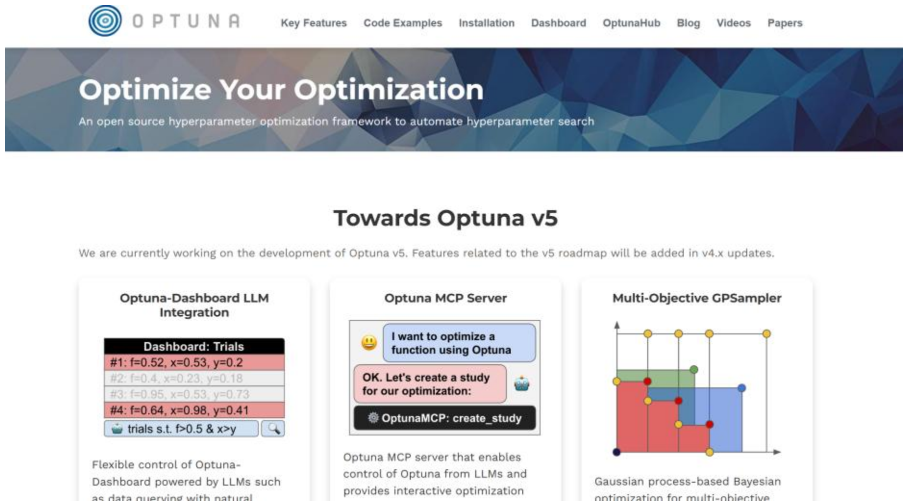
Hyperparameter optimization using Optuna



Clinical AI Study Group

Amit Saha

Hyperparameter tuning

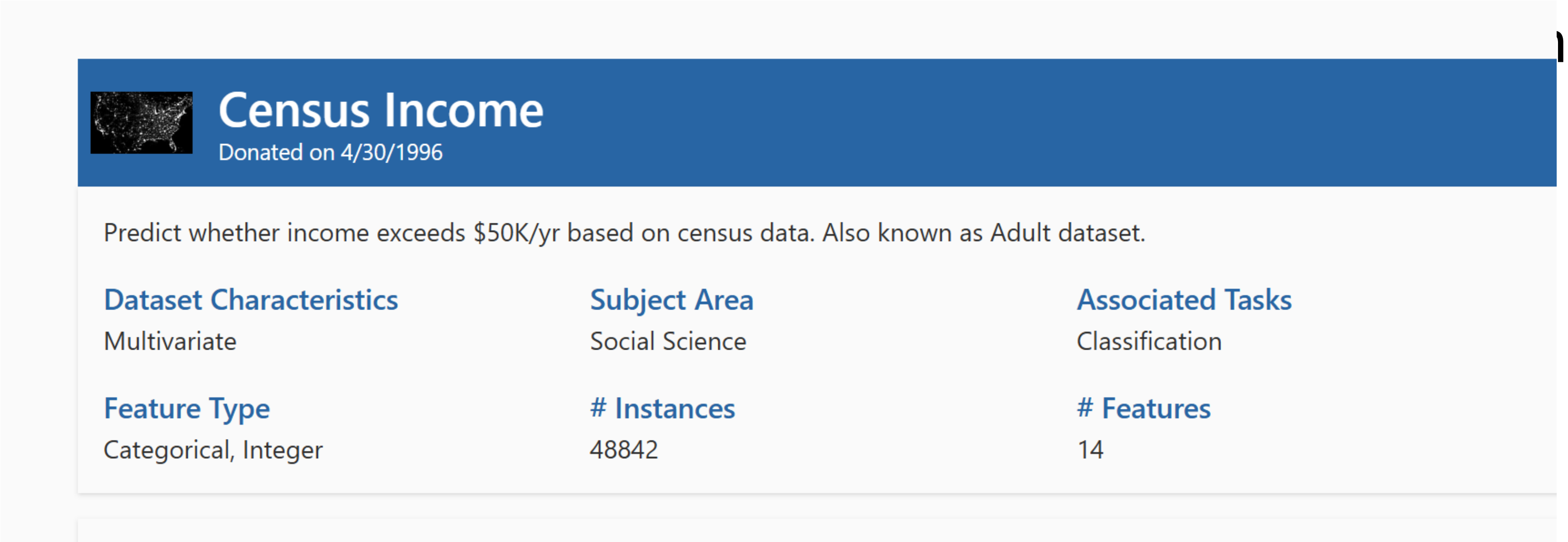
- ML algorithms have model parameters and objective functions – which are optimized during the learning process

- <u>Hyperparameter tuning</u> is the process of intelligently searching the search space of the <u>parameters</u> that are specified for the learning – i.e., <u>user input/configuration</u>

Today’s demo

- XGBoost (eXtreme Gradient Boosting) as the ML algorithm for classification
  - Ensemble method, multiple trees

- e income



XGBoost

$$\text{Final Prediction} = \sum_{m=1}^{M} \eta \cdot f_m(x)$$

- M: number of estimators (trees)
- $f_m(x)$: the m-th tree's prediction
- η: learning rate
- each f_m is a small regression tree (a "weak learner")

XGBoost – Hyperparameters of interest

- <u>Learning rate</u>: step size of model weight update
- <u>Number of estimators</u>: number of trees
- <u>Max depth</u>: depth of each tree

Guidelines

1. Small learning rate with larger number of estimators
    1. Balance between stability and training speed (lower learning rate)
2. More depth results in potential overfitting the training data
3. Higher number of estimators (boosting rounds), better accuracy

Hyperparameter tuning using Optuna

- ML algorithm agnostic

- <u>Study</u> – set up range of the parameters, which sampling (optimization algorithm) algorithm to use, and whether to maximize or minimize <u>objective(s)</u>

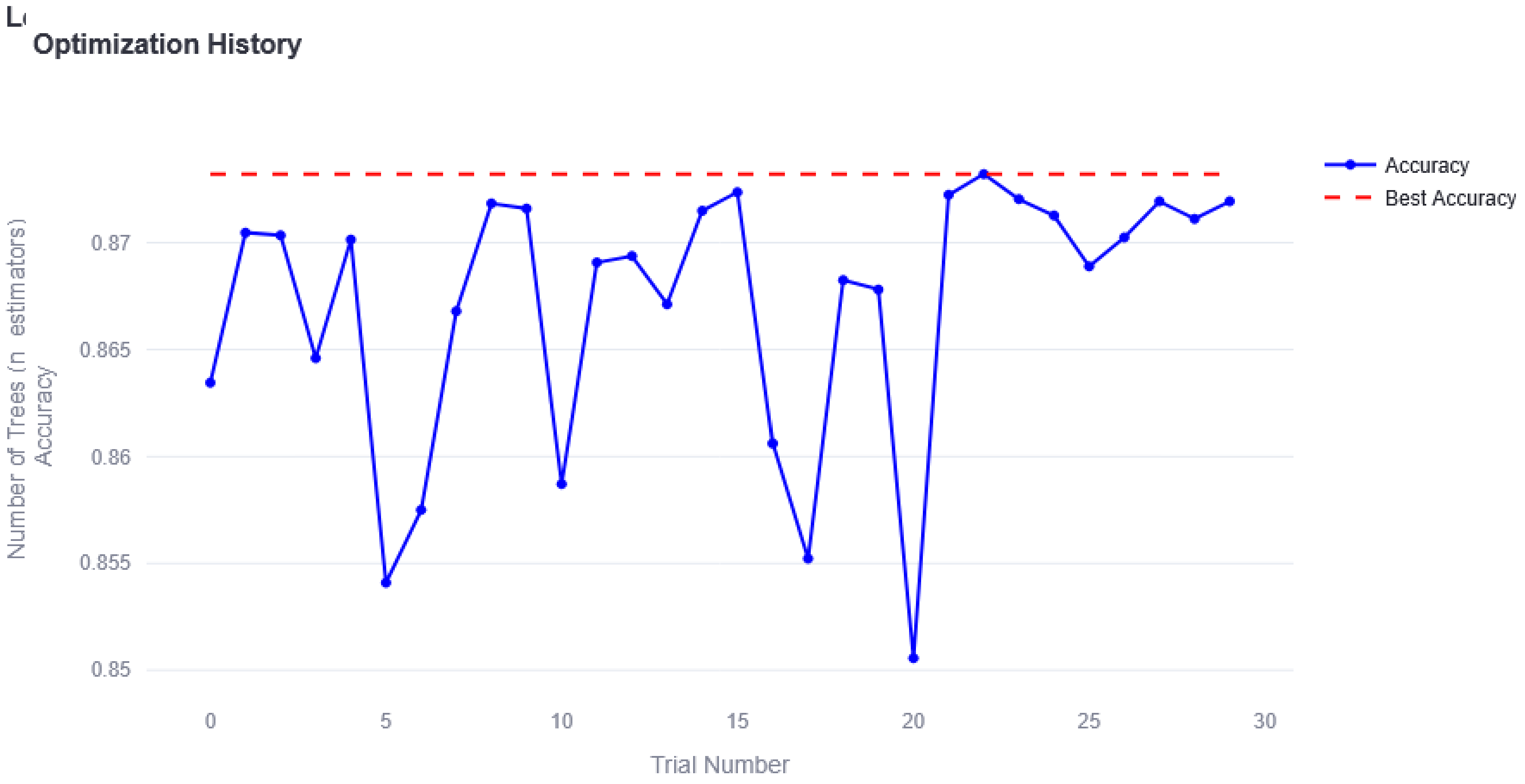
- <u>Trial</u> – a single run of the ML algorithm + the parameters chosen from the study configuration

Objective function – single objective

```python
def objective(trial, X_train, y_train):
    param = {
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3, log=True),
        'n_estimators': trial.suggest_int('n_estimators', 50, 300)
    }
    model = XGBClassifier(**param)

    scores = cross_val_score(model, X_train, y_train, cv=3, scoring='accuracy', n_jobs=-1, error_score='raise')

    return float(scores.mean())
```

Study definition – single objective

```python
def run_single_objective_optimization(n_trials=50):

    study = optuna.create_study(direction='maximize', study_name='xgboost_single_objective')

    study.optimize(lambda trial: objective(trial, X_train, y_train), n_trials=n_trials)

    best_params = study.best_params.copy()
    best_params.update({
        'random_state': 42,
        'eval_metric': 'logloss',
    })

    final_model = XGBClassifier(**best_params)
    final_model.fit(X_train, y_train)

    # Evaluate on test set
    test_accuracy = final_model.score(X_test, y_test)

    return study
```



L
Optimization History
Accuracy
Best Accuracy
Number of Trees (n_estimators)
Accuracy
0.87
0.865
0.86
0.855
0.85
0
5
10
15
20
25
30
Trial Number


Learning Rate vs Number of Trees
Number of Trees (n_estimators)
300
250
200
150
100
50
0
0.05
0.1
0.15
0.2
0.25
0.3
Learning Rate
Best
Accuracy
0.87
0.865
0.86
0.855

DEMO

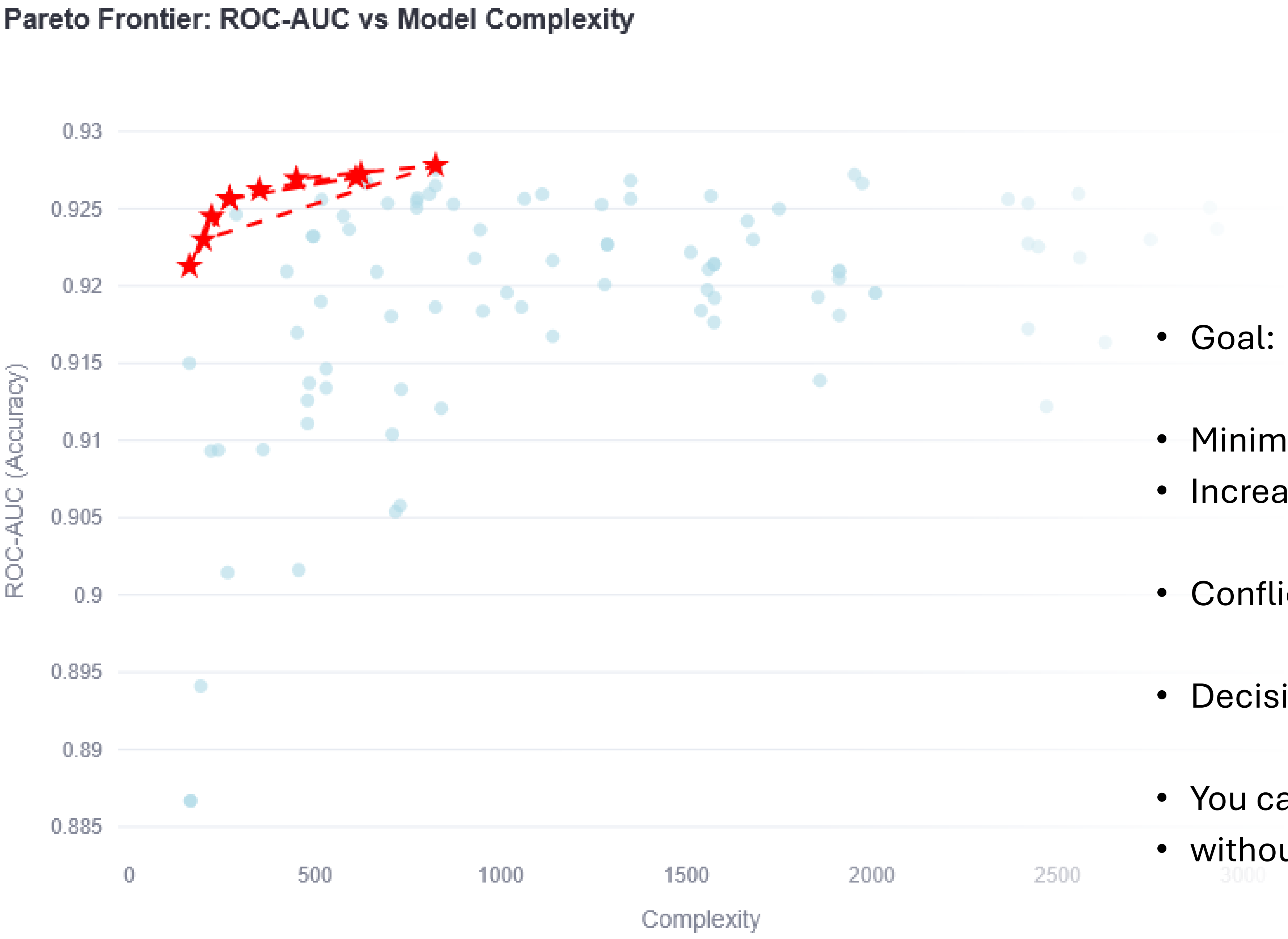
Objective function – multi objective

```python
def objective(trial, X_train, y_train):
    # Define hyperparameter search space
    param = {
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3, log=True),
        'n_estimators': trial.suggest_int('n_estimators', 50, 300),
    }

    # Create XGBoost classifier
    model = XGBClassifier(**param)

    # Objective 1:
    auc_scores = cross_val_score(
        model, X_train, y_train, cv=3, scoring='roc_auc', n_jobs=-1, error_score='raise'
    )
    val_auc = auc_scores.mean()
    performance = val_auc

    complexity = param['n_estimators'] * param['max_depth']

    return performance, complexity
```

Study definition – multi objective

```
def run_multi_objective_optimization(n_trials=50):
    study = optuna.create_study(
        directions=['maximize', 'minimize'],  # maximize roc auc, minimize complexity
        study_name='xgboost_multi_objective'
    )

    study.optimize(lambda trial: objective(trial, X_train, y_train), n_trials=n_trials)

    # Pareto optimal solutions
    study.best_trials
```

DEMO



- Goal:

- Minimize complexity
- Increase performance

- Conflicting objectives

- Decision making:

- You cannot move up
- without moving right

	RandomSampler	GridSampler	TPESampler	CmaEsSampler	NSGAIISampler	QMCSampler	GPSampler	BoTorchSampler
Float parameters	✓	✓	✓	✓	▲	✓	✓	✓
Integer parameters	✓	✓	✓	✓	▲	✓	✓	▲
Categorical parameters	✓	✓	✓	▲	✓	▲	✓	▲
Pruning	✓	✓	✓	▲	× (▲ for single-objective)	✓	▲	▲
Multivariate optimization	▲	▲	✓	✓	▲	▲	✓	✓
Conditional search space	✓	▲	✓	▲	▲	▲	▲	▲
Multi-objective optimization	✓	✓	✓	×	✓ (▲ for single-objective)	✓	×	✓
Batch optimization	✓	✓	✓	✓	✓	✓	▲	✓
Distributed optimization	✓	✓	✓	✓	✓	✓	▲	✓
Constrained optimization	×	×	✓	×	✓	×	×	✓
Time complexity (per trial) (*)	$O(d)$	$O(dn)$	$O(dn \log n)$	$O(d^3)$	$O(mp^2)$ (***)	$O(dn)$	$O(n^3)$	$O(n^3)$
Recommended budgets [illegible]	[illegible]	[illegible]	[illegible]	[illegible]	100 – 10000	as many as one likes	– 500	10 – 100

Optimization algorithms used

- Optuna calls them samplers
    - Single objective – TPESampler (Tree-structured Parzen Estimator)
    - Multi-objective – NSGA-IISampler (Evolutionary algorithm based optimizer)
- There is also an option to choose the sampling algorithm "automatically" based on your data - AutoSampler

Conclusion

- Optuna makes hyperparameter tuning more manageable rather than having to manually try out various ranges

- Slides, and demos: https://github.com/amitsaha/optuna-demo