g., $height|age, sex \sim \text{Normal}(\mu = b + a_1 \cdot age + a_2 \cdot sex, \sigma)$. The model form might be specified a priori by the statistician, or chosen by a model-selection procedure.
- The **fitted model** is a concrete instance of the model form where all parameters have been estimated from data, and the model can be used to generate predictions. For example, $height|age, sex \sim \text{Normal}(\mu = 10 + 5 \cdot age + 3 \cdot sex, \sigma = 4)$. Typically, the fitted model optimizes some criterion for the given data, e.g., the linear model minimizes the total squared deviation between responses and predictions.

Bayesian models can be described in a similar manner, although there we tend to focus on the distributions of the parameters given the data, rather than a single estimated value.

Not all techniques used by statisticians have these explicit probabilistic models, but the basic breakdown remains the same. For example, neural networks are a model family, the model form specifies the variables and number of nodes, and the fitted model will have estimates for the parameters of each node. Neural networks are a good example of where going from the model form to the fitted model is difficult. We must use numerical methods, and convergence to a global optimum is not guaranteed. This means that to find the "best" fitted model (i.e., the model with the highest criterion value) we need multiple random starts.

Knowing the model family and form does not guarantee we know what the fitted model looks like. For example, the parameters of the linear model can be interpreted as the change in the response when the predictor is changed by one unit, if all other variables are held constant. This seems easy to interpret, but most models have interdependencies between the predictors which means variables do not change independently, as with interactions or polynomial terms. For more complex models, there may never be any direct interpretation of any of the model parameters. For example, in generalized additive models (**?**), knowing the form of the model and the parameter values does not give us insight into what the model says about the data.

2.2. *Tools for visualizing high-d data and models.* To visualize data and models in many dimensions, we need good tools. While it is possible to investigate high-d objects with only static graphics, it is much easier to do so interactively. We will use two basic tools extensively: the grand tour, to look at many projections of the data; and linked brushing, to look at many sections of the data. These tools are described briefly below, and in more detail in **?**.

The grand tour. When we have a 3D object, how do we figure out what shape it is? We pick it up and look at it from multiple sides, joining a sequence of 2d views into a 3D model. The grand tour generalizes this idea to more than 2D, generating a sequence of 2d projections of an p-d object. It chooses new projections randomly and smoothly rotates between them. The new projections are selected at random so that if we watch the tour long enough we'll see every possible view, but we'll never get stuck in the same uninteresting region for long. Among other things, the grand tour can be used to identify multivariate outliers and clusters in the data, as the snapshots in Figure 1 illustrate. The plots in Figure 1 show projections from a grand tour of the wine data from **?**. This dataset contains information on 178 wines made from three varieties of grapes, with 13 variables describing physical composition (including chemical constituents and wine color). Just four variables are entered into the grand tour (alcohol, flavonoids, phenols, dilution) and the points are colored by the grape variety. In these two projections the groups are reasonably well-separated suggesting that they are distinguishable based on the variables in the data set. This data is used several times during the paper for illustrating different ideas. A variant of the grand tour is the guided tour, where new projections are selected according to an index of interestingness, and is described in Section 5.1.

Linked brushing. Brushing is an interactive conditioning tool, analogous to trellis plots, but much more flexible. Brushing links observations across multiple views with consistent styles. We use a brush to colour observations in one plot and see the same observations with the same key change colour in other views. Figure 2 shows one use of the brush, investigating outlier diagnostics in regression modeling of the New Haven Housing data from the `YaleToolkit` package in R (**?**). The data describe the 2006 real estate market in New Haven, CT. There are 18221 observations and multiple characteristics of the houses are recorded. The left plot shows outlier diagnostics (Cooks D, residuals) for many models, and the right plot shows the data. There are different number of points in each plot, but points are linked by house id's. An observation deemed influential in one model

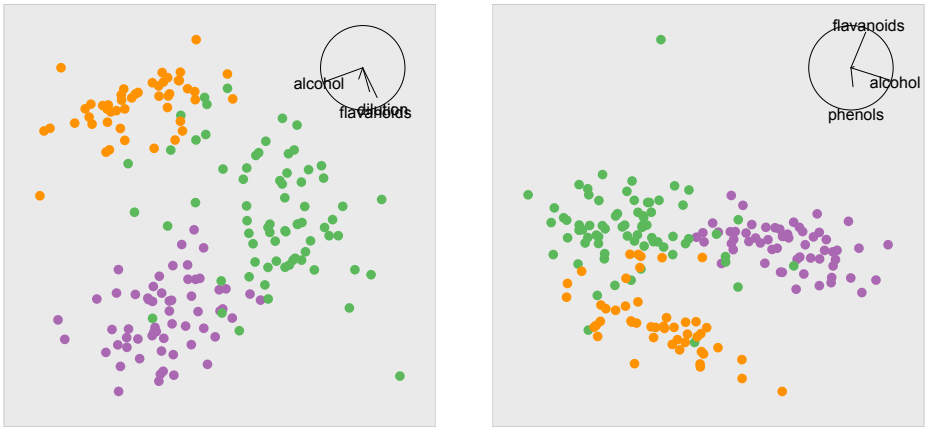

Fig 1: Two interesting 2D projections from the grand tour of four variables of the wine data, illustrating that the three groups are fairly distinct, and the existence of a single outlying point belonging to the green group. A movie of the data touring is available at http://vimeo.com/823263.

has been brushed (left plot), all other influence values for this case in other models are highlighted in the same plot, and its corresponding value in a plot of the raw data is also highlighted (right plot). This case corresponds to a house that is very big but relatively cheap. It has high Cooks D and an extreme residual in many, but not all, models. More information on this example can be found in Section 4.1.

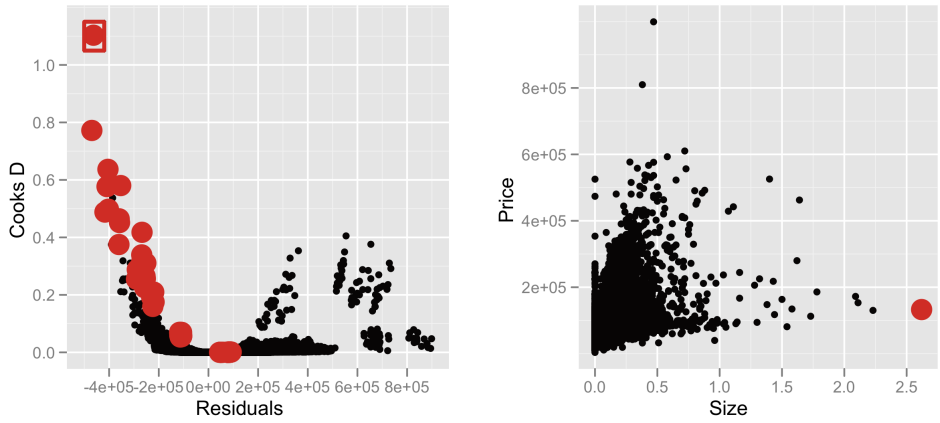

Fig 2: Selecting a single observation (red rectangle) highlights the values of that observation for all models (left) and in the raw data (right).

3. Display the model in data-space. A major theme of this paper is the importance of visualizing the model in the context of the data, displaying the model in the high-dimensional data space, or for short, showing the m-in-ds. We take the data space to be the region over which we can reliably make inferences, usually a hypercube containing the data, unless the variables are highly dependent, in which case the data space might be a small subspace of the full cube. Displaying

the m-in-ds is important because it allows us see how well (or how poorly) the model responds to the data. The converse of visualizing the model in data space is visualizing the data in the model space (d-in-ms), where the data is displayed in some low-dimensional space generated by the model. Both methods have their uses, but m-in-ds displays are particularly helpful for understanding and critiquing the model.

To compare these approaches, think of a linear model. One of the most common diagnostics is a plot of fitted values versus residuals. This is a d-in-ms display because we are plotting summary statistics from the model in a low-d view. An m-in-ds approach would be to generate the full regression surface, and visualize it in the context of the data; this is hard, but often revealing. For an m-in-ds approach, we can still use the residuals, but we would want to see them in the context of the entire predictor space. For categorical predictors, another m-in-ds approach would be to use population marginal means (**?**) to summarize the categorical coefficients in such a way that they are interpretable in the data space.

Representing the model as data is challenging and requires some creativity and imagination. You can draw inspiration from existing d-in-ms plots, but otherwise coming up with a good representation requires some thinking and experimentation. First, it's worth considering what a model embedded in the data space will look like. We can say a few things about the form of models in general because almost all models share two important characteristics: they summarize the data and interpolate the data space. This means that the model will have many fewer dimensions than the data, and it will take values over the entire data space, even where there are no data. This implies that many models are manifolds, or high-d surfaces.

For predictive models with a continuous response, this manifold is the prediction surface, $z = f(a, b, c, ...)$. For complex models, it's often difficult to describe this surface parametrically, and we have few tools to display high-d surfaces, so we suggest using an approximation. The simplest method is to generate a dense sample of points lying on the surface. One way to do this is to sample from the predictor space and compute a prediction for each sample. Borrowing from the terminology of spatial statistics, we can sample the predictor space in a stratified, random or non-aligned manner. Non-aligned sampling tends to work best: stratified sampling creates a grid that can produce distracting visual artifacts, and uniform sampling generally needs many more points to create the illusion of a continuous surface.

If we want to display more information about the predictive distribution than its expected value, what can we do? One approach is to also display other quantities from the distribution. For example, with a linear model, we might display the mean and the 5% and 95% quantiles. This is straightforward when the predictor only enters the distribution through a single parameter (e.g., the mean), but maybe complex if it enters through multiple parameters. In that case, an alternative is to draw random samples from the predictive distribution (**?**).

For models that incorporate connections between components it can be useful to display these connections explicitly with lines. This technique is used for hierarchical clustering in Section 3.3 and for self organizing maps in Section 5.2. This is useful for any model with a neighborhood structure.

The following three case studies highlight some of these different techniques: for MANOVA, Section 3.1, we display a quantile from the predictive distribution; for classification algorithms, Section 3.2, we sample from the data space and display prediction regions; and for hierarchical clustering, Section 3.3, we draw inspiration from 2D plots and make explicit the implicit connections.

3.1. *Case study: MANOVA.* MANOVA is a multivariate extension of ANOVA where there are multiple continuous responses instead of just one. Correspondingly, the error distribution of MANOVA is the multivariate normal, with mean 0 and variance-covariance matrix Σ. Because MANOVA looks

at response variables simultaneously it can identify differences that individual ANOVAs cannot, as illustrated by Figure 3.

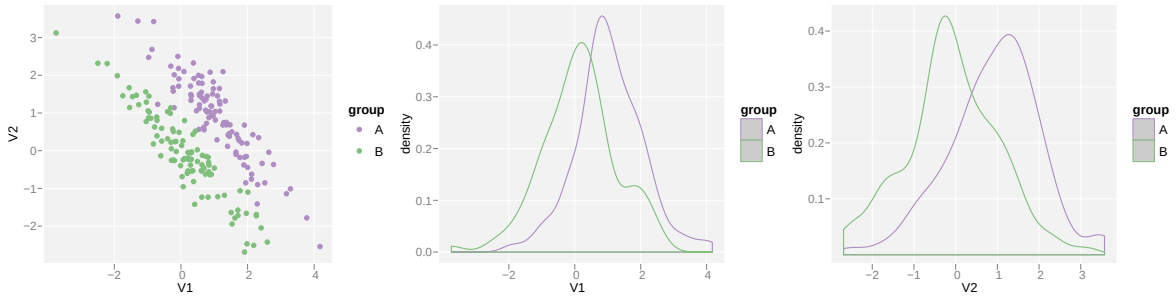

Fig 3: Example where MANOVA would detect a difference between the groups, but two ANOVAs would not. Groups are distinct in 2D (left), but overlap on both margins (right).

This case study explores adding MANOVA model information to the raw data. We will investigate a simple MANOVA with a single categorical explanatory variable, and discuss how we might extend the ideas to the general multivariate linear model. In this simple case, the model is that each group is multivariate normal with the same variance-covariance matrix. To summarize this model we'll display a 95% confidence region around the mean.

Figure 4 shows the result of this approach applied to wine. As the variables are not directly comparable, they have been standardized to range $[0, 1]$.

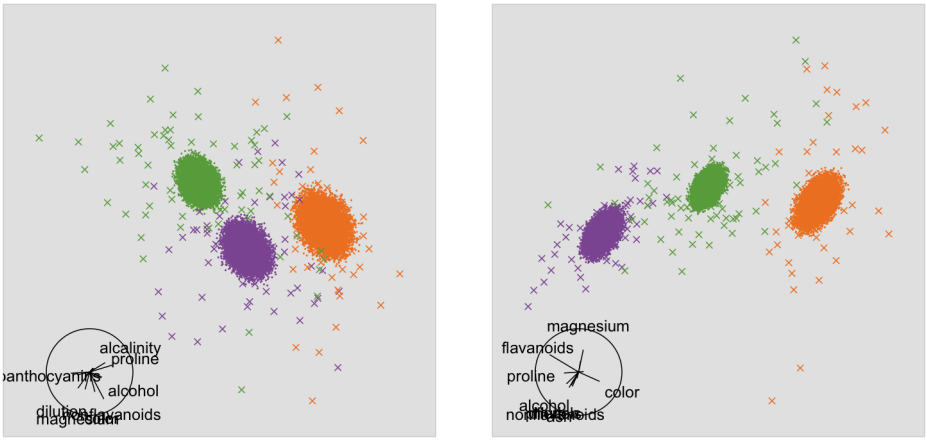

Fig 4: Two projections of the wine data with 95% confidence regions around the group means. Large points are data, small points are a sample on the surface of the confidence region. While the confidence ellipsoids appear close in a few projections (left), in most views we see that the means are distant (right).

Extending this idea to deal with multivariate linear models is straightforward, although visualizing the resulting surfaces will be difficult. If the model contains continuous variables, then we will no longer have point predictors of the mean, but instead, continuous functions each surrounded by a confidence region. We can generate these confidence regions by randomly sampling the predictor space and then generating the confidence region for each sampled point as for MANOVA. Visualization is complicated when we have multiple response and predictor variables as we want to keep

them separate, e.g., a linear combination of a predictor and a response variable probably doesn't make much sense. A potentially useful tool here is the correlation tour (**?**), a version of the tour which uses separate sets of variables for the x and y axes.

3.2. *Case study: Classification models.* A classifier is a model with a categorical response. Typically, the classifier is treated as a black box and the focus is on finding classifiers with high predictive accuracy. For many problems the ability to predict new observations accurately is sufficient, but it is interesting to learn about how the algorithms operate by looking at the boundaries between groups. Understanding the boundaries is important for the underlying real problem because it tells us where the groups differ. Simpler, but equally accurate, classifiers may be built with knowledge gained from visualizing the boundaries, and problems of over-fitting may be intercepted.

Much of what we know of how a classifier works is based on the knowledge of the algorithm. Each family of classifiers partitions the data space in a different way: LDA (**?**) divides up the data space with hyperplanes, while trees (**?**) recursively split the space into boxes. Most classifiers produce connected areas for each group, but there are exceptions, e.g., k-nearest neighbors. To visualize the model in the data space, we can either display the complete prediction regions or just their boundaries. These two approaches are shown in Figure 5.

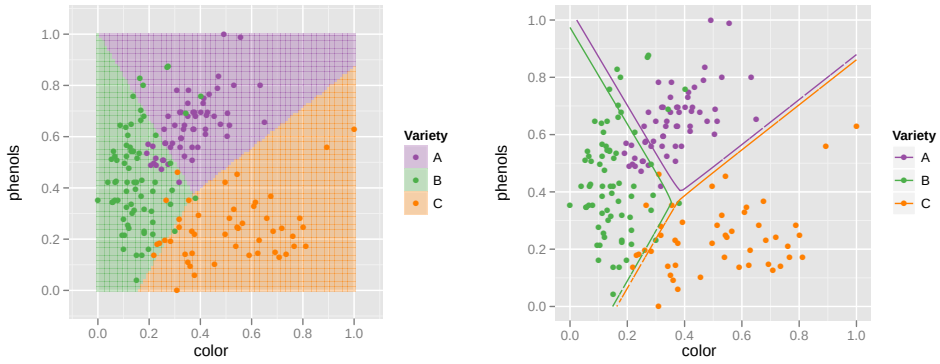

Fig 5: A 2D LDA classifier on the wine dataset. (Left) entire classification region shown and (right) only the boundary between regions. Data points are shown as large circles.

Generating the complete prediction region is straightforward: sample many points from data space and classify each one. Generating only the boundary is more difficult. We may be able to use a closed form expression if available, but otherwise we can start with the full region and then remove non-boundary points. To determine if a point is near a boundary, we look at the predicted class probabilities. If the highest and second highest probabilities are nearly equal, then the classifier is uncertain about the class, so the point must be near a boundary. We call this difference in probabilities the *advantage*, and to find the boundary we discard points with values above a tolerance. The thickness of the resulting boundary is controlled by this cut off, and it should be small enough to give a sharp boundary, but large enough that the set of boundary points is dense. This is a tricky trade-off. It may be possible to use adaptive sampling, sampling more densely in regions closer to the boundary, to do better.

If the classification function does not generate posterior probabilities, a k-nearest neighbors approach on the grid of predictions can be used. If the neighbors of a point are all the same class, then the point is not on a boundary. This method can be applied to any classification function, but is slow, $O(n^2)$, because it computes all pairwise distances to find the nearest neighbors. In practice,

this imposes a limit of around 20,000 points.

Figure 6 illustrates the results of a support vector machine (?) with radial kernel fitted to three variables from the wine data. We see the purple region is contained almost completely inside the green region, except where it abuts the orange region. The boundaries and regions are straightforward to understand because our brains are adept at 3D modeling. The video at http://www.vimeo.com/821284 explores the boundaries in another way, by brushing along the advantage variable. This shows that the boundaries are equally sharp between the different groups.

As we move beyond 3D it gets harder to see what is going on, but if we persevere we can find informative views. Figure 7 shows three informative views of a polynomial kernel with five variables. It looks like the boundary is relatively flat (linear) between the orange and purple groups, but quadratic around the green group. This is easier to see interactively, as at http://vimeo.com/823271.

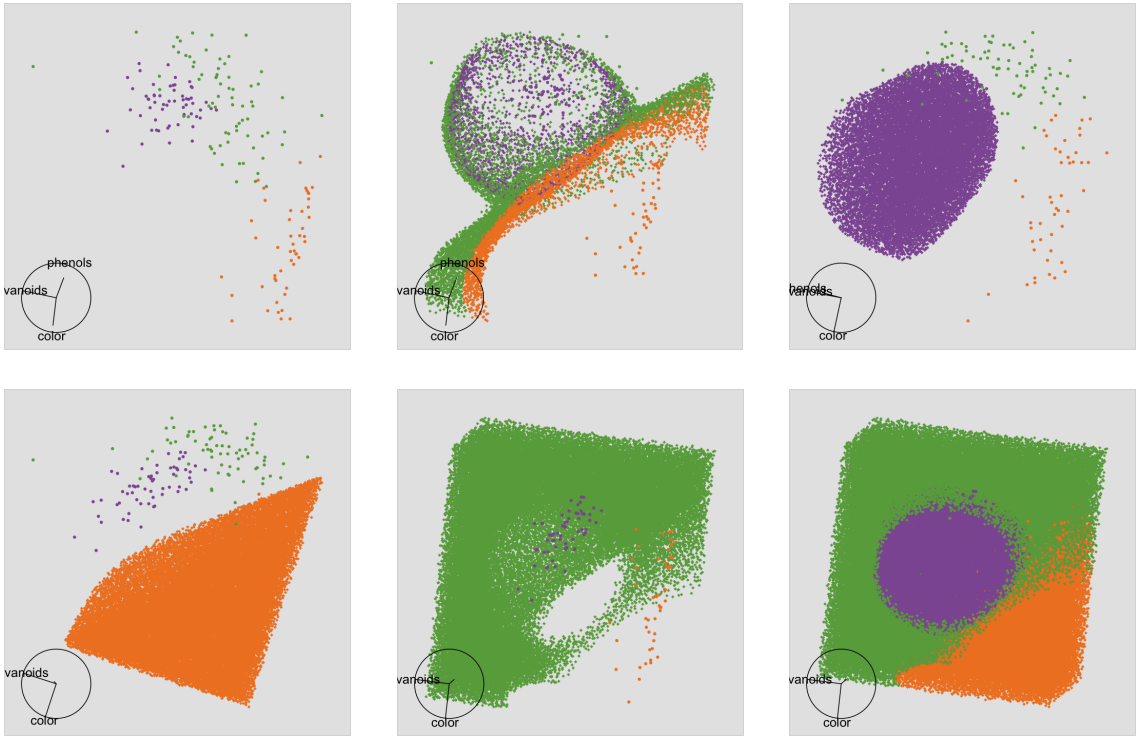

Fig 6: Views of a 3D radial SVM classifier on the wine data using the variables color, phenols, and flavonoids. The same projection is shown with selected subsets of all of the features. (Top left) Data only, (Top middle) Boundary points between the three groups, revealing the highly non-linear decision regions, (Top right) Prediction region for the purple group shown illustrating the way that the classifier wraps the boundary tightly around this group, (Bottom left) Prediction region for the orange group carves out a slice of the cube, (Bottom middle) Prediction region for the green group is basically the rest of the cube, (Bottom right) all three prediction regions colored showing how the cube is carved into the three prediction regions. Animated version available at http://vimeo.com/821284.

These ideas are implemented in the R package `classifly` (?), which can visualize classifiers generated by LDA and QDA (?), SVM (?), neural networks (?), trees (?), random forests (?) and logistic regression; and it can easily to be extended to deal with other classifiers. More specific visualizations may also be useful for each of these families of models: ?, support vector machines;

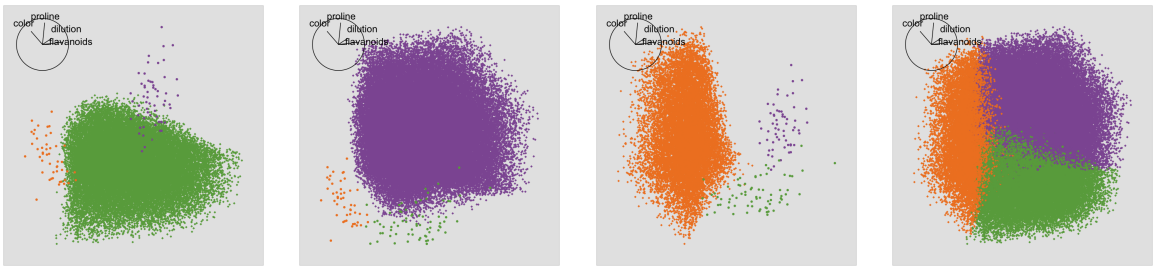

Fig 7: Views of a 5d SVM with polynomial kernel on the wine data using the variables color, phenols, flavonoids, proline and dilution. It's possible to see that the boundaries are largely linear, with a "bubble" of green pushing into the orange and purple. A video presentation of this tour is available at http://vimeo.com/823271.

and KLIMT provides many tools for visualizing trees (**???**).

3.3. *Case study: Hierarchical clustering.* Agglomerative hierarchical clustering methods (**?**) build up clusters point by point, iteratively joining the two closest points or clusters. The most common visualization of a hierarchical clustering is a dendrogram, a d-in-ms display. There are also a number of other methods of this type: icicles (**?**), silhouettes (**?**) and clustergrams (**?**). Figure 8 shows two dendrograms produced by clustering the wine dataset with Wards linkage and single linkage. These dendrograms aren't very informative. For the Wards clustering we can see that there are three major clusters, and for single linkage the clusters seem to grow mainly by adding a single point to an existing cluster. We can't see what the clusters have in common, or what variables are important for the clustering.

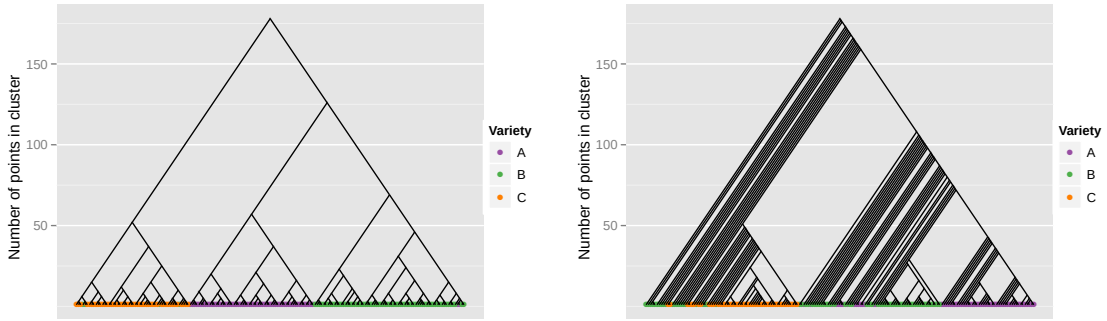

Fig 8: Dendrograms from a hierarchical clustering performed on the wine dataset. (Left) Wards linkage and (right) single linkage. Points colored by wine variety. Wards linkage finds three clusters of roughly equal size, which correspond fairly closely to three varieties of wine. Single linkage creates many clusters by adding a single point, producing the dark diagonal stripes.

To do better we need to display the model in data space. **?** draws inspiration from the dendrogram and suggests an interesting idea: connect the observations in the data space according to their hierarchy in the clustering. For intermediate nodes in the hierarchy, representing clusters containing more than one point, we supplement the data with extra points located at the cluster centroid. An extension would be to connect clusters in the same way as inter-cluster distances are calculated,

i.e., connect closest points for single linkage, most distant for complete, and centroids for Wards. Another approach is described in **?** which uses MDS for dimension reduction, and overlays the minimum spanning tree.

Figure 9 displays some views of the wine dataset supplemented with a hierarchical clustering with Wards linkage. It is easy to see the three groups and the hierarchical links in the tour, as shown at `http://www.vimeo.com/768329`.

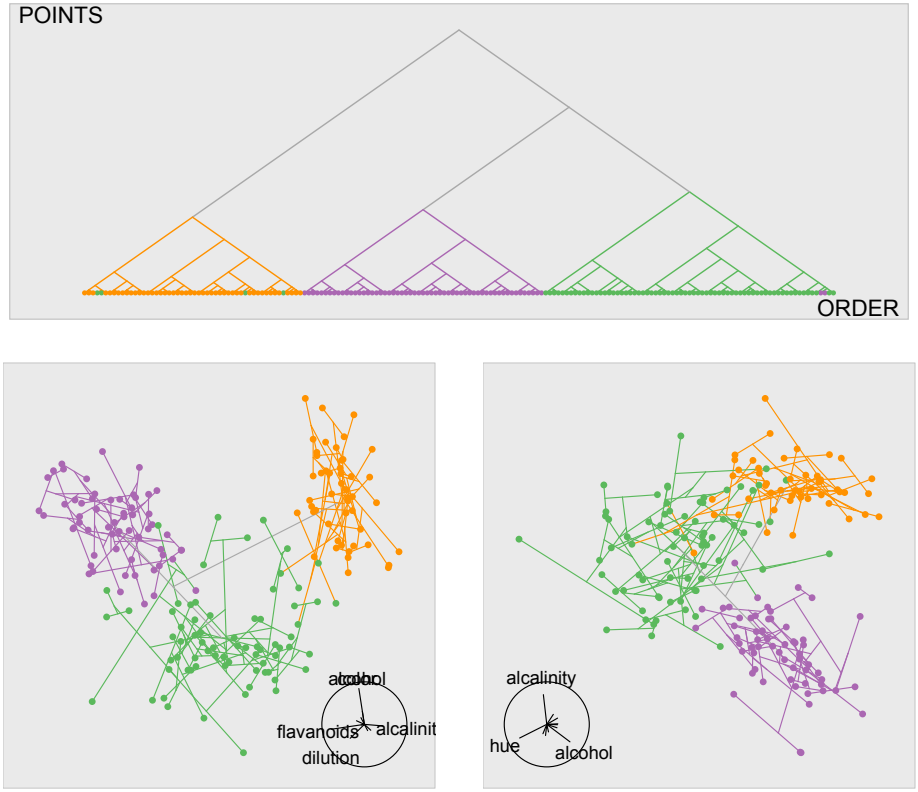

Fig 9: (Top) A dendrogram with points colored by variety and edges colored by majority membership. (Bottom) Two informative projections of the data space show that the varieties are arranged in a rough U shape, with a little overlap, and some of the outlying orange points have been clustered with the green points.

An implementation of these ideas is available in the `clusterfly` package (**?**).

4. Collections are more informative than singletons. So far we have looked at visualizing a single model in the data space, but collections of models arise in many ways, and in most cases most of the models are ignored and we only look at the best model. This is perilous as we may miss alternative models that explain the data almost as well as the best model, and suggest substantially different explanations for the phenomenon of interest. A few of the ways that collections are produced include:

- From exploring the space of all possible models. For a given model family, we can generate all possible model forms, e.g., for a linear model, we can generate all possible linear models with main effects. This will often produce too many models to possibly fit or compare, so typically

we will use some model selection algorithm to explore the space of "best" models. Typically, only the best, or maybe the two or three best models are examined.

- During the the process of data analysis. When analyzing a data set, we may create and discard many models in an attempt to reveal the salient features of the data. It may be useful see all of these models simultaneously.
- While trying to find a global optima. When model fitting is not guaranteed to converge to a global optimum, we may have a collection generated from multiple random starts. This is useful for multidimensional scaling, neural networks, k-means clustering, and self organizing maps. Typically, only the model with the highest criterion value is examined.
- By varying model settings. If a model has some tuning parameters, we can systematically alter them and observe the result, e.g., the penalty parameter in lasso, or the degrees of freedom in a smoothing term. Often, cross-validation is used to select a single optimal model.
- By fitting the same model to different datasets. These datasets might be groups within a larger dataset (perhaps a prequel to fitting a mixed effects model), or might have been generated by leaving-one-out, cross validation, bootstrapping, permuting, simulating, as part of a sensitivity analysis or by adding random noise to the response (**?**). Typically, these multiple models are collapsed into a small set of summary statistics.
- As an intrinsic part of the model. Certain model families use ensembles of simple sub-models: neural networks (ensembles of logistic models), random forests (ensembles of trees), bagging, and boosting. Typically, we just look at the overall model, and ignore how each part contributes to the whole.
- By treating each iteration as a model. If we are interested in exploring the process of model fitting, and the algorithm is iterative, we might treat each iteration as a separate model. This gives a unique ordering to the models and is described in depth in Section 5. Typically, we only look at the last iteration.

What can we do if we want to visualize such a collection? One approach is to visualize all models simultaneously. This is the strategy of trace plots (**?**), which show hundreds of tree models simultaneously. This makes it possible to see the space of models, showing both the common and the unusual. However, in most cases we will not be able to view more than a couple of models simultaneously because they will overlap and obscure one another.

To work around this problem, we need to be able to fluidly explore smaller sets of interesting models. We can do this by calculating descriptive statistics on multiple levels, then using linked brushing to connect these statistics to one another and to displays of the model. These ideas grow out of exploratory model analysis (**??**) and draw on fundamental ideas of descriptive statistics (**??**). Here we focus on displays of model space, with the assumption that these displays will be linked to additional m-in-ds displays to help us tune our descriptive statistics to match the model features of interest.

The first level of descriptive statistics summarizes model quality: a trade-off between complexity and fit. Model complexity is often summarized by degrees of freedom, but there may be more informative measures for specific model families, e.g., the number of hidden nodes in a neural network. The fit can be summarized by the likelihood at the parameter estimates of models fit using the ML algorithm, or by some measure of predictive ability. These statistics help us explore overall model quality and the tradeoff between quality and complexity, but do not give any information about how the models differ, how the fit of a given observation varies between models, or what exactly the parameter estimates are.

To explore these aspects of the model, we need descriptive statistics at other levels. A good example of a statistical procedure with built in summary statistics at additional levels is the random forest (**?**), an ensemble method which uses many simple trees fit to random subsets of the variables and observations. As well as the model-level summaries described above, random forests provide tree-level, variable-level and observation-level descriptive statistics:

- Tree-level. Each tree in the ensemble has its own test and training data sets and so can compute an unbiased estimate of classification error. Inspecting these errors allows us to find trees that perform particularly well or poorly. Looking at many good trees in the data space, allows us to see commonalities (suggesting important variables) and differences (suggesting correlations between variables).
- Variable-level. Each variable is ranked by the drop in model performance when that variable is randomly permuted. This is a summary of a tree-variable statistic, which computes variable importance for each tree.
- Observation-level. For each observation we have the distribution of predictions across all trees. This can show which observations are easy (or hard) to classify, and which classes are most often confused.

For other models, we will need to develop and calculate our own descriptive statistics. Often there will be a large existing literature which can be probed for ideas. Another approach is to calculate a familiar summary statistic over an unfamiliar population. For example, we could calculate an observation-level summary by averaging the observation's residual over all the models. If models in the collection have common parameters, we might create parameter-level summaries of the distribution of the estimates over the models. The important thing is to generate descriptive statistics at multiple levels, in order to gain maximum insight into the models.

Once we have computed the summary statistics, we need to explore them. Static plots are helpful, but make it difficult to link between summaries at different levels. For example, we might want to see how the model-estimate summaries differ between the best and second best models. Linked brushing is particularly useful here. We can have one plot of model-level statistics and one of model-estimate-level statistics and use brushing to link between them. This idea is described in more detail in the case study.

The RAFT tool (**?**) for visualizing random forests provides a good set of static graphics, but provides no way to link between the plots, and no m-in-ds visualizations. This is a problem: if we found a single tree that did a particularly good job, we would like to see how it divides up the data space. Without linked graphics it is difficult to see exactly what is going on within a random forest, and to see how they compare to other models. The following case study shows how summary statistics and linked graphics can be used to gain insight into ensembles of linear models.

4.1. *Case study: Linear models.* In this case study we'll explore a collection of linear models containing a large subset of all possible main effects models. We will assume we have m models describing a data set with n observations and p variables. If all possible main effects models are fit, there will be $2^p - 1$ models in the collection. We will explore summary statistics on five levels:

- Model level: model fit statistics. m observations.
- Model-estimate level: coefficient estimates on various scales. $m \times p$ observations.
- Estimate level: summary of estimates over the models. p observations.
- Model-observation level: residuals and influence measures. $m \times n$ observations.
- Observation level: the original data, plus summaries of residual behavior. n observations.

The relationship between these different levels is shown schematically in Figure 10.

In this case study, we will use a data set on 2006 house prices in New Haven, Connecticut (**?**). We are interested in predicting price based on characteristics of the house. There are 18221 observations and 6 predictor variables, giving a collection of 63 ($2^6 - 1$) models. The predictors include size of the living area (`livingArea`), size of the house (`size`), zone (`zoneRM`), presence of air conditioning (`acType`), number of bedrooms (`bedrms`) and bathrooms (`bathrooms`). We will focus on the the model and model-estimate level summaries. The remainder are discussed more fully in **?**.

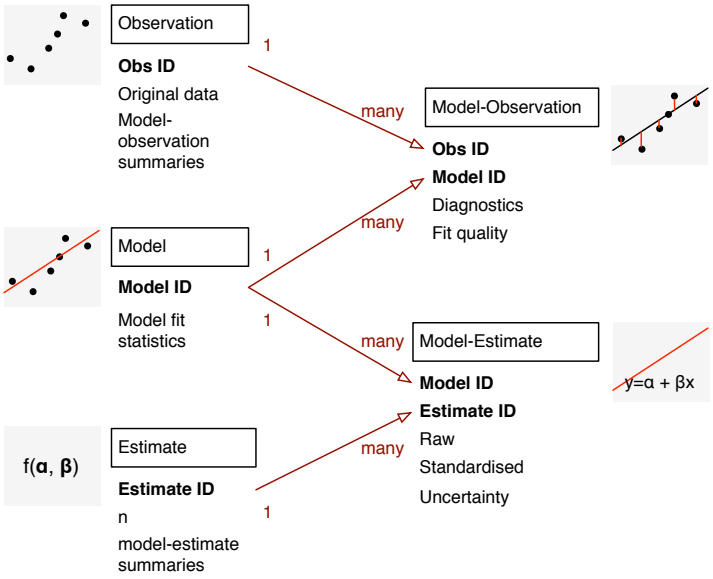

Fig 10: Relationship between five levels of summary statistics for a collection of linear models. Arrows indicate one-to-many relationships, e.g., for each model-level summary, there are many model-observation and model-estimate statistics.

Model summaries. Figure 11 shows model-level summary statistics: model complexity and model fit. We use one measurement of model complexity (degrees of freedom) and five standardized measurements of model fit: log-likelihood, AIC, BIC, R^2 and adjusted R^2. The relationship between complexity and fit is very similar across measures, with improvement in model quality decelerating as model complexity increases. The improvement plateaus after five degrees of freedom (three variables), but small improvements can still be gained by including two more variables.

Model-estimate summaries. Each model contains between 1 and p variables, and for each variable in each model we calculate:

- The raw estimate, useful when all predictors are on the same scale.
- The standardized estimate, generated from models fit to standardized predictors. These are useful as measures of relationship strength for each variable because they can be interpreted as the change in response when the predictor changes by one standard deviation if all other variables are held constant.
- The t-statistic and absolute t-value, allowing us to assess the significance and direction of the relationship between the response and predictor.

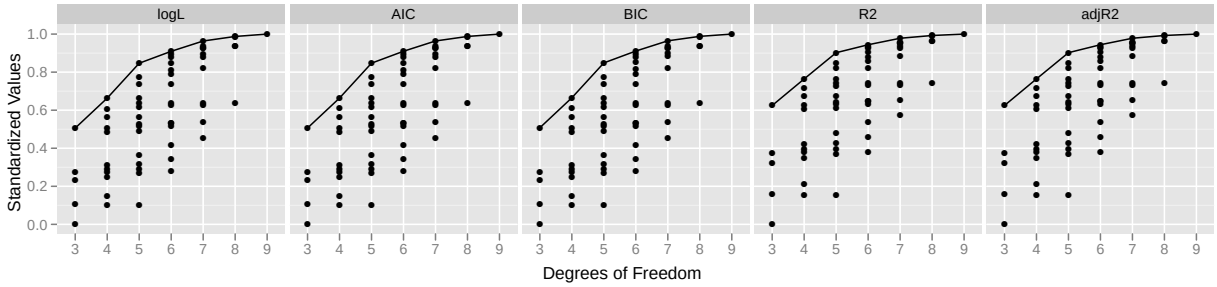

Fig 11: Model summary statistics, each scaled to $[0, 1]$ to aid comparison. A line connects the best models for each degree of freedom. The intercept only model is not displayed. Degrees of freedom include calculation of intercept and variance. These summary statistics suggest that a either five df/three variable or a seven df/five variable model are good tradeoffs between complexity and performance.

We can use this graphic to explore the importance of the predictors. There are two interesting examples of this in Figure 12. First, the standardized estimates of `livingArea` are very similarly positive and large across all models, which suggests that it is the most important variable in predicting housing price. (A value of 0 indicates models in which the variable was not included.) Another interesting phenomenon is the behavior of the estimates for the `bedrms` variable. For some of the models the estimates are negative, suggesting that houses with more bedrooms have lower prices! For other models the relationship is what we would expect, positive coefficients and a positive relationship with price.

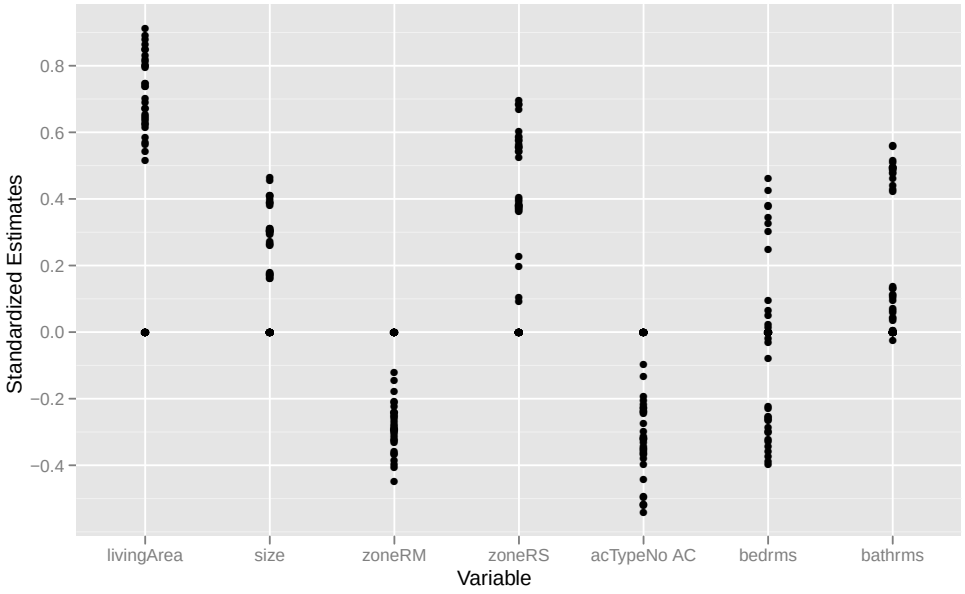

Fig 12: Standardized coefficients by variable, so we can judge relative strength on a common scale. The `livingArea`, `size`, `zoneRS`, and `bathrm` variables always have positive coefficients suggesting positive association with price. The `bdrms` variable has mixed coefficients, some positive and some negative.

Figure 13 uses linked brushing to highlight these models and to examine the fit more closely. We have two plots visible simultaneously, and link the same models between them. The plots on the left side are parallel coordinate plots of the standardized estimates by variable, and plots on the right side show the R^2 by degrees of freedom of the models. In the top row, models with positive coefficients for bedrooms are highlighted in red. These models are surprisingly bad! They have the smallest R^2 values for their relative degrees of freedom. Looking at the plot of estimates shows that these models are all missing the living area variable. Models with negative coefficients for bedrooms are highlighted in the bottom row of plots. These are all good models, with high R^2, and they are also models where living area is included. This suggests collinearity between living area and bedrooms. Exploring these pairs of plots generally helps us to analyze the covariance structure between predictors and the effect on the response.

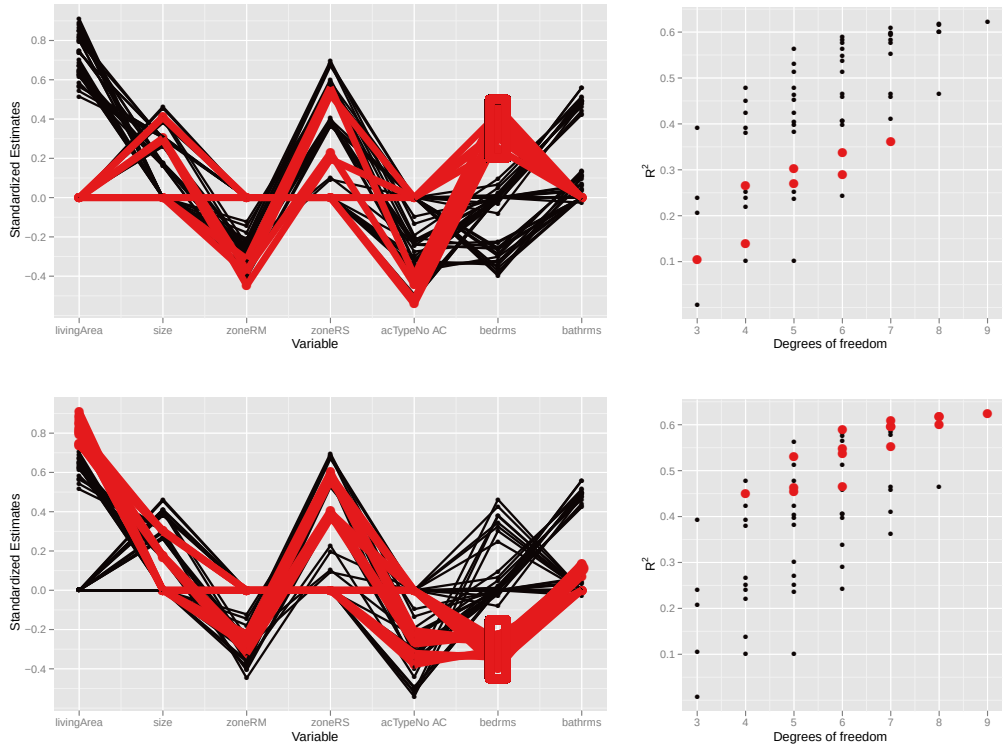

Fig 13: (Left) Parallel coordinates plot of standardized estimates vs variable.(Right) Scatterplot of R^2 vs degrees of freedom. Subsets of models are selected (red) based on their estimates for `bedrms`. (Top) Models with positive estimates have low R^2 values and do not include `livingArea`. (Bottom) Models that have negative estimates for `bedrms` have high R^2 and also high coefficients for livingArea. This would suggest collinearity between these two variables.

These ideas are implemented in the `meifly` package (**m**odels **e**xplored **i**nteractively) (**?**) which uses R to fit the models and calculate the summary statistics, and ggplot2 and GGobi to display them.

5. Don't just look at the final result; explore how the algorithm works. Whenever we can gain insight into the process of model fitting, we should. Observing iterations helps us understand how the algorithm works and can reveal potential pitfalls. Developing suitable visualizations

forces us to think deeply about exactly what the algorithm does and can suggest possible avenues for improvement. For some algorithms, it may be possible to intervene, contributing a more global perspective to help the algorithm escape local maxima.

Many statistical algorithms are inherently iterative: IRLS in the generalized linear model, the Newton-Raphson method in maximum likelihood, the EM algorithm, multidimensional scaling, and k-means clustering. Some of these methods are guaranteed to converge to the global optimum (e.g., IRLS, EM), but most are not. We can think of each iteration as its own model, and we want to explore the changes over time. Many of the messages of Section 4 also apply here: we should look at each step, not just the last one; we should try and display all models on a single plot where possible; and if not, we should develop good summary statistics. The specific feature of collections generated by iteration is that there is a unique ordering of the models.

This ordering suggests two approaches to visualization: ordering in space, to make time series plots, and ordering in time, to make movies. The time series plot displays time on the x-axis and some summary statistic on the y-axis, and shows at a glance the progress of a single parameter. This only works for a small number of numeric summaries, so for more complicated summaries we can make a movie. A movie strings together many static plots, each of which captures the state of the model at a single point in time. The `animation` package (**?**) is a rich source of animations in R, but only provides one animation that displays the progress of an algorithm, k-means clustering.

To capture the data we need for these plots, we can either stream data updates, replacing the values from the previous iteration; or store all iterations for later exploration. Streaming updates allows us to intervene in the algorithm and observe the effect of our changes, while storing everything allows us to later run time forwards and backwards. A particularly nice example of intervening in the progress of an algorithm is `ggvis` (**?**), an interactive tool for multidimensional scaling that allows you to manually drag points out of local optima.

Being able to extract this data is dependent on the implementation of the algorithm. At a minimum we either need a hook into the algorithm which calls a function every n iterations; or we need to be able to run the algorithm for a fixed number of steps, and to be able to start the fitting process from the results of a previous run. This is not always straightforward as many algorithms also have time-varying parameters that must be captured and recreated. Typically, extracting and saving this data will considerably slow the algorithm, often by an order of magnitude.

The following two case studies illustrate some of these ideas. The first case study, on projection pursuit, visualizes the use of simulated annealing to find interesting projections of the data. The second case study explores the fitting process of self organizing maps, as implemented by the `kohonen` package.

5.1. *Case study: Projection pursuit and simulated annealing.* The guided tour (**?**) is a combination of the grand tour and projection pursuit. Instead of randomly picking new projections, as in the grand tour, the guided tour only picks new projections that are more interesting, by optimizing an index of interestingness with simulated annealing (**?**). This case study investigates how we can gain insight into the path that the simulated annealing takes. The indices we use in this example come from the graphic theoretic scagnostics of **?** and are implemented in the `scagnostics` package (**?**).

Figure 14 shows a time series of the scagnostic index being optimized, clumpiness, which takes high values when the data is grouped into distinct clumps. We can see that the paths are not monotone: there is a stochastic component to simulated annealing which helps it avoid local optima. Each of the points on the plot corresponds to a projection matrix, and it is interesting to see if all the peaks correspond to the same or different projections. Figure 15 shows the four projections

with highest clumpiness. Visualizing the projection matrices themselves is difficult (each column corresponds to a point on a p-dimensional hypersphere) but can be revealing (**?**).

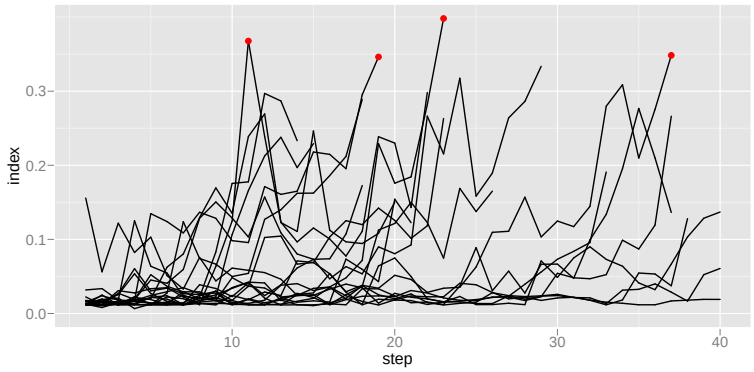

Fig 14: Variation of the clumpy index over time. Simulated annealing with 20 random starts, run until 40 steps or 400 tries. Red points indicate the four highest values.

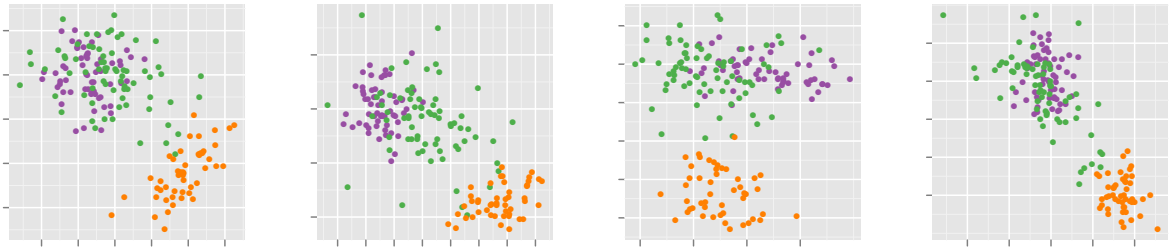

Fig 15: Four projections of the data with highest values of clumpiness, found by the simulated annealing shown in Figure 14. Plots are ordered left to right from highest to lowest. All these projections have good separation between the orange cluster at the others, but not between the purple and green clusters. Looking at the top 15 maxima does not find any projections that separate these two groups.

We can also use interaction with this plot to do something rather interesting: restart the simulated annealing from one of the local optima and see if it can do better in that same location. Figure 16 restarts from a local optima and shows that it is difficult to do better.

5.2. *Case study: self organizing maps.* A self-organizing map (SOM) is a machine learning algorithm which simultaneously performs clustering and dimension reduction (**?**). SOMs are often described as a type of neural network, but they are more easily understood as a type of constrained k-means. The idea is simple: we are wrapping a net of points into the data cloud. The knots in the net are called nodes, and are constrained by their neighbors. Each node has a position in the data space (like a cluster centroid), and the model space (a grid coordinate on the net). In this case study we will look at the self organizing maps implemented in the `kohonen` package (**?**).

We first need a method for visualizing the model in the data space. The majority of SOM displays show the data in the model space, and give little insight into the quality of the model. An idea for a visualization comes immediately from the net metaphor: we'll display the data, the nodes, and the connections between neighboring nodes; a net wrapped through the data. Figure 17 contrasts a

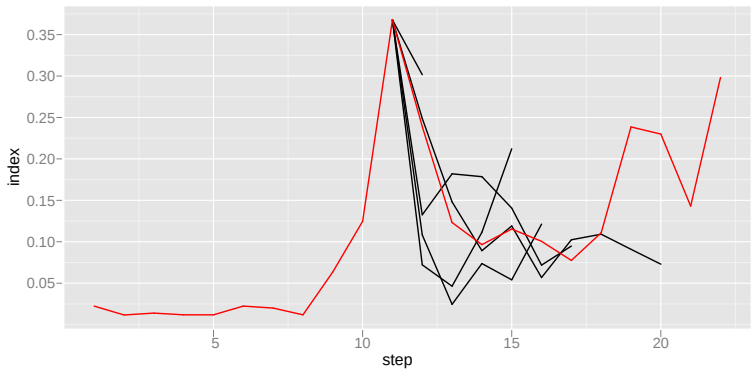

Fig 16: One of the guided tour runs of Figure 14 with new guided tours launched from particularly good projections. Red line shows path of original simulated annealing.

typical d-in-ms view (left) with a m-in-ds view (right) for 10×3 node som fit to the wine data. The d-in-ms view shows the positions of each of the nodes of the net using a Coxcomb display. Nodes in the middle of the net have large values on all variables, that is they are close to the (1,1,1,1,1) region in the data space, and nodes on the right and left side have all low values, indicating they are close to (0,0,0,0,0) region in the data space, assuming the data falls inside a 5D unit cube. In contrast the m-in-ds view shows the nodes relative to the data points, and allows us to see how the model fits to the data, better evaluating the goodness-of-fit. In this one projection the model can be seen to be fairly linear in the 5D space, winding through the three clusters.

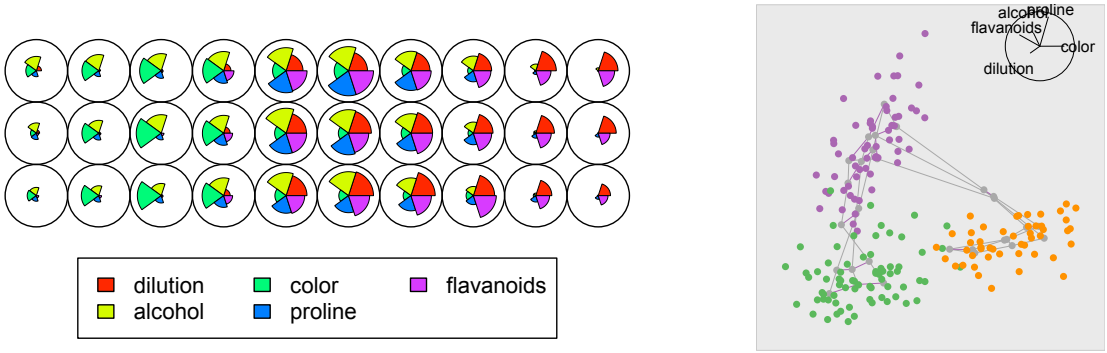

Fig 17: (Left) A visualization of the model space. Each node is represented by a Coxcomb plot which represents the position of that node in the data space. (Right) A projection of the model embedded in the data space.

The algorithm for fitting a SOM is very similar to k-means, but instead of updating a single node at a time, we also update that node's neighbors:

1. Randomly initialize nodes to positions in data space.
2. Iterate through each data point, and assign it to the closest node
   (a) Update the location of node: $(1 - \alpha) \cdot \text{node} + \alpha \cdot \text{point}$
   (b) Also update the location of all neighboring nodes within distance r on grid
3. Decrease α and r, and repeat step 2 until stopping condition is met

For the wine dataset, we'll fit a 10×3 net: we know the data seems to fall along a 1d path, but maybe there's something interesting along another dimension. To record the progress of the model fit, we'll save the following information at each of 100 iterations: the positions of the nodes, the values of r and α, and the distance from each point to its corresponding node.

Figure 18 summarizes the quality of the model fit over time, by plotting the distance between each point and its corresponding node. Looking at the mean alone is misleading: there is a dramatic drop in mean distance when switching from a radius of three to two. However, when we look at the complete distribution, we can see that this drop is small compared to the total variation. We also see a number of outlying points that never get close to a node. The variation in those lines might also make us wonder if we have stopped the algorithm too soon.

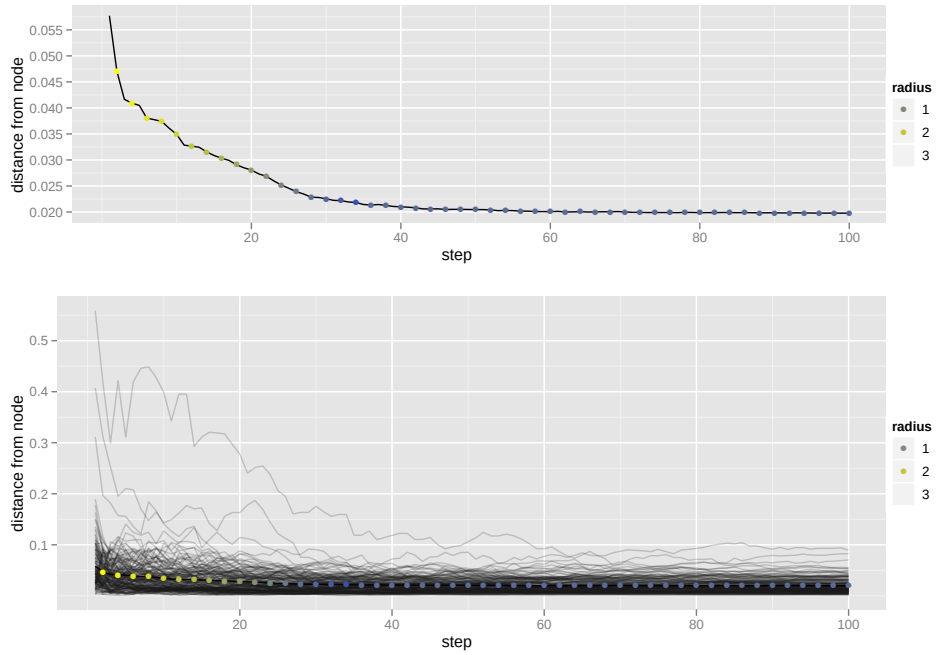

Fig 18: (Top) Time series of mean distance from point to corresponding node, over time. (Bottom) Distance to corresponding node for each individual point. Note the difference in scale! Mean points are colored by radius. Alpha is not displayed, but deceases linearly from 0.05 to 0.

To see how the model changes over time, Figure 19 displays the model in the data space for four selected iterations. A movie showing more iterations, more views of the model, and interaction between the model and model summaries is available at http://vimeo.com/823541. It's interesting to note that the green and orange clusters are close in the data space, but distant in the model space. We could improve our SOM model by joining the two ends. We also might have expected that the middle of the net would go through the green group, not the purple group, but this is likely a function of the initialization. With increasing iterations the model expands out and is closer to joining the two ends, but also becomes more twisted. It may be too many iterations.

When we first used this package, it was not possible to break up the iterations into single steps, because there wasn't sufficient control over the changes to r and α. After asking the package author to give access to the iterations, this visualization revealed some problems with the implementation of the SOM. The m-in-ds visualization also revealed that the nets were twisted and tangled up. The author has since fixed these initial bugs.

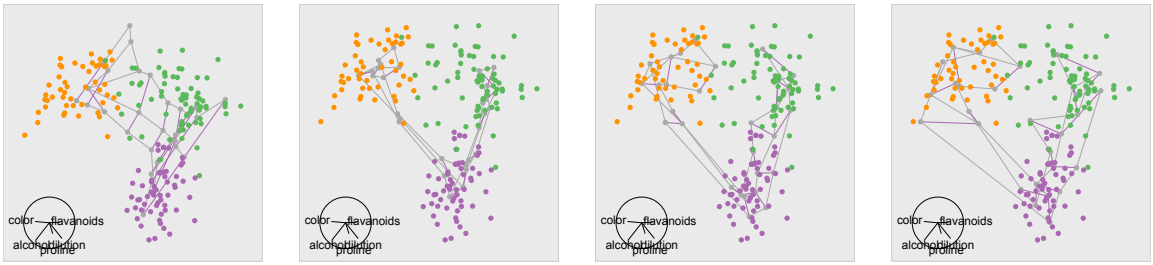

Fig 19: Iterations 1, 5, 25 and 100 shown in the same tour projection. After five iterations the model seems to run through the center of clusters fairly well, and continues to change some over the next 99 iterations. The net extracts a 1d path between the 3 groups, and then expands further.

6. Conclusion. We have presented three important strategies for visualizing models: visualize the model in data space, collections are more informative than singletons and explore the process of model fitting. These were illustrated with many case studies that developed informative visualizations using the strategies outlined in the paper. Many of these techniques have grown out of our work with `rggobi` (**?**), an R package which connects the statistical programming environment of R (**?**) with the interactive and dynamic graphics of GGobi (**?**). This paper ties together the ideas underlying three R packages that use `rggobi`: `classifly` (**?**), `clusterfly` (**?**), and `meifly` (**?**). The tight connection between R and GGobi facilitated the development of all of the examples, and has resulted in the development of R packages that one can easily download and try out.

To make these tools useful in practical model building we need to improve our ability to visualize high-d data and surfaces. We have the basic tools, but as the number of dimensions increases, the number of potential views becomes overwhelming. How can we provide support for finding revealing views, while maintaining the ability of the analyst to look for the unexpected? We have touched on projection pursuit for finding interesting views of the data, and we expect developing descriptive statistics that summarize both the model and data to be a fruitful approach. The challenge is, as always, developing good summaries, but our graphical tools can help us develop these for low-d data for application to high-d data.

We also need better tools to visualize high-d surfaces. In this paper we have gone to considerable lengths to approximate high-dimensional surfaces with a dense set of points. Development of better sampling methods would be useful, but the development of higher dimensional drawing primitives is more of a priority. GGobi currently provides 0d (points) and 1d (lines) primitives, can we supplement these with 2D primitives (triangles?) as well? The development of adaptive contouring algorithms for high-d would also be useful.

The case studies in this paper are just a beginning and we hope that others pick up these ideas to develop visualizations for more models in a wider array of statistical and graphical environments.

7. Acknowledgements. This material is based upon work supported by the National Science Foundation under Grant No. 0706949.

Department of Statistics MS-138
6100 Main St
Houston TX 77081
E-mail: hadley@rice.edu

Department of Statistics
Snedecor Hall
Ames IA 50010
E-mail: dcook@iastate.edu

Department of Statistics
Snedecor Hall
Ames IA 50010
E-mail: hofmann@iastate.edu