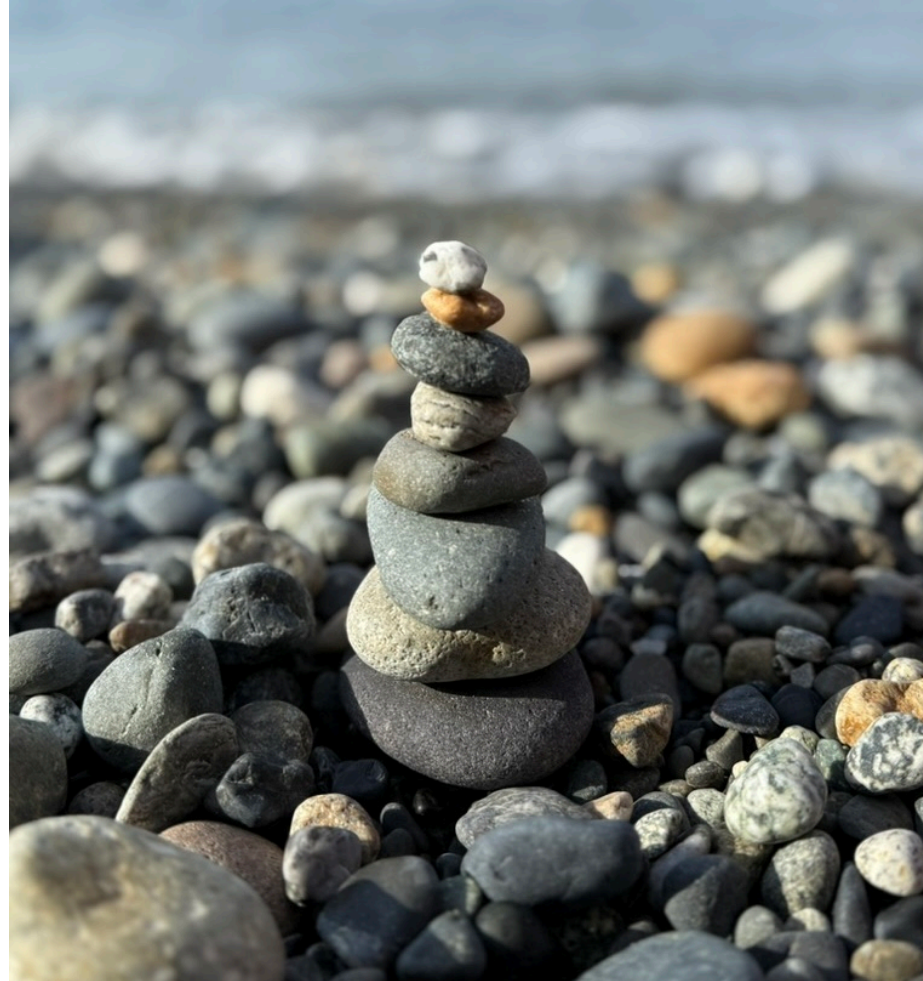


Lecture 8:

Hyperparameter optimization

Varada Kolhatkar

Focus on the breath!



Check-in

How are you feeling today?

- a. Things are more or less under control!
- b. Excited about hyperparameter optimization!
- c. Lost 😞
- d. Tired / sleepy 😴
- e. Thinking of lunch 🥗

Today's big questions

1. How do we choose settings for a **complete pipeline**?
2. How do grid and random search spend a **search budget**?
3. How much should we trust the **winning validation score**?

Recap: Parameters vs. hyperparameters

Component	Learned during <code>fit</code>	Chosen before fitting
Logistic regression	Coefficients, intercept	<code>C</code>
Decision tree	Splits, leaf predictions	<code>max_depth</code>
Bag of words	Vocabulary	<code>max_features</code>

How should we choose the hyperparameters?

For each setting, we already know how to:

1. Fit a model within each CV training fold.
2. Score it on the corresponding validation fold.
3. Compare mean validation scores.

Now automate this workflow for **combinations of settings**.

Choosing a candidate pipeline

Our demo predicts whether a listener **likes a song**.

We will tune a pipeline with:

- Preprocessing for numerical, categorical, binary, and text features.
- An RBF SVM classifier.

The settings interact: **changing the representation can change which model settings work well.**

Why does tuning get expensive?

The demo tries every combination of:

- 2 vocabulary sizes (`max_features`).
- 4 regularization settings (`C`).
- 4 RBF kernel settings (`gamma`).

How many model fits does **five-fold cross-validation** require?

$$2 \times 4 \times 4 = 32 \text{ candidate pipelines}$$

$$32 \times 5 = 160 \text{ CV fits}$$

More combinations and more folds mean more fits.

Demo 1: Data, pipeline, and baseline

Lecture 8 Class Demo

Run **Section 1**:

- Reserve the test set.
- Inspect the Spotify data and preprocessing pipeline.
- Compare the dummy and default SVC baselines.

Which rows may the scaler and vocabulary learn from in each CV fold?

Hyperparameter optimization with loops

- Define a parameter space.
- Iterate through possible combinations.
- Evaluate model performance.

What are some limitations of this approach?

Two ways to search in `sklearn`

Method	Candidates evaluated	Useful when...
<code>GridSearchCV</code>	Every specified combination	We have a small set of combinations to compare
<code>RandomizedSearchCV</code>	A fixed number of sampled combinations	We want to explore a larger space within a budget

Both use **CV on training data**, select a candidate, and by default **refit on all training data**.

Naming pipeline hyperparameters

```
1 pipe_svm = make_pipeline(preprocessor, SVC())
```

Search key	What changes?
<code>svc__C</code>	SVM's <code>C</code>
<code>svc__gamma</code>	SVM's <code>gamma</code>
<code>columntransformer__countvectorizer__max_features</code>	Song-title vocabulary limit

Use `step__parameter`, nesting further when needed.

How to use grid search

```
1 param_grid = {  
2     "columntransformer__countvectorizer__max_features": [100, 400],  
3     "svc__gamma": [0.001, 0.01, 0.1, 1.0],  
4     "svc__C": [0.01, 0.1, 1.0, 10.0],  
5 }  
6 grid_search = GridSearchCV(  
7     pipe_svm, param_grid, cv=5, scoring="accuracy",  
8     n_jobs=-1, return_train_score=True,  
9 )  
10 grid_search.fit(X_train, y_train)
```

Exhaustive within this grid, including preprocessing in every fit.

What does the fitted search contain?

Attribute	Meaning
<code>best_params_</code>	Selected settings
<code>best_score_</code>	Winning mean CV validation score
<code>best_estimator_</code>	Selected pipeline refitted on all training data
<code>cv_results_</code>	Scores and timings for all candidates

`mean_test_score` in `cv_results_` refers to CV validation folds.

It is not a score on our reserved test set.

Choosing useful search ranges

- For positive scale settings such as **C** and **gamma**, explore orders of magnitude.
- Inspect promising regions and interactions, rather than only the winner.
- If a winner lies at a boundary, consider extending that range.

Use training-set CV to guide further searches. Keep the test set reserved.

How to use random search

```
1 list_search = RandomizedSearchCV(  
2     pipe_svm, param_distributions=param_grid, n_iter=20,  
3     cv=5, scoring="accuracy", random_state=123,  
4     n_jobs=-1, return_train_score=True,  
5 )  
6 list_search.fit(X_train, y_train)
```

- Sample 20 of the grid's 32 combinations.
- `random_state` makes candidate sampling reproducible.
- `n_iter` counts **candidates**, not folds or elapsed time.

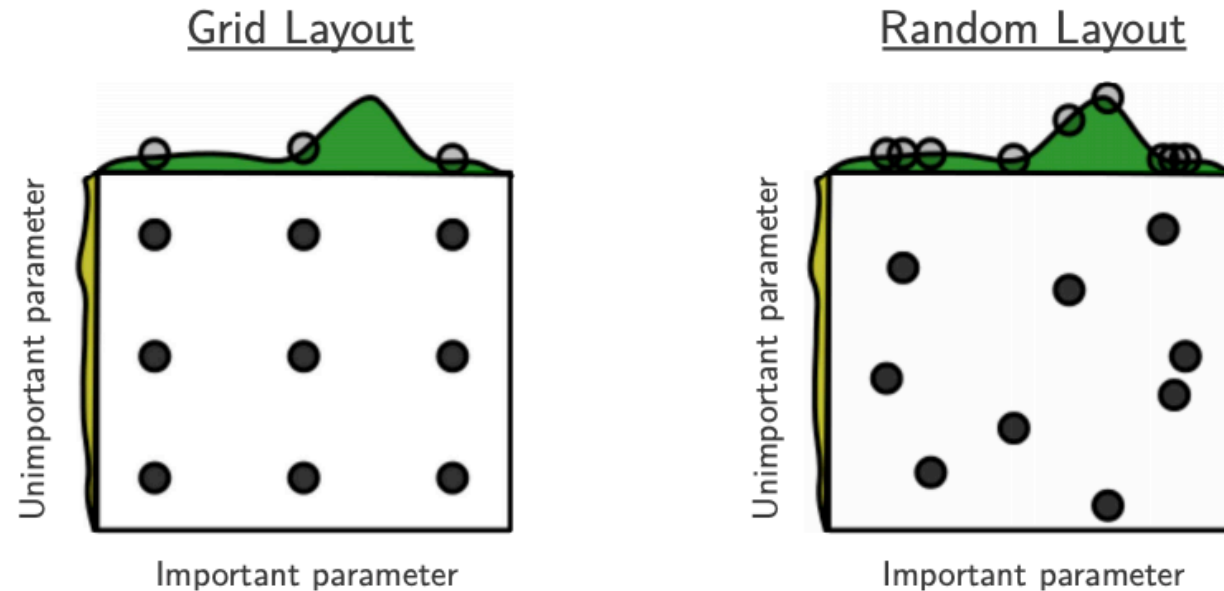
Sample distributions instead of lists

```
1 param_dist = {  
2     "columntransformer__countvectorizer__max_features":  
3         randint(100, 2001),  
4     "svc__C": loguniform(1e-4, 1e4),  
5     "svc__gamma": loguniform(1e-4, 1e4),  
6 }
```

Pass `param_dist` as `param_distributions` to `RandomizedSearchCV`.

- `randint`: integer values; the upper endpoint is **excluded**.
- `loguniform`: equal-sized intervals on log scale have equal probability.

Effectiveness of random search



Nine grid evaluations and nine random evaluations; random search tries more distinct values along the important dimension.

If one dimension matters more, a grid repeats its values while varying other dimensions.

Source: Bergstra and Bengio (2012)

Your turn: Count the fits

The demo uses **32 grid candidates**, **five-fold CV**, and the default final refit.

1. How many pipeline fits does grid search require?
2. How many does random search require with `n_iter=20`?
3. What do we give up by evaluating fewer candidates?

Demo 2: Run and inspect the searches

Run **Sections 2 and 3**, stopping before final test evaluation.

- Inspect selected settings, scores, and timings.
- Compare the heatmaps for vocabulary sizes 100 and 400.
- Try random search with lists and distributions.

Where would you spend more computation?

More trials, higher score, better model?

A validation score reflects:

- Predictive ability.
- Variation from the particular evaluation examples.

Selecting the **highest score** can select a candidate that benefited from favourable noise.

The winning validation score can be **optimistic** about new-data performance.

Winning by guessing

A quiz has **10 questions**, each with **4 options**.

- One randomly completed answer sheet: expected accuracy **25%**.
- Generate **100** random answer sheets.
- Keep the sheet that scores highest on these questions.

Will the winning sheet usually exceed 25%?

Final evaluation: Protect the test set

1. Make modelling decisions using **training-set CV**.
2. Select the final configuration and refit on **all training data**.
3. Evaluate once on the **untouched test set**.

If a test result guides another modelling decision, that set is now part of development.

Your turn: Find two evaluation problems

A student:

1. Fits `CountVectorizer` on all song titles **before splitting**.
2. Searches 200 SVM settings.
3. Reports the winning `mean_test_score` as **final test accuracy**.

What went wrong? How would you fix it?

Demo 3: Evaluate the selected model

Run **Section 4** of the notebook.

```
1 selected_name = max(searches, key=lambda name: searches[name].b  
2 selected_search = searches[selected_name]  
3 selected_search.score(X_test, y_test)
```

Why might this differ from `best_score_`?

Summary

- Tune the **complete pipeline** within CV.
- Grid search evaluates every specified combination.
- Random search controls the **candidate budget**.
- Search ranges matter; inspect more than the winning score.
- Count **candidates** $\times$ **folds** + **final refit**.
- Selection can make the winning CV score optimistic.
- Finish with an **untouched test set**.

Optional: Nested cross-validation

An **outer** CV loop evaluates the whole selection procedure:

1. Run a complete **inner search** within each outer training portion.
2. Evaluate its selected pipeline on the outer held-out portion.
3. Summarize the outer evaluation scores.

Evaluation across multiple splits, at greater computational cost.

Optional: Searches that adapt

- **Successive halving:** begin with many candidates and small resource allocations; give more resources to promising candidates.
- **Bayesian optimization:** use previous trial results to guide later choices.

Both require sensible search spaces and protected evaluation data.

Further reading

- Chapter 8 lecture notes
- scikit-learn hyperparameter-tuning guide
- GridSearchCV API
- RandomizedSearchCV API
- Bergstra and Bengio (2012)