Machine Learning Project

Adrian Patricio
Universitat Politècnica de Catalunya

Elnara Yerbolatova
Universitat Politècnica de Catalunya

I. Introduction

The goal of this project is to predict the root node in syntactic dependency trees derived from a parallel corpus of 21 languages. Each sentence is represented as an undirected tree where nodes are words and edges are syntactic dependencies. The root node, typically the most central word (e.g., the main verb or subject), must be identified using machine learning techniques. We frame this as a binary classification problem: for each node in a tree, we predict whether it is the root or not. The challenge lies in identifying features that are most predictive of roots, comparing different models and proposing the best-performing model with generalization estimates.

A. Considerations

The analysis was conducted under the following constraints:

- **Undirected Tree Structure**: The data consists of undirected edges defining the tree, with the root node ID serving as the target variable.
- **Class Imbalance**: The classification problem is highly imbalanced, as only one root node exists per tree.
- **Data Dependencies**: Nodes from the same sentence are not independent as their centrality measures are relative to the tree structure. Cross-validation must ensure that all nodes from a sentence stay in the same fold to avoid leakage.
- **Cross-Language Dependencies**: Since each sentence ID appears across all 21 languages (representing translations of the same sentence), nodes from the same sentence in different languages may exhibit dependencies.

II. Data Exploration

A. Pre-processing

To transform the data set into an expanded node level data set, we parsed the edgelist of each sentence into a tree using the networkx library, computed centrality measures for each node and assigned class labels (root = 1 and non-root = 0). No further cleaning was needed, as the original data did not contain missing values or outliers. Although the feature values contained outliers (see Table I for selected features), we preserved these values, as they are important in understanding the nature of root and non-root nodes.

B. Addressing Imbalance

The expanded dataset is inherently imbalanced, as each sentence tree has only one root node. To address this imbalance, we considered resampling methods, but ultimately decided against using them for these reasons:

Feature	Number of Outliers	Avg. Outliers per Sentence
degree	12,259	1.168
harmonic	3,526	0.336
betweenness	8,057	0.767
pagerank	9,905	0.943

TABLE I: Total count and average number of outliers per sentence for some example features (based on 10,500 sentences)

For **oversampling methods** like SMOTE, we decided that artificially generating synthetic roots would be illogical because each sentence can only have one true root. This violates the natural imbalance that is present within the tree structure and introduces unrealistic data.

For **undersampling**, we realized that removing non-root nodes would risk losing valuable structural information about the tree, particularly for longer sentences where non-root nodes exhibit diverse centrality patterns. Much like oversampling, this approach could also distort the natural class imbalance, potentially reducing the model's ability to generalize to real-world, inherently imbalanced data. Interestingly, applying Tomek links to identify borderline or noisy samples resulted in the removal of only a small portion of the majority class (see Table II for reference), suggesting minimal overlap between classes in the feature space. This shows that centrality measures for roots were distinct enough from non-roots, and that there would be a significant loss of information from undersampling. We also tested the Random Undersampler, but it slightly degraded the performance of our linear model (logistic regression), reducing the score from 0.297 to 0.295. This suggests limited benefit—and even a small cost—in applying undersampling. It's worth noting that while undersampling can help balance classes, it is most effective when the majority class includes redundant or less informative samples, which was not the case in our data.

Class	Original Count	Resampled Count
0 (non-root)	186,979	185,528
1 (root)	10,500	10,500

TABLE II: Comparison of original and resampled class distributions after applying Tomek Links

Instead, we prioritized class-weighted models (e.g., class_weight='balanced' in scikit-learn), which adjust misclassification penalties during training to favor the root class. This approach preserves the natural data distribution while ensuring the model focuses on correctly identifying roots. Additionally, we focused on metrics such as F1-score over accuracy when evaluating our models, as they better capture performance on

imbalanced data.

C. Feature Engineering

Throughout the process, we updated our feature set to ensure compatibility across different model types:

- For logistic regression, all features needed to be numerical or properly one-hot encoded, as the model cannot handle raw categorical data.
- For Random Forest and XGBoost, categorical features with many levels can lead to overfitting or non-ideal splits, so we prioritized binary or low-cardinality features.
- For GCNs, features were incorporated as node attributes, with consideration taken to normalize features as needed to ensure aggregation during model training.

This feature engineering process was essential to ensure that each model could take advantage of available information and avoid poor performance related to feature type incompatibilities.

Some of the features we explored are listed below.

1) Centrality Features: We used the networkx library to compute various centrality measures that capture different aspects of node importance in dependency trees. Our initial feature set included degree, closeness, betweenness, and PageRank centralities. We later expanded it to include additional centrality metrics: eigenvector, harmonic, eccentricity, average neighbor degree and load centrality. This allows us to thoroughly capture each node's structural properties within its tree.

2) Interaction Features: To capture nonlinear relationships between centrality measures, we engineered interaction features including multiplicative terms (e.g., degree × betweenness) among the top performing features. These features potentially enhance linear models by explicitly encoding how different centrality measures combine to indicate root nodes - for instance, the product of degree and betweenness centrality identifies nodes that are both highly connected and strategically positioned in the tree structure, which could possibly characterize roots better. These features, in general, could particularly be beneficial for linear models, which otherwise cannot capture these interactions without explicit engineering. For tree-based models, while these effects can be learned, these features can improve performance by making these relationships more accessible to the model.

3) Structural Features: We computed articulation points—nodes whose removal would split the tree—as root nodes often serve as key connectors. These were encoded as binary features (0/1) to ensure compatibility with tree-based models, which can be sensitive to high-cardinality categorical inputs. Additional structural features included a tree balance estimate (how evenly components are sized after node removal, scaled 0–1), path asymmetry (variation in path lengths from a node), and the number of connected components after removal. We also flagged whether a node lies on the diameter path—the longest path between any two nodes—since such nodes often hold structural significance. Finally, we calculated balance variance, the normalized variance of component sizes after node removal, where lower values indicate more balanced splits. While articulation points already signal node importance, the connected component count added nuance by quantifying the degree of fragmentation, making it a useful complementary feature.

4) Language Features: Upon evaluating the feature distributions across languages, we observed that most languages showed clear separations between root and non-root nodes across all centrality measures, while Japanese exhibited significant overlap in these values (see Fig A1 in the Appendix with an example case of betweenness feature, or refer to the Python notebook for additional analysis of features). To account for this, we added a binary feature to explicitly model if the sentence is in Japanese. In addition, we created binary features to model different language family groupings based on syntactic structure, which included Head-final SOV, Romance SVO, Germanic, Free Order and Analytic groups. These categorical features required careful encoding in either one-hot or binary for use in logistic regression and XGBoost, while for random forest, high-cardinality categorical data can lead to suboptimal splits.

D. Scaling and Normalization

Most of the centrality measures provided by NetworkX are already normalized if we consider them at the sentence level. Those include:

- Degree: obtained by dividing the degree of each node by the maximum possible degree
- Closeness: normalized by dividing by the maximum possible closeness
- Pagerank: where the sum of all PageRank values in the network equals 1.
- Betweenness: obtained by dividing the raw count of shortest paths passing through a node by the total number of possible pairs of nodes
- Eigencentrality: where the sum of the squares of the eigenvector centralities is 1

Scaling was not applied to binary features, such as language family groups and articulation point labels, as it is unnecessary in such cases. For other features that were initially unscaled (such as eccentricity, path_asymmetry, etc.), the MinMaxScaler was applied to normalize values to the [0, 1] range. This made them more comparable to other features, including centrality measures and binary indicators, which also fall within this range. Scaling was only considered for Logistic Regression, as it is sensitive to feature magnitudes. In contrast, tree-based models and GCNs are not affected by feature scaling.

Whole-dataset normalization was not considered since it would force one common scale for nodes from all sentences, leading to distorted relative centrality of the nodes within their respective dependency trees.

E. Normality of Features

As an exploration step, the normality of features was also assessed. Since number of nodes per sentence varied between 3 and 70 across languages and the average number of nodes were around 15-20 per sentence, Shapiro-Wilk test seemed appropriate to test the gaussianity. For reference, Table III shows, for each selected feature, the proportion of sentences for which its distribution approximates a normal distribution.

Feature	Fraction Normal
betweenness	0.106
closeness	0.940
degree	0.029
eigencentrality	0.464
load	0.106
pagerank	0.129

TABLE III: Fraction of normal sentences per feature based on Shapiro-Wilk Test

This shows that most feature distributions (except for closeness) are far from being Gaussian. Since some linear models, such as Linear Discriminant Analysis, assume normally distributed features, we opted for Logistic Regression, which is more robust to non-Gaussian features.

F. Feature Selection

Before training any model, we performed an initial feature selection step to remove highly correlated features, which is particularly important for linear models such as Logistic Regression, which assumes feature independence for optimal performance. This also helps to eliminate redundancy, since correlated features model the same information, which could eventually lead to overfit. In addition, this helps improve computational efficiency, especially for models such as GCN. For example, as shown in Fig A4 from the Appendix, we see that load and betweenness have perfect correlation, meaning that they yield the same information. The same applies for PageRank and degree centrality. Other than that, harmonic centrality did not show clear separation between classes (see Fig A2 in the Appendix). Consequently, we excluded it to prioritize more predictive centrality measures. Some highly correlated features, such as 'tree_balance', 'balance_variance' were also excluded from the analysis of Logistic Regression, but retained for tree-based models, since the latter are robust to collinearity.

After this, we further refined the feature set based on model-specific feature selection approaches, which will be further discussed in the next section.

III. Methodology

Due to the nature of our problem, it was necessary to account for two main aspects:

- **One-root scoring**: since each sentence in every language has only a single root node, our models needed to be constrained to predict exactly one root per sentence-language pair. To achieve this, we computed the model's probability predictions for each node within a sentence-language group and designated the node with the highest probability as the root.
- **Cross-validation per sentence id**: to prevent information leakage across languages, we ensured that no sentence appeared in both the training and validation sets, even if it was in a different language. In other words, sentence IDs were kept unique across train and validation folds, regardless of the language, to avoid cross-lingual data contamination.

A. Custom Optuna

To incorporate the functionalities mentioned above and tune the hyperparameters, we created customised scoring and objective functions for Optuna. Optuna was chosen over random search or grid search due to its efficiency in finding optimal hyperparameter values more quickly.

The tuning process was carried out in the following steps:

1) Define hyperparameters appropriate for the model in use
2) Run a custom function to create cross-validation folds using GroupKFold, grouping by sentence ID. The k-fold cross-validation approach was chosen specifically because we were working with a relatively small dataset, and this method allowed us to make the most of the limited data available.
3) For each fold, train the model and perform validation. Where applicable (as in the case of logistic regression), feature transformations (e.g. scaling) were applied using a pipeline—fit_transform on the training set and transform on the validation set.
4) Evaluate the F1-score on validation folds using the customised one-root scorer, and average the scores to compare different hyperparameter configurations and identify the best-performing model.

To speed up the tuning process, we used the Median Pruner to prune trials that failed to show promising results. Specifically, a trial was pruned if it did not show any improvement after two folds.

B. Evaluating Feature Importances

After completing the initial hyperparameter tuning, feature importances were assessed to identify the most predictive features. This information was then used to retune the models with the selected subset of features.

C. Model Comparison and Testing

Once the best hyperparameters were identified, the final models were trained on the entire training set and evaluated on the test set provided on Kaggle. To compare several models — when we wanted to assess relative model performance without relying solely on the test set — we used the last fold from our custom fold-creation function (also used in step 2 of the tuning process) as a validation set. This allowed us to train the models on three folds, validate on the fourth, and compare validation scores across models.

D. Model Choices

Several models were considered, including both linear and non-linear classifiers, which will be further discussed in detail.

1) Logistic Regression: We hypothesized that some centrality measures might provide linear separability for root nodes, making linear methods suitable for their classification. We selected logistic regression, as its probabilistic outputs allow selecting the node with the highest predicted probability as the root, matching the requirements of our problem. In addition, it does not impose strict assumptions about the feature distribution, making it robust to our data. The main assumption is that features should not be multicollinear and correlation should be minimised, both of which were mitigated in the feature selection step.

Settings: To account for class imbalance, we set the class_weight hyperparameter to penalize misclassifications of the root class more heavily. The solver was set to liblinear for its efficiency with smaller datasets [5].

Feature Selection: Logistic regression employs L1, L2 or a mix of both (Elasticnet) to perform feature selection and prevent overfitting. The strength of regularization is controlled by the hyperparameter C where a smaller value increases penalization [5]. Hyperparameter tuning showed that L1 was the optimal choice. Since our data had correlated features, this was expected because L1 regularization completely removes redundant features automatically.

We further tried tuning another logistic regression model with only the features selected by L1. We expected that this would improve the performance, as any redundancies and correlation would have been removed, but it did not produce any significant improvements.

2) Random Forest: As shown in Fig. A4, the feature correlations with the target value are all very low. This indicates that the relationship between features and the target is highly non-linear. Random Forest is well-suited for capturing such complex, non-linear patterns. Moreover, due to its random subsampling of data points when constructing decision trees within the ensemble, Random Forest is naturally more resistant to overfitting. This is especially important in our case, as the dataset contains a relatively small number of unique trees (21 languages × 500 sentences = 10,500 trees), which increases the risk of overfitting.

Settings: To address class imbalance, we applied class weighting by setting the class_weight hyperparameter to 'balanced', as previously mentioned. Other hyperparameters were fine-tuned based on insights from several sources [1], [2].

Feature Selection: Random Forest classifier is also helpful in identifying features with high predictive power, evaluating the decrease in the impurity index at each threshold split as a criterion. This capability made it valuable for identifying features most influential in root prediction and for filtering out less predictive ones.

3) Boosting Models: Boosting-based ensemble methods appeared promising, as they iteratively correct the errors of previous classifiers, enabling them to capture complex patterns in the data—similar to Random Forest. Furthermore, since both Random Forest and boosting models use decision trees as base learners, they make no assumptions about the underlying data distribution. This is ideal for our dataset, which contains many dependencies and deviates significantly from a Gaussian distribution (as discussed earlier). We experimented with both XGBoost and LightGBM as representative boosting models.

Settings: Key hyperparameters for both models were selected with the help of relevant literature and documentation [3], [4]. To address class imbalance, we used the scale_pos_weight parameter to assign weights to the positive class. The weight was calculated as the ratio of non-root nodes to root nodes. For XGBoost, this was set manually, while in LightGBM the same effect was achieved by enabling the is_unbalance parameter.

Feature Selection: Both models included regularisation hyperparameters - lambda_l1 and lambda_l2 for LightGBM, reg_alpha and reg_lambda for XGBoost - to reduce overfitting and support feature selection in our highly correlated feature space.

4) Graph Neural Networks: Given the inherent graph structure of the data, we explored Graph Convolutional Networks (GCNs) to leverage the natural tree structure and capture dependencies that the features alone might miss. Interestingly, GCNs aggregate information from neighboring nodes, which help capture dependencies between nodes as well and could allow better generalization of the data [6].

Settings: We modeled the data as an undirected tree by adding bidirectional edges between nodes. The model begins with a linear transformation of the input node features, followed by stacked GCN layers with ReLU activation and dropout. ReLU introduces non-linearity to enable the model to learn more complex patterns, while dropout adds regularization.

Since GCN operates directly on the edge list, the model predicts the node index that it considers to be the root for each tree. To ensure comparability with other models, we converted the predicted root node into a binary attribute for each node to ensure that the F1-score could be calculated. This adjustment allows for a direct comparison of performance metrics across all models evaluated.

Feature Selection: For GCN, initial node features are important as they provide initial context about the nodes which the model then refines to better generalize the graph patterns. The node features were initially defined to be the centrality features to get a baseline performance, with language features being added soon afterwards. Interestingly, using structural features did not improve the performance of the model, possibly because GCN already learns the graph structure by nature. To manage computational costs and training time, we limited the features used, and the grid size for hyperparameter tuning - both of which could be better explored to improve the model.

To prevent overfitting, we applied L2 regularization, which penalizes large model weights and encourages simpler, more generalizable solutions. The strength of this penalty is controlled by the weight_decay parameter in the Adam optimizer.

The dropout layer also adds regularization by randomly deactivating nodes during training to prevent overfitting and avoid relying heavily on specific nodes.

IV. Results

Table IV presents the overall performance of all evaluated models on both the validation and test sets. The corresponding hyperparameters for each model are documented in Table A2 in the Appendix.

As shown, XGBoost achieved the highest performance on both the validation set and the test set, closely followed by LightGBM and Random Forest. The table below also shows the result of Ensemble method that used the three tree-based models with soft voting.

Model	Validation F1 (Overall)	Test Accuracy[1]
XGBoost	0.3124	0.3181
LightGBM	0.3097	0.3161
Random Forest	0.3063	0.3151
Ensemble	0.3059	0.3175
Logistic Reg.	0.3044	0.2985
GCN	0.3025	0.3064

TABLE IV: Comparison of overall validation F1-score and test set accuracy for all models and the ensemble.

V. Discussion

Among all models, XGBoost, LightGBM and Random Forest achieved the highest F1-scores and demonstrated close performance to each other, suggesting that tree-based methods are particularly well-suited for the root prediction task.

In addition to evaluating overall performance, we examined model F1-scores for each language. As shown in Table IV (Appendix), performance differences between models were generally modest, though certain models performed better on specific languages. Further, feature importance analyses (see Fig. A3) revealed that different models prioritized different features, indicating the potential for complementary strengths. Given that tree-based models achieved the highest scores and were trained on the same feature set, we combined them using a soft voting ensemble. Although the ensemble did not surpass XGBoost in aggregate metrics, its ability to integrate diverse feature importances from the base models suggests that it may offer greater robustness and improved generalization, making it a potentially good candidate in such variable syntactic contexts.

While logistic regression offered a good base model due to its simplicity and interpretability, it produced suboptimal results compared to tree-based models. Its linear nature could have limited its ability to capture complex, nonlinear relationships present in the data. In addition, GCNs also underperformed compared to other models. This could be due to the relatively small size of the data or the fact that performance constraints limited the search space.

[1]The reported accuracy scores were computed using the test set that obtained perfect evaluation metrics on Kaggle when employing directed edge representations.

In general, however, the F1-scores across models were quite similar, which could suggest that performance is capped by the feature space we are working with. Additional model tuning could be explored, but it is possible that richer features would be needed to push the performance further.

A. Additional Insights

When it comes to features, after the first round of hyperparameter tuning for Random Forest, interaction terms between the most predictive features were added to see if they would provide additional useful information. However, a slight drop in performance suggested that it was better to focus on simpler features instead. Several tests and trials eventually showed that centrality measures had the strongest predictive power, followed by a few structural features. Language-based features contributed very little. It is also worth noting that since LightGBM can handle categorical features directly, the model was tuned using a feature set that included language as a categorical feature, rather than using binary indicators for language groupings. Even so, this did not improve the model's performance.

One of the most interesting observations was that models trained on a single feature—or those that relied heavily on one feature according to feature importance—performed surprisingly well on the dataset where the order of vertices was not shuffled. For example, Random Forest initially performed best among all models (achieving around 0.56 F1-score on validation and 0.51 accuracy on test), largely due to its reliance on the 'betweenness-degree' interaction feature. But when its performance was tested on a version of the dataset with shuffled vertex order, it worsened as expected. This revealed a form of indirect data leakage: the model had learned to identify the root node based on the consistent ordering of nodes and applied that pattern to the validation and test sets. The performance drop highlights the risk of leakage, showing that even simple preprocessing choices—like keeping the original node order—can introduce patterns that models end up exploiting.

VI. Conclusion

The study evaluated various modeling approaches for predicting the root status of a node in a syntactic dependency tree. It tested linear models such as Logistic Regression, tree-based models including Random Forest, XGBoost, and LightGBM, as well as a Graph Convolutional Network (GCN). The results indicated that tree-based methods were the most effective for this task. Based on this finding, a voting-based ensemble model incorporating the three top-performing methods was also tested to assess potential complementary effects. Although the ensemble model did not outperform each individual base model in terms of F1-score or test accuracies, it was selected as the best overall approach due to its ability to combine the strengths of multiple top-performing models, offering greater robustness and more consistent performance across diverse languages and syntactic structures.

REFERENCES

[1] W. Turing, "Hyperparameter tuning the random forest in Python using Scikit-Learn," *Towards Data Science*, 2020. [Online]. Available: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74/. [Accessed: May 30, 2025].

[2] Scikit-learn Developers, "sklearn.ensemble.RandomForestClassifier," *Scikit-learn Documentation*, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed: May 30, 2025].

[3] Tutorialspoint, "LightGBM - Parameter Tuning," *Tutorialspoint*, 2024. [Online]. Available: https://www.tutorialspoint.com/lightgbm/lightgbm-parameter-tuning.htm. [Accessed: May 30, 2025].

[4] P. Banerjee, "A guide on XGBoost hyperparameters tuning," *Kaggle*, 2020. [Online]. Available: https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning. [Accessed: May 30, 2025].

[5] GeeksforGeeks, "How to optimize Logistic Regression performance?," *GeeksforGeeks*, 2024. [Online]. Available: https://www.geeksforgeeks.org/how-to-optimize-logistic-regression-performance/. [Accessed: May 31, 2025].

[6] M. Bronstein, J. Bruna, T. Cohen, P. Veličković, "A Gentle Introduction to Graph Neural Networks," *Distill*, 2021. [Online]. Available: https://distill.pub/2021/gnn-intro/. [Accessed: May 31, 2025].

APPENDIX

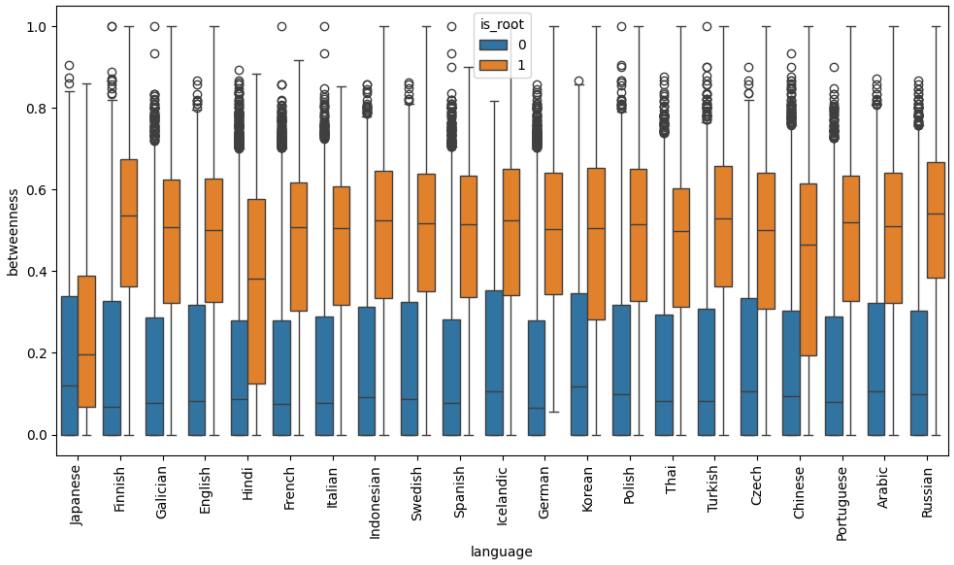

Fig. A1: The separation of classes per language for Betweenness feature

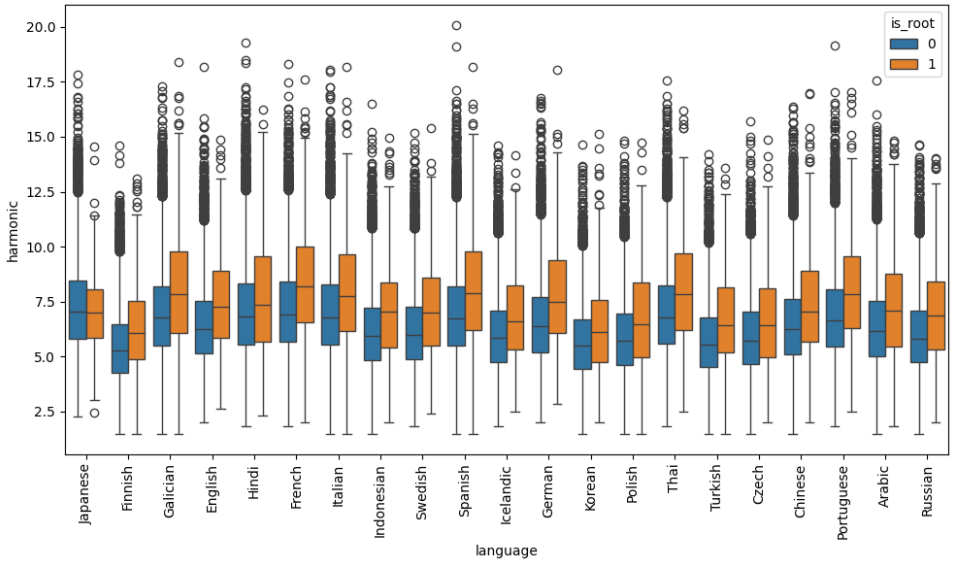

Fig. A2: The separation of classes per language for Harmonic Centrality feature

Language	F1_LGB	F1_RF	F1_XGB	F1_LogReg	F1_Ensemble	F1_GCN	Best Model(s)
Overall	0.3097	0.3063	**0.3124**	0.3044	0.3059	0.3025	XGB
Japanese	0.112	0.072	**0.128**	0.0480	0.0960	0.0960	XGB
Finnish	0.384	0.384	**0.400**	0.3920	0.3840	0.3600	XGB
Galician	**0.320**	**0.320**	0.288	0.3040	0.3040	0.2880	LGB/RF/LogReg/Ensemble
English	0.272	0.256	0.248	0.2480	0.2720	**0.3120**	GCN
Hindi	0.248	0.256	0.256	0.2640	0.2560	0.2800	GCN
French	0.328	0.320	**0.352**	0.3040	0.3280	0.3040	XGB
Italian	0.272	0.256	0.280	0.2720	0.2640	**0.3200**	GCN
Indonesian	0.312	0.320	0.312	**0.3280**	0.3200	0.2800	LogReg
Swedish	0.328	0.352	0.360	**0.3680**	0.3520	0.3600	LogReg
Spanish	**0.368**	**0.368**	0.352	0.3280	0.3520	0.2960	LGB/RF
Icelandic	0.296	0.256	0.272	0.2720	0.2640	**0.3600**	GCN
German	0.288	0.272	0.288	**0.2960**	0.2640	0.288	LogReg
Korean	**0.328**	0.304	0.320	**0.3280**	0.3200	0.2960	LGB/LogReg
Polish	0.328	0.336	**0.352**	0.3120	0.3280	0.3040	XGB
Thai	0.312	**0.320**	0.312	0.3040	0.3120	**0.320**	RF/GCN
Turkish	0.328	0.344	0.304	**0.3520**	0.3120	0.2720	LogReg
Czech	0.360	**0.368**	0.360	0.3520	0.3600	0.3520	RF
Chinese	0.248	0.240	0.248	**0.2560**	0.2560	0.2560	LogReg/Ensemble/GCN
Portuguese	0.336	0.336	**0.368**	0.3440	0.3440	0.3280	XGB
Arabic	0.360	0.360	0.360	0.3440	0.3440	0.2800	LGB/RF/XGB
Russian	0.376	0.392	**0.400**	0.3760	0.3920	**0.4000**	XGB/GCN

TABLE A1: F1-score comparison on Validation set by language for LGBM, Random Forest, XGBoost, Logistic Regression (updated), Ensemble and GCN. Bold indicates the best F1 per row.

Feature	LogReg	XGBoost	LightGBM	Random Forest
betweenness	0.3006	0.3803	0.0803	0.2111
closeness	0.2455	0.0176	0.1429	0.0765
pagerank	0.1235	0.0171	0.1319	0.0989
eigencentrality	0.0842	0.0384	0.1523	0.1656
is_articulation	0.0766	0.3001	0.0009	0.0223
eccentricity	0.0625	0.0129	0.0474	0.0186
path_asymmetry	0.0215	0.0133	0.1671	0.0344
is_japanese	0.0200	0.0419	0.0155	0.0134
diameter_path	0.0172	0.0151	0.0085	0.0038
avg_neighbor_degree	0.0132	0.0152	0.0605	0.0246
lang_group_germanic_v2	0.0101			
lang_group_romance_svo	0.0083			
lang_group_free_order_case	0.0073			
lang_group_analytic	0.0057			
lang_group_head_final_sov	0.0038			

Fig. A3: Comparison of Feature Importances Across Models (Normalized Coefficients for Logistic Regression and LightGBM)

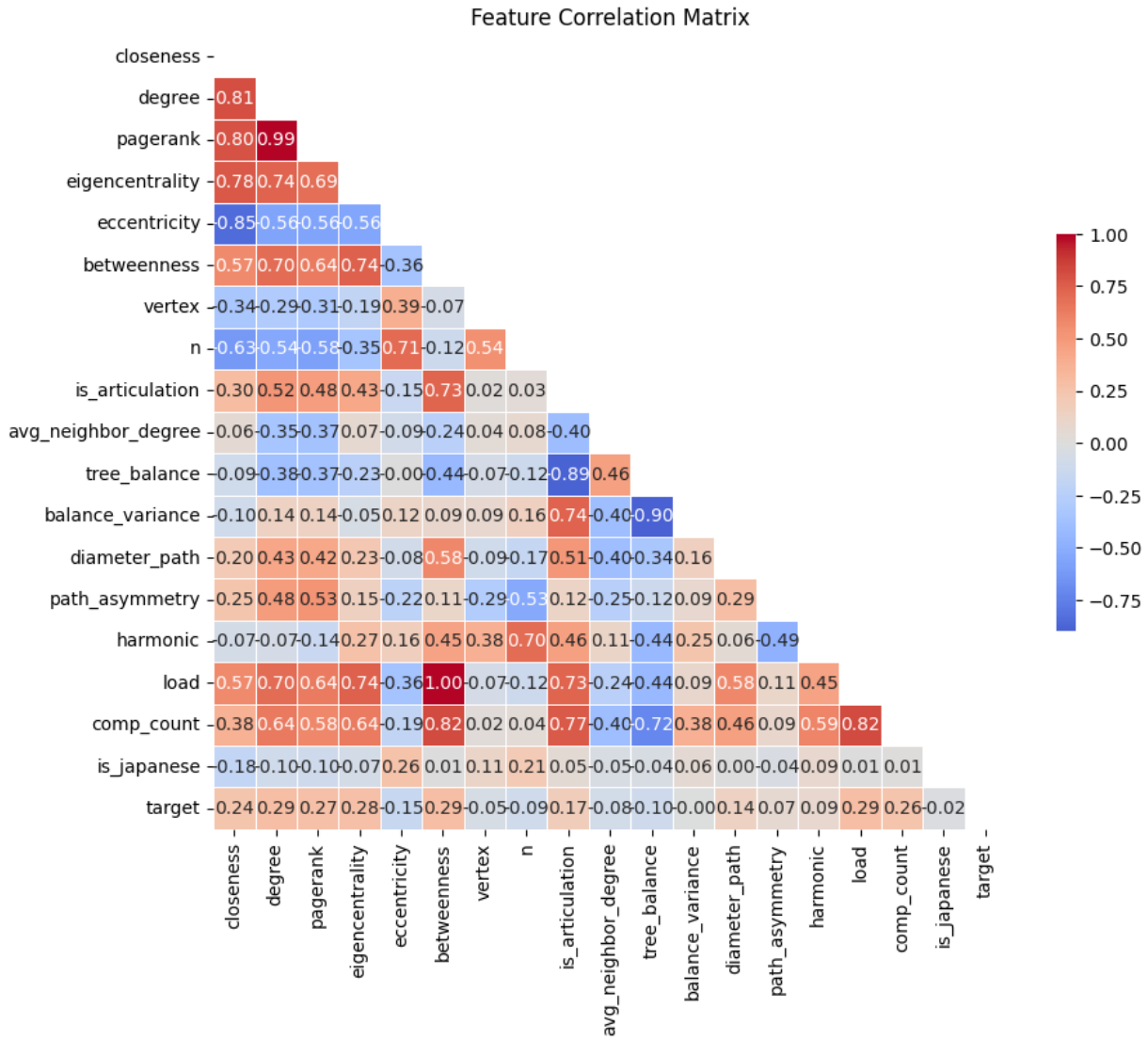

Fig. A4: The correlation heatmap of features

Linear Model	
Model	**Parameters**
Logistic Regression	• C=06800570611214875 • penalty='l1' • solver='liblinear' • class_weight={0: 1, 1: 10} • max_iter=2000
Non-Linear Models	
Model	**Parameters**
Random Forest	• n_estimators=400 • max_depth=10 • min_samples_split=12 • min_samples_leaf=9 • max_features='sqrt' • bootstrap=True • class_weight='balanced' • random_state=42
LightGBM	• n_estimators=500 • learning_rate=0.025140403939356265 • max_depth=19 • num_leaves=57 • min_data_in_leaf=92 • lambda_l1=0.004511956674608554 • lambda_l2=0.00473058130320583 • bagging_freq=6 • min_gain_to_split=0.10097122764552519 • feature_fraction=0.8131231416490341 • bagging_fraction=0.9966270789710742 • objective='binary' • metric='binary_logloss' • is_unbalance=True • random_state=42
XGBoost	• n_estimators=350 • learning_rate=0.04121474901281044 • max_depth=5 • min_child_weight=7 • gamma=0.25707828578238207 • subsample=0.8349214979910949 • colsample_bytree=0.9566868735297376 • reg_alpha=0.04435803326494737 • reg_lambda=0.24303259299874216 • objective='binary:logistic' • eval$_metric =' logloss'$ • tree_method='hist' • random_state=42
GCN	• hidden_channels=128 • num_layers=2 • dropout_rate=0.24007533818963195 • learning_rate=0.00967001569505872 • weight_decay=0.00017412411662641109

TABLE A2: Final hyperparameters