Machine Learning Final Project

Tree Root Prediction

Authors:

Sara Sherif Daoud Saad

Oluwanifemi Favour Olajuyigbe

Email for Kaggle Competition

oluwanifemi.favour.olajuyigbe@estudiantat.upc.edu

Professor

Marta Arias Vicente

Universitat Politècnica de Catalunya

Machine Learning

1 Business Understanding

1.1 Summary of the Problem

This project aims to predict the root node in syntactic trees, which depict the sentence structure. Every tree is constructed from a sentence in one of 21 languages, with each word serving as a *node* and the connections between them serving as *edges*, signifying syntactic dependencies. Since these trees are initially *unrooted*, we are unsure of which word should be the sentence's most central or starting point.

We solve this by transforming the task into a *binary classification problem*, where we have to predict whether each node in the tree is the root (label 1) or not (label 0). We can use node-level features to train a classifier with this structure.

However, there are a number of difficulties with this issue:

- **Imbalanced data:** Since every tree has a single root, the majority of examples are negative.
- **Sentence-level dependencies:** A tree's nodes are not independent because they all originate from the same sentence.
- **Sign lengths that vary:** We must normalize these values within each tree since centrality scores change depending on the size of the sentences.

We calculate *centrality measures* from graph theory, including degree, closeness, betweenness, and PageRank centrality, in order to construct meaningful features. Each node's structural significance in the graph is intended to be captured by these scores.

1.2 Use Cases for Root Selection

Several fields of **Natural Language Processing (NLP)** benefit from knowing a syntactic tree's root:

- **Sentence parsing:** assists in determining a sentence's primary structure.
- **Translation by machine:** Sentence mapping can be enhanced by aligning root nodes across languages.
- **Keyword extraction and summarization:** The primary idea in a sentence is frequently represented by the root nodes.
- **Researchers can better understand how sentence structures are formed** in various languages by using linguistic analysis.

This task is comparable to issues in other graph-based domains outside of language processing, like: Identifying key nodes in communication or transport networks, locating the origins of errors or fault propagation systems, and locating influencers in social networks. It is also possible to view these methods as a component of a larger endeavor to examine and comprehend intricate networked data structures.

2 Data Understanding

2.1 Dataset Description

The dataset consists of syntactic dependency trees from 21 different languages. Each language includes the same set of sentences, translated into its corresponding linguistic structure. Each data point represents one sentence in a specific language, encoded as a dependency tree. The training set contains 10,500 examples (500 sentences multiplied by 21 languages).
Each example includes five key variables:

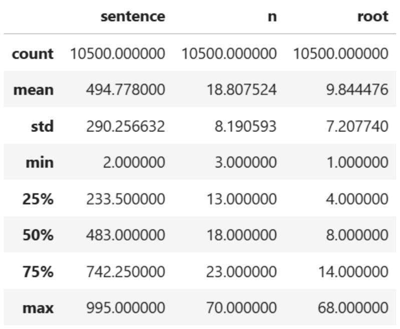
- Language indicates the language of the sentence (e.g., English, Spanish, Finnish).
- Sentence is a unique identifier that corresponds to the same sentence across all languages.
- n represents the number of nodes in the tree, which corresponds to the number of words in the sentence.
- Edgelist is a list of edges representing dependencies between words
- Root is the target variable indicating the root node

2.2 Descriptive Analytics of the Data (Initial Data Exploration)

An initial exploration was carried out to understand the structure, identify potential errors, and better understand the underlying characteristics of the syntactic dependency trees. The exploration revealed several key findings about the data quality and structural patterns.

2.2.1 Data Quality Assessment and Summary Statistics

The dataset showed good quality with no missing values or duplicate entries in all variables. However, However, one notable issue was that the `edgelist` variable was stored as a string and needed to be converted to a list for further analysis.
The summary statistics showed that sentence complexity varied a lot. The number of nodes ranged from as few as 3 nodes to as many as 70 nodes per sentence, with the majority of sentences (50% of the data) containing between 13 and 23 nodes, suggesting that the dataset comprised of both simple and complex sentence structures. The root node positions also showed a logical correlation with sentence length. Most root nodes appeared between positions 4 and 14, which aligns with the predominant sentence length range.



	sentence	n	root
count	10500.000000	10500.000000	10500.000000
mean	494.778000	18.807524	9.844476
std	290.256632	8.190593	7.207740
min	2.000000	3.000000	1.000000
25%	233.500000	13.000000	4.000000
50%	483.000000	18.000000	8.000000
75%	742.250000	23.000000	14.000000
max	995.000000	70.000000	68.000000

Figure 1: Summary Statistics

2.2.2 Tree Structure Visualization

The tree visualizations provided important insights into the directed nature of syntactic dependencies. For example, the visualization of sentence 777 (Japanese) in Figure 5 shows that each sentence forms a clear directed tree structure, where the root node is easily identified as the only node without incoming edges. Similarly, sentence 188 (Russian) exhibits the same property, with the root node clearly visible in the dependency tree. These visualizations confirm that every sentence follows a proper tree structure, having exactly one root node and directed dependencies from heads to dependents.
Although root nodes can be clearly identified through this structural view, we chose not to use this information during model training. Instead, our method lets the model learn to identify root nodes based on its centrality features.

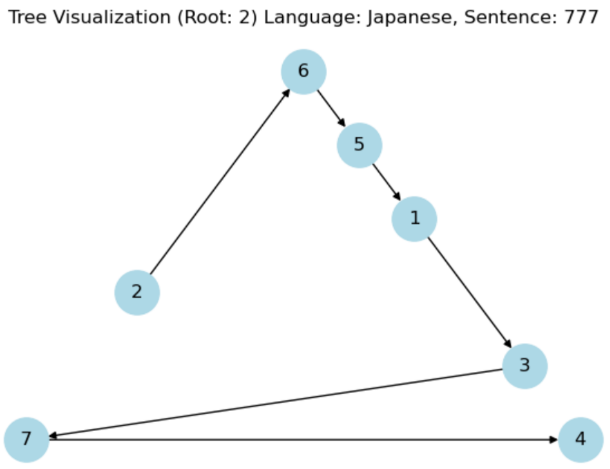

Tree Visualization (Root: 2) Language: Japanese, Sentence: 777
6
5
1
2
3
7
4

Figure 2: Tree Visualization of Sentence 777

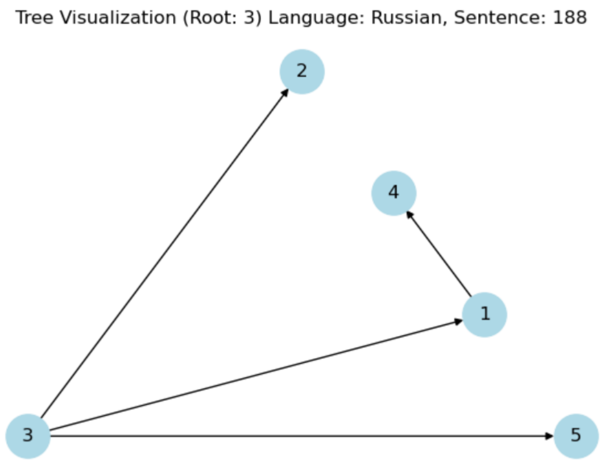

Tree Visualization (Root: 3) Language: Russian, Sentence: 188
2
4
1
3
5

Figure 3: Tree Visualization of Sentence 188

3 Data Preprocessing

3.1 Feature Engineering

We developed a comprehensive set of node-level features based on each syntactic dependency tree's graph structure in order to get the data ready for classification. First, we used NetworkX to reconstruct each sentence's tree by parsing its edge list. We calculated several feature groups for every node in the tree:

- **Centrality Measures:** Standard graph centrality metrics, such as degree, betweenness, closeness, Katz, eigenvector, harmonic, and PageRank centralities, were computed.
- **Distance-Based Features:** These consist of the range, median, standard deviation, and sum of the shortest paths between every node and every other node in the tree.
- **Subtree Features:** We calculated the size of the largest component that results from removing each node, the number of resulting subtrees (subtree count), and the degree of balance of these subtrees (subtree balance).
- **Neighborhood Features:** We computed the variance in neighbor degrees (local degree variance), the number of second-order neighbors, and the average degree and average centrality of immediate neighbors.
- **Depth and Reachability:** We measured the depth of each node from the most central node, which served as a proxy root. Along with defining metrics for neighborhood efficiency and reach ratio, we also determined how many nodes could be reached within 2- and 3-hop distances.
- **Topological Vulnerability:** We calculated the graph's edge loss upon removing a node (edge connectivity impact) and assessed how many bridges a node is a part of.
- **Combinatorial Features:** We included the ratios between specific centralities, such as betweenness-to-degree and closeness-to-pagerank, as well as the product and sum of the three main centralities: degree, betweenness, and closeness.
- **Flow and Dominance Measures:** We calculated each node's degree dominance and flow concentration, or how much its degree and betweenness dominated over its neighbors.

In order to determine whether a node is the root of its sentence tree, its features were compiled into a structured format and given a binary target label. Please consult Appendix 4, Table A.1 for a comprehensive list of engineered features utilized in our model.

3.2 Analysis of Centrality Metrics

3.2.1 Correlation

For proper analysis of the centrality metrics, we checked the correlation between the features and between each feature with the target variable. We show just some of the features considered in the correlation matrix for easier explanation of the analysis.

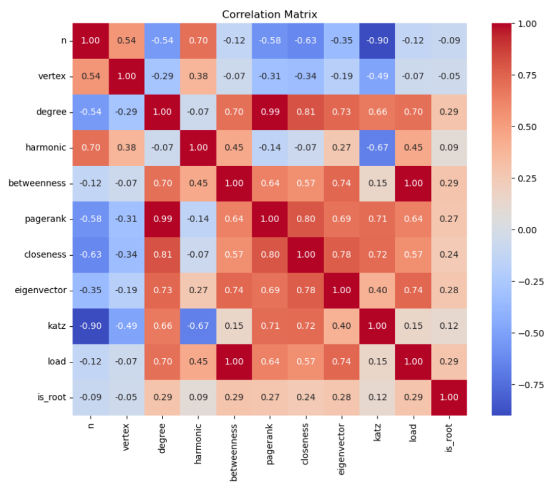


Figure 4: Correlation Matrix of Some Features Considered

The features with the highest correlation to the target variable is_root are degree (0.29), load (0.29), and betweenness (0.29). However, using all of these features together is not ideal because several are highly correlated with each other, which can introduce redundancy and multicollinearity into the model. For example:

- **Betweenness** and **load** have a perfect correlation (1.0), meaning they provide essentially the same information.
- **Pagerank** and **degree** are also very strongly correlated (0.99).

We can further see the similarity in the distribution of highly correlated features with the target variable in Figure 5.

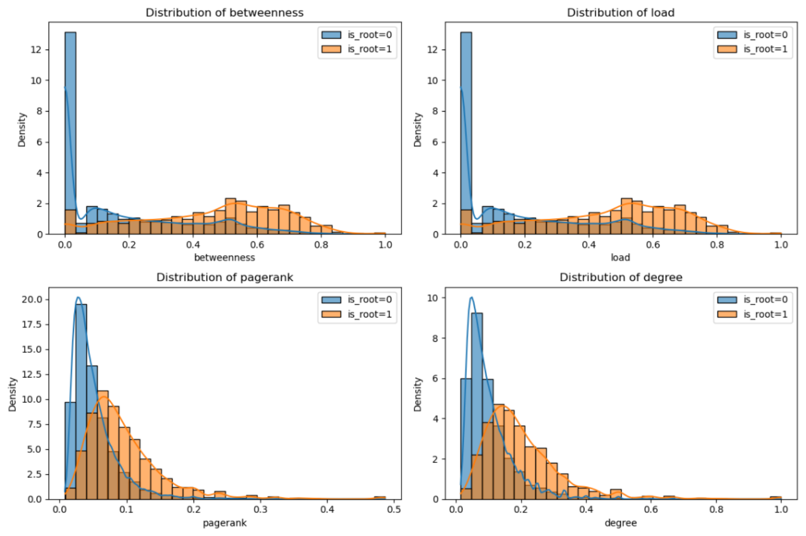


Figure 5: Distribution of Highly Correlated Features with Target Variable

Including highly correlated features together can lead to unstable model coefficients, reduce interpretability, and may not improve predictive performance. Therefore, during our feature selection we avoided using groups of features that are strongly correlated with each other and instead chose features that are both individually predictive and provide complementary information (features less correlated with each other). This approach would help build more robust and generalizable models.

3.3 Feature Selection

To improve model performance and reduce dimensionality, we implemented a two-stage feature selection process. First, we addressed multicollinearity by removing features with correlation coefficients greater than 0.85, keeping the feature from each highly correlated pair that demonstrated stronger correlation with the target variable. Then, we applied Random Forest feature importance analysis to identify the most influential predictors, removing features with importance scores below 0.01 that did not significantly contribute to model performance.

3.4 Initial Data Split

To ensure robust model evaluation, we divided our dataset into training (80%) and test (20%) sets. The test set evaluates the generalization performance of our models on unseen data and is crucial for comparing and selecting the best-performing model.
However, due to the hierarchical nature of our data, we couldn't use standard random splitting. Since multiple nodes belong to the same sentence, it is essential that all nodes from a single sentence are placed entirely in either the training or test set, but not both. Otherwise, information from the test set could unintentionally influence training, leading to data leakage and over-optimistic performance estimates. To prevent this, we employed group-aware splitting based on sentence IDs using `GroupShuffleSplit` from scikit-learn, which ensures all nodes from the same sentence remain in the same split.

3.5 Within Sentence Normalization

Since sentences in our dataset vary significantly in length and structure, normalization was essential to make centrality features comparable across different tree sizes and reduce bias from sentence length. To this effect, a global normalization across the whole dataset would not work well, as it would mix values from sentences of different sizes and structures, leading to misleading results. Instead, we normalized features within each sentence, so that node features are scaled in relation to others within the same sentence.

Given that our dataset contains parallel sentences in multiple languages, we grouped by both language and sentence ID when normalizing. This was important because the same sentence ID appears across languages, and scaling only by sentence ID would combine all languages for that sentence in the same scaling calculation. Hence, it was crucial to group by language by sentence to prevent mixing different linguistic structures during normalization and ensure each sentence is scaled relative to its own language-specific syntactic context.
`MinMaxScaler` was then used to normalize each centrality feature within each sentence to the range [0,1]. This means the node with the lowest value in a sentence gets 0, and the one with the highest gets 1, with others scaled in between. We chose MinMax scaling over standardization because it maintains positive values (important for centrality metrics which are inherently non-negative) and ensures compatibility across diverse machine learning algorithms.
We implemented separate normalization for training and test sets to prevent data leakage, ensuring our models evaluate on truly unseen data distributions.

3.6 Imbalance Handling

We looked at a number of methods to reduce bias toward the majority class and enhance recall for the minority class (i.e., root nodes) because of the stark class imbalance between root and non-root nodes in our classification task. These included class weight adjustments at the model level, synthetic oversampling, and graph-based pruning as a type of undersampling.

3.6.1 Undersampling: Pruning

We employed a custom pruning technique to remove structurally less significant nodes from the training set. Specifically, we hypothesized that nodes with low degree centrality are unlikely to be potential roots. Consequently, we eliminated the bottom 25% of nodes by degree from the majority class (non-root nodes) using the function below. Undersampling can speed up training and reduce bias toward the majority class, but it may also remove potentially useful information.

The code for undersampling and oversampling is attached in the appendix.

3.6.2 Oversampling

We used `RandomOverSampler` from the `imblearn` library to balance the underrepresented root class. Examples from the minority class are artificially replicated using this technique until a balanced distribution of classes is reached. Although it duplicates data, it can aid models in learning more about the minority class and may result in overfitting.

Despite their theoretical promise, these approaches resulted in a decrease in evaluation metrics, especially F1-score and precision. As a result, we removed these steps from our final pipeline and used the original, unbalanced dataset to train the model.

4 Modeling and Validation

4.1 Methods

After preprocessing the data, we considered the following training models.We selected modeling tecniques based on the nature of our data and the binary classification problem and implemented both linear and non-linear methods wheich are as follows.

4.2 Linear Methods

- **Logistic Regression**: This basic linear model is applied to binary classification. It uses a linear combination of a node's features to estimate the likelihood that it is the root. Simple and easy to understand, logistic regression offers a solid foundation for more intricate models.

- **Linear Discriminant Analysis (LDA)**: LDA determines which linear feature combination best divides two or more classes. It makes the assumptions that the features are normally distributed and that every class has the same covariance matrix. It was appropriate for our problem since it improves efficiency and performance in early experiments by reducing dimensionality while preserving class separation.

4.3 Non-Linear Methods

- **Quadratic Discriminant Analysis (QDA)**: QDA, unlike LDA, allows each class to have its own covariance matrix. It will be particularly effective in capturing more complex boundaries if the features of the root and non-root nodes have different variance structures. It was also a good choice since the number of examples per class was much larger than the number of dimensions, ensuring that we will not end up with a singular covariance matrix.
- **Decision Trees**: Decision Trees are interpretable models capable of capturing non-linear relationships between features. They were chosen as a baseline to evaluate the performance of more advanced ensemble methods such as Random Forests and LightGBM. However, they are prone to overfitting, especially when the tree becomes too deep or the dataset is noisy.
- **Random Forest**: Random Forest builds on the strengths of decision trees while addressing their key limitations, particularly overfitting. It does this by training multiple decision trees on bootstrapped subsets of the data and aggregating their predictions. This ensemble method hence improves generalization because while individual trees may overfit, the aggregation would help mitigate it. Random Forests also retain some interpretability through feature importance analysis, which was especially helpful in identifying the most influential features for root prediction and guiding our feature selection. Additionally, they are very valuable as they provide methods for handing imbalanced dataset, a key characteristic of our dataset. With this model, the training process can also be parallelized, enhancing computational efficiency.
- **Multi-Layer Perceptron (MLP)**: MLPs were chosen because of its ability to capture complex, non-linear relationships in our data and capture patterns missed by simpler models.
- **LightGBM**: LGBM is a gradient boosting algorithm that builds decision trees in a leaf-wise manner and is able to capture complex non-linear relationships between features. Similar to Random Forest, it also has the class weigthing feature that lets it handle imbalanced data and also has feature importance analysis.
- **K-Nearest Neighbors (KNN)**: A node is categorized using the majority class of its k nearest neighbors by the non-parametric KNN algorithm. KNN is capable of capturing local structure in the data despite its simplicity. Because of the size of our dataset, it was computationally costly, but it provided a baseline to test how well local feature similarities alone could differentiate root nodes.
- **Naive Bayes**: Given the class label, Naive Bayes makes the assumption that every feature is conditionally independent. For high-dimensional data, it can be surprisingly effective despite its simplicity and strong independence assumptions. It was employed to evaluate the predictive power of the probabilistic distribution of individual features for root status.

4.4 Resampling and Validation Protocols

Given the nature of our data and the dependencies between nodes in the same syntactic tree, we used group-aware cross-validation to avoid data leakage and overoptimistic performance estimation. Specifically, we used GroupKFold (5 folds), grouping on the sentence ID to ensure all nodes from a sentence were entirely contained in either the training or validation set for each fold.

To handle imbalance in the data, we used either:

- `class_weight='balanced'` for Logistic Regression, Decision Trees and LightGBM.
- `class_weight='balanced_subsample'` for Random Forest.

This makes the models give importance to the minority class.

Additionally, we used F1-score as the evaluation metric, which prioritizes performance on the minority class by balancing precision and recall.

4.5 Hyperparameter Tuning

All models were tuned using `GridSearchCV` with the same `GroupKFold` setup (5 splits grouped by sentence ID). We optimized models based on the F1-score to handle class imbalance effectively and enabled prallel processing `n_jobs=-1`) for computational efficiency.

All the hyperparameter values were chosen from a grid of commonly used values.
The following are the tuning strategies implemented for each model:

Model	**Tuned Parameters**
Logistic Regression	Regularization strength, penalty, and solver
LDA	Shrinkage and solver
KNN	Number of nearest neighbours and distance metric (to adapt to feature scaling and data distribution)
MLP	Number of hidden layers, activation function, alpha, and solver
QDA	Regularization parameter
GaussianNB	No parameters to tune
Decision Tree	Maximum depth, splitting criteria (min_samples_split and min_samples_leaf)
Random Forest	Maximum depth, number of trees, and splitting criteria (min_samples_split and min_samples_leaf)
LightGBM	Number of trees, maximum depth, number of leaves, splitting criteria, and learning rate

Table 1: Tuned parameters for each model

4.6 Best Parameters per model

Model	**Best Parameters**
Logistic Regression	C: 1, penalty: l2, solver: lbfgs
LDA	Shrinkage: None, solver: svd
KNN	Metric: manhattan, n_neighbors: 1
MLP	Activation: tanh, alpha: 0.0001, hidden_layer_sizes: (100, 50), solver: adam
QDA	reg_param: 0.1
GaussianNB	No parameters tuned
Decision Tree	max_depth: 10, min_samples_leaf: 5, min_samples_split: 5
Random Forest	max_depth: 50, min_samples_leaf: 10, min_samples_split: 5, n_estimators: 250
LightGBM	learning_rate: 0.1, max_depth: 50, min_child_samples: 10, n_estimators: 250, num_leaves: 64

Table 2: Final tuned hyperparameters for each model

5 Evaluation

5.1 Evaluation Metrics

Given the severe class imbalance in our dataset, we did not consider accuracy in evaluating the performance of our models as it will be misleading and uninformative. Therefore, we focused exclusively on evaluating the minority class where the root label is 1 using **Precision, Recall, and F1-Score**. These metrics directly measure the model's ability to correctly identify roots, which is our primary goal. We also did not consider overall accuracy and average scores (macro and weigthed) of the Precision, Recall and F1-score metrics since they would be dominated by the majority non-root class and would not reflect root prediction performance.

5.2 Model Comparison

To compare the performance of each model, we printed the classification report to evaluate how well it performed in classifying the root node.
We have documented the results in the table below:

Model	Precision	Recall	F1-Score
Logistic Regression	0.14	0.80	0.25
LDA	0.32	0.29	0.31
KNN	0.21	0.22	0.21
MLP	0.59	0.06	0.11
QDA	0.31	0.31	0.31
Gaussian Naive Bayes	0.15	0.72	0.25
Decision Tree	0.14	0.75	0.24
Random Forest	**0.28**	**0.49**	**0.36**
LightGBM	0.20	0.65	0.31

Table 3: Comparison of Models Based on Precision, Recall, and F1-Score

Estimation of Generalization Performance: We used the test set previously split to estimate the generalization error on unseen data. The model's performance on the test set showed minimal overfitting and stable generalization, which was in line with the validation scores.

5.2.1 Performance Analysis by Model

The table shows several important patterns:

- With the highest F1-score (0.36), **Random Forest** demonstrated the best trade-off between recall and precision. This demonstrates how well it handles class imbalance and structured, high-dimensional feature sets.
- **Logistic Regression** and **Naive Bayes** demonstrated very low precision (0.14 and 0.15, respectively) but high recall (0.80 and 0.72, respectively). These models frequently produce false positives because they categorize an excessive number of nodes as roots.
- **MLP** showed a very low recall (0.06) and a high precision (0.59), indicating that it was overly conservative and missed the majority of real roots.
- **QDA** and **LDA** did fairly well in terms of balanced precision and recal. This task seems to have benefited somewhat from QDA's adaptability in modeling various covariance structures.
- **Tree-based models** continuously achieved Strong recall (Decision Tree, Random Forest, and LightGBM), with Random Forest performing the best overall.

5.3 Best Model Chosen

Because of its balanced performance and superior F1-score, Random Forest was chosen as the best model. Random Forest successfully found roots while keeping a manageable false positive rate, in contrast to models that were too conservative or overfitted to the dominant class. Other benefits consist of: model interpretability via feature importance scores, robustness to overfitting via ensemble aggregation, computational efficiency with parallel training support, and built-in handling of class imbalance through class weights.

5.4 Fit the Chosen Model on Entire Train-Test Split

After Random Forest was chosen, we tested its ultimate generalization ability on the held-out test set and retrained it on the complete training set. This stage made sure the model produced an objective performance estimate and benefited from all available labeled data.

> *Note: All hyperparameter tuning was performed exclusively on training and validation folds. The test set was never used in model selection.*

5.5 Top-1 Root Prediction using Best Random Forest Model

We started by creating probability scores for every node in the test set using our top-performing model, the Random Forest classifier. We calculated the likelihood that each node is the root following feature scaling and preprocessing:

We first applied our best-performing model, the Random Forest classifier, to generate probability scores for each node in the test set. After preprocessing and feature scaling, we computed the probability that each node is the root:

1. The Random Forest model forecasted a root probability for every node in every sentence.
2. The node with the highest probability was chosen as the predicted root (Top-1) after we grouped the predictions by sentence. (Top-1).

Together with the initial test dataframe, this prediction was formatted for submission in accordance with the competition's specifications.

5.6 Interpretability of Results

The Random Forest model's capacity to give input features importance scores is one of its main advantages. This allowed us to determine which graph-based features were most important for root prediction.

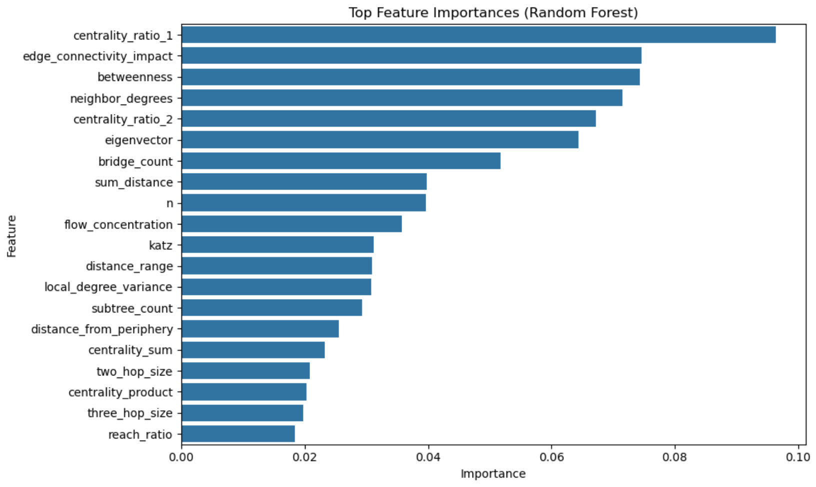


Figure 6: Top Feature Importances (Random Forest)

We examined feature importances from the trained Random Forest classifier in order to comprehend how the model made its predictions. Every feature is ranked by the model according to how much it influences decision-making for every tree in the ensemble.

centrality_ratio_1, which is the ratio of a node's betweenness centrality to degree centrality, was the most significant feature. In relation to the number of its direct neighbors (degree), this ratio expresses how structurally significant a node is in terms of connecting paths (betweenness). An insightful signal for locating root nodes, which frequently serve as central pivots in sentence trees, a high value indicates that a node plays a crucial bridging role despite not being overly connected.

Other features that are highly ranked are:

- **Impact of Edge Connectivity**: calculates the decrease in graph connectivity caused by the removal of a node. The removal of root nodes severely fragments the tree's structure because they frequently act as vital connectors.
- **Betweenness Centrality:** demonstrates that root nodes play a crucial role in syntactic structure by frequently being located on the shortest paths between other nodes.
- **Neighbor Degrees:** Additionally predictive was the average degree of nearby nodes, indicating that highly connected substructures frequently encircle root nodes.
- **Centrality_Ratio_2:** Another relational metric that compares PageRank and closeness shows how powerful a node is based on both connection strength and distance.
- **Eigenvector Centrality and Bridge Count:** These characteristics highlight how crucial it is to participate in the crucial linkages in the graph and to be connected to other significant nodes.
- **Sum Distance and Node Count (n):** The structural patterns of root nodes scale with the size of the tree, and they are typically centered.

Subtree counts, neighborhood sizes, and local variance measures were lower-ranked but still significant features that improved predictions in more unclear situations.

In summary, the model's dependence on subtle structural ratios, particularly **centrality_ratio_1**, shows that it can identify roots not just by their raw centrality but also by how their influence relates to their local connectivity.

6 Scientific and Personal Conclusions

This project demonstrated that although using structural graph features to predict root nodes in syntactic dependency trees is a promising approach, the task's difficulty under significant class imbalance was reflected in the modest overall F1-score. Nevertheless, we learned a lot about the limitations of existing models, how to create meaningful evaluation strategies, and which graph-based features are most informative. Working with noisy, unbalanced real-world data requires careful feature engineering, appropriate validation, and reasonable expectations, as the experience demonstrated.

7 Apendix

7.1 Engineered Features and Their Descriptions

Table 4: Engineered Features and Their Descriptions

Feature Name	**Description**
degree	Degree centrality (number of direct connections)
betweenness	Betweenness centrality (control over information flow)
closeness	Closeness centrality (inverse of average distance to all nodes)
katz	Katz centrality (influence including distant neighbors)
eigenvector	Eigenvector centrality (influence of neighbors)
sum_distance	Sum of shortest path distances to all nodes
distance_from_periphery	Distance to the closest periphery node
subtree_size	Size of largest component after removing the node
neighbor_degrees	Average degree of immediate neighbors
neighbor_centralities	Average centrality of immediate neighbors
second_order_neighbors	Count of second-order neighbors
relative_centrality	Node's degree centrality relative to max in tree
depth_from_proxy	Shortest path length from a proxy root node
bridge_count	Number of bridges the node is part of
local_degree_variance	Variance in degrees of neighboring nodes
median_distance	Median of shortest path distances
distance_std	Standard deviation of shortest path distances
distance_range	Range (max-min) of shortest path distances
subtree_balance	Balance of component sizes after node removal
subtree_count	Number of components after node removal
two_hop_size	Size of the 2-hop neighborhood
three_hop_size	Size of the 3-hop neighborhood
neighborhood_efficiency	Fraction of graph reached within 3 hops
reach_ratio	Fraction of graph reached within 2 hops
centrality_product	Product of degree, betweenness, and closeness
centrality_sum	Sum of degree, betweenness, and closeness
centrality_ratio_1	Betweenness to degree ratio
centrality_ratio_2	Closeness to PageRank ratio
edge_connectivity_impact	Edge loss after removing the node
flow_concentration	Betweenness concentration vs. neighbors

7.2 Oversampling and Undersampling Code

7.2.1 Undersampling

```
import ast
import networkx as nx
from tqdm import tqdm

def prune_low_degree_nodes(edgelist_str, retain_percent=0.75):
    edges = ast.literal_eval(edgelist_str)
    G = nx.Graph(edges)
    if len(G.nodes) == 0:
        return edges
    degrees = dict(G.degree())
    sorted_nodes = sorted(degrees.items(), key=lambda x: x[1])
    keep_n = int(len(sorted_nodes) * retain_percent)
    keep_nodes = {node for node, _ in sorted_nodes[-keep_n:]}
    pruned_edges = [(u, v) for u, v in edges if u in keep_nodes and v in keep_nodes]
    return pruned_edges

tqdm.pandas()
X_train_normalized['edgelist'] = X_train_normalized.progress_apply(
    lambda row: prune_low_degree_nodes(row['edgelist']) if row['language'] == major_class else
     ↪  row['edgelist'],
    axis=1
)
```

7.2.2 Oversampling

```
from imblearn.over_sampling import RandomOverSampler
import pandas as pd

X_features = X_train_normalized.drop(columns=['target', 'language', 'sentence', 'node',
 ↪  'group'])
y_target = train_df['target']

ros = RandomOverSampler(sampling_strategy='minority', random_state=42)
X_balanced, y_balanced = ros.fit_resample(X_features, y_target)

resampled_ids = train_df[['language', 'sentence',
 ↪  'node']].reset_index(drop=True).iloc[ros.sample_indices_]
resampled_data = pd.concat([
    resampled_ids.reset_index(drop=True),
    pd.DataFrame(X_balanced, columns=X_features.columns),
    pd.Series(y_balanced, name='target')
], axis=1)
```

7.3 Github Repository

Kindly review the full code in the following Github repository :
https://github.com/saracherif123/tree-root-prediction