Finding the root in free undirected syntactic trees

Biel Manté Marc Parcerisa
Facultat d'Informàtica de Barcelona (FIB), UPC, 08034 Barcelona, Spain

***Abstract*—This paper addresses the prediction of root nodes in free, undirected syntactic dependency trees from 21 languages, reframing it as a binary classification problem. The methodology relies on graph-theoretic centrality measures and engineered topological features. A vast set of machine learning models were systematically evaluated, with most undergoing hyperparameter optimization. The core objective was maximizing sentence-level root identification accuracy.**

I. Introduction

This project tackles the task of predicting the root node in syntactic dependency trees that are provided in the form of free, undirected graphs. While the problem may not be a central concern in linguistic theory, it serves as an ideal setting for experimenting with modern machine learning techniques on structured data. The dataset comprises syntactic trees for 995 sentences across 21 different languages, where each tree corresponds to a sentence and consists of nodes representing words and undirected edges encoding syntactic dependencies. The challenge lies in identifying the root word of the sentence, a task that we reframe as a binary classification problem at the node level, predicting whether each node is the root or not. The core of our approach relies on leveraging centrality measures derived from graph theory, as well as complex engineered features based on the tree's topology to characterize the structural prominence of each node.

In Section II, we dive into the structure and properties of the dataset, describe the feature engineering process, and analyze inter-feature relationships and potential language-specific patterns. Section III presents the various models we explored, detailing the data representations, training procedures, and evaluation strategies used to compare them. The subsequent Section IV focuses on the results obtained from the most promising models, offering a closer look at their predictive performance and how they relate to the structural characteristics of the data. Finally, Section V summarizes our main findings and evaluates the effectiveness of our best model.

II. Data Exploration

The entirety of the data set used is comprised of 995 sentences in 21 languages each. However, only 500 of them are available for training, which means that it contains exactly 10,500 trees with their root labeled, where each node in the tree corresponds to a word in the sentence, the node labels are the position, and each edge corresponds to a syntactic dependency of which the direction is unknown. Apart from the list of edges of the tree and the label of the root, the dataset also contains the language of the sentence from which the tree was taken, as well as the number of nodes in the tree. An example of an un-rooted and rooted tree taken from the dataset can be seen in figure 1.

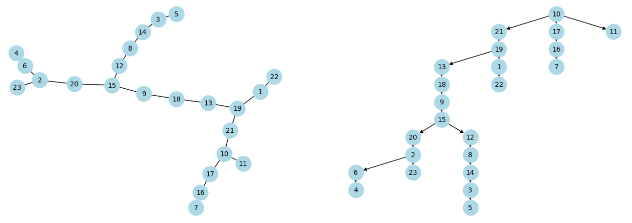

Fig. 1: A) Un-rooted, free syntactic tree generated from a sentence in Japanese. B) Same sentence, rooted.

If we take a look at the distribution of the lengths of the sentences (Figure 2), the first issue with this format of data becomes apparent: There's a high variability in the number of nodes per each sentence. Thus, all attempts of modeling the problem as a per-sentence classification would require a variable length of input, and a model capable of exploiting the topology of the tree, which is intrinsically non linear.

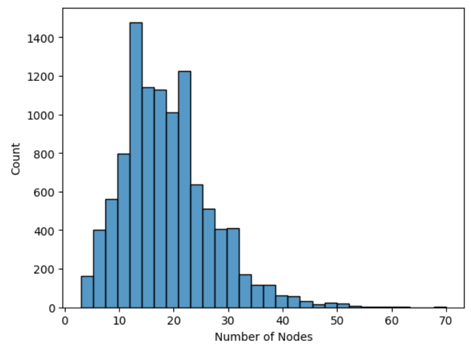

Fig. 2: Distribution of the count of nodes per sentence in the dataset.

Thus, a new dataset was created, which consisted of an unwound dataset, where each row corresponds to a node in a sentence of the original dataset. It's important to note that in this new dataset, a huge class imbalance is present between the classes ”root” and ”no root”, as the sentences have an average length of 18.8, from which only one node can be a root. The resulting dataset contains 197479 rows, from which 186979 are classified as ”not root”, as can be seen in figure 3.

In the process of unwinding the trees, several features are computed per each node, which will hopefully provide a strong basis for the modeling task at hand. Table I in appendix B provides an outline of all the features that were ultimately

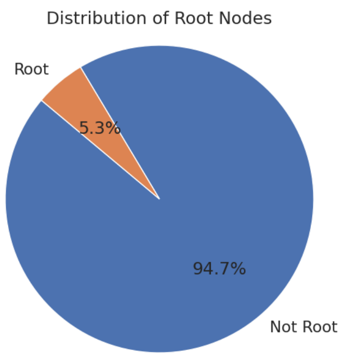

Fig. 3: Distribution of ”root” nodes versus ”not root” nodes in the unwound dataset.

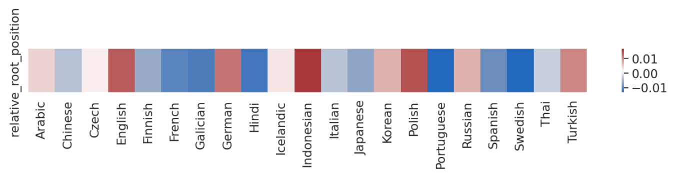

Fig. 4: Correlation between the relative position of the syntactic root of a sentence and the language it is written in.

taken into consideration. As can be seen in figure 6 in the appendix, many of the features inserted are strongly correlated between them, as expected. Most importantly, in the same figure, some features with a slight correlation with variable ”is_root” can be observed, which we expect to be the ones to gain more importance in the subsequent modeling steps.

As a preliminary observation that can be discovered by simple exploratory data analysis, the expected relative position of the root node in the sentence shows a slight correlation with some of the languages, as is shown in figure 4, as per which, Portuguese and Swedish sentences are expected to have the root slightly towards the start of the sentence, as opposed to Indonesian, Polish or English, which have them at the end.

III. Model Exploration

In this section, several modeling techniques were explored systematically to predict the root of each of the sentences. All models explored in this section follow the same paradigm: Predicting the probability of each node to be a root of the sentence, given any combination of the features explained in the previous section. Then, by grouping the nodes based on the sentence they belong to, the one with higher probability of being the root is chosen as such.

Two metrics were used as the main comparison point between any two models: Node-based accuracy, and sentence-based accuracy, which are computed as the number of correctly classified nodes over the number of nodes in the dataset, and the number of sentences for which the actual root is the one with higher probability over the total number of sentences, respectively.

The general strategy used for the model selection was as follows: The training dataset (197479 nodes) was split into two smaller ”private training” and ”private testing” datasets, with 80% and 20% of the data respectively, ensuring that all nodes from the same sentence ended up in the same dataset, to avoid information spills. Then, several models were tested by either training them using only the ”private training” dataset, or by further splitting it down into five cross-validation folds to use them to optimize their hyper-parameters. At the end, all models were evaluated against the ”private testing” dataset to choose a handful that yielded the highest test accuracy, which was taken as a measure of the model's generalization capabilities.

Once a model was selected, a final tuning of its hyper-parameters was performed by using five-fold cross validation on the entire training dataset. Then, a proper training was performed, using the full dataset, as well as the optimized hyper-parameters. See figure 7 appendix A for the outline of the complete model selection strategy followed.

Here's a list of the models tested and the results obtained:

- **Random Model Baseline**
  Before proceeding, let us introduce two important baselines for our accuracies: The node-based and sentence-based accuracies of a random model, which is defined as a completely unfitted model, which makes fully random predictions.

  Trivially, if a model predicted randomly whether a node is or not a root (50% of the times predicting either option), only 50% of the predictions would be correct — Half of the *not root* nodes would be classified as *root* and vice-versa. Thus, this would result on a 50% node-based accuracy. A more intelligent approach would be to predict all the nodes as the *not root* category, which would yield a 94.7% accuracy, as seen in figure 3.

  Less intuitively, if a model predicted randomly which node in a sentence is the root, choosing a word from the sentence with equal probability, the probability of having chosen the correct one would depend on the length of the sentence:

$$P(Y = y_R|L) = \frac{1}{L}, \tag{1}$$

  where Y is the chosen word as a random variable, y_R is the index of the correct root, and L is the length of the sentence. This is also the expected accuracy of a random model classifying sentences of size L. Thus, the expected accuracy overall can be computed as follows:

$$P(Y = y_R) = \sum_{i=1}^{\infty} P(L = i)P(Y = y_R|L) \tag{2}$$

  where $P(L = i)$ is the probability of a sentence picked at random from the dataset to have length i, which can be computed from the histogram in figure 2. The resulting expected sentence-based accuracy of a random model is 6.6%.

- **Statement Baseline**
  As a first fitted model baseline, a Random Forest Classifier was trained on a subset of the features used — specifically, the features proposed in the project statement. These features are: `harmonic_centrality`, `degree_centrality`, `betweenness_centrality`, `pagerank` and `language`. For this test, the default parameters of the `RandomForestClassifier` object from `sklearn` were used: 100 classifiers, without a set maximum depth, with *gini* as the criterion. The only parameter that was changed was the class weights, which were set to `"balanced"`, meaning that the module computes the weights based on the following formula: `n_samples / (n_classes * np.bincount(y)`.

  The model was fitted to the entire *inner training* dataset, and validated with the *inner testing* dataset, and resulted in a node-based accuracy of 93.8%, and a sentence-based accuracy of 21.3%. Predictably, the node-based accuracy of the baseline model is extremely high, although lower than the naive approach explained in the previous baseline. This is most likely because of the class weights, which cause the model to predict more often the *not root* category than the real proportion in the dataset. Also as expected, it yielded better sentence-based results than the naive approach of the previous baseline, but still fairly bad ones, as very little features were used, and no tuning was performed.

- **Logistic Regression**
  This model was also used as sort of a baseline for what a model could achieve with the extended set of features. Its hyper parameters were not optimized using cross validation. Instead, the default parameters were used, with the following added modifications: `"balanced"` class weights were chosen (explained in the previous model), `penalty` was set to `"l1"`, with the value for the regularization strength of 0.1 (`C` parameter was set to 1/0.1=10).

  The model was then fitted to the full *inner training* dataset using the *liblinear* solver, and tested using the entire *inner testing* one, and yielded a 73.5% node-based accuracy, and a 27.1% sentence-based. Not surprisingly, the new model is much better at detecting the actual root of the sentence, but at the expense of predicting many more nodes incorrectly. This, however, is not necessarily bad, as the only thing important is that it predicts a higher probability of being the root for the actual root.

- **Linear Discriminant Analysis**
  The initial exploration within the Bayesian family of models involved Linear Discriminant Analysis (LDA). Given the complexity of our classification task and the amount of features considered, we anticipated that LDA, which constructs simple linear decision boundaries, would face limitations. We expected that linear boundaries would not be sufficient for effectively classifying our target classes.

  The model yielded a node-based accuracy of 93.4%. While this surpassed the performance of Logistic Regression, it did not surpass the Baseline result. In terms of sentence-based accuracy, the outcome was reversed: the model outperformed the Baseline, reaching up to 26.3%, but failed to improve upon the Logistic Regression model, indicating a decrease in performance for this specific metric relative to Logistic Regression.

- **Quadratic Discriminant Analysis**
  Following the evaluation of LDA, Quadratic Discriminant Analysis (QDA) was assesed. A slight improvement in performance was anticipated, given QDA's's capacity to establish more complex, non-linear decision boundaries.

  However, it is needed to note a known limitation of it: potential sensitivity to collinear predictor variables. Our prior exploratory data analysis indicated the presence of such collinearity within our dataset.

  The final results were far better than expected. The node-based accuracy registered at only 51.6%, a figure considerably lower than previous models. This initially suggested a low sentence-based accuracy as well. However, the sentence-based accuracy unexpectedly reached 41.5%, a substantial improvement that significantly outperformed all prior models on this metric.

- **Naive Bayes**
  Our model exploration continued with Naive Bayes, from which we anticipated performance characteristics comparable to its related discriminant analysis counterparts, LDA and QDA.

  Using the default NB configuration, the model achieved a node-based accuracy of 78.4% and a sentence-based accuracy of 28.9%. While these results surpassed the performance of LDA, they did not reach the levels attained by QDA.

  To potentially enhance robustness and optimize performance, we tunned the var_smoothing hyperparamete using a 5-fold cross-validation . The cross-validation process identified an optimal var_smoothing value of $\sigma = 0.1$.

  However, when the NB model was retrained on the entire training dataset using this supposedly optimal v value and evaluated on a separate validation set, we observed a decrease in performance compared to the default configuration.

- **Random Forest**
  We expected this model to yield fairly good results,

due to its intrinsic non-linearity. Before fitting it to the entire *inner training* dataset, five cross-validation folds were created, and used to determine the best combination of hyper parameters from a list of 648 options, which include all possible combinations of the following values:

`n_estimators`	50, 100 or 200
`criterion`	*gini*, *entropy* or *log loss*
`max_depth`	20, 40, 50 or *None*
`max_features`	*sqrt*, *log2* or *None*
`oob_score`	*True* or *False*
`class_weight`	*balanced*, *balanced subsample* or *None*

To select which values were acceptable for the `max_depth` parameter, a first Random Forest Classifier was fitted to the *inner training* dataset, and its estimators (the trees themselves) were inspected to get their depth, which ranged from 30 to 60. However, the *no limit* option was also left for the fine tuning.

The model with highest node-based accuracy[1] that was found used 200 estimators, with the *entropy* criterion, $log_2(N)$ of the features for the training of each tree, scoring each one of them based on their Out Of Bag (OOB) score, and, most surprisingly, without class weights and with a maximum depth of 20, which is much lower than the counts found in the aforementioned first random forest trained.

Training this model with the full *inner training* dataset, and testing it on the *inner testing*, it yielded a 94.6% node-based accuracy, and a 27.4% for sentence-based. This model showed a higher node accuracy than the linear model without loosing any sentence accuracy, which hints that it is properly learning to discern root and non root nodes.

- **eXtreme Gradient Boosting**
  Similarly to Random Forest we also excepcted XGBoost to boost our perfomance metrics and be a better classifier than than the more simplistic ones we tested previously. A 5-fold cross validation was also performed in order to find the configuration that optimized the **node / sentence accuracy**. The grid that was generated contained the combinations of the following hyperparamter values.

`Learning Rate`	0.01 or 0.1
`Max Depth`	10, 20 or 50
`N estimators`	50 or 100 *None*
`Min Child Weight`	1, 5 or 10
`Randomly Selected features`	70% or 60%
`L2 Regularization $\lambda$`	0.1, 1 or 5

Other possible hyperparamters such as .... were also considered, as well as adding more values to the existing ones. Both experiments were discarded due to de high computational cost that they implied, and the features and values we believed that would add more value to the study were kept.

Out of he 216 configurations tested, the one that led to a better **node/sentence** accuracy was:

`Learning Rate`	0.1
`Max Depth`	10
`N estimators`	50
`Min Child Weight`	5
`Randomly Selected features`	0.7
`L2 Regularization $\lambda$`	5

The optimal XGBoost configuration utilized a of 0.1, moderately faster than typical starting defaults. The model constructed an ensemble of 50 trees , where each tree had a maximum depth of 10 levels. To promote diversity and reduce variance, 70% of features were randomly selected for building each tree. A minimum child weight of 5 was applied, making splits more conservative. Finally, a relatively strong `L2` regularization $\lambda = 5$ was selected.

With this optimized configuration, the XGBoost model achieved a node-based accuracy of 94.7% and a sentence-based accuracy of 27.8%. This performance was comparable to the one with the Random Forest model, which had been tuned with the same methodology.

- **K Nearest Neighbors**
  KNN classifiers employ a voting strategy for determining the class of an unseen data point. Due to the extreme class imbalance of the dataset, we expected this model to have a very low accuracy, as we considered that there weren't enough *root* nodes to form proper neighborhoods.
  Nonetheless, a cross-validation strategy was applied to select the based hyper parameters combination between the following values:

`n_neighbors`	2, 3, 5, 8 or 10
`weights`	*uniform* or *distance*
`algorithm`	*auto*, *ball tree*, *kd tree* or *brute*
`p`	1 or 2

Where `p` is the value of p in the *Minkowski distance*.

80 combinations of hyper parameters were explored and, surprisingly, the model that yielded the best node-based accuracy used $k = 10$ neighbors with uniform weights, with the *kd tree* algorithm, and using the $p = 1$ Minkowski distance, and it yielded a 94.6% node accuracy and a 28.1% sentence accuracy. This unexpected success, combined with the high value for k, disproved our initial hypothesis of there not being proper neighborhoods, and meant that there should be some neighborhoods present in the dataset. However, after plotting the result of an MDS (figure 5), the supposed region was not very prominent.

- **Multi-Layer Perceptron**
  Another model considered for its strong non-linearity was an MLP, which we expected to yield rather good

[1]Node-based accuracy is much faster to compute than sentence-based, reason for which it was chosen for the hyper parameter tuning.

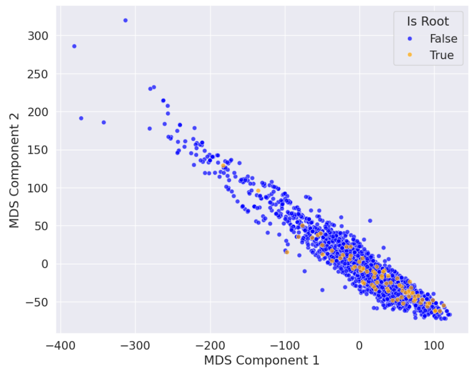

Fig. 5: MDS plot of 2000 extended node data points, colored by the *is root* flag.

results. Instead of implementing it using the existing MLP implementation from *sklearn*, which didn't allow for class weights to be specified, we went for a custom implementation using *tensorflow* and *keras*. Our implementation has the hyper parameters shown in table II in appendix B.

Using five-fold cross-validation to train each model for 10 epochs, 8748 combinations of hyper parameters were compared, and the model with best accuracy had 3 layers of sizes 64, 64 and 32, plus the final one, with regularization of 0.1 L1 and 0.01 L2 only in the first one, and dropout of 0.2 after each of the layers. Finally, the optimizer that showed the best performance had a learning rate of 0.001, with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$.

However, after training a model with these parameters over the entire *inner training* dataset with a limit of 100 epochs and *early stopping*, it only yielded an 85.0% node-based accuracy, and a 26.3% sentence-based. These results impressed us considerably, as we initially assumed that this kind of model would be the one to give higher accuracies, and we still think that a deeper model would have been able to capture deeper intricacies of the dataset, and correlations with the language of the sentences.

As a bonus test, we tried to fit a Graph Neural Network on the untreated dataset, by using the library `torch-geometric` to train a GraphSAGE model, which creates low-dimensional vector representations of the nodes based on both some centrality measures, as well as the actual topology of the graph, to then classify each node into either root or not root. The results obtained were so good that they led us to discover that the ordering of the edges in the original dataset caused the first node in the `networkx` representation of the tree to be the root in most cases. Of course, we considered our results as invalid and discarded the approach.

IV. Results

Across all models tested we found that QDA, a model for which we didn't need to find it's best configuration nor apply any kind of tuning methodology, proved to be the best one at classifying at a sentence level, which was the core objective of the project. This result was unexpected, as we had anticipated that more complex models involving elaborate tuning processes would ultimately achieve better outcomes.

This model has a main drawback that still needs to be taken care of: dealing with collinearity. We observed in the data exploration that some variables are strongly correlated, some of them being even linear combinations of each other or subsets of one another. This can lead the model to be very volatile, meaning that it's results are highly influenced by the data splits.

To deal with this we decided to apply CV to find the best combination of features that yield a stable model, that stills achieves high accuracies with good generalization capabilities. However, in the curated set of features shown in table I, there are roughly 40 different features that can be taken into account. To inspect all possible combinations of them, we would need to examine the following amount of combinations:

$$\text{Total} = \sum_{k=1}^{N} \binom{N}{k} = 2^N - 1,$$

which is roughly 10^{12} combinations. To optimize the search, a greedy search algorithm was implemented, by which in each iteration get the set of features that yielded the best results to that point, then add or remove one single feature from it at random, perform a CV with the resulting feature set, and finally add it back to the search tree. To optimize the search, a combination of stable hashing algorithms for frozen sets and heaps were used.

With this algorithm, a grand total of 154 combinations of features were inspected, out of which the one that yielded the best average sentence-based accuracy dropped the features *laplacian_centrality*, *current_flow_closeness* and *degree*, and yielded a sentence-based accuracy of 33.7%. We were surprised to see that it was lower than what we got in the section III, however this is due to the aforementioned low stability of the model.

Finally, all that was left to do was to perform a final training of the model using the entire *training* dataset, and predict the roots of the *test* dataset.

V. Conclusions

This project investigated methods for identifying root nodes in undirected syntactic dependency trees, employing engineered graph-based features and a range of machine learning models. The primary goal was to maximize sentence-level root prediction accuracy. Our experiments yielded a surprising finding: Quadratic Discriminant Analysis, applied with no specific tuning, achieved the best sentence-level accuracy at 41.5%. This performance surpassed that of more complex models, including tuned Random Forests, XGBoost, and neural networks.

APPENDIX A
FIGURES

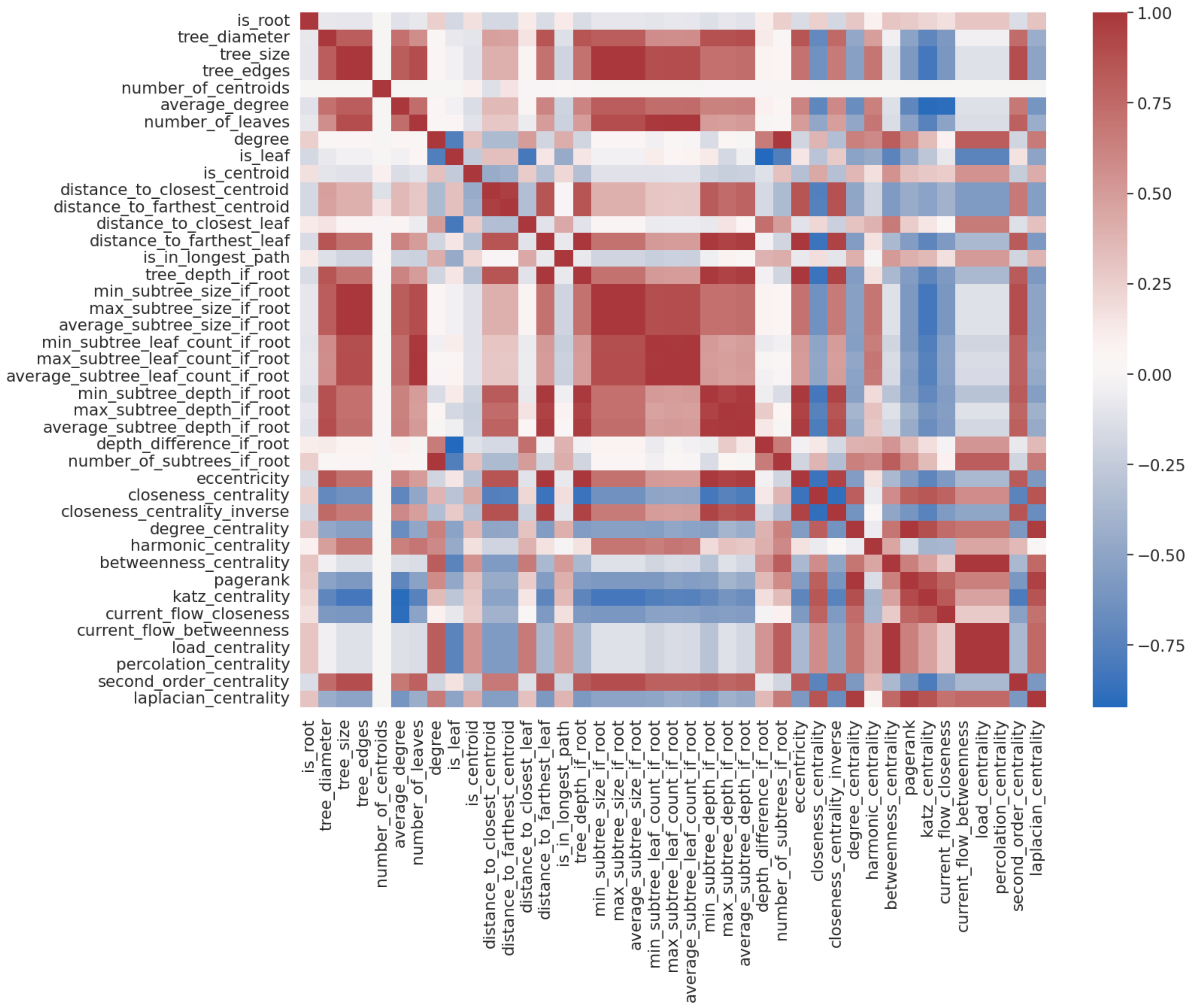

Fig. 6: Correlation between all variables in the unwound dataset.

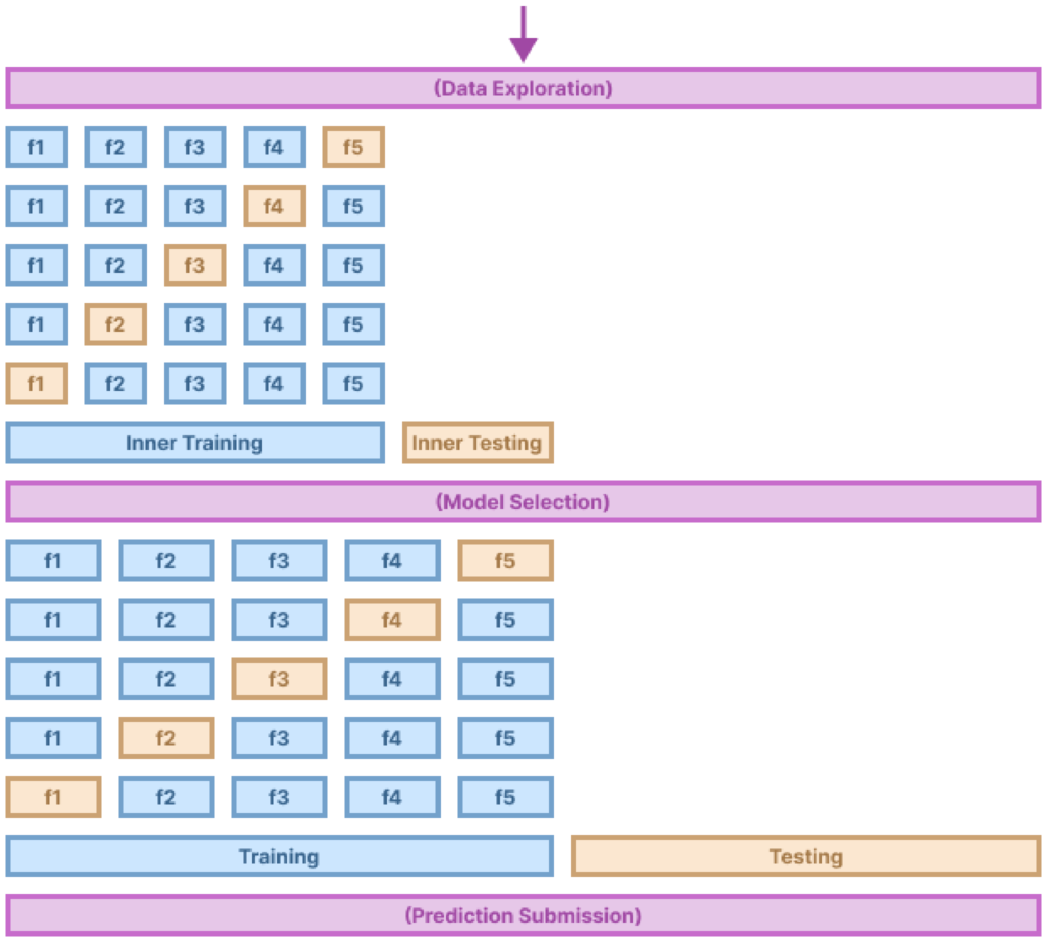

Fig. 7: Strategy followed for model selection.

APPENDIX B
TABLES

TABLE I: Description of the engineered Node Features

Feature Name	**Brief Description**
General Identifiers (Not Features)	
row_index	Original index of the data row this node corresponds to.
node	Identifier of the current node.
is_root	Boolean; True if the current node is the main root of the entire tree.
Global Tree Properties	
language	Language associated with the tree.
tree_diameter	The longest shortest path between any pair of nodes in the tree.
tree_size	Total number of nodes in the tree.
tree_edges	Total number of edges in the tree.
number_of_centroids	Count of centroid nodes in the tree. A centroid is a node whose removal splits the tree into components of size at most n/2.
average_degree	Average degree of all nodes in the tree.
number_of_leaves	Total count of leaf nodes (nodes with degree 1) in the tree.
Local Node Properties	
degree	Number of edges connected to the current node.
is_leaf	Boolean; True if the current node is a leaf node (degree 1).
is_centroid	Boolean; True if the current node is a centroid of the tree.
Path-based Properties for the Node	
distance_to_closest_centroid	Shortest path distance from the current node to the nearest centroid.
distance_to_farthest_centroid	Shortest path distance from the current node to the farthest centroid.
distance_to_closest_leaf	Shortest path distance from the current node to the nearest leaf node.
distance_to_farthest_leaf	Shortest path distance from the current node to the farthest leaf node.
is_in_longest_path	Boolean; True if the current node lies on any diameter path (a longest shortest path) of the tree.
Properties if Tree is Re-rooted at Current Node	
tree_depth_if_root	Depth of the tree if the current node is considered the root (longest path from this node to any descendant).
min_subtree_size_if_root	Minimum size (node count) among subtrees rooted at children of the current node.
max_subtree_size_if_root	Maximum size (node count) among subtrees rooted at children of the current node.
average_subtree_size_if_root	Average size (node count) of subtrees rooted at children of the current node.
min_subtree_leaf_count_if_root	Minimum leaf count in any subtree rooted at a child of the current node.
max_subtree_leaf_count_if_root	Maximum leaf count in any subtree rooted at a child of the current node.
average_subtree_leaf_count_if_root	Average leaf count in subtrees rooted at children of the current node.
min_subtree_depth_if_root	Minimum depth among subtrees rooted at children of the current node.
max_subtree_depth_if_root	Maximum depth among subtrees rooted at children of the current node.
average_subtree_depth_if_root	Average depth of subtrees rooted at children of the current node.
depth_difference_if_root	Difference between the maximum and minimum depths of subtrees rooted at children.
number_of_subtrees_if_root	Number of subtrees formed by children of current node.
Node Centrality Measures	
eccentricity	Maximum shortest path distance from the current node to any other node in the tree.
closeness_centrality	Reciprocal of the sum of shortest path distances from the current node to all other nodes.
closeness_centrality_inverse	Sum of shortest path distances from the current node to all other nodes.
degree_centrality	The fraction of nodes the current node is connected to (degree / (N-1)).
harmonic_centrality	Sum of reciprocals of shortest path distances from the current node to all other nodes.
betweenness_centrality	Fraction of all-pairs shortest paths that pass through the current node.
pagerank	PageRank score of the node, indicating its importance based on link structure.
katz_centrality	Measures influence by counting attenuated paths of all lengths from the node.
current_flow_closeness	Closeness centrality based on an electrical current flow model in the graph.
current_flow_betweenness	Betweenness centrality based on an electrical current flow model.
load_centrality	Fraction of all-pairs shortest paths that pass through the node (often unnormalized version of betweenness).
percolation_centrality	Measures node importance in network percolation processes.
second_order_centrality	Number of paths of length two starting from the node, weighted by inverse of neighbor degrees.
laplacian_centrality	Centrality based on the graph Laplacian, often related to diffusion processes or stability.

TABLE II: Hyper-Parameters of our custom implementation of the MLP.

`hidden_layer_sizes`	A list of integers, for each of which a layer is added with the specified number of nodes.
`first_layer_l1`	L1 kernel regularization to apply to the first layer of the MLP.
`first_layer_l2`	L2 kernel regularization to apply to the first layer of the MLP.
`hidden_layer_l1`	L1 kernel regularization to apply to all layers after the first one.
`hidden_layer_l1`	L2 kernel regularization to apply to all layers after the first one.
`first_layer_dropout`	Percentage of random dropout to apply right after the first layer during training.
`hidden_layer_dropout`	Percentage of random dropout to apply right after each hidden layer during training.
`initial_learning_rate`	Learning rate of the optimizer.
`beta_1`	β_1 hyper parameter of the *Adam* optimizer.
`beta_2`	β_2 hyper parameter of the *Adam* optimizer.
`epsilon`	ε hyper parameter of the *Adam* optimizer.