Prediction of Root Node in a Free Tree

Machine Learning

Haonan Jin & Hang Yao

2025-06-03

Abstract

This project focuses on predicting the root node in syntactic tree structures derived from sentences translated into 21 languages. Each node represents a word, and the goal is to identify the root node in each sentence using machine learning. We compare two strategies: training separate models for each language to capture language-specific patterns, and developing a multilingual model to learn cross-linguistic features. After selecting the best approach, we apply cross-validation and hyperparameter tuning to optimize performance. We also explore ensemble methods to enhance accuracy and robustness. The experimental results reveal the trade-offs between multilingual and language-specific models for the root prediction task.

1 Introduction

This project aims to build models that identify the root node in tree structures representing sentence syntax. We applied **Decision Tree**, **Random Forest**, and **AdaBoost** models to the provided datasets. The original sentence-level data included features like `language`, `sentence`, and `edge list`, with `root` labels in the training set. We converted the data to a node-level format and split the training set into training and validation subsets, as no separate validation set was provided (see Section 3).

The main objective of this project are:

- To identify the most significant features contributing to accurate root node prediction.
- To preprocess the data according to the tree structure, ensuring sentence-level and node-level granularity are respected.
- To compare different modeling strategies for different models.
- To perform hyperparameter tuning for the best-performing model.
- To evaluate and analyze the results to determine the strengths and limitations of each approach.

2 Feature Justification

At the beginning of the project, we explored various features for predicting the target variable `is_root`. Guided by the **NetworkX** centrality documentation [1], we initially selected a wide range of features such as `eigenvector`, `PageRank`, `degree`, and others. However, this initial feature selection yielded poor results, as we had not yet identified which features were most informative for our specific task.

As shown in Table 1, we evaluated this initial feature set using a basic **Logistic Regression** model as a baseline. The performance was suboptimal, with a `Macro-Avg F1 score` of 0.52 and a `True F1 score` of only 0.22. These early results highlighted the need for a more informed and discriminative feature selection process adjusted to the prediction of root nodes.

After further analysis, we selected a set of centrality and structural features that capture various aspects of node importance and position in tree (key properties for identifying the root node) based on relevant information provided in [2]. These features (Table 2) collectively offer diverse and complementary insights into node connectivity and centrality, enabling our models to more effectively distinguish root nodes across sentences and languages.

3 Data Exploration

3.1 Data Splitting

Since our data is structured as trees—where each sentence forms a tree and each node represents a word—standard splitting methods from the `sklearn` library are not suitable. This is because all nodes (words) belonging to a single sentence must remain in the same dataset (either training or validation). Splitting them arbitrarily would break the sentence structure and lead to information leakage or distorted patterns. To address this, we implemented a custom splitting function that ensures all nodes from a sentence stay together, preserving tree integrity and enabling the model to learn meaningful relationships between nodes.

3.2 Preprocessing

After extracting features for each node, we performed feature normalization at the sentence level. Specifically, we applied `Min-Max Scaling` [3] independently within each sentence to ensure that the scaled values reflect the local structure of that sentence, rather than the global distribution across the dataset.

Normalizing across the entire dataset would distort the relative relationships between nodes within a sentence and could mislead the model, especially given the variability in tree size and structure across languages. By scaling at the sentence level, we maintain the internal consistency of each tree and allow the model to capture the tree syntactic patterns.

3.3 Feature Engineering

To analyze feature structure, we applied **Principal Component Analysis** (PCA). The heatmap in Figure 1 shows correlations between original features and the first seven principal components.

PC1 and PC3 showed the strongest (though still weak) correlation with the target—PC1 negatively, PC3 positively—indicating a diffuse predictive signal spread across components. As such, we chose to retain all original features rather than reduce dimensionality, preserving the dataset's full information for classification tasks.

3.4 Resampling

The final step in our data exploration was addressing class imbalance (applied only on train dataset to fit the model). Since each sentence contains only one root node, the dataset is heavily imbalanced, with root nodes (minority class) significantly outnumbered by non-root nodes (majority class).

To mitigate this imbalance, we employed `SMOTEEN` technique [4], which combines both over-sampling (`SMOTE`) and under-sampling (`ENN`) methods. We set the sampling strategy to 0.5, meaning that after resampling, the number of minority samples is 50% of the majority class. This balanced approach reduces class imbalance while limiting the risk of overfitting or introducing noise, ultimately enhancing the model's capacity to generalize effectively to unseen data.

4 Architecture Selection

In this project, we considered two primary architectural strategies for training our models:

- **Language-Specific Models:** We trained one model per language, allowing each to specialize in the structural and linguistic characteristics unique to that language. After training, we assessed the similarity between models by comparing their prediction outputs across different language-specific datasets. For models that exhibited similar behavior (this similarity was computed by the accuracy score on their predictions to validation data), we applied an ensemble technique—specifically, **stacking**—to combine them into a unified meta-model [5]. This stacking approach uses logistic regression as the final estimator to learn from the combined outputs of the base models.

  This strategy helped reduce redundancy while preserving performance. Since this approach required training a model for each language, the original dataset was first partitioned by language to ensure that each model was trained exclusively on language-specific data.

- **Global (Giant) Model:** Train a single model using the full multilingual dataset, treating all languages jointly.

Hypothesis: Language-specific models are expected to perform better in individual languages, as they can capture fine-grained patterns and language-specific graph structures. Different languages often vary in graph size, syntactic tree depth, and centrality distributions, which a shared model may fail to represent equally well. However, this approach comes with the drawback of limited data per language, especially for low-resource ones, which can lead to underfitting or instability in training.

On the other hand, the global model benefits from a much larger and more varied training set. This can help improve generalization and stability during training, especially for high-resource languages. However, the diversity of the data may also introduce noise and conflicting patterns between languages, making it harder for the model to learn language-specific nuances. In particular, ”weak” languages risk being overshadowed during training, as the model may bias toward dominant patterns from more frequent languages. To assess both strategies, we implemented and tested them experimentally.

5 Model Selection

We experimented with three models **Decision Tree**, **Random Forest**, and **AdaBoost** each chosen for specific strengths in handling the root prediction task.

Decision Tree: We started with **Decision Tree** for its simplicity, interpretability, and ability to model non-linear feature interactions. It provided a strong baseline and initial insights into feature importance, while offering some resilience to class imbalance through pruning and custom splits.

Random Forest: **Random Forest**, as an ensemble of **Decision Tree**, improved generalization by reducing overfitting and capturing diverse patterns—crucial for our structurally complex, imbalanced dataset. However, a key limitation is that random sampling may split nodes from the same sentence across different trees, disrupting the data's inherent structure and leading to inconsistent predictions. Despite this, we included **Random Forest** to evaluate their performance in this context.

AdaBoost: AdaBoost was chosen for its strength in focusing on misclassified samples. Using **Decision Tree** as the base learner, it incrementally corrected errors, making it particularly suited to the challenges of imbalanced data [6]. It offered a trade-off between simplicity and performance, and outperformed single **Decision Tree** in some scenarios [7].

6 Model & Architecture Comparison

In this section, we evaluated model performance across both **Giant** and **Language-Specific** architectures, using `Macro-Avg F1` as the primary metric. This captures balanced performance across both classes—`True F1` and `False F1`—with `Accuracy` also reported for broader context.

For the **Language-Specific** models, we grouped together models that demonstrated similar performance based on `F1 score` similarity with threshold of 0.85. This means that if a model trained on one language could predict the results of another language with at least 85% of similarity in `F1 score`, they were considered behaviorally similar. These similar models were then combined using the stacking method described earlier, implemented via `StackingClassifier` from `scikit-learn` library [8].

As shown in Table 3, the **Random Forest** under the **Giant Architecture** performed best overall, achieving the highest `True F1` (0.26) and `Macro F1` (0.60), indicating strong generalization to unseen data. In the **Language-Specific** setup, while all models had similar `Macro F1 scores`, **Random Forest** again led in `True F1`—a critical metric for detecting minority root classes. Though **Decision Tree** had slightly better `False F1` and `Accuracy`, **Random Forest** aligned better with our core objective (as visually illustrated in Figure 2).

Overall, the **Giant Architecture** outperformed **Language-Specific** models, likely due to larger training data. Splitting data into 21 language-specific subsets weakened the model's ability to learn reliable patterns.

Lastly, **AdaBoost** underperformed expectations. While designed to focus on hard-to-classify samples, its performance was comparable—or worse—than **Decision Tree**, especially in the **Language-Specific** setting. This likely reflects sensitivity to imbalance, small data size, and the need for more precise tuning.

7 Hyperparameter Tuning

Based on previous results, the **Random Forest** model using the **Giant Architecture** achieved the best overall performance. Consequently, we selected this model-architecture combination for hyperparameter tuning using `RandomizedSearchCV` with 500 randomly sampled hyperparameter combinations [9]. This approach was chosen over `GridSearchCV`, which would have required evaluating all 3240 possible combinations (as shown in Table 5), resulting in an expensive computational cost.

The tuning process was informed by the official **Random Forest** documentation [10], and focused on the hyperparameters detailed in Table 4. To ensure meaningful `cross-validation`, we implemented a custom strategy rather than relying on the default random 5-fold split. Since our data are structured at the node level—where multiple nodes may come from the same sentence—we ensured that all nodes from the same sentence were assigned to the same fold. This approach prevents data leakage and better reflects real-world performance.

8 Final Evaluation

Table 6 presents the top 10 parameter combinations out of the 500 tested. Displaying all combinations would not be meaningful, so only the most promising ones are shown. Notably, these top configurations share certain parameters—specifically, the same values for `criterion` and `bootstrap`—suggesting these settings are particularly effective in this context.

Across all metrics, performance improved significantly. In particular, the `f1_class_1 score` increased to approximately 0.84 across the best configurations, indicating strong generalization on the training set.

The best-performing configuration is detailed in Table 7, with corresponding results shown in Table 8. However, the `True F1 score` for this configuration is only 0.27, which is much lower than the scores achieved during hyperparameter tuning. This drop suggests that the model is overfitting to the training data. However, the optimal configuration outperforms the default setting slightly, reflecting some overall improvement.

The limited effectiveness may be influenced from inherent constraints in the dataset, such as class imbalance or suboptimal feature representation, which restrict the model's ability to generalize further.

For the final **Kaggle** submission, predictions were made on the preprocessed test set using the optimal model. For each sentence, the node with the highest predicted probability was selected as the root. The resulting submission achieved a score of 0.29, which aligns closely with the observed `True F1 score`, further confirming the model's generalization limits.

9 Conclusion

In this project, we explored the task of root prediction using several machine learning models: **Decision Tree**, **Random Forest**, and **AdaBoost**. We also applied ensemble methods such as **Stacking Classifier** and hyperparameter tuning via **Random Search**. These models were evaluated across different architectural strategies, including a large unified model (**Giant Architecture**) and **Language-Specific** models.

The best performance came from the **Random Forest** using the **Giant Architecture**. This outcome is likely due to the limited amount of data available for training separate **Language-Specific** models. In this context, a single, generalized model was better at learning broad patterns and handling unseen data.

Initially, we expected **AdaBoost** to outperform the others due to its theoretical strengths in boosting weak learners. However, in practice, it underperformed. The imbalance in the dataset and the potential predictions of the features may have limited its effectiveness.

Given more time, an interesting task for future work would be to revisit the **Language-Specific** approach using **Support Vector Machines** (SVM). Since the data would be split by language, the training sets per model would be smaller, and **SVM**'s performance—especially in imbalanced scenarios—could prove valuable. Despite its higher computational cost (worst-case cubic complexity), using SVM per language might offer improved accuracy. We also considered introducing a new architecture that involves training two separate models: one focused on predicting the positive (true) labels, and another on the negative labels. This idea stemmed from the core issue that we observed throughout the project, **data imbalance**. By separating the learning process, each model could specialize in handling its respective class distribution and make a kind of merging so that may lead to better performance and a more balanced overall prediction rate.

Nonetheless, the insights gained from this project provide a solid foundation for future classification tasks across various domains, along with a fundamental understanding of the model development workflow. This experience also helped us become familiar with the practical application of different machine learning methods.

References

[1] NetworkX Developers. Networkx: Centrality algorithms. https://networkx.org/documentation/stable/reference/algorithms/centrality.html, 2024. Accessed: 2025-05-06.

[2] Wikipedia Contributors. Centrality. https://en.wikipedia.org/wiki/Centrality, March 11 2025. Accessed: 2025-05-20.

[3] Scikit-learn. Minmaxscaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html, n.d.. Accessed: 2025-05-11.

[4] imbalanced learn. Smoteenn — version 0.13.0. https://imbalanced-learn.org/stable/references/generated/imblearn.combine.SMOTEENN.html, n.d. Accessed: 2025-05-11.

[5] Scikit-learn. 1.11. ensembles: Gradient boosting, random forests, bagging, voting, stacking. https://scikit-learn.org/stable/modules/ensemble.html, n.d.. Accessed: 2025-05-11.

[6] GeeksforGeeks. Boosting in machine learning — boosting and adaboost. https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/, May 14 2025. Accessed: 2025-05-11.

[7] Scikit-learn. Multi-class adaboosted decision trees. https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_multiclass.html, n.d.. Accessed: 2025-05-11.

[8] Scikit-learn. Stackingclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html, n.d.. Accessed: 2025-05-24.

[9] Scikit learn Developers. Randomizedsearchcv. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html, 2023. Accessed: 2025-05-29.

[10] Scikit learn Developers. Random forest classifier documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, 2023. Accessed: 2025-05-29.

A Initial Feature Selection

Label	Precision	Recall	F1-Score	Support
False	0.98	0.70	0.82	38134
True	0.13	0.81	0.22	2100
Accuracy			0.70	40234
Macro Avg	0.56	0.75	0.52	40234
Weighted Avg	0.94	0.70	0.79	40234

Table 1: Logistic Regression using Initial Features

Parameter	**Definition**	**Justification**
`Degree Centrality`	Counts direct connections.	Root nodes typically have a higher degree, linking to multiple dependents.
`Closeness Centrality`	Indicate how close a node is to all others.	Root nodes tend to have higher closeness due to their central position.
`Betweenness Centrality`	Measures how often a node lies on shortest paths.	Roots often serve as syntactic hubs, resulting in higher betweenness.
`Eccentricity`	The greatest distance from the node to any other node.	Root nodes usually have lower eccentricity, indicating central placement.
`Leaf Node Indicator`	A binary feature identifying leaf nodes.	Roots are rarely leaves, making this feature useful for exclusion. We identify the leaf node using the number of neighbors, if a node only has a neighbor, then it is a leaf.
`Farness`	The sum of all shortest path distances from the node.	Lower values suggest a more central location, characteristic of root nodes.
`Subtree Height`	Height of the subtree at the node.	The root usually has the maximum height, spanning the full sentence structure.

Table 2: Feature Justification

B PCA (Heatmap Correlation)

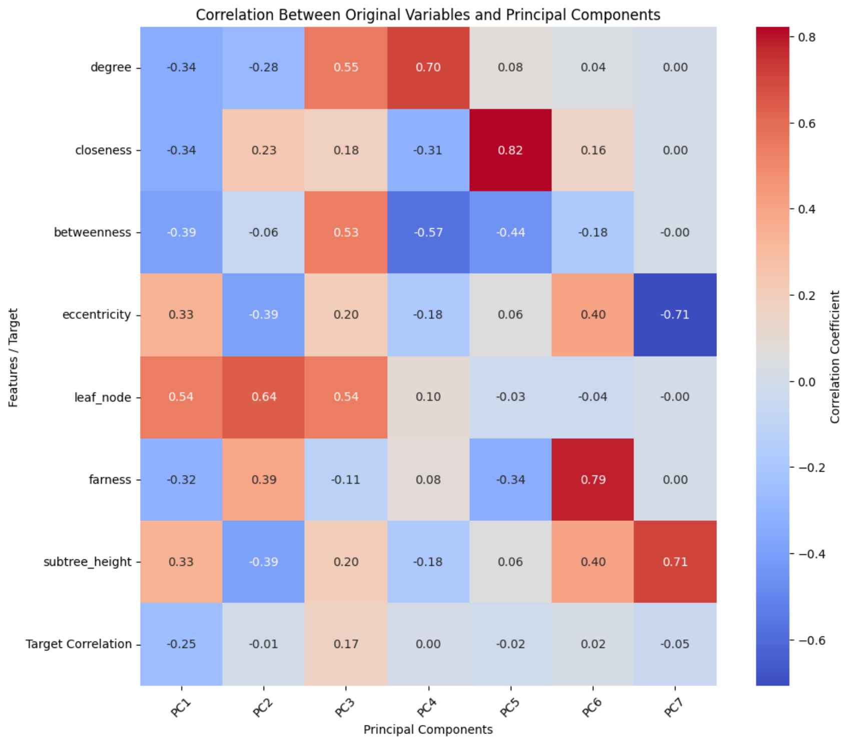

Figure 1: Heatmap Correlation

C Model & Architecture Results

Architecture	Model	True F1	False F1	Macro-Avg F1	Accuracy
Giant Model	Decision Tree	0.24	0.92	0.58	0.86
	Random Forest	0.26	0.93	0.60	0.88
	AdaBoost	0.24	0.92	0.58	0.86
Language-Specific Models	Decision Tree	0.21	0.92	0.57	0.86
	Random Forest	0.24	0.91	0.57	0.84
	AdaBoost	0.23	0.90	0.57	0.83

Table 3: Model comparison grouped by architecture

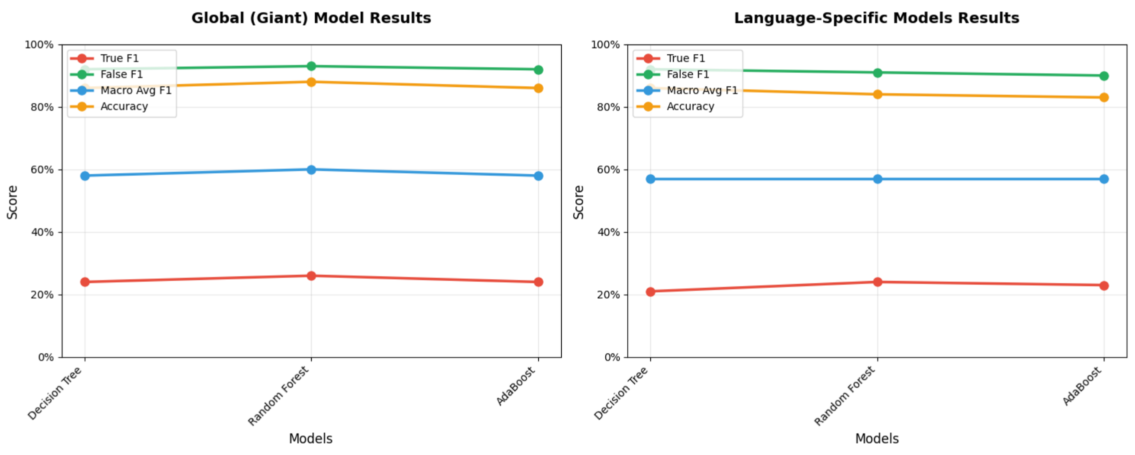

Figure 2: Model Comparison by architecture

D Hyperparameter Tuning

Parameter	**Definition**	**Justification**
`n_estimators`	The number of trees in the forest.	A higher number can improve performance by reducing variance, but also increases computational cost.
`criterion`	The function used to measure the quality of a split.	Exploring different criteria may lead to different tree structures and decision boundaries.
`max_depth`	The maximum depth of the tree.	Deeper trees can capture more complex patterns but may also overfit if not properly regularized.
`min_samples_split`	The minimum number of samples required to split an internal node.	Controls the granularity of splits and helps manage model complexity.
`min_samples_leaf`	The minimum number of samples required at a leaf node.	Helps prevent trees from learning overly specific patterns (overfitting).
`max_features`	The number of features considered when looking for the best split.	Limiting this number increases randomness among trees, enhancing generalization and reducing overfitting.
`bootstrap`	Whether bootstrap samples are used when building trees.	If `False`, each tree uses the full training set, which may address the limitation discussed.
`class_weight`	Weights associated with class labels.	Helps handle imbalanced datasets by penalizing misclassification of the minority class.

Table 4: Random Forest Selected Hyperparameters

Parameter	**Values**
`n_estimators`	100, 200, 300
`criterion`	`gini`, `entropy`
`max_depth`	50, 100, 200
`min_samples_split`	4, 6, 8
`min_samples_leaf`	2, 4, 6
`max_features`	`sqrt`, `log2`, `None`, 0.5, 0.7
`bootstrap`	`True`, `False`
`class_weight`	`balanced`, `balanced_subsample`

Table 5: Random Forest Hyperparameter Tuning

E Final Evaluation

n_estimators	**max_depth**	**min_samples_split**	**min_samples_leaf**	**class_weight**	**max_features**	**criterion**	**bootstrap**	**f1_macro**	**f1_class_0**	**f1_class_1**	**accuracy**
200	100	6	2	balanced	sqrt	entropy	True	0.886425	0.926852	0.845997	0.900831
200	200	6	2	balanced	sqrt	entropy	True	0.886425	0.926852	0.845997	0.900831
300	100	6	2	balanced	log2	entropy	True	0.886040	0.926601	0.845480	0.900492
300	200	6	2	balanced	sqrt	entropy	True	0.886040	0.926601	0.845480	0.900492
300	50	6	2	balanced	sqrt	entropy	True	0.885985	0.926560	0.845409	0.900440
100	200	6	2	balanced	log2	entropy	True	0.885785	0.926426	0.845144	0.900261
100	200	6	2	balanced	sqrt	entropy	True	0.885785	0.926426	0.845144	0.900261
100	50	6	2	balanced	sqrt	entropy	True	0.885722	0.926359	0.845085	0.900188
200	50	8	2	balanced	log2	entropy	True	0.885712	0.926222	0.845202	0.900087
300	50	4	2	balanced_subsample	log2	entropy	True	0.885711	0.926505	0.844916	0.900286

Table 6: Top 10 Random Forest Hyperparameter Combinations and Performance Metrics

Parameter	**Optimal Value**
`n_estimators`	200
`criterion`	`entropy`
`max_depth`	100
`min_samples_split`	6
`min_samples_leaf`	2
`max_features`	`sqrt`
`bootstrap`	`True`
`class_weight`	`balanced`

Table 7: Optimal Parameters for Random Forest

Label	**Precision**	**Recall**	**F1-Score**	**Support**
False	0.96	0.93	0.95	38134
True	0.23	0.34	0.27	2100
Accuracy			0.90	40234
Macro Avg	0.59	0.64	0.61	40234
Weighted Avg	0.92	0.90	0.91	40234

Table 8: Hypertuning Model Validation Results