Predicting Vanishing Coefficients of Denominator Graphs: From l-loops to $(l+1)$-loops OR Zero and few shot learning

Contents

1 Introduction

We consider the denominator graphs. It is understood that if the denominator has zero coefficient then any full graph that uses it will also have a coefficient of zero. The denominator graph is likely to represent the easiest task we can solve.

In this short document we will investigate predicting higher-loop coefficients from lower-loop coefficients. This goes towards the overall ambition of predicting coefficients of currently unseen or unknown loops.

2 Tabular Features

The notebook produces a diverse set of graph-based features, each reflecting a specific structural property of the graph. Below we describe each feature group and its potential usefulness.

Core-related Features

- `Core_core_index_mean` — The average *core index* across all nodes. A node's core index measures its position in the k-core decomposition, indicating how deeply embedded it is in the network. Useful for quantifying overall network cohesion.
- `Core_max_core_index` — The maximum core index found in the graph. High values suggest a highly interconnected core, indicating robustness or a central structure.

Cycle-based Features

- `Cycle_num_cycles_len_5`, `Cycle_num_cycles_len_6` — Counts of cycles of length 5 and 6. Cycle counts can reveal repeating motifs in networks, important in chemistry (ring structures in molecules) or in network motif analysis.

Basic Connectivity Metrics

- `EDGES`, `NUM_EDGES` — Number of edges in the graph. A fundamental density measure.
- `DEN_EDGES` — Likely edge density (ratio of actual edges to possible edges). High density means more connections between nodes.

Spectral Graph Theory Features

- `Spectral_algebraic_connectivity` — The second-smallest eigenvalue of the Laplacian; higher values mean the graph is harder to disconnect, useful for robustness analysis.
- `Spectral_laplacian_mean`, `Spectral_laplacian_std`, `Spectral_laplacian_skew` — Statistical moments of the Laplacian spectrum, capturing the shape of the spectrum which reflects graph connectivity patterns.
- `Spectral_spectral_gap` — The difference between the first two eigenvalues of the adjacency or Laplacian matrix; often relates to community structure detectability.

Planarity-based Features

- `Planarity_num_faces` — Number of faces in a planar embedding (if planar). More faces can suggest more complex local structures.
- `Planarity_face_size_max`, `Planarity_face_size_mean` — Maximum and average number of edges per face; reflects how "spread out" cycles are in a planar representation.

Symmetry Features

- `Symmetry_automorphism_group_order` — Size of the automorphism group (number of graph symmetries). Higher values indicate more structural regularity.

- `Symmetry_num_orbits` — Number of distinct node equivalence classes under symmetries. Fewer orbits mean more symmetry.
- `Symmetry_orbit_size_max` — Largest equivalence class of nodes under symmetries; can highlight repeated patterns.

Robustness Features

- `Robust_articulation_points` — Number of articulation points (nodes whose removal increases the number of connected components). Fewer points means more robust connectivity.
- `Robust_bridge_count` — Number of bridges (edges whose removal disconnects the graph). Fewer bridges imply greater robustness.

Global Network Descriptors

- `Kirchhoff_index` — A resistance-based distance measure, summing effective resistances over all pairs of nodes. Useful for characterising transport efficiency or redundancy.

3 Methodology

In this work, our objective is to predict the properties of *unseen loops* by leveraging the currently known lower-loop coefficients. We frame this task as a supervised learning problem using the `XGBoost` algorithm, which is widely recognized as one of the most competitive approaches for tabular data, often outperforming or matching the performance of deep neural networks in such settings. Importantly, this setup naturally corresponds to a *zero-shot learning* scenario: at inference time, the model is required to make predictions for $(l+1)$-loop quantities without having observed any corresponding training data for that loop order. While the primary focus is on the zero-shot setting, we will also explore the impact of incorporating a small amount of $(l+1)$-loop data into training to assess potential performance gains.

This setup corresponds to a zero-shot learning scenario in the strict case where no $(l+1)$-loop data are included during training (with the model generalizing solely from patterns learned at lower loops), whereas the inclusion of even a small fraction of $(l+1)$-loop examples would instead place it in a few-shot learning regime.

We will use cross validation in all our experiments with target stratification - with the average cross validated score over folds being our main

result. We will use ROC-AUC to measure performance. In general - the interpretation of the ROC-AUC score is if I randomly choose a graph with coefficient non-zero and a graph with coefficient zero, what is the probability that the model output score of first graph is larger than the second.

Finally, our analysis covers loops 7,8,9 and 10 - mostly because other edge data was not available.

4 Results and conclusions

4.1 Intra-loop and mixed loops

Pair	Loop	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg ROC AUC	Std Dev
Single Loops								
–	7	0.789	0.847	0.817	0.845	0.662	0.792	0.068
–	8	0.859	0.865	0.882	0.879	0.869	0.871	0.009
–	9	0.904	0.893	0.900	0.904	0.904	0.901	0.004
–	10	0.914	0.916	0.914	0.916	0.918	0.916	0.001
–	11	0.919	0.920	0.920	0.918	0.919	0.919	0.000
Mixed Loops								
7 & 8	7	0.884	0.888	0.810	0.873	0.814	0.854	0.035
	8	0.847	0.884	0.864	0.874	0.839	0.862	0.017
8 & 9	8	0.868	0.887	0.883	0.882	0.876	0.879	0.007
	9	0.899	0.894	0.906	0.898	0.909	0.901	0.006
9 & 10	9	0.902	0.895	0.896	0.912	0.905	0.902	0.006
	10	0.916	0.914	0.913	0.914	0.916	0.914	0.001
10 & 11	10	0.906	0.905	0.905	0.907	0.909	0.906	0.002
	11	0.919	0.919	0.918	0.918	0.918	0.919	0.000
8 & 10	8	0.850	0.856	0.864	0.872	0.845	0.858	0.010
	10	0.917	0.915	0.914	0.917	0.914	0.915	0.001
7 & 9	7	0.744	0.847	0.782	0.877	0.714	0.793	0.061
	9	0.910	0.888	0.905	0.896	0.902	0.900	0.008
7 & 10	7	0.756	0.736	0.774	0.810	0.662	0.747	0.049
	10	0.918	0.917	0.915	0.916	0.915	0.916	0.001

Table 1: ROC AUC scores for single loops and mixed loop combinations across 5 folds, with mean and standard deviation.

A few notables:

- Mixing $l+1$ with l always improves performance on l

- Mixing $l+1$ with l seems to degrade performance on $l+1$

$$\forall f_{j,\alpha}^{\nu^{(n-1)}} : \sum_{f_{i,\mu}^{\rho^{(n)}}} \frac{\left| f_{j,\alpha}^{\nu^{(n-1)}} \right|}{\left| f_{i,\mu}^{\rho^{(n)}} \right|} c_i^{(n)} = c_j^{(n-1)}; \quad \text{in particular,} \quad c_0^{(n-1)} \equiv 0$$

Cusp rule tells us that pinching on an l loop coefficient is equivalent to the weighted sum of $l+1$ loop coefficients. This would mean that there is information in $l+1$ loops that l loops can use and we indeed observe this in the results. (Note - we should do experiments or study ways to deconstruct this from simply having more data - not sure how much an issue this is).

4.2 Directionality: Zero-shot learning

In this section - we are modelling upwards - what we can call directionality. We train a model on lower loops in order to predict on higher loops. If we want to make use of a model to predict loops currently unseen we would want to do this.

This is an example of zero-shot learning. We will not show the model any information about higher loops - instead we want it to learn enough structure in lower loops and make good inferences of high loops based on information available.

As an exercise we will show the precision recall curves - recall that high precision means fewer false positives whilst higher recall means fewer false negatives.

4.2.1 $[7] \to [8, 9, 10]$

Dataset	ROC AUC
Train [7]	1.000
Test [8]	0.760
Test [9]	0.754
Test [10]	0.699

Table 2: Directionality summary: trained on loop 7, evaluated on loops 8–10.

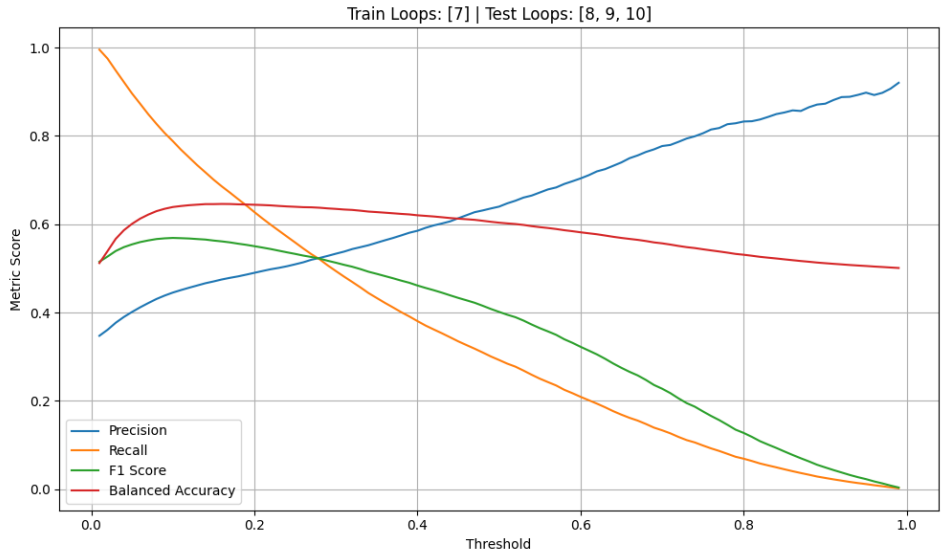

Figure 1: Threshold plots

Clearly - these are expectantly poor results.

- Recall drops off quickly indicating large amount false negatives comes in when we increase the threshold - i.e. bringing up the threshold makes the model think there are zero coefficients when there aren't (we expect this - but its happening quickly).
- Precision never reaches 1. If we push the threshold to near 1, a well calibrated model will reveal the high confidence coefficients - we do not get that here.

4.2.2 $[7, 8] \rightarrow [9, 10]$

Dataset	ROC AUC
Train [7, 8]	1.000
Test [9]	0.790
Test [10]	0.799

Table 3: Directionality summary: trained on loops 7 and 8, evaluated on loops 9 and 10.

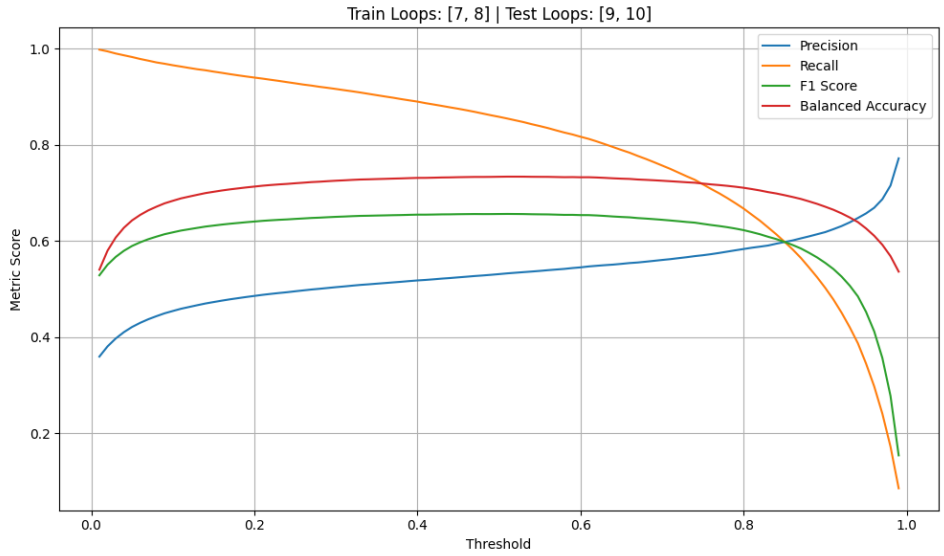

Figure 2: Threshold plots

- Recall drops off much slower than before - bringing the threshold up does not quickly drop coefficients - it is likely that the model is not well calibrated.
- Precision never reaches 1. If we push the threshold to near 1, a well calibrated model will reveal the high confidence coefficients - we do not get that here.

4.2.3 $[7, 8, 9] \rightarrow [10]$

Dataset	ROC AUC
Train [7, 8, 9]	0.997
Test [10]	0.817

Table 4: Directionality summary: trained on loops 7, 8, and 9, evaluated on loop 10.

This is the case that is of interest to us in this current study.

Really importantly - the training AUC-ROC does not reach 1. This indicates conflicting datapoints in loop 9 and data in loops 7 and 8. We

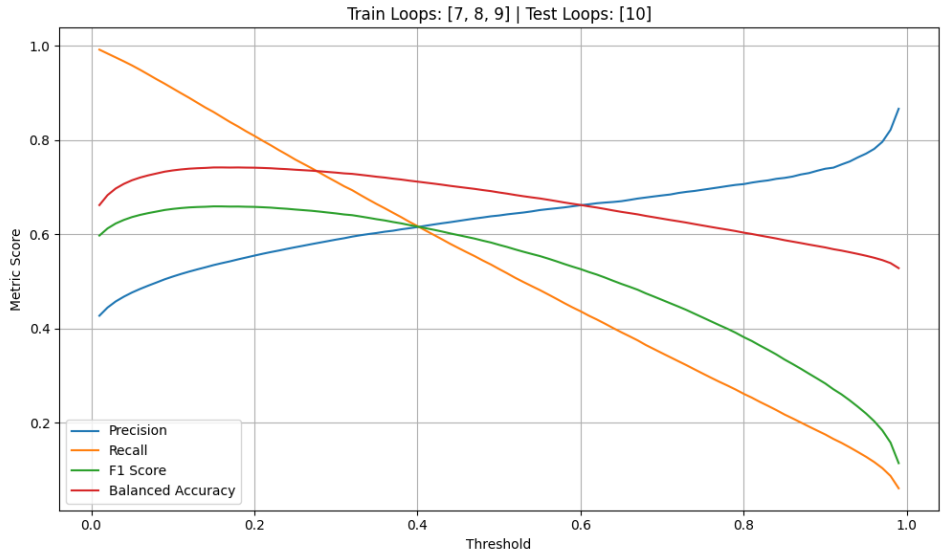

Figure 3: Threshold plots

need to enhance the feature space further to remove this conflict.

- Recall drops off quite linearly indicating the best calibrated we have had so far.
- Again precision never reaches 1. If we push the threshold to near 1, a well calibrated model will reveal the high confidence coefficients - we do not get that here.

4.2.4 $[7, 8, 9, 10] \rightarrow [11]$

Dataset	ROC AUC
Train [7, 8, 9,10]	0.943
Test [11]	0.857

Table 5: Directionality summary: trained on loops 7, 8, 9, adn 10 evaluated on loop 11.

4.3 Directionality + pieces of loop 10 needed to get intra-loop 10 level performance: Few-shot learning

In this section we look a the $[7, 8, 9] \rightarrow [10]$ problem again but we now cluster the 10 loop graphs and work out which clusters have the mosts value.

This is something akin for Few-shot learning (although we are taking quite a few more 10 loop examples into a training set than we might normally).

Cluster	Rows → Train	Remaining Test Rows	AUC
0	21310	147510	0.889
1	4927	163893	0.831
2	12960	155860	0.864
3	2294	166526	0.850
4	1424	167396	0.852
5	15156	153664	**0.890**
6	2467	166353	0.849
7	16574	152246	0.878
8	21277	147543	**0.890**
9	143	168677	0.834
10	6077	162743	0.838
11	654	168166	0.828
12	2600	166220	0.871
13	6539	162281	0.830
14	2476	166344	0.867
15	8740	160080	0.851
16	16018	152802	0.872
17	12670	156150	0.862
18	164	168656	0.835
19	14350	154470	0.867

Table 6: Per-cluster AUC results when promoting one cluster from loop 10 into training. Highest values in bold.

Promotion Step	Clusters Promoted	Overall AUC
Best 1st Cluster	[5]	0.890
After 2 Clusters	[5, 8]	0.908
After 3 Clusters	[5, 8, 19]	0.917

Table 7: Progressive improvement in AUC as additional clusters are promoted into the training set.

The additional clusters bring in 50783 more datapoints which are 10-loop specific. Training with the same amount of 10-loop data on a 10-loop problem alone gives a ROC AUC of about 0.907. Ultimately - its not so clear if this is a useful thing to do until we do further comparisons.

5 Uncertainty Quantification

TODO

6 Next steps

Questions to ask ourselves in this work

1. Try this on higher loops again
2. Move to full graph
3. Data Augmentation
   a) In vanilla modelling - we expect there to be a rich generating function - not particularly obvious how invariance we have that is not already encoded in features
   b) Fix the 9-loop problem.
4. Confidence scores and how in particular to use for 13-loops
5. Are we finding coefficients for graphs that cannot be found through rules like cusp rules?

On the GNN approach - we should get graph representations that are far richer.

7 Tree Misclassification by inclusion of loop order

Table 8: Overall by combo

train_loops	n_train	train_auc_overall	test_loop	test_auc
(9, 10)	167,224	0.942,495	11	0.869,956
(7, 9, 10)	167,388	0.943,104	11	0.864,540
(8, 9, 10)	168,656	0.942,171	11	0.862,627
(7, 8, 9, 10)	168,820	0.942,916	11	0.856,828
(8, 10)	154,684	0.945,206	11	0.840,198
(7, 8, 10)	154,848	0.945,260	11	0.838,728
(7, 8)	1596	1.000,000	11	0.811,331
(10,)	153,252	0.945,841	11	0.809,871
(8,)	1432	1.000,000	11	0.807,204
(9,)	13,972	0.997,817	11	0.798,707
(7, 9)	14,136	0.998,211	11	0.793,281
(8, 9)	15,404	0.996,969	11	0.789,976
(7, 10)	153,416	0.946,195	11	0.786,496
(7, 8, 9)	15,568	0.996,618	11	0.779,690
(7,)	164	1.000,000	11	0.636,116

Table 9: Training per-loop AUCs (long-form)

train_loops	loop_id	n_train_loop	train_auc_loop
(7,)	7	164	1.000,000
(7, 8)	7	164	1.000,000
(7, 8)	8	1432	1.000,000
(7, 8, 9)	7	164	1.000,000
(7, 8, 9)	8	1432	0.999,772
(7, 8, 9)	9	13,972	0.996,046
(7, 8, 9, 10)	7	164	0.996,557
(7, 8, 9, 10)	8	1432	0.970,624
(7, 8, 9, 10)	9	13,972	0.946,507
(7, 8, 9, 10)	10	153,252	0.942,077
(7, 8, 10)	7	164	0.986,719
(7, 8, 10)	8	1432	0.970,495
(7, 8, 10)	10	153,252	0.944,890
(7, 9)	7	164	1.000,000
(7, 9)	9	13,972	0.998,165
(7, 9, 10)	7	164	0.971,635
(7, 9, 10)	9	13,972	0.952,468
(7, 9, 10)	10	153,252	0.942,129
(7, 10)	7	164	0.999,016
(7, 10)	10	153,252	0.946,104
(8,)	8	1432	1.000,000
(8, 9)	8	1432	0.999,653
(8, 9)	9	13,972	0.996,556
(8, 9, 10)	8	1432	0.962,762
(8, 9, 10)	9	13,972	0.944,510
(8, 9, 10)	10	153,252	0.941,615
(8, 10)	8	1432	0.977,820
(8, 10)	10	153,252	0.944,844
(9,)	9	13,972	0.997,817
(9, 10)	9	13,972	0.953,727
(9, 10)	10	153,252	0.941,386
(10,)	10	153,252	0.945,841