4.2 - Model Selection

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

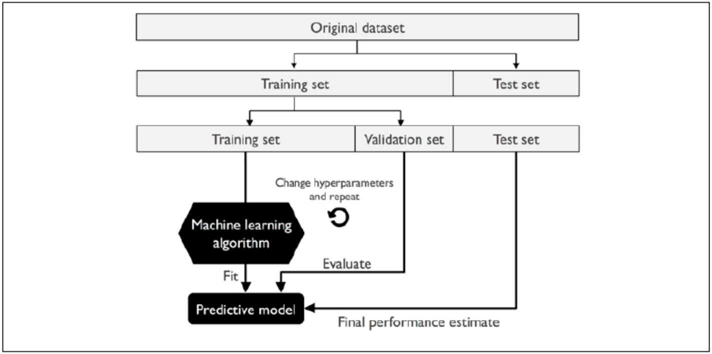
Bologna - February-April, 2025

Introduction

- Model selection is crucial for developing machine learning solutions.
- It involves choosing an algorithm and hyperparameters to optimize generalization.
- Balancing bias and variance is essential to prevent underfitting and overfitting.
- Cross-validation is a key technique for robust evaluation.

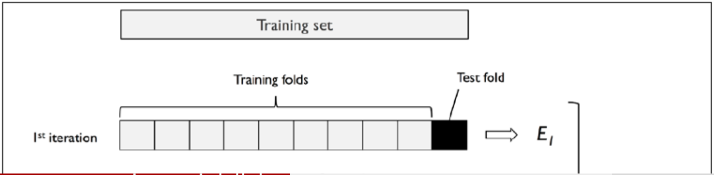
Theoretical Foundations

- Bias-variance tradeoff: high bias leads to underfitting, high variance leads to overfitting.
- K-fold cross-validation: partitions data into k subsets, ensuring fair validation.
- Each subset is used as validation once, while others are used for training.



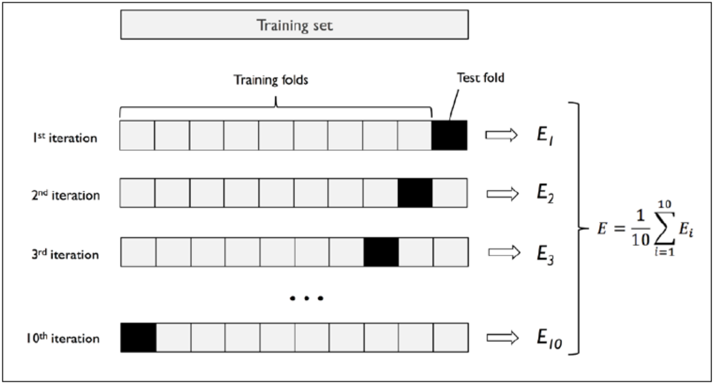
What is Cross-Validation?

- Cross-validation is a resampling technique used to evaluate machine learning models.
- The dataset is split into multiple subsets (folds), and the model is trained and validated on different folds.
- The most common method is **K-fold cross-validation**, where:
  - The dataset is divided into K equally-sized subsets.
  - The model is trained on K-1 subsets and tested on the remaining subset.
  - This process repeats K times, with each subset used as the test set once.
- The final model performance is computed as the average of the results across all folds.



Practical Model Selection Using Python

- Compare multiple models using cross-validation.
- Example: Predicting housing prices using Scikit-learn.
- `cross_val_score` function evaluates model performance using folds.



Python Code Example

- Import necessary libraries and load dataset.
- Split data into training and testing sets.
- Define multiple models (Linear Regression, Random Forest, etc.).
- Perform 5-fold cross-validation and compute R^2 scores.

```
from sklearn.model_selection import cross_val_score
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=1(
}
results = {}
for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5
    results[name] = np.mean(scores)
```

Interpretation of Results

- Compare R^2 scores to determine the best model.
- Consider interpretability, computational efficiency, and robustness.
- Example: Linear Regression vs. Random Forest.

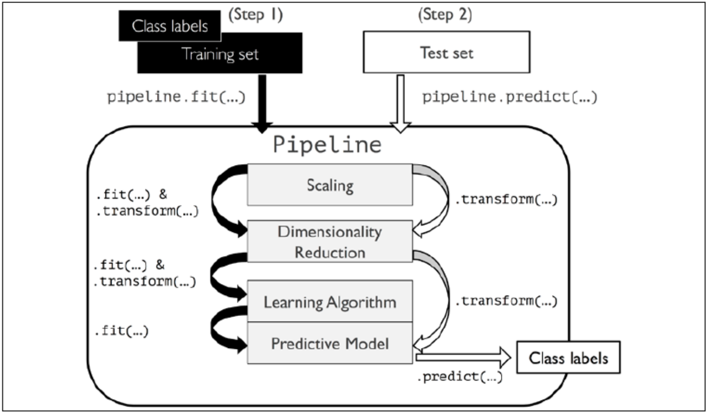
Hyperparameter Tuning

- Optimizing hyperparameters improves model performance.
- Common techniques: Grid Search, Random Search.
- Example: Tuning Random Forest with GridSearchCV.

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20]
}
grid_search = GridSearchCV(RandomForestRegressor(), param_
grid_search.fit(X_train, y_train)
```

Pipelines

- Ensures consistency across training and testing.
- Chains preprocessing steps with model training.
- Simplifies hyperparameter tuning.

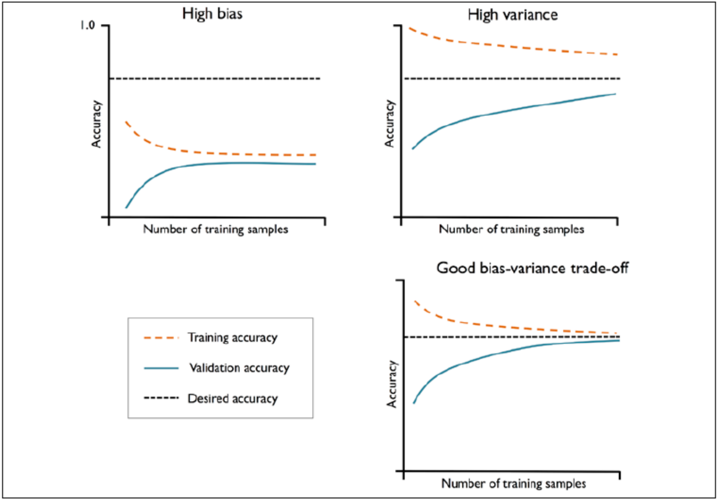


Learning Curves

- Show model performance evolution with training data size.
- Help diagnose underfitting and overfitting.
- Training and validation error curves provide key insights.

Learning Curves in Python

- Example: Linear Regression vs. Decision Tree learning curves.
- Use Scikit-learn's `learning_curve` function.



Grid Search

- Systematically evaluates hyperparameter combinations.
- Example: Random Forest tuning.
- Trade-off between computational cost and performance.

Conclusion

- Model selection and hyperparameter tuning are crucial.
- Cross-validation ensures robust evaluation.
- Use learning curves to optimize models effectively.

References

- Python Machine Learning, Third Edition by Sebastian Raschka.
- Code Repository: `https://github.com/rasbt/python-machine-learning-book-3rd-edition`