MATH 497 (Fall 2020)

Algorithmic Redistricting using Reinforcement Learning

Final Report

Illinois Institute of Technology

Authors

Divyani Audichya

Chinnies Chibuko

Sohan Puthran

Zawad Sadaf

Instructors

Dr. Robert Ellis,
Associate Professor
Illinois Institute of Technology

Emily Webber,
Machine Learning Specialist
AWS

ALGORITHMIC REDISTRICTING

Introduction

This project is addressing unethical redistricting plans. This may dilute certain demographic group's voting power, which is an important problem as it infringes upon the Voting Rights Act of 1965 which protects minority group's votes. Also, the bias involved in the redistricting plan is an unfair use of power to weaken the individual's vote; which is highly unethical and borderline predatory. Working on this problem will provide us with the opportunity to create a more neutral districting plan through deep reinforcement learning algorithms since it is not being swayed by human biases.

Gerrymandering is one of the most significant issues in US politics, and it's a substantial problem for civic interests. The district maps can be drawn in a partisan manner that may violate the constitutional rights by shifting the borders to maximize the chances of success for their desired representatives.

This is a form of voter suppression that needs to be removed and our reinforcement learning model can be an effective tool that can reduce the chances of Gerrymandering and draws a fairer district map. Since the model learns by itself, it can reduce biases and may not favor any political party.

We are using Iowa state as the main focus throughout this project. We will use our fairness metrics to figure out how fair the current map of Iowa is. Then, we will build a Deep Q-Network that will give us a fairer redistricting plan that will reduce any kind of Gerrymandering.

Data

The data used for this project can be found at Iowa Counties. In the exploratory data analysis section, we show how this data was analysed.

Exploratory Data Analysis

We first performed an exploratory data analysis on the Iowa dataset. This analysis was done through the use of the geopandas library to read the dataset. The shapefile visualization was then obtained by making use of matplotlib library to plot the resulting Iowa dataframe. Snippet of the code used is shown below:

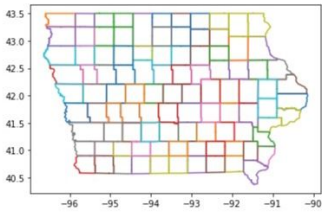
```
In [1]: import geopandas as gp

In [2]: import shapefile as shp

In [3]: import matplotlib.pyplot as plt

In [4]: sf = shp.Reader("IA_counties.shp")

In [5]: plt.figure()
        for shape in sf.shapeRecords():
            x = [i[0] for i in shape.shape.points[:]]
            y = [i[1] for i in shape.shape.points[:]]
            plt.plot(x,y)
        plt.show()
```




```
In [6]: IA = gp.read_file("IA_counties.shp")

In [21]: IA.shape
Out[21]: (99, 57)

In [22]: IA.head()
```

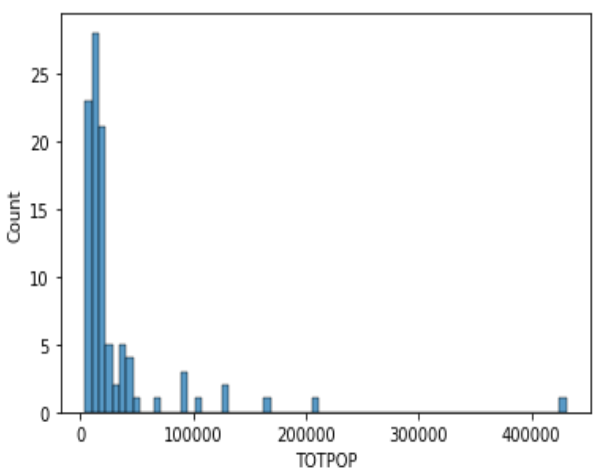
Out[22]:

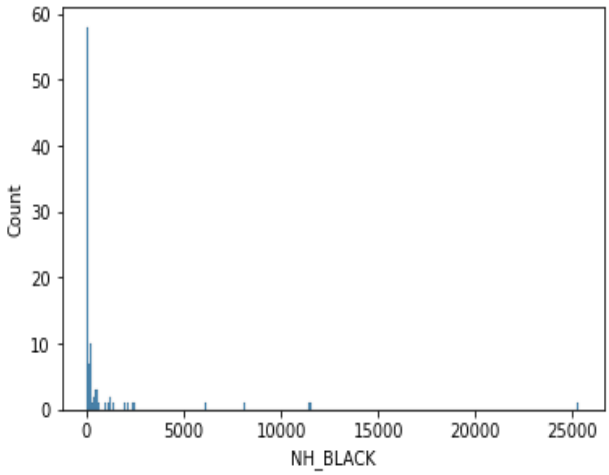
	STATEFP10	COUNTYFP10	GEOID10	NAME10	NAMELSAD10	ALAND10	AWATER10	INTPTLAT10	INTPTLON10	TOTPOP	...	TOTVOT12	PRES12D	PR
0	19	127	19127	Marshall	Marshall County	1482770678	1803086	+42.0416910	-092.9814523	40648	...	19064	10257	
1	19	011	19011	Benton	Benton County	1855117342	5760770	+42.0925474	-092.0576300	26076	...	14023	6862	
2	19	041	19041	Clay	Clay County	1469139214	13866941	+43.0798220	-095.1497261	16667	...	8502	3385	
3	19	165	19165	Shelby	Shelby County	1530110414	1486135	+41.6790143	-095.3089173	12167	...	6483	2469	

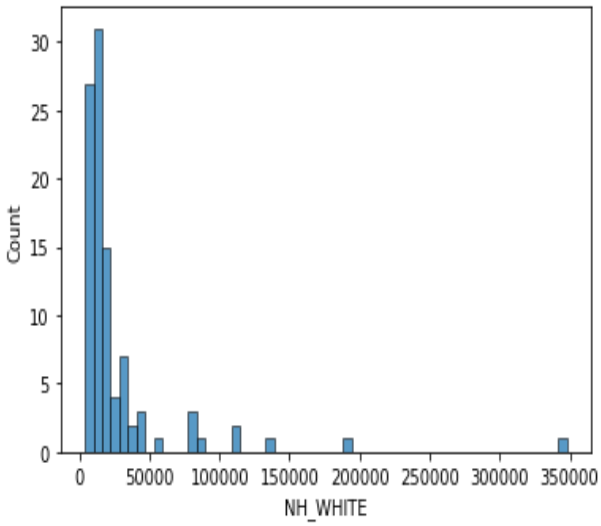
```
In [7]: fig, ax = plt.subplots(figsize = (20,20))
        IA.plot(ax = ax,cmap = 'jet')
        plt.show()
```

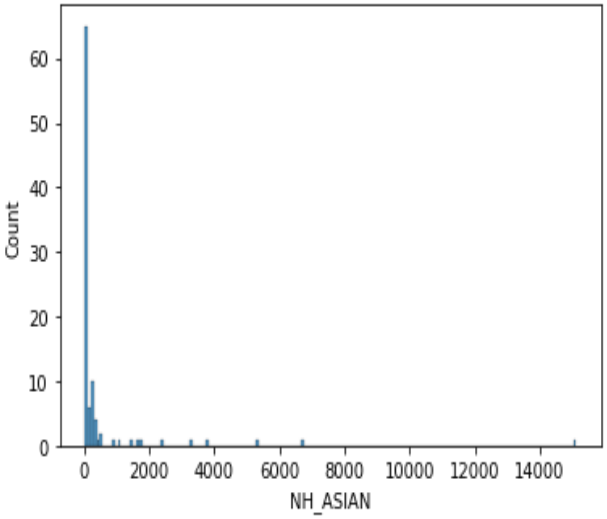
ALGORITHMIC REDISTRICTING

Next, we made use of histograms to show the distribution of the various ethnicities demographics. The histograms are shown below:









As you can see from the above figures, the histograms show the number of counties in Iowa with that specific population. The histogram clearly shows us that the white population is the majority in Iowa and we can also see how miniscule the black and asian population are.

These visualizations above help show the distributions of various ethnicities and the number of counties in Iowa with a particular population for a specific ethnicity.

Fairness Metrics

We did different kinds of visualization and analysis on the Iowa data. Now, we will look into our definition of fairness metrics for the Reinforcement Learning model.

Fairness metrics helps us to understand whether the redistricting has been done fairly or not. We are thankful to the Fairness Metrics team for providing us with the below fairness metric equation. They used three popular fairness metrics to generate this equation. Three metrics that they used are Efficiency Gap, Partisan Bias and Mean-Median Difference. The equation is given below:

$$FM = e^{-k\sum(\frac{x_i}{m_i})^2}$$

[Source]

The values for the variables in the equation are as follows:
$k = 0.2231$, $m_1 = 0.0861$, $m_2 = 0.1210$, $m_3 = 0.0293$, x_1 = Efficiency Gap, x_2 = Partisan Bias and x_3 = Mean-Median Difference.

Our implementation for this equation:

```
def efficiency_gap(data):

    wasteD = sum(data['d_wasted'])
    wasteR = sum(data['r_wasted'])

    return  (wasteD - wasteR) / len(data)


def partisan_bias(data):
    total_votes_d = sum(data['d_votes'])
    total_votes_r = sum(data['r_votes'])

    shift = (total_votes_d - total_votes_r) / (2 * len(data))

    data['d_wasted'] = data['d_wasted'] - shift
    data['r_wasted'] = data['r_wasted'] - shift

    return efficiency_gap(data)

def mean_median_difference(data):

    total_votes_d = sum(data['r_votes'])  # The value will not change even if we consider d_votes
    average = total_votes_d / len(data)
    med = statistics.median(data['r_votes'])

    return (med - average) / len(data)

def fairness_metric(data) :

    eg = efficiency_gap(data)
    pb = partisan_bias(data)
    mmd = mean_median_difference(data)
    value = -0.2231 * ((eg / 0.0861) + (pb / 0.1210) + (mmd / 0.0293)) ** 2

    return math.exp(value)
```

ALGORITHMIC REDISTRICTING

These functions take votes and wasted votes from both parties as inputs and give fairness metrics. We calculated wasted votes of Iowa elections like this:

```
req_data['d_wasted'] = req_data['d_votes']
req_data['r_wasted'] = req_data['r_votes']
for index in range(len(req_data)):
    if req_data['d_votes'][index] > 0.5:
        req_data['d_wasted'][index] = req_data['d_votes'][index] - 0.5
        req_data['r_wasted'][index] = req_data['r_votes'][index]

    elif req_data['r_votes'][index] > 0.5:
        req_data['r_wasted'][index] = req_data['r_votes'][index] - 0.5
        req_data['d_wasted'][index] = req_data['d_votes'][index]

req_data.head()
```

	d_votes	r_votes	CD	d_wasted	r_wasted
0	0.425584	0.508676	1	0.425584	0.008676
1	0.337908	0.594626	1	0.337908	0.094626
2	0.260996	0.682024	4	0.260996	0.182024
3	0.260911	0.684772	4	0.260911	0.184772
4	0.354584	0.582430	1	0.354584	0.082430

And these are the fairness metric values:

```
Efficiency Gap : 0.13855227368301543
Partisan Bias : -0.0146368267
Mean-Median Difference : 8.31012147971028e-05
Fairness Metric value : 0.3149162131754735
```

Now, we need to use this equation in Reinforcement Learning as a reward.

Reinforcement Learning

Reinforcement Learning is an area of Machine Learning where an agent is trained to maximize some kind of reward in a given environment. The below diagram shows the basic structure of the Reinforcement Learning model.

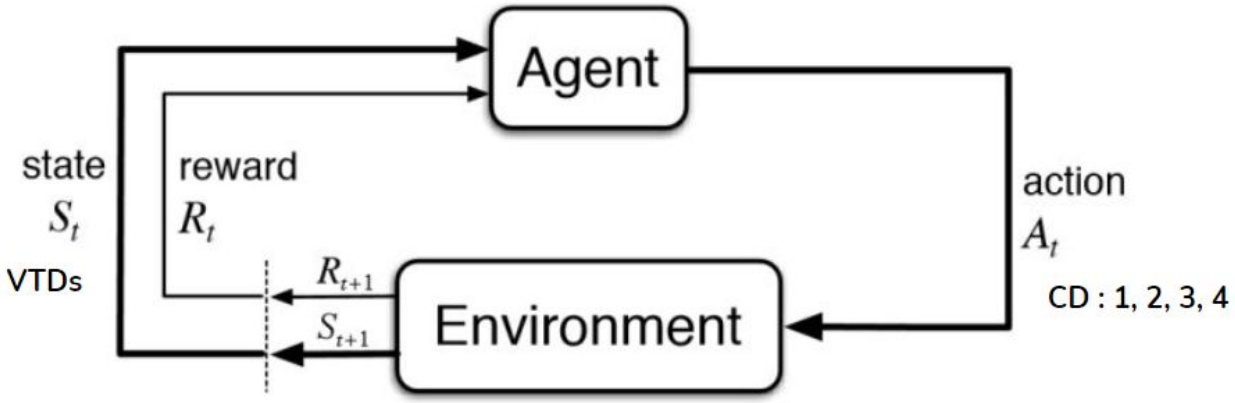


Image Source

In this project, we are trying to classify the VTDs into 4 congressional districts by using fairness metrics to redistrict Iowa counties as fair as possible.

- **Agent** is the decision maker that interacts with the environment it's placed in.
- **Environment** is a set of states that the agent attempts to influence its choice of actions.
- **State** in the environment is the VTD of the data. Since we have 99 VTDs, we will have 99 states for the Reinforcement Learning model.
- **Action** of an agent is the congressional districts. Since Iowa has 4 congressional districts, we will have 4 actions for the agent (1,2,3,4).
- The agent receives a **reward** based on the action it took. Fairness metric can be used as the reward for this RL model.

The agent's goal is to maximize the reward. The agent will try to maximize the fairness metrics value by distributing the VTDs into 4 congressional districts which will generate a fairer redistricting plan and will not have gerrymandering.

Since we were not sure about the correct approach for this Reinforcement Learning model, we implemented two different approaches for this project: Basic Q-Learning approach and Deep Q-Network which uses a neural network in its implementation.

Q-Learning

The objective of Q-Learning is to achieve the maximum expected value of the total reward over all successive steps. It created a Q-table and tries to find the optimal value for each state-action pair.

Q-Learning uses the Bellman optimality equation for q_* for generating the values for the q-table.

$$q_*(s,a) = R_{t+1} + \gamma max_{a'} q_*(s',a')$$

Bellman Optimality Equation

This equation states that, for any given pair (s,a) at time t, the expected return from starting in state s, selecting action a and following the Q-value of the pair is going to be the expected reward we get from taking action a in state s, which is R_{t+1}, plus the maximum expected discounted return that can be achieved from any possible next state-action pair. [Source]

		Actions			
		a_1	a_2	a_3	a_4
States	s_1	0	0	0	0
	s_2	0	0	0	0
	s_3	0	0	0	0
	s_4	0	0	0	0
	⋮	...	...	...	...
	s_{99}	0	0	0	0

Our Q-Learning algorithm creates a Q-table as shown in the above table where the rows are different states and columns are the possible actions. Initially, all the action-pair values are zeros but they get updated as the agent goes through several episodes. During later episodes, the agent looks at the updated Q-table and bases its next action on the highest Q-value for the current state. Since all the Q-values are zeros on the first episode, the agent cannot differentiate between the Q-values to discover which one is best. To overcome this, we need to introduce a trade-off between *exploration* and *exploitation*.

We will be using this trade-off throughout the entire training process and it helps the agent to choose actions in a better way.
Exploration is the act of exploring the environment and Exploitation is the act of exploiting the information that is already available to maximize the return.

During the initial episodes, the agent needs to explore the environment as much as possible and during the last episodes, the agent needs to exploit the information that it knows to maximize the returns. If the exploration is not done properly, the agent will keep on repeating the same set of actions where it received good returns and may miss out on gaining more rewards since it did not explore the entire environment.

In order to introduce a balanced trade-off between exploration and exploitation, we need to use *epsilon greedy strategy*.

In *epsilon greedy strategy,* we use a exploration rate ϵ which is initialised to 1. By $\epsilon = 1$, the agent focuses on exploration during the initial episodes. As the agent goes through more episodes, the ϵ value decays by a delay constant. A significant decrease in ϵ value signals the agent to focus on exploitation rather than exploration.

Now, by including a learning rate (α) into our q-learning equation, we get the following equation:

$$q^{new}(s,a) = (1 - \alpha)\, q(s,a) + \alpha(R_{t+1} + \gamma max_{a'}\, q'(s',a'))$$

Using $\alpha = 0.8$ and γ(*discount rate*) $= 0.9$, we can calculate the q-values during the initial episode as follows:

The q(s,a) and $q'(s',a')$ will be zero for all action-state pairs initially. Let's say, we get a reward (R_{t+1}) for (s_1,a_a) as 0.67 which is the fairness value.

Now, the equation will be:

$$q^{new}(s_1,a_1) = (1 - 0.8)\,(0) + (0.8)\,(0.67 + (0.9)(0))$$
$$q^{new}(s_1,a_1) = 0 + 0.536$$
$$q^{new}(s_1,a_1) = 0.536$$

So, the updated Q-table will be:

States	Actions			
	a_1	a_2	a_3	a_4
s_1	0.536	0	0	0
s_2	0	0	0	0
s_3	0	0	0	0
s_4	0	0	0	0
...	...	...	...	...
s_{99}	0	0	0	0

We can summarise the algorithm like this:

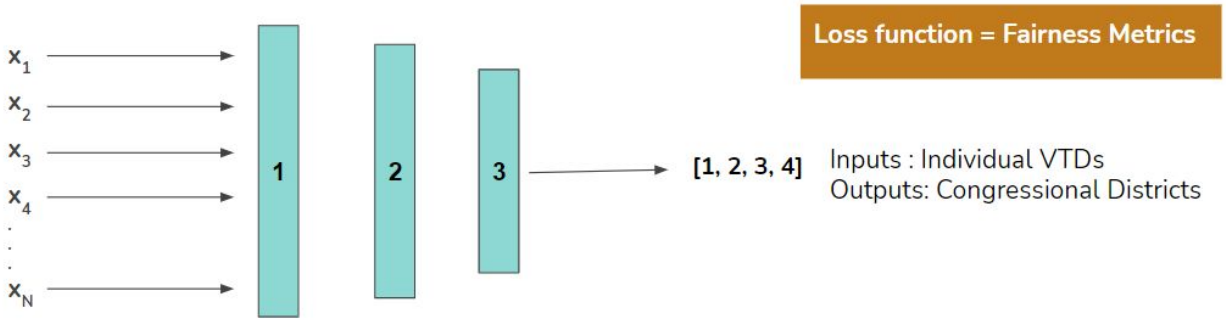
1. Initialize all Q-values in the Q-table to 0.
2. For each time-step in each episode:
    a. Choose an action (*exploration* and *exploitation*).
    b. Observe the reward and next state.
    c. Update the Q-value function using the updated Q-values equation given above.

We could not build the required model for this project because we could not implement the custom environment that is needed.
Since the Q-learning approach is an iterative approach, the time to update the q-values will increase as the number of states or actions increases. We have 99 states and 4 actions which gives us 396 action-state pairs which will take a lot of time to process.
So, we need a different approach like introducing the neural network into our Q-Learning approach. This approach is called the *Deep Q-Network* approach.

ALGORITHMIC REDISTRICTING

Deep Q-Network

In this approach, we are introducing the below neural network to our implementation.



We are using a simple neural network with just three layers: Input layer, hidden layer and output layer. The network takes features of a state as input and classifies it into one of the 4 congressional districts. It uses a custom loss function that uses the fairness metrics that we defined earlier. The agent uses this network to perform actions and uses the loss function as reward.

We can summarise the algorithm like this:

1. Initialize the neural network with random weights.
2. For each episode:
    a. Initialize the state to a VTD.
    b. For each time step:
        i. Select an action (Exploration or Exploitation)
        ii. Execute the action on the environment
        iii. Get the reward and next state.
        iv. Calculate the loss.
        v. Generate the gradient descent weights to minimize the loss.

We were unsuccessful in building a working model because we failed to create a custom environment for this project even though we tried many times.

Conclusions

- Using the fairness metrics, we were able to figure out the current Iowa map is not fair.
- We were able to properly understand the Iowa data by performing various kinds of exploratory data analysis.
- Made a few improvements on the fairness metric equation given by the fairness metric team so that we can properly use it on our model.
- Tried to build the custom environment for our project but could not get the model to work.

Future work

- Build a proper working environment for the Deep Q-Network.
- Improve the Fairness Metric to include more metrics and not just three metrics that we are used here.
- Develop an efficient redistricting model that can be used for redistricting by removing any kind of Gerrymandering.
- Create the map after redrawing the Iowa districts.
- Work on other states where Gerrymandering has happened and create a fairer redistricting map.

References

- https://deeplizard.com/learn/playlist/PLZbbT5o_s2xrfNyHZsM6ufI0iZENK9xgG
- https://medium.com/@shashikachamod4u/excel-csv-to-pytorch-dataset-def496b6bcc1
- https://mygeodata.cloud/converter/shp-to-csv
- https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html
- https://deeplizard.com/learn/video/nyjbcRQ-uQ8
- https://medium.com/ixorthink/using-deep-q-learning-in-the-classification-of-an-imbalanced-dataset-22ee5e868efc
- Reinforcement Learning for Meal Planning in Python | Kaggle
- https://github.com/mggg-states/IA-shapefiles