Week 5: Hyperparameter Optimization & Advanced Search Strategies

Heart Failure Survival Analysis

MDST Project

Winter 2026

Outline

Quick Recap: Week 4 - Classification Models

Algorithms Trained:

1. Logistic Regression
2. Random Forest
3. Support Vector Machines
4. K-Nearest Neighbors

Key Concepts:

- Train/Test split with stratification
- Feature normalization
- Evaluation metrics (accuracy, precision, recall, F1, ROC-AUC)

This Week: Make these models even better by optimizing their hyperparameters!

From Basic Models to Optimized Models

Last Week: We trained models with default hyperparameters

Example - Random Forest:

- n_estimators = 100 (default)
- max_depth = None (default)
- Test accuracy = 73%

This Week: Find better settings!

- n_estimators = 50 (tuned)
- max_depth = 10 (tuned)
- Test accuracy = ??? (could be better!)

Question: How do we find these optimal values systematically?

What Are Hyperparameters?

Two Types of Parameters:

Model Parameters	**Hyperparameters**
Learned from data during training	Set before training
Weights in neural networks Coefficients in logistic regression	Learning rate, batch size Number of iterations

Random Forest Hyperparameters:

- **n_estimators:** Number of trees (more → better but slower)
- **max_depth:** Maximum tree depth (deeper → more complex)
- **min_samples_split:** Minimum samples to split a node
- **max_features:** Number of features to consider per split

The Problem: Too Many Combinations!

Scenario: Fine-tune Random Forest

Parameter Ranges:

- n_estimators: [20, 50, 100, 150, 200] (5 values)
- max_depth: [5, 10, 15, 20] (4 values)
- min_samples_split: [2, 5, 10] (3 values)
- max_features: ['sqrt', 'log2'] (2 values)

Total Combinations:

$$5 \times 4 \times 3 \times 2 = 120 \text{ models}$$

If each model takes 1 second to train:

$$120 \text{ seconds} = 2 \text{ minutes}$$

With more parameters?

$$10 \times 10 \times 10 \times 10 = 10,000 \text{ models!}$$

Solution: Use smart search strategies instead of trying everything!

GridSearchCV: Exhaustive Search

Strategy: Test all combinations in the parameter grid

Process:

1. Define parameter grid (all combinations)
2. For each combination:
    - Train model on training data
    - Evaluate with k-fold cross-validation
    - Record performance
3. Return best parameters

Code Pattern: `grid = GridSearchCV(model, param_grid, cv=5)`

```
grid.fit(X_train, y_train)
best_params = grid.best_params_
```

Pros: Exhaustive, guaranteed best in grid
Cons: Slow, exponential growth with parameters

Results from GridSearchCV

Parameter Grid:

- n_estimators: [20, 50, 100, 150, 200]
- max_depth: [None, 10, 20]
- min_samples_split: [2, 5, 7, 10, 13]
- max_features: ['sqrt', 'log2']

Total Models Evaluated: 150

Results:

- Best Validation Accuracy: **76.53%**
- Best Parameters: n_estimators=50, max_depth=10, min_samples_split=5, max_features='sqrt'
- Test Accuracy: **73.33%**

Observation: Works well, but 150 evaluations is expensive with larger models!

Random Search: Smarter Sampling

Strategy: Randomly sample from parameter space (not all combinations)

Key Idea:

- Instead of grid: define ranges (continuous or discrete)
- Sample randomly from these ranges
- With enough trials, likely to find good parameters
- Much faster than GridSearch

Why It Works:

- You don't need to evaluate every grid point
- Randomness helps explore the space
- Embarrassingly parallel (can run trials in parallel)

Optuna Advantage: Better API than GridSearch, cleaner code

Optuna: Modern Framework

What is Optuna?

- Lightweight hyperparameter optimization framework
- Supports multiple search strategies
- Easy to use, flexible parameter types

Basic Workflow:

1. Define objective function (returns metric to maximize) 2. Define parameter ranges within objective function 3. Create study with desired sampler 4. Optimize by running trials

Code:
```
study = optuna.create_study(sampler=RandomSampler())
study.optimize(objective_fn, n_trials=50)
best_params = study.best_params
```

Random Search Results (50 trials)

Setup:

- Sampler: RandomSampler (random exploration)
- Number of trials: 50 (4x less than GridSearch!)
- Same parameter ranges as GridSearch

Results:

- Best Validation Accuracy: **?**
- Best Parameters: **varies each run**
- Test Accuracy: **?**

Advantages:

- 3x faster than GridSearch (50 vs 150 models)
- Can handle continuous parameter ranges
- Good for exploratory tuning

Disadvantages:

- No learning from previous trials

Bayesian Optimization: Intelligent Search

Key Idea: Learn from previous trials to propose better parameters

How It Works:

1. Start with random trials to explore
2. Build probability model of objective function
3. Use model to propose promising parameters
4. Evaluate new parameters
5. Update model with new observation
6. Repeat steps 3-5

Advantage Over Random Search:

- **Exploitation:** Try parameters near previous good results
- **Exploration:** Still explore uncertain regions
- Uses all past information, not just random samples

Result: Better performance with fewer trials!

Tree-structured Parzen Estimator (TPE)

What is TPE?

- State-of-the-art Bayesian optimization algorithm
- Used by Optuna by default
- Very effective and fast

Key Property: "Exploitation vs Exploration"

Early Trials:

- Mostly explore randomly
- Build initial understanding

Later Trials:

- Focus on promising regions
- Still explore uncertain areas

Code:
```
study = optuna.create_study(sampler=TPESampler())
study.optimize(objective_fn, n_trials=50)
```

Bayesian Optimization Results (50 trials)

Setup:

- Sampler: TPESampler (Bayesian + intelligent search)
- Number of trials: 50
- Same parameter ranges and objective function

Results:

- Best Validation Accuracy: **?**
- Best Parameters: **likely better than random**
- Test Accuracy: **?**

Key Observation:

- Same number of trials as Random Search (50)
- But Bayesian typically finds better parameters
- Because it learns from previous trials

Practical Impact: Get GridSearch quality in 1/3 the time!

Comparing All Three Approaches

Quick Comparison Table:

	GridSearch	**Random**	**Bayesian**
Trials	150	50	50
Speed	Slow	Medium	Fast
Learning	None	None	YES
Best For	Small grids	Exploration	**Most cases**

Efficiency: Bayesian $>$ Random $>$ GridSearch

Quality: Bayesian $\geq$ GridSearch $>$ Random

When to Use Each Approach

Use GridSearchCV When:

- Parameter space is small and discrete
- You have plenty of compute resources
- Parameters are well-understood

Use Random Search When:

- Quick exploration needed
- Easily parallelizable
- Simple implementation preferred

Use Bayesian Optimization (Optuna) When:

- **[RECOMMENDED FOR MOST PROJECTS]**
- Training models is expensive/slow
- Mix of continuous and discrete parameters
- Want best results with reasonable compute
- Working on real problems (not toy examples)

Pruning: Early Stopping

Idea: If a model looks bad halfway through, stop training it!

Example Scenario:

- Trial 1 (bad hyperparams): epoch 1 acc=45%, epoch 10 acc=46%
- **Prune this trial** instead of training all 100 epochs
- Save computation time!

Useful For:

- Neural networks (many epochs)
- Expensive models
- When training takes hours/days

Code Pattern:
```
for epoch in range(100):
  acc = train_epoch()
  trial.report(acc, epoch)
  if trial.should_prune():
    raise optuna.TrialPruned()
```

Kaggle-Style Challenge

Your Task:

1. Use GridSearch with different ML classifiers
2. Find best hyperparameters for each
3. Compare test accuracy across models
4. Goal: Highest test accuracy wins!

Fixed Parameters:

- Train/Test split: 70/30, stratified, random_state=21
- Cross-validation: 5-fold
- Metric: accuracy

Models to Try:

- Logistic Regression
- Random Forest
- Support Vector Machines
- Gradient Boosting (new!)

Key Takeaways

Hyperparameters matter! Default parameters are rarely optimal.

Three Search Strategies:

1. **GridSearchCV:** Exhaustive but slow
2. **Random Search:** Faster, less learning
3. **Bayesian (Optuna):** Smart and efficient [RECOMMENDED]

Practical Recommendations:

- Start with GridSearch for small spaces
- Use Optuna for anything larger/expensive
- Always validate on test set, never train set!
- Compare multiple models, not just one

Next Steps:

- Implement hyperparameter tuning
- Compare models in the competition
- Explore advanced techniques (ensembles, stacking)

Questions?

Week 5: Hyperparameter Optimization

Key Resources:

- GridSearchCV docs: scikit-learn
- Optuna: `https://optuna.org`
- Week 5 notebook has working examples