Dan Yawitz

CUNY SPS / Data 622 ML

12 October 2025

Context
This assignment asks that we run six “experiments”: 3 algorithms x 2 experiments each. What we call an “experiment” is left up to us.

I have been working ML models for the last decade, but have always left the work of fine-tuning the training to other scientists on my team. So I'm challenging myself to tune the models parameters as best I can, and focus on what I learn if I tune something incorrectly.

Assumptions
Rather than attempt to test every part of the model building process (feature selection, feature scaling, model selection, feature engineering, etc), this assignment focuses on the following processes:
- Model selection
- Hyperparameter tuning
- Feature selection

For each experiment we run the following:
- Select a model
- Select a hyperparameter to tune (and range of values)
- apply `StratifiedKFolds` cross validation with 5 folds.

The goal is to select the best model, with the best-tuned parameters.

What is not included? There are hundreds of moving parts in an ML model. In order to minimize experimental variation, we hold the following elements constant:
- **Engineered features** - I only use the 29 features I engineered in Assignment 1
- **Dimensionality Reduction** - Other than searching for the best features,
- **Scaled features** - I use the MinMaxScaler for every experiment. This lets me apply the same operation even when I use binary features. But these are not normalized around zero, which is more useful in linear models.
- **Decision Threshold** - I evaluate on F1 Score, but keep the decision threshold constant, rather than cross validate based on confidence values.

Experiment 0: Control
All experiments need a control case. I test an un-tuned version of each of our models to indicate which models are most powerful out of the box.

Model	Train F1	Test F1	Train ROCAUC	Test ROCAUC
LogisticRegression	0.28	0.27	0.58	0.58
DecisionTreeClassifier	0.91	0.32	0.92	0.62
RandomForestClassifier	0.92	**0.35**	0.93	**0.62**
AdaBoostClassifier	0.18	0.17	0.54	0.55

Based on these early results, RandomForest has the best F1 and ROC_AUC scores. This implies it does well at fitting to the complex, non-linear nature of this data.

It's also worth noting that Decision Trees and Random Forest have the greatest variance between train and test scores (high variance). Logistic Regression and AdaBoost have consistent but low scores (e.g. high bias, low variance). This indicates tree-based models help stabilize variance.

Experiment 1: Tune hyperparams with RandomSearch
Goal: tune a DecisionTree to the best of our ability. Tun a random search with 5-fold cross validation and keep the best result

I tuned the following range of parameters:

```
param_dist_dt = {
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_leaf': [1, 2, 6, 10],
    'max_leaf_nodes': [10, 20, 30, None],
    'min_samples_split': [2,5,7,10],
    'class_weight': ['balanced', 'balanced_subsample']
}
```

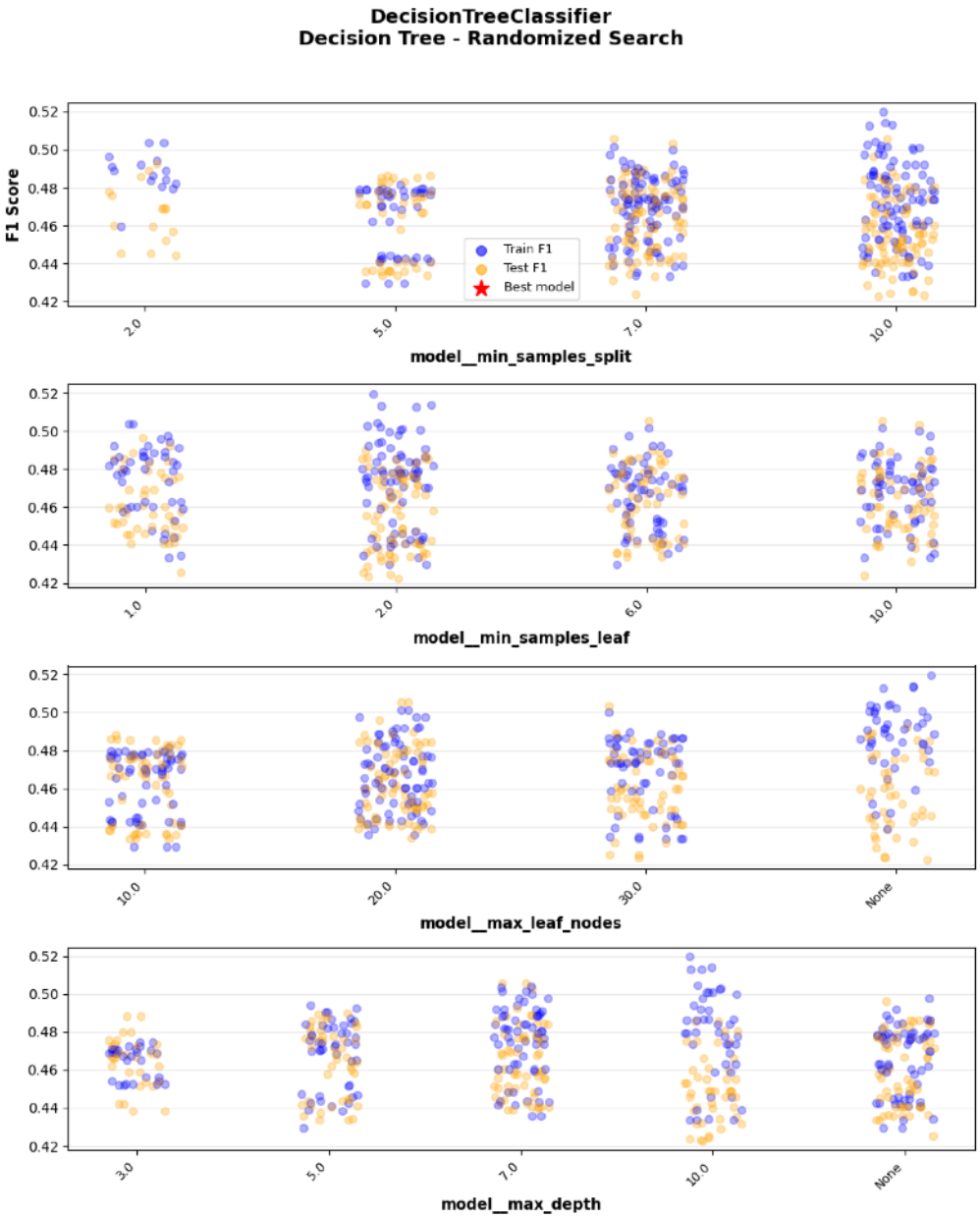
Findings -

```
Best parameters: {'min_samples_split': 7, 'min_samples_leaf': 10, 'max_leaf_nodes': 30,
'max_depth': 7, 'class_weight': 'balanced'} Best F1 score: 0.4715
```

Findings:

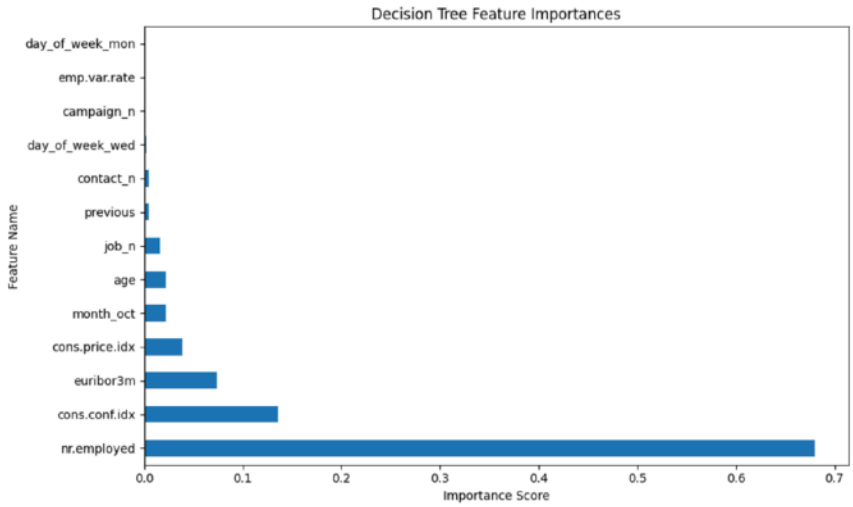
- 7 min sample split
- 10 samples per leaf
- 30 max leaf nodes
- 7 max depth
- balanced class weight - although we have an unbalanced training set, this parameter does not help tune the Decision Tree.

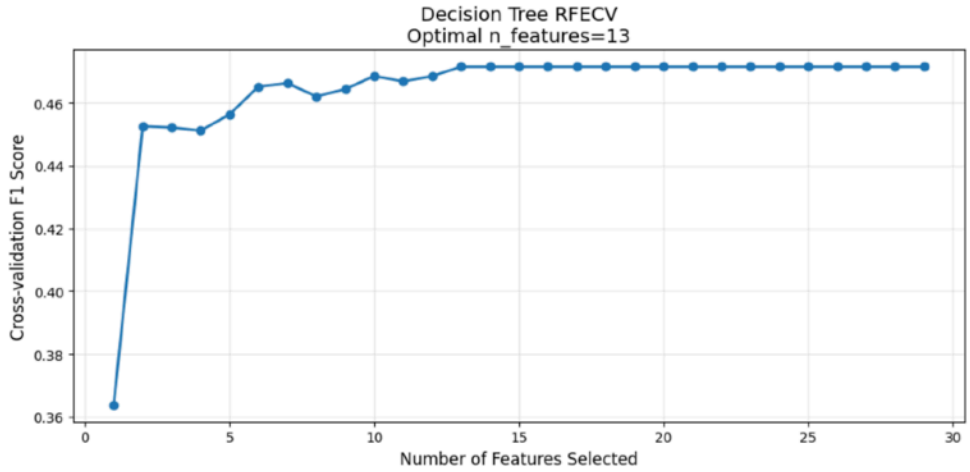
Across each of these parameters, variance remains consistently low (train and test scores are close to each other). The best-performing model has medium bias (0.4715 F1 score).



Experiment 2: Decision Tree - recursive feature elimination
Goal: by dropping out features one at a time, identify which features add the most predictive power to the model, and at which point we are at risk for overfitting.

Findings: recursive feature elimination selects 13 as the optimal number of features, but F1 Scores start to plateau after 2-5 features. “Employment rate” is by far the most important feature, with very little variance explained by the other features. Shallower trees reduced variance but increased bias, which is consistent with theory.





Experiment 3: Random Forest: Tune Hyperparameters

Goal: Find the best hyperparameters of the following values:

```
param_dist_rf = {
    'n_estimators': [10, 25, 50, 100],
    'max_depth': [3, 5, 7, 10, 15, 20, None],
    'min_samples_split': [2, 5, 10, 15, 20],
    'min_samples_leaf': [1, 2, 6, 10],
    'max_features': [7,10,None],
    'class_weight':['balanced_subsample', 'balanced']

}
```

Results:

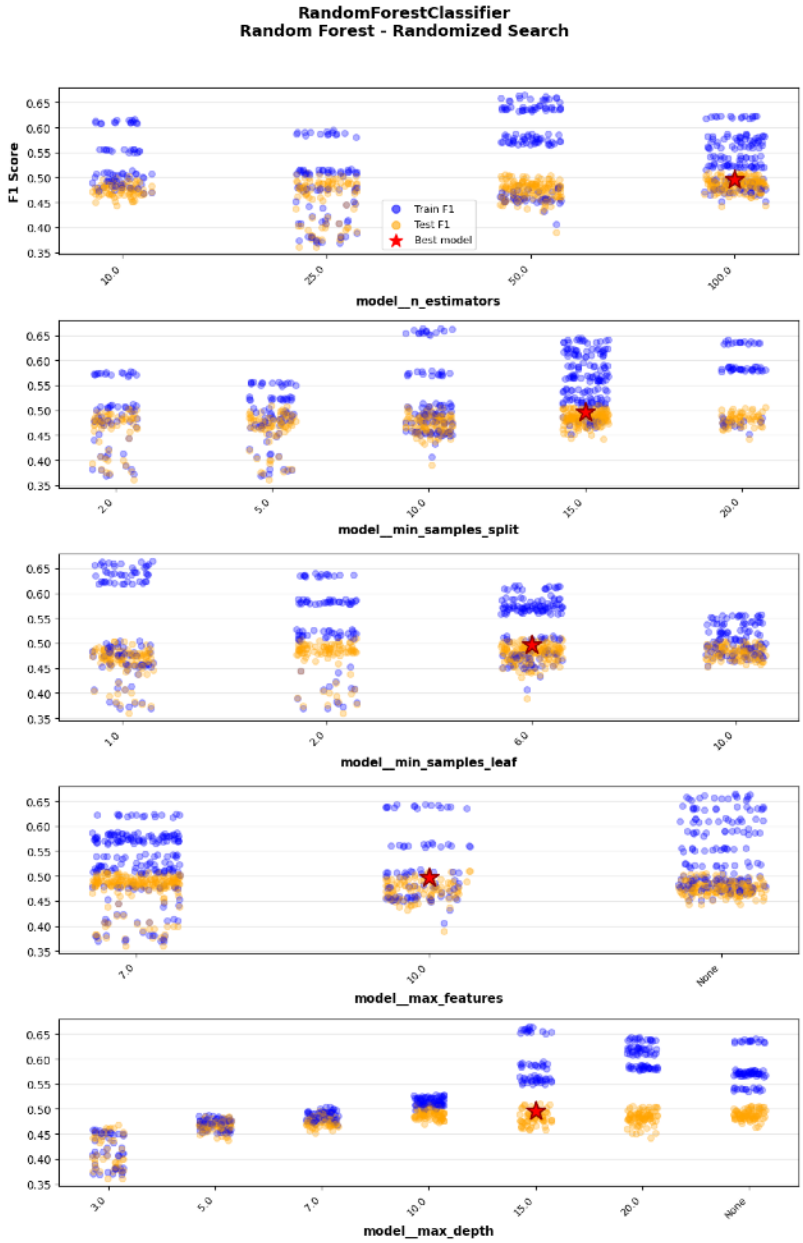
```
Best parameters: {'n_estimators': 100, 'min_samples_split': 15, 'min_samples_leaf': 6,
'max_features': 10, 'max_depth': 15, 'class_weight': 'balanced_subsample'}
Best F1 score: 0.4963
```

Findings:

- 100 Estimators - Random Forest will always fit better as you add complexity.

- 15 Min Sample Split - At greater values, I see greater divergence between train and test scores. Variance increases rapidly above this value
- 6 min samples per leaf - we could hold this smaller. Our F1 scores plateau at 2
- 10 max features - our F1 scores drop off after the first 7. This indicates we have too many similar dimensions, and need to regularize or select new features
- 15 max depth - This is where there is the greatest divergence between train and test. The model no longer generalizes well if we do not prune trees to get smaller than this.

Random Forest significantly reduces variance relative to a single tree but introduces some bias as trees are averaged. The smaller generalization gap in our tuned Random Forest confirms this improved robustness.

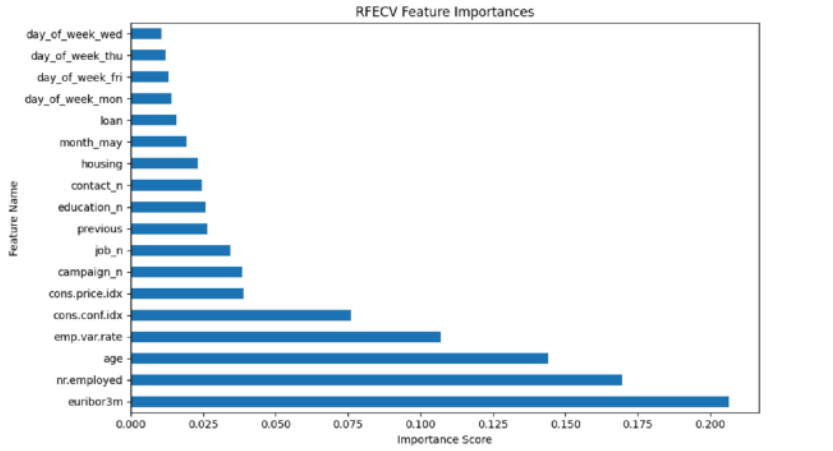


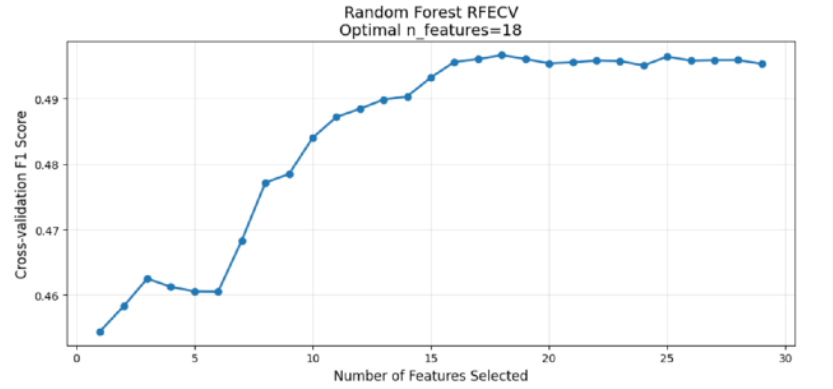
Experiment 4: Random Forest: Recursive feature elimination

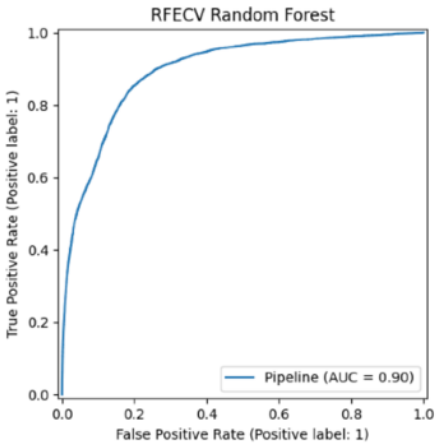
Results

```
RFECV result: 18 features, F1=0.4965
Selected features: ['age', 'education_n', 'job_n', 'housing', 'loan', 'campaign_n', 'previous',
'emp.var.rate', 'cons.conf.idx', 'euribor3m', 'cons.price.idx', 'nr.employed', 'month_may',
'day_of_week_fri', 'day_of_week_mon', 'day_of_week_thu', 'day_of_week_wed', 'contact_n']
```

Findings:
Recursive feature elimination with Random Forest identifies more features than doing so with Random Forest alone. The day of the week features are included. But we get the most variance explained from the exogenous economic indicators: euribor 3 month index and employment rate. Age is also a strong predictor.

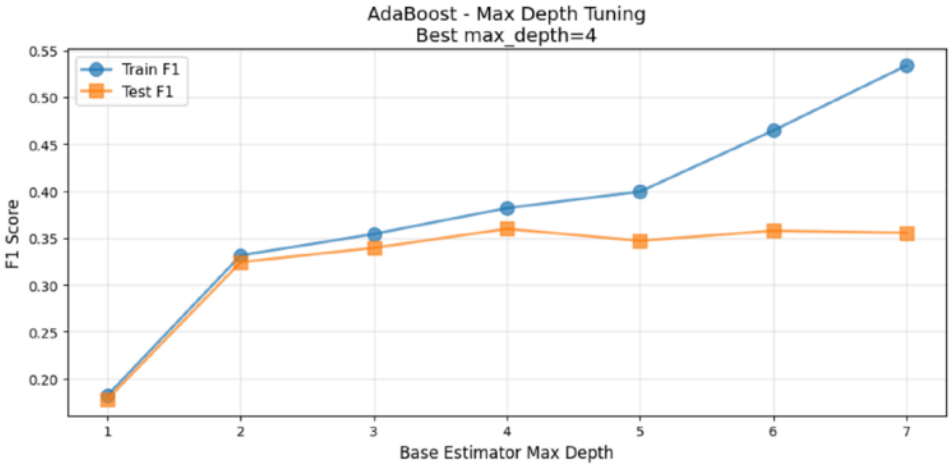






Experiment 5: Adaboost - Tune Max Depth
Goal: before tuning the hyperparameters of an Adaboost model, identify how complex the "weak learners" could be before we start adding hundreds of estimators. We hold "N Estimators" fixed at 50 for this round, so that we have the opportunity to boost based on these models.

Findings: test scores start to fall off after Max Depth of 4. If trees are more complex than that, the boosted model becomes prone to overfitting.




```
Best max_depth: 4, F1=0.3598
```

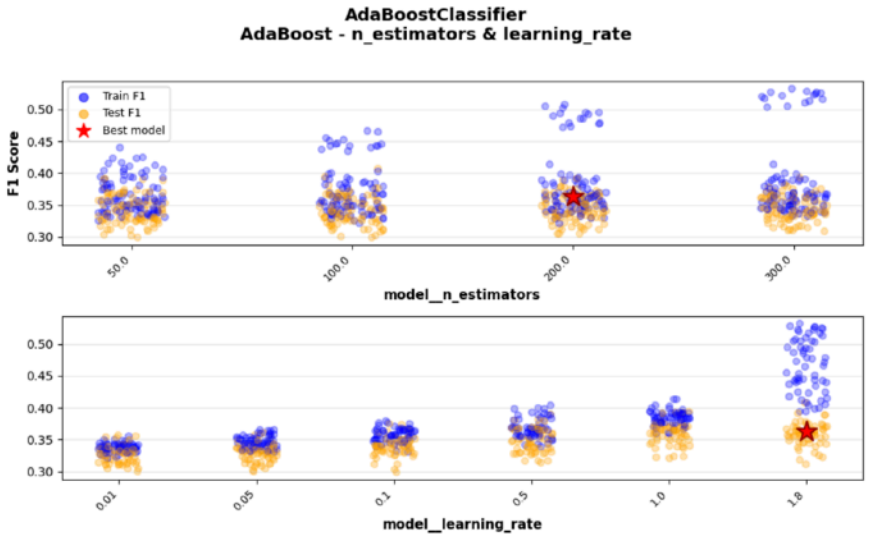
Experiment 6: Adaboost - Tune learning rate and number of estimators

```
param_dist_ada = {
    'n_estimators': [50,100,200,300],
    'learning_rate': [0.01, 0.05, 0.1, 0.5, 1.0, 1.8]
}
Best parameters: {'n_estimators': 200, 'learning_rate': 1.8}
```

Findings:

- 200 estimators - this is very complex, but the model overfits at 300, showing clear diminishing returns
- 1.8 learning rate - Even though this had the highest F1 score selected from RandomSearch, we see how much training scores and test scores diverge as learning rate increases. Since we already have a high number of estimators, I would rather use a smaller

Increasing learning rate reduces bias but rapidly inflates variance, leading to overfitting



Final Results
After tuning, all of our models have improved significantly above their baselines. Even so, Random Forest has the best performance. AdaBoost performs significantly worse than RandomForest - this may indicate sensitivity to noise or weakly predictive features. Or it may indicate that we need additional tuning.

Decision Trees exhibited high variance; Random Forest reduced variance by averaging multiple weak learners; AdaBoost reduced bias initially but overfit at high learning rates. The results confirm ensemble methods' theoretical advantages in stabilizing predictions.

I would not recommend bringing any of these models to production yet. There are too many experiments left to run (feature engineering, dimensionality reduction, experimenting with scalers, etc. See **Assumptions, p1**).

To take this further, I would create many more combinations of features, and build an engine to reduce dimensions (run PCA, or tSNE) and reduce features. Once I have sets of predictive features, I would continue to refine these models based on narrowed hyperparameter ranges that we uncovered during this process.

Business Outcome - the best model only has 0.43 precision and .59 recall. It performs better than random guessing, so I would only use it if there is not a model currently deployed to predict client subscription (assuming cost of deployment is low).

experiment_name	**train_f1**	**test_f1**	**train_roc_auc**	**test_roc_auc**	test_recall	test_precision
Decision Tree - Randomized Search	0.478	0.472	0.750	0.745	0.6189	0.3818
Decision Tree RFECV	0.476	0.470	0.750	0.745	0.62	0.3799
Random Forest - Randomized Search	0.561	0.496	0.795	0.748	0.5998	0.4233
Random Forest RFECV	0.549	**0.497**	0.788	**0.750**	0.6044	0.4214
AdaBoost (max_depth=4)	0.382	0.360	0.628	0.619	0.2647	0.564
AdaBoost - n_estimators & learning_rate	0.489	0.363	0.678	0.623	0.2791	0.5206