Create Plots

April 3, 2017

Notebook for creating several Plots

- Sum of Squares plot
- Many numerical fingerprints
- Example tournament and results

In [ ]:
```
import string
import numpy as np
import axelrod as axl
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
import matplotlib
```

In [ ]:
```
def format_filename(s):
    """
    Take a string and return a valid filename constructed from the string.
    Uses a whitelist approach: any characters not present in valid_chars ar
    removed. Also spaces are replaced with underscores.
    Note: this method may produce invalid filenames such as ``, `.` or `..`
    When I use this method I prepend a date string like '2009_01_15_19_46_3
    and append a file extension like '.txt', so I avoid the potential of us
    an invalid filename.
    Borrowed from https://gist.github.com/seanh/93666
    """
    valid_chars = "-_.() {}{}".format(string.ascii_letters, string.digits)
    filename = ''.join(c for c in s if c in valid_chars)
    filename = filename.replace(' ','_')
    return filename
```

In [ ]:
```
strats = axl.strategies

strategies = [s.name for s in strats]
# need access to the fingerprint csv files from Axelrod-fingerprint repo
path = '/Users/James/Projects/Axelrod-fingerprint/assets/'
filenames = [path + format_filename(s) + '.csv' for s in strategies]
dataframes = [pd.read_csv(f) for f in filenames]
```

```
        def score_for_df(A, B):
            """
            Compute the sum of squares score for two dataframes, A and B
            """
            result = pd.merge(A, B, on=['x', 'y'], suffixes=('_A', '_B'))
            result['SQ_difference'] = (result['score_A'] - result['score_B'])**2
            result.drop(['score_A', 'score_B'], axis=1, inplace=True)
            return sum(result.SQ_difference)


        sum_squares_df = pd.DataFrame(index=strategies, columns=strategies)

        for indexA, strategyA in enumerate(tqdm(strategies)):
            A_df = dataframes[indexA]
            for indexB, strategyB in enumerate(strategies):
                B_df = dataframes[indexB]
                similarity_score = score_for_df(A_df, B_df)
                sum_squares_df.set_value(strategyA, strategyB, similarity_score)

        sum_squares_df
```

In [ ]:
```
        sum_squares_df = sum_squares_df.apply(pd.to_numeric)
        size = len(dataframes[0].index)
        mean_squares_df = sum_squares_df.divide(size)
        mean_squares_df.head()
```

In [ ]:
```
        plt.figure(figsize=(50, 75))
        sns.heatmap(mean_squares_df, cbar_kws={"orientation": "horizontal"})
        cbar = ax.figure.colorbar(ax.collections[0])
        cbar.set_ticks([0, 1])
        cbar.set_ticklabels(["0%", "100%"])
        # cbar.set_ticks([0, .2, .75, 1])
        # cbar.set_ticklabels(['low', '20%', '75%', '100%'])
        # fig.colorbar(ax, orientation="horizontal",fraction=0.07,anchor=(1.0,0.0))
        plt.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/mean_squa
        plt.show()

        # f, ax = plt.subplots(figsize=(50, 50))
        # sns.heatmap(sum_squares_df, cbar=False)
```

In [ ]:

In [ ]:
```
        axl.seed(0)  # Set a seed
        players = [axl.TitForTat(), axl.Cooperator(), axl.Random(), axl.Gradual()]
        tournament = axl.Tournament(players)  # Create a tournament
        results = tournament.play()  # Play the tournament
        plot = axl.Plot(results)
```

```
        p = plot.boxplot()
        q = plot.payoff()
```

```
In [ ]: p.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/examples/sm
        p
```

```
In [ ]: q.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/examples/sm
        q
```

1 Analytical Plots

```
In [ ]: def TFT(coord):
            x, y = coord
            numerator = y**2 + 5*x*y + 3*x**2
            denominator = (x + y)**2
            return numerator/denominator

        def WSLS(coord):
            x, y = coord
            numerator = (3*x + y)*(x - 1) + 5*y*(y - 1)
            denominator = (x + 2*y)*(x - 1) + y*(y - 1)
            return numerator/denominator

        def Psycho(coord):
            x, y = coord
            numerator = 4*(y - 1)*(x - 1) + 5*(y - 1)**2
            denominator = 2*(y - 1)*(x - 1) + (x - 1)**2 + (y - 1)**2
            return numerator/denominator

        def Coop(coord):
            x, y = coord
            return 3 - 3*y

        def Defect(coord):
            x, y = coord
            return 4*x + 1
```

```
In [ ]: from collections import namedtuple

        Point = namedtuple('Point', 'x y')

        def reshape_data(data, points, size):
            """Shape the data so that it can be plotted easily.
            Parameters
            ----------
            data : dictionary
                A dictionary where the keys are Points of the form (x, y) and
                the values are the mean score for the corresponding interactions.
```

```
        points : list
            of Point objects with coordinates (x, y).
        size : int
            The number of Points in every row/column.
        Returns
        ----------
        plotting_data : list
            2-D numpy array of the scores, correctly shaped to ensure that the
            score corresponding to Point (0, 0) is in the left hand corner ie.
            the standard origin.
        """
        ordered_data = [data[point] for point in points]
        shaped_data = np.reshape(ordered_data, (size, size), order='F')
        plotting_data = np.flipud(shaped_data)
        return plotting_data


    def create_points(step, progress_bar=False):
        """Creates a set of Points over the unit square.
        A Point has coordinates (x, y). This function constructs points that ar
        separated by a step equal to `step`. The points are over the unit
        square which implies that the number created will be (1/`step` + 1)^2.
        Parameters
        ----------
        step : float
            The separation between each Point. Smaller steps will produce more
            Points with coordinates that will be closer together.
        progress_bar : bool
            Whether or not to create a progress bar which will be updated
        Returns
        ----------
        points : list
            of Point objects with coordinates (x, y)
        """
        num = int((1 / step) // 1) + 1

        if progress_bar:
            p_bar = tqdm(total=num ** 2, desc="Generating points")

        points = []
        for x in np.linspace(0, 1, num):
            for y in np.linspace(0, 1, num):
                points.append(Point(x, y))

                if progress_bar:
                    p_bar.update()

        if progress_bar:
```

```
            p_bar.close()

        return points


    def plot(plotting_data, col_map='seismic', interpolation='none', title=None,
             colorbar=True, labels=True):
        """Plot the results of the spatial tournament.
        Parameters
        ----------
        col_map : str, optional
            A matplotlib colour map, full list can be found at
            http://matplotlib.org/examples/color/colormaps_reference.html
        interpolation : str, optional
            A matplotlib interpolation, full list can be found at
            http://matplotlib.org/examples/images_contours_and_fields/interpola
        title : str, optional
            A title for the plot
        colorbar : bool, optional
            Choose whether the colorbar should be included or not
        labels : bool, optional
            Choose whether the axis labels and ticks should be included
        Returns
        ----------
        figure : matplotlib figure
            A heat plot of the results of the spatial tournament
        """
        fig, ax = plt.subplots()
        cax = ax.imshow(
            plotting_data, cmap=col_map, interpolation=interpolation)

        if colorbar:
            max_score = np.nanmax(plotting_data)
            min_score = np.nanmin(plotting_data)
            ticks = [min_score, (max_score + min_score) / 2, max_score]
            fig.colorbar(cax, ticks=ticks)

        plt.xlabel('$x$')
        plt.ylabel('$y$', rotation=0)
        ax.tick_params(axis='both', which='both', length=0)
        plt.xticks([0, len(plotting_data) - 1], ['0', '1'])
        plt.yticks([0, len(plotting_data) - 1], ['1', '0'])

        if not labels:
            plt.axis('off')

        if title is not None:
            plt.title(title)
```

```
        return fig
```

In [ ]:
```
step=0.01
size = int((1 / step) // 1) + 1
points = create_points(step)
```

In [ ]:
```
TFT_Data = {p: TFT(p) for p in points}
WSLS_Data = {p: WSLS(p) for p in points}
Psycho_Data = {p: Psycho(p) for p in points}
Coop_Data = {p: Coop(p) for p in points}
Defect_Data = {p: Defect(p) for p in points}
```

In [ ]:
```
TFT_Data = reshape_data(TFT_Data, points, size)
WSLS_Data = reshape_data(WSLS_Data, points, size)
Psycho_Data = reshape_data(Psycho_Data, points, size)
Coop_Data = reshape_data(Coop_Data, points, size)
Defect_Data = reshape_data(Defect_Data, points, size)
```

In [ ]:
```
np.nanmax(TFT_Data)
```

In [ ]:
```
TFT_plot = plot(TFT_Data)
WSLS_plot = plot(WSLS_Data)
Psycho_plot = plot(Psycho_Data)
Coop_plot = plot(Coop_Data)
Defect_plot = plot(Defect_Data)
```

In [ ]:
```
TFT_plot.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/Anal
WSLS_plot.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/Ana
Psycho_plot.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/A
Coop_plot.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/Ana
Defect_plot.savefig('/Users/James/Projects/FinalYearReport-Manuscript/img/A
```