Effective Programming Practices for Economists

Data Analysis in Python

Cross-validation and hyperparameters in scikit-learn

Janoś Gabler, Hans-Martin von Gaudecker, and Tim Mensinger

The bias-variance trade-off

- For prediction, want to be as close to the values to be predicted as possible
- Very simple models, e.g. just an intercept and a couple of regressors
  - Large bias, low variance, no overfitting
- Very large models, e.g. including squares, interactions, …
  - Small bias, high variance, danger of overfitting
- Typically, one or more parameters govern the bias variance trade-off

Example: Penalty in a logit model

- Logistic regression is fit by maximizing a log likelihood function
- Can augment likelihood by a term that penalizes model complexity
- Typically, model complexity means many non-zero parameters
- Penalty is a function of the parameter vector

$$\theta^* = \arg\min_\theta \ell(\theta; X, y) + \lambda \cdot p(\theta)$$

Different penalties

- L1: $p(\theta) = \sum_i |\theta_i|$
  - Penalizes all deviations from zero equally
  - Induces sparsity
  - Harder numerical optimization, not compatible with all optimizers
- L2: $p(\theta) = \sum_i \theta_i^2$
  - Penalizes values close to zero very weakly
  - Does not induce sparsity
  - Simpler numerical optimization

Two splits are not enough

- Want to set tuning parameters optimally
- Naive approach:
  - Fit models with different parameters on training set
  - Evaluate performance on test set
  - Keep the best
- Problem: Hyperparameters are over-fit to the test set
- Use cross-validation to avoid this

K-fold cross validation

- Idea: Split the training data repeatedly into:
  - Data used for actual training
  - Data used for evaluation
- Repeat k times to get k scores
- Keep model that achieves best average score
- Use actual test set only once in the end to measure model quality

Cross-validaton

```
>>> from sklearn.model_selection import cross_val_score
>>> scores = cross_val_score(
...   LogisticRegression(max_iter=3000),
...   X_train,
...   y_train,
...   cv=5
... )
>>> scores
array([
  0.84844291,
  0.84532872,
  0.85709343,
  0.84492904,
  0.86396677
])

>>> scores.mean()
0.8519521727205328
```

- Import and create instance as normal, do not call `fit()`
- L2 penalty is default
- Provide data to `cross_val_score`
- `cv` argument specifies number of folds
- `cross_val_score` will call `fit()` repeatedly

Systematic hyperparameter tuning

- Specify a combination of hyperparameters we want to try
- Calculate cross validation score for each set of parameters
- Keep model with best performance
- Re-fit best model on entire dataset
- Implement in `GridSearchCV`

Grid Search

```
>>> from sklearn.model_selection import GridSearchCV
>>> param_grid = {
...     "penalty": ["l2", "l1"],
...     "C": [0.1, 1, 10],
... }
>>> grid = GridSearchCV(
...     LogisticRegression(solver="liblinear"),
...     param_grid,
...     cv=5,
... )
>>> grid.fit(X_train, y_train)

>>> grid.best_params_
{'C': 10, 'penalty': 'l1'}

>>> grid.best_estimator_.score(
...     X_test,
...     y_test
... )
0.8430232558139535
```

- `param_grid` keys are names of arguments of `LogisticRegression`
- `param_grid` values are lists of possible values for the arguments
- Setting up the grid does not fit models yet
- `grid.fit()` takes some time and often produces warnings