Computational Gerrymandering

Members: Mac, Jessica, Casmali, Liz

Description: This project aimed to do three things

1) Generate gerrymandered and neutral training data
2) Train a neural net to label this data accurately
3) Use on real state voting data

The first part of this project we ended up attempting in two ways. First by generating planar graphs, gluing those graphs together using a boundary algorithm and then gerrymandering those graphs. Sadly, this was more difficult than anticipated and this code remains incomplete. It can be found under the title “planar_graphs.ipynb”. In this file, there is code to generate planar graphs, find the outside boundary, glue planar graphs together into a larger planar graph, as well as assign legal districting data to both the precincts and the districts. Although not complete, there is a significant amount of work in this file. We recommend opening it in Google Colaboratory, because the indentations do not transfer neatly to Jupyter Lab.

The second way we attempted this was by using grid graphs. This code can be found under “FINAL_GRID_GRAPHS.ipynb”. This code has the generation of grid graphs and the generation of legal districting data. To generate a sample grid graph and view its division into districts, run cells 1-9. Running up through cell 12 will display numbers indicating how evenly split the generated graph is between political parties and the other characteristics incorporated into the data sets.

This training data is then fed to a pytorch optimizer to gerrymander the graphs using matrix multiplication and gradient freezing to swap nodes between districts to modify the graph to favor one party. The non-gerrymandered and gerrymandered graphs are labeled and sent to the pytorch clustering algorithm, which should weight and identify the node features most important to the specific districting that the graph has. This data is then fed to a neural network for training and classification. This code can be found in “Gerrymandering and detecting.ipynb”.

Finally, you will also find a file entitled “coloradoGraph.ipynb”. This file imports the colorado shapefile and turns it into a dual graph. It then visualizes it in a few different ways. It next has code to set up and prepare the graph to run MCMC analysis. Then, at the end, features are added to the dual graph in the format of the graphs we generated. This means with a short implementation it can be fed to the above neural net for classification.

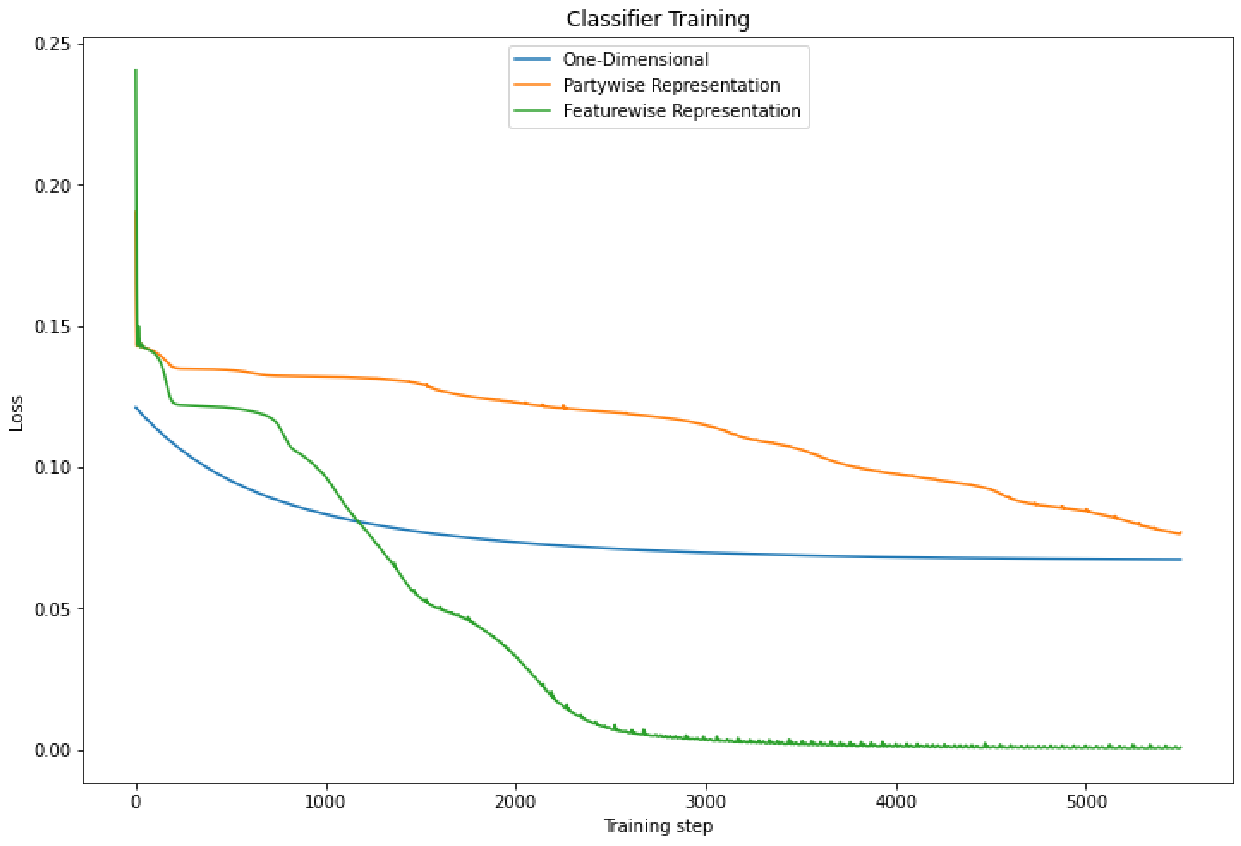
Results and Notes:

Initially, it seemed as though the classifier would not be able to learn anything at all from the data, but after correcting a few technical errors, we did manage to get it to run correctly. We trained three classifier versions, with the structures of each corresponding to three respective sets of data extracted from the partitioned graphs. The results are shown below.

The first model was an extremely simple one-dimensional classifier that took in the ratio of the expected electoral outcome over the statewide partisan breakdown. Essentially, it was just looking for a divergence threshold beyond which it becomes more likely than not that a particular graph was intentionally gerrymandered. You can see from the plot that once the model has found this value, it can't figure out a way to do any better. It would also probably have had a harder time if we had randomized which party was favored in the gerrymandering.

The second model takes in four values: the first two corresponding to the variation in the hypothesis learned by our clustering representation generator when fed only the partisanship data of each precinct, and the last two values representing the total partisanship breakdown in the state (technically one of these last two values is redundant, since the two are complements of each other.) It makes sense that this model would learn more slowly than the one-dimensional version, because the data is not quite as straightforward to interpret, but this also means that it might not have the same floor.

The last model was the same as the second, except the representation learner was fed all of the demographic data of each precinct rather than just the two partisanship values. This model performed the best by a wide margin, but this does not necessarily mean that it can be applied outside the test case—It's highly likely that the high degree of information available for each data point, along with the relatively small dataset (around 120 graphs) gave the model the opportunity to learn a tight overfitting of the sample space. With more time to generate data, it would be a good idea to split the training set into batches.



We sadly did not have the time to run state data, but the information is all in the files and could be implemented quickly, both for gerrymandering state data which would be interesting as well as running an optimized neural net.

Related, the gerrymandering algorithm was a huge and fast success. Further study of its inner workings could allow both legislatures as well as academics to understand how gerrymandering can be done, as well as local minima and maxima of gerrymandering for specific graphs. We hypothesize that this could also be used to learn about structures of graphs and is useful even on its own.

Finally, the issues with gluing together the planar graphs continue to elude us. The infinite loops around the boundaries are a confusing phenomenon. There remain a few avenues of attack and we wonder if more abstract planar graphs would be better to train on given a more complicated structure that might give more uniqueness to district boundaries(maybe spectral analysis could be useful here).

Conclusion:

The main conclusion is that more work is needed, but we have completed significant problem solving including:

1) District Creation in Grid Graphs and Planar Graphs
2) Gerrymandering Algorithm
3) Geometric Clustering Algorithm
4) Neural Network Implementation
5) Planar Graph Generation
6) Legal District Data Generation
7) Boundary Detection on Planar Graphs
8) Importing and Parsing shapefiles
9) Generating Dual Graphs

We hope you enjoy!