logomaker Documentation

Release 0.8

Ammar Tareen and Justin B. Kinney

Dec 14, 2019

Contents:

Logomaker is a Python package for generating publication-quality sequence logos. Logomaker can generate both standard and highly customized logos illustrating the properties of DNA, RNA, or protein sequences. Logos are rendered as vector graphics embedded within native matplotlib Axes objects, making them easy to style and incorporate into multi-panel figures. The *Installation*, *Quick Start*, *Examples*, and tutorial sections below are provided to help users quickly get Logomaker working for their own research needs. For more information about Logomaker, please see Tareen and Kinney (2019)[1].

[1] Tareen A, Kinney JB (2019) Logomaker: beautiful sequence logos in Python. Bioinformatics btz921. bioRxiv doi:10.1101/635029.

CHAPTER 1

Installation

Logomaker has minimal dependencies and is compatible with both Python 2.7 and Python 3.6. The code for Logomaker is available on GitHub under an MIT open source license. Logomaker can be installed from PyPI using the `pip` package manager by executing the following at the commandline:

```
pip install logomaker
```

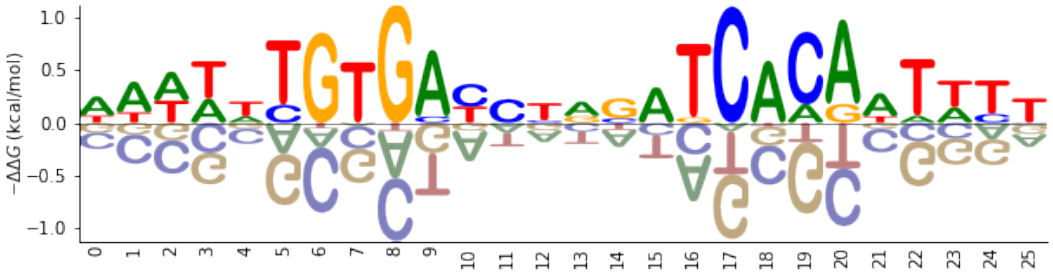
CHAPTER 2

Quick Start

For a quick demonstration of Logomaker, execute the following within Python:

```
import logomaker
logomaker.demo('fig1b')
```

This command will generate a sequence logo representing the DNA binding specificity of CRP, a major transcription factor in *Escherichia coli*:



This command will also print out the code used to generate the logo. We note that the logo shown here is from Figure 1B of Tareen and Kinney (2019)[1], and that the other logos in Figure 1 can be generated in a similar manner.

CHAPTER 3

Tutorial

A tutorial is available via a series of Jupyter notebooks, each of which focuses on a different aspect of Logomaker's functionality. To run each notebook interactively, click the Binder badge below. To instead view the notebooks statically on GitHub, click here.

CHAPTER 4

Resources

4.1 Examples

As described in *Quick Start*, the five logos shown in Figure 1 of Tareen and Kinney (2019)[1] can be generated using the function `logomaker.demo`. Here we describe each of these logos, as well as the snippets of code used to generate them. All snippets shown below are designed for use within a Jupyter Notebook, and assume that the following header cell has already been run.

```
# standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# displays logos inline within the notebook;
# remove if using a python interpreter instead
%matplotlib inline

# logomaker import
import logomaker
```

4.1.1 CRP energy logo

The following code creates an energy logo for the *E. coli* transcription factor CRP. The energy matrix illustrated by this logo was reported by Kinney et. al. (2010)[2] based on the analysis of data from a massively parallel reporter assay. This energy matrix is included with Logomaker as example data, and is loaded here by calling `logomaker.get_example_matrix` with the argument `'crp_energy_matrix'`. A Logo object named `crp_logo` is then created using the styling parameters `shade_below`, `fade_below`, and `font_name`. Subsequent styling is then performed using the Logo object methods `style_spines` and `style_xticks`. Additional styling is also performed using methods of `crp_logo.ax`, the matplotlib Axes object on which the logo is drawn.

[1] Tareen A, Kinney JB (2019). Logomaker: beautiful sequence logos in Python. bioRxiv doi:10.1101/635029.

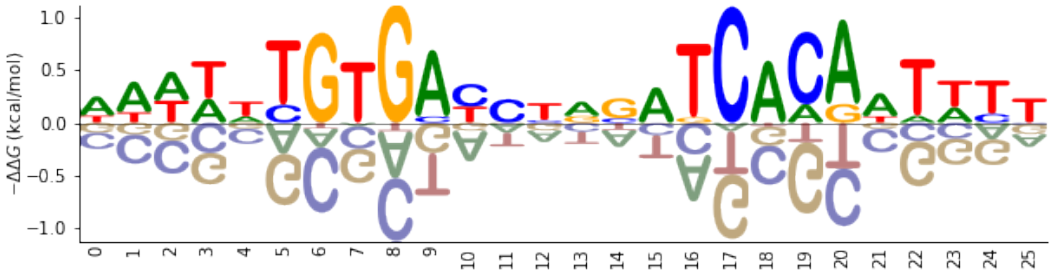
[2] Kinney JB, Murugan A, Callan CG, Cox EC (2010). Using deep sequencing to characterize the biophysical mechanism of a transcriptional regulatory sequence. Proc Natl Acad Sci USA 107:9158-9163. PubMed.

```
# load crp energy matrix
crp_df = -logomaker.get_example_matrix('crp_energy_matrix',
                                       print_description=False)

# create Logo object
crp_logo = logomaker.Logo(crp_df,
                          shade_below=.5,
                          fade_below=.5,
                          font_name='Arial Rounded MT Bold')

# style using Logo methods
crp_logo.style_spines(visible=False)
crp_logo.style_spines(spines=['left', 'bottom'], visible=True)
crp_logo.style_xticks(rotation=90, fmt='%d', anchor=0)

# style using Axes methods
crp_logo.ax.set_ylabel("$-\Delta \Delta G$ (kcal/mol)", labelpad=-1)
crp_logo.ax.xaxis.set_ticks_position('none')
crp_logo.ax.xaxis.set_tick_params(pad=-1)
```



4.1.2 Splice site probability logo

The following code creates a probability logo derived from all 5' splice sites annotated in the human genome[3]. Here the probability of each RNA nucleotide at each position is indicated by both character height and character opacity. The dashed line indicates the intron/exon boundary, with exonic sequence on the left and intronic sequence on the right. This probability matrix is included with Logomaker as example data, and is loaded into a pandas DataFrame object named `ss_df` by calling `logomaker.get_example_matrix` with the argument `'ss_probability_matrix'`. A Logo object named `ss_logo` is then created using the styling parameters `width`, `vpad`, `fade_probabilities`, `stack_order`, `color_scheme`, and `font_name`. Subsequent styling is performed using the Logo object method `style_spines`, as well as multiple Axes object methods.

```
# load ss probability matrix
ss_df = logomaker.get_example_matrix('ss_probability_matrix',
                                     print_description=False)

# create Logo object
ss_logo = logomaker.Logo(ss_df,
                         width=.8,
                         vpad=.05,
                         fade_probabilities=True,
                         stack_order='small_on_top',
                         color_scheme='dodgerblue',
```

(continues on next page)

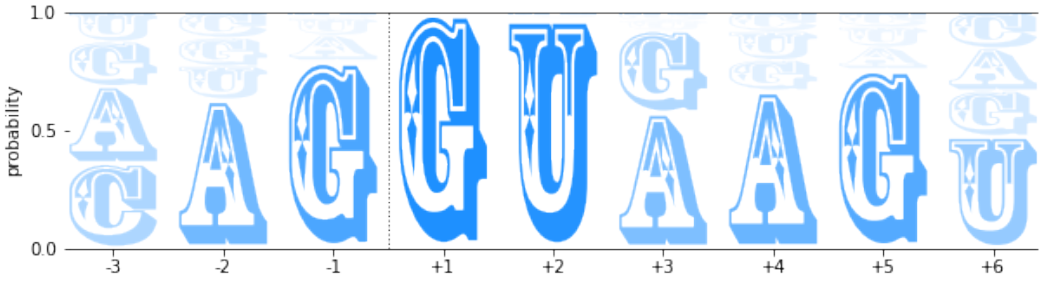
[3] Frankish A et al. (2019). GENCODE reference annotation for the human and mouse genomes. Nucl Acids Res, 47(D1):D766–D773. PubMed.

(continued from previous page)

```
                          font_name='Rosewood Std')

# style using Logo methods
ss_logo.style_spines(spines=['left', 'right'], visible=False)

# style using Axes methods
ss_logo.ax.set_xticks(range(len(ss_df)))
ss_logo.ax.set_xticklabels('%+d'%x for x in [-3, -2, -1, 1, 2, 3, 4, 5, 6])
ss_logo.ax.set_yticks([0, .5, 1])
ss_logo.ax.axvline(2.5, color='k', linewidth=1, linestyle=':')
ss_logo.ax.set_ylabel('probability')
```



4.1.3 WW domain information logo

The following code creates an information logo derived from a multiple sequence alignment (obtained from PFam[4]) of protein WW domains. Here the height of each stack of characters indicates information content, as described by Schneider and Stevens (1990)[5]. First, the information matrix is loaded into `ww_df` by calling `logomaker.get_example_matrix` with the argument `'ww_information_matrix'`. A Logo object named `ww_logo` is then generated. Among other styling options, setting the `color_scheme` parameter to `'NajafabadiEtAl2017'` causes Logomaker to use a color scheme extracted from Najafabadi et al. (2017)[6]; the list of all available color schemes can be viewed by calling `logomaker.list_color_schemes()`. The Logo object method `highlight_position` is also used to highlight the two eponymous positions of the WW domain.

```
# load ww information matrix
ww_df = logomaker.get_example_matrix('ww_information_matrix',
                                     print_description=False)

# create Logo object
ww_logo = logomaker.Logo(ww_df,
                         font_name='Stencil Std',
                         color_scheme='NajafabadiEtAl2017',
                         vpad=.1,
                         width=.8)

# style using Logo methods
ww_logo.style_xticks(anchor=0, spacing=5, rotation=45)
ww_logo.highlight_position(p=4, color='gold', alpha=.5)
ww_logo.highlight_position(p=26, color='gold', alpha=.5)
```

(continues on next page)

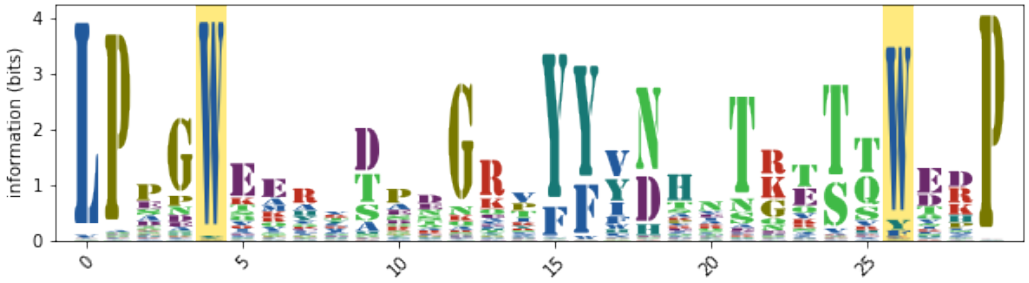
[4] Finn RD, et al. (2014). Pfam: the protein families database. Nucl Acids Res 42(Database issue):D222–30. PubMed.

[5] Schneider TD, Stephens RM (1990). Sequence logos: a new way to display consensus sequences. Nucl Acids Res.18(20):6097–100. PubMed.

[6] Najafabadi HS, et al. (2017). Non-base-contacting residues enable kaleidoscopic evolution of metazoan C2H2 zinc finger DNA binding. Genome Biol. 18(1):1–15. PubMed.

(continued from previous page)

```
# style using Axes methods
ww_logo.ax.set_ylabel('information (bits)')
ww_logo.ax.set_xlim([-1, len(ww_df)])
```



4.1.4 ARS enrichment logo

The following code creates an enrichment logo that illustrates the results of a mutARS-seq experiment (unpublished; performed by JBK) analogous to the one reported by Liachko et al. (2013)[7]. In this logo, the height of each character indicates the log-fold enrichment observed in a plasmid selection experiment performed on a large library of mutated ARS1 origins of replication. First, the enrichment matrix is loaded into `ars_df` by calling `logomaker.get_example_matrix` with the argument `'ars_enrichment_matrix'`. Next, we call `logomaker.open_example_datafile` with argument `'ars_wt_sequence.txt'`; this returns a file handle from which the wild-type ARS1 DNA sequence is parsed. Both the enrichment matrix and the ARS1 sequence are then trimmed. Next, a Logo object named `ars_logo` is created with all characters colored `'dimgray'`. The wild-type ARS1 sequence is then colored in orange by calling `ars_logo.style_glyphs_in_sequence` with the argument `color` set to `'darkorange'`. Three functional elements within ARS1 (termed A, B1, and B2, from left to right) are then highlighted using `ars_logo.highlight_position_range`. Some additional Axes styling is then performed.

```
# load ARS enrichment matrix
ars_df = logomaker.get_example_matrix('ars_enrichment_matrix',
                                      print_description=False)

# load wild-type ARS1 sequence
with logomaker.open_example_datafile('ars_wt_sequence.txt',
                                     print_description=False) as f:
    lines = f.readlines()
    lines = [l.strip() for l in lines if '#' not in l]
    ars_seq = ''.join(lines)

# trim matrix and sequence
start = 10
stop = 100
ars_df = ars_df.iloc[start:stop, :]
ars_df.reset_index(inplace=True, drop=True)
ars_seq = ars_seq[start:stop]

# create Logo object
ars_logo = logomaker.Logo(ars_df,
                          color_scheme='dimgray',
```

(continues on next page)

[7] Liachko I et al. (2013). High-resolution mapping, characterization, and optimization of autonomously replicating sequences in yeast. Genome Res, 23(4):698-704. PubMed.

(continued from previous page)

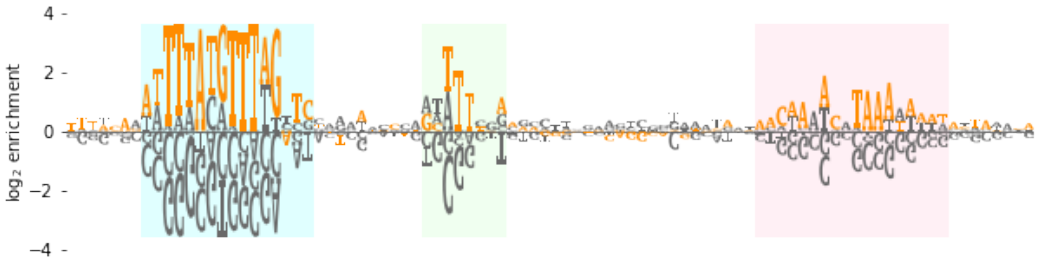
```
                          font_name='Luxi Mono')

# color wild-type ARS1 sequence within logo
ars_logo.style_glyphs_in_sequence(sequence=ars_seq, color='darkorange')

# highlight functional regions of ARS1
ars_logo.highlight_position_range(pmin=7, pmax=22, color='lightcyan')
ars_logo.highlight_position_range(pmin=33, pmax=40, color='honeydew')
ars_logo.highlight_position_range(pmin=64, pmax=81, color='lavenderblush')

# additional styling using Logo methods
ars_logo.style_spines(visible=False)

# style using Axes methods
ars_logo.ax.set_ylim([-4, 4])
ars_logo.ax.set_ylabel('$\log_2$ enrichment', labelpad=0)
ars_logo.ax.set_yticks([-4, -2, 0, 2, 4])
ars_logo.ax.set_xticks([])
```



4.1.5 Neural network saliency logo

Saliency logos provide a useful way to visualize the features (within a specific biological sequence) that a deep neural network model deems to be important. Saliency logos differ from more standard logos in that only one character is drawn at each position. Below we reproduce (with permission) the saliency logo from Figure 1D of Jaganathan et al. (2019)[8], which illustrates sequence features important for the proper splicing of *U2SUR* exon 9. First, the saliency matrix is loaded into `nn_df` by calling `logomaker.get_example_matrix` with the argument `'nn_saliency_matrix'`. Next, a Logo object named `nn_logo` is created and its methods are used to style the Axes spines. More axes styling is then carried out using native Axes methods. Finally, a gene body diagram with annotations is drawn below the logo.

```
# load saliency matrix
nn_df = logomaker.get_example_matrix('nn_saliency_matrix',
                                     print_description=False)

# create Logo object
nn_logo = logomaker.Logo(nn_df)

# style using Logo methods
nn_logo.style_spines(visible=False)
nn_logo.style_spines(spines=['left'], visible=True, bounds=[0, .75])
```

(continues on next page)

[8] Jaganathan K. et al. (2019). Predicting Splicing from Primary Sequence with Deep Learning. Cell, 176(3):535-548.e24. PubMed.

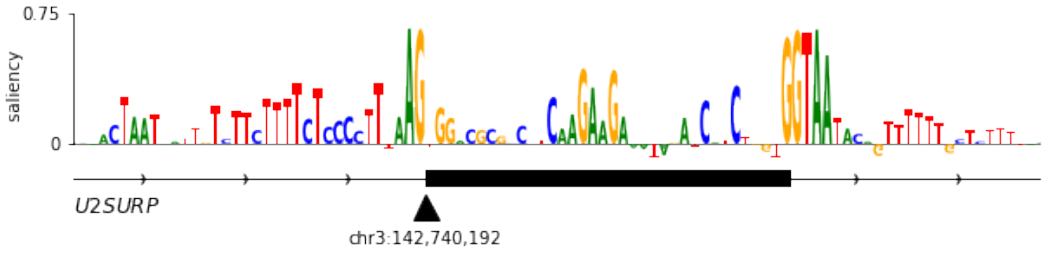
(continued from previous page)

```
# style using Axes methods
nn_logo.ax.set_xlim([20, 115])
nn_logo.ax.set_xticks([])
nn_logo.ax.set_ylim([-.6, .75])
nn_logo.ax.set_yticks([0, .75])
nn_logo.ax.set_yticklabels(['0', '0.75'])
nn_logo.ax.set_ylabel('                saliency', labelpad=-1)

# set parameters for drawing gene
exon_start = 55-.5
exon_stop = 90+.5
y = -.2
xs = np.arange(-3, len(nn_df),10)
ys = y*np.ones(len(xs))

# draw gene
nn_logo.ax.axhline(y, color='k', linewidth=1)
nn_logo.ax.plot(xs, ys, marker='4', linewidth=0, markersize=7, color='k')
nn_logo.ax.plot([exon_start, exon_stop],
                [y, y], color='k', linewidth=10, solid_capstyle='butt')

# annotate gene
nn_logo.ax.plot(exon_start, 1.8*y, '^k', markersize=15)
nn_logo.ax.text(20,2*y,'$U2SURP$',fontsize=12)
nn_logo.ax.text(exon_start, 2.5*y,'chr3:142,740,192', verticalalignment='top',␣
↪horizontalalignment='center')
```



4.1.6 Logomaker logo

Below is the code used to make the Logomaker logo. First, Figure and Axes objects of the desired size are created. The data matrix for the logo is then loaded into `logo_df`. Next, a custom color scheme is defined in the form of a `dict` object. A Logo object is then created using a variety of optional arguments that, among other things, specify the Axes and color scheme to use. Subsequently, the second 'O' in 'LOGO' is recolored, after which the characters in 'marker' are flipped right-side up, rendered in font `'ORC A Std'`, and widened slightly. Finally, tick marks are removed and the Axes is rescaled to fill the Figure.

```
# make Figure and Axes objects
fig, ax = plt.subplots(1,1,figsize=[4,2])

# load logo matrix
logo_df = logomaker.get_example_matrix('logomaker_logo_matrix',
                                       print_description=False)
```

(continues on next page)

(continued from previous page)

```
# create color scheme
color_scheme = {
    'L' : [0, .5, 0],
    'O' : [1, 0, 0],
    'G' : [1, .65, 0],
    'maker': 'gray'
}

# create Logo object
logo_logo = logomaker.Logo(logo_df,
                           ax=ax,
                           color_scheme=color_scheme,
                           baseline_width=0,
                           font_name='Arial',
                           show_spines=False,
                           vsep=.005,
                           width=.95)

# color the 'O' at the end of the logo a different color
logo_logo.style_single_glyph(c='O', p=3, color=[0, 0, 1])

# change the font of 'maker' and flip characters upright.
logo_logo.style_glyphs_below(font_name='OCR A Std', flip=False, width=1.0)

# remove tick marks
ax.set_xticks([])
ax.set_yticks([])

# tighten layout
logo_logo.fig.tight_layout()
```

4.1.7 Color schemes

The following code creates a figure that illustrates all of Logomaker's built-in color schemes. To use one of these color schemes, set the `color_scheme` parameter to the indicated color scheme name when creating a `Logo` object.

```
# get data frame of all color schemes
all_df = logomaker.list_color_schemes()

# set the two types of character sets
char_sets = ['ACGTU', 'ACDEFGHIKLMNPQRSTVWY']
```

(continues on next page)

(continued from previous page)

```
colspans = [1, 3]
num_cols = sum(colspans)

# compute the number of rows
num_rows_per_set = []
for char_set in char_sets:
    num_rows_per_set.append((all_df['characters'] == char_set).sum())
num_rows = max(num_rows_per_set)

# create figure
height_per_row = .8
width_per_col = 1.5
fig = plt.figure(figsize=[width_per_col * num_cols, height_per_row * num_rows])

# for each character set
for j, char_set in enumerate(char_sets):

    # get color schemes for that character set only
    df = all_df[all_df['characters'] == char_set].copy()
    df.sort_values(by='color_scheme', inplace=True)
    df.reset_index(inplace=True, drop=True)

    # for each color scheme
    for row_num, row in df.iterrows():
        # set axes
        col_num = sum(colspans[:j])
        col_span = colspans[j]
        ax = plt.subplot2grid((num_rows, num_cols), (row_num, col_num),
                              colspan=col_span)

        # get color scheme
        color_scheme = row['color_scheme']

        # make matrix for character set
        mat_df = logomaker.sequence_to_matrix(char_set)

        # make and style logo
        logomaker.Logo(mat_df,
                       ax=ax,
                       color_scheme=color_scheme,
                       show_spines=False)
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(repr(color_scheme))

# style and save figure
fig.tight_layout()
```

4.1.8 References

4.2 Implementation

4.2.1 Logo class

class logomaker.**Logo**(***kwargs*)

Logo represents a basic logo, drawn on a specified axes object using a specified matrix, which is supplied as a pandas dataframe.

Attributes

df: (pd.DataFrame) A matrix specifying character heights and positions. Rows correspond to

positions while columns correspond to characters. Column names must be single characters and row indices must be integers.

color_scheme: (str, dict, or array with length 3) Specification of logo colors. Default is 'gray'. Can take a variety of forms.

- (str) A built-in Logomaker color scheme in which the color of each

  character is determined that character's identity. Options are,

  - For DNA/RNA: 'classic', 'grays', or 'base_paring'.
  - For protein: 'hydrophobicity', 'chemistry', or 'charge'.

- (str) A built-in matplotlib color name such as 'k' or 'tomato'
- (list) An RGB array, i.e., 3 floats with values in the interval [0,1]
- **(dict) A dictionary that maps characters to colors, E.g.,**

  {'A': 'blue', 'C': 'yellow', 'G': 'green', 'T': 'red'}

font_name: (str) The character font to use when rendering the logo. For a list of valid font names, run logomaker.list_font_names().

stack_order: (str) Must be 'big_on_top', 'small_on_top', or 'fixed'. If 'big_on_top', stack characters away from x-axis in order of increasing absolute value. If 'small_on_top', stack glyphs away from x-axis in order of decreasing absolute value. If 'fixed', stack glyphs from top to bottom in the order that characters appear in the data frame.

center_values: (bool) If True, the stack of characters at each position will be centered around zero. This is accomplished by subtracting the mean value in each row of the matrix from each element in that row.

baseline_width: (float >= 0.0) Width of the logo baseline, drawn at value 0.0 on the y-axis.

flip_below: (bool) If True, characters below the x-axis (which correspond to negative values in the matrix) will be flipped upside down.

shade_below: (float in [0,1]) The amount of shading to use for characters drawn below the x-axis. Larger numbers correspond to more shading (i.e., darker characters).

fade_below: (float in [0,1]) The amount of fading to use for characters drawn below the x-axis. Larger numbers correspond to more fading (i.e., more transparent characters).

fade_probabilities: (bool) If True, the characters in each stack will be assigned an alpha value equal to their height. This option only makes sense if df is a probability matrix. For additional customization, use Logo.fade_glyphs_in_probability_logo().

vpad: (float in [0,1]) The whitespace to leave above and below each character within that character's bounding box. Note that, if vpad > 0, the height of the character's bounding box (and not of the character itself) will correspond to values in df.

vsep: (float >= 0) Amount of whitespace to leave between the bounding boxes of rendered characters. Unlike vpad, vsep is NOT relative to character height.

alpha: (float in [0,1]) Opacity to use when rendering characters. Note that, if this is used together with fade_below or fade_probabilities, alpha will multiply existing opacity values.

show_spines: (None or bool) Whether a box should be drawn around the logo. For additional customization of spines, use Logo.style_spines().

ax: (matplotlib Axes object) The matplotlb Axes object on which the logo is drawn.

zorder: (int >=0) This governs what other objects drawn on ax will appear in front or behind the rendered logo.

figsize: ([float, float]): The default figure size for the rendered logo; only used if ax is not supplied by the user.

****kwargs:** Additional key word arguments to send to the Glyph constructor.

Methods

draw(self[, clear])	Draws characters in Logo.
draw_baseline(*args, **kwargs)	Draws a horizontal line along the x-axis.
fade_glyphs_in_probability_logo(*args, ...)	Fades glyphs in probability logo according to value.
highlight_position(*args, **kwargs)	Draws a rectangular box highlighting a specific position.
highlight_position_range(*args, **kwargs)	Draws a rectangular box highlighting multiple positions within the Logo
style_glyphs(*args, **kwargs)	Modifies the properties of all characters in a Logo.
style_glyphs_below(*args, **kwargs)	Modifies the properties of all characters drawn below the x-axis.
style_glyphs_in_sequence(*args, **kwargs)	Restyles the glyphs in a specific sequence.
style_single_glyph(*args, **kwargs)	Modifies the properties of a single character in Logo.
style_spines(*args, **kwargs)	Styles the spines of the Axes object in which the logo is drawn.
style_xticks(*args, **kwargs)	Formats and styles tick marks along the x-axis.

draw(*self*, *clear=False*)
Draws characters in Logo.

Parameters

clear: (bool) If True, Axes will be cleared before logo is drawn.

Returns

None

draw_baseline(**args*, ***kwargs*)
Draws a horizontal line along the x-axis.

Parameters

zorder: (number) This governs what other objects drawn on ax will appear in front or behind the baseline. Logo characters are, by default, drawn in front of the baseline.

color: (matplotlib color) Color to use for the baseline. Can be a named matplotlib color or an RGB array.

linewidth: (number >= 0) Width of the baseline.

****kwargs:** Additional keyword arguments to be passed to ax.axhline()

Returns

None

fade_glyphs_in_probability_logo(**args*, ***kwargs*)
Fades glyphs in probability logo according to value.

Parameters

v_alpha0, v_alpha1: (number in [0,1]) Matrix values marking values that are rendered using alpha=0 and alpha=1, respectively. These values must satisfy v_alpha0 < v_alpha1.

Returns

None

highlight_position(**args*, ***kwargs*)
Draws a rectangular box highlighting a specific position.

Parameters

p: (int) Single position to highlight.

****kwargs:** Other parameters to pass to highlight_position_range()

Returns

None

highlight_position_range(**args*, ***kwargs*)
Draws a rectangular box highlighting multiple positions within the Logo

Parameters

pmin: (int) Lowest position to highlight.

pmax: (int) Highest position to highlight.

padding: (number >= -0.5) Amount of padding to add on the left and right sides of highlight.

color: (None or matplotlib color) Color to use for highlight. Can be a named matplotlib color or an RGB array.

edgecolor: (None or matplotlib color) Color to use for highlight box edges. Can be a named matplotlib color or an RGB array.

floor: (None number) Lowest y-axis extent of highlight box. If None, is set to ymin of the Axes object.

ceiling: (None or number) Highest y-axis extent of highlight box. If None, is set to ymax of the Axes object.

zorder: (number) This governs which other objects drawn on ax will appear in front or behind of the highlight. Logo characters are, by default, drawn in front of the highlight box.

Returns

None

style_glyphs(**args*, ***kwargs*)
Modifies the properties of all characters in a Logo.

Parameters

color_scheme: (str, dict, or array with length 3) Specification of logo colors. Default is 'gray'. Can take a variety of forms.

- (str) A built-in Logomaker color scheme in which the color of each

character is determined that character's identity. Options are,

- For DNA/RNA: 'classic', 'grays', or 'base_paring'.
- For protein: 'hydrophobicity', 'chemistry', or 'charge'.

- (str) A built-in matplotlib color name such as 'k' or 'tomato'
- (list) An RGB array, i.e., 3 floats with values in the interval [0,1]
- **(dict) A dictionary that maps characters to colors, E.g.,**

  {'A': 'blue', 'C': 'yellow', 'G': 'green', 'T': 'red'}

****kwargs:** Keyword arguments to pass to Glyph.set_attributes()

Returns

None

style_glyphs_below(**args*, ***kwargs*)

Modifies the properties of all characters drawn below the x-axis.

Parameters

color: (color specification) Color to use before shade is applied.

alpha: (number in [0,1]) Opacity to use when rendering characters, before fade is applied.

shade: (number in [0,1]) The amount to shade characters below the x-axis.

fade: (number in [0,1]) The amount to fade characters below the x-axis.

flip: (bool) If True, characters below the x-axis will be flipped upside down.

****kwargs:** Keyword arguments to pass to Glyph.set_attributes(), but only for characters below the x-axis.

Returns

None

style_glyphs_in_sequence(**args*, ***kwargs*)

Restyles the glyphs in a specific sequence.

Parameters

sequence: (str) A string the same length as the logo, specifying which character to restyle at each position. Characters in sequence that are not in the columns of the Logo's df are ignored.

****kwargs:** Keyword arguments to pass to Glyph.set_attributes()

Returns

None

style_single_glyph(**args*, ***kwargs*)

Modifies the properties of a single character in Logo.

Parameters

p: (int) Position of modified glyph. Must index a row in the matrix df passed to the Logo constructor.

c: (str of length 1) Character to modify. Must be the name of a column in the matrix df passed to the Logo constructor.

****kwargs:** Keyword arguments to pass to Glyph.set_attributes()

Returns

None

style_spines(**args*, ***kwargs*)

Styles the spines of the Axes object in which the logo is drawn. Note: "spines" refers to the edges of the Axes bounding box.

Parameters

spines: (tuple of str) Specifies which of the four spines to modify. The default value for this parameter lists all four spines.

visible: (bool) Whether to show or not show the spines listed in the parameter spines.

color: (matplotlib color) Color of the spines. Can be a named matplotlib color or an RGB array.

linewidth: (float >= 0) Width of lines used to draw the spines.

bounds: (None or [float, float]) If not None, specifies the values between which a spine (or spines) will be drawn.

Returns

None

style_xticks(**args*, ***kwargs*)

Formats and styles tick marks along the x-axis.

Parameters

anchor: (int) Anchors tick marks at a specific number. Even if this number is not within the x-axis limits, it fixes the register for tick marks.

spacing: (int > 0) The spacing between adjacent tick marks

fmt: (str) String used to format tick labels.

rotation: (number) Angle, in degrees, with which to draw tick mark labels.

****kwargs:** Additional keyword arguments to be passed to ax.set_xticklabels()

Returns

None

4.2.2 Glyph class

class logomaker.**Glyph**(***kwargs*)

A Glyph represents a character, drawn on a specified axes at a specified position, rendered using specified styling such as color and font_name.

Attributes

p: (number) x-coordinate value on which to center the Glyph.

c: (str) The character represented by the Glyph.

floor: (number) y-coordinate value where the bottom of the Glyph extends to. Must be < ceiling.

ceiling: (number) y-coordinate value where the top of the Glyph extends to. Must be > floor.

ax: (matplotlib Axes object) The axes object on which to draw the Glyph.

width: (number > 0) x-coordinate span of the Glyph.

vpad: (number in [0,1]) Amount of whitespace to leave within the Glyph bounding box above and below the actual Glyph. Specifically, in a glyph with height h = ceiling-floor, a margin of size h*vpad/2 will be left blank both above and below the rendered character.

font_name: (str) The name of the font to use when rendering the Glyph. This is the value passed as the 'family' parameter when calling the matplotlib.font_manager.FontProperties constructor.

font_weight: (str or number) The font weight to use when rendering the Glyph. Specifically, this is the value passed as the 'weight' parameter in the matplotlib.font_manager.FontProperties constructor. From matplotlib documentation: "weight: A numeric value in the range 0-1000 or one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'."

color: (matplotlib color) Color to use for Glyph face.

edgecolor: (matplotlib color) Color to use for Glyph edge.

edgewidth: (number >= 0) Width of Glyph edge.

dont_stretch_more_than: (str) This parameter limits the amount that a character will be horizontally stretched when rendering the Glyph. Specifying a wide character such as 'W' corresponds to less potential stretching, while specifying a narrow character such as '.' corresponds to more stretching.

flip: (bool) If True, the Glyph will be rendered upside down.

mirror: (bool) If True, a mirror image of the Glyph will be rendered.

zorder: (number) Placement of Glyph within the z-stack of ax.

alpha: (number in [0,1]) Opacity of the rendered Glyph.

figsize: ([float, float]): The default figure size for the rendered glyph; only used if ax is not supplied by the user.

Methods

draw(self)	Draws Glyph given current parameters.
set_attributes(self, **kwargs)	Safe way to set the attributes of a Glyph object

draw (*self*)
Draws Glyph given current parameters.

Parameters

None.

Returns

None.

set_attributes (*self*, ***kwargs*)
Safe way to set the attributes of a Glyph object

Parameters

****kwargs:** Attributes and their values.

4.2.3 matrix functions

logomaker.**transform_matrix**(**args*, ***kwargs*)

Performs transformations on a matrix. There are three types of transformations that can be performed:

1. **Center values:** Subtracts the mean from each row in df. This is common for weight matrices or energy matrices. To do this, set center_values=True.
2. **Normalize values:** Divides each row by the sum of the row. This is needed for probability matrices. To do this, set normalize_values=True.
3. **From/To transformations:** Transforms from one type of matrix (e.g. 'counts') to another type of matrix (e.g. 'information'). To do this, set from_type and to_type arguments.

Here are the mathematical formulas invoked by From/To transformations:

from_type='counts' -> to_type='probability': P_ic = (N_ic + l)/(N_i + C*l), N_i = sum_c(N_ic)
from_type='probability' -> to_type='weight': W_ic = log_2(P_ic / Q_ic)
from_type='weight' -> to_type='probability': P_ic = Q_ic * 2^(W_ic)
from_type='probability' -> to_type='information': I_ic = P_ic * sum_d(P_id * log2(P_id / W_id))
from_type='information' -> to_type='probability': P_ic = I_ic / sum_d(I_id)
notation: i = position c, d = character l = pseudocount C = number of characters N_ic = counts matrix element P_ic = probability matrix element Q_ic = background probability matrix element W_ic = weight matrix element I_ic = information matrix element

Using these five 1-step transformations, 2-step transformations are also enabled, e.g., from_type='counts' -> to_type='information'.

Parameters

df: (dataframe) The matrix to be transformed.

center_values: (bool) Whether to center matrix values, i.e., subtract the mean from each row.

normalize_values: (bool) Whether to normalize each row, i.e., divide each row by the sum of that row.

from_type: (str) Type of input matrix. Must be one of 'counts', 'probability', 'weight', or 'information'.

to_type: (str) Type of output matrix. Must be one of 'probability', 'weight', or 'information'. Can be 'counts' ONLY if from_type is 'counts' too.

background: (array, or df) Specification of background probabilities. If array, should be the same length as df.columns and correspond to the probability of each column's character. If df, should be a probability matrix the same shape as df.

pseudocount: (number >= 0) Pseudocount to use when transforming from a counts matrix to a probability matrix.

Returns

out_df: (dataframe) Transformed matrix

logomaker.**sequence_to_matrix**(**args*, ***kwargs*)

Generates a matrix from a sequence. With default keyword arguments, this is a one-hot-encoded version of the sequence provided. Alternatively, is_iupac=True allows users to get matrix models based in IUPAC motifs.

Parameters

seq: (str) Sequence from which to construct matrix.

cols: (str or array-like or None) The characters to use for the matrix columns. If None, cols is constructed from the unqiue characters in seq. Overriden by alphabet and is_iupac.

alphabet: (str or None) The alphabet used to determine the columns of the matrix. Options are: 'dna', 'rna', 'protein'. Ignored if None. Overrides cols.

is_iupac: (bool) If True, it is assumed that the sequence represents an IUPAC DNA string. In this case, cols is overridden, and alphabet must be None.

to_type: (str) The type of matrix to output. Must be 'probability', 'weight', or 'information'

center_weights: (bool) Whether to subtract the mean of each row, but only if to_type='weight'.

Returns

seq_df: (dataframe) the matrix returned to the user.

logomaker.**alignment_to_matrix**(**args*, ***kwargs*)

Generates matrix from a sequence alignment

Parameters

sequences: (list of strings) A list of sequences, all of which must be the same length

counts: (None or list of numbers) If not None, must be a list of numbers the same length os sequences, containing the (nonnegative) number of times that each sequence was observed. If None, defaults to 1.

to_type: (str) The type of matrix to output. Must be 'counts', 'probability', 'weight', or 'information'

background: (array, or df) Specification of background probabilities. If array, should be the same length as df.columns and correspond to the probability of each column's character. If df, should be a probability matrix the same shape as df.

characters_to_ignore: (str) Characters to ignore within sequences. This is often needed when creating matrices from gapped alignments.

center_weights: (bool) Whether to subtract the mean of each row, but only if to_type=='weight'.

pseudocount: (number >= 0.0) Pseudocount to use when converting from counts to probabilities.

Returns

out_df: (dataframe) A matrix of the requested type.

logomaker.**saliency_to_matrix**(**args*, ***kwargs*)

Takes a sequence string and an array of values values and outputs a values dataframe. The returned dataframe is a L by C matrix where C is the number ofcharacters and L is sequence length. If matrix is denoted as S, i indexes positions and c indexes characters, then S_ic will be non-zero (equal to the value in the values array at position p) only if character c occurs at position p in sequence. All other elements of S are zero.

example usage:

saliency_mat = logomaker.saliency_to_matrix(sequence,values) logomaker.Logo(saliency_mat)

Parameters

seq: (str or array-like list of single characters) sequence for which values matrix is constructed

values: (array-like list of numbers) array of values values for each character in sequence

cols: (str or array-like or None) The characters to use for the matrix columns. If None, cols is constructed from the unqiue characters in seq. Overridden by alphabet and is_iupac.

alphabet: (str or None) The alphabet used to determine the columns of the matrix. Options are: 'dna', 'rna', 'protein'. Ignored if None. Overrides cols.

Returns

saliency_df: (dataframe) values matrix in the form of a dataframe

logomaker.**validate_matrix**(**args*, ***kwargs*)

Checks to make sure that the input dataframe, df, represents a valid matrix, i.e., an object that can be displayed as a logo.

Parameters

df: (dataframe) A pandas dataframe where each row represents an (integer) position and each column represents to a (single) character.

matrix_type: (None or str) If 'probability', validates df as a probability matrix, i.e., all elements are in [0,1] and rows are normalized). If 'information', validates df as an information matrix, i.e., all elements >= 0.

allow_nan: (bool) Whether to allow NaN entries in the matrix.

Returns

out_df: (dataframe) A cleaned-up version of df (if possible).

4.2.4 dataset functions

logomaker.**demo**(**args*, ***kwargs*)

Performs a demonstration of the Logomaker software.

Parameters

name: (str) Must be one of {'fig1b', 'fig1c', 'fig1d', 'fig1e', 'fig1f', 'logo'}.

Returns

None.

logomaker.**list_example_matrices**(**args*, ***kwargs*)

Return list of available matrices.

logomaker.**get_example_matrix**(**args*, ***kwargs*)

Returns an example matrix from which a logo can be made.

Parameters

name: (None or str) Name of example matrix.

print_description: (bool) If true, a description of the example matrix will be printed

Returns

df: (data frame) A data frame containing an example matrix.

logomaker.**list_example_datafiles**(**args*, ***kwargs*)

Return list of available data files.

logomaker.**open_example_datafile**(**args*, ***kwargs*)

Returns a file handle to an example dataset

Parameters

name: (None or str) Name of example matrix.

print_description: (bool) If true, a description of the example matrix will be printed

Returns

f: (file handle) A handle to the requested file

4.2.5 functional tests

logomaker.**run_tests**()

Run all Logomaker functional tests. There are 547 tests as of 14 May 2019.

Parameters

None.

CHAPTER 5

Reference

CHAPTER 6

Contact

For technical assistance or to report bugs, please contact Ammar Tareen (Email: tareen@cshl.edu, Twitter: @AmmarTareen1) . For more general correspondence, please contact Justin Kinney (Email: jkinney@cshl.edu, Twitter: @jbkinney).

CHAPTER 7

Links

- Logomaker preprint on bioRxiv
- Logomaker on GitHub
- Logomaker on PyPI
- Kinney Lab
- Cold Spring Harbor Laboratory

Index