

Hyperparameter optimization using Optuna

Clinical AI Study Group

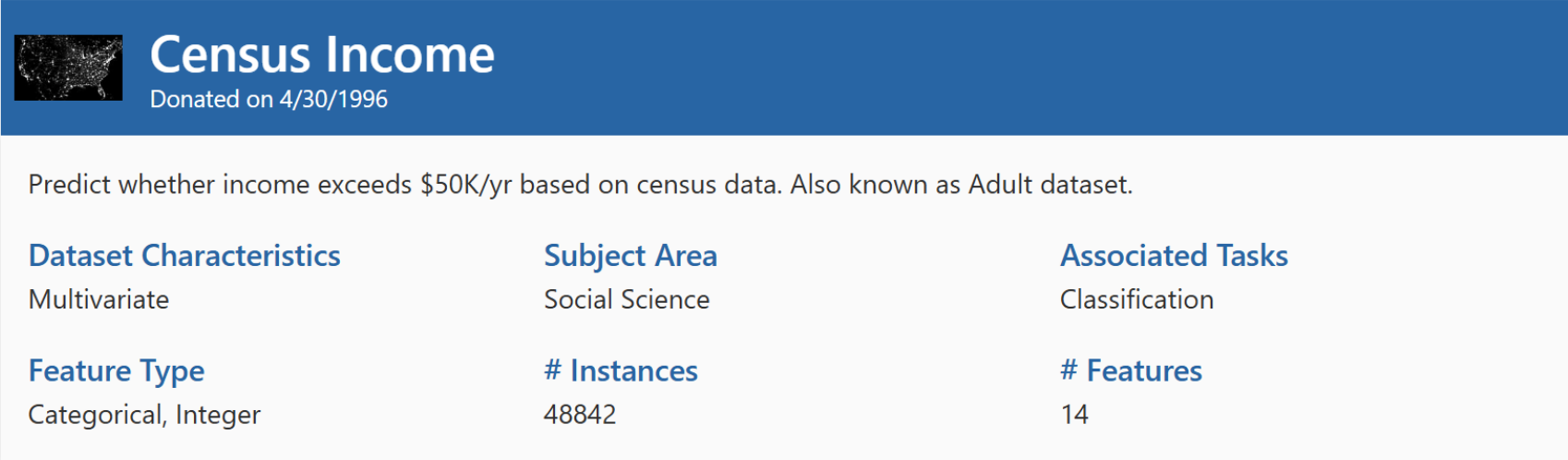
Amit Saha


Hyperparameter tuning

- ML algorithms have model parameters and objective functions – which are optimized during the learning process
- Hyperparameter tuning is the process of intelligently searching the search space of the parameters that are specified for the learning – i.e., user input/configuration

Today's demo

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

- e income



Census Income

Donated on 4/30/1996

Predict whether income exceeds \$50K/yr based on census data. Also known as Adult dataset.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Social Science	Classification
Feature Type	# Instances	# Features
Categorical, Integer	48842	14

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

- Learning rate: step size of model weight update
- Number of estimators: number of trees
- Max depth: 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
- Study – set up range of the parameters, which sampling (optimization algorithm) algorithm to use, and whether to maximize or minimize objective(s)
- Trial – a single run of the ML algorithm + the parameters chosen from the study configuration

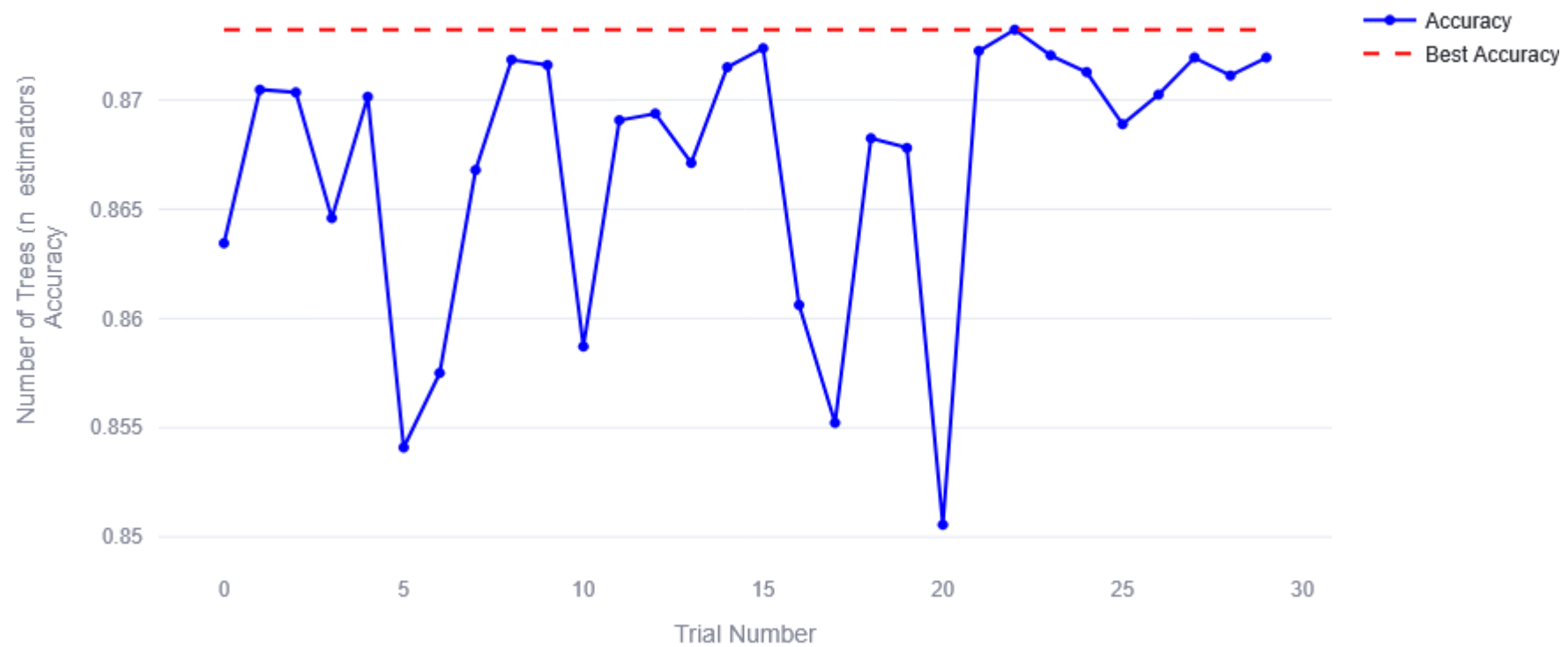
Objective function – single objective

```
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

```
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



Learning Rate vs Number of Trees



DEMO

Objective function – multi objective

```
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']

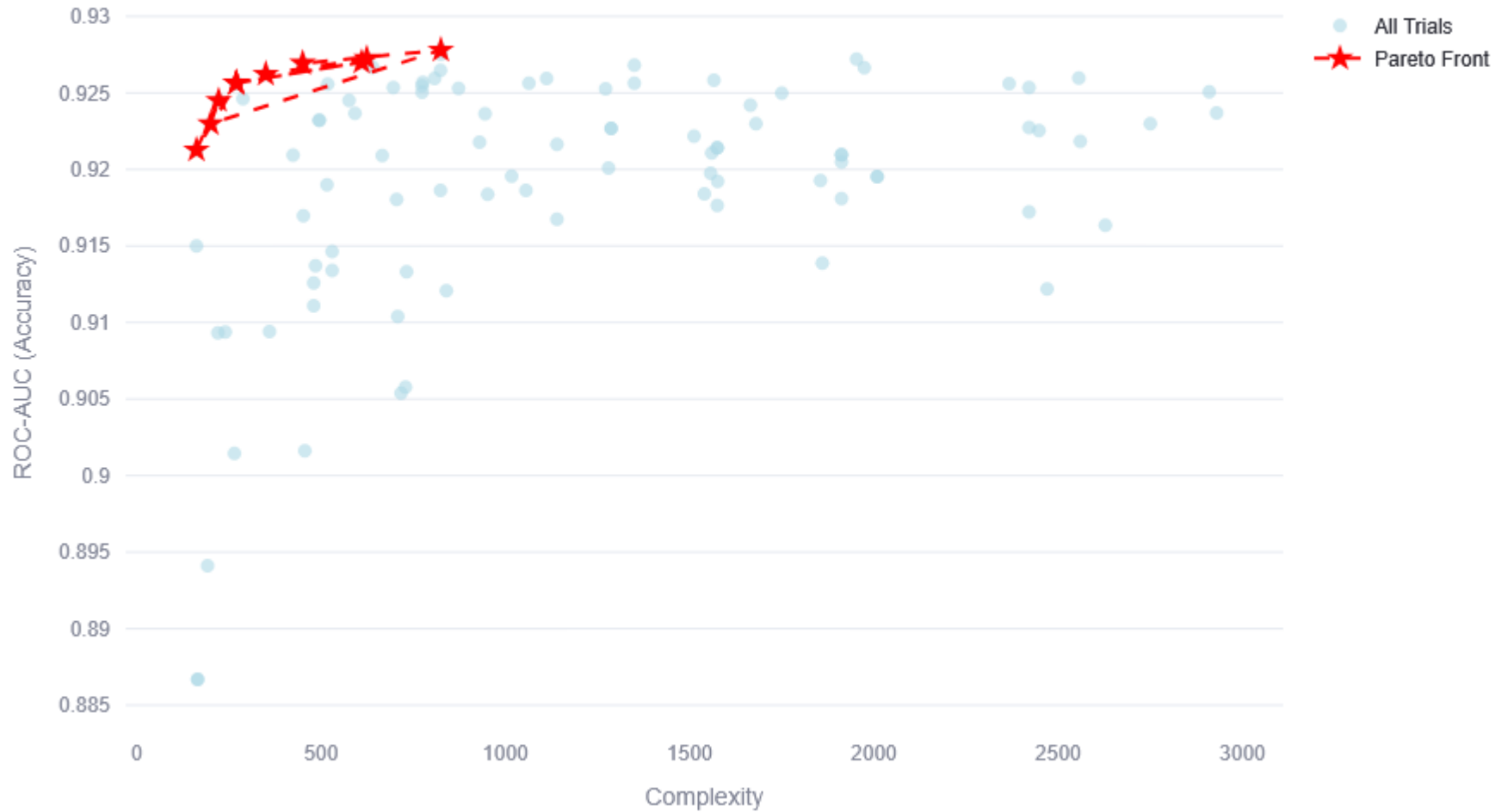
    return performance, complexity
```

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

Pareto Frontier: ROC-AUC vs Model Complexity



Goal:

Minimize complexity
Increase performance

Conflicting objectives

Decision making:

You cannot move up
without moving right

Conclusion

- Optuna makes hyperparameter tuning more manageable
- Slides, and demos: <https://github.com/amitsaha/optuna-demo>