Machine Learning & Data Mining

Performance Evaluation

Kyung-Ah Sohn

Ajou University

Outline

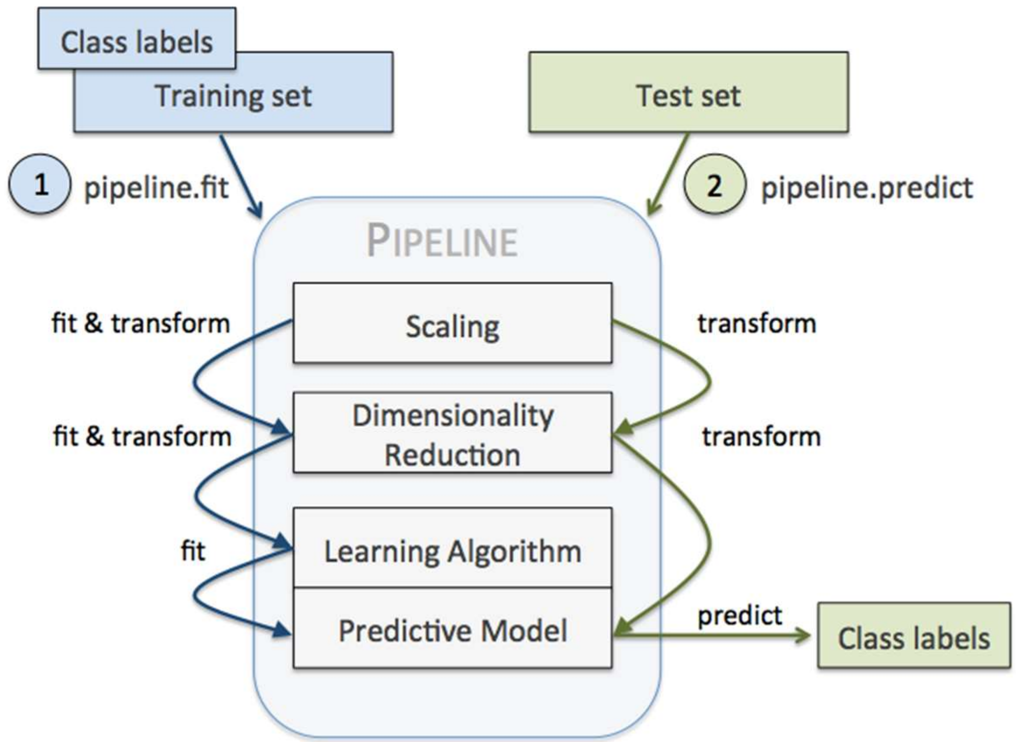
- Overfitting
- Cross-validation
- Performance measure

OVERFITTING

General ML workflows

- Input preprocessing
  - e.g. feature scaling, standardization
- Feature selection, extraction
  - PCA, embedding, … (will be discussed later)
- Model fitting, testing

Streaming workflows with pipeline



https://github.com/rasbt/python-machine-learning-book/blob/master/code/ch06/images/06_01.png

Combining transformers and estimators in a pipeline

- Instead of going through model fitting and transformation for the training and test datasets separately, we can chain the necessary objects in a pipeline
- We can think of a scikit-learn `Pipeline` as a meta-estimator or a wrapper around those individual transformers and estimators

Example:

```
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=1)))
pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
# pipe_lr.score(X_test, y_test)
```

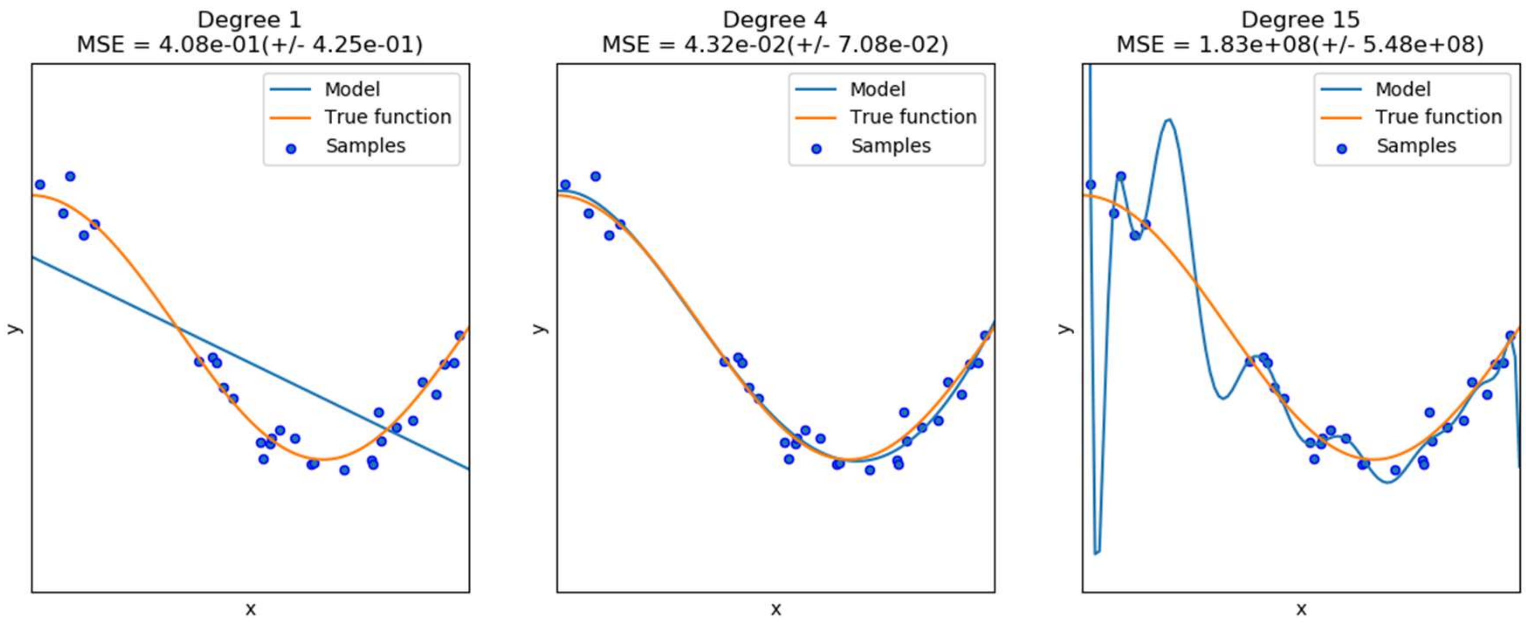
Assessing model performance

- One of the key steps in a building ML model is to estimate the performance on data that the model hasn't seen before

- Model too simple → underfitting (high bias)
- Model too complex → overfitting (high variance)

- To find an acceptable bias-variance tradeoff, we need to evaluate the model carefully.

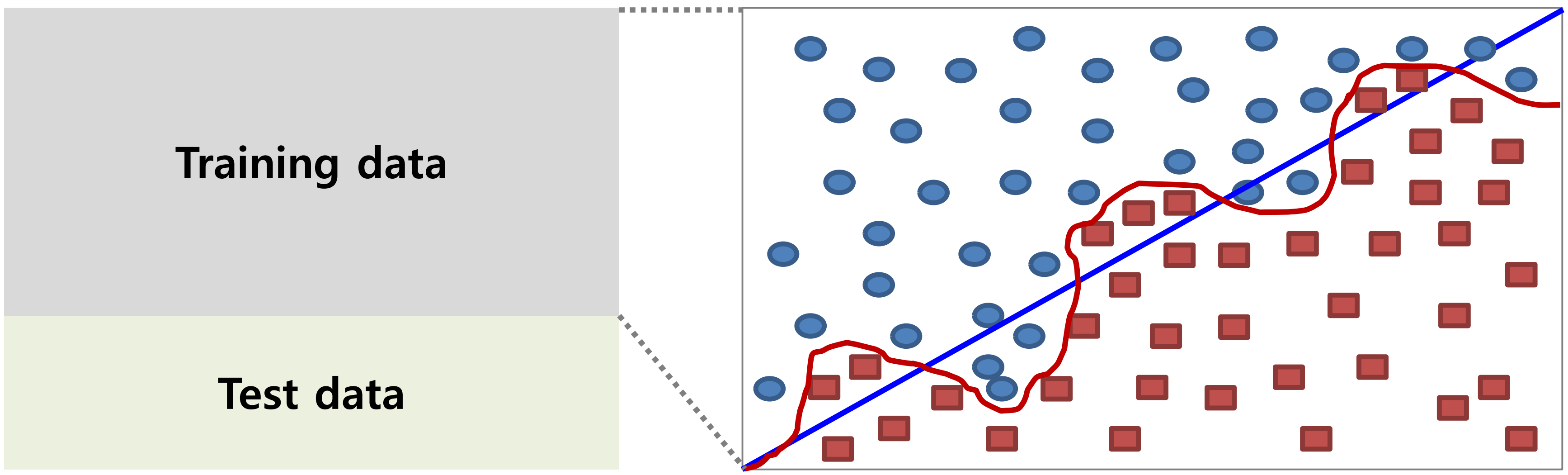
Underfitting and Overfitting



underfitting is not as prevalent as overfitting

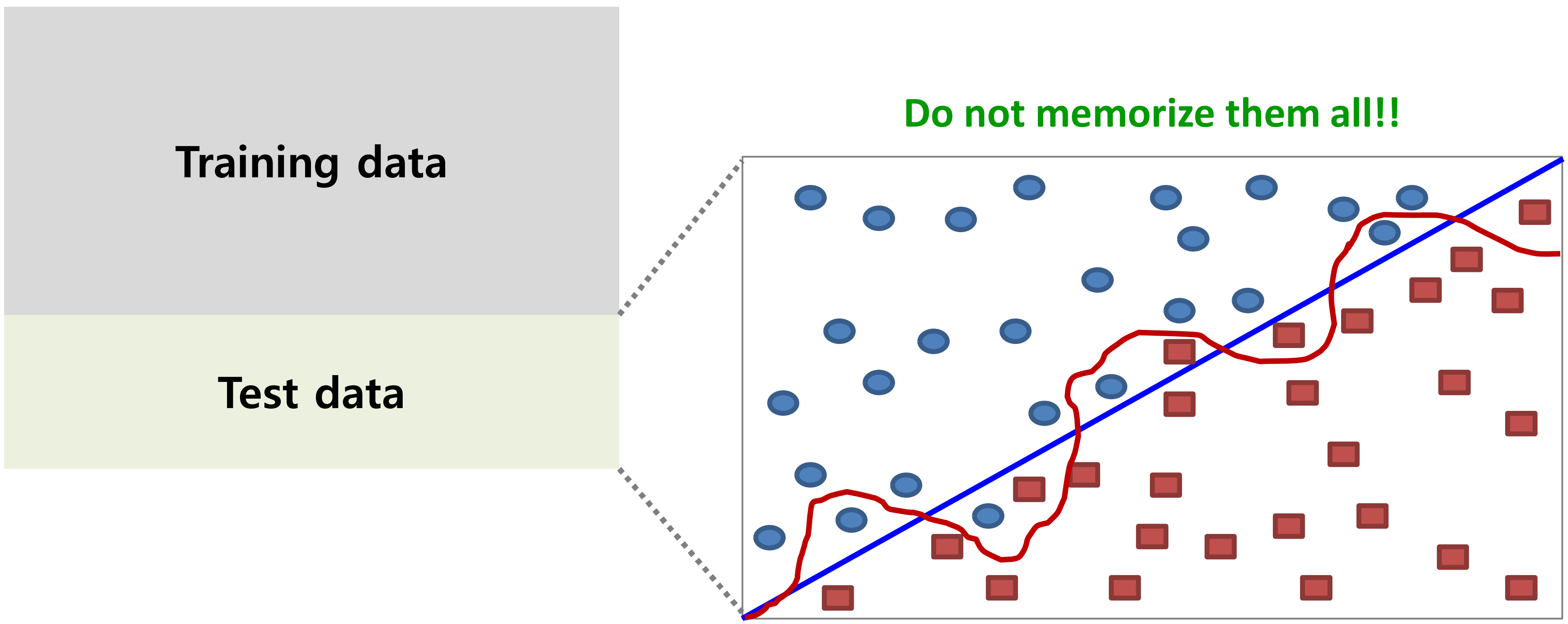
Image source: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

❖ Over-fitting for training data



Is red boundary is better than blue one?

❖ Over-fitting for training data



Validation

- Internal validation: validate your model on your current data set (cross-validation)

- External Validation: Validate your model on a completely new dataset

CROSS VALIDATION

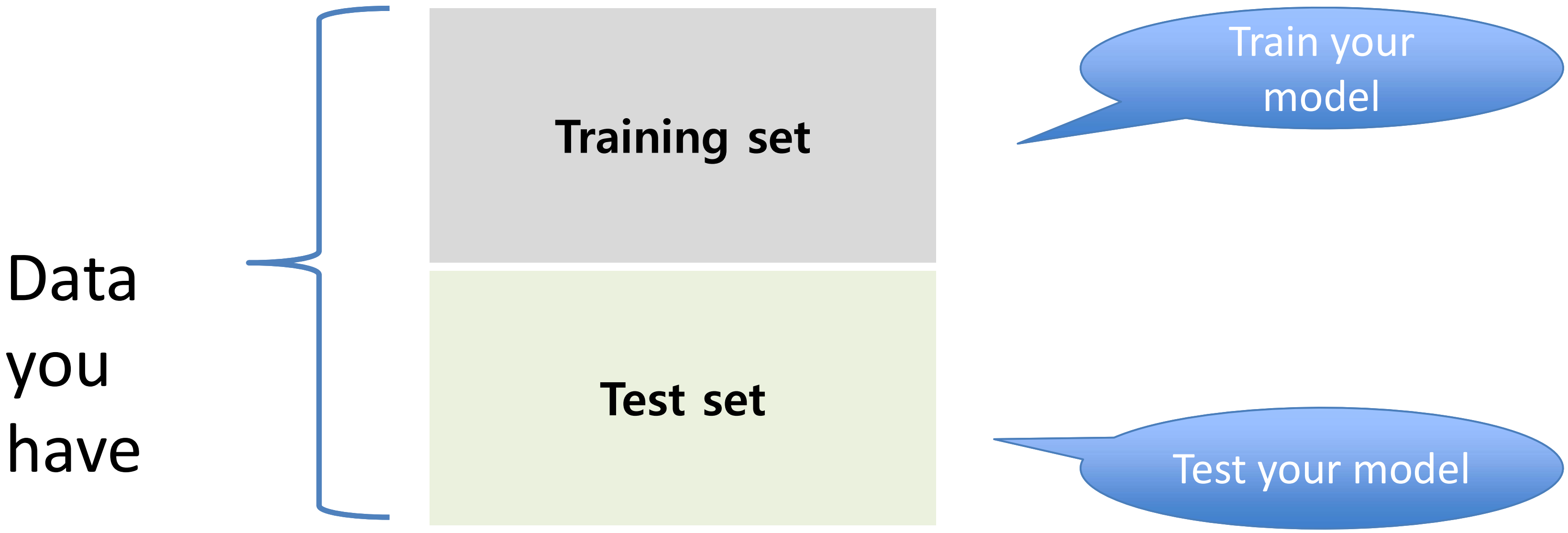
Cross-validation

- When to use?
  - To choose the best hyper-parameter setting
  - Anytime you want to prove that your model does not over-fit the training data and it will have good prediction in new datasets

Cross-validation method

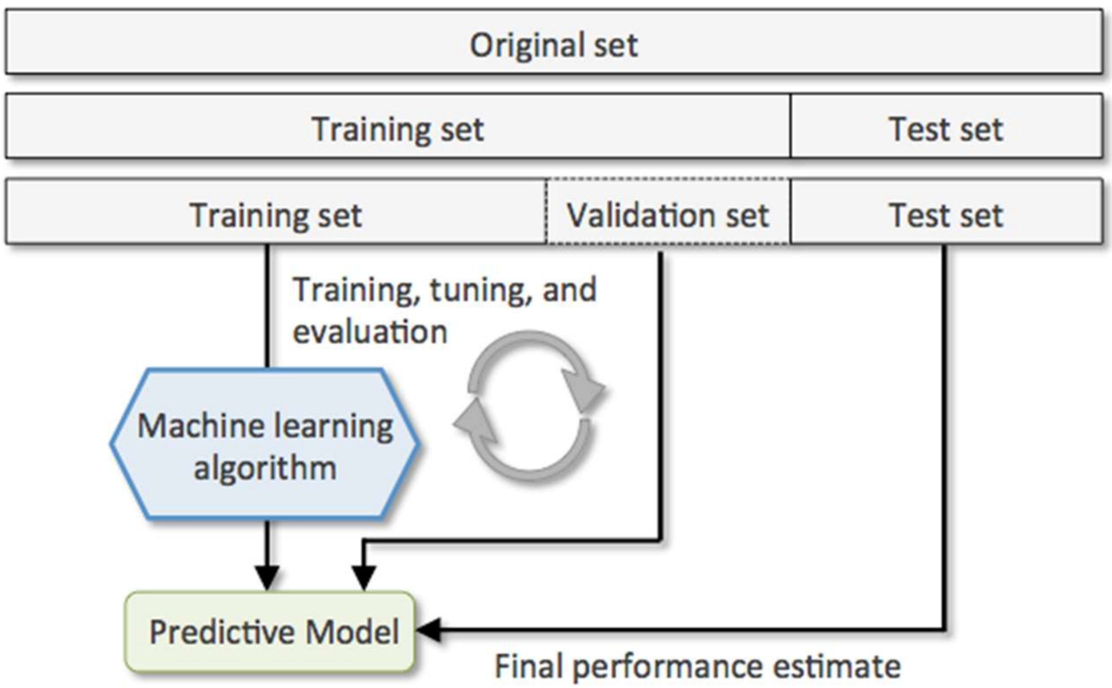
- Holdout validation
- K-fold cross validation
- Leave-one-out validation

Holdout (or Test-set) validation



- “waste” half of your data
- often you don’t have enough data to spare
- still can overfit

Train/validation/test split



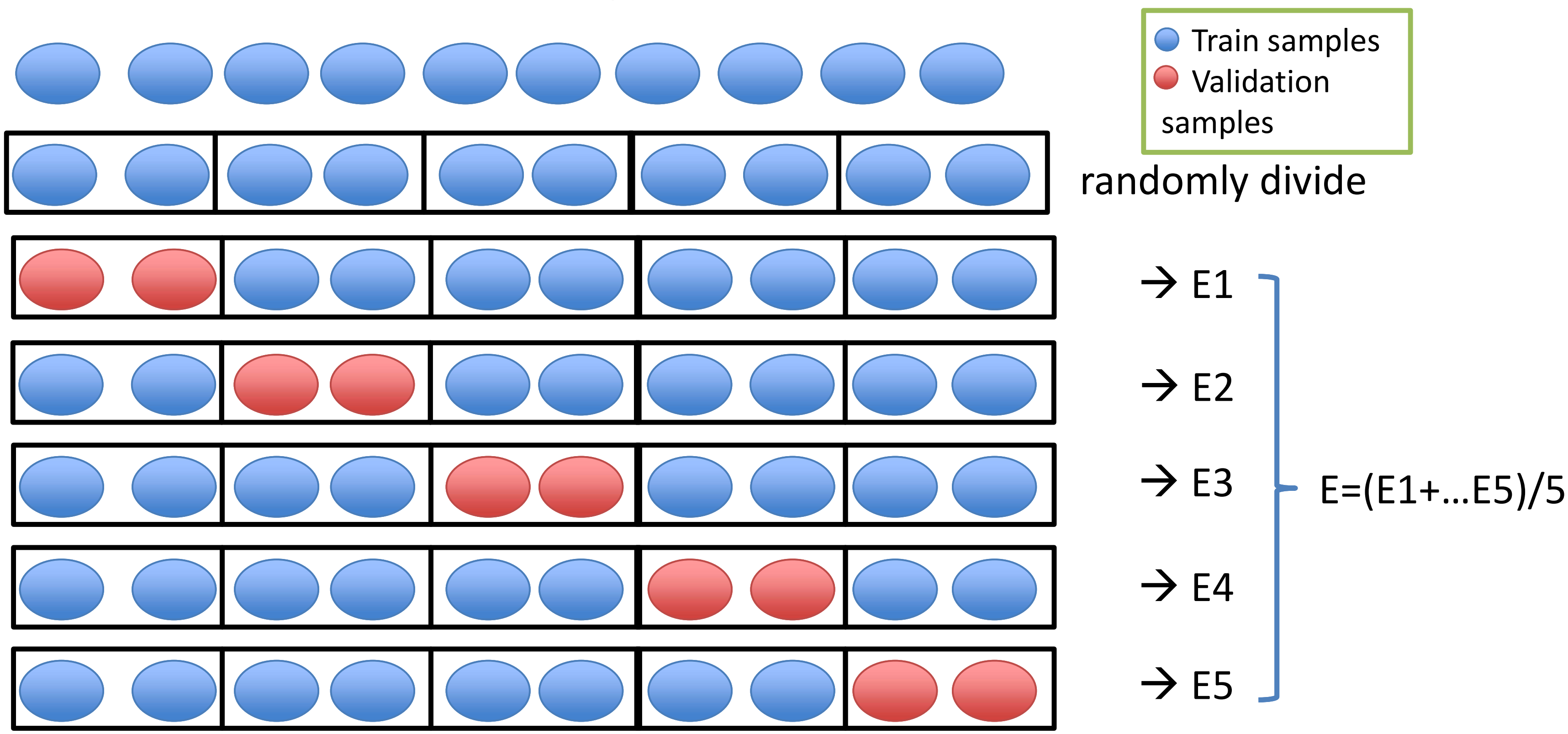
- Three-way partitioning for model selection
- Once we tune the hyper-parameters on validation set, the model's generalization performance is measured on the test dataset
- Results sensitive to how we partition the dataset

Image source: https://github.com/rasbt/python-machine-learning-book/blob/master/code/ch06/images/06_02.png

K-fold cross validation

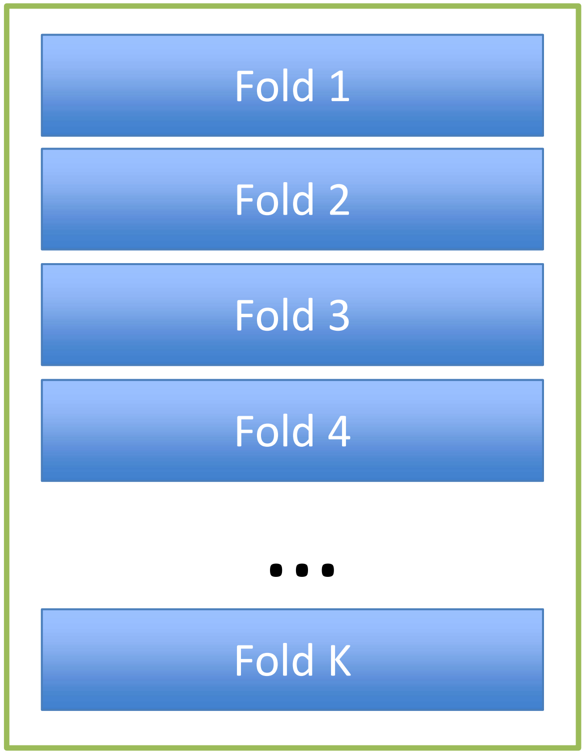
- One way to improve the holdout method
- The data set is divided into k subsets, and the holdout method is repeated k times
- Each time, one of the k subsets is used as the test set, and the remaining subsets are used in training
- We calculate the average performance across folds

Example: 5-fold CV



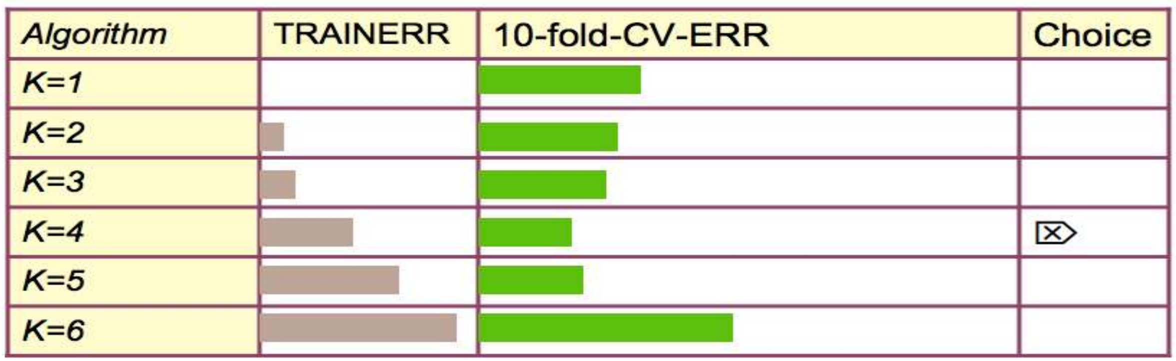
K-fold CV

- The average error rates (or accuracy measures) across all k trials is computed.
- Typically used for model tuning (finding the optimal hyperparameter values)
- Once the optimal hyperparameter values are found, we can retrain the model on the complete training dataset and obtain a final performance estimate using the independent test dataset
  - Train using more data to get more accurate and robust results



CV-based model selection

- Example: choosing “k” for k-NN
- Step 1: compute 10-fold-CV error for six different model classes



- Step 2: Whichever model class gave best CV score: train it with all the data, and that’s the predictive model you will use

K-fold CV advantages

- It matters less how the data is divided
- Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times
  - Lower-variance estimate than the holdout method
- Typical choice is 10-fold (or 5-fold) CV

Improvement

- Stratified K-fold CV (층화 K-fold CV)
  - Slight improvement over the standard K-fold CV, especially in cases of unequal class proportions
- The class label proportions are preserved in each fold

```
# pipe_lr = make_pipeline(…)
```

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

kfold = StratifiedKFold(n_splits=10, random_state=1).split(X_train, y_train)
scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test], y_train[test]) scores.append(score)
    score.append(score)
```

Or

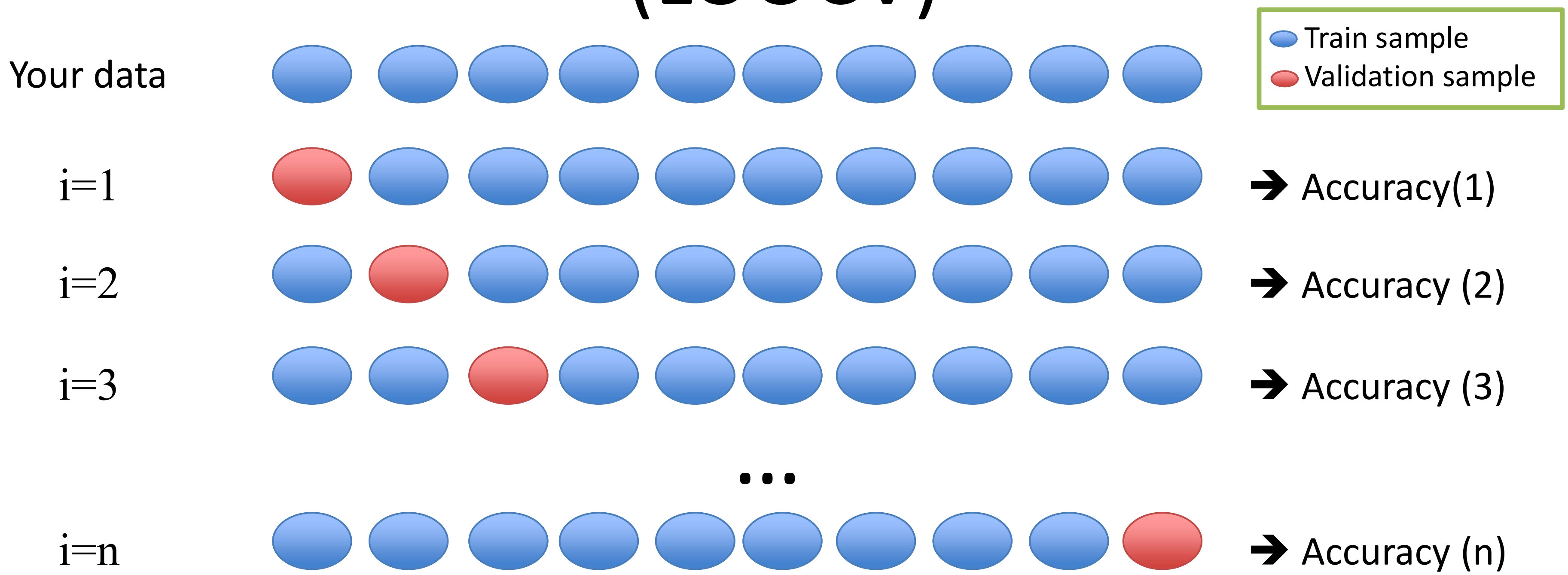
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator=pipe_lr,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)
# check np.mean(scores), np.std(scores)
```

Leave-out-out CV

- Special case of K-fold CV
- K = n (the number of training samples)
- Recommended for working with very small datasets

Leave-One-Out Cross Validation (LOOCV)



Final accuracy = (Accuracy (1)+…+Accuracy(n) / n

LOOCV

- Leave one sample out at a time
- Learn the model on the remaining training data
- Test on the held out data point

- Summarize the performance of each run

LOOCV

- The evaluation result is good, but it is very expensive to compute
  - n runs of the learning algorithm if you have n data points

    n x (running time of the algorithm)

Which kind of Cross Validation?

	Downside	Upside
Holdout	Variance: unreliable estimate of future performance	Cheap
LOO	Expensive. Has some weird behavior	Doesn't waste data
10-fold	Wastes 10% of the data. 10 times more expensive than test set	Only wastes 10%. Only 10 times more expensive instead of R times.
3-fold	Wastier than 10-fold. Expensivier than test set	Slightly better than test-set
n-fold	Identical to Leave-one-out	

http://www.autonlab.org/tutorials/

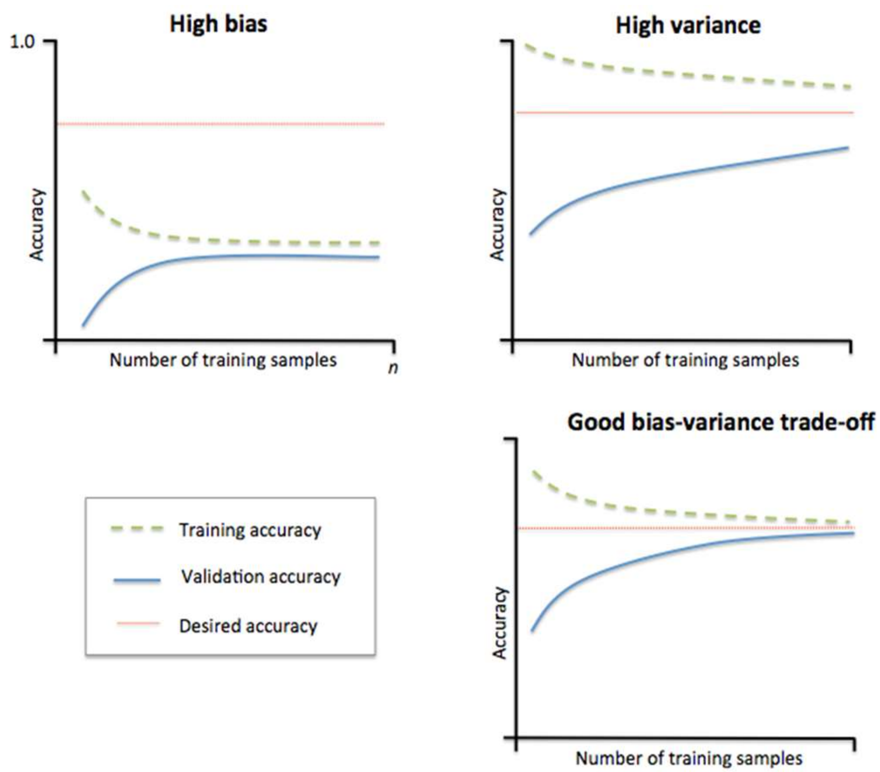
Cross-validation is useful

- Preventing over-fitting
- Comparing different algorithms
- Choosing the optimal hyper-parameters
- For any supervised learning approaches

DEBUGGING WITH LEARNING AND VALIDATION CURVES

Diagnosing bias and variance problems

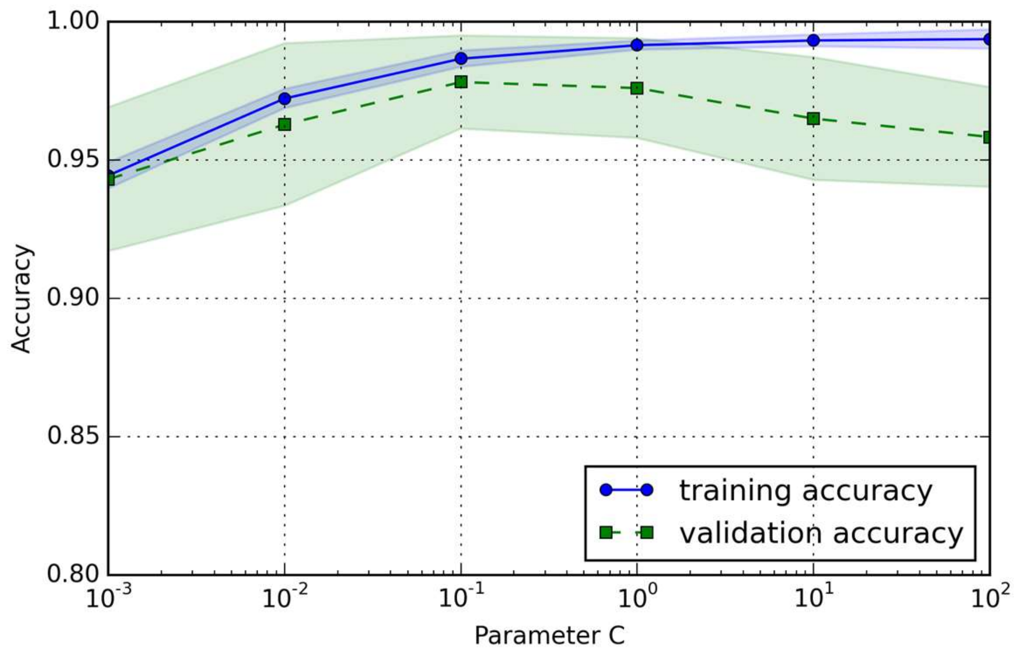
underfitting
- Increase model complexity (add more parameters)



overfitting
- Collect more data
- Reduce model complexity
- Increase model regularization

Image source: https://github.com/rasbt/python-machine-learning-book/tree/master/code/ch06/images

Addressing over- and underfitting with validation curves



- Optimal parameter choice?

Image source: https://github.com/rasbt/python-machine-learning-book/tree/master/code/ch06/images

Grid search

- A brute-force exhaustive search method for tuning hyperparameters

- Specify a list of different values and evaluate the model performance for each combination

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

pipe_svc = make_pipeline(StandardScaler(),
                         SVC(random_state=1))
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'svc__C': param_range,
              'svc__kernel': ['linear']},
             {'svc__C': param_range,
              'svc__gamma': param_range,
              'svc__kernel': ['rbf']}]
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)
gs = gs.fit(X_train, y_train)

print(gs.best_params_)
```

Algorithm selection with nested cross-validation

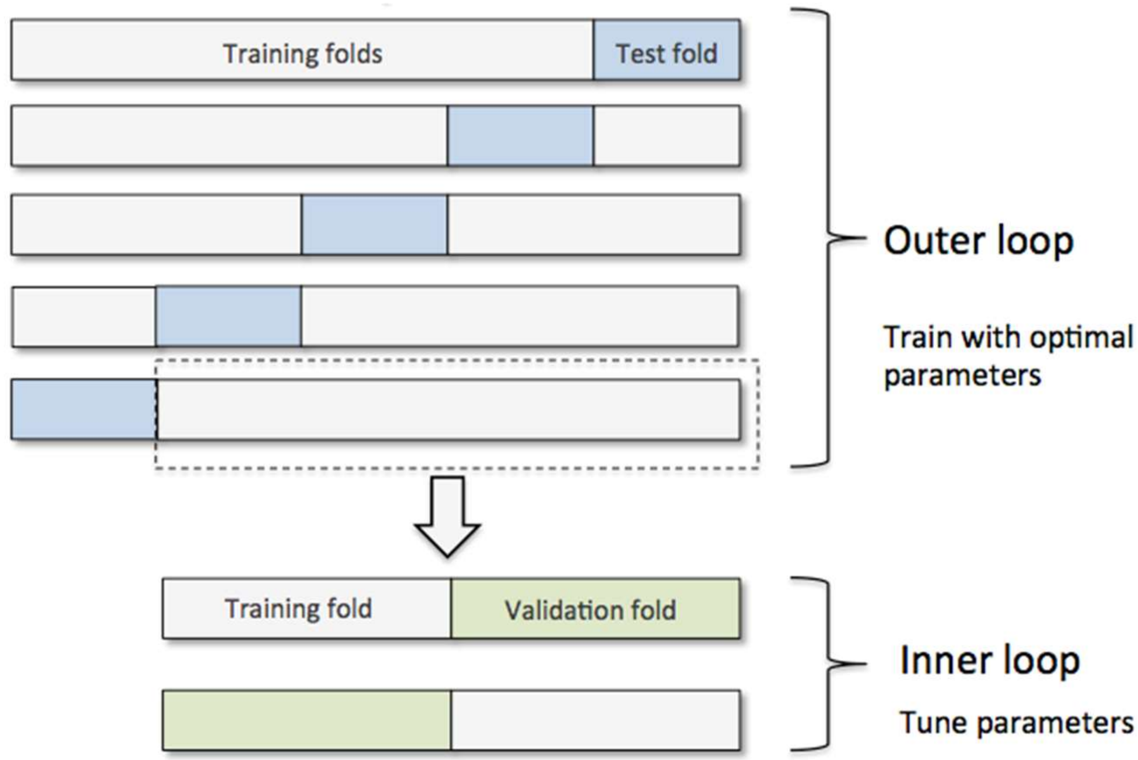


Image source: https://github.com/rasbt/python-machine-learning-book/tree/master/code/ch06/images

Algorithm selection with nested cross-validation

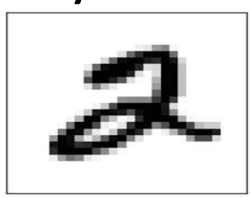
- Useful when comparing different algorithms
- Outer loop to split data into training and test folds
- Inner loop for tuning the parameter using k-fold cv

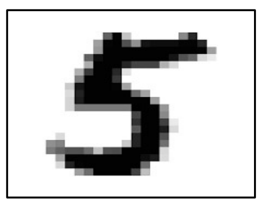
```
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy’,
                  cv=2)

score = cross_val_score(gs, X_train, y_train, scoring=‘accuracy’, cv=5)
# check np.mean(scores), np.std(scores)
# e.g. 0.974 +/- 0.015
```

Example: MNIST handwritten digits

Consider Binary classification





'2' or 'not 2'?

Performance evaluation metrics

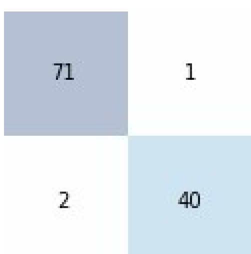
- Confusion metrics

		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Example

```python
from sklearn.metrics import confusion_matrix
pipe_svc.fit(X_train, y_train)
y_pred = pipe_svc.predict(X_test)
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)
```

```
[[71  1]
 [ 2 40]]
```



- Accuracy =
- Error =
- Other performance metrics?

Performance metrics

- Accuracy (error) may not be enough to represent the true performance of a classification algorithm

- Precision/Recall, F1 score
- Sensitivity/specificity
- ROC curve, AUC