CSCI 4170 - Project in AI and ML
Homework 2 - Summer 2025

Haruto Suzuki

Due date: 11:59 pm, June. 5th

Task 1 (30 points): Implement a Decision Tree Classifier for your classification problem. You may use a built-in package to implement your classifier. Additionally, do the following:

- Visualize the decision tree structure for at least three different parameter settings. Comment on how the depth and complexity change the tree.

  I used Gini impurity as the splitting criterion and set the random state to 42 for all three visualizations shown below. Although decision trees offer many hyperparameters to tune, I selected a few key ones to for better understanding.

  In Figure 1, the tree was generated with a maximum depth of 3, a minimum samples split of 5000, and a minimum samples per leaf of 10. One clear restriction in action we see is in the right most leaf. Since the number of sample size is less than 5000, it doesn't split even more after a single split. Based on how mixed the value for the leaf nodes suggests that allowing more splits could lead to improved classification performance.

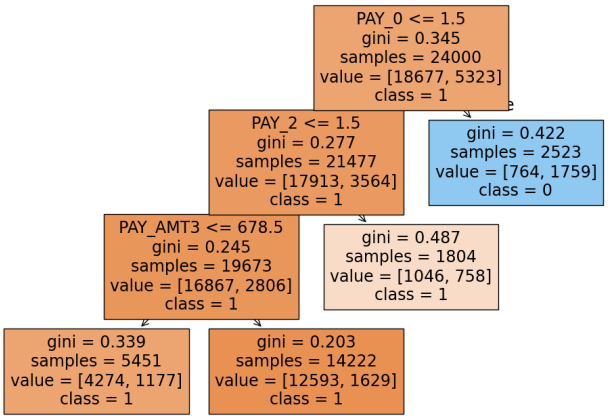

Figure 1: Decision tree visualization for Combination 1 using Gini impurity, maximum depth of 3, minimum 5000 samples required to split, and minimum 10 samples per leaf.

In Figure 2, I used a maximum depth of 4, a minimum samples split of 2000, and a minimum samples per leaf of 500. Compared to Figure 1, the reduced min samples split threshold allowed the rightmost leaf to split further. However, this structural change did not lead to an improvement in accuracy, as the newly formed leaf nodes produced the same classifications as before. If the goal is to improve accuracy, allowing deeper or more frequent splits might be necessary to capture more nuanced patterns in the data.

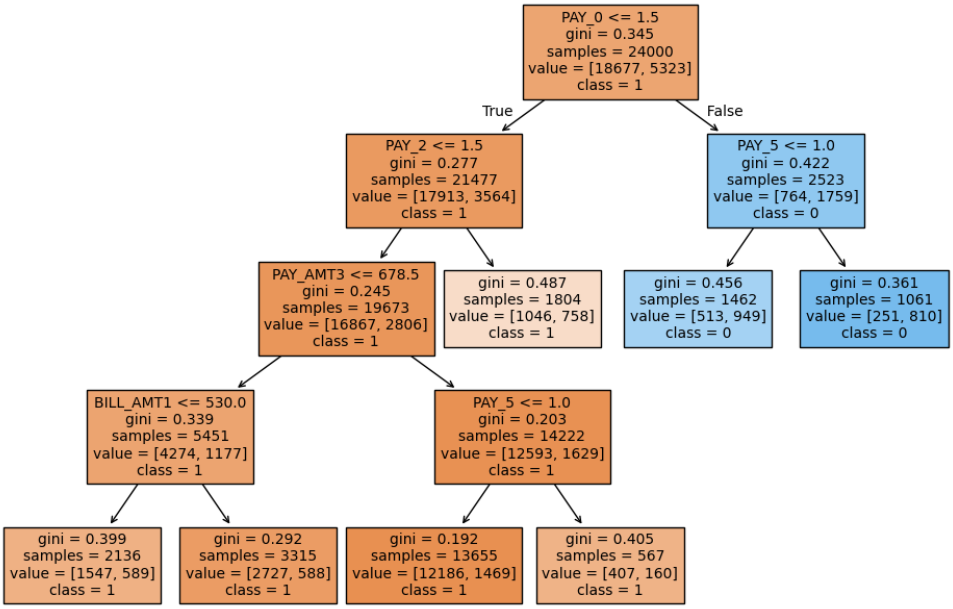

Figure 2: Decision tree visualization for Combination 2 using Gini impurity, maximum depth of 4, minimum 2000 samples required to split, and minimum 500 samples per leaf.

In Figure 3, I used a maximum depth of 5, a minimum samples split of 100, and a minimum samples per leaf of 1000. I decreased min samples split to encourage the model to split more frequently, while simultaneously increasing min samples leaf to guard against overfitting. Striking the right balance between avoiding overfitting and maximizing accuracy is particularly challenging. However, in this case, the parameter adjustments had little effect: the leaf node classifications remained nearly identical to those in Figure 2, indicating that further tuning or a different strategy may be necessary to achieve meaningful changes in model behavior.

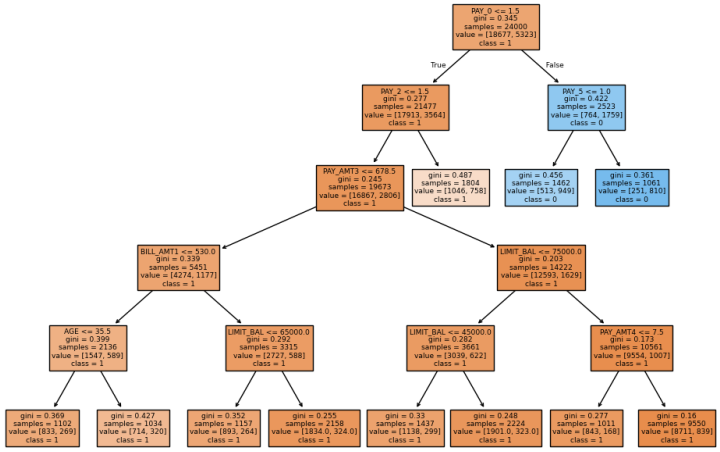

Figure 3: Decision tree visualization for Combination 3 using Gini impurity, maximum depth of 5, minimum 100 samples required to split, and minimum 1000 samples per leaf.

In Figure 4, observe more significant changes, I used a configuration with a maximum depth of 5, a minimum samples split of 2, and a minimum samples per leaf of 1. This allowed the tree to split more aggressively, resulting in leaf nodes from previous configurations being further partitioned into separate classes. These deeper splits occurred because the model had more opportunities to divide the data and capture finer patterns. However, this flexibility comes at a cost: the model begins to overfit, learning noise in the training data rather than generalizable patterns.

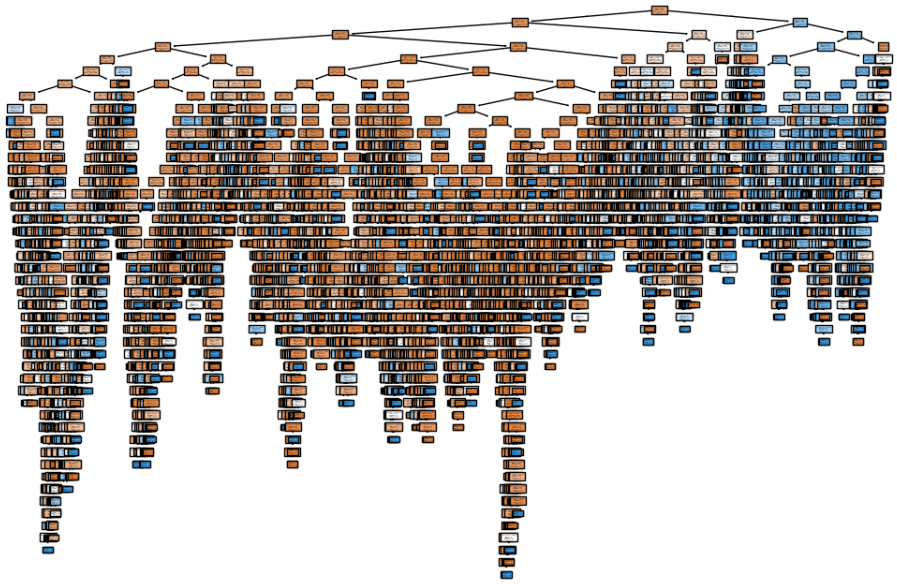

Figure 4: Decision tree visualization for Combination 4 using Gini impurity, maximum depth of 100, minimum 2 samples required to split, and minimum 1 samples per leaf.

Despite variations in parameters across Figures 1, 2, and 3, the accuracy remained consistent at 81.93%, indicating that structural changes alone did not improve model performance as there was not enough splitting with a small difference in hyperparameter. In Figure 4, more aggressive splitting led to deeper partitions, but accuracy dropped to 72.55% due to overfitting. This highlights the challenge of balancing model complexity and generalization—more splits can capture finer patterns, but may harm performance if not properly constrained.

Tree	**Accuracy**
Tree 1	0.8193
Tree 2	0.8193
Tree 3	0.8193
Tree 4	0.7255

Table 1: Accuracy results for different decision tree parameter settings.

- Do some research on what sensitivity analysis is and how it is performed (include citations). Perform a sensitivity analysis to measure the impact of at least two input features on your model's decision boundary.

  Decision trees have many input features that can influence the model's structure and predictions. Although decision trees are generally interpretable, the interaction between features and their hierarchical splits can make it difficult to assess the relative importance of individual inputs. Sensitivity analysis is a useful technique to quantify and understand how variation in input features affects the model's predictions or decision boundaries.

  The purpose of sensitivity analysis is to identify which features have the most significant impact on the model's output. In the context of decision trees, this can be done by varying feature values and observing how the decision boundary or predictions change. Alternatively, one can measure the change in performance metrics when a feature is perturbed, omitted, or fixed. This helps improve interpretability and informs decisions about feature selection and model refinement.

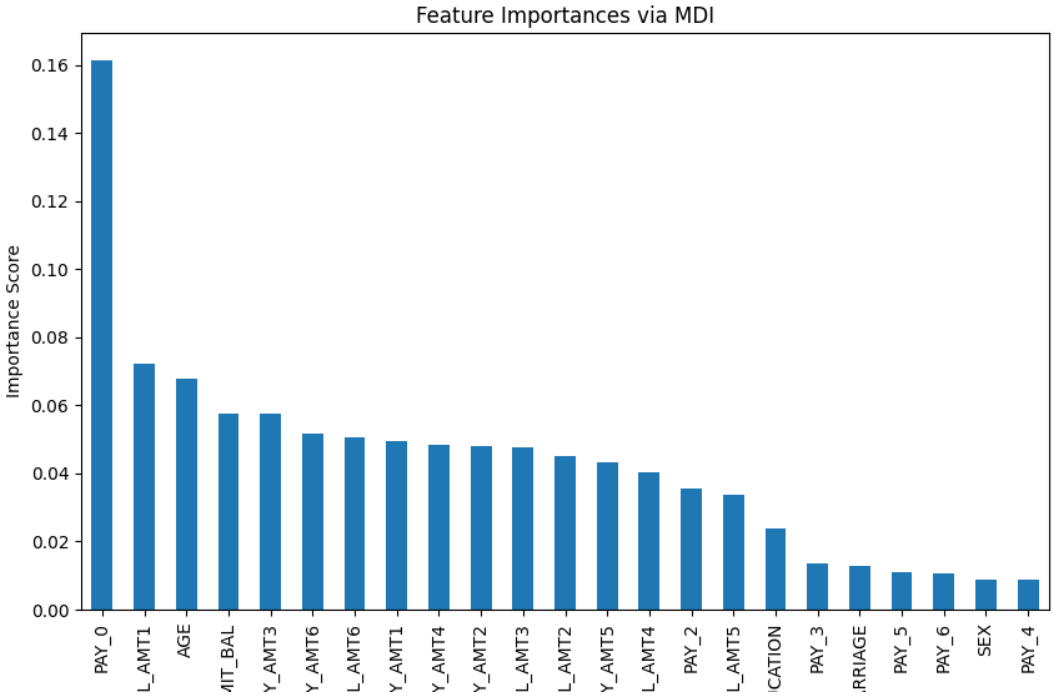

Figure 5: Feature importances computed from a decision tree trained on the full dataset.

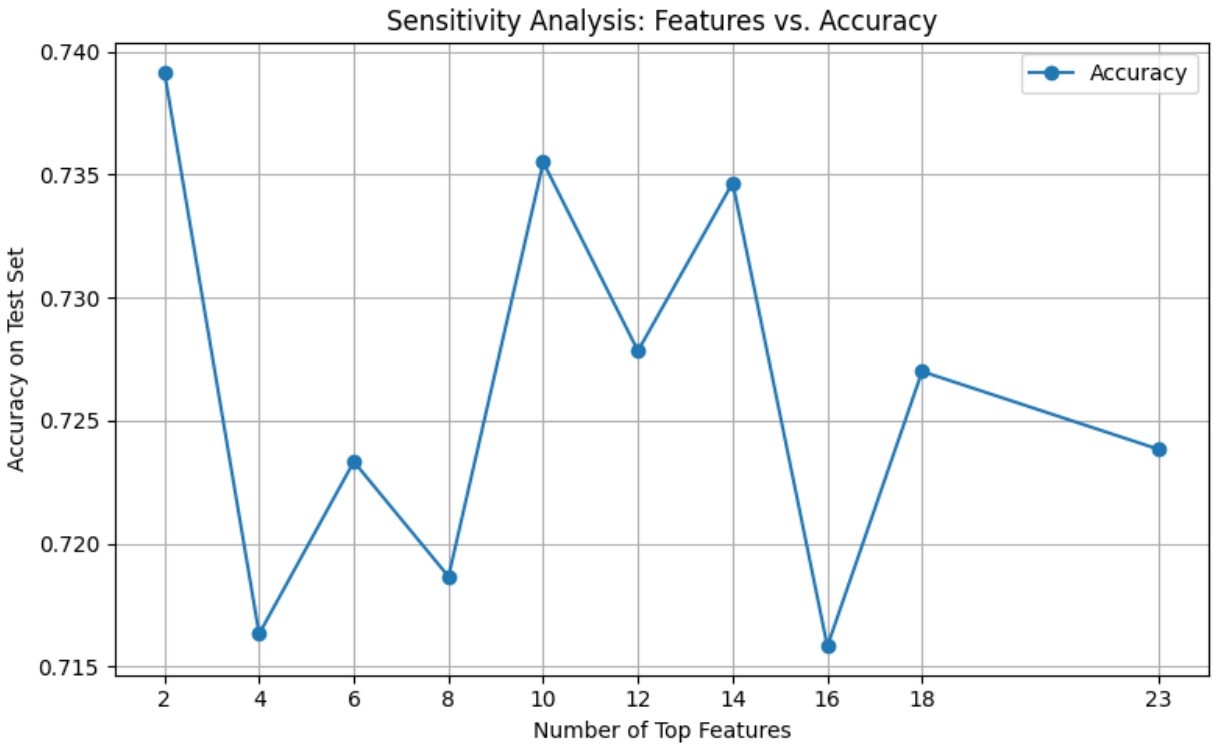

Figure 6: Accuracy of the model on the test set as the number of top features increases.

Since I wasn't entirely sure how to approach the sensitivity analysis at first, I followed an example I found in one of the sources and adapted the method to analyze how the number of input features affected model accuracy.

The sensitivity analysis shows that model accuracy does not consistently improve with more features. Using only the top two features yields the highest accuracy at 0.7392, while larger sets, such as 23 features, result in lower accuracy at 0.7238. This suggests that a small, well-chosen subset can outperform models with more features, likely due to reduced noise and overfitting. Since I wasn't entirely sure how to approach the analysis, I adapted a method I found in a source and applied it to explore how feature count affects performance. I also chose not to set any hyperparameters, as it gives identical accuracy across different feature sizes if I do.

Source:

- HERC: Decision Analysis: Decision trees, Simulation Models, Sensitivity Analyses
- Sensitivity Analysis of Dataset Size vs. Model Performance

Task 2 (30 points): From the Bagging and Boosting ensemble methods pick any one algorithm from each category. Implement both the algorithms using the same data.

I chose Random Forest as Bagging and XGBoost as a Boosting ensemble method.

- Use stratified k-fold cross-validation with at least three different folds (e.g., 5, 10, 15). You may do your own research on this technique (include citations).

  Stratified K-Fold Cross Validation is a variation of K-Fold used in classification tasks to ensure that each fold maintains the same class distribution as the full dataset. This is especially important for imbalanced datasets, as it prevents biased splits and provides more reliable performance metrics compared to random or standard K-Fold splitting.

  Source:

  – GeeksForGeeks: Stratified K Fold Cross Validation

- Evaluate the models using any three-evaluation metrics of your choice (e.g. accuracy, Precision, F1-score etc.).

Folds	**Model**	**Accuracy**	**Precision**	**Recall**	**F1-score**
5	Random Forest	0.8149	0.6893	0.2975	0.4155
5	XGBoost	0.8193	0.6672	0.3660	0.4727
10	Random Forest	0.8154	0.6890	0.3012	0.4188
10	XGBoost	0.8205	0.6734	0.3666	0.4745
20	Random Forest	0.8148	0.6861	0.2994	0.4164
20	XGBoost	0.8204	0.6726	0.3669	0.4745
30	Random Forest	0.8145	0.6868	0.2966	0.4135
30	XGBoost	0.8192	0.6690	0.3627	0.4698

Table 2: Stratified K-Fold Cross-Validation Results Comparing Random Forest and XGBoost

- Comment on the behavior of each algorithm under the metrics. Does the performance ranking change based on the metric used? Why?

  Across all folds, XGBoost consistently outperforms Random Forest in recall and F1-score, despite slightly lower precision. This is particularly important given the imbalanced target distribution (approximately 22% positive class), where high recall is crucial to correctly identify minority class instances. While accuracy differences between the models are minor (typically under 1%), XGBoost provides notably better recall (36%) than Random Forest (30%), which translates to higher F1-scores, reflecting a better balance between precision and recall. This suggests that XGBoost is more effective at capturing the minority class despite the risk of reduced precision.

  The rankings shift slightly depending on the evaluation metric used. If one were to rely solely on accuracy, the performance gap would appear negligible. However, metrics like

recall and F1-score offer a more meaningful view in imbalanced classification settings, where detecting the minority class matters more than overall correctness. Random Forest tends to favor the majority class due to its tendency to reduce variance, whereas XGBoost adapts through sequential learning and performs better on hard-to-classify minority instances. This highlights the importance of choosing evaluation metrics aligned with the problem context—especially in real-world applications where false negatives carry higher costs.

Task 3 (40 points): Compare the effectiveness of the three models implemented above. Analyze the results using the following:

- A confusion matrix for one selected test fold.

In Figure 7 is the confusion matrix of the three models implemented showcasing True Positive, True Negative, False Positive, and False Negative. These matrices allow us to see exactly how each model performed in distinguishing between the classes. Although the visual difference may appear not be clear, we can observe that XGBoost achieved the highest number of true positives, suggesting it was more effective at identifying the minority class and reducing variance. On the other hand, Random Forest did the best in identifying True Negative than the other models, indicating it was more conservative and likely emphasized bias reduction. This trade-off reflects the different learning strategies of boosting and bagging: XGBoost sequentially refines its predictions, while Random Forest builds robustness through aggregation.

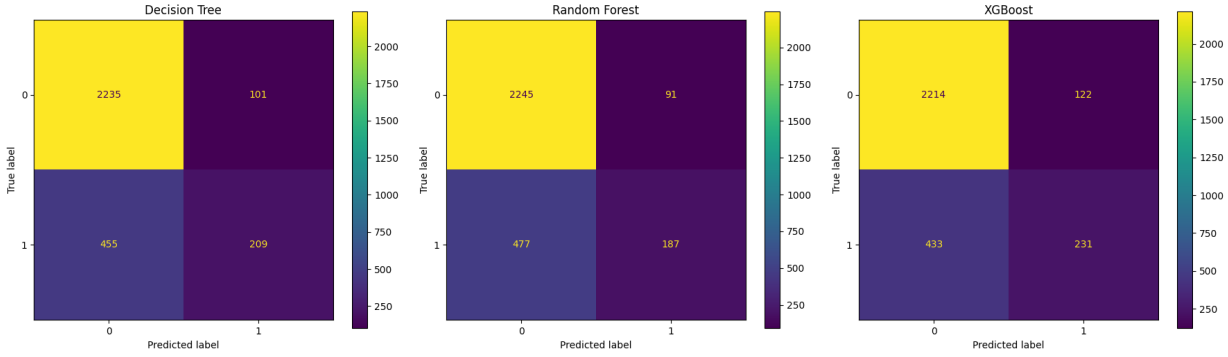

Figure 7: Confusion Matrix of Decision Model, Random Forest, and XGBoost for a selected test fold

Below is Table 3, which lists the exact metrics corresponding to the confusion matrix shown above. The results support our earlier analysis: XGBoost achieved the highest recall (0.3479) and F1-score, confirming its effectiveness in capturing more true positives and reducing variance. Random Forest showed the highest precision (0.6727), indicating it was more conservative and focused on reducing false positives, consistent with bias reduction. While the Decision Tree had similar accuracy, it underperformed in both recall and F1-score due to its limited capacity. These findings align with the

observed behavior in the confusion matrix and highlight the distinct strengths of each model.

Model	Accuracy	Precision	Recall	F1-score
Decision Tree	0.8147	0.6742	0.3148	0.4292
Random Forest	0.8107	**0.6727**	0.2816	0.3970
XGBoost	**0.8150**	0.6544	**0.3479**	**0.4543**

Table 3: Evaluation metrics for Decision Tree, Random Forest, and XGBoost on a selected test fold.

- A statistical test (e.g., paired t-test) to determine if differences between models are significant.

  A paired t-test was conducted to evaluate whether the differences in model accuracy across cross-validation folds were statistically significant. As shown in Table 4, a positive t-value indicates that the first model in the comparison performed better on average than the second, while a negative t-value suggests the opposite.

  The test results reveal that Random Forest significantly outperforms Decision Tree, highlighting the performance boost gained from ensemble methods. On the other hand, Decision Tree and XGBoost showed no statistically significant difference in accuracy. Finally, XGBoost significantly outperforms Random Forest, as indicated by a negative t-statistic and a very low p-value. These results suggest that boosting provides more accurate predictions than bagging in this context, especially when handling complex patterns or imbalanced class distributions.

Model Comparison	t-statistic	p-value
Decision Tree vs Random Forest	7.7744	0.0000
Decision Tree vs XGBoost	-0.9728	0.3561
Random Forest vs XGBoost	-5.4058	0.0004

Table 4: Paired t-test results comparing model performance (accuracy) across 10 folds.

- A discussion on the trade-off between bias and variance for each model.

  Decision Trees typically have low bias but high variance, meaning they can capture complex patterns but are prone to overfitting, especially with deep trees or small `min_samples_split`. Random Forests, which average multiple Decision Trees, reduce variance by aggregating diverse models trained on different subsets of the data, while maintaining relatively low bias. XGBoost also builds on Decision Trees but does so sequentially, allowing it to correct previous errors and reduce both bias and variance, though it may risk overfitting if not properly regularized. The Decision Tree model in this study, constrained with shallow depth and large split thresholds, exhibited underfitting due to high bias. In contrast, both ensemble methods achieved better generalization by effectively managing the bias-variance trade-off.