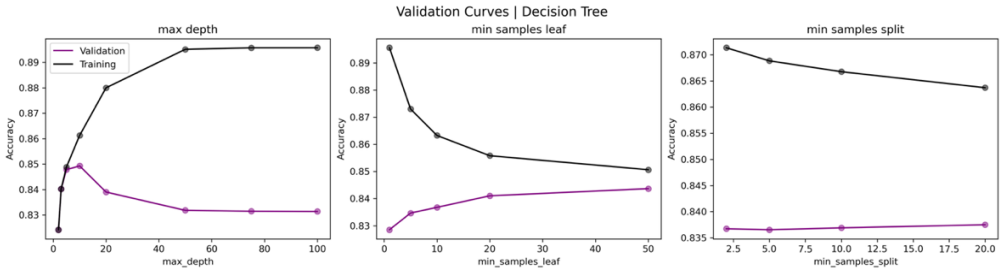
Note: I prioritized accuracy in the evaluation, therefore in training I did not assign weights to the minority dataset (earning over 50K) – except for Decision Tree notebook 02 I showed both routes of balancing and leaving unbalanced without weighing the smaller sized class. In the case of mislabeling someone with higher income as earning less income, I judged this not harmful and therefore valued overall accuracy.

Section 00: Data Preprocessing

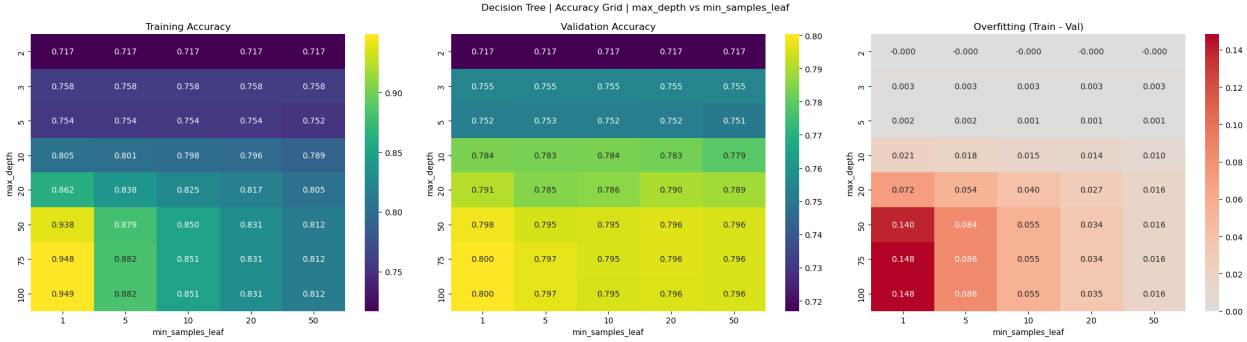
The dataset, pre-split into training and testing sets, was sourced from the UCI Machine Learning Repository. As an initial step, I loaded both sets and performed an exploratory check for class imbalance in the target variable. I also identified missing values represented by "?" and quantified the proportion of missing data both across features and individual observations. Given that the extent of missingness was minimal, I opted to drop the affected rows rather than impute, preserving the overall data integrity. To prepare the data for machine learning models, all categorical features were transformed using one-hot encoding. This ensured that the input data was in a fully numeric and model-compatible format, with consistent preprocessing applied across both training and test sets.

Section 01: Overfitting

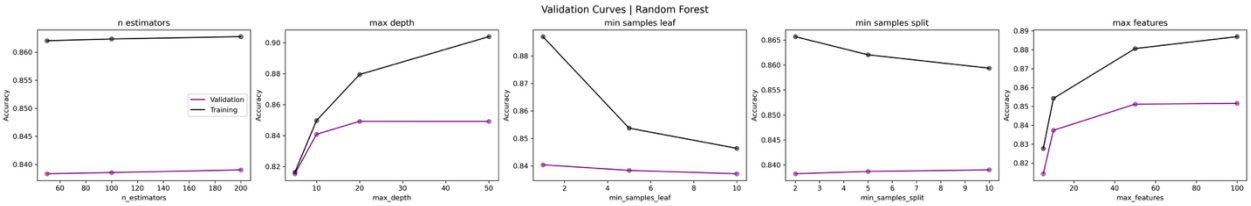
■ **Decision Tree**: I was able to overfit (and underfit) using a Decision Tree – evidenced by divergence in accuracy between training and validation set depending on some key hyperparameters. The hyperparameters: $[max\_depth, min\_sample\_leaf]$ were significant in determining if the model overfit to the training set, for example we can see by the visualization below that for $max\_depth > 10$ the training accuracy continues to grow while the validation accuracy levels off – this is a clear sign of overfitting! Likewise, when $min\_sample\_leaf < 20$ – the nodes become too pure and overfit to the training set.



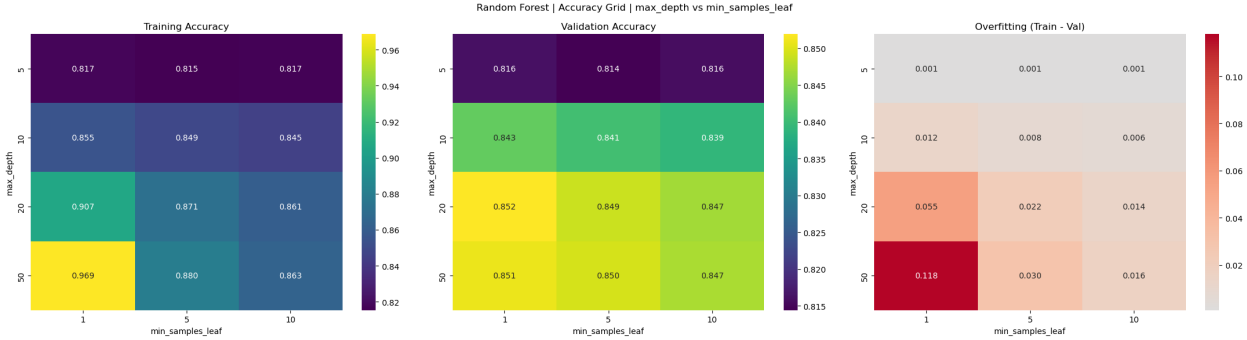
The plot below helps to verify this by taking both training and validation accuracy across a grid of hyperparameters and visualizing the difference (right) between the two. We can clearly see that as max_depth increases, and min_sample_leaf decreases we see a larger difference between the training and validation (and it is useful to note as max_depth decreases we see training = validation which supports the idea that we are underfitting and the model lacks complexity).



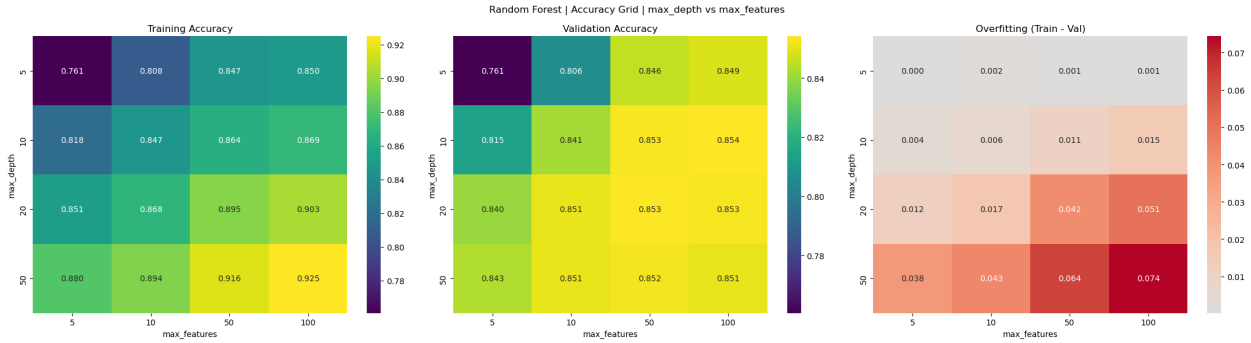
■ **Random Forest**: I observed overfitting in the Random Forest model, indicated by a widening gap between training and validation accuracy as certain hyperparameters were adjusted. Specifically, when $max\_depth > 10$, $min\_samples\_leaf < 5$, and $max\_features > 10$, the training accuracy increased while validation accuracy plateaued — a classic signal of overfitting. These hyperparameters control the tree complexity and generalization ability. Since a Random Forest is an ensemble of decision trees trained via bootstrapping and feature subsampling (bagging), its generalization benefits rely on the Law of Large Numbers (LLN) and the assumption of i.i.d. base learners. However, when individual trees overfit, this assumption weakens, and the ensemble can still overfit if not properly regularized.



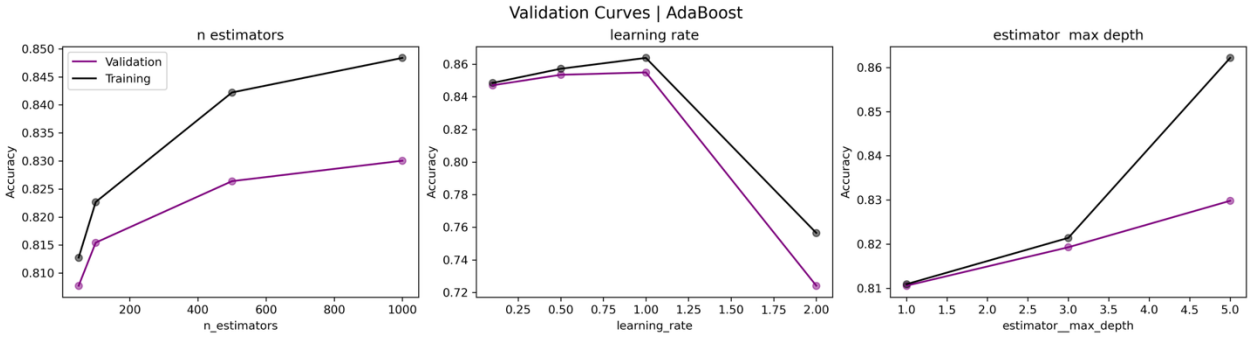
We can clearly see in the two plots below that as max_depth increases, and min_sample_leaf decreases we see a larger difference between the training and validation!



Furthermore, we can see that as $max\_features$ increases and max_depth increases we see the training and validation accuracy diverge supporting the claim the model is overfitting.

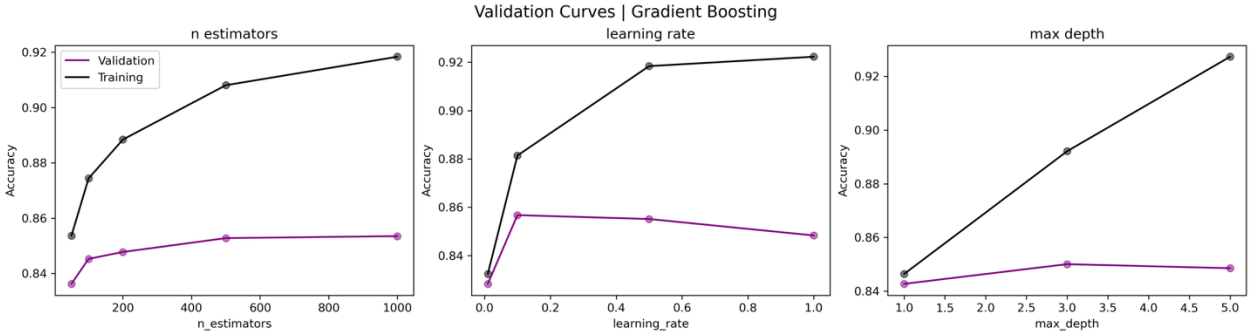


■ **AdaBoost**: With AdaBoost it was difficult to overfit the default model using decision trees with $max\_depth = 1$ as the base learner. Therefore, I also adjusted the hyperparameter max_depth of the base learner – Decision Tree. It is worth noting that the default model for AdaBoost was rather tricky to overfit! Below we can see that after adjusting the base learner as $n\_estimators > 50$, $learning\_rate > 1.00$ and $max_{depth} > 3$ we see overfitting, especially as we increase the max_depth of the base learner, the model learns far too fast and overfits early on in the training process defeating the purpose of gradually learning by fitting the residuals.

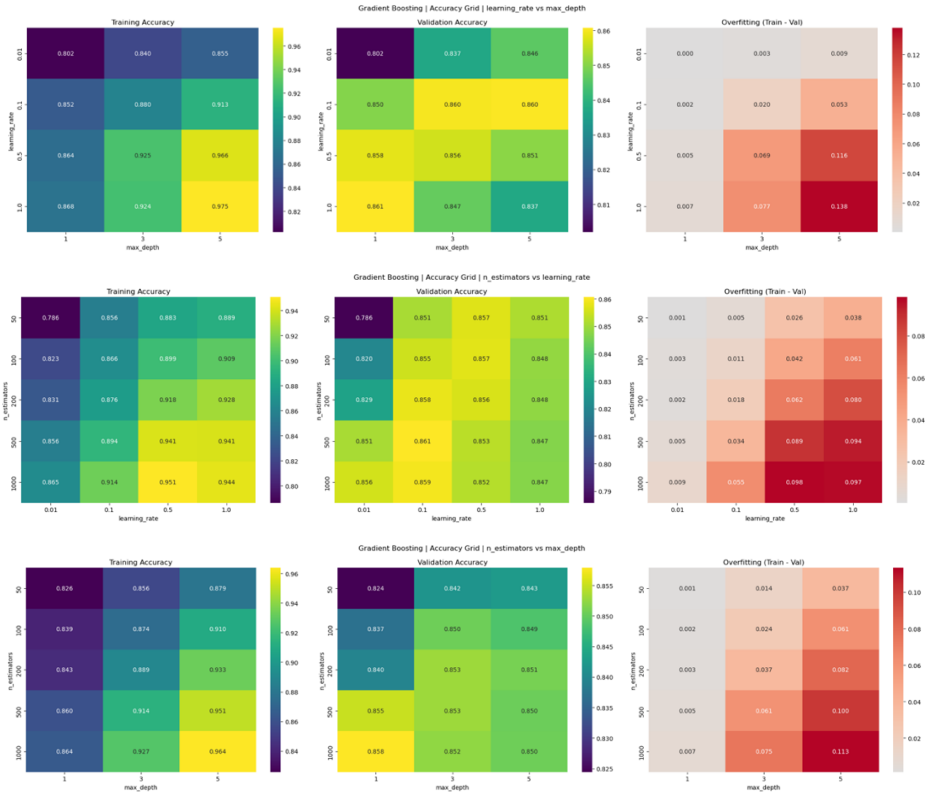


It is clear that the max_depth contrbibutes to overfitting but if we see the heatmap below it also is evident that as the $learning\_rate$ increases and $n\_estimators$ increase overfitting occurs!

■ **Gradient Boosting**: With sklearns Gradient Boosting it was much easier to overfit the default model. Below we can see that as $n\_estimators > 50$, $learning\_rate > 0.05$ and $max_{depth} > 1$ we see overfitting, as the accuracy and validation accuracy significantly diverge – training curve continues climbing and the validation curve levels off or even declines.



This is further emphasized by the heatmaps below where the darker the red becomes the larger the gap between the training and validation curves become providing a great visualization of the curves across two parameters!



Section 02: Results

To tune each model, I used GridSearchCV over a targeted hyperparameter grid (a very large grid was not viable due to computational limitations therefore, determining the grid was informed by model intuition and exploratory experiments). For efficiency, I reduced k_fold to 2 and selected ranges that spanned underfitting to overfitting regions, allowing me to capture both model capacity and generalization behavior. Furthermore, I used these same CV results to conduct the analysis of overfitting and all the visualizations above. All the models were run with $n\_jobs = -1$ to utilize all CPU cores, though this benefits methods like Random Forest more directly due to the independence of its base estimators. Boosting methods, being inherently sequential, see more limited speedup from parallelization. This is the standard approach to fitting models (unless runtime becomes a significant issue and GPU is required), alternative methods could be Randomized Search Cross-Validation. Runtime did begin to become an issue which is why k_fold was reduced from default = 5 to 2 to drastically reduce the number of models fit which makes a significant difference. Below is the accuracy report across methods including Decision Tree with both balanced and unbalanced weights (if unspecified assume weights not applied). The method that **performed best** on the test set was **Gradient Boosting** which aligns with what we expect as it takes longer to fit, fitting to the residuals which in theory will lead it to eventually outperform other methods such as Random Forests by better generalizing to the test set! Also note, when balancing the weights to the Decision Tree it is expected that the accuracy will suffer but other metrics such as recall will perform better but for the purposes of this comparison, we are using scoring=accuracy (which is unfair in that regard). I have provided the tuned hyperparameters for the top two methods which seek a smaller learning rate, sufficiently sized iterations and fitting base learner on max_depth>1!

Accuracy Report

Decision Tree (balanced)	Decision Tree	Random Forest	AdaBoost	Gradient Boosting
DecisionTreeClassifier(class_weight='balanced') Grid - Hyperparameters: ['max_depth', 'min_samples_leaf', 'min_samples_split'] **Train Accuracy: 99.811%** **CV Validation Accuracy: 80.741%** **Test Accuracy: 80.857%**	Grid - Hyperparameters: ['max_depth', 'min_samples_leaf', 'min_samples_split'] CV-Tuned Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 20, 'min_samples_split': 2} **Train Accuracy: 85.893%** **CV Validation Accuracy: 85.021%** **Test Accuracy: 85.126%**	Grid - Hyperparameters: ['n_estimators', 'max_depth', 'min_samples_leaf', 'min_samples_split', 'max_features'] CV-Tuned Hyperparameters: {'max_depth': 20, 'max_features': 50, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200} **Train Accuracy: 86.718%** **CV Validation Accuracy: 85.561%** **Test Accuracy: 85.691%**	Grid - Hyperparameters: ['n_estimators', 'learning_rate', 'estimator__max_depth'] CV-Tuned Hyperparameters: {'estimator__max_depth': 5, 'learning_rate': 0.5, 'n_estimators': 1000} AdaBoostClassifier(algorithm='SAMME', estimator=DecisionTreeClassifier()) **Train Accuracy: 87.202%** **CV Validation Accuracy: 86.413%** **Test Accuracy: 86.786%**	GradientBoostingClassifier() Grid - Hyperparameters: ['n_estimators', 'learning_rate', 'max_depth'] CV-Tuned Hyperparameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500} **Train Accuracy: 88.190%** **CV Validation Accuracy: 86.486%** **Test Accuracy: 86.932%**

The hyperparameters that performed "best" on the test set were achieved by exhaustively searching all possible combinations across the Grid training each combination on a training split and evaluating that trained model on a validation split. By using this technique we find the combination of hyperparameters that generalized best to the validation split which we assume is the most likely to also generalize best to the test split (assuming there are no distribution shifts between the training-validation and test sets).

Runtime Report

	# Grid Combinations	CV - k_fold	Runtime (Seconds)	Runtime (Minutes)
Decision Tree	160	5	27.4	0.46
Random Forest	432	2	269.45	4.49
AdaBoost	48	2	487.49	8.12
Gradient Boosting	60	2	420.62	7.01

Note in Runtime Report figure above: I included # Grid Combinations and CV-k_fold to not make the runtime purely subjective – because in theory we could manufacture Decision Trees to take longer than Random Forest by creating a very bad/small grid. However, boosting methods use a base learner, i.e., we can use Decision Trees as the base learner and continuoslty fir on the previous base learner's residual and iterate this – this accounts for 1 Boosting model fit - so clearly there is a significant jump in runtime. Additionally, we might expect Gradient Boosting to take longer than AdaBoost due to its gradient-based optimization, but in this case, AdaBoost had a longer runtime. This is because the base learners in AdaBoost were tuned during cross-validation, leading to deeper decision trees that required more computation. Although fewer models were fit, each model was more complex and contributed more significantly to runtime than the shallow stumps typically used in boosting.

Section 03: Interpretations

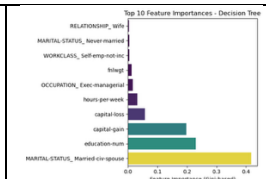
Clearly, applying weights when fitting the decision tree changed the results significantly. We observe that after applying class weights, interpretability decreases and accuracy drops, which aligns with prior observations. This supports the decision to avoid applying weights in the other ensemble methods. In the unweighted tree, the most significant features are ones we would expect, whether someone is married, their education level, hours worked, and capital gains. These features are intuitive and economically meaningful. Interestingly, several noisy or redundant one-hot encoded features—such as age-related indicators—were filtered out, which again makes sense given that broader

categorical variables like "marital-status" or "relationship" may already capture their effects. In this context, multicollinearity between features exists, but decision trees handle it well by selecting the most informative feature among correlated ones and ignoring the rest.

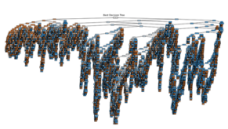
Across all model families—Random Forest, AdaBoost, and Gradient Boosting—we see a **pattern in which features are deemed important**. "marital-status_Married-civ-spouse" consistently appears as one of the top predictors, along with "education-num", "capital-gain", and "hours-per-week". These features are repeatedly selected by both bagging and boosting methods, indicating that they hold substantial predictive power regardless of the ensemble strategy. While some variation exists—such as AdaBoost elevating specific age indicators or relationship categories—there is overall agreement on which variables matter most. This agreement increases confidence in the models and suggests that despite algorithmic differences, they are learning similar underlying patterns in the data.

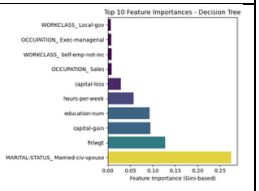
Decision Tree

No Weights Applied

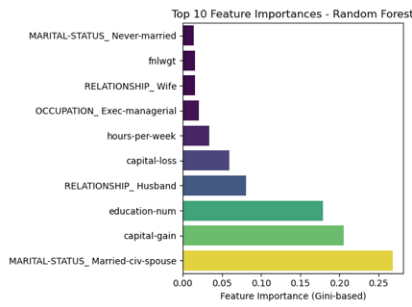


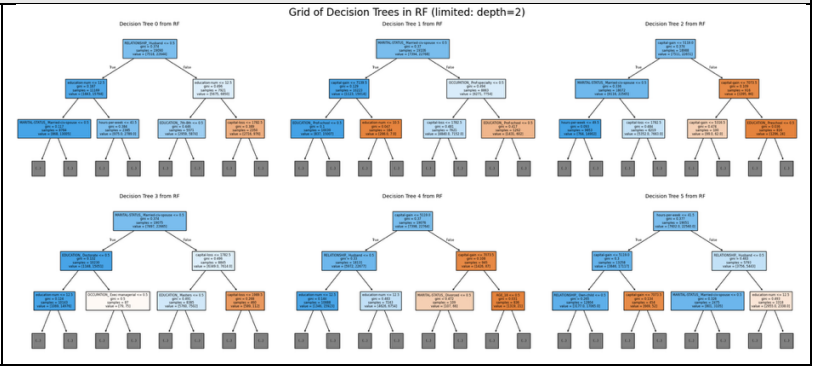
Weights Applied



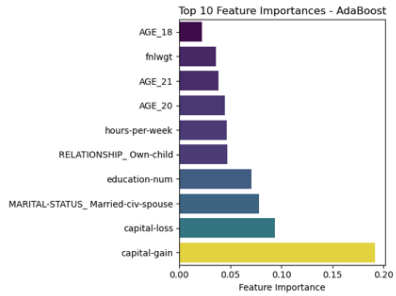


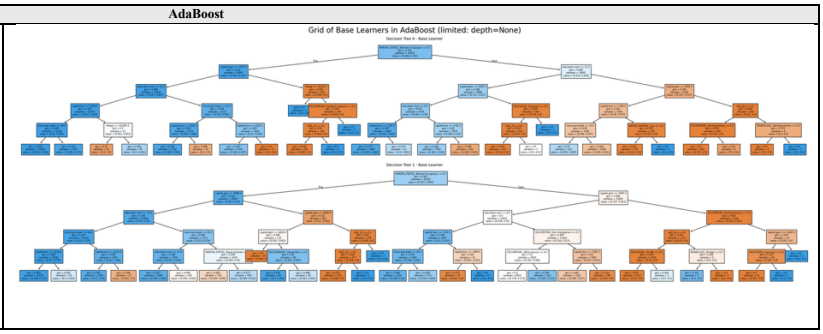
Random Forest



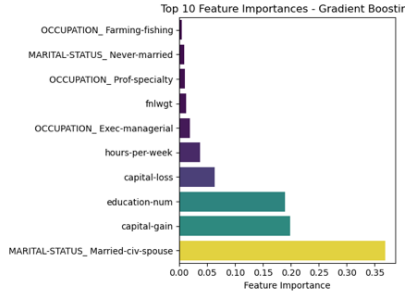


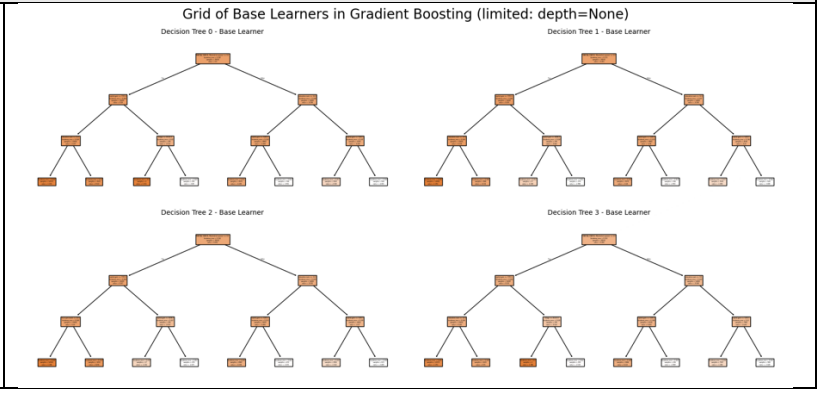
AdaBoost





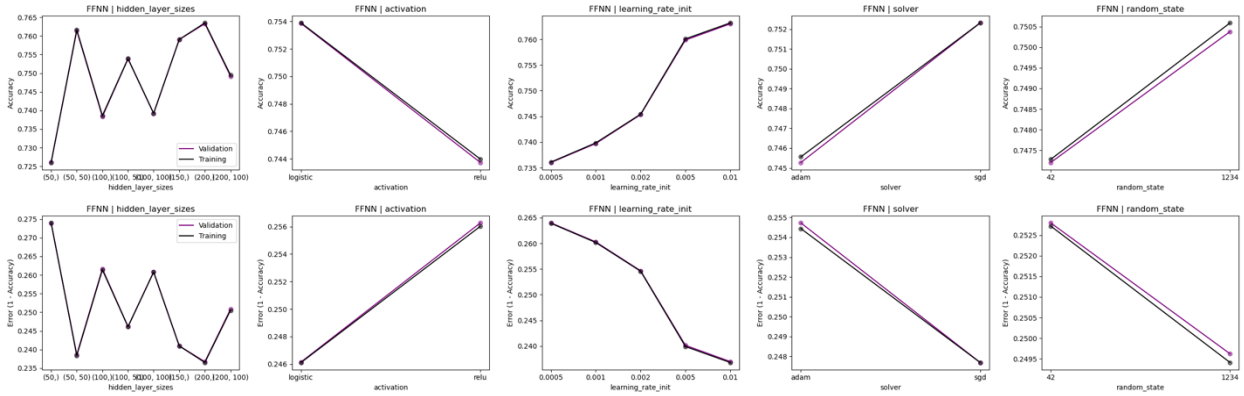
Gradient Boosting





Section 04: MLP

Accuracy/Loss across hyperparameters in Grid (holding others fixed).



The hyperparameters that performed better on the grids search are generally deeper layers with more units and a higher learning rate closer to 0.01. The activation, solver and random_state (initialization) did not have a very significant impact on the grid search – likely due to both being good initializations or potentially both being bad… The 'best', tuned hyperparameters are given by {'activation': 'relu', 'hidden_layer_sizes': (150,), 'learning_rate_init': 0.005, 'random_state': 42, 'solver': 'adam'}. After running the grid search on the following grid, we get the results (in blue):

```
param_grid = {
    'hidden_layer_sizes': [
        (50,), (100,), (150,),
        (50, 50), (100, 50), (100, 100),
        (200,), (200, 100)
    ],
    'activation': ['relu', 'logistic'],
    'solver': ['adam', 'sgd'],
    'learning_rate_init': [0.0005, 0.001, 0.002, 0.005, 0.01],
    'random_state': [42, 1234]
}
```

Accuracy Report

FF Neural Network	Decision Tree	Random Forest	AdaBoost	Gradient Boosting
Grid - Hyperparameters: ['activation', 'hidden_layer_sizes', 'learning_rate_init', 'random_state', 'solver'] Best Hyperparameters: {'activation': 'relu', 'hidden_layer_sizes': (150,), 'learning_rate_init': 0.005, 'random_state': 42, 'solver': 'adam'} **Train Accuracy: 80.495%** **CV Validation Accuracy: 84.384%** **Test Accuracy: 80.538%**	Grid - Hyperparameters: ['max_depth', 'min_samples_leaf', 'min_samples_split'] CV-Tuned Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 20, 'min_samples_split': 2} **Train Accuracy: 85.893%** **CV Validation Accuracy: 85.021%** **Test Accuracy: 85.126%**	Grid - Hyperparameters: ['n_estimators', 'max_depth', 'min_samples_leaf', 'min_samples_split', 'max_features'] CV-Tuned Hyperparameters: {'max_depth': 20, 'max_features': 50, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200} **Train Accuracy: 86.718%** **CV Validation Accuracy: 85.561%** **Test Accuracy: 85.691%**	Grid - Hyperparameters: ['n_estimators', 'learning_rate', 'estimator__max_depth'] CV-Tuned Hyperparameters: {'estimator__max_depth': 5, 'learning_rate': 0.5, 'n_estimators': 1000} AdaBoostClassifier(algorithm='SAMME', estimator=DecisionTreeClassifier()) **Train Accuracy: 87.202%** **CV Validation Accuracy: 86.413%** **Test Accuracy: 86.786%**	GradientBoostingClassifier() Grid - Hyperparameters: ['n_estimators', 'learning_rate', 'max_depth'] CV-Tuned Hyperparameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 500} **Train Accuracy: 88.190%** **CV Validation Accuracy: 86.486%** **Test Accuracy: 86.932%**

The FFNN test accuracy is significantly worse compared to the tree/boosting methods which is expected as tabular data structures are not where this model will thrive/perform well.

Runtime Report

	# Grid Combinations	CV - k_fold	Runtime (Seconds)	Runtime (Minutes)
FF Neural Network	320	2	333.45	5.56
Decision Tree	160	5	27.4	0.46
Random Forest	432	2	269.45	4.49
AdaBoost	48	2	487.49	8.12
Gradient Boosting	60	2	420.62	7.01

Note: I included # Grid Combinations and CV-k_fold to not make the runtime purely subjective – because in theory we could manufacture Decision Trees to take longer than Random Forest by creating a very bad/small grid. However, boosting methods use a base learner, i.e., we can use Decision Trees as the base learner and continuoslty fir on the previous base learner's residual and iterate this – this accounts for 1 Boosting model fit - so clearly there is a significant jump in runtime.

To tune the multi-layer perceptron (MLP), I employed GridSearchCV to systematically explore combinations of hyperparameters which if computationally feasible is standard procedure to attain a well tuned model! This process was iterated and refined multiple times to identify a viable configuration. The best-performing hyperparameters included: activation='relu', hidden_layer_sizes=(150,), learning_rate_init=0.005, solver='adam', and random_state=42. As anticipated, the MLP underperformed relative to tree-based methods—an outcome that aligns with known limitations of neural networks on tabular data. Among the ensemble models, both AdaBoost and Gradient Boosting yielded strong performance, with Gradient Boosting achieving slightly better test accuracy and runtime. While decision trees alone were less accurate, they were notably efficient and still performed competitively given their simplicity. With greater computational resources, further tuning of the MLP's architecture—particularly increasing the depth and number of units in the hidden layers—could enhance its capacity and performance. In fact, as the number of layers increases, the model can begin to exhibit **benign overfitting**, a phenomenon where highly overparameterized neural networks still generalize well to unseen data—an effect that is often desirable and unique to deep learning methods.