Glass Box UMAP: A Python package for interpretable UMAP via exact feature contributions

Glass Box UMAP transforms UMAP into an interpretable embedding workflow by decomposing each point's coordinates into exact contributions from the original input features, letting users identify which variables drive clusters, gradients, and outliers in high-dimensional data.

Published May 16, 2026

Arcadia Science

DOI: 10.57844/arcadia-4ye8-8tun

Purpose

Uniform manifold approximation and projection (UMAP) is widely used to embed high-dimensional data into lower-dimensional manifolds. However, the nonlinear mapping learned by UMAP makes it difficult to understand which input features are responsible for the position of any particular point, limiting its analytical utility.

We built a Python package called Glass Box UMAP ("glass-box-umap" on PyPI) that enables interpretation of the exact feature contributions within UMAP embeddings. We're sharing it for scientists and data practitioners who use UMAP to explore high-dimensional datasets and want to understand which measured variables organize the structure they see.

Statement of need

UMAP is used to visualize high-dimensional data on a manifold, revealing clusters, gradients, neighborhoods, and outliers associated with patterns of variation [1].

Unfortunately, the nonlinear mapping it learns makes the embeddings difficult to interpret. A researcher may see separated clusters or local substructure, but still not know which measured variables placed a point where it appears. Linear methods like PCA have the opposite trade-off: they're interpretable but can't separate complex nonlinear structures. Given these trade-offs, methods that can bridge desirable components of linear and nonlinear methods are of obvious interest.

One approach is to explain model outputs post hoc with attribution methods such as SHAP [2], LIME [3], and Grad-CAM [4], as well as tools for dimensionality reduction like MCML [5]. These methods are valuable but produce linear approximations: attributions come from local sampling or surrogate models rather than from the embedding function itself, and they aren't guaranteed to sum to the coordinate they aim to explain. The nonlinear components that often make the representation useful are simply discarded.

We created Glass Box UMAP to preserve these nonlinear components while making their relationships with input features interpretable. In a previous pub [6], we introduced the fundamental principles behind the method. It uses parametric UMAP [7] to learn a mapping function between embedding coordinates and inputs via a neural network. The neural network architecture is constrained to be homogeneous of order 1 at inference so that the learned embedding function can be decomposed exactly [8] [9] [10]. The embedding of each sample can be written as a sum of feature-level contributions:

$$z_i = \sum_{j=1}^{p} c_{ij}$$

where z_i is the embedding of sample i, p is the number of input features, and c_{ij} is the contribution of feature j to that sample's embedding coordinates. Equivalently, for each embedding dimension k, $z_{ik} = \sum_j c_{ijk}$.

To illustrate the theory, we previously analyzed a single-cell gene expression dataset and showed that exact feature contributions revealed which genes shaped the positions of individual cells in the embedding, offering a direct view of what UMAP had learned [6]. However, our analysis was written in a bespoke Jupyter

Notebook, with no stable, documented, reusable interface. Thus, until now, there's been no general-purpose software package that allows users to fit glass-box UMAP models and calculate exact feature contributions in their own datasets.

We created Glass Box UMAP to fill this need by providing an installable Python package that fits interpretable UMAP-like embeddings and computes exact per-feature contributions. It's designed for researchers who want interpretable, nonlinear embeddings.

Software design

Following the lead of umap-learn **[11]**, the authoritative Python implementation of UMAP, Glass Box UMAP adopts an API convention borrowed from scikit-learn **[12]**, in which models are represented as classes with standard methods like `fit` and `transform`. The primary class in the package is `GlassBoxUMAP`, which, in simple cases, is a drop-in replacement for the `UMAP` object in umap-learn.

The encoder stack is built with PyTorch **[13]** and PyTorch Lightning **[14]**. PyTorch Lightning handles the training loop, device placement, checkpointing, and custom callbacks. Users can take advantage of GPU acceleration when available without changing the API.

The package separates three conceptual stages: embedding, attribution, and plotting. The embedding stage trains a parametric UMAP model using a neural network architecture designed to support exact decomposition into local linear representations. The attribution stage computes per-sample feature contributions using the model's local linear mapping, returning a feature contribution array of shape `(n_samples, n_embedding_dimensions, n_features)` that can be used however the user sees fit. Summing across the feature axis recovers the embedding coordinates exactly, so the reconstruction property is easy to verify on any fit. The package provides visualization utilities as an optional layer for users who want to explore embeddings using out-of-the-box interactive solutions (Figure 1). We designed these plotting utilities as opt-in in order to maintain a minimal core dependency set.

Example application

Since feature contributions are still an emerging object of analysis, the package doesn't prescribe a single interpretation workflow. Instead, the documentation presents a range of example datasets that span different structures and domains, including tabular, image-derived, synthetic-manifold, and gene-expression examples. These examples show how users can reason about contribution tensors in different contexts while making clear that best practices for interpreting Glass Box UMAP outputs will continue to evolve with community use.

In Figure 1, we include one particular dataset as a compact example. The dataset contains 178 wines produced from three cultivars, with 13 chemical measurements for each wine. The features in this dataset are few enough and familiar enough to interpret directly, including chemical measurements such as proline, flavonoids, color intensity, ash, alcohol, and magnesium.

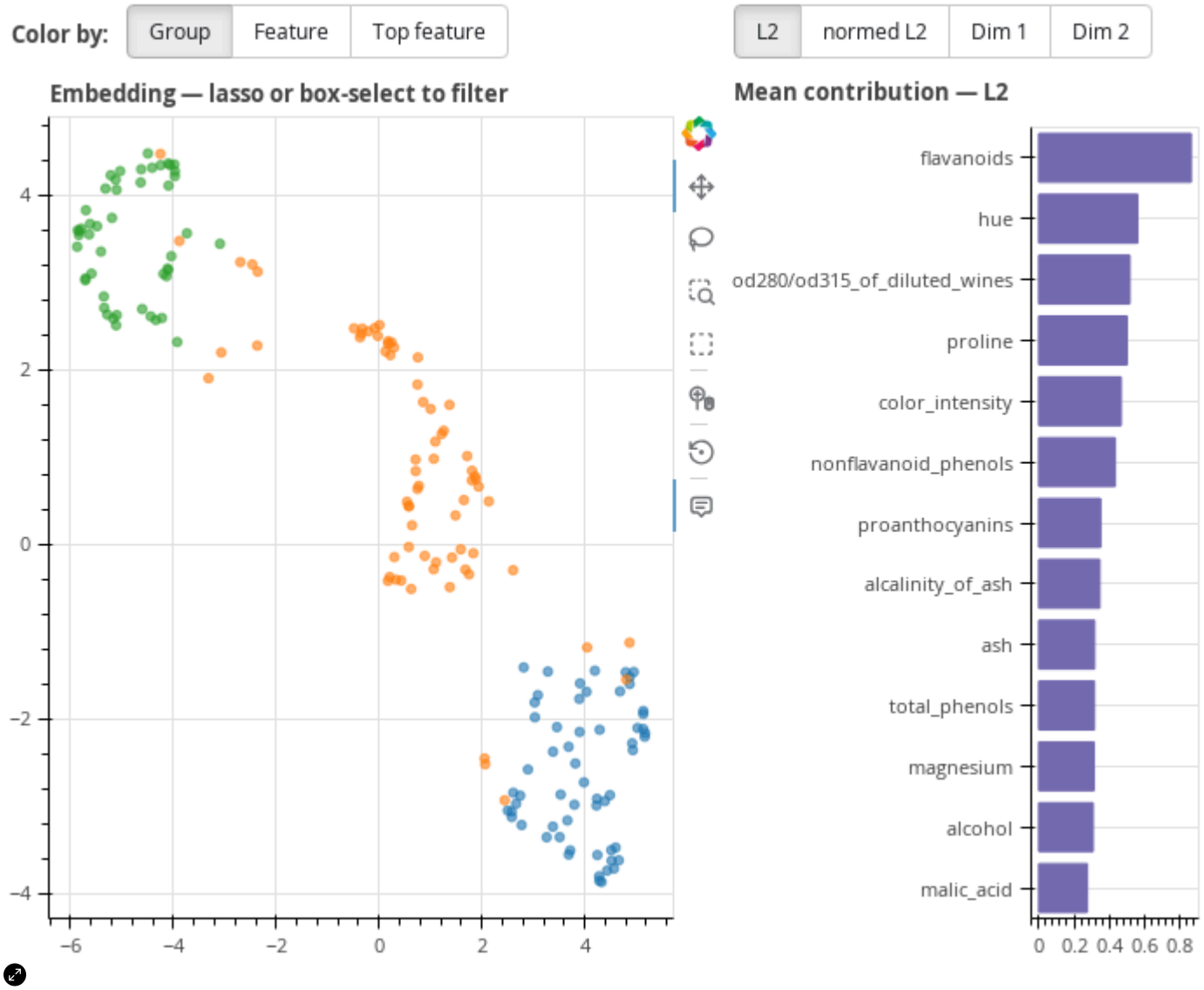

Figure 1. **Interactive exploration of feature contributions in a Glass Box UMAP embedding of the UCI Wine dataset.**

Points represent wines colored by cultivar. The linked bar chart summarizes which chemical features contribute most to the selected points' positions in the embedding. In this example, proline helps hold cultivar 0 together, flavonoids act as a signed polarity feature pushing cultivar 0 toward one end of the embedding and cultivar 2 toward the other, color intensity is prominent in cultivar 2, and ash explains local substructure within the rightmost portion of cultivar 0. The full walkthrough is available in the package documentation.

AI usage

We used Claude (Opus 4.6 and 4.7) to help write code, clean up code, comment our code, and write text that we edited.

Key takeaways

UMAP is a popular method for visualizing high-dimensional data in two dimensions, but it's historically been a black box: You can see clusters and patterns, but you don’t know why. Glass Box UMAP solves this by producing UMAP-like embeddings where every point's position can be broken down into exact contributions from each input feature. If you work with high-dimensional data and use UMAP to explore it, this package lets you go beyond "these points cluster together" to "these points cluster together *because of these specific features*." The theoretical foundations are described in our previous pub [6], and the package itself is designed as a near drop-in replacement for the standard UMAP Python library, so it's straightforward to integrate into existing workflows.

> Glass Box UMAP is and will always be open source.
> The **documentation** is hosted on Read the Docs and the **codebase** is hosted on GitHub (DOI: 10.5281/zenodo.20220755).

Contributors (A–Z)

- **James R. Golden**: Formal analysis, Software, Supervision
- **Evan Kiefl**: Formal analysis, Software, Visualization, Writing
- **Ryan York**: Editing, Supervision

References

1. McInnes L, Healy J, Melville J. (2020). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. https://arxiv.org/abs/1802.03426
2. Lundberg S, Lee S-I. (2017). A Unified Approach to Interpreting Model Predictions. https://doi.org/10.48550/arxiv.1705.07874
3. Ribeiro MT, Singh S, Guestrin C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. https://doi.org/10.48550/arxiv.1602.04938
4. Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D. (2016). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. https://doi.org/10.48550/arxiv.1610.02391

5. Chari T, Pachter L. (2021). The Specious Art of Single-Cell Genomics. https://doi.org/10.1101/2021.08.25.457696
6. Golden J, York R. (2025). From black box to glass box: Making UMAP interpretable with exact feature contributions. https://doi.org/10.57844/arcadia-tnr4-7n9h
7. Sainburg T, McInnes L, Gentner TQ. (2020). Parametric UMAP embeddings for representation and semi-supervised learning. https://doi.org/10.48550/arxiv.2009.12981
8. Wang S, Mohamed A-R, Caruana R, Bilmes J, Plilipose M, Richardson M, Geras K, Urban G, Aslan O. (2016). Analysis of deep neural networks with the extended data Jacobian matrix. https://proceedings.mlr.press/v48/wanga16.html
9. Mohan S, Kadkhodaie Z, Simoncelli EP, Fernandez-Granda C. (2019). Robust and interpretable blind image denoising via bias-free convolutional neural networks. https://doi.org/10.48550/arxiv.1906.05478
10. Elhage N, Nanda N, Olsson C, Henighan T, Joseph N, Mann B, Askell A, Bai Y, Chen A, Conerly T, DasSarma N, Drain D, Ganguli D, Hatfield-Dodds Z, Hernandez D, Jones A, Kernion J, Lovitt L, Ndousse K, Amodei D, Brown T, Clark J, Kaplan J, McCandlish S, Olah C. (2021). A Mathematical Framework for Transformer Circuits. https://transformer-circuits.pub/2021/framework/index.html
11. McInnes L, Healy J, Saul N, Großberger L. (2018). UMAP: Uniform Manifold Approximation and Projection. https://doi.org/10.21105/joss.00861
12. Buitinck L, Louppe G, Blondel M, Pedregosa F, Mueller A, Grisel O, Niculae V, Prettenhofer P, Gramfort A, Grobler J, Layton R, Vanderplas J, Joly A, Holt B, Varoquaux G. (2013). API design for machine learning software: experiences from the scikit-learn project. https://doi.org/10.48550/arxiv.1309.0238
13. Ansel J, Yang E, He H, Gimelshein N, Jain A, Voznesensky M, Bao B, Bell P, Berard D, Burovski E, Chauhan G, Chourdia A, Constable W, Desmaison A, DeVito Z, Ellison E, Feng W, Gong J, Gschwind M, Hirsh B, Huang S, Kalambarkar K, Kirsch L, Lazos M, Lezcano M, Liang Y, Liang J, Lu Y, Luk CK, Maher B, Pan Y, Puhrsch C, Reso M, Saroufim M, Siraichi MY, Suk H, Zhang S, Suo M, Tillet P, Zhao X, Wang E, Zhou K, Zou R, Wang X, Mathews A, Wen W, Chanan G, Wu P, Chintala S. (2024). PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. https://doi.org/10.1145/3620665.3640366
14. Falcon W, Borovec J, Wälchli A, Eggert N, Schock J, Jordan J, Skafte N, Ir1dXD , Bereznyuk V, Harris E, Murrell T, Yu P, Præsius S, Addair T, Zhong

J, Lipin D, Uchida S, Bapat S, Schröter H, Dayma B, Karnachev A, Kulkarni A, Komatsu S, Martin.B , SCHIRATTI J-B, Mary H, Byrne D, Eyzaguirre C, Cinjon , Bakhtin A. (2020). PyTorchLightning/pytorch-lightning: 0.7.6 release. https://doi.org/10.5281/zenodo.3828935