Optimizing Random Forest with Adaptive Feature Weighting

By: Dev Makwana and Kaavya Borra

Our Problem

- When constructing Decision Trees, the Random Forest algorithm treats all features equally.
- In datasets where attributes have varying importance, it performs suboptimally.

Our Solution

- By scoring each feature based on its overall importance in the dataset, we can allow for the model to build more accurate Decision Trees, increasing the total accuracy of the model with minimal additional computation overhead and training time.
- We also dynamically update these scores as the trees build to allow for more precise feature selection.
- We used TPE (Tree-structured Parzen Estimator) to optimize hyperparameters for the model.

Dataset

- The Nursery Dataset
    - 8 Features: parental situation, nursery evaluation, application form completion, number of children, housing condition, financial standing, social assessment, and health priority
- Train/Test Split: 80/20

Preprocessing

- No preprocessing, because the dataset was from Kaggle

The Baseline Model

- Standard implementation of Random Forest using the scikit-learn library, using entropy loss.

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

Shapely Additive Explanations (SHAP)

Calculates the average shapley value.

N is the number of instances, SHAP(x) is the Shapley value.

$$importance = \frac{1}{N}\sum_{j=1}^{N}\left|SHAP(x_j^{(i)})\right|,$$

- F is the set of all features
- S is the subset of features excluding *i*
- v(S) is the model’s predictions using only features in S

$$SHAP(x) = \sum_{S\subseteq F\setminus\{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!}\left[v(S \cup \{i\}) - v(S)\right],$$

Mutual Information

- Calculates the entropy of the dataset before and after a transformation
- Essentially the same as Info Gain

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

Permutation Importance

- Evaluates the predictive impact of a feature by measuring accuracy difference after shuffling feature values.

$$Permutation\ importance = Baseline\ Accuracy - Accuracy(X^{(i)}_{permuted}),$$

- Baseline Accuracy is the accuracy using the normal data
- Accuracy(X) is the accuracy after randomizing the data of the feature

Pearson Correlation

- Calculates linear relationships between features and the class

$$Pearson\ importance = \left| \frac{Cov(X_i, Y)}{\sigma_{X_i} \sigma_Y} \right|$$

$$Cov(x,\ y) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})(y_i - \overline{y}),$$

Variance

- Used to filter out features with little variance

$$Var(X_i) = \frac{1}{N}\sum_{j=1}^{N}(x_j^{(i)} - \mu_i)^2,$$

Feature Scoring Metrics

SHAP (Shapely Additive Explanations)	Non-linear relationships
Mutual Information	Classification
Permutation Importance	Overall accuracy
Pearson Correlation	Linear relationships
Variance	Filters out features

Dynamic Weight Updates

- We update the weights periodically, recalculating the weights using the formula:

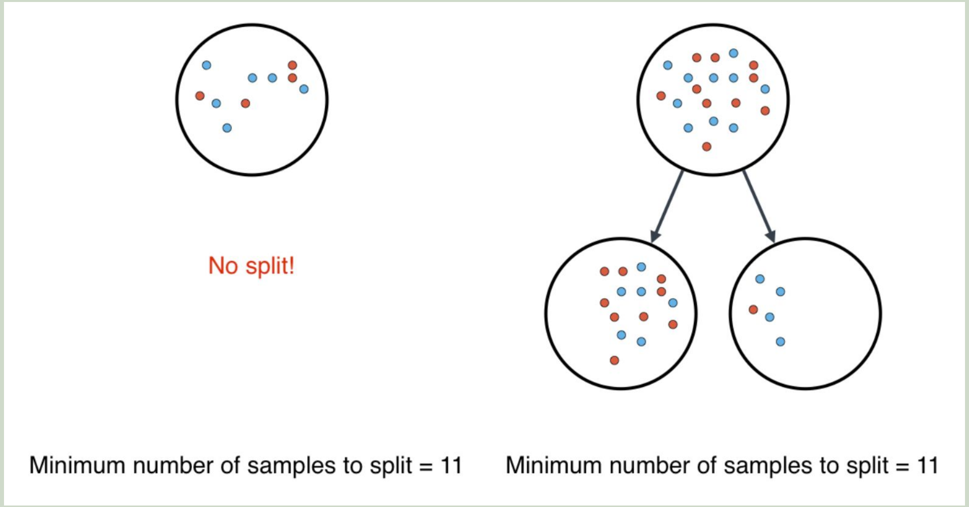
$$new\,weights = \alpha \cdot w + (1 - \alpha) \cdot \phi,$$

- α is the decay factor
- w is the current feature weights
- Φ is the current scores of the features

- This allows the model to incorporate emerging feature interactions while avoiding major weight changes.

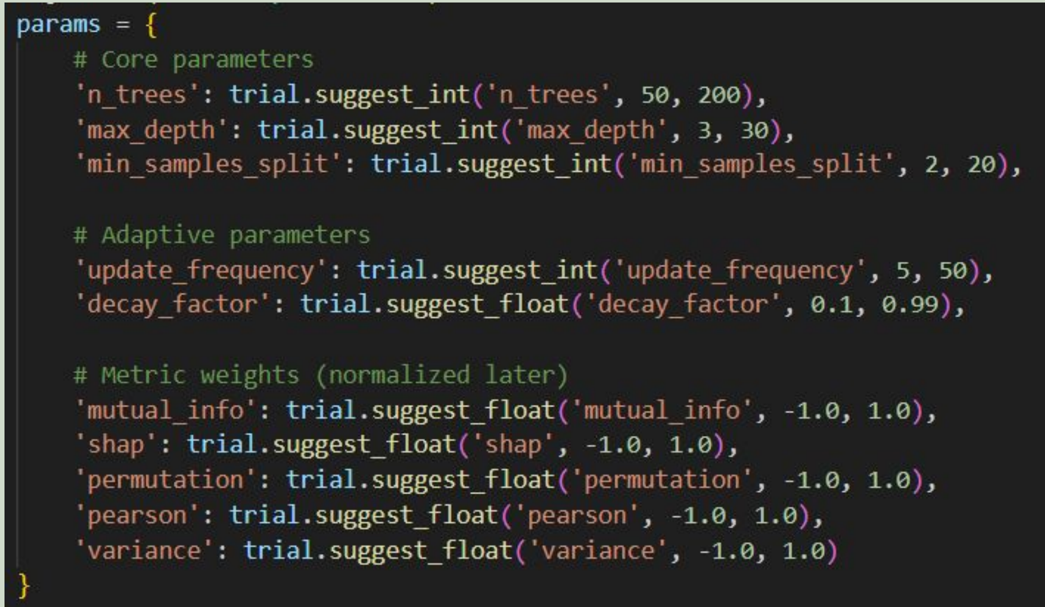
Minimum Sample Split

- Extra parameter to only allowed Decision Trees to split if there is enough instances to split.
- Used to possible end branches early


No split!
Minimum number of samples to split = 11
Minimum number of samples to split = 11

Hyperparameter Optimization

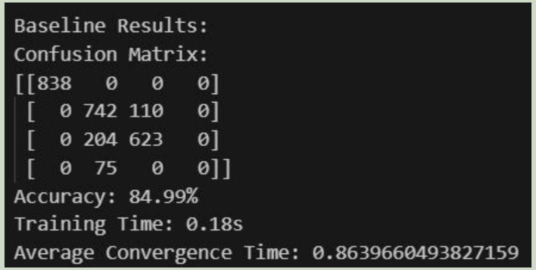
- We used Optuna’s Tree-structured Parzen Estimator (TPE) to optimize 10 parameters.
- TPE models the probability distribution of high-performing parameters.
- We used Hyperband pruning to terminate trials that are unlikely to yield high accuracies to save computation time.



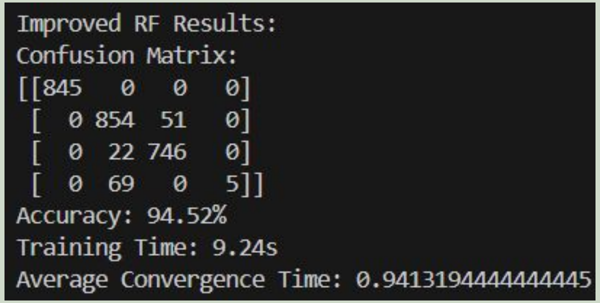
```
params = {
    # Core parameters
    'n_trees': trial.suggest_int('n_trees', 50, 200),
    'max_depth': trial.suggest_int('max_depth', 3, 30),
    'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),

    # Adaptive parameters
    'update_frequency': trial.suggest_int('update_frequency', 5, 50),
    'decay_factor': trial.suggest_float('decay_factor', 0.1, 0.99),

    # Metric weights (normalized later)
    'mutual_info': trial.suggest_float('mutual_info', -1.0, 1.0),
    'shap': trial.suggest_float('shap', -1.0, 1.0),
    'permutation': trial.suggest_float('permutation', -1.0, 1.0),
    'pearson': trial.suggest_float('pearson', -1.0, 1.0),
    'variance': trial.suggest_float('variance', -1.0, 1.0)
}
```

Results

- Accuracy went from ~85% (standard RF) to ~95% (our RF)



```
Baseline Results:
Confusion Matrix:
[[838   0   0   0]
 [  0 742 110   0]
 [  0 204 623   0]
 [  0  75   0   0]]
Accuracy: 84.99%
Training Time: 0.18s
Average Convergence Time: 0.8639660493827159
```



```
Improved RF Results:
Confusion Matrix:
[[845   0   0   0]
 [  0 854  51   0]
 [  0  22 746   0]
 [  0  69   0   5]]
Accuracy: 94.52%
Training Time: 9.24s
Average Convergence Time: 0.9413194444444445
```

+ ~25 minutes optimization time

Number of Trees	84
Max Tree Depth	25
Minimum Sample Split	2
Update Frequency	13
Decay Factor	0.564037135585795
SHAP Weightage	0.3097563418541779
Mutual Info Weightage	0.21521438245654642
Permutation Weightage	0.0709454962848854
Pearson Correlation Weightage	0.1380438030555161
Variance Weightage	0.2660399763488742

Conclusion

- Our improved Random Forest algorithm was more accurate but had a much longer training time.
- GPU-accelerated SHAP (e.g., cuML) could reduce training time, and adjusting weight updates could help with rare classes.
- Testing on more datasets would provide insights for further improvements.

THANK YOU