IT496: Introduction to Data Mining

Lecture 09

Evaluation - I

[Schemes for Data Split and Handling Bias-Variance]

Arpit Rana

17th August 2023

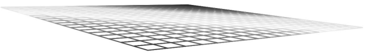
Learning = Representation + Evaluation + Optimization

Representation ✔

Choosing a representation of the learner: the *hypotheses space* or *the model class* — the set of models that it can possibly learn.

$$h_\beta(X) = \beta_0 + \beta_1 X_1 \;+\; \beta_2 X_2 + .\;.\;. + \beta_m X_m$$

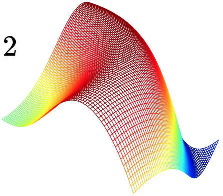
$$= \sum_{i=1}^{m} \beta_i X_i$$



Evaluation

Choosing an evaluation function (also called objective function, utility function, loss function, or scoring function) is needed to distinguish good classifiers from bad ones.

$$J(\beta) = \quad \sum_{i=1}^{m} (h_\beta(X_i) - y_i)^2$$

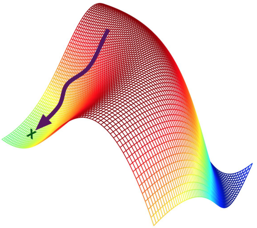


Optimization

Choosing a method to search among the models in the hypothesis space for the highest-scoring one.

$$min\ J(\beta)$$

��

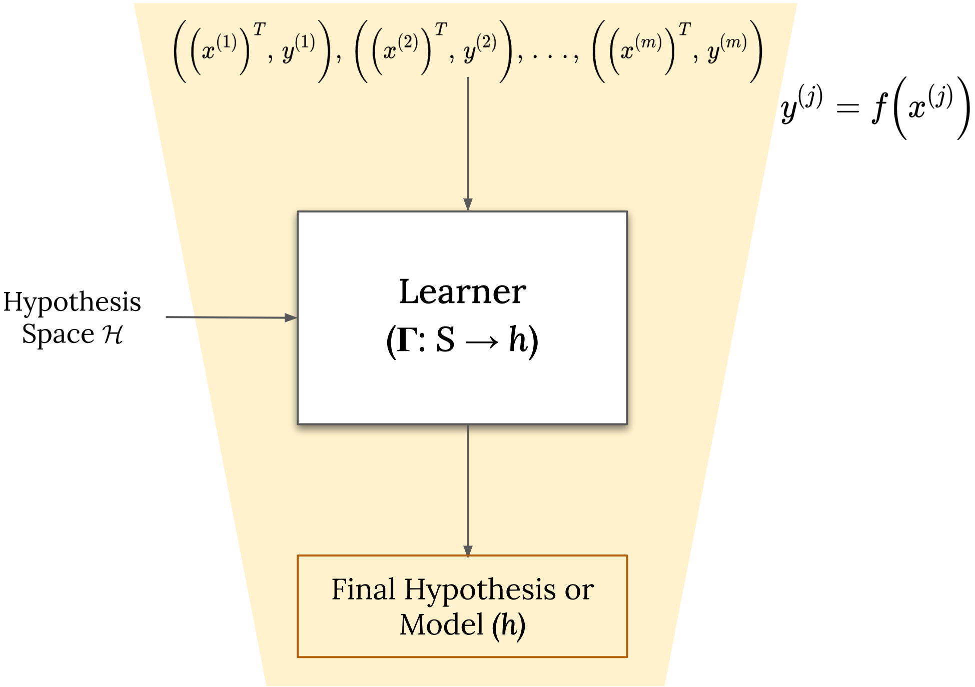


Experimental Evaluation of Learning Algorithms

The overall *objective* of the Learning Algorithm is to find a *hypothesis* that -

- is *consistent* (i.e., fits the training data), but more importantly,
- *generalizes well* for previously unseen data.

Experimental Evaluation defines ways to Measure the **Generalizability** of a Learning Algorithm.



$$\left(\left(x^{(1)}\right)^T, y^{(1)}\right), \left(\left(x^{(2)}\right)^T, y^{(2)}\right), \ldots, \left(\left(x^{(m)}\right)^T, y^{(m)}\right)$$

$$y^{(j)} = f\left(x^{(j)}\right)$$

Hypothesis Space $\mathcal{H}$

Learner

(Γ: S → *h*)

Final Hypothesis or Model (*h*)

Given a *representation*, *data, and a bias*, the learning algorithm returns a final hypothesis.

Experimental Evaluation of Learning Algorithms

Sample Error

The *sample error* of hypothesis *h* with respect to the target function *f* and data sample S is:

$$error_S(h) = \frac{1}{n} \sum_{x \in S} \delta(h(x), f(x))$$

It is *impossible* to asses ***true error***, so we try to estimate it using ***sample error***.

True Error

The *true error* of hypothesis *h* with respect to the target function *f* and the distribution *D* is the probability that *h* will misclassify an instance drawn at random according to *D*:

$$error_D(h) = P_{x \in D}[h(x) \neq f(x)]$$

Generalizing to Unseen Data

The error on the *training set* is called the ***training error*** (a.k.a. ***resubstitution error*** and ***in-sample error***).

- The training error is not, in general a good indicator of performance on unseen data. It's often too *optimistic*.
- Why?

Generalizing to Unseen Data

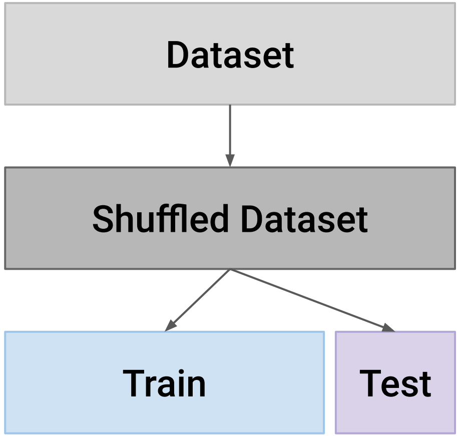
To predict future performance, we need to measure error on an *independent dataset*:

- We want a dataset that has *played no part in creating the model*.
- This second dataset is called the *test set*.
- The error on the test set is called the ***test error*** (a.k.a. ***out-of-sample error*** and ***extra-sample error***).

Given a sample data S, there are methodologies to better approximate the true error of the model.

Holdout Method

- Shuffle the dataset and partition it into two *disjoint sets*:
  - ***training set*** (e.g., 80% of the full dataset); and
  - ***test set*** (the rest of the full dataset).
- Train the estimator on the training set.
- Test the model (evaluate the predictions) on the test set.



It is essential that the test set is not used in any way to create the model. Don't even look at it!

- 'Cheating' is called *leakage*.
- 'Cheating' is one cause of *overfitting*

Holdout Method: Class Exercise

Standardization, as we know, is about scaling the data. It requires calculation of the *mean* and *standard deviation*.

When should the mean and standard deviation be calculated? And Why?

(a) before splitting, on the entire dataset, or

(b) after splitting, on just the training set, or

(c) after splitting, on just the test set, or

(d) after splitting, on the training and test sets separately,

What to do when the model is deployed?

Facts about Holdout Method

- The disadvantages of this method are:
  - Results can vary quite a lot across different runs.
  - Informally, you might get lucky — or unlucky

    i.e., in any one split, the data used for training or testing might not be *representative*.

- We are training on only a subset of the available dataset, perhaps as little as 50% of it. From so little data, we may learn a worse model and so our error measurement may be ***pessimistic***.

- In practice, we only use the holdout method when we have a very large dataset. The size of the dataset mitigates the above problems.

- When we have a smaller dataset, we use a *resampling* method:
  - The examples get re-used for training and testing.

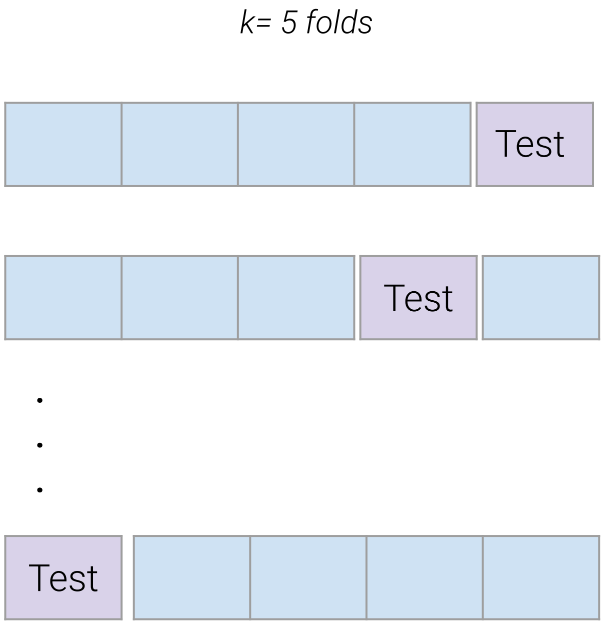
K-fold Cross-Validation Method

The most-used *resampling* method is *k*-fold cross-validation:

- Shuffle the dataset and partition it into *k* disjoint subsets of equal size.
  - Each of the partitions is called a *fold*.
  - Typically, *k*=10, so you have 10 folds.
- You take each fold in turn and use it as the test set, training the learner on the remaining folds.
- Clearly, you can do this *k* times, so that each fold gets 'a turn' at being the test set.
  - By this method, each example is used exactly once for testing, and *k*-1 times for training.

K-fold Cross-Validation: Pseudocode

```
● Shuffle the dataset $D$ and partition it into $k$ disjoint equal-sized subsets, $D_1, \dots, D_k$
● for $i = 1$ to $k$:
    ○ train on $D \setminus D_i$
    ○ make predictions for $D_i$
    ○ measure error (e.g. MAE)
● Report the mean of the errors
```



Facts about *K*-fold Cross-Validation

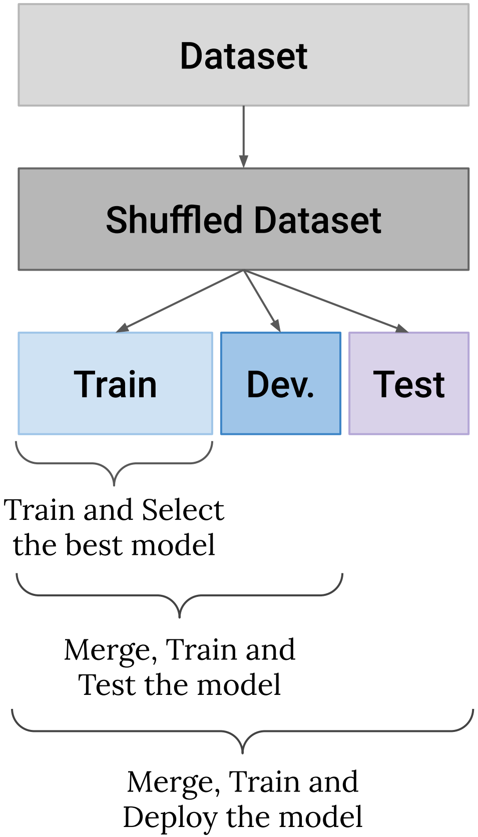
- The disadvantages of this method are:
  - The number of folds is constrained by the size of the dataset and the desire sometimes on the part of statisticians to have folds of at least 30 examples.
  - It can be costly to train the learning algorithm k times.
  - There may still be some variability in the results due to 'lucky'/'unlucky' splits.
- The extreme is $k = n$, also known as *leave-one-out cross-validation or* LOOCV.

Nested K-fold Cross-Validation Method

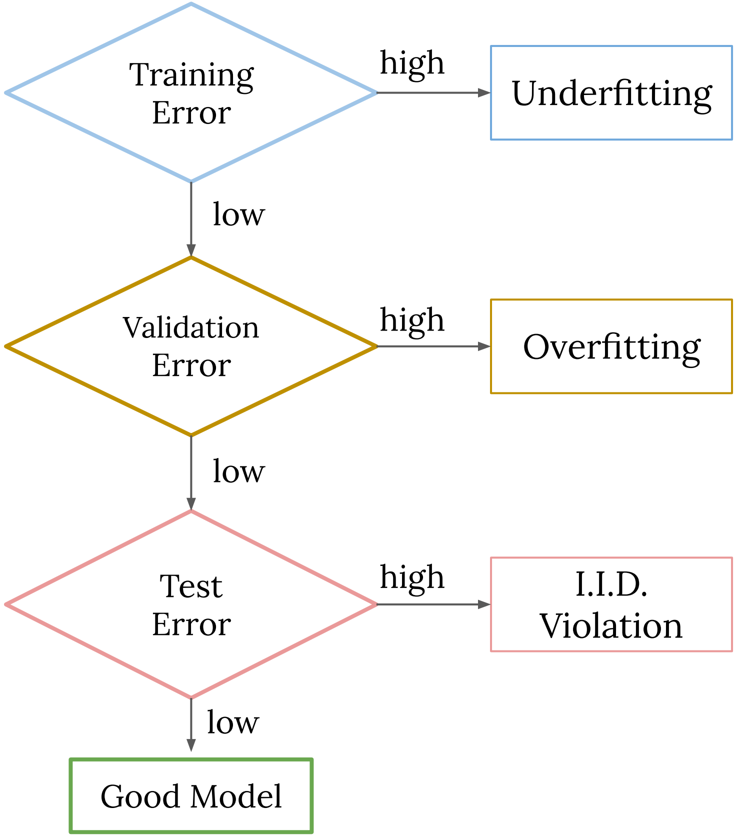
In case of *hyperparameter* (parameters of the model class, not of the individual model) or *parameter tuning*, we partition the whole dataset into *three* disjoint sets:

- A **training set** to train candidate models.
- A **validation set**, (a.k.a. a *development set* or *dev set*) to evaluate the candidate models and choose the best one.
- A **test set** to do a final unbiased evaluation of the best model.

K-fold Cross-Validation can be applied to validation set (inner CV) and test set (outer CV) in a nested way.


Dataset
Shuffled Dataset
Train
Dev.
Test
Train and Select the best model
Merge, Train and Test the model
Merge, Train and Deploy the model

Model's Performance



Training Error

high

Underfitting

low

Validation Error

high

Overfitting

low

Test Error

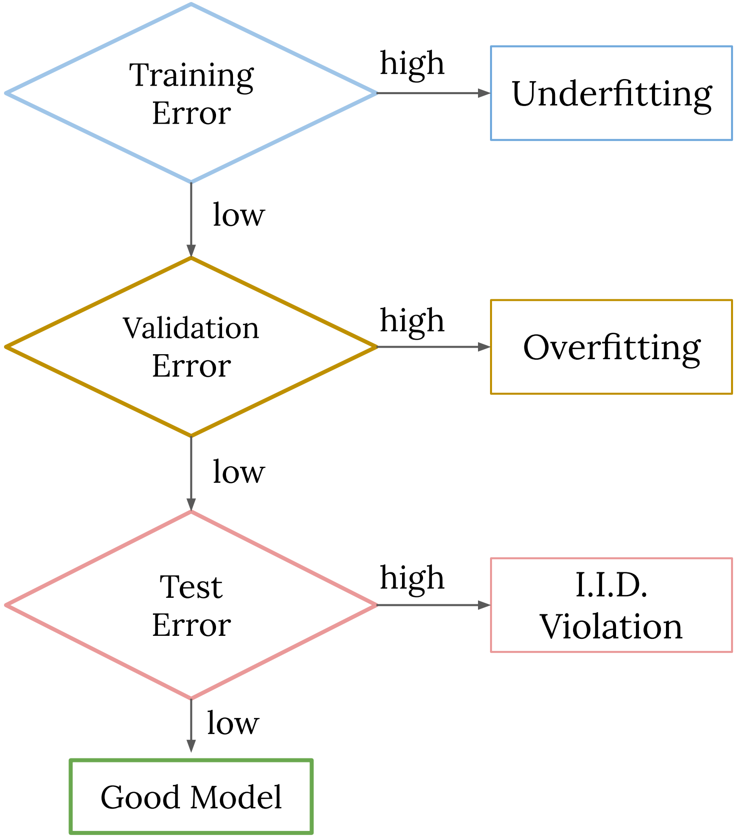
high

I.I.D. Violation

low

Good Model

Model's Performance



Underfitting	Overfitting
Need More Complex Model	Need Simpler Model
Need Less Regularization	Need More Regularization
Need More Features	Remove Extra Features
More Data Doesn't Work	Need More Data

Next lecture

Evaluation - II

18th August 2023