SHORT TECHNICAL NOTE

New and simplified manual controls for projection and slice tours, with application to exploring classification boundaries in high dimensions

Alex Aumann[a], German Valencia[a], Ursula Laa[b], Dianne Cook[c]

[a]School of Physics and Astronomy, Monash University; [b]Institute of Statistics, University of Natural Resources and Life Sciences, Vienna; [b]Department of Econometrics and Business Statistics, Monash University

ARTICLE HISTORY
Compiled July 8, 2022

ABSTRACT
Something here

KEYWORDS
data visualisation; grand tour; statistical computing; statistical graphics; multivariate data; dynamic graphics

1. Introduction

From a statistical perspective it is rare to have data that are strictly 3D, and so unlike in most computer graphics applications, the more useful methods for data analysis show projections from an arbitrary dimensional space. These are dynamic data visualizations methods and are collected under the term *tours*. Tours involve views of high-dimensional (p) data with low-dimensional (d) projections. In his original paper on the grand tour, Asimov (1985) provided several algorithms for tour paths that could theoretically show the viewer the data *from all sides*. Prior to Asimov's work, there were numerous preparatory developments including Fisherkeller, Friedman, and Tukey (1974)'s PRIM-9. PRIM-9 had user-controlled rotations on coordinate axes, allowing one to manually tour through low-dimensional projections. (A video illustrating the capabilities is available through video library of ASA Statistical Graphics Section (2022).) Steering through all possible projections is impossible, unlike Asimov's tours which allows one to quickly see many, many different projections. After Asimov there have been many, many tour developments, which are summarized in Lee et al. (2021).

One such direction of work develops the ideas from PRIM-9, to provide manual control of a tour. Cook and Buja (1997) describe controls for 1D (or 2D) projections, respectively in a 2D (or 3D) manipulation space, allowing the user to select any variable axis, and rotate it into, or out of, or around the projection through horizontal, vertical, oblique, radial or angular changes in value. Spyrison and Cook (2020) refined this algorithm and implements them to generate animation sequences.

CONTACT Alex Aumann. Email: aaum0002@student.monash.edu, German Valencia. Email: german.valencia@monash.edu, Ursula Laa. Email: ursula.laa@boku.ac.at, Dianne Cook. Email: dicook@monash.edu

Manual controls are especially useful for assessing sensitivity of structure to particular elements of the projection. There are many places where it is useful. In exploratory data analysis, where one sees clusters in a projection, can some variables be removed from the projection without affecting the clustering. For interpreting models, one can reduce or increase a variable's contribution to examine the variable importance. These controls can also be used to interactively generate faceted plots (**?**), or spatiotemporal glyphmaps (**?**). Having the user interact with a projection is extremely valuable for understanding high-dimensional data. However, these algorithms have two problems: (1) the pre-processing of creating a manipulation space overly complicates the algorithm, (2) extending to higher dimensional control is difficult.

Through experiments with the relatively new interactive graphics capabilities in mathematica(?), we have realized that there is a simpler approach, which is more direct, and extensible for generating user interaction. This paper explains this, and is organized as follows. The next section describes the new algorithm for manual control. This is followed by details on implementation. The applications section illustrate how these can be used.

2. How to construct a manual tour

A manual tour allows the user to alter the coefficients of one (or more) variables contributing to a $d - D$ projection. The initial ingredients are an orthonormal basis ($A_{p\times d}$) defining the projection of the data, and a variable id ($m \in \{1, ..., p\}$) specifying which coefficient will be changed. A method to update the values of the component of the controlled variable V_m is then needed.

2.1. *Existing methods*

The method for updating component values in Cook and Buja (1997) (and utilised in Spyrison and Cook (2020)) are prescribed primarily for a 2D projection, to take advantage of (then) newly developed 3D trackball controls made available for computer gaming. The first step was to construct a 3D manipulation space from a 2D projection, as illustrated in Figure 1. In this space the coefficient of the controlled variable ranges between -1 and 1. Movements of a cursor are recorded and converted into changes in the values of V_m thus changing the displayed 2D projection. The algorithm also provided constraints to horizontal, vertical, radial or angular motions only.

2.2. *A new simpler and broadly applicable approach*

The new approach emerged from experiments in mathematica. The components corresponding to V_m are directly controlled by cursor movement, which updates row m of A. The updated matrix is then orthonormalised.

2.2.1. Algorithm

1. Provide A, and m. (Note that m could also be automatically chosen as the component that is closest to the cursor position.)
2. Change values in row m, giving A^*. A large change in these values would correspond to making a large jump from the current projection. Small changes would correspond to tracking a cursor, making small jumps from the current projection.

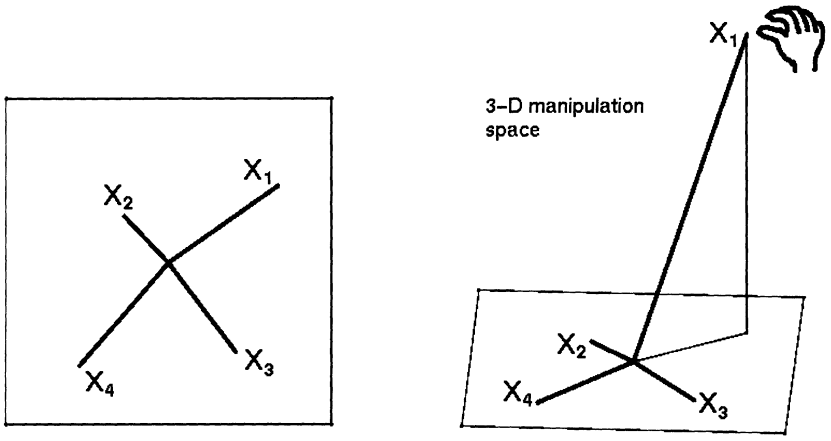

Figure 1. Original contruction of the manual tour designed for 2D projections and created a 3D space from which to utilise track ball controls to change it's contribution. (Figure 3 from Cook and Buja (1997).)

3. Orthonormalise A^*, using Gram-Schmidt. For $d = 2$, and $A^* = [a_{.1}\ a_{.2}]$, the steps are:
   i. Normalise $a_{.1}$, and $a_{.2}$.
   ii. $a^*_{.2} = a_{.2} - a^T_{.1} a_{.2} a_{.1}$.
   iii. Normalise $a^*_{.2}$.

This algorithm will produce the changes to a projection as illustrated in Figure 2 (top row). The controlled variable, V_m, corresponds to the black line, and sequential changes to row m of A can be seen to roughly follow a specified position (orange dot). Changes in the other components happen as a result of the orthonormalisation, but are uncontrolled.

2.3. *Refinements to enforce exact position*

The problem with new simple method (Algorithm 1) is that the precise values for V_m cannot be specified because the orthonormalisation will change them.

2.3.1. Adjustment method 1

A small modification to algorithm 1 will maintain the components of V_m precisely (Figure 2 (bottom row)). It is as follows:

1. Provide A, and m.
2. Change values in row m, giving A^*.
3. Store row m separately, and zero the values of row m in A^*, giving A^{*0}.
4. Orthonormalise A^{*0}, using Gram-Schmidt.
5. Replace row m with the original values, giving A^{**}.
6. For $d = 2$, adjust the values of $a^{**}_{.2}$ using

$$a^{**}_{j2} + \frac{a_{m1} a_{m2}}{p-1}, j = 1, ..., p, j \neq m$$

.

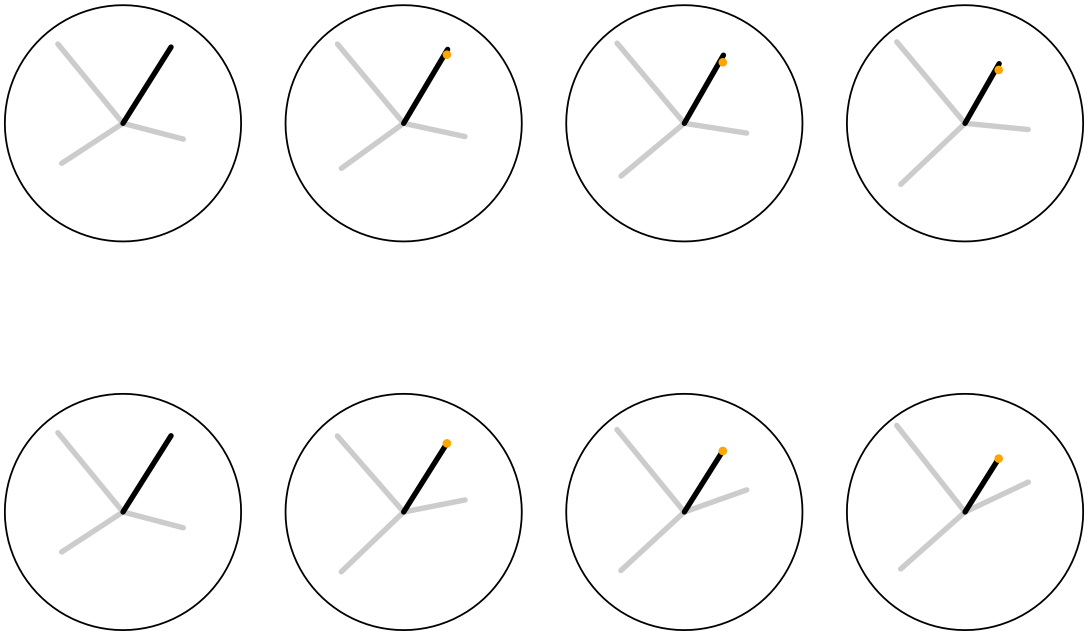

Figure 2. Sequence of projections where contribution of one variable is controlled (black) is changed: (top) unconstrained orthonormalisation, (bottom) constrained as specified. The dot (orange) indicates the chosen values for the controlled variable. For the constrained orthonormalisation it can be seen to precisely match the axis, but not so for the unconstrained orthonormalisation.

which ensures that

$$\sum_{j=1,j\neq m}^{p} a_{j1}^{**} a_{j2}^{**} + a_{m1} a_{m2} = 0$$

.

If $d > 2$ the process would be sequentially repeated in the same manner that Gram-Schmidt is applied sequentially to orthormalise the columns of a matrix. If $d = 1$ no orthonormalisation is needed, and the projection vector would simply need to be normalized after each adjustment.

2.3.2. Adjustment method 2

For $d = 2$ projections, the projection matrix is the sub-matrix of A formed by its first two columns. Whereas orthonormality of the basis for the p-dimensional space is given by $e_i \cdot e_j = \delta_{ij}, i, j, = 1, \cdots ,$, orthonormality of the projection matrix is expressed as $P_i \cdot P_j = \delta_{ij}, i, j = 1, 2$. Movement of the cursor takes the two components x_{m1}, x_{m2} into a selected new value a, b. Although the motion is constrained by $a^2 + b^2 \leq 1$, this is not sufficient to guarantee orthonormality of the new projection matrix. One possible algorithm to achieve this is

(1) Cursor movement takes $x_{m1}, x_{m2} \rightarrow a, b$
(2) The freedom to change the components $A_{i3} \cdots A_{ip}$ (the columns of A not corresponding to the projection matrix) is used to select a new orthonormal basis as follows:
   (a) For row m one chooses $A_{m3} = \sqrt{1 - a^2 - b^2},\ A_{m,k>3} = 0$

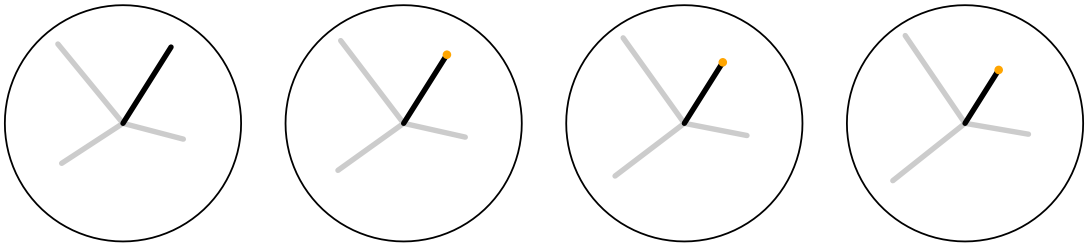

Figure 3. Manual controls with algorithm 2. The precise location of the axis is maintained.

(b) For other rows, $A_{i\neq m, j\geq 3}$ random selections in the range $(-1, 1)$ are made.
(c) The Gram-Schmidt algorithm is then used to obtain an orthonormal basis taking e_m as the first vector (which is already normalized), and then proceeding as usual $e_1 \to e_1 - (e_1 \cdot e_m)e_m$, $e_1 \to e_1/(e_1 \cdot e_1)$, etc.

(3) this results in the orthonormal basis A^* and a new projection matrix with $P_{m1} = a,\ P_{m2} = b$.

The random choice for the components of A not in the projection matrix allows the exploration of dimensions perpendicular to the projection plane.

$$A = \begin{array}{c} \\ e_1 \to \\ e_2 \to \\ \vdots \\ e_m \to \\ \vdots \\ e_p \to \end{array} \begin{array}{c} \begin{array}{cc} P_1 \downarrow & P_2 \downarrow \end{array} \\ \begin{pmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdot & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdot & \cdots & x_{mp} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \cdot & \cdots & x_{pp} \end{pmatrix} \end{array}, \quad (x_{m1}, x_{m2}) \to (a, b) \tag{1}$$

2.3.3. Adjustment method 3

For now just sketching the idea:

- first step is to click on the axis display to change the contribution of one variable m
- capture that position and replace the corresponding row in the projection matrix
- for 2D projection this gives components m_1 along "x" direction and m_2 along y direction
- next step: rotate basis such that direction of row m now corresponds to the first basis vector, this means we apply 2x2 rotation matrix to each row of the projection matrix, where the rotation angle is $tan(\theta) = m_2/m_1$ (that matrix can be written just in terms of that ratio, $x = m_2/m_1$, when $m_1 < 0$ I think we just need to translate $\theta \to \theta + \pi$)
- take the rotated basis, apply Gram-Schmidt, now the direction of m will not change (but the length will change during normalization)
- rotated back to the original xy basis to update the plots

XXX need to check calculation of angle when $m_1 < 0$

3. Experimenting with new techniques using Mathematica

- **Why is this a good sandbox**
- **Explain the functionality available in the notebooks**

The following functions are implemented in the Mathematica package `mmtour.wl`:

- `ProjectionPlot[data, projmat, xRange(=Automatic), yRange(=Automatic), colour(=Automatic)]` with arguments
  - `data`: A list of data matrices (they must all have the same dimensionality)
  - '`projmat`: A projection matrix which describes the projection plane. This is the initial projection and does not need to be orthonormal (XXX is this right?).
  - '`xRange`: A list which details the range of the horizontal axis.
  - '`yRange`: A list which details the range of the vertical axis.
  - '`colour`: A list of graphics directives to be applied to each data matrix.

There are problems with the default values for x and y ranges and colour. It would be better to fix the ranges and colour based on the data as well as the origin of axes and the size of the points.

It would be good if the initial projection matrix (and maybe the center) had default values. Perhaps a random matrix with the correct dimensions read from the data? Fix 1

The format for the printed projection matrix can be improved also. Maybe fix the number of digits and the position of the label.

- `SlicePlot[data, projmat, centrePoint, height, heightRange, ptSize1(=0.005), ptSize2(=0.004), minDist(=0)]` with arguments (used to be called SliceDynamic in Alex's first version, the name SliceDynamic was taken by something else below)
  - `data`: A data matrix
  - `projmat`: The inital projection matrix
  - `centrepoint`: The initial position of the slice
  - `height`: The initial height of the slice
  - `heightRange`: The range of heights
  - `ptSize1`: The size of the points that exist within the slice
  - `ptSize2`: The size of the points that exist outside of the slice

This function requires some auxiliary functions. It does not accept more than one data set.

It would be good if the initial projection matrix (and maybe the center) had default values. Perhaps a random matrix with the correct dimensions read from the data?

It may be good to fix the size of the points, the center of the axes and the scales of the axes rather than allow them to change as the projection and slicing changes

- `SliceDynamic[data, projmat, centrePoint, height, heightRange, ptSize1 : 0.005, ptSize2 : 0.004]` with arguments
- `data`: A data matrix with an extra grouping or clustering column (numerical)
- `projmat`: The initial projection matrix
- `centrepoint`: The initial position of the slice
- `height`: The initial height of the slice

- `heightRange`: The range of heights
- `ptSize1`: The size of the points that exist within the slice
- `ptSize2`: The size of the points that exist outside of the slice

Does this need to be a separate function from SlicePlot? Fix 2

It would be good if the initial projection matrix (and maybe the center) had default values. Perhaps a random matrix with the correct dimensions read from the data? Fix 3

It may be good to fix the size of the points, the center of the axes and the scales of the axes rather than allow them to change as the projection and slicing changes. Fix 4

I have found it useful to input different data sets rather than have this function sort them automatically using the last column. One data set with a flag column at the end, is best! Is the code too complicated? Fix 5

- `Projected2DSliderPlot[]`

This function uses a different display for the projection that I find very useful. However as it stands it is specific to one data set. Needs to be generalised to arbitrary data if we want to keep it.

For Alex to do

- **We would need a screenshot (and video) for each of these.**
- **Fix 1, 2, 3, 4, 5**
- **Code consistent for applications to work - see slice_tour_compare_sets.nb. See below**
  - **We need to make sure the drawing range is fixed (no resizing of the axis when changing projection or slicing). It could be nice if there was a manual zoom in/zoom out to change scale occasionally.**
  - **Should also make sure the size of the points is fixed (maybe this will be automatic if the drawing range is fixed?), bigger points would be preferred, currently the size is sometimes really tiny and it becomes difficult to see anything.**
  - **Can we have some heuristic to decide the range for the slice thickness slider? In the second graph there is a tiny part of the slider where the changes are interesting, difficult to navigate since this makes it super sensitive to any small movement, while most of the options are not interesting. My suggestion would be to keep (as I understand is the case now) the maximum thickness (all points in) as the upper bound, but instead of zero use a thickness below which slices tend to be empty as the lower bound.**
  - **For the olives example I think the relation between x2 and the blue region could be interesting.**
  - **pdfsense data: can we use random samples instead of the first 10k rows? Then we should still be able to see the bigger picture (with picking the first ones we get some artificial structure from how the data was sampled initially).**
  - **I was playing around with this today. Looks like the axes are fixed when changing slice thickness, but not when rotating?**
  - **Anyways, some observations that might be useful, but all of this**

seems difficult for including in a paper:

- **In the first example we can show how the yellow group is associated with the first variable (x1, I guess this is C9?) - when slicing through the center it only appears when x1 has a bigger component on the projection, it is associated with low values of x1. Since they are not near the average value of x1 the points do not get captured in a slice where x1 is not important in the projection. We can move the center point to -1 in the first component to then explore how the yellow group relates to the other three variables. Much less clear, maybe some indication of smaller x2 and larger x3 for this.**
- **The second example is too busy for manual exploration I would say (too many axes to move around separately).**
- **Third example could be similar to the first: blue region requires small values of x2, can be understood from the exploration, then move center point to -0.5 in x2 to explore further. Now interpretation becomes tricky. Sometimes blue and yellow seem just randomly everywhere, sometimes blue more contained in an ellipsoid around the center (indicating some correlation between variables). For example try zeroing out components 2 and 3 to see this slice in x1 vs x4.**
- **For the last example x4 seems to be the one to work with. I feel like this is again very similar, maybe better to instead explore other examples (maybe something without slicing?)**

4. Application

To illustrate the usefulness of the manual tour we use the 4D penguins data (Horst, Hill, and Gorman 2020). We will show how classification boundaries can be explored and better understood using a manual tour, on projections and slices through 4D space. Figure **??**fig:penguins-scatmat) shows a scatterplot matrix of this data. There are four variables (`bl = bill_length_mm, bd = bill_depth_mm, fl = flipper_length_mm, bm = body_mass_g`) measuring the size of the penguins from three species (Adelie, Chinstrap and Gentoo). The scatterplot matrix shows that the three species appear to be likely separable, and that at least the Gentoo can be distinguished from the other two species when `bd` is paired with `fl` or `bm`. The steps for exploring boundaries are as follows:

1. Build classification model, here we use a random forest and linear discriminant analysis.
2. Predict the class for a dense grid of values covering the data space.
3. Examine projections, using a manual tour so that the contribution of any variable is controlled.
4. Slice through the center, to explore where the boundaries will likely meet.
5. Move the slice by changing the center in the direction of a single variable to explore the extent of a boundary for a single group relative to a variable.

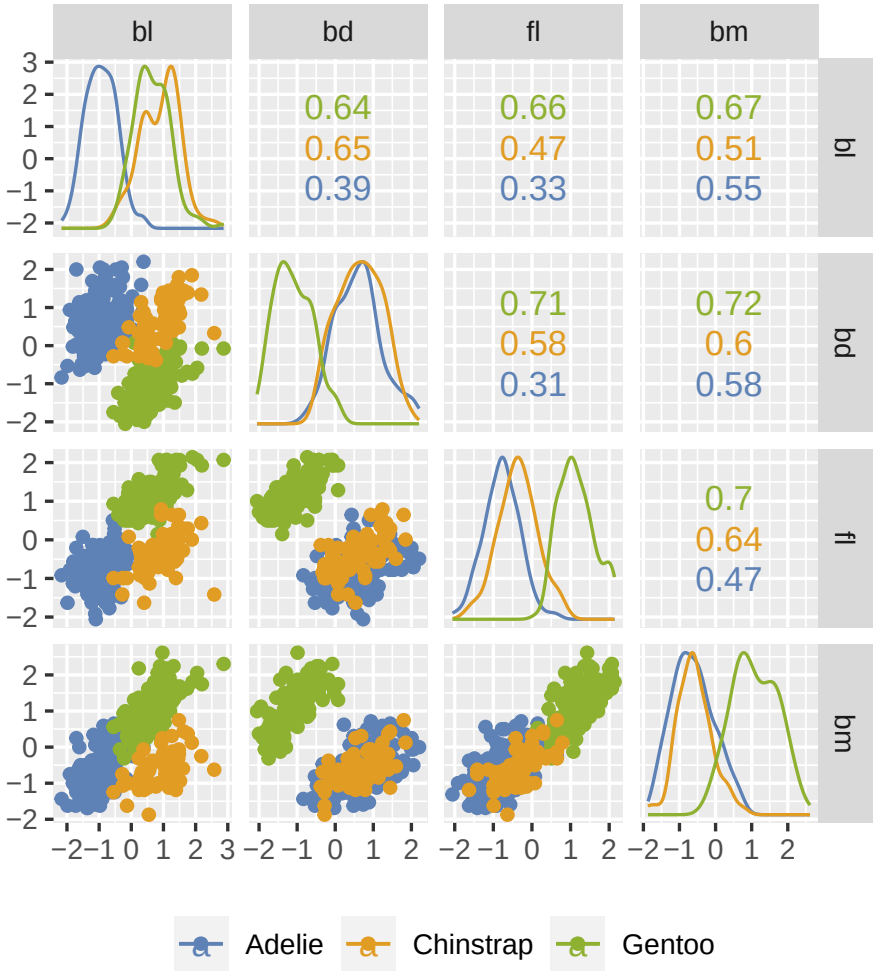

Figure 4. Scatterplot matrix of the (standardised) penguins data. The three species are reasonably different in size, with Gentoo distinguised from the other two on body depth relative to flipper length and body mass.

4.1. *Constructing the 4D prediction regions*

We use the `classifly` package (Wickham 2022) to generate predictions across the 4D space spanned by the data, with two classification models (linear discriminant analysis and random forest).

4.2. *Exploring projections manually*

The first projection we will start from is looking at bill depth vs flipper length (x2 vs x3). In a projection of the data we see that the Gentoo penguins are separate from the other two species which are on top of each other. If we instead look at a slice through the center of the distribution, we see that shifting the center point to large values of x1 (bill length), e.g. x1=2, results in Adelie penguins dropping out of the slice, while a slice with low values (x1=-2) in the center point only contains Adelie penguin observations.

We can next explore how this is mapped in the two classification models. Both random forest and LDA can separate out the region for Gentoo penguins in a thin slice through the center (h = 0.5), while Adelie and Chinstrap predictions overlap (blue and yellow, an effect of the slice thickness). We also notice the different boundaries: while LDA produces linear boundaries by construction, random forest insead produces "boxes" from cutting up the data in the tree structure.

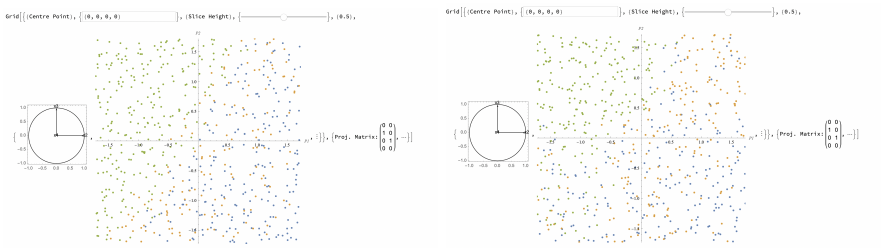

Figure 5. Model predictions in a centered slice defined by the projection onto bill depth vs flipper length, from an LDA and RF model. The colors are the same as used in the SPLOM, we see that Gentoo penugins are clearly separated, while the other two species are overlapping, and we can see the distinugishing features of the two models in terms of the different boundaries.

4.3. *Slicing through the center*

We can then start shifting the slice along x1 as we did when looking at the data, again moving to ±1 will separate the Adelie from the Chinstrap penguins, now in terms of the model predictions. (Could include screenshot here as well.)

4.4. *Shiftinh the slice center*

Thus, stepping through the different slices shows the relation between the species and the first variable (x1, bill length). Instead of moving the center point we could also keep the slice fixed as passing through the center of the distribution, and use the manual controls to rotate x1 into the projection. Here we started from the projected view of the data to identify a combination that can separate all three species, and then use the projection matrix as input to the slice displays for the model. Interestingly,

the two models agree on how to classify the "empty" regions (where there are no data points) for this example. (this is applying A3 as the starting projection).

Finally we use the manual tour on the random forest predictions first, to identify a centered slice with an interesting classification boundary. Here a combination of all four predictors is found to result in an interesting slice, where the predictions for the Gentoo penguins gets split into two regions of the slice, indicating the non-linearity of the random forest model. While x4 (body mass) is already contributing a lot in the projection, we can learn more by also moving the center point along that direction. Moving to high values (x4 = 2) results in a much larger part of the slice being predicted as Gentoo penguins, while now the yellow region (Chinstrap penguins) gets split. Moving in the opposite direction (x4 = -1) instead shows primarily Chinstrap penguins in the predictions, with Gentoo only appearing along the edge.

Looking at the data, and shifting a fat slice (h = 1.5) as specified above (x4 = -1, 0, 2) we see that slices through x4 = -1, 0 do not have observations from the Gentoo species, while x4 = 2 contains the Gentoo observations, but hardly any of the Chinstrap penguins.

4.5. *Comparison of boundaries produced by different classifiers*

The last step is comparing to the LDA predictions in those same slices. With RF we could see the non-linear aspects in the boundaries, while LDA can only produce linear boundaries, so these look very different. On the other hand, the same dependence is observed when shifting the slice along the x4 direction.

5. Extension added in the R package `tourr`

Explain `radial_tour`

6. What would be desirable for implementations in R?

7. Discussion

Acknowledgements

The authors gratefully acknowledge the support of the Australian Research Council. The paper was written in `rmarkdown` (Xie, Allaire, and Grolemund 2018) using `knitr` (Xie 2015).

Supplementary material

The source material and animated gifs for this paper are available at

References

ASA Statistical Graphics Section. 2022. "Video Library." `https://community.amstat.org/jointscsg-section/media/videos`.

Asimov, D. 1985. "The Grand Tour: A Tool for Viewing Multidimensional Data." *SIAM Journal of Scientific and Statistical Computing* 6 (1): 128–143.

Cook, Dianne, and Andreas Buja. 1997. "Manual Controls for High-Dimensional Data Projections." *Journal of Computational and Graphical Statistics* 6 (4): 464–480. http://www.jstor.org/stable/1390747.

Fisherkeller, M.A., J.H. Friedman, and J.W. Tukey. 1974. "PRIM-9, an interactive multidimensional data display and analysis system." In *The Collected Works of John W. Tukey: Graphics 1965-1985, Volume V*, edited by William S. Cleveland, 340–346.

Horst, Allison Marie, Alison Presmanes Hill, and Kristen B Gorman. 2020. *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*. R package version 0.1.0, https://allisonhorst.github.io/palmerpenguins/.

Lee, Stuart, Dianne Cook, Natalia da Silva, Ursula Laa, Earo Wang, Nick Spyrison, and H. Sherry Zhang. 2021. "Advanced Review: The State-of-the-Art on Tours for Dynamic Visualization of High-dimensional Data." *arXiv:2104.08016 [cs, stat]* http://arxiv.org/abs/2104.08016.

Spyrison, Nicholas, and Dianne Cook. 2020. "spinifex: an R Package for Creating a Manual Tour of Low-dimensional Projections of Multivariate Data." *The R Journal* 12 (1): 243. https://journal.r-project.org/archive/2020/RJ-2020-027/index.html.

Wickham, Hadley. 2022. *classifly: Explore Classification Models in High Dimensions*. R package version 0.4.1, https://CRAN.R-project.org/package=classifly.

Xie, Yihui. 2015. *Dynamic Documents with R and knitr*. 2nd ed. Boca Raton, Florida: Chapman and Hall/CRC. https://yihui.name/knitr/.

Xie, Yihui, Joseph J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Chapman and Hall/CRC. https://bookdown.org/yihui/rmarkdown.