Clustergrammer Documentation

Release 1.1.0

Nicolas F. Fernandez

Dec 21, 2020

Contents

The Clustergrammer project consists of web-based tools for visualizing and analyzing high-dimensional data as interactive and shareable hierarchically clustered heatmaps (see *Introduction to Clustergrams*). Clustergrammer produces highly interactive visualizations that enable intuitive exploration of high-dimensional data and has several optional biology-specific features (e.g. enrichment analysis; see *Biology-Specific Features*) to facilitate the exploration of gene-level biological data. The project is free, open-source (all code is available on GitHub), and being actively developed at the Human Immune Monitoring Center and the Ma'ayan Lab at the Icahn School of Medicine at Mount Sinai.

CHAPTER 1

What's New

1.1 NCI Data Science Seminar Series

We recently presented *Clustergrammer2* at the NCI Data Science Seminar Seriesand showed off our latest examples (Citi Bike on Observable) and features (e.g. manual annotation of categories, integration with deck.gl).

1.2 Scipy 2020

We recently presented *Clustergrammer2* at SciPy 2020 Virtual Conference and showed off new examples (Citi Bike, 10X Genomics Visium) and features (e.g. manual annotation of categories).

1.3 Visium Spatial Transcriptomics Data from 10X Genomics

We used *Clustergrammer2*, the plotting library bqplot, the Jupyter dashboard library voila, and the Jupyter notebook hosting service Binder to build an interactive data exploration dashboard for Visium data from the mouse brain from 10X Genomics (try dashboard: Visium-Clustergrammer2 Dashboard, code: https://github.com/ismms-himc/visium-clustergrammer2). This dashboard generates linked views of spatial tissue data and high-dimensional gene expression data - see GitHub repo https://github.com/ismms-himc/visium-clustergrammer2 for more information.

1.4 Single Cell Immune Profiling of Atherosclerotic Plaques

Our collaborators in the Giannarelli Lab used single-cell proteomics and transcriptomics to investigate the immune landscape of atherosclerotic plaques (Fernandez et al.) and identify features of T cells and macrophages that were associated with clinical symptomatic disease state (see *Single-cell Gene Expression and Proteomics from Human Atherosclerotic Plaques*). We used Clustergrammer2 to analyze scRNA-seq and CITE-seq data as well as infer cell-cell communication pathways.

1.5 Clustergrammer2

Clustergrammer2 is a WebGL (specifically regl) visualization tool that enables researchers to easily visualize and explore large high-dimensional single-cell datasets (e.g. scRNA-seq data) without the need for traditional dimensionality reduction methods (e.g. t-SNE or UMAP). Please see the tutorial video above and *Case Studies and Tutorials* for more information.

1.6 Case Studies and Examples

Clustergrammer was developed to visualize high-dimensional biological data (e.g. genome-wide expression data), but it can also generally be applied to any high-dimensional data. Please refer to the *Case Studies and Tutorials* and links below for more information:

- *Single-cell Gene Expression and Proteomics from Human Atherosclerotic Plaques*
- *scRNA-seq Gene Expression 2,700 PBMC*
- *CITE-seq 7,800 PBMC*
- *Cancer Cell Line Encyclopedia Gene Expression Data*
- Lung Cancer PTM and Gene Expression Regulation
- Single-Cell CyTOF Data
- Kinase Substrate Similarity Network
- MNIST Notebook
- Iris Flower Dataset
- USDA Nutrient Dataset

1.7 JupyterCon 2018

The Clustergrammer-Widget was presented at JupyterCon 2018.

1.8 Contact

Please contact Nicolas Fernandez (nicolas.fernandez@mssm.edu) and Avi Ma'ayan (avi.maayan@mssm.edu) for support, comments, and suggestions. Users can also visit the discussion forum Clustergrammer2-Gitter.

1.9 Citing Clustergrammer

Please consider supporting Clustergrammer by citing our publication:

Fernandez, N. F. et al. Clustergrammer, a web-based heatmap visualization and analysis tool for high-dimensional biological data. Sci. Data 4:170151 doi: 10.1038/sdata.2017.151 (2017).

1.10 Funding

Clustergrammer is being developed by the Ma'ayan Lab and the Human Immune Monitoring Center at the Icahn School of Medicine at Mount Sinai for the BD2K-LINCS DCIC and the KMC-IDG.

1.11 Contents:

1.11.1 Getting Started

Clustergrammer is a web-based tool for visualizing and analyzing high-dimensional data as interactive and shareable hierarchically clustered heatmaps (see *Introduction to Clustergrams*). This section will provide information on how to interact with the visualization and how to quickly visualize your own data using the *Clustergrammer-Web*, *Clustergrammer2*, and *Clustergrammer-Widget*.

See *Case Studies and Tutorials* for examples of how Clustergrammer can be used to explore and analyze real world data. For developers interested in building their own web page using Clustergrammer, please refer to the *Web-Development* section.

What's New

Scipy 2020

We recently presented *Clustergrammer2* at SciPy 2020 Virtual Conference and showed off new examples (Citi Bike, 10X Genomics Visium) and features (e.g. manual annotation of categories).

Visium Spatial Transcriptomics Data from 10X Genomics

We used *Clustergrammer2*, the plotting library bqplot, the Jupyter dashboard library voila, and the Jupyter notebook hosting service Binder to build an interactive data exploration dashboard for Visium data from the mouse brain from 10X Genomics (try dashboard: Visium-Clustergrammer2 Dashboard, code: https://github.com/ismms-himc/visium-clustergrammer2). This dashboard generates linked views of spatial tissue data and high-dimensional gene expression data - see GitHub repo https://github.com/ismms-himc/visium-clustergrammer2 for more information.

Using Clustergrammer

Clustergrammer can be used as a web application *Clustergrammer-Web*, a Jupyter Widget (*Clustergrammer2* and *Clustergrammer-Widget*) or stand alone JavaScript libraries (*Clustergrammer-GL* and *Clustergrammer-JS*). The newer *Clustergrammer2* and *Clustergrammer-GL* libraries are built to handle larger datasets such as scRNA-seq data.

This tutorial shows how Clustergrammer2 can be run on the cloud using MyBinder. For additional examples with real world data (e.g. scRNA-seq data), please see *Case Studies and Tutorials*.

Interacting with Clustergrammer Heatmaps

Clustergrammer produces highly interactive visualizations that enable intuitive exploration of high-dimensional data including:

- *Zooming and Panning*
- *Row and Column Reordering* (e.g. reorder based on sum)
- *Interactive Dendrogram*
- *Interactive Dimensionality Reduction* (e.g. filter rows based on variance)
- *Interactive Categories*
- *Cropping*
- *Row Searching*

Clustergrammer also has *Biology-Specific Features* for working with gene-level data including:

- Mouseover gene names and description look-up (using Harmonizome)
- Enrichment analysis to find biological information (e.g. up-stream transcription factors) specific to your set of genes (using Enrichr)

Clustergrammer Web-App

Users can easily generate an interactive and shareable heatmap visualization using the *Clustergrammer-Web* (see upload section screenshot below). Simply upload a tab-separated matrix file (see *Matrix Formats and Input/Output* for more information) at the homepage to be redirected to a permanent and shareable visualization of your data.

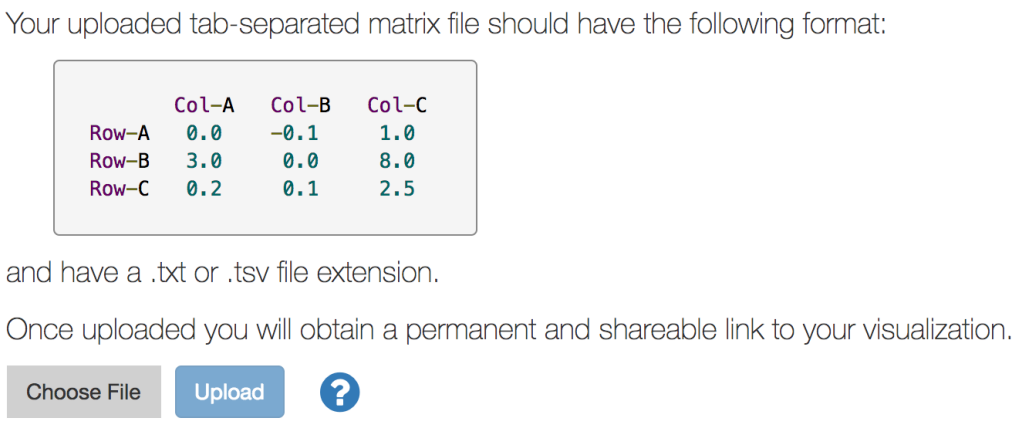
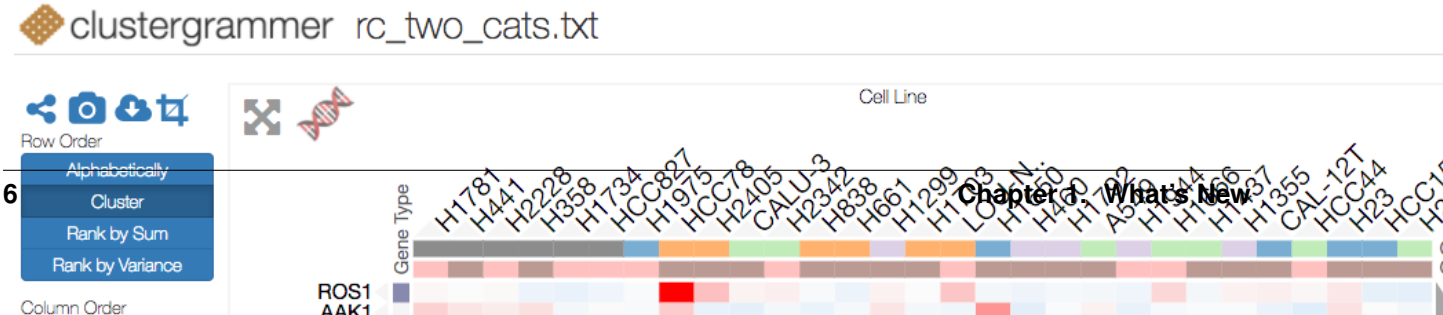


Fig. 1: Users can upload their data using the web app homepage. Simply choose your file and upload to be redirected to your permanent and shareable visualization.

Once you upload your data, the *Clustergrammer-Web* clusters your data and produces three views: a heatmap of your input matrix, a similarity matrix of your columns, and a similarity matrix of your rows. See the screenshots below and the example visualization for an example results page.

Heatmap **View**



tion.

Jupyter Widgets

Jupyter notebooks are ideal for generating reproducible workflows and analysis. They are also the best way to share Clustergrammer's interactive visualizations while providing context, analysis, and the underlying data to enable reproducibility (see *Sharing with nbviewer*).

Clustergrammer-Widget and *Clustergrammer2* enable users to easily produce interactive visualizations within a Jupyter notebook that can be shared with collaborators (using nbviewer). The *Clustergrammer-Widget* can be used to visualize a matrix of data from a matrix file or from a Pandas DataFrame (see *Matrix Formats and Input/Output* for more information).

and Tutorials for more information.

links to several case studies and examples using Clustergrammer to explore high-dimensional data. All examples are below are publically available through GitHub.

Visium Spatial Transcriptomics Data from 10X Genomics

for Visium data from the mouse brain from 10X Genomics (try dashboard: Visium-Clustergrammer2 Dashboard, code: https://github.com/ismms-himc/visium-clustergrammer2). This dashboard generates linked views of spatial tissue data and high-dimensional gene expression data - see GitHub repo https://github.com/ismms-himc/visium-clustergrammer2 for more information.

Single-cell Gene Expression and Proteomics from Human Atherosclerotic Plaques

Our collaborators in the Giannarelli Lab used single-cell proteomics and transcriptomics to investigate the immune landscape of atherosclerotic plaques (Fernandez et al.) and identify features of T cells and macrophages that were associated with clinical symptomatic disease state. We used Clustergrammer2 to analyze scRNA-seq and CITE-seq data as well as infer cell-cell communication pathways. Interactive notebooks can be found in the Giannarelli lab GitHub repo: Single-Cell-Immune-Profiling-of-Atherosclerotic-Plaques.

Fig. 4: Immune Profiling of Atherosclerotic Plaques Identifies

scRNA-seq Gene Expression 2,700 PBMC

Single cell RNA-seq (scRNA-seq) is a powerful method to interrogate gene expression across thousands of single cells. This method provides thousands of measurements (single cells) across thousands of dimensions (genes). Clustergrammer2 is used to interactively explore an example dataset of 2,700 PBMCs obtained from 10X Genomics. Bulk gene expression signatures of cell types obtained from CIBERSORT were used to obtain a tentative cell type for each cell. The data and code can be found on GitHub at clustergrammer2-notebooks and the notebook can be viewed and re-run on the cloud - see below.

CITE-seq 7,800 PBMC

CITE-seq (a.k.a feature barcoding from 10X genomics) is a new method that enabels researchers to simultaneously measure gene expression and protein levels in single cells. This notebook uses Clustergrammer2 to interactively explore an example dataset measuring the gene expression and surface marker proteins of 7,800 PBMCs obtained from 10X Genomics. Cell type was assigned based on unbiased hierarchical clustering of cells in surface marker space (ADTs) and transferred to cells in gene expression space. Please see the video tutorial above for more information.

Mouse Organogenesis Cell Atlas 2 Million Cells

Cao, J and Spielmann, M et al profiled gene expression from ~2 million mouse cells between 9.5 and 13.5 days of gestation. They identified 38 major cell types and measured ~25,000 genes. We generated a downsampled view of this data representing the ~1.3 million single cells (excluding ~600K suspected doublets) in the dataset by averaging expression for each cell type in each embryo, resulting in ~2,000 cell-type and embryo representative clusters. We demonstrate how Clustergrammer2 can be used to explore cell type clustering, find genes associated with cell type clusters, as well as identify genes that are differentially regulated across developmental stage. For more information, see the video tutorial above and launch or view the notebook using the badges.

CODEX Single Cell Multiplexed Imaging Dashboard

Goltsev et al used a highly multiplexed cytometric approach called CODEX to measure ~30 surface markers in spatially resolved single cells from mouse spleens. We utilized Clustergrammer2 to hierarchically cluster ~5,000 sinlge cells (from a subset of a segmented spleen image). We also used the Jupyter Widget bqplot to visualize single cell location data using voronoi plots. We then built a dasnboard using the library voila, which converts Jupyter notebooks to dashboards/web-apps, and linked our heatmap to the spatial map. This allows to interact with the Clustergrammer2 heatmap and highlight cells in the spatially resolved map. These kind of linked views are crucial for exploration of spatially resolved high-dimensional single cell data. Finally, we are running this dashboard using MyBinder. See CODEX Dashboard for code.

Cancer Cell Line Encyclopedia Gene Expression Data

The Cancer Cell Line Encyclopedia (CCLE) is a publicly available project that has characterized (e.g. genetic characterization) over 1,000 cancer cell lines. We used Clustergrammer to re-analyze and visualize CCLE's gene expression data in the CCLE Explorer. The CCLE Explorer allows users to explore the CCLE by tissue type and visualize the most commonly differentially expressed genes for each tissue type as an interactive heatmap. The CCLE Jupyter Notebook generates an overview of the CCLE gene expression data, investigates specific tissues, and explains how to use *Enrichrgram* to understand the biological functions of differentially expressed genes.

Lung Cancer Post-Translational Modification and Gene Expression Regulation

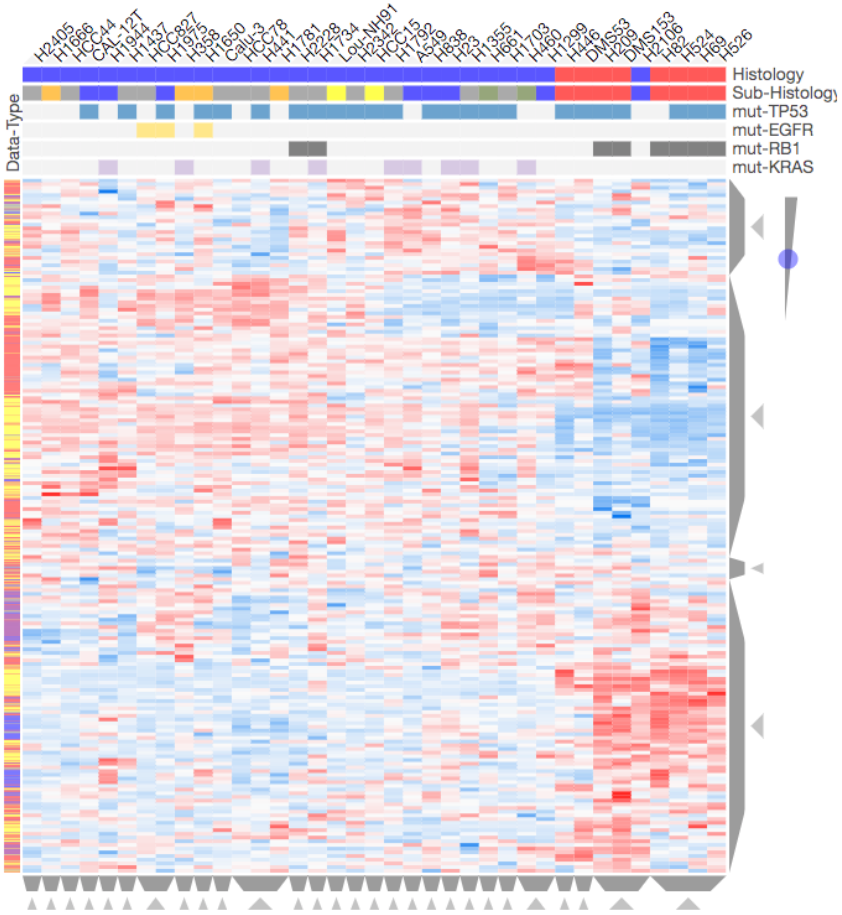
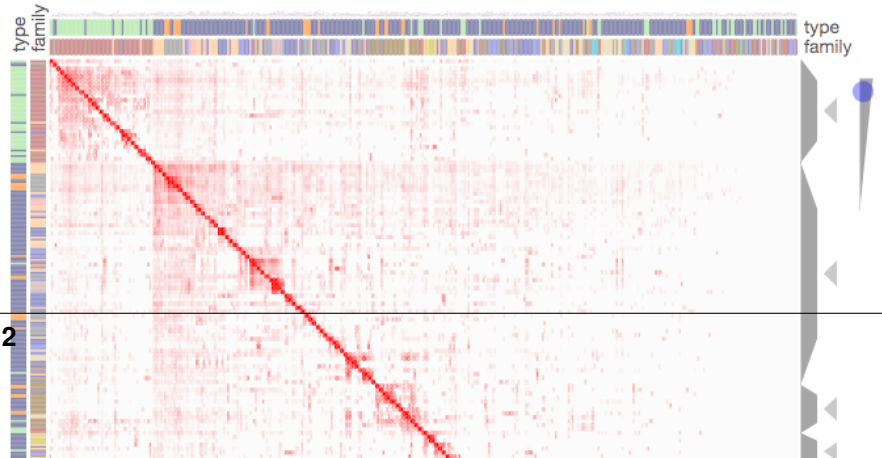


Fig. 5: Screenshot from the CST_Data_Viz.ipynb Jupyter notebook showing hierarchical clustering of differential phosphorylation, methylation, acetylation, and gene expression data across 37 lung cancer cell lines. See the interactive Jupyter notebook CST_Data_Viz.ipynb for more information.

Lung cancer is a complex disease that is known to be regulated at the post-translational modification (PTM) level, e.g. phosphorylation driven by kinases. Our collaborators at Cell Signaling Technology Inc used Tandem Mass Tag (TMT) mass spectrometry to measure differential phosphorylation, acetylation, and methylation in a panel of 42 lung cancer cell lines compared to non-cancerous lung tissue. Gene expression data from 37 of these lung cancer cell lines was also independently obtained from the publicly available Cancer Cell Line Encyclopedia (CCLE). In the Jupyter notebook CST_Data_Viz.ipynb we:

- Visualize PTM data, gene expression data, and merged PTM/gene-expression data
- Identify co-regulated clusters of PTMs/genes in distinct lung cancer cell line subtypes
- Perform enrichment analysis to understand the biological processes involved in PTM/expression clusters

Large Network: Kinase Substrate Similarity Network



Clustergrammer can be used to visualize large networks without the formation of 'hairballs'. In the Kinase Substrate Similarity Network example we use Clustergrammer to visualize a network kinases based on shared substrate that includes 404 kinases and 163,216 links. Ki-

nases are shown as rows and columns. For more information see the Kinase Substrate Similarity Network example.

Machine Learning and Miscellaneous Datasets

Clustergrammer was used to visualize several widely used machine learning Datasets and other miscellaneous Datasets:

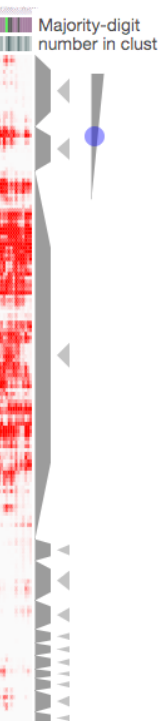
- MNIST Handwritten Digit Dataset
- Iris Flower Dataset
- USDA Nutrient Dataset

These examples demonstrate the generality of heatmap visualizations and enable users to interactively explore familiar Datasets.

1.11.3 Clustergrammer2

Clustergrammer2 is an interactive heatmap Jupyter widget built using the widget-ts-cookiecutter library and the WebGL library regl. Clustergrammer2 is built to help researchers interactively explore single cell data (e.g. scRNA-seq). Please see *Case Studies and Tutorials* for examples.

Single Cell Gene Expression 2,700 PBMC



Single cell RNA-seq (scRNA-seq) is a powerful method to interrogate gene expression across thousands of single cells. This method provides thousands of measurements (single cells) across thousands of dimensions (genes). Clustergrammer2 is used to interactively explore an example dataset of 2,700 PBMCs obtained from 10X Genomics. Bulk gene expression signatures of cell types obtained from CIBERSORT were used

to obtain a tentative cell type for each cell. The data and code can be found on GitHub at clustergrammer2-notebooks and the notebook can be viewed and re-run on the cloud - see below.

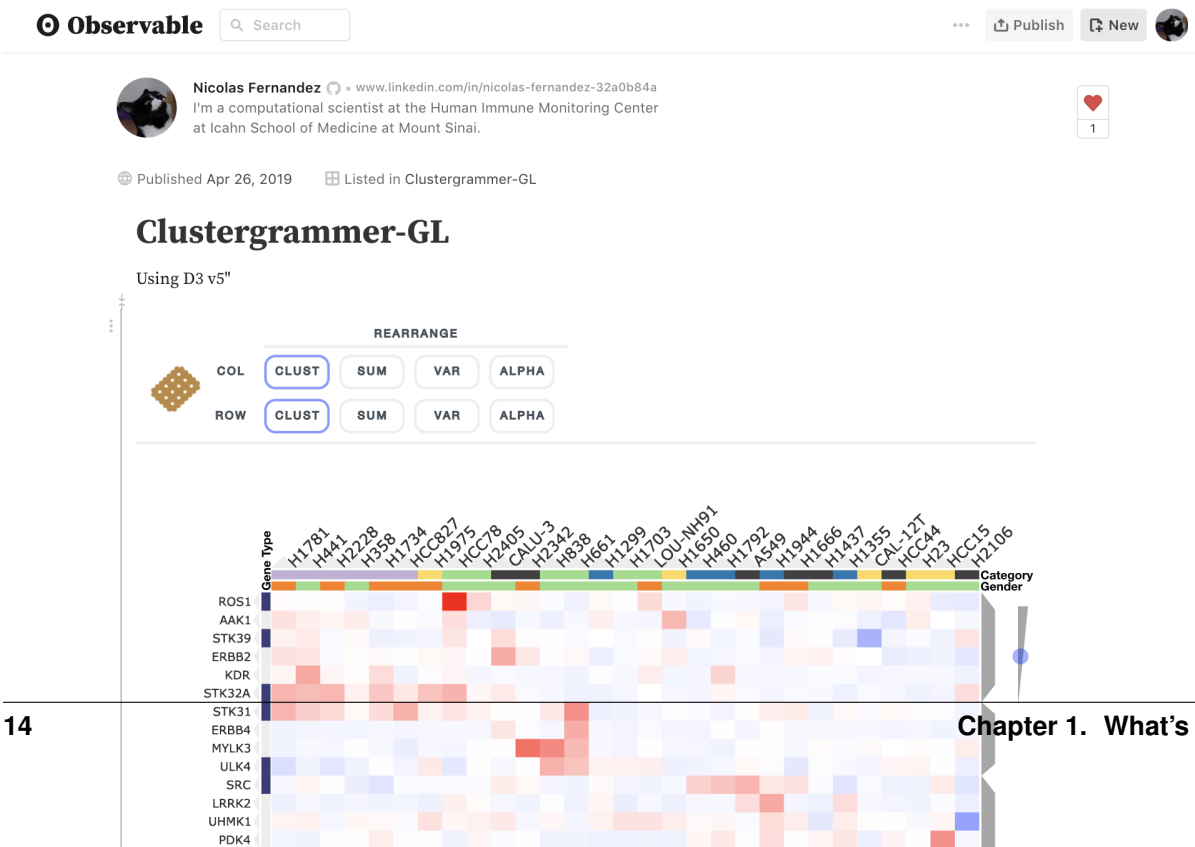
Clustergrammer2 Development

Clustergrammer2's source code can be found in the Clustergrammer2 GitHub repo. Clustergrammer2 is built using the jupyter-widgets framework (using the cookie cutter template).

Contact

Please *Contact* Nicolas Fernandez and Avi Ma'ayan with questions, use the GitHub issues feature to report an issue, or use the gitter discussion board to discuss the project.

1.11.4 Clustergrammer-GL



Clustergrammer-GL is the new WegGL front end JavaScript library. This new library can visualize much larger

datasets (matrices with ~millions of matrix-cells) and is being utilized by the new in-development Jupyter Widget, Clustergrammer2. Clustergrammer-GL can be used as a stand-alone visualization library and as well as the JavaScript notebook service Observable (see Clustergrammer-GL Observable Notebook)

Clustergrammer-GL is being built using the WebGL library regl, is free and open-source, and can be found on GitHub.

Check back soon for more updates.

1.11.5 Clustergrammer-Web

The Web App (http://amp.pharm.mssm.edu/clustergrammer) enables users to easily generate interactive and shareable heatmap visualizations by uploading their data as a tab-separated file.

Uploading Data using the Web App Homepage

Users can easily generate an interactive and shareable heatmap visualization using the *Clustergrammer-Web* (see upload section screenshot below). Simply upload a tab-separated matrix file (see *Matrix Formats and Input/Output* for more information) at the homepage to be redirected to a permanent and shareable visualization of your data.

Clustergrammer-Web Visualization

Your uploaded tab-separated matrix file should have the following for

	Col-A	Col-B	Col-C
Row-A	0.0	-0.1	1.0
Row-B	3.0	0.0	8.0
Row-C	0.2	0.1	2.5

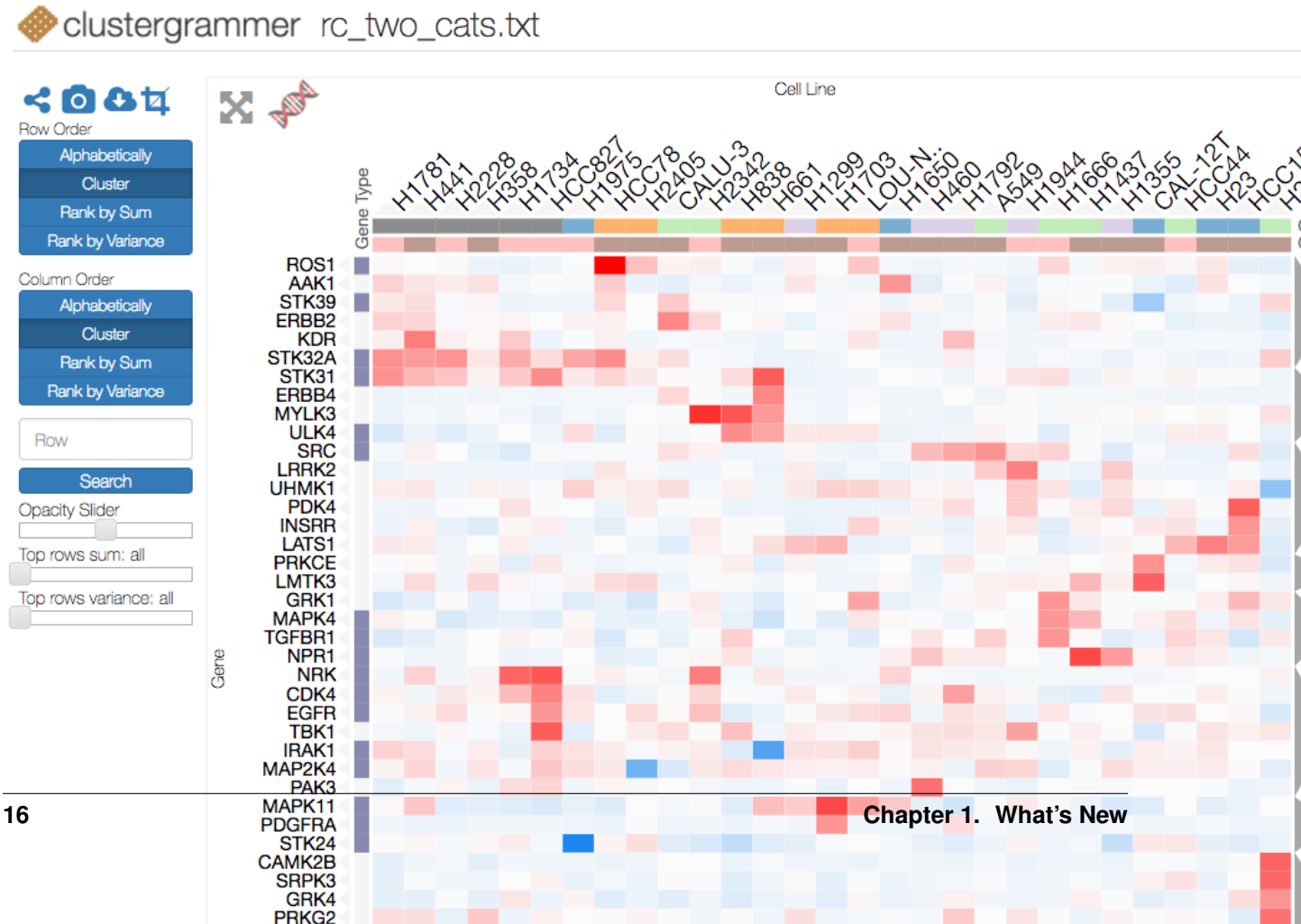
and have a .txt or .tsv file extension.

views of their data:

- Heatmap view of their matrix
- Similarity matrix of the columns in their original matrix
- Similarity matrix of the rows in their original matrix

See the screenshots below and the example visualization for an example Web App visualization page.

Heatmap **View**



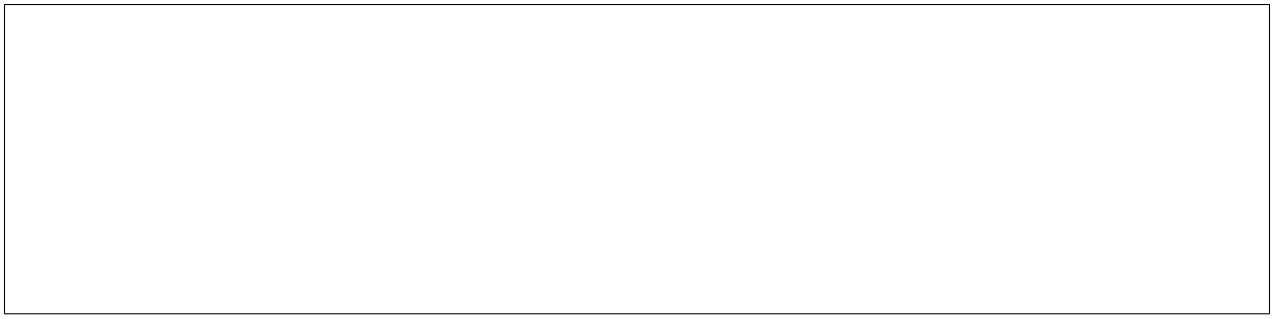
enables sharing with collaborators. See *Interacting with the Visualization* for more information.

Clustergrammer-Web API

iew in full page

e is an example

velopers who need to automatically generate visualizations for their own Web application.



ject:

(continues on next page)

(continued from previous page)

Clustergrammer-Web Development

the clustergrammer-web GitHub repo.

Data visualization benefits enormously from user interaction – particularly interactions that allow users to explore their data and interactively generate new views. Clustergrammer produces highly interactive heatmaps that enable users to intuitively explore their data and perform complex data transformations. Clustergrammer visualizations are built using the *Clustergrammer-JS* library and are consistent across the *Clustergrammer-Web* and the *Clustergrammer-Widget*. This section will overview heatmaps as a visualization tool and cover Clustergrammer's interactive features.

Clustergrammer2 Video Tutorial

This tutorial shows how Clustergrammer2 can be run on the cloud (using MyBinder) and some of Clustergrammer2's interactive features. For additional examples with real world data (e.g. scRNA-seq data), please see *Case Studies and Tutorials*.

Introduction to Clustergrams

Clustergrammer visualizes high-dimensional data as a hierarchically clustered matrix with colored matrix-cells (red for positive numbers and blue for negative numbers) and row/column labels. This type of visualization is commonly referred to as a heatmap or clustergram and this documentation uses these terms interchangeably; refer to Eisen et al., 1998 for an early example using biological data. Clustergrams also typically use dendrogram trees to depict the hierarchy of row and column clusters produced by hierarchical clustering.

Heatmaps are powerful visualization tools that enable users to directly visualize high-dimensional data without the loss of information and interpretability associated with dimensionality reduction techniques (e.g. t-SNE). For instance, columns can depict data-points (e.g. measured entities) and rows can depict data-dimensions (e.g. measured variables). In this way, heatmaps can visualize thousands of data-points in thousands of dimensions (e.g. data in thousand(s)-dimensional space). However, static heatmaps are of limited use for visualizing large datasets because visualization

elements and labels become too small to read. Furthermore, static heatmaps prevent users from interactively exploring their data, e.g. reordering rows/columns. We built Clustergrammer to address these problems and to extend the capabilities of heatmap visualizations.

Zooming and Panning

Clustergrammer allows users to zoom into and pan across their heatmap by scrolling and dragging. Double-clicking the heatmap resets zooming and panning. This is useful for working with large datasets where labels are not readable without zooming and for closely investigating regions of interest. Users can also increase the size of the visualization using the Expand button to hide the sidebar, see *Expanding* and, when in full-screen mode by adjusting the size of their window (see *Clustergrammer-Web Visualization* for information about full-screen mode).

Zooming and Panning Detailed Behavior

In general, zooming and panning occur in two stages. First zooming/panning occurs in the direction in which matrix-cells have been more compressed (e.g. if there are more rows than columns, then matrix-cells will be compressed in the vertical direction and the matrix-cells will be wide). Once zooming has decompressed matrix-cells (e.g. matrix-cells height and width are the same) then zooming/panning occurs in both directions. For instance, when visualizing a matrix with many more columns than rows zooming/panning will occur in the horizontal direction first until matrix-cells have equal width and height, then zooming/panning will be allowed in the vertical and horizontal directions. For symmetrical matrices, e.g. adjacency matrices, matrix-cells always have equal width and height and zooming/panning always occurs in both directions.

Large Matrix Zooming and Panning Behavior

Clustergrammer is capable of visualizing matrices with up to ~500,000 to ~750,000 matrix-cells, but is optimized to visualize matrices with more rows than columns – this has been done to accommodate datasets with many dimensions (rows) and few measurements (columns) that are common in biology. Clustergrammer uses front-end reversible row-downsampling to improve visualization performance for large matrices. If a user visualizes a matrix with a large number of rows (e.g. >1000-2000 rows) such that each matrix-cell is less than 1 pixel tall, then Clustergrammer will perform row downsampling. When zoomed out, the user will see a downsampled (e.g. coarse grained) version of their data. Zooming into the matrix will bring up successively less downsampled views until the original data is shown (e.g. when the original matrix-cells are >1 pixel tall). Clustergrammer will only display row labels when their font size is at a readable level (above ~5 pixels). Clustergrammer will also hide row/column labels while zooming into large matrices to improve zooming performance.

Mouseover Interactions

Mousing over elements in the heatmap (e.g. row names) brings up additional information using tooltips. For instance, mousing over matrix-cells brings up a tooltip with the row name, column name, and value of the matrix-cell (see below).

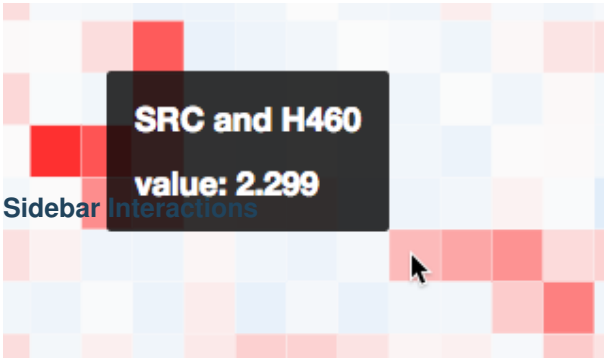


Fig. 11: Mousing over visualization elements (e.g. matrix cell) brings up additional information as a tooltip.

See *Clustergrammer-JS API* for information about adding callback functions to mouseover events and *Mouseover Gene Name and Description* for biology-specific mouseover behavior.

Clustergrammer visualizations have a sidebar section that contains the following interactive components:

- Optional About section (see *Clustergrammer-JS API*)

- Icon buttons: *share*, *snapshot*, *download*, *crop*
- *Row and Column Reordering Buttons*
- Row Search Box
- *Opacity Slider*
- *Row Filter Sliders*

Row and Column Reordering

Clustergrammer's sidebar reordering-buttons allows users to order rows and columns based on:

- sum or variance
- hierarchical clustering order
- label order

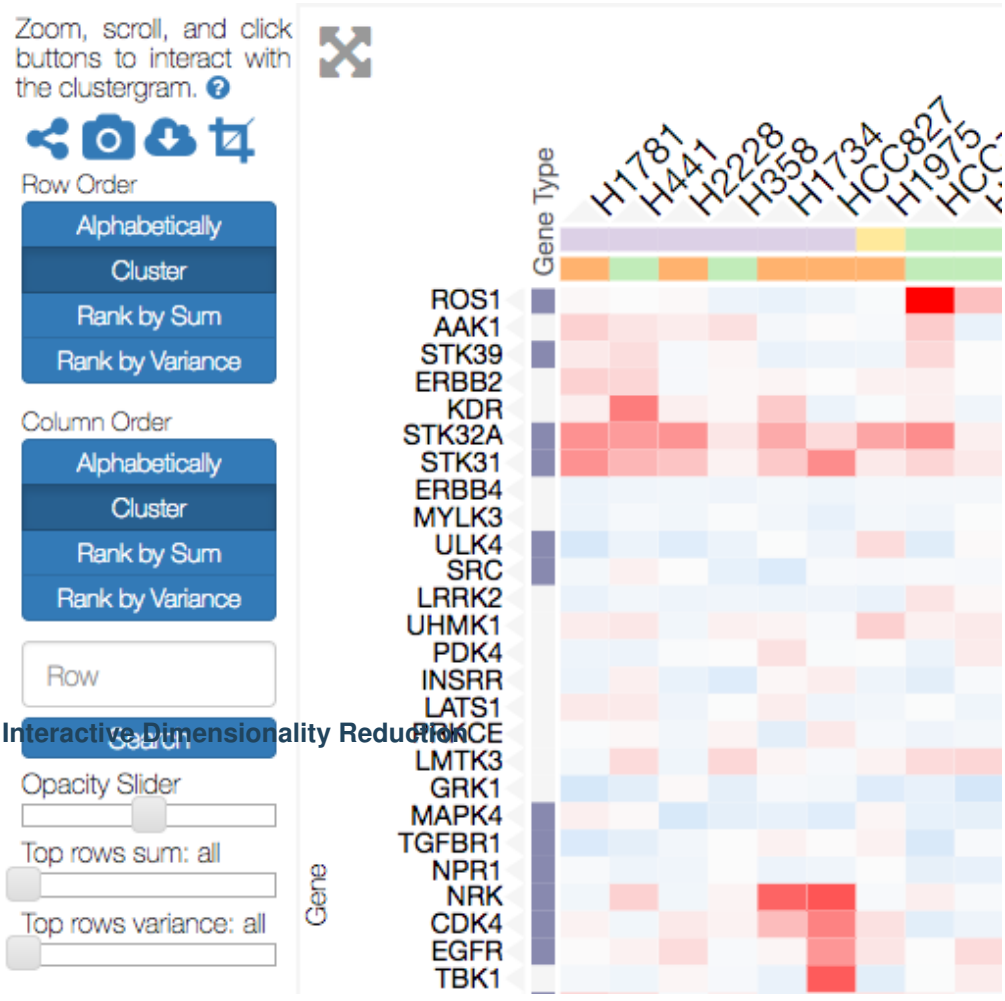


Fig. 12: The sidebar contains an optional About section and interaction elements (e.g. reordering buttons) and can be hidden by clicking the gray Expand buutton (and restored by clicking the Menu button).

This can be useful for identifying broad patterns in the data. Users can also reorder their matrix based on the values in a single row/column by double-clicking the row/column labels. Similarly, users can reorder based on categorical information by double-clicking category labels (see *Interactive Categories*). For small matrices reordering events are animated to help users visually track the results of this transformation.

Interactive Dimensionality Reduction

Dimensionality reduction is a useful data analysis technique (e.g. PCA , t-SNE) that is often used to reduce the dimensionality of high-dimensional datasets (e.g. hundreds to thousands of dimensions)

down to a number that can be easily be visualized (e.g. two or three dimensions). Heatmaps are capable of directly visualizing high-dimensional data, but can also benefit from dimensionality reduction.

Clustergrammer enables users to interactively perform dimensionality reduction by filtering rows based on sum or variance and instantaneously observe the effects of this transformation on clustering. Users can filter for the top rows based on sum or variance using the row-filter sliders in the sidebar and choose to show the top 500, 250, 100, 50, 20, and 10 rows. This can be useful for filtering out dimensions that are not of interest (e.g. dimensions with low absolute value sum) and determining the effect of these dimensions on clustering. For instance, we may see that columns cluster in largely the same manner when we filter out rows with low variance. Clustered views of the filtered matrices are pre-calculated by *Clustergrammer-PY*.

Fig. 13: The row filter sliders in the sidebar can be used to perform interactive dimensionality reduction. Here we are filtering for t

tion is animated to help the user visualize the effects this transformation. Clustergrammer employs the concept of object constancy by using animations to help the user visually follow changes to their data. Filtering out dimensions (rows) occurs in two steps: first filtered rows fade out, then the remaining rows rearrange themselves into their new positions (e.g. clustering order). Adding rows back also in occurs in two steps: the current rows rearrange themselves into their new positions, then the new rows fade into view.

Interactive Dendrogram

Clustergrams typically have dendrogram trees (for both rows and columns) to depict the hierarchy of row and column clusters produced by hierarchical clustering. The height of the branches in the dendrogram depict the distance between clusters. Clustergrammer depicts this hierarchical tree one slice at a time using trapezoids (see below). *Clustergrammer-PY* calculates hierarchical clustering using SciPy's hierarchy clustering functions (the default linkage type is set to average, see calc_clust.py) and saves ten slices of the dendrogram sampled evenly across the height of the tree.

Visualizing Dendrogram Clusters

Rather than visualize the dendrogram as a large branching tree, which uses a lot of visualization-space and is

difficult to interact with, Clustergrammer uses a more compact and easy to interact with visualization. Only a single slice of the dendrogram tree is visualized at a time as a set of non-overlapping adjacent clusters that are depicted using gray trapezoids (see screenshot below). Different slices of the dendrogram can be toggled using the dendrogram-sliders (blue circles that move along a gray triangle). Moving the slider up or down shows slices that are taken at higher or lower levels in the dendrogram tree, and thereby depicts larger or smaller clusters respectively. This interactive visualization allows users to identify clusters at different scales in their data.

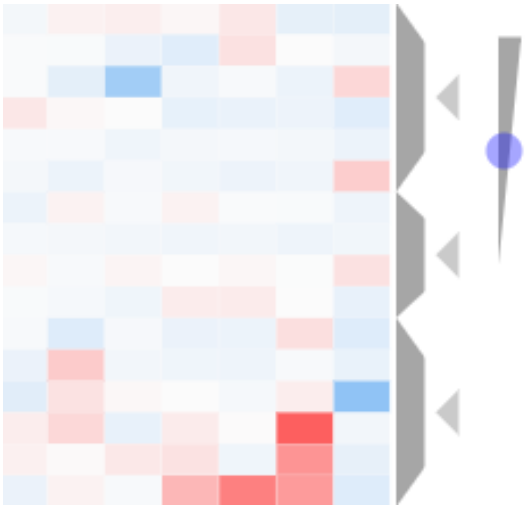


Fig. 14: A subset of the column dendrogram along with the dendrogram slider is shown above. The slider (blue circle and gray triangle) can be used to adjust dendrogram cluster sizes – move up for larger clusters and down for smaller clusters. Each dendrogram cluster has a Crop button (gray triangle) above it that can be used to filter the heatmap to show only this cluster.

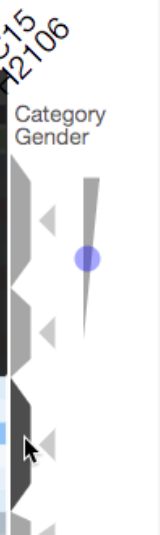
Interacting with Dendrogram Clusters

Dendrogram clusters are depicted as gray trapezoids, which are easy for a user to interact with (e.g. click). Mousing over a dendrogram cluster (gray trapezoid) highlights the current group of rows or columns (by adding a shadows over the rows or columns not in the cluster) and brings up a tooltip with cluster information (see screenshot below). If the rows or columns have categories, this tooltip will show a breakdown of the rows and columns into their categories, which can be useful for understanding how prior knowledge compares to clusters identified in a data-driven manner (e.g. we can ask whether columns with the same category cluster together based on the data). Clicking a dendrogram cluster brings up the same information in a pop-up window and also allows users to export the names of the rows or columns in the cluster. When a user visualizes biological gene-level data (row names must be genes), users have the option to export their clustered genes to the enrichment analysis tool, Enrichr (see *Biology-Specific Features* for more information).

Dendrogrm Cropping

Each dendrogram cluster has a small triangular crop button above it pointing towards the cluster (see the above images). Clicking the crop button filters out the rows or columns that not in the cluster, resizes the visualization to show the remaining data, and reverses the orientation of the crop button. Clicking on the outward facing crop button undoes the cropping and restores the full matrix. For small matrices, this transformation is animated. Dendrogram cropping can be useful for focusing in on a cluster of interest and when used in combination with *Enrichrgram* to investigate the biological functions specific to a cluster of genes (see *Biology-Specific Features* for more information).

Interactive Categories



of origin. Overlaying categories on our heatmap can help us understand the relationship between prior knowledge and the structures we find in our data (e.g. clusters). For instance, we may find that columns with the same category (e.g. the same tissue) cluster near each other based on the underlying data (e.g. gene expression) and we can conclude that the prior knowledge agrees with clusters identified in a data-driven manner. Similarly, we can explore how categories are re-distributed when the matrix is *reordered*. We can also use categories to overlay numerical information (e.g. the duration of drug treatment) and ask similar questions. Please refer to *Matrix Formats and Input/Output* for more information on how to encode categories.

column labels (see screenshot below). Categories can be of type *string* or *value* (see *Matrix Formats and Input/Output*): each *string*-type category has a different color, while each *value*-type category has a different opacity. The categories also have titles positioned adjacent to the category-cells.

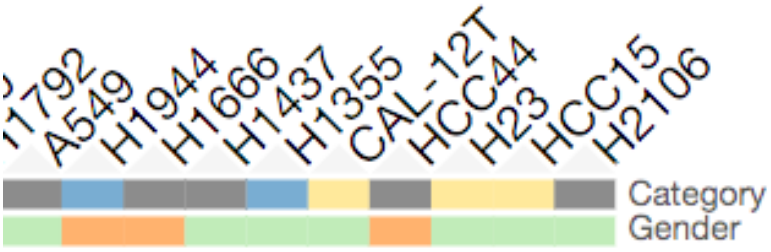
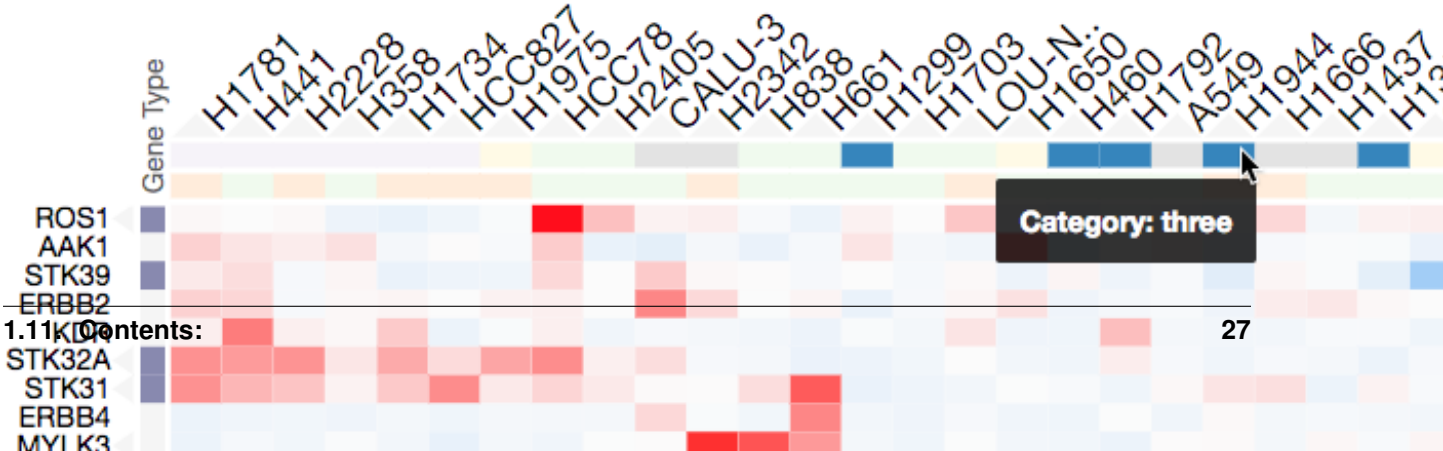


Fig. 16: A subset of column categories is shown above. In this example columns have two categories, 'Category' and 'Gender', which are depicted as colored cells under the column labels

Interacting with Categories

Mousing over a category will show the category name in a tooltip and highlight the instances of this category (while also dimming the instances of the other categories) to facilitate visualization of a specific category (see screenshot below). Double-clicking a category-title will reorder the matrix based on this category, which can be useful for getting an overview of all categories. Mousing over a dendrogram cluster will also show a breakdown of the rows/columns in a cluster based on their categories (see *Interactive Dendrogram*). Users can also reversibly filter a visualization to only show rows or columns of a particular category by clicking on a category while holding down the shift key (and undo this filtering by doing the same).



used by developers to add dynamically categories. This feature is used by *Enrichrgram* to visualize enrichment analysis results (see *Biology-Specific Features* for more information).

Cropping

The Brush-Cropping icon in the sidebar can be used to crop the matrix to a region of interest (see screenshot below). To crop, click the crop icon and then drag the cursor to define your region of interest. Once dragging has finished, the matrix will crop to show only the selected region of interest. Cropping can be undone by clicking the Undo button in the sidebar (which appears after cropping). This can be useful for focusing in on a small region of your overall matrix. Cropping can be used in combination with the *Download Icon* to export a small region of the matrix or in combination with *Enrichrgram* to perform enrichment analysis on a subset of clustered genes.

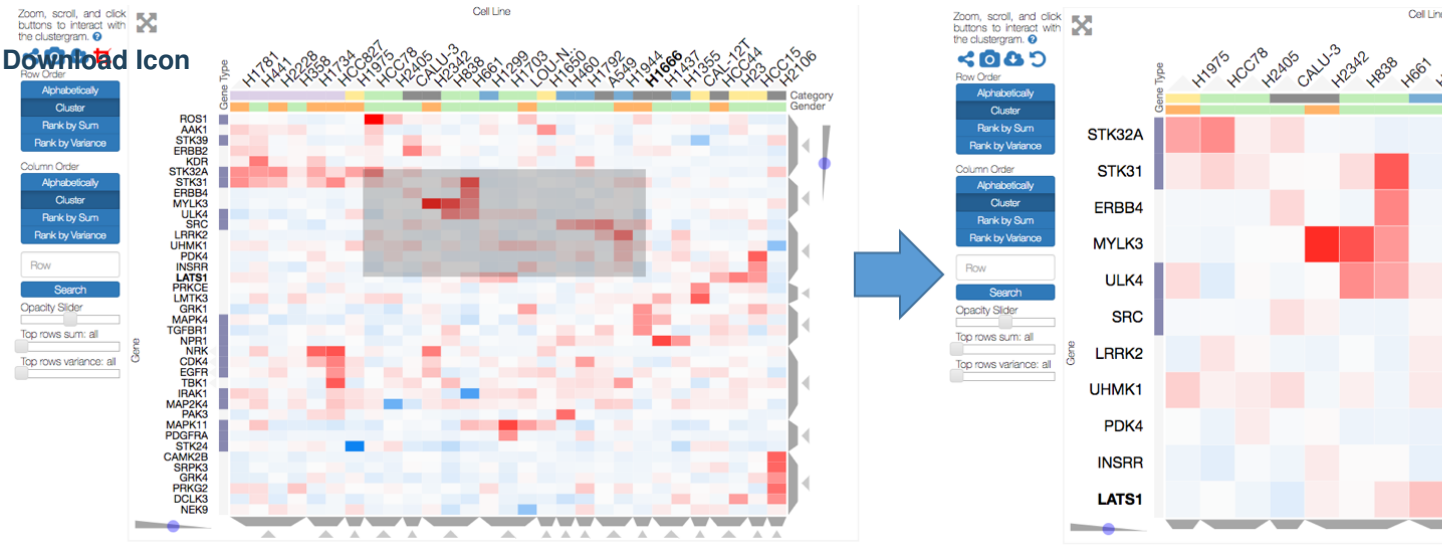
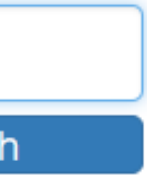


Fig. 18: The above example shows the result of brush-cropping into a section of the heatmap. To brush-crop, click the Crop button (th
the left panel) and drag/brush your cursor over your area of interest. To undo cropping, click the Undo button (circular arrow) on the rig

be a tedious task. To facilitate this common task, Clustergrammer's sidebar has a download icon, shown below, that allows users to download the matrix of data in the visualization. The downloaded data reflects the current state of the matrix; e.g. filtering, cropping, and reordering will be reflected in the downloaded data.

Download icon
download a tab-

e Snapshot icon
take a SVG or
the matrix in its
luding reorder-



earch for rows
in the sidebar.
the matrix will
ow.

ve heatmaps can
he current URL,
tained from the
sidebar.

Snapshot Icon

The Snapshot icon in the sidebar allows users to take a SVG or PNG snapshot of their visualization. This snapshot will reflect the current state of the visualization (e.g. reordering, etc) as well as zooming and panning.

Opacity Slider

The Opacity slider in the sidebar allows users to toggle the overall opacity levels of the heatmap. Moving the slider to the left reduces the opacity, while moving to the right increases the opacity. This can be useful for working with 'dim' matrices that can occur as a result of outlier values.

Row Searching

Users can search for rows in their matrix using the search box. Row search includes autocomplete and animated zooming into the matrix to display the row of interest.

Expanding

Users can hide the sidebar *Sidebar Interactions* panel using the Expand button at the top left of the matrix. Clicking the Menu button, when expanded, returns the sidebar.

Sharing your Interactive Heatmap

Interactive heatmaps produced with the *Clustergrammer-Web* and the *Clustergrammer-Widget* (when notebooks are rendered through nbviewer) can easily be shared with collaborators by sharing the URL of the visualization on the web app or the notebook. Users can also click the share button on the sidebar (see *Sidebar Interactions*) sidebar to get this shareable URL.

Biology-Specific Interactions

Clustergrammer has biology-specific features for working with gene-level data including:

- mouseover gene names and description look-up (using Harmonizome)
- enrichment analysis to find biological information (e.g. up-stream transcription factors) specific to your set of genes (using Enrichr)

See *Biology-Specific Features* for more information.

1.11.7 Biology-Specific Features

Clustergrammer was developed to visualize high-dimensional biological data (e.g. genome-wide expression data), but it can also generally be applied to any high-dimensional data. Clustergrammer has several biology-specific features that facilitate the analysis of gene-level biological data, such as: gene expression data, proteomics-data, etc. To take advantage of these features, row names must be official gene names. See the CCLE Explorer for examples of visualizing gene expression data. These optional biology-specific features are available in the

Clustergrammer-Web as well as the *Clustergrammer-Widget* and will automatically activate if the row-names are genes.

Mouseover Gene Name and Description

The human genome consists of over 20,000 genes and modern high-throughput measurements are capable of making measurements across the entire genome (e.g. genome-wide expression studies). Human genes have official gene symbols (e.g. *EGFR*) that are frequently used to label genes in these datasets. Since no biologist can be knowledgeable about every gene in the genome a common and repetitive task is looking up the names and descriptions of genes in a dataset or visualization. To streamline this activity, Clustergrammer automatically displays the full name and description of a gene (provided by data aggregated through the Harmonizome) as a tooltip when a user mouses over a gene label (see screenshot below). This simple feature speeds up analysis of large gene-level datasets.

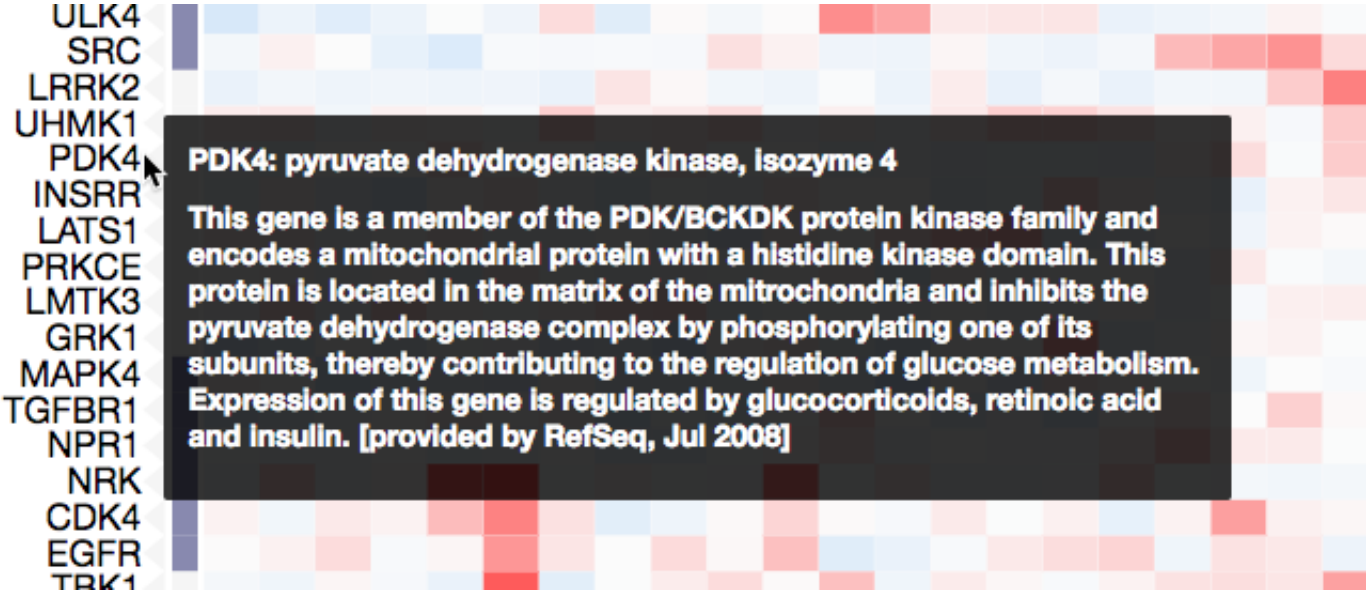


Fig. 23: Mousing over a gene name row brings up the full gene name and description (provided by data aggregated through the Harmonizome).

tain gene names and descriptions on mouseover events. hzome_functions.js is passed to *Clustergrammer-JS* as a callback function. See load_clustergram.js for an example use case. Mouseover callback functions can be used by developers to extend functionality for other domain-specific problems.

Enrichment Analysis

The field of biology has amassed an enormous amount of information about the genes in living organisms such as: function, disease-association, up-stream regulators, protein-level binding partners, etc. Integration of this information can help biologists understand patterns in their data. For instance, enrichment analysis is a popular method to identify biological information specific to a list of genes – e.g. a biologist can use enrichment analysis to identify up-stream regulatory transcription factors that specifically target their a set of up-regulated genes and thereby form hypotheses about potential up-stream regulators.

Export to Enrichr

When a user visualizes a matrix with genes as rows, Clustergrammer automatically enables integration with the enrichment analysis tool Enrichr. Users can export a set of clustered genes to Enrichr using the interactive dendrogram (see screenshot) or import enriched terms into the visualization using the *Enrichrgram* functionality.

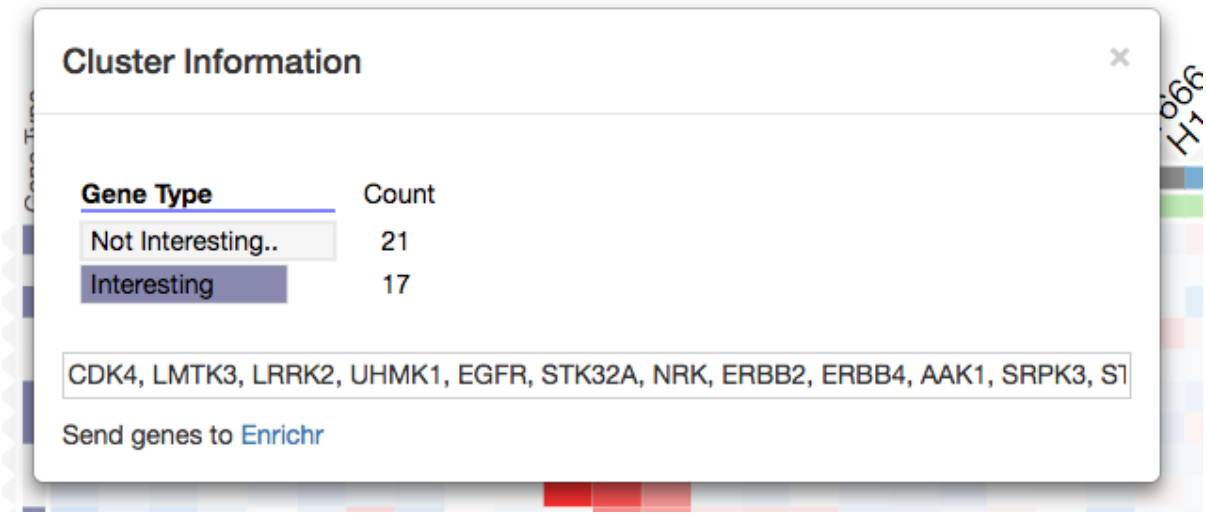


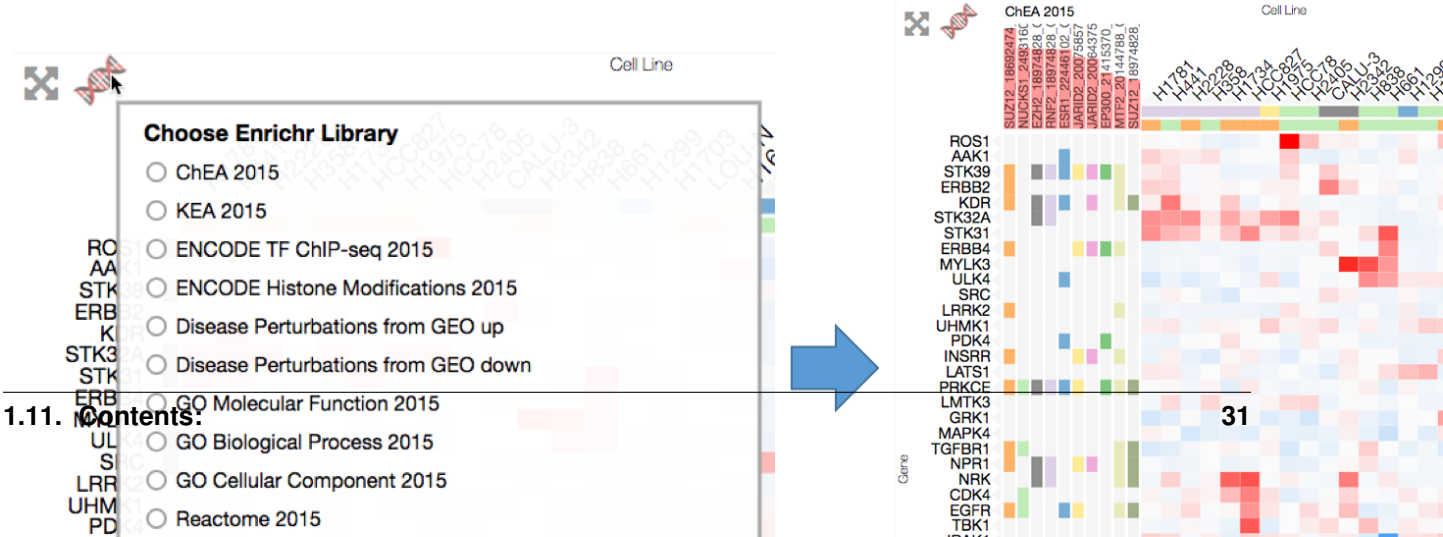
Fig. 24: Clicking a row dendrogram cluster opens a modal window with cluster information, row names, and a 'Send genes to Enrichr' link that allows users to export their gene list (e.g. cluster of row-genes) to Enrichr.

Enrichrgram

Enrichrgram enables users to find biological information specific to their genes of interest (using Enrichr) and import this information directly into the visualization as row categories (see screenshot below). Enrichrgram can be run on the front or back end (using the *Clustergrammer-PY API* to precalculate results). This feature enables enrichment analysis to be performed within the visualization itself by both the original author of the visualization and subsequent viewers.

Enrichrgram Front End

Enrichrgram on the front end is available to anyone viewing the visualization and can be used by simply clicking the red DNA-like Enrichr logo on the top left of the heatmap, which brings up a list of Enrichr libraries to choose from. To perform enrichment analysis, choose a library and Enrichrgram will return enriched terms from this library that are specifically associated with your list of genes (P-value bars indicate the degree of specificity). For instance, clicking on *ChEA 2016* will calculate enrichment for up-stream transcription factors. The enriched terms are shown as row categories, which enables users to see which genes are associated with each term. Row-category titles show the enriched term and the red-bars represent the significance of the enrichment (see Enrichr combined score). Users can run enrichment analysis on a specific cluster of genes by filtering the matrix to only show only their genes of interest. This filtering can be done using the Dendrogram Crop buttons (see *Interactive Dendrogram*) or Brush-Crop button (see *Cropping*) to select a subset of genes for analysis.



trix with enrichment results can be downloaded using the *Download Icon* button. Enrichment results can be permanently added to the visualization from the back end using the `enrichgram` method described below.

Enrichrgram Back End

To permanently add pre-calculated enrichment results to a visualization run the `enrichrgram` method described in the *Clustergrammer-PY API* before clustering. The Jupyter notebook Clustergrammer_CCLE_Notebook.ipynb demonstrates how to use the `enrichrgram` method to pre-calculate enrichment analysis results for a visualization.

The Enrichrgram.js library provides this functionality on the front end and works with the *Clustergrammer-JS* API to depict enriched terms and their associated genes as row categories. The update-row-category functionality can be extended by developers for other domain-specific problems.

1.11.8 Matrix Formats and Input/Output

Clustergrammer takes as input either:

- a tab-separated matrix file
- a Pandas DataFrame (using *Clustergrammer-PY*)

The tab-separated matrix file can take several formats shown below, which can include row/column categories and name/category titles. In all cases, row and column names must be unique (if input names are not unique then unique integers will be appended to names). Users are encouraged to arrange their matrix with data-points as columns and dimensions as rows, which enables users to take advantage of Clustergrammer's *Interactive Dimensionality Reduction*.

The front-end *Clustergrammer-JS* library can visualize matrices up to ~500,000 to ~1,000,000 cells in size and is also optimized to visualize matrices with more rows than columns – this has been done to accommodate datasets with many dimensions (rows) and few measurements (columns) that are common in biology. However very large matrices may take a long time to cluster using the *Clustergrammer-PY* library.

Simple Matrix Format

The simplest tab-separated file format is shown here:

```
      Col-A   Col-B   Col-C
Row-A 0.0     -0.1    1.0
Row-B 3.0     0.0     8.0
Row-C 0.2     0.1     2.5
```

The first row gives the column names and starts with a blank tab. All other rows start with the row name followed by the row data. Row and column titles can be added by prefixing each row or column name with `'Title:  '` (not shown in this example). See example_tsv.txt for an example of this matrix format.

Simple-Category Matrix Format

Row and column categories can be included in two ways. The first, simple-category format, is shown below:

		Cell Line: A549	Cell Line: H1299	Cell Line: H661
		Gender: Male	Gender: Female	Gender: Female
Gene: EGFR	Type: Interesting	-3.234	5.03	0.001
Gene: TP53	Type: Not Interesting	8.3	4.098	-12.2
Gene: IRAK1	Type: Not Interesting	7.23	3.01	0.88

Fig. 26: A matrix with row and column categories in 'simple' format.

categories as extra rows underneath column labels and row categories as an extra columns next to row labels. The above screenshot of an Excel spreadsheet shows a single row category being added as an additional column of strings (e.g. `Type:  Interesting`) and a single column category being added as an additional row of strings (e.g. `Gender:  Male`). Up to 15 categories can be added in a similar manner. Titles for row or column names or categories can be added by prefixing each string with `'Title:  '` (note the space after the colon). For example, the title of the column names is `Cell Line` and the title of the row categories is `Gender`. See rc_two_cats.txt for an example of this matrix format. Titles, if given, will be shown as labels above row/column names or adjacent to row/column categories.

Tuple-Category Matrix Format

Row/column names and categories can also be encoded as Python tuples as shown below:

```
        ('Cell Line: A549', 'Gender: Male')     ('Cell Line: H1299', 'Gender: Female
↪')  ('Cell Line: H661', 'Gender: Female')
('Gene: EGFR','Type: Interesting')      -3.234  5.03    0.001
('Gene: TP53','Type: Not Interesting')  8.3     4.098   -12.2
('Gene: IRAK','Type: Not Interesting')  7.23    3.01    0.88
```

The tuple-category format is easier to work with in Python and can be imported/exported into Pandas DataFrames and as tab-separated files. Note that titles have been added to row/column names and categories as discussed above. See tuple_cats.txt for an example of this matrix format.

Category Types: String and Value

Row and column categories can be of type: *string* or *value*. If categories are given as strings (e.g. containing letters), then categories will be depicted using colors. If categories are of type value (e.g. all categories contain only numbers), then value-categories will be depicted using color and opacity (gray for positive and orange for negative).

Value-based categories can be useful for adding data to your visualization (e.g. drug-dosage value) that you would like to compare to your other dimensions, but that should not influence your clustering. Both *value* and *string* categories can also be used to reorder your matrix by double-clicking their labels (see *Interactive Categories*).

Matrix File Examples

Several example tab-separated matrix files can be found in example matrix files.

Matrix Input/Output to Clustergrammer.py

Clustergrammer-PY can load a matrix directly from a file or from a Pandas DataFrame as well as export to a file or Pandas DataFrame (for more information see *Clustergrammer-PY API*):

```
# initialize Network object
from clustergrammer import Network
net = Network()

# load matrix from file or DataFrame
###################################

# load data from file
net.load_file('your_matrix.txt')

# load data from DataFrame, df
net.load_df(df)

# export matrix
###############

# write matrix to tab separated file
net.write_matrix_to_tsv(filename)

# export data to Pandas DataFrame
df_export = net.export_df()
```

1.11.9 Web-Development

Clustergrammer can be used by developers to add interactive heatmap visualizations to their web pages and web applications (see *App Integrations*).

Embedding Clustergrammer

The Clustergrammer web app can be used to produce visualizations that are embedded into another page using an IFrame; see below:

```
<iframe id="iframe_preview" src="https://amp.pharm.mssm.edu/clustergrammer/viz/
↪5734a7399fee36034aeb787e/rc_two_cats.txt" frameborder="0"></iframe>
```

Users can obtain a permanent link to their visualization by manually uploading their data using the Upload section of Clustergrammer-Web's homepage and copying the URL to the full-screen version of their visualization. Alternatively users can programmatically upload their data using the *Clustergrammer-Web API* and obtain their permanent links through the API.

Adding Clustergrammer to a Page

In addition to embedding a visualization hosted by the *Clustergrammer-Web* application developers add a Clustergrammer to directly their own page using the core libraries:

Clustergrammer-JS: The front-end *Clustergrammer-JS* JavaScript library generates the interactive visualization and can be installed via npm: `npm install Clustergrammer`. See the *Example Pages* for templates to build a site with your visualization

Clustergrammer-PY: The back-end *Clustergrammer-PY* Python library clusters a matrix of data and makes the JSON for the front end and can be installed using pip: `pip install --upgrade clustergrammer`. See the *Python Workflow Examples* for examples of how cluster your matrix and generate the *Visualization-JSON*

To make a page, simply include the *Clustergrammer-JS* script in your page and load the pre-calculated *visualization-JSON* to generate a visualization (use *Clustergrammer-PY* to generate this JSON).

Clustergrammer-JS can also be included as a node module (see Clustergrammer-Widget source code for an example with Webpack), or can be used with RequireJS (see Clustergrammer RequireJS example).

Jupyter Notebook Webpages

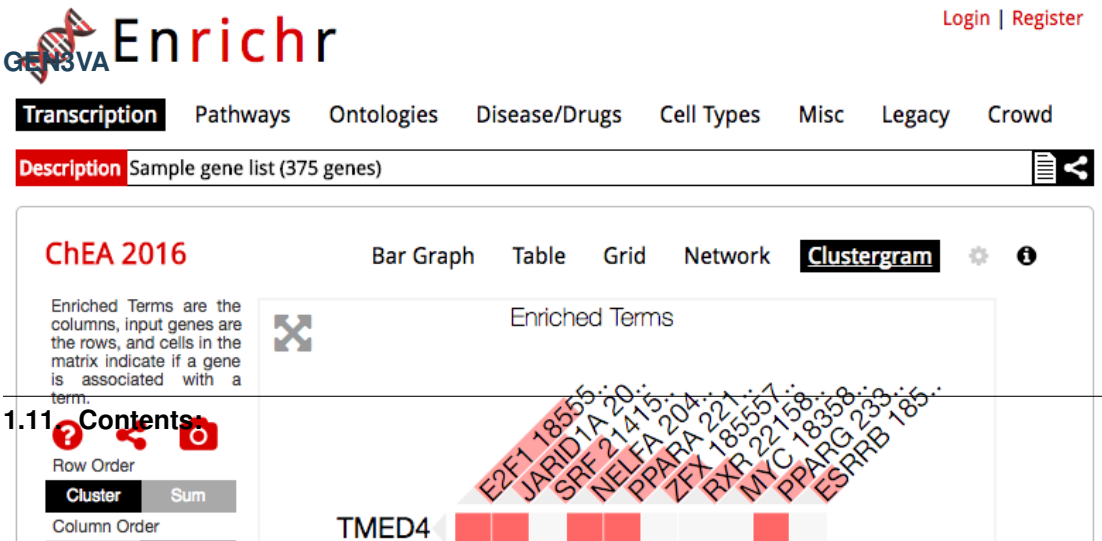
The *Clustergrammer-Widget* can also be used in combination with nbviewer to share static Jupyter notebook web pages with embedded interactive Clustergrammer visualizations. This is one of the easiest ways to generate a web page with Clustergrammer visualizations and several of the *Case Studies and Tutorials* are Jupyter notebooks.

1.11.10 App Integrations

Clustergrammer can be integrated into web applications to dynamically produce interactive visualizations – see *Web-Development* for information. Clustergrammer is currently being utilized to visualize data for the following Ma'ayan lab web applications:

Enrichr

The enrichment analysis tool, Enrichr, uses Clustergrammer to produce dynamic heatmaps of enriched terms as columns and user input genes as rows, which helps users understand the relationships between their input genes and enriched terms.

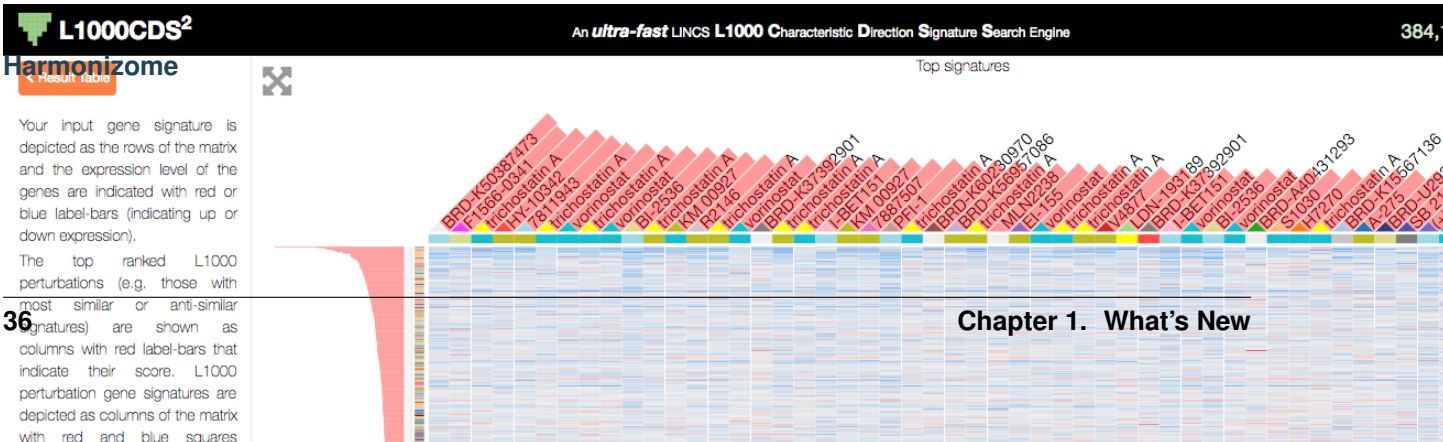


The gene signature

analysis and visualization tool, GEN3VA, uses Clustergrammer's core libraries, *Clustergrammer-JS* and *Clustergrammer-PY*, to dynamically visualize collections of gene expression signatures collected by users from GEO as interactive heatmaps. GEN3VA also uses Clustergrammer to visualize enrichment analysis results (obtained from Enrichr) and perturbations that reverse or mimic gene expression signatures (obtained from L1000CDS2)

L1000CDS2

L1000CDS2 uses the *Clustergrammer-Web* API to produce interactive heatmaps of perturbagen gene signatures that mimic or reverse an input gene signature. This can be useful for users who are interested in the specific genes that are differentially regulated by the identified perturbagens.



Harmonizome

(e.g. to depict networks). The Harmonizome also uses the Clustergrammer to visualize the amount of biological information that is available for different families of genes in the Harmonogram

1. *Clustergrammer-JS Development*
2. *Clustergrammer-PY Development*
3. *Clustergrammer-Widget Development*
4. *Clustergrammer-Web Development*

Clustergrammer-JS and *Clustergrammer-PY* are the two core libraries that are used to build the *Clustergrammer-Widget* and the *Clustergrammer-Web*; these can be used by developers to build their own web pages and apps.

See Clustergrammer2 for Latest Widget

Clustergrammer2 is the new WebGL widget that is being developed to handle larger datasets (e.g. scRNA-seq data). This widget will be the focus of future development and feature additions. The original Clustergrammer-Widget will still be maintained, but users are encouraged to migrate to *Clustergrammer2*.

About Clustergrammer-Widget

Jupyter notebooks are ideal for generating reproducible workflows and analysis. They are also the best way to share Clustergrammer's interactive visualizations while providing context, analysis, and the underlying data to enable reproducibility (see *Sharing with nbviewer*). The Clustergrammer Widget enables users to easily produce interactive visualizations within a Jupyter notebook that can be shared with collaborators (using nbviewer). Clustergrammer-Widget can be used to visualize a matrix of data from a file or from a Pandas DataFrame (see *Matrix Formats and Input/Output* for more information). The library is free and open-source and can be found on GitHub.

more information:

- Running_clustergrammer_widget.ipynb
- DataFrame_Example.ipynb
- CCLE Jupyter Notebook
- Lung Cancer PTM and Gene Expression Regulation
- Single-Cell CyTOF Data
- MNIST Notebook
- USDA Nutrient Dataset
- Iris Dataset.ipynb

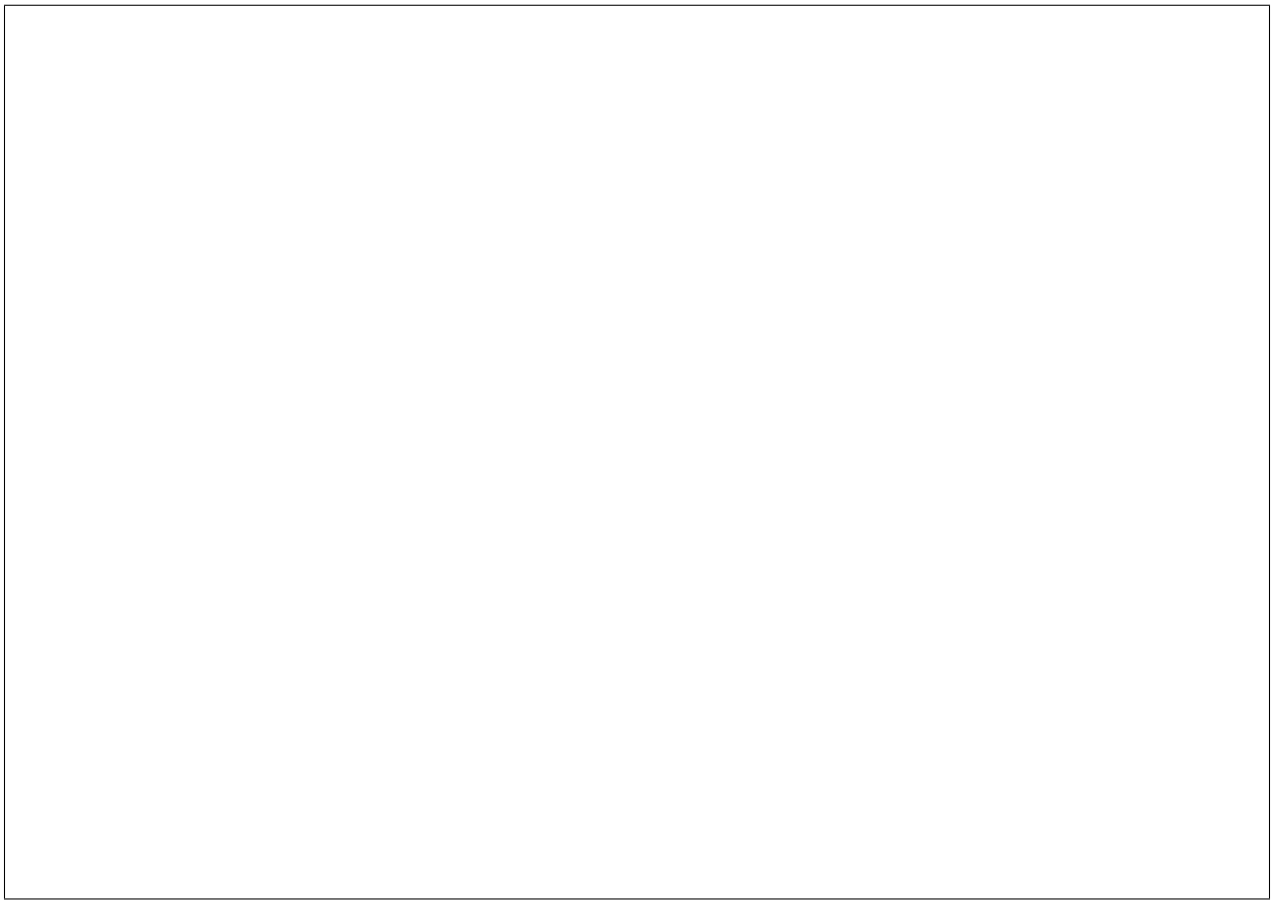
Jupyter Widget Dependencies

- Numpy


- SciPy
- Pandas
- ipywidgets

Installation

distribution that includes the Jupyter notebook as well as other scientific computing libraries, to easily obtain the necessary dependencies (except ipywidgets version 6.0.0). The `clustergrammer_widget` can the be installed (with pip) and enabled using the following commands:

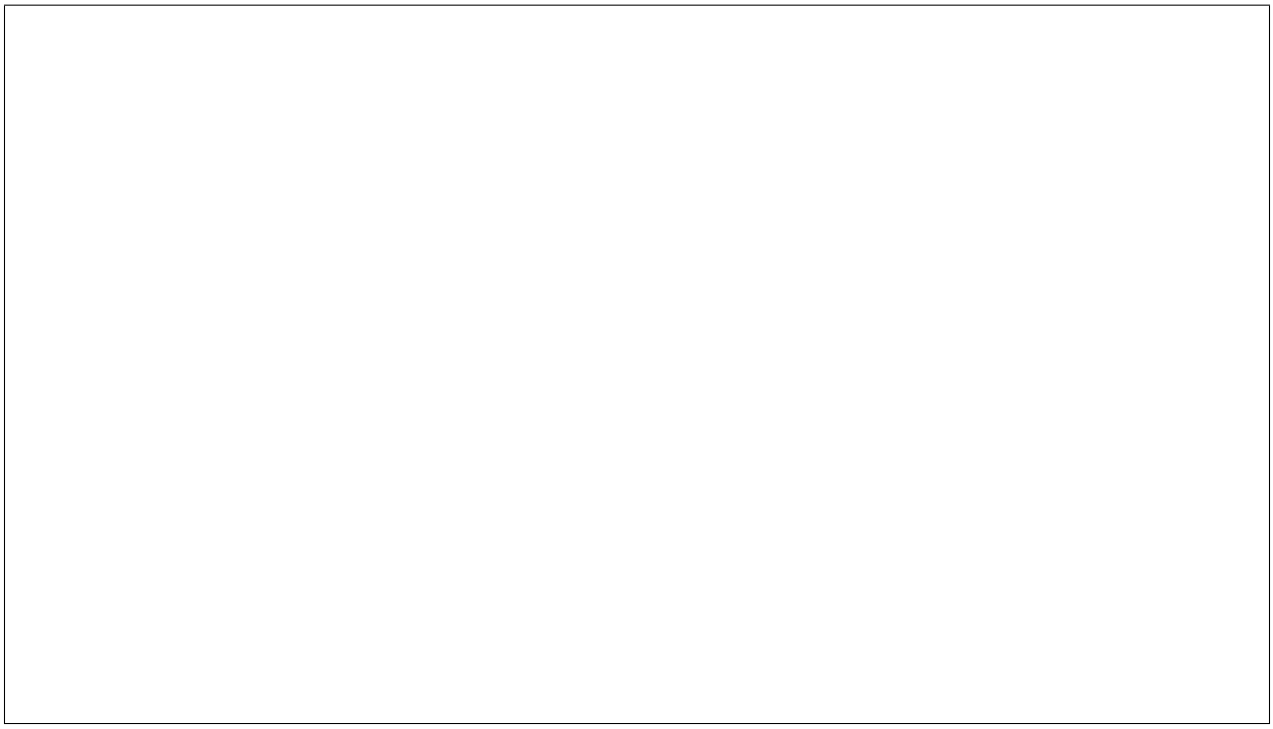


Clustergrammer-Widget Workflow Example

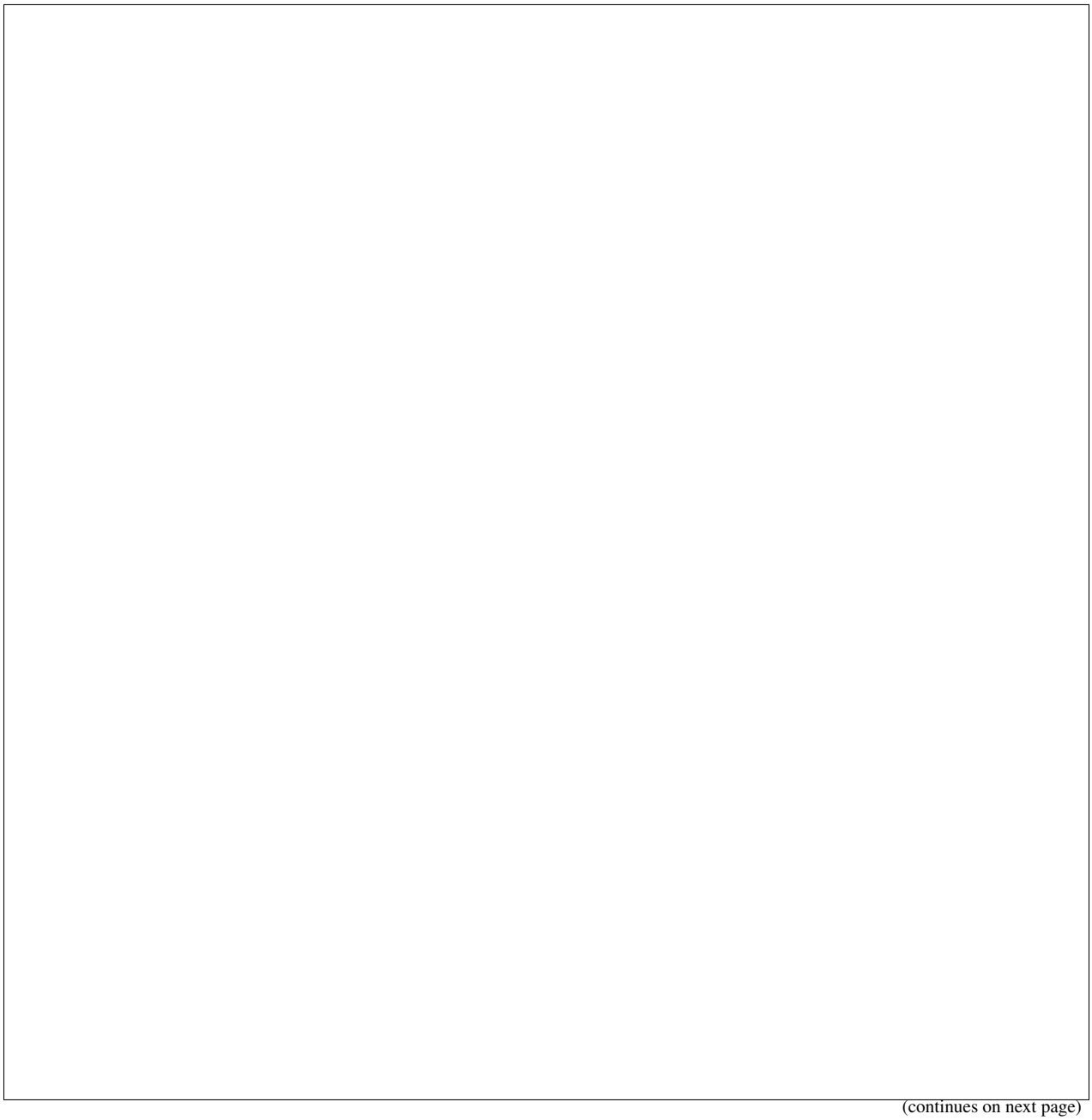
ter, normalize, cluster, and render the widget. For more information about the `Network` class, refer to the *Clustergrammer-PY API*.

(continues on next page)

(continued from previous page)



filters to keep the top 200 rows based on their absolute value sum, calculates clustering, and finally renders the widget:



(continues on next page)

(continued from previous page)

trix (e.g. cropped matrix) to the Python kernel as a DataFrame. This can be used to select a cluster of interest (e.g. by *Cropping* or using the *Interactive Dendrogram*) and pass this cluster to a new DataFrame. Alternatively, this can be used to export data to a DataFrame after running front-end enrichment analysis using *Enrichrgram*. See the `df_widget` method below for an example:

(continues on next page)

(continued from previous page)

Sharing with nbviewer

Widgets on nbviewer documentation and screenshot below):

Clustergrammer-Widget Development

Clustergrammer-Widget's source code can be found in the clustergrammer-widget GitHub repo. Clustergrammer-

Widget is built using the ipywidgets framework (using the cookie cutter template).

Please *Contact* Nicolas Fernandez and Avi Ma'ayan with questions or use the GitHub issues feature to report an issue.

1.11.13 Clustergrammer-JS

Clustergrammer-JS is the front end JavaScript library that builds the interactive clustergram visualization in SVG using the visualization library D3.js. The library is free and open-source and can be found on GitHub.

Clustergrammer-JS Dependencies

- D3.js
- JQuery
- Underscore
- Bootstrap

Installation

Clustergrammer.js can be installed using node package manager (npm package) with the following:

```
npm␣
↪install␣
↪clustergrammer
```

or the source code and library, `clustergrammer.js`, can be obtained from the Clustergrammer GitHub repo.

JavaScript Workflow Example

This workflow shows how to generate a visualization using the JSON produced by Clustergrammer.py

```
/
↪/
↪␣
↪the␣
↪visualization␣
↪JSON␣
↪(produced␣
↪by␣
↪Clustergrammer-
↪PY)
var network_
↪data = {
                ␣
↪         "row_
↪nodes":[...],
                ␣
↪         "col_
↪nodes":[...],
          ␣
↪                ␣
↪ "mat": [...]
          ␣
↪              }

/
↪/
↪␣
↪args␣
↪must␣
↪contain␣
↪root␣
↪of␣
↪container␣
↪and␣
↪the␣
↪visualization␣
↪JSON
```

(continues on next page)

(continued from previous page)

```
var args = {
  'root
↪': '#id_of_
↪container',
  'network_data
↪': 'network_
↪data'
}

/
↪/
↪␣
↪Clustergrammer␣
↪returns␣
↪a␣
↪Clustergrammer␣
↪object␣
↪in addition␣
↪to making
// the␣
↪visualization
var␣
↪cgm␣
↪=␣
↪Clustergrammer(args);
↪
```

The `id` of the container where the visualization will be made is passed as `root` (this root container must be made by the user). The *Visualization-JSON* (produced by *Clustergrammer-PY*) contains all the information necessary to generate the visualization and is passed in this example as `network_data`.

See the *Clustergrammer-JS API* for additional arguments that can be passed to Clustergrammer.js.

Example Pages

The Clustergrammer GitHub repo contains several example pages demonstrating how to build a

web page with a Clustergrammer visualization. The page index.html and corresponding script load_clustergram.js show how to make a full-screen resizable visualization. The page multiple_clust.html and corresponding script load_multiple_clustergrams.js show how to visualize multiple clustergrams on one page. Note that each visualization requires its own container.

Clustergrammer-JS API

class Clustergrammer(*args*)

The Clustergrammer JavaScript object takes the `args` object and produces a visualization on the page.

This `args` object has two required arguments, `network_data` and `root`:

`Clustergrammer.args.`**network_d**

This required attribute is where the visualization JSON should be passed as a JavaScript object.

`Clustergrammer.args.`**root**

This required attribute is the `id` (passed as a string) of the container where Clustergrammer will be built. Each Clustergrammer visualization in a page should be passed a unique `id`.

`Clustergrammer.args.row_label`

Pass strings that will be used as 'super-labels' for rows and columns.

Clustergrammer.args.row_label
Scaling factor to increase/decrease the size of the rows and column labels.

Clustergrammer.args.**super_lab**
Scaling factor to increase/decrease the size of the row/column 'super-labels'.

Clustergrammer.args.**opacity_s**
Name of the scaling function, e.g. *linear*, *log*, used to map matrix cell values to cell opacity. The default is *linear*.

Clustergrammer.args.**input_dom**
The *input_domain* defines the maximum absolute value of matrix cells that are mapped to the maximum opacity of 1. The default *input_domain* is defined using the maximum absolute value of the matrix. Lowering the *input_domain* value increases the opacity of the overall visualization by setting a cutoff.

Clustergrammer.args.**do_zoom**
This boolean value turns on or off zooming. The default is *true*.

Clustergrammer.args.**tile_colo**
Set the positive and negative colors in the heatmap using an array with color names or hexcode, e.g. `['#ED9124', '#1C86EE']`. The default is *red*

and *blue* for positive and negative, respectively.

`Clustergrammer.args.row_order`
Set the initial ordering for rows and columns. The default is *clust* and the options are:
- *alpha*: order based on names
- *clust*: order based on clustering
- *rank*: order based on sum
- *rank_var*: order based on variance

`Clustergrammer.args.`**`ini_view`**
Load clustergram using an initial filtered *view*.

`Clustergrammer.args.`**`about`**
This attribute is a string (which can include HTML) that will produce a small About section at the top of the sidebar. This can be used to provide a quick description about the data or visualization.

`Clustergrammer.args.`**`row_tip_c`**
Users can pass a callback function that will run when mousing over row labels.

`Clustergrammer.args.`**`col_tip_c`**
Users can pass a callback function that will run when mousing over column labels.

`Clustergrammer.args.`**`tile_tip_`**
Users can pass a callback function that will run when mousing over a matrix-cell (e.g. matrix tile).

Clustergrammer.args.**dendro_ca**
Users can pass a callback function that will run when mousing over a dendrogram cluster (e.g. gray trapezoid)

Clustergrammer.args.**dendro_cl**
Users can pass a callback function that will run when clicking a dendrogram cluster (e.g. gray trapezoid)

Clustergrammer.args.**matrix_up**
Users can pass a callback function that will run anytime the matrix has been updated, for instance when filtering/unfiltering, cropping, etc.

Clustergrammer.args.**ini_expan**
Initialize the visualization in 'expanded' mode where the sidebar is not visible. The sidebar can be shown by clicking the menu button on the top left of the visualization.

Clustergrammer.args.**sidebar_w**
Users can modify the width of the sidebar by specifying the width of the sidebar in pixels as a number.

Clustergrammer.args.**ini_view**
Users can initialize the 'view' of their matrix, e.g. initialize the matrix at a particular row filtering level.

Clustergrammer.args.**make_moda**
This boolean option gives users have the option to not

make any Bootstrap modals (e.g. dendrogram group modals) and the default is `true`.

Clustergrammer's attributes and functions are listed below:

`Clustergrammer.`**`params`**
The Clustergrammer parameters object, which contains all the parameters necessary to generate the visualization.

`Clustergrammer.`**`update_cats`**(*row*
Update the visualization row categories.

Arguments

- **`row_data`** – Row category data.

`Clustergrammer.`**`reset_cats`**`()`
Reset the row categories to their original state.

`Clustergrammer.`**`resize_viz:`**`()`
Call this function to resize the visualization to fit in its resized container (if the user has resized the container).

`Clustergrammer.`**`d3_tip_custom`**`()`
Generate a D3 tooltip for SVG elements.

`Clustergrammer.`**`update_view`**(*filte*
Update the heatmap with a specified row filter 'view'.

Arguments

- **`filter_type`** – The available filter types sum or variance:

e.g. N_row_sum, N_row_var

- **inst_state** – The value of the row filter, e.g. 500

Clustergrammer.**filter_viz_usir**
Update the visualization to show the row and column names specified in the `names` object.

Arguments

- **names** – Object with `row` and/or `col` attributes that specify the row and column names that will be visible after updating. Row and column names should be given as a list. Users can include only one attribute, e.g. filter rows only by including no `col` attribute, to only filter rows or columns (or users can specify an empty list to not filter).

Clustergrammer.**filter_viz_usir**
Update the visualization to show the row and column names specified in the `nodes` object.

Arguments

- **names** – Object with `row` and `col` attributes that specify the row and column nodes that will be visible after updating.

Clustergrammer.**zoom**(*pan_x*, *pan_y*,
Zoom and pan into the visualization.

Arguments

- **pan_x** – Panning in the *x* direction
- **pan_y** – Panning in the *y* direction
- **zoom** – The zoom level applied to the visualization.

`Clustergrammer.`**`export_matrix`**`()`
Save the current matrix (e.g. after cropping) as a tab-separated file.

Visualization-JSON

The visualization-JSON is calculated by *Clustergrammer-PY* and encodes everything needed for the front end Clustergrammer-JS to produce the visualization. The visualization-JSON format is described here (see clustergrammer_example.json for an example file). An overview of the format is shown below (note that the group arrays are not shown):

```
{
  "row_nodes":[
     {
      "name
↪": "ATF7",
   ␣
↪  "clust": 67,
↪   ␣
      "rank": 66,
       "rankvar
↪": 10,
↪   ␣
      "group": []
     }
  ],
  "col_nodes":[
     {
       "name
↪": "Col-0",
```

(continues on next page)

(continued from previous page)

```
      "clust": 4,
      "rank": 10,
      "rankvar": 120,
      "group": []
    }
  ],
  "mat":[
    [1, 2],
    [3, 4],
    [5, 6]
  ],
  "links":[
    {
      "source": 0,
      "target": 0,
      "value": 0.023
    }
  ]
}
```

Optional 'views' of the matrix (e.g. row-filtered views) are encoded into the `views` attribute at the base level of the object. These views are used to store a filtered version of the matrix. Only the row and column names are stored in these views since all views share the same matrix cells. The view attributes are stored in the view object (e.g. `N_row_sum`):

```
"views":[
  {
    "N_row_sum": "all",
    "dist": "cos",
    "nodes":{
      "row_nodes": [],
```

(continues on next page)

(continued from previous page)

```
                "col_
↪nodes": []
            }
        }
```

There are three required properties for the Visualization-JSON: `row_nodes`, `col_nodes`, and `mat` (`links` can be used in place of `mat` and will continue to be supported, but the default format will use `mat`). Each of these properties is an array of objects and these objects are discussed below.

Nodes

`row_nodes` and `col_nodes` objects are required to have three properties: `name`, `clust`, `rank`. `name` specifies the name given to the row or column. `clust` and `rank` give the ordering of the row or column in the clustergram. Two optional properties are `group` and `value`. `group` is an array that contains group-membership of the row/column at different dendrogram distance cutoffs and is necessary for displaying a dendrogram. If nodes have the `value` property, then semi-transparent bars will be displayed behind the labels to represent this value.

Mat

`mat` is an JavaScript array that stores the matrix data. The `source` and `target` of each value (row and column) are inferred from the position

of the data in the two-dimensional array.

Links

Note: `mat` will be used by default instead of `links`, but both formats will be supported (`mat` is usually a more compact format). `links` have three properties: `source`, `target`, and `value`. `source` and `target` give the integer value of the row and column of the matrix-cell in the visualization. `value` specifies the opacity and color of the matrix-cell, where positive/negative values results in red/blue matrix-cells in the visualization. The optional properties `value_up` and `value_dn` allow the user to have a split matrix-cell that has an up-triangle and a down-triangle.

Users can also generate the visualization-JSON using their own scripts provided that they adhere to the above format.

Clustergrammer-JS Development

The Clustergrammer-JS source code can be found in the Clustergrammer GitHub repo. The Clustergrammer-JS library is utilized by the *Clustergrammer-Web* and the *Clustergrammer-Widget*. Clustergrammer-JS is built with Webpack Module Bundler from the source files in the src directory.

Please *Contact* Nicolas Fernandez and Avi Ma'ayan with questions or use the GitHub issues feature to report an issue.

1.11.14 Clustergrammer-PY

Clustergrammer-PY is the back end Python library that is used to hierarchically cluster the data and generate the *Visualization-JSON* for the front end *Clustergrammer-JS* visualization library. Clustergrammer-PY is compatible with Python 2 and 3. The library is free and open-source and can be found on GitHub.

Clustergrammer-PY Dependencies

- Numpy
- SciPy
- Pandas
- scikit-learn

Installation

Clustergrammer-PY can be installed using pip (package index) with the following:

```
pip␣
↪install␣
↪-
↪-
↪upgrade␣
↪clustergrammer
```

or the source code can be obtained from the GitHub repo.

Python Workflow Examples

This workflow shows how to cluster a matrix of data from a file (see *Matrix Formats and Input/Output*) and generate a *Visualization-JSON* (for use by *Clustergrammer-JS*):

```
# make network object and load file
from clustergrammer import Network
net = Network()
net.load_file('your_matrix.txt')

# calculate clustering using default parameters
net.cluster()


# save visualization JSON to file for use by front end
net.write_json_to_file('viz', 'mult_view.json')
```

The file `mult_view.json` will be loaded by the front end and used to build the interactive visualization. See

clusterergrammer.py for an additional example.

Clustergrammer can also load data from a Pandas DataFrame and perform normalization and filtering. In this example, we will load data from a DataFrame, normalize the rows, and filter the columns:

```
# make network
↪object
↪and load
↪DataFrame, df
net = Network()
net.load_df(df)

# Z-score
↪normalize
↪the rows
net.
↪normalize(axis=
↪'row', norm_
↪type='zscore
↪', keep_
↪orig=True)

# filter
↪for the top
↪100 columns
↪based
↪on their
↪absolute
↪value sum
net.filter_
↪N_top('col',
↪ 100, 'sum')

# cluster
↪using
↪default
↪parameters
net.cluster()


↪#
↪save
↪visualization
↪JSON to
↪file for use
↪by front end
net.write_
↪json_to_file(
↪'v12', 'mult_
↪view.json')
```

(continues on next page)

(continued from previous page)

Note that filtering done on the `Network` object before clustering is permanent, unlike the filtering done within `cluster` which can be toggled on and off in the front end visualization. The `keep_orig` parameter in the `normalize` function allows us to show un-normalized data a user mouses over a matrix-cell in the visualization. See the *Clustergrammer-PY API* documentation below for more information.

Clustergrammer-PY API

Clustergrammer-PY generates a Network object (see Network class definition), which is used to load a matrix (e.g. from a Pandas DataFrame), optionally normalize or filter the matrix, cluster the matrix, and finally generate the visualization JSON for the front end Clustergrammer.js.

When a matrix is loaded into an instance of `Network` (e.g. `net.`

load_file('your_file.txt')) it is stored in the data, dat, attribute. Normalization and filtering will permanently modify the dat representation of the matrix. When the matrix is clustered (by calling cluster) this produces the *Visualization-JSON*, which is stored in the viz attribute. This JSON can then be exported as a string using net.export_net_json('viz') or saved to a file using net.write_json_to_file('viz', filename).

The function cluster calculates hierarchical clustering of your data and hierarchical clustering of successive-row-filtered versions of your data. These alternate filtered-views are stored as views within the *Visualization-JSON*.

class clustergrammer_py.**Networ**
version 1.13.5

Clustergrammer.py takes a matrix as input (either from a file of a Pandas DataFrame), normalizes/filters, hierarchically clusters, and produces the *Visualization-JSON* for *Clustergrammer-JS*.

Networks have two states:

1. the data state, where they are stored as a matrix and nodes
2. the viz state where they are stored as viz.links, viz.row_nodes, and viz.col_nodes.

The goal is to start in a data-state and produce a viz-state of the network that will be used as input to clustergram.js.

add_cats(*axis*, *cat_data*)

Add categories to rows or columns using cat_data array of objects. Each object in cat_data is a dictionary with one key (category title) and value (rows/column names) that have this category. Categories will be added onto the existing categories and will be added in the order of the objects in the array.

Example `cat_data`:

```
[
  {
    "title": "First Category",
    "cats": {
      "true": [
        "ROS1",
        "AAK1"
      ]
    }
  },
  {
    "title": "Second Category",
    "cats": {
      "something": [
        "PDK4"
      ]
    }
  }
]
```

clip (*lower=None*, *upper=None*)
Trim values at input thresholds using pandas function

cluster (*dist_type='cosine'*, *run_clu*
'N_row_var'], *linkage_ty*
calc_cat_pval=False, *run_enric*
The main function per-

forms hierarchical clustering, optionally generates filtered views (e.g. row-filtered views), and generates the

:visualization_json.

dat_to_df()
Export Pandas DataFrams (will be deprecated).

dendro_cats(*axis*, *dendro_level*)
Generate categories from dendrogram groups/clusters. The dendrogram has 11 levels to choose from 0 -> 10. Dendro_level can be given as an integer or string.

df_to_dat(*df*, *define_cat_colors=False*)
Load Pandas DataFrame (will be deprecated).

downsample(*df=None*, *ds_type='kmeans*
Downsample the matrix rows or columns (currently supporting kmeans only). Users can optionally pass in a DataFrame to be downsampled (and this will be incorporated into the network object).

enrichrgram(*lib*, *axis='row'*)
Add Enrichr gene enrichment results to your visualization (where your rows are genes). Run enrichrgram before

clustering to include enrichment results as row categories. Enrichrgram can also be run on the front-end using the Enrichr logo at the top left.

Set lib to the Enrichr library that you want to use for enrichment analysis. Libraries included:

- ChEA_2016
- KEA_2015
- ENCODE_TF_ChIP-seq_2015
- ENCODE_Histone_Modifications_201
- Disease_Perturbations_from_GEO_up
- Disease_Perturbations_from_GEO_do
- GO_Molecular_Function_2015
- GO_Biological_Process_2015
- GO_Cellular_Component_2015
- Reactome_2016
- KEGG_2016
- MGI_Mammalian_Phenotype_Level_4
- LINCS_L1000_Chem_Pert_up
- LINCS_L1000_Chem_Pert_down

export_df()
Export Pandas DataFrame/

export_net_json(*net_type='viz'*, *inde*
Export dat or viz JSON.

export_viz_to_widget (*which_viz='*
Export viz JSON, for use with clustergrammer_widget. Formerly method was named widget.

filter_N_top (*inst_rc*, *N_top*, *rank_type*
Filter the matrix rows or columns based on sum/variance, and only keep the top N.

filter_cat (*axis*, *cat_index*, *cat_name*)
Filter the matrix based on their category. cat_index is the index of the category, the first category has index=1.

filter_names (*axis*, *names*)
Filter the visualization using row/column names. The function takes, axis ('row'/'col') and names, a list of strings.

filter_sum (*inst_rc*, *threshold*, *take_abs*
Filter a network's rows or columns based on the sum across rows or columns.

filter_threshold (*inst_rc*, *threshold*, *
Filter the matrix rows or columns based on num_occur values being above a threshold (in absolute value).

load_data_file_to_net (*filename*)
Load Clustergrammer's dat format (saved as JSON).

load_df (*df*)
Load Pandas DataFrame.

load_file (*filename*)

Load TSV file.

load_file_as_string (*file_string*, *file*
Load file as a string.

load_stdin ()
Load stdin TSV-formatted string.

load_tsv_to_net (*file_buffer*, *filename*=
This will load a TSV matrix file buffer; this is exposed so that it will be possible to load data without having to read from a file.

load_vect_post_to_net (*vect_post*)
Load data in the vector format JSON.

make_clust (*dist_type='cosine', run_ 'N_row_var'], linkage calc_cat_pval=False, run_e*
... Will be deprecated, renaming method cluster ... The main function performs hierarchical clustering, optionally generates filtered views (e.g. row-filtered views), and generates the

:visualization_json.

normalize (*df=None*, *norm_type='zscore*
Normalize the matrix rows or columns using Z-score (zscore) or Quantile Normalization (qn). Users can optionally pass in a DataFrame to be normalized (and this will be incorporated into the Network object).

produce_view (*requested_view=None*)
This function is un-

der development and will produce a single view on demand.

random_sample(*num_samples*, *df=No*
axis='row')
Return random sample of matrix.

reset()
This re-initializes the Network object.

set_cat_color(*axis*, *cat_index*, *cat_na*
Set row/column category colors using index, name and specified color.

swap_nan_for_zero()
Swaps all NaN (numpy NaN) instances for zero.

widget(*which_viz='viz'*)
Generate a widget visualization using the widget. The export_viz_to_widget method passes the visualization JSON to the instantiated widget, which is returned and visualized on the front-end.

widget_df()
Export a DataFrame from the front-end visualization. For instance, a user can filter to show only a single cluster using the dendrogram and then get a DataFrame of this cluster using the widget_df method.

write_json_to_file(*net_type*, *filenar*
Save dat or viz as a JSON to file.

write_matrix_to_tsv(*filename=Non*
Export data-matrix to file.

Clustergrammer-PY Development

Clustergrammer-PY's source code can be found in the clustergrammer-py GitHub repo. The Clustergrammer-PY library is utilized by the *Clustergrammer-Web* and the *Clustergrammer-Widget*.

Please *Contact* Nicolas Fernandez and Avi Ma'ayan with questions or use the GitHub issues feature to report an issue.

1.11.15 License

Clustergrammer is being developed by the Ma'ayan lab at the Icahn School of Medicine at Mount Sinai for the BD2K-LINCS DCIC and the KMC-IDG. Please contact Nicolas Fernandez (nicolas.fernandez@mssm.edu) and Avi Ma'ayan (avi.maayan@mssm.edu) with any questions about the License.

Clustergrammer's license and third-party licenses are in the LICENSES directory.

Python Module Index

C

Index