https://doi.org/10.2352/EI.2022.34.1.VDA-415

Nirmaan: Dataset Generation for Multiclass Scatterplot Studies

Allison Wong; Geico; Washington, D.C.
Alark Joshi; University of San Francisco; San Francisco, CA
Sophie Engle; University of San Francisco; San Francisco, CA

Abstract

This paper presents Nirmaan, an open-source web-based tool for generating synthetic datasets of multiclass blobs for use in research related to scatterplots. We demonstrate how to use Nirmaan to generate datasets in the context of a user study where users must determine the centers of each class, but this tool can be used to generate datasets for other scatterplot tasks as well.

Introduction

Reproducibility and replicability are often cited as hallmarks of good science [19]. As the visualization field continues to grow, we are increasingly seeing efforts to provide code, data, and experimental methodology, along with the manuscript to truly allow the community to reproduce and validate the presented findings [4, 14, 15, 24].

Researchers frequently use a combination of synthetic data along with real world data to evaluate the efficacy of their contribution [4, 11, 16, 18, 26, 27, 28]. The process of generating synthetic data can be tedious and time-consuming. Not only do you need to generate synthetic data with varying sizes and features, but you have to also consider the various parameters that can be controlled (and reported for replication).

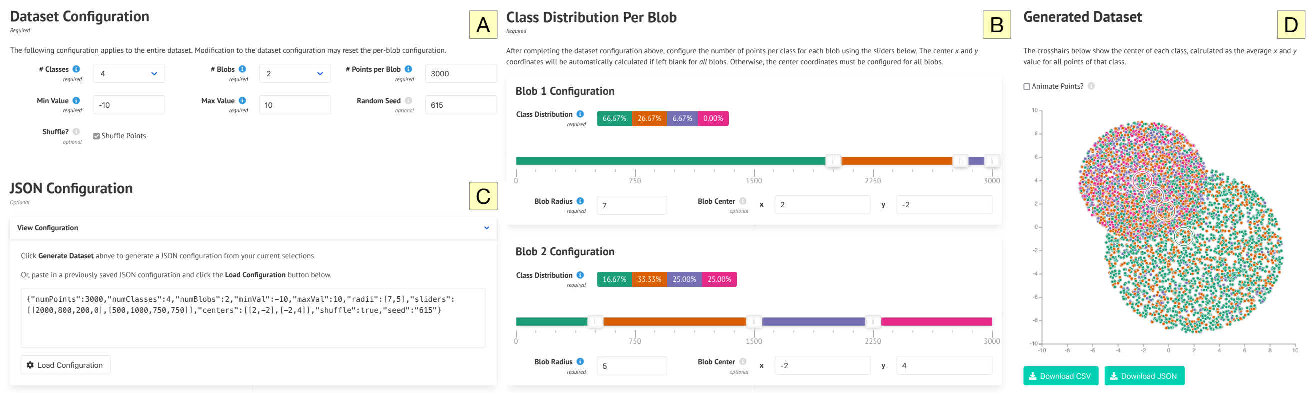
To alleviate the tedium of generating synthetic data (and reporting it), we present *Nirmaan*—a new tool that allows a user to generate multiclass datasets with control over the number of classes, number of blobs, points in a blob, the positions of the blobs, the radius of each blob, as well as the number of points in each class in the dataset. In this paper, we introduce the various features of Nirmaan that allow the generation of multiclass datasets for scatterplots-related research. Users can not only save their data in comma-separated value (CSV) files, but they can also save the JavaScript Object Notation (JSON) format of the dataset configuration for compact storage and sharing. Users can also vary the random seed to generate numerous similar datasets that have the same configuration (number of classes, blobs, number of points per blob, etc.), but different random point values. Having multiple similar datasets can be valuable when designing a study where you do not want participants to see the same datasets repeatedly.

Additionally, we demonstrate the use of Nirmaan for a completed user study where we investigated strategies to address overdraw in multiclass scatterplots. We later discuss other possible use cases for the tool that allows participants to generate datasets to test "numerosity," "object comparison," "find anomalies," and "density comparison" scatterplot visualization tasks [22].

Related Work

Multiclass scatterplots are popular in data visualization research due to the potential to impact the legibility of large, multiclass datasets with overlap. A combination of synthetic and real world datasets are used in the validation process. Kim and Heer [16] generate data with varying cardinality, category, and entropy to study the impact of data distribution on the efficacy of visual encodings.

Chen et al. [4] explored the use of animation to address challenges associated with exploring multiclass data visualized using scatterplot matrices. Wang et al. [26] investigated optimal color maps for perceptual separability for multiclass scatterplots. For their evaluation, they used a combination of real-world datasets and generated synthetic data with six classes. In recent work, Wei



Dataset Configuration
Required

The following configuration applies to the entire dataset. Modification to the dataset configuration may reset the per-blob configuration.

# Classes *required*	4	# Blobs *required*	2	# Points per Blob *required*	3000
Min Value *required*	-10	Max Value *required*	10	Random Seed *optional*	615
Shuffle? *optional*	☑ Shuffle Points				

JSON Configuration
Optional

Click **Generate Dataset** above to generate a JSON configuration from your current selections.

Or, paste in a previously saved JSON configuration and click the **Load Configuration** button below.

```
{"numPoints":3000,"numClasses":4,"numBlobs":2,"minVal":-10,"maxVal":10,"radii":[7,5],"sliders":
[[2000,800,200,0],[500,1000,750,750]],"centers":[[2,-2],[-2,4]],"shuffle":true,"seed":"615"}
```

Class Distribution Per Blob
Required

After completing the dataset configuration above, configure the number of points per class for each blob using the sliders below. The center *x* and *y* coordinates will be automatically calculated if left blank for *all* blobs. Otherwise, the center coordinates must be configured for all blobs.

Blob 1 Configuration

Class Distribution *required*: 66.67% | 26.67% | 6.67% | 0.00%

0 750 1500 2250 3000

Blob Radius *required*: 7 — Blob Center *optional*: x 2, y -2

Blob 2 Configuration

Class Distribution *required*: 16.67% | 33.33% | 25.00% | 25.00%

0 750 1500 2250 3000

Blob Radius *required*: 5 — Blob Center *optional*: x -2, y 4

Generated Dataset

The crosshairs below show the center of each class, calculated as the average *x* and *y* value for all points of that class.

Figure 1. *Screenshots of the **Nirmaan** web interface at **vgl.cs.usfca.edu/nirmaan**. **A (top left):** The interface to configure the parameters that apply to the entire dataset. These are the first parameters configured by the user. **B (middle):** The interface to configure per-blob parameters, including the class distribution, radius, and center. **C (bottom left):** The interface to view the current configuration in JSON format or to load a saved JSON configuration. **D (right):** The dataset preview displayed after the dataset is generated. Users may opt to animate the points, download the CSV, and/or download the JSON configuration.*

et al. [27] examined the impact of geometric scaling on scatterplot interpretation. Heimerl et al. [13] explored various aggregation-based design choices to address problems associated with clutter and density when visualizing multiclass scatterplots. They used a combination of real-world data as well as synthetic data. Yuan et al. [28] used eight real-world multiclass data sets in their recent paper on evaluating sampling strategies when visualizing dense multiclass scatterplots. Lu et al. [17] presented the Winglets technique to communicate uncertainty in multiclass scatterplots. They used a within subjects design where they generated datasets with a number of clusters and varying overlap in the clusters.

Finally, Sedlmair et al. [23] presented a taxonomy of visual cluster separation factors. Our tool allows users to configure the count, size, density, and separation data characteristics, but not factors like the shape or mixture. Nirmaan is inspired by (but does not directly use) the `make-blobs` dataset generator in the `scikit-learn` Python library [21]. The name, pronounced *Nirmaan*, is inspired by the Hindi word for construction and building.

Tool Features

Nirmaan is an open-source web-based tool that allows users to generate synthetic datasets with one or more multi-class blobs. A "blob" is a set of randomly generated points in a circle around a central (x,y) coordinate within a given radius. Visit **vgl.cs.usfca.edu/nirmaan** for a live demo and **github.com/usfvgl/nirmaan** for the source code.

Dataset Configuration

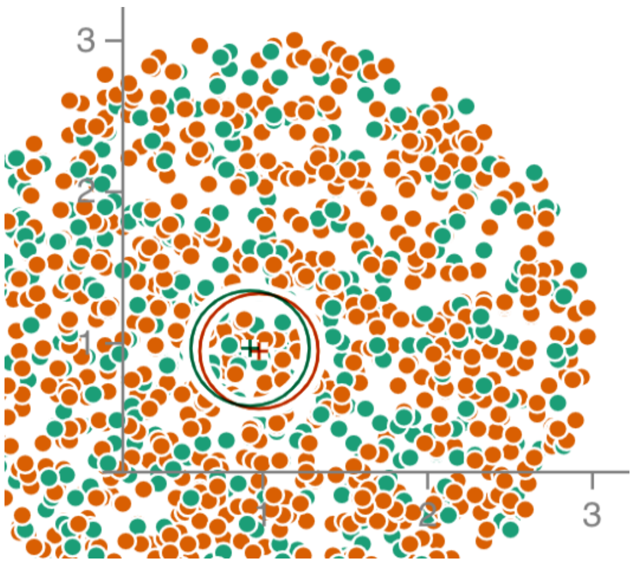
Users must first configure the parameters that apply to the entire dataset. See Figure 1 (A) for the interface. The required parameters include the number of classes to use, the number of blobs to generate, and the number of points per blob. Users must also specify a minimum and maximum value, which is used to determine the blob centers. Depending on the radius specified per blob later, it is possible that some point values fall outside this range (see Figure 2).

Users may optionally set a random seed to make the dataset generation repeatable. This value is used to seed the random number generator used to generate point positions and shuffle the dataset. Users may also optionally shuffle the generated dataset; otherwise the points are sorted by blob then class. Figure 3 shows the result of enabling shuffle in the tool.

Per Blob Configuration

Once the dataset parameters are configured, users may then configure the per-blob parameters. Each blob has its own configuration section based on the dataset configuration, as shown in Figure 1 (B). Each blob has sliders to configure the number of points per class within the blob. The class distribution percentages update automatically as the sliders are moved. Any individual class may be set to 0 points, so that none of the points within that blob are assigned that class. Users must also specify the blob radius, relative to value range configured for the dataset earlier.

Users may also optionally specify the blob center coordinates. The center must be within the value range for the dataset, but the points generated may be outside of that range depending on the per-blob radius (see Figure 2). Either all or none of the blob centers must be specified; the center coordinates will be automatically calculated unless the coordinates are provided for all blobs.



json: {"numPoints": 1200, "numClasses": 2, "numBlobs": 1, "**minVal**": 0.15, "**maxVal**": 9.85, "**radii**": [2], "sliders": [[400, 800]], "**centers**": [[1, 1]], "shuffle": true, "seed": "120"}

Figure 2. *Shows that while our tool requires the center of each blob to be in the user specified range of values, certain points can fall outside the range depending on the size of the radius.*

Generate and Preview Dataset

Once the dataset and per-blob configurations are complete, users may click on the "Generate Dataset" button. This will copy the current configuration into the "JSON Configuration" section, generate the dataset, and display a preview of that dataset as a scatterplot.

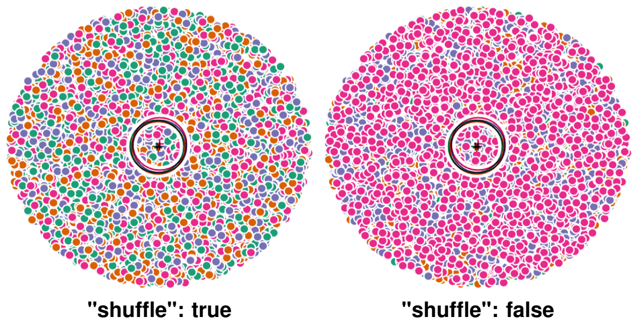
The scatterplot preview includes crosshairs for the centers of each class, which is calculated as the average x and y value for points of that class across all blobs. Users may also optionally animate the scatterplot preview, which continuously redraws the points. The preview is especially helpful early in the design phases of a user study; allowing quick iteration to determine the optimal parameters for a specific task.

If satisfied with the generated dataset, users can then click the "Download CSV" button to download the dataset as a CSV file. Users may optionally download the JSON configuration as well, which is useful to re-generate the dataset from the same set of parameters. See Figure 1 (D) for the interface.

JSON Configuration

When a dataset is generated, its configuration is copied in JavaScript Object Notation (JSON) format into the "JSON Configuration" text area. Users may click the "Download JSON" button to download this configuration to file. Alternatively, users may copy/paste a past configuration into this textarea and click the "Load Configuration" button. This will set all the parameters to the specified configuration and regenerate the dataset. See Figure 1 (C) for the interface.

This feature is useful for repeatable dataset generation, but also for iteration. Users can download both the dataset and config-



json: {"numPoints": 5000, "numClasses": 4, "numBlobs": 1, "minVal": 0.15, "maxVal": 9.85, "radii": [2], "sliders": [[1250, 1250, 1250, 1250]], "centers": [[5, 5]], "**shuffle**": **true**, "seed": "1200"}

Figure 3. *Demonstrates the shuffle feature with one blob and four classes. **Left:** The dataset with shuffle enabled. The four classes (depicted in pink, green, orange, and blue) are visible. **Right:** The dataset with shuffle disabled. Points from the last class drawn (pink) are more predominant. This could result in viewers interpreting the class distribution incorrectly.*

uration, and later decide to tweak a single setting by reloading the configuration and then editing the parameters. A JSON configuration can also be replicated multiple times with different random seeds to generate multiple datasets with near identical configurations. Figure 4 shows an example varying the random seed.

Implementation

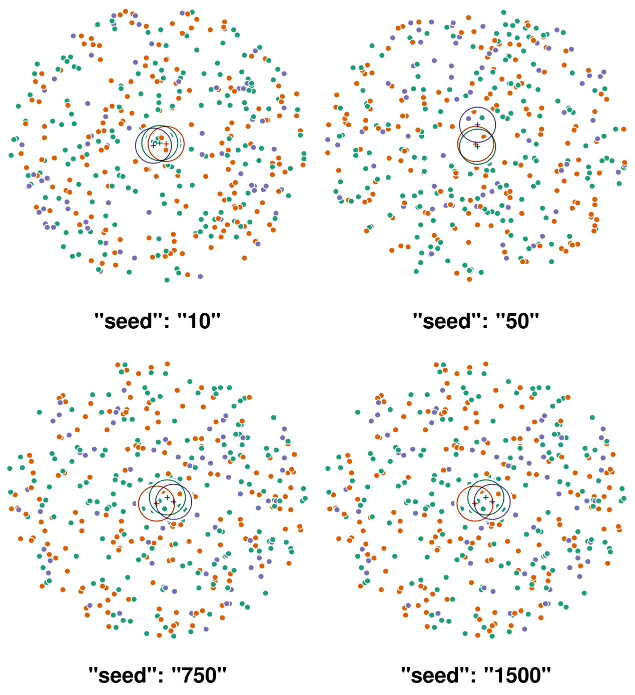
Nirmaan is responsive, mobile-friendly, and works in most modern web browsers. At its core, Nirmaan is implemented with HTML, CSS, JavaScript, and the jQuery JavaScript library [20]. Nirmaan utilizes custom JavaScript to generate datasets similar to the `make-blobs` dataset generator in the `scikit-learn` Python library [21]. The seedrandom.js JavaScript library [6] is used to generate random numbers in a repeatable way when a seed is specified.

The web-based front-end uses HTML, the open-source Bulma CSS framework [25] with the bulma-tooltip plugin [5] for tooltips, and the Font Awesome icon toolkit [7] for icons. The per-blob class distribution slider widgets use the noUiSlider JavaScript library [8] and the wNumb JavaScript library for formatting numbers [9]. The scatterplot preview is generated using the D3 version 4 [1, 2] JavaScript library. The widgets and built-in preview use the "Dark2" ColorBrewer scheme [3], which can be modified by changing the `colorScheme` variable in the JavaScript code.

Nirmaan both relies on open-source libraries and is itself open-source, allowing other researches to modify this tool according to their needs. For novel visualization techniques implemented in JavaScript, this may allow for more rapid iterative development of datasets than relying on other programming languages and tools for synthetic dataset generation. The tool is hosted via Github Pages [10]. See **github.com/usfvgl/nirmaan** for the source code.

Example Case Study

We tested Nirmaan by using it to generate synthetic datasets in a user study exploring overdraw in multiclass scatterplots. We used the Experimentr [12] framework to conduct the user study. See Figure 5 for screenshots.



json: {"numPoints": 400, "numClasses": 3, "numBlobs": 1, "minVal": 0.15, "maxVal": 9.85, "radii": [3], "sliders": [[167, 166, 67]], "centers": [[5, 5]], "shuffle": true, "**seed**": "1500"}

Figure 4. *Demonstrate the ability to generate similar datasets by only varying the random seed value. This feature can be used to generate multiple, similar synthetic datasets for user evaluation.*

Visualization Task

Our goal was to measure how well users are able to estimate the center of each class in a multiclass scatterplot with different levels of overdraw. The center of a class is defined as the average x and y value for all points belonging to that class.

We used synthetic datasets with three blobs for this task, where a "blob" is a set of randomly generated points in a circle around a central (x, y) coordinate within a given radius. The blobs have fixed locations arranged in a triangle, so that all class centers would lie within this triangle. Each blob had the same number of points and equal radius. See Figure 5 for an example.

Study Design

The study started with an information and consent page, collected optional user demographics, and asked users their familiarity with multiclass scatterplots. Users had to be 18 years or older, and could not have a form of color blindness nor photosensitive epilepsy to participate.

Next, the study included four questions to train the user on the visualization task. Users were asked to drag four slightly transparent filled circles from the top of the plot to where they felt the center was for the class with the corresponding color. See Figure 5 (left) for an example of the initial interface.

After users dragged each of these circles to the plot area, the "Click for real answer" button was enabled. Clicking this button displayed the real centers of each class, allowing users to see how

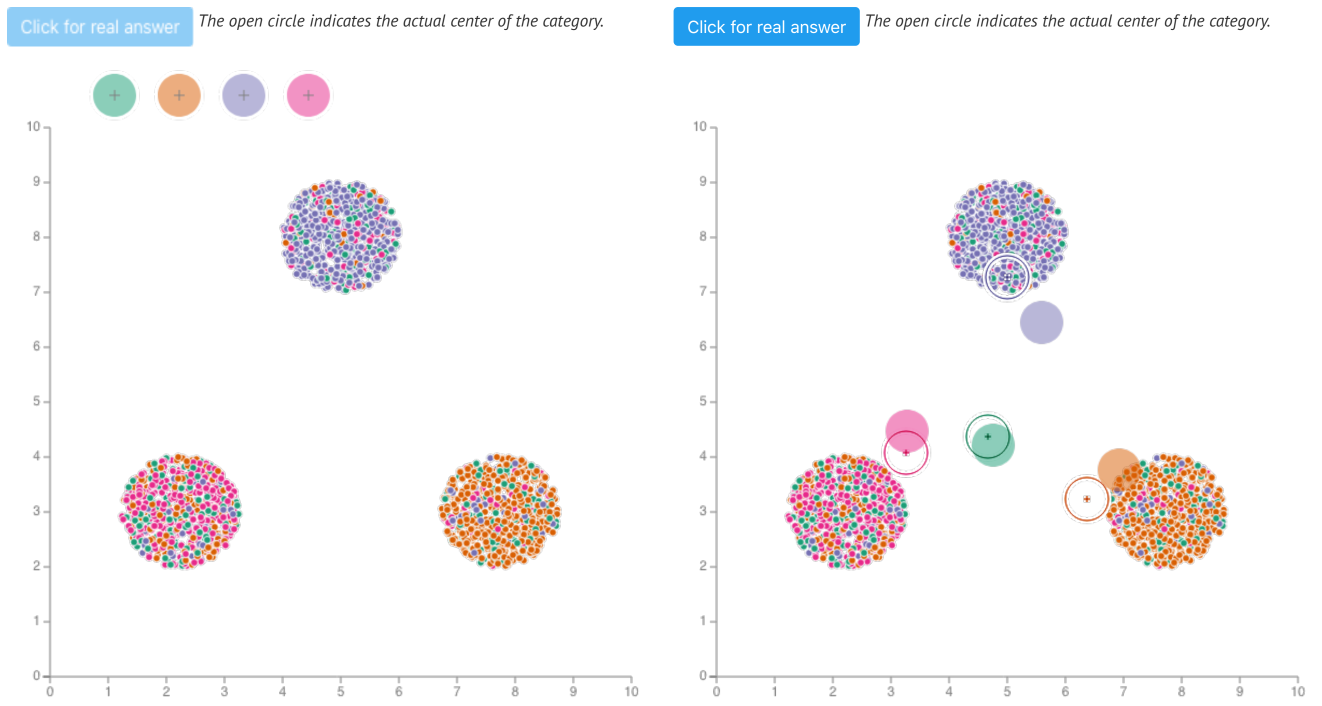

Click for real answer
The open circle indicates the actual center of the category.
Click for real answer
The open circle indicates the actual center of the category.
0
1
2
3
4
5
6
7
8
9
10

Figure 5. *Screenshots from the example user study.* ***Left:*** *An example training question in its initial state.* ***Right:*** *The same training question after the user has guessed the centers of each class and clicked the "Click for real answer" button.*

well they were performing on the training questions. See Figure 5 (right) for an example. Users had to click this button before being able to continue to the next question.

After a break, we asked the user to perform the visualization task four times followed by an engagement check and additional break. We then asked the user to perform the visualization task another four times after the break, followed by debrief questions at the end. Except for the engagement check, the order users saw a specific dataset was randomized.

Dataset Generation

We tested two overdraw conditions: (1) a "sparse" condition with less overdraw having 350 points per blob, and (2) a "dense" condition with more overdraw having 700 points per blob. The blobs for both conditions had the same radius and center positions.

For the training questions, we made sure users saw each condition twice in random order. Users could not continue until they attempted to find the centers for each class and clicked the "Click for real answer" button to see how well they performed. Here is an example Nirmaan JSON configuration for one of the "dense" training datasets, which is also depicted in Figure 5:

```
{"numPoints": 700, "numClasses": 4, "numBlobs": 3, "minVal": 0.15, "maxVal": 9.85, "radii": [1, 1, 1], "sliders": [[102, 95, 470, 33], [352, 145, 42, 161], [39, 105, 42, 514]], "centers": [[5, 8], [2.25, 3], [7.75, 3]], "shuffle": true, "seed": "217"}
```

For the non-training questions, we generated 4 dense datasets and 4 sparse datasets with various class distributions per blob. Here is an example sparse dataset used for a non-training question:

```
{"numPoints": 350, "numClasses": 4, "numBlobs": 3, "minVal": 0.15, "maxVal": 9.85, "radii": [1, 1, 1], "sliders": [[140, 70, 103, 37], [141, 79, 48, 82], [35, 66, 110, 139]], "centers": [[5, 8], [2.25, 3], [7.75, 3]], "shuffle": true, "seed": "529"}
```

For the engagement check inserted midway just before the break, we used a sparse dataset. We designed the dataset to be trivial to answer, with a distinct class in each blob making up over 70% of the points. The engagement check configuration was:

```
{"numPoints": 350, "numClasses": 4, "numBlobs": 3, "minVal": 0.15, "maxVal": 9.85, "radii": [1, 1, 1], "sliders": [[35, 235, 45, 35], [35, 35, 245, 35], [35, 35, 35, 245]], "centers": [[5, 8], [2.25, 3], [7.75, 3]], "shuffle": true, "seed": "217"}
```

We piloted the study twice, first with 11 users and then with 48 users. Based on the results and collected feedback, we are confident we can use Nirmaan and its generated datasets to conduct a larger user study measuring this visualization task to compare different techniques for alleviating overdraw.

Possible Use Cases

The utility of this tool goes beyond the case study presented. As discussed in the introduction and related work, several visualization user studies rely on synthetic datasets [4, 11, 16, 18, 26, 27, 28]. Nirmaan may be used to generate synthetic datasets for similar low-level visualization tasks on multiclass scatterplots. Consider the dataset depicted in Figure 1. This multiclass dataset was generated with the following configuration:

```
{"numPoints": 3000, "numClasses": 4, "numBlobs": 2, "minVal": −10, "maxVal": 10, "radii": [7, 5], "sliders": [[2000, 800, 200, 0], [500, 1000, 750, 750]], "centers": [[2, −2], [−2, 4]], "shuffle": true, "seed": "615"}
```

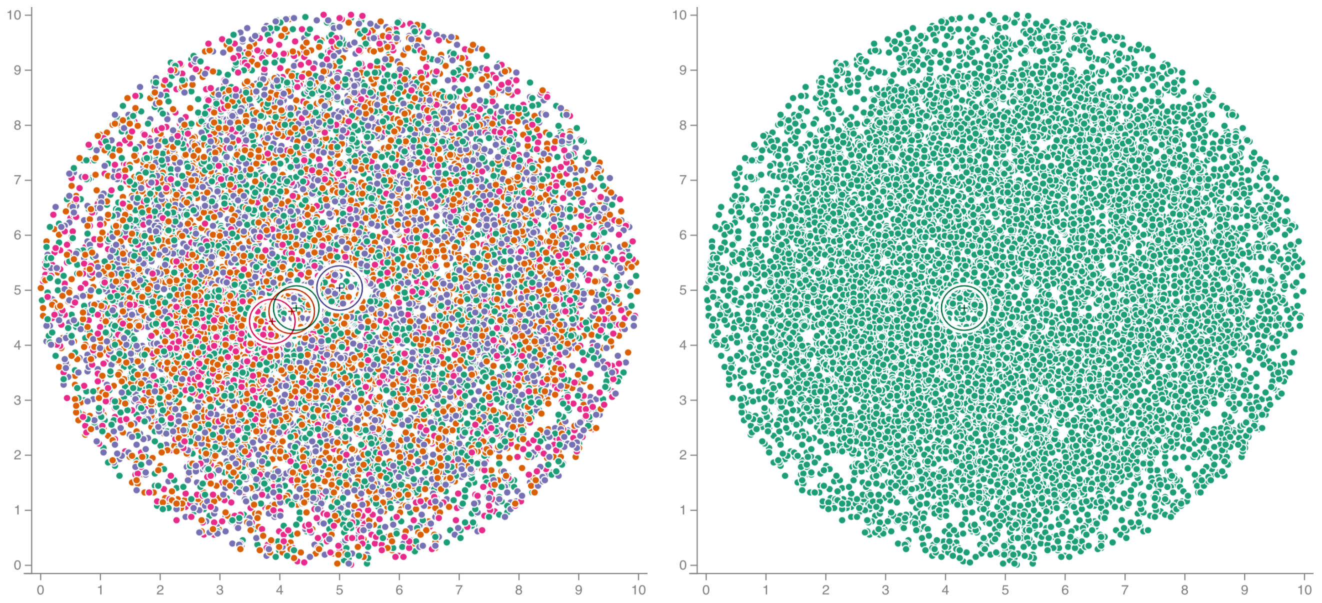


Figure 6. *Example datasets for density or diversity-related visualization tasks. See the "Possible Use Cases" section for the JSON configurations.* ***Left:*** *The dataset with four classes. The densest and most diverse region is centered at (3, 4).* ***Right:*** *The same dataset, except with only a single class. The densest region is again centered in the same position.*

This dataset and its variations may be used for several tasks that fall under the "numerosity comparisons" tasks identified by Sarikaya and Gleicher [22]. For example, we could ask users to identify the class with the most points (green), or the most prevalent class for each blob (or cluster).

We can also use Nirmaan to generate datasets for tasks related to identifying the densest the most diverse (in terms of classes) region in a scatterplot, similar to the tasks studied by Chen et al. [4]. Consider the following configuration, as depicted in Figure 6 (left):

```
{"numPoints": 6000, "numClasses": 4, "numBlobs": 3, "minVal":
–0.15, "maxVal": 10.15, "radii": [5, 4, 1], "sliders": [[1500, 1500,
1500, 1500], [2000, 2000, 2000, 0], [2000, 2000, 0, 2000]], "
centers": [[5, 5], [5, 5], [3, 4]], "shuffle": true, "seed": "532"}
```

Note that this example also works even if there is only one class, as depicted in Figure 6 (right):

```
{"numPoints": 6000, "numClasses": 1, "numBlobs": 3, "minVal":
–0.15, "maxVal": 10.15, "radii": [5, 4, 1], "sliders": [[6000, 0],
[6000, 0], [6000, 0]], "centers": [[5, 5], [5, 5], [3, 4]], "shuffle":
true, "seed": "532"}
```

In both cases, the region near the center of blob 3 will be the most dense. It will also be the most diverse for the multiclass variant.

It is also possible to generate datasets focused on outlier or anomaly detection. See Figure 7 for an example dataset and configuration where there are five blobs, but only one contains a point of the second class. We can raise the difficulty of this task by increasing the number of "distractor" classes present in each blob.

Conclusion

We present Nirmaan, an open-source web-based tool available at **vgl.cs.usfca.edu/nirmaan** for generating synthetic datasets with one or more multiclass blobs for use in scatterplot user studies. This tool allows users to specify the number of classes, blobs, and points per blob. Users may then configure the class distribution, radius, and center for each blob. A preview of the generated dataset is displayed, and users may opt to download both the dataset and the configuration JSON needed to regenerate the dataset.

We used Nirmaan to rapidly iterate different parameters and preview the resulting datasets to design a user study on overdraw in multiclass scatterplots. We were also able to save specific configurations as JSON, load those configurations, and tweak them as needed to generate different conditions for our study.

We also discussed how Nirmaan may be used to generate datasets for other visualization tasks as well. Nirmaan is open-source and the source code is available at **github.com/usfvgl/nirmaan** online. Other researchers may adapt this tool to meet their needs, including integrating other custom visualization techniques into the preview feature for rapid iterative development.

Acknowledgments

Work on this project was partially funded by the College of Arts and Sciences Faculty Development Fund (FDF) and Newmark Women in Technology Fund for Research at the University of San Francisco.

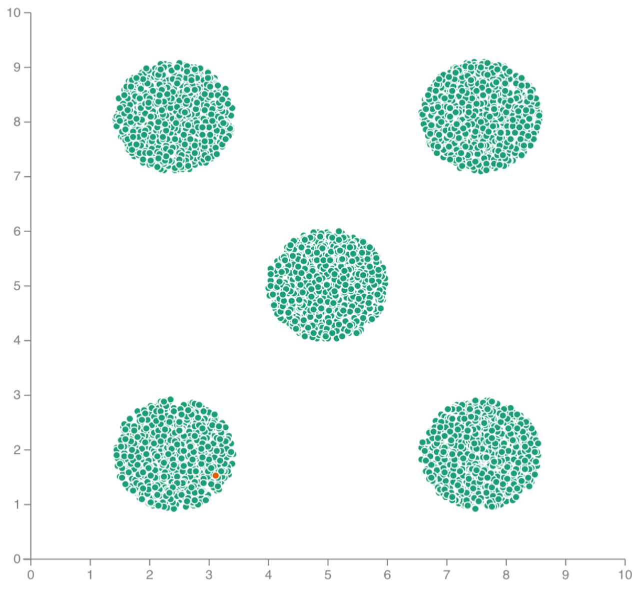
References

[1] Michael Bostock. D3: Data-Driven Documents. d3js.org, accessed 2021.

[2] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, December 2011.

[3] Cynthia A. Brewer. ColorBrewer v2.0. colorbrewer2.org, accessed 2021.

[4] Helen Chen, Sophie Engle, Alark Joshi, Eric D Ragan, Beste F Yuksel, and Lane Harrison. Using animation to alleviate overdraw in multiclass scatterplot matrices. In *Proceedings of the 2018 CHI*



json: {"numPoints": 1000, "numClasses": 2, "numBlobs": 5, "**minVal**": 0, "**maxVal**": 10, "**radii**": [1, 1, 1, 1, 1], "sliders": [[1000, 0], [1000, 0], [999, 1], [1000, 0], [1000, 0]], "**centers**": [[], [], [], [], []], "shuffle": false, "seed": "534"}

Figure 7. *Example dataset for outlier or anomaly detection tasks. Shuffling and displaying the centers are disabled to help highlight the outlier located in the lower left corner. Also demonstrates the automatic generation of centers when none are provided.*

Conference on Human Factors in Computing Systems, pages 1–12, 2018.

[5] CreativeBulma. bulma-tooltip. github.com/CreativeBulma/bulma-tooltip, accessed 2021.

[6] David Bau. seedrandom.js. github.com/davidbau/seedrandom, accessed 2021.

[7] Fonticons, Inc. Font Awesome Free. fontawesome.com, accessed 2021.

[8] Léon Gersen. noUiSlider. refreshless.com/nouislider, accessed 2021.

[9] Léon Gersen. wNumb.js. refreshless.com/wnumb, accessed 2021.

[10] GitHub, Inc. Github Pages. pages.github.com, accessed 2021.

[11] Michael Gleicher, Michael Correll, Christine Nothelfer, and Steven Franconeri. Perception of average value in multiclass scatterplots. *IEEE transactions on visualization and computer graphics*, 19(12):2316–2325, 2013.

[12] Lane Harrison. Experimentr: A hosting/data-collection backend and module-based frontend for web-based visualization studies, 2019.

[13] Florian Heimerl, Chih-Ching Chang, Alper Sarikaya, and Michael Gleicher. Visual designs for binned aggregation of multi-class scatterplots. *arXiv preprint arXiv:1810.02445*, 2018.

[14] Jaemin Jo, Frédéric Vernier, Pierre Dragicevic, and Jean-Daniel Fekete. A declarative rendering model for multiclass density maps. *IEEE transactions on visualization and computer graphics*, 25(1):470–480, 2018.

[15] Alex Kale, Matthew Kay, and Jessica Hullman. Visual reasoning strategies for effect size judgments and decisions. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

[16] Younghoon Kim and Jeffrey Heer. Assessing effects of task and data distribution on the effectiveness of visual encodings. *Computer Graphics Forum*, 37(3):157–167, 2018.

[17] Min Lu, Shuaiqi Wang, Joel Lanir, Noa Fish, Yang Yue, Daniel Cohen-Or, and Hui Huang. Winglets: Visualizing association with uncertainty in multi-class scatterplots. *IEEE transactions on visualization and computer graphics*, 26(1):770–779, 2019.

[18] Luana Micallef, Gregorio Palmas, Antti Oulasvirta, and Tino Weinkauf. Towards perceptual optimization of the visual design of scatterplots. *IEEE transactions on visualization and computer graphics*, 23(6):1588–1599, 2017.

[19] National Academies of Sciences, Engineering, and Medicine and others. *Reproducibility and replicability in science*. National Academies Press, 2019.

[20] OpenJS Foundation. jQuery. jquery.com, accessed 2021.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[22] Alper Sarikaya and Michael Gleicher. Scatterplots: Tasks, data, and designs. *IEEE transactions on visualization and computer graphics*, 24(1):402–412, 2017.

[23] M. Sedlmair, A. Tatu, T. Munzner, and M. Tory. A taxonomy of visual cluster separation factors. *Computer Graphics Forum*, 31(3pt4):1335–1344, 2012.

[24] Danielle Albers Szafir. Modeling color difference for visualization design. *IEEE transactions on visualization and computer graphics*, 24(1):392–401, 2017.

[25] Jeremy Thomas. Bulma: The Modern CSS Framework. bulma.io, accessed 2021.

[26] Yunhai Wang, Xin Chen, Tong Ge, Chen Bao, Michael Sedlmair, Chi-Wing Fu, Oliver Deussen, and Baoquan Chen. Optimizing color assignment for perception of class separability in multiclass scatterplots. *IEEE transactions on visualization and computer graphics*, 25(1):820–829, 2018.

[27] Yating Wei, Honghui Mei, Ying Zhao, Shuyue Zhou, Bingru Lin, Haojing Jiang, and Wei Chen. Evaluating perceptual bias during geometric scaling of scatterplots. *IEEE transactions on visualization and computer graphics*, 26(1):321–331, 2019.

[28] Jun Yuan, Shouxing Xiang, Jiazhi Xia, Lingyun Yu, and Shixia Liu. Evaluation of sampling methods for scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 2020.

Author Biography

Allison Wong received her BS in Managerial Economics from the University of California, Davis and MS in Computer Science from the University of San Francisco. She now works at Geico.

Alark Joshi received his BS in Computer Science from the University of Pune, India, MS in Computer Science from the University of Minnesota, and PhD in Computer Science at the University of Maryland Baltimore County. He is now an Associate Professor and Department Chair of Computer Science at the University of San Francisco.

Sophie Engle received her BS in Computer Science from the University of Nebraska and PhD in Computer Science from the University of California, Davis. She is now an Associate Professor of Computer Science at the University of San Francisco.