# PREDICTING MAX CLIQUE SIZES OF GRAPHS

NILOY DEY

## 1. Problem

Given a graph, a clique of the graph is a subgraph where all pairs of nodes are connected by an edge. The max clique of the graph is a clique with the largest size. Finding the max-clique (and hence max-clique size) of a graph is an NP-Hard problem i.e no polynomial time algorithm exists to solve the problem, and has applications in various fields, for example in social networks (community detection) and biology (protein interaction networks).

The main idea of our project was to curate data of a list of different features of graphs and their max clique sizes, train machine learning models on this data, and use it to find a prediction of the maximum clique size of a graph given it's features. In the process, we make many inferences through the performance of our models and comment on different aspects of machine learning in context of this problem.

## 2. Setup

In our project, we focused on small graphs (sizes $n \leq 100$) since finding clique sizes of larger graphs (for data to train our models) is computationally very expensive. We first generated 5000 samples of random graphs, a list of their features, and calculated their max-clique sizes using a brute-force algorithm.

- The classic method to generate random graphs is to choose a $p \in (0, 1)$ uniformly and generate a graph so that the probability between two nodes having an edge is p.
- Realising that this may bias our data by inducing a linear relationship between the features and clique size, we used many other different graph generation methods and produced a diverse set of graphs.
- For the features of our data, we computed 17 different properties of our graphs (number of nodes/edges, average degree, triangle count, .etc.)

The networkX library has a vast amount of graph generation methods and tools that we used for graph generation and computing graph features. In tabular form, our data looks like this:

| | **X** | | | | **y** |
|---|---|---|---|---|---|
| Graph no. | no. of nodes | no. of edges | ... | avg degree | Max-clique size |
| 1 | $X_{1,1}$ | $X_{1,2}$ | ... | $X_{1,17}$ | $Y_1$ |
| 2 | $X_{2,1}$ | $X_{2,2}$ | ... | $X_{2,17}$ | $Y_2$ |
| ... | ... | ... | ... | ... | ... |
| 5000 | $X_{5000,1}$ | $X_{5000,2}$ | ... | $X_{5000,17}$ | $Y_{5000}$ |

A note on bias in our data: Since our graph sizes are relatively small, there could possibly be low variance in the data that we chose. Regardless, there are $2^{(100 \cdot 99)/2}$ possible graphs of size 100, and we tried to do our best to select diverse graphs for our samples.

## 3. Machine Learning Models

We split our data to a train and test set in the ratio 1 : 2 respectively. We trained and evaluated our data on three machine learning models that we believed were appropriate for our problem:

- **Linear Regression** We included this as a baseline. Although clique size is inherently non-linear with respect to graph structure, we hypothesized that certain summary features (e.g., average degree, number of edges) might exhibit quasi-linear correlations in small graphs.
- **Neural Network (MLP)** Given the non-linear nature of the problem, we expected a neural network to outperform linear models by learning complex feature interactions. We believed that, with sufficient tuning, the MLP would generalize well despite the limited sample size.

- **Random Forest** This model was chosen for its ability to capture non-linear patterns without requiring intensive hyperparameter tuning. We expected it to perform well, especially due to the discrete, combinatorial nature of graph properties and clique structure.

Initially, all our models applied regression. We realised that since the max-clique size of small graphs is usually small, we could also consider classification to see if it does any better. We first used the machine learning models from sklearn library, then wrote our own implementation of MLP and Random Forest to improve on our results.

## 4. Results

We used mean-squared error (MSE) and accuracy (after rounding values of regressors) to evaluate our model performance. In general, we had good results on all our models:

| | **Regressors** | | | **Classifiers** | |
|---|---|---|---|---|---|
| **Metric** | **MLP** | **Linear** | **Random Forest** | **MLP** | **Random Forest** |
| **MSE** | 0.4378 | 0.2177 | 0.1824 | 0.6184 | 0.2478 |
| **Accuracy (%)** | 59.7 | 72.2 | 77.0 | 57.4 | 76.4 |

This means that, for example, the Random Forest Regressor predicted the max-clique size of a graph off by $\sqrt{0.19} \approx 0.43$ on average on the test data. The results above are after finding optimal parameters (regularization/hidden layers/max-iterations for MLP, no. of estimators for random forests) by plotting our results against the different parameters.

It was interesting to see that the classifiers almost always did worse than the regressors. This could perhaps be explained by there being too many different output classes and/or the difference of the loss functions. Some comments on the performance of models:

- It was initially surprising that our Linear Regression performed performed well and always better than the MLP, even after a lot of hyperparameter tuning of the MLP. Either there is perhaps a more linear relationship between the graph features we chose and the max-clique size, or this could be due to the low variance of small graphs.
- Our Random Forest always did better than the other models. A decision tree perhaps can find non-linear relationships between the features and clique sizes much better.

We next used the models to give us the most important graph features that it believes contribute to the max-clique sizes of graphs (in order):

- MLP Regressor : average no. triangles, no. of edges, total no. triangles, average k-core (and the other features contribute close to nothing or even negatively)
- Random Forest : average no. of triangles, total triangles, average shortest path length, density, transitivity, clustering coefficient (and the other features contribute trivially)

Although some of the models' feature importances are similar and make sense (such as triangles contributing heavily to graph clique size), it may be surprising at first that the other feature importances are very different for the two models. But this is expected based on the fundamental differences of the two models. Given that the MSE of the Random Forest was half the one of the MLP, we believe that the Random Forest's feature importances are more correct (and in such a problem, it makes sense that a decision tree finds relationships and patterns between features better).

We were interested in what would happen if we increased the no. of nodes in our graphs to $n \leq 200$. We had to decrease the number of graph samples to 2000 (1000 train and 1000 test) and even then it took about 40 minutes to do process the data. This time, our MLP did much worse, getting an MSE of $\approx 11$, although the Linear Regression and Random Forest Regressor performed well with $\approx 0.2$ MSE. The MLP's performance is expected, since MLPs don't usually perform well on small number of samples and also perhaps overfit the data.

## 5. Complexity Analysis and Time Plots

Do our models do any better than the brute-force algorithm to compute max-clique size? Below is a table showing the time complexities of different models and the brute force. We denote:

- $N$ the number of graphs we want to compute cliques for
- $n$ the maximum number of nodes in our graphs
- $d$ the number of features we used for our models
- $Q \leq N$ the number of training samples
- $E$ the max number of iterations for MLP
- $L$ the number of hidden layers of the MLP
- $h$ the max. number of nodes in each hidden layer of MLP
- $G$ the max number of iterations for Gradient Descent in Linear Regression
- $K$ the max. number of features considered at every split in a Random Forest
- $T$ the number of Decision Trees of a Random Forest
- $D$ the max depth of a Decision Tree

| Model | Pre-processing/Training | Computation/Prediction |
|---|---|---|
| Brute-Force Algorithm | - | $O(N \cdot 2^n)$ |
| MLP | $O(Q \cdot 2^n + E \cdot Q \cdot (dh + Lh^2))$ | $O((N-Q) \cdot dhL)$ |
| Linear Regression (G.D) | $O(Q \cdot 2^n + QdG)$ | $O((N-Q) \cdot d)$ |
| Random Forest | $O(Q \cdot 2^n + QKT \log Q)$ | $O((N-Q) \cdot TD)$ |

As we can see, the training complexities of the ML models are costly since we need to calculate exact max-clique sizes, but the testing complexity is constant when we fix our model parameters!

The above can be shown visually by plotting the testing times of the different algorithms for randomly sampled graphs (making sure the MSE was low enough).