Resampling: Cross Validation and Bootstrap
Ch5 ISLP; Ch2 IMLP

Zhongjian Lin
University of Georgia

September 22, 2024

Resampling

Resampling methods involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model.

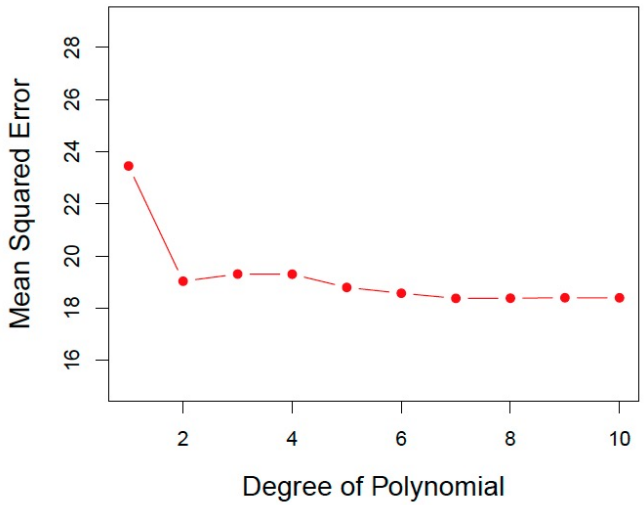
- Cross-Validation: can be used to estimate the test error associated with a given statistical learning method in order to evaluate its performance, or to select the appropriate level of flexibility.
- Bootstrap: is used in several contexts, most commonly model to provide a measure of accuracy of a parameter estimate or of a given selection statistical learning method.

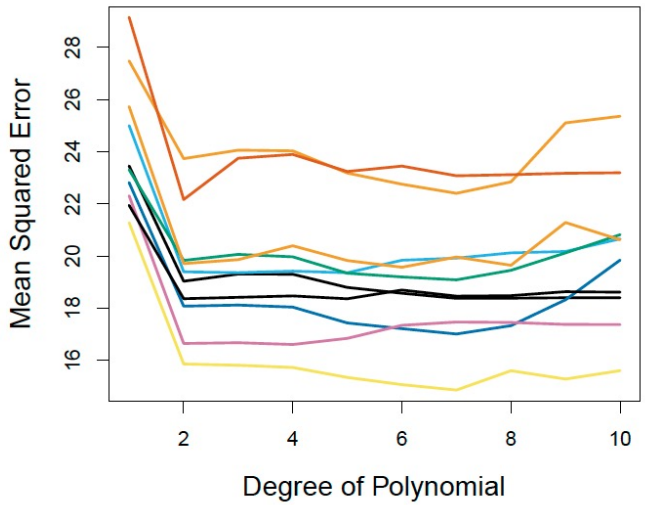
Cross Validation

The test error is the average error that results from using a statistical learning method to predict the response on a new observation- that is, a measurement that was not used in training the method. Given a data set, the use of a particular statistical learning method is warranted if it results in a low test error. The test error can be easily calculated if a designated test set is available. Unfortunately, this is usually not the case. In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training. In the absence of a very large designated test set that can be used to directly estimate the test error rate, a number of techniques can be used to estimate this quantity using the available training data. We consider a class of methods that estimate the test error rate by holding out a subset of the training observations from the fitting process, and then applying the statistical learning method to those held out observations.

The Validation Set Approach

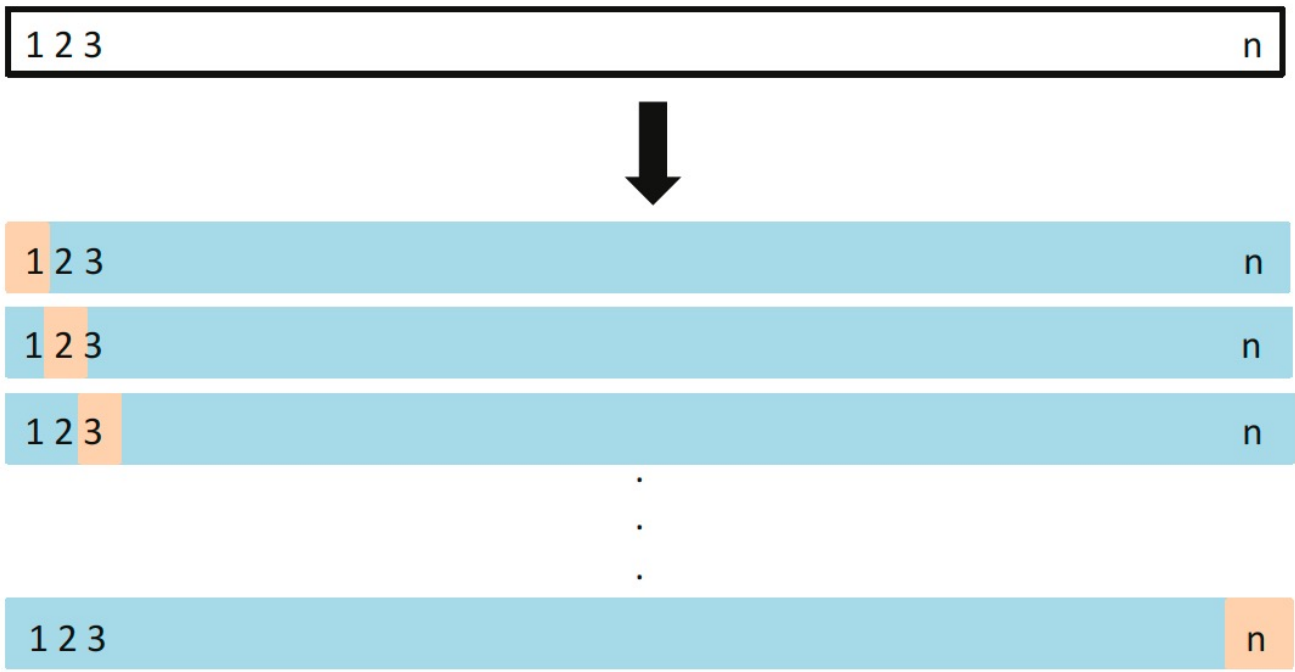
It involves randomly dividing the available set of observations into two parts, a training set and a validation set or hold-out set. The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set. The resulting validation set error rate—typically assessed using MSE/R^2 in the case of a quantitative response—provides an estimate of the test error rate.





Leave-One-Out Cross Validation

Leave-one-out cross-validation (LOOCV) is closely related to the validation, but it attempts to address that method's drawbacks.



$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Leave-One-Out cross-validation

- We repeatedly fit the statistical learning method using training sets that contain $n - 1$ observations, almost as many as are in the entire data set. Consequently, the LOOCV approach tends not to overestimate the test error rate as much as the validation set approach does.
- Second, in contrast to the validation approach which will yield different results when applied repeatedly due to randomness in the training/validation set splits, performing LOOCV multiple times will always yield the same results: there is no randomness in the training/validation set splits.

```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
print("number of cv iterations: ", len(scores))
print("mean accuracy: ", scores.mean())
number of cv iterations: 150
mean accuracy: 0.953333333333
```

Models

- Nearest neighbors: for small datasets, good as a baseline, easy to explain.
- Linear models: Go-to as a first algorithm to try, good for very large datasets, good for very high-dimensional data.
- Decision trees: Very fast, don't need scaling of the data, can be visualized and easily explained.
- Random forests: Nearly always perform better than a single decision tree, very robust and powerful. Don't need scaling of data. Not good for very highdimensional sparse data.
- Gradient Boosted Decision Trees: Often slightly more accurate than random forest. Slower to train but faster to predict than random forest, and smaller in memory. Need more parameter tuning than random forest.
- Support Vector Machines: Powerful for medium-sized datasets of features with similar meaning. Needs scaling of data, sensitive to parameters.
- Neural Networks: Can build very complex models, in particular for large datasets. Sensitive to scaling of the data, and to the choice of parameters. Large models need a long time to train.

Model Selection

- When working with a new dataset, it is in general a good idea to start with a simple model, such as a linear model, naive Bayes or nearest neighbors and see how far you can get. After understanding more about the data, you can consider moving to an algorithm that can build more complex models, such as random forests, gradient boosting, SVMs or neural networks.
- Playing around with the algorithms on different datasets will give you a better feel on how long they need to train, how easy it is to analyze the model, and how sensitive they are to the representation of the data.
- While we analyzed the consequences of different parameter settings for the algorithms we investigated, building a model that actually generalizes well to new data in production is a bit trickier than that. We will see how to properly adjust parameters, and how to find good parameters automatically.

Model Selection

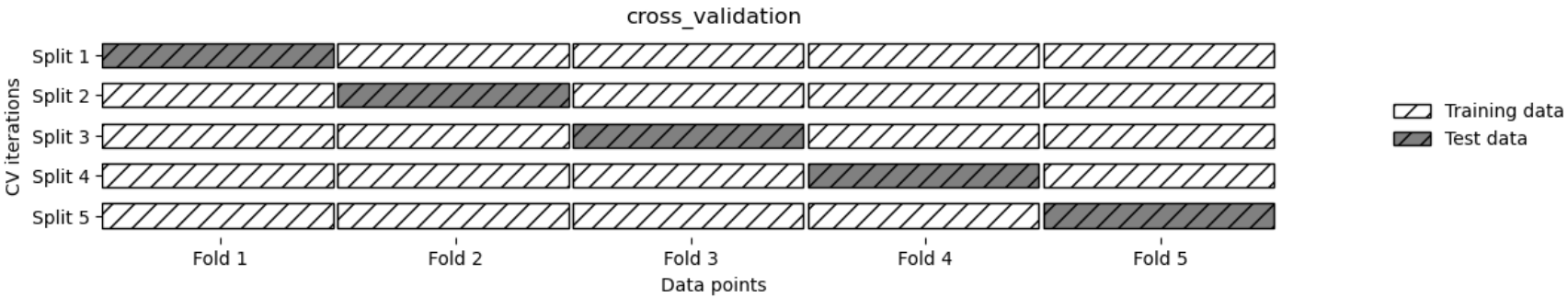
- Having discussed the fundamentals of supervised learning, and having explored a variety of machine learning algorithms, we will now dive more deeply into evaluating models and selecting parameters.
- To evaluate our supervised models, so far we have split our data set in to a training set and a test set using the train_test_split function, built a model on the training set calling the fit method, and evaluated it on the test set using the score method, which, for classification, computes the fraction of correctly classified samples.
- The reason we split our data into training and test sets is that we are interested in measuring how well our model generalizes to new, unseen data. We are not interested in how well our model fit the training set, but rather, how well it can make predictions for data that was not observed during training.

Cross Validation

Cross valiation is a more robust way to assess generalization performance than a single split of the data into a training and a test set. We will also discuss methods to evaluate classification and regression performance that go beyond the default measures of accuracy and R^2 provided by the score method.
The most commonly used version of cross-validation is k-fold cross-validation, where k is a user specified number, usually five or ten.

k-fold cross-validation

1. When performing five-fold crossvalidation, the data is first partitioned into five parts of (approximately) equal size, called folds.
2. Next, a sequence of models is trained. The first model is trained using the first fold as the test set, and the remaining folds 2-5 as the training set.
3. Then another model is build, this time using fold 2 as the test set, and the data in folds 1, 3, 4 and 5 as the training set.
4. This process is repeated using the folds 3, 4 and 5 as test sets. For each of these five splits of the data into training and test set, we computed the accuracy. In the end, we have collected five accuracy values.



Cross Validation

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
iris = load_iris()
logreg = LogisticRegression()
scores = cross_val_score(logreg, iris.data, iris.target)
print("cross-validation scores: ", scores)
cross-validation scores:  [0.96666667 1.         0.93333333 0.96666667 1.        ]
scores = cross_val_score(logreg, iris.data, iris.target, cv=3)
scores
array([0.98, 0.96, 0.98])
```

A common way to summarize the cross-validation accuracy is to compute the mean: scores.mean()

Benefits of Cross Validation

There are several benefits of using cross-validation instead of a single split into a training and test set.

- First, remember that train_test_split performs a random split of the data. Imagine that we are "lucky" when randomly splitting the data, and all examples that are hard to classify are in the training set. Then, the test set will only contain "easy" examples, and our test set accuracy will be unrealistically high. In cross-validation, each example will be in the training set exactly once. Therefore the model needs to generalize well to all of the samples for all of the cross-validation scores to be high.
- Having multiple splits of the data also provides some information about how sensitive our model is to the selection of the training dataset. Looking at the scores for the iris dataset above, we see accuracies between 90% and 100%. This is quite a range, and provides us with an idea about how the model might perform in the worst case and the best case scenarios when applied to new data.
- Another benefit of cross-validation as compared to using a single split of the data is that we use our data more effectively. When using train_test_split, we usually use 75% of the data for training. When using five-fold cross-validation, in each iteration we can use 4/5 of the data to fit the model. More data will usually result in more accurate models.

Stratified k-fold Cross Validation

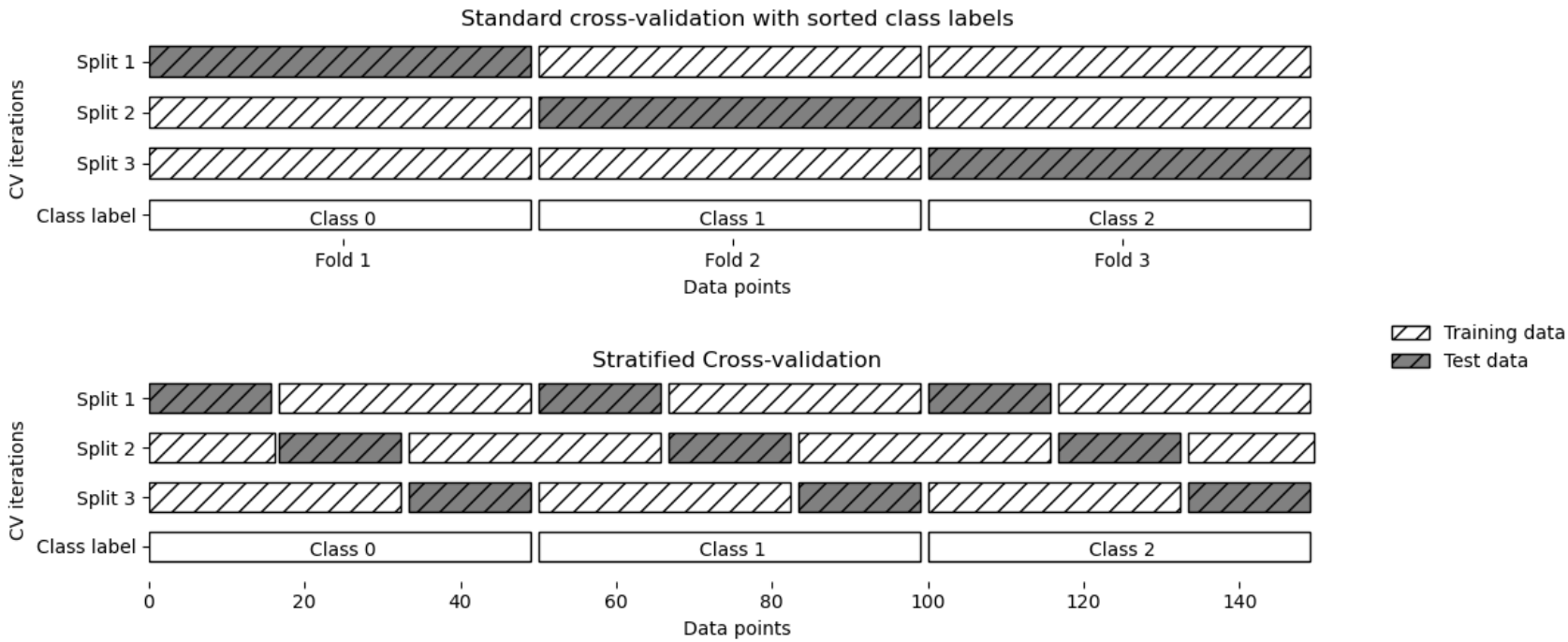
Splitting the dataset into k-folds by starting with the first 1/k-th part of the data as described above might not always be a good idea. Let's have a look at the iris dataset for example:

```
from sklearn.datasets import load_iris
iris = load_iris()
print(iris.target)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

As you can see above, the first third of the data is the class 0, the second third is the class 1, and the last third is class 2. Imagine doing three-fold cross-validation on this dataset. The first fold would be only class 0, so in the first split of the data, the test set would be only class zero, and the training set would be only class 1 and 2. As the classes in training and test set would be different for all three splits, the threefold cross-validation accuracy would be zero on this dataset. That is not very helpful, as we can do much better than 0% accuracy on iris.

Stratified k-fold Cross Validation

As the simple k-fold strategy fails here, scikit-learn does not use k-fold for classification, but rather stratiied k-fold cross-validation. In stratified cross-validation, we split the data such that the proportions between classes are the same in each fold as they are in the whole dataset, as illustrated in Figure stratified_kfold.



It is always a good idea to use stratified k-fold cross-validation instead of k-fold crossvalidation to evaluate a classifier, because it results in more reliable estimates of generalization performance.

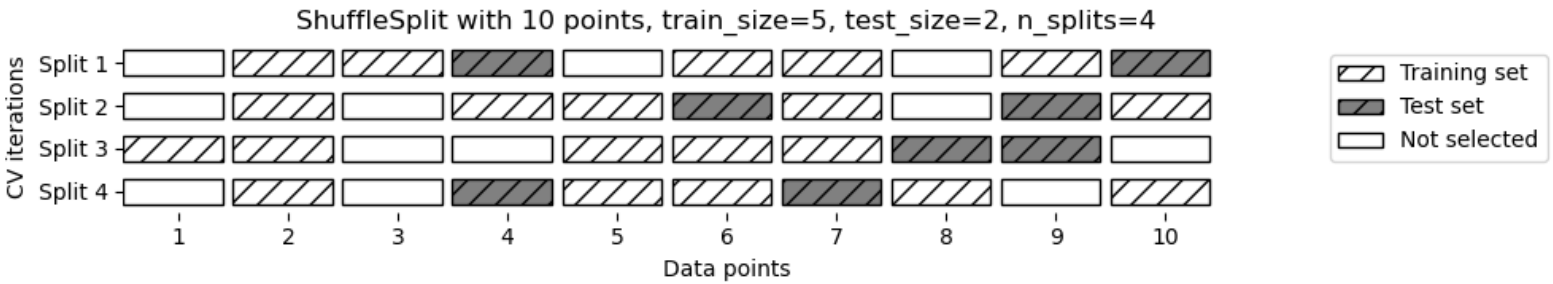
More control over cross-validation

Another way to resolve this problem instead of stratifying the folds is to shuffle the data, to remove the ordering of the samples by label. We can do that setting the *shuffle* parameter of KFold to True. If we shuffle the data, we also need to fix the *random_state* to get a reproducible shuffling. Otherwise, each run of cross_val_score would yield a different result, as each time a different split would be used

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=3, shuffle=True, random_state=0)
cross_val_score(logreg, iris.data, iris.target, cv=kfold)
array([ 0.9 , 0.96, 0.96])
```

Shuffle-Split cross-validation

Another, very flexible strategy for cross validation is shuffle-split cross-validation. In shuffle-split cross-validation, each split samples train_size many points for the training set, and test_size many (disjoint) point for the test set. This splitting is repeated n_splits many times.

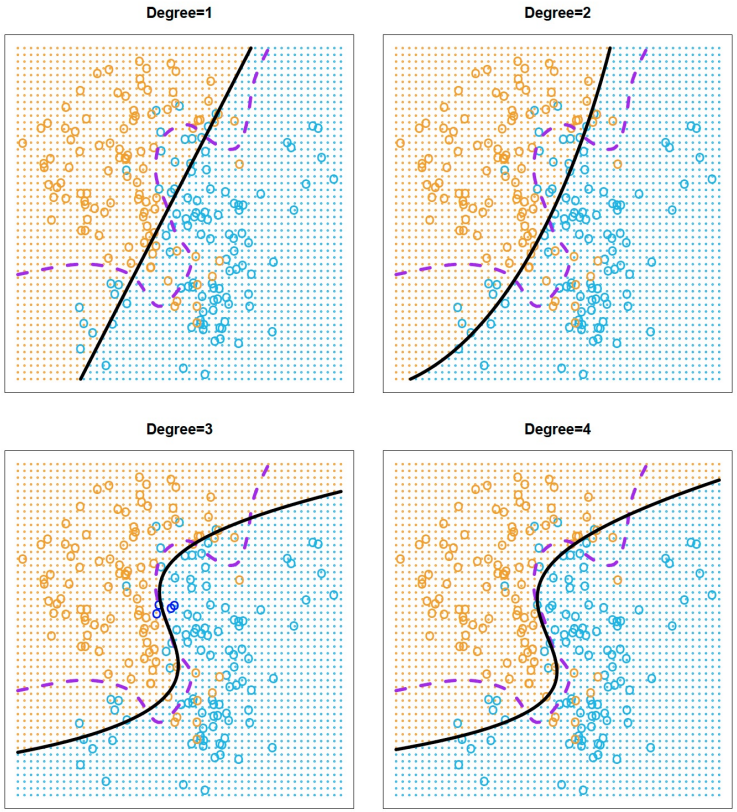



```
from sklearn.model_selection import ShuffleSplit
shuffle_split = ShuffleSplit(test_size=.5, train_size=.5, n_splits=10)
cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
```

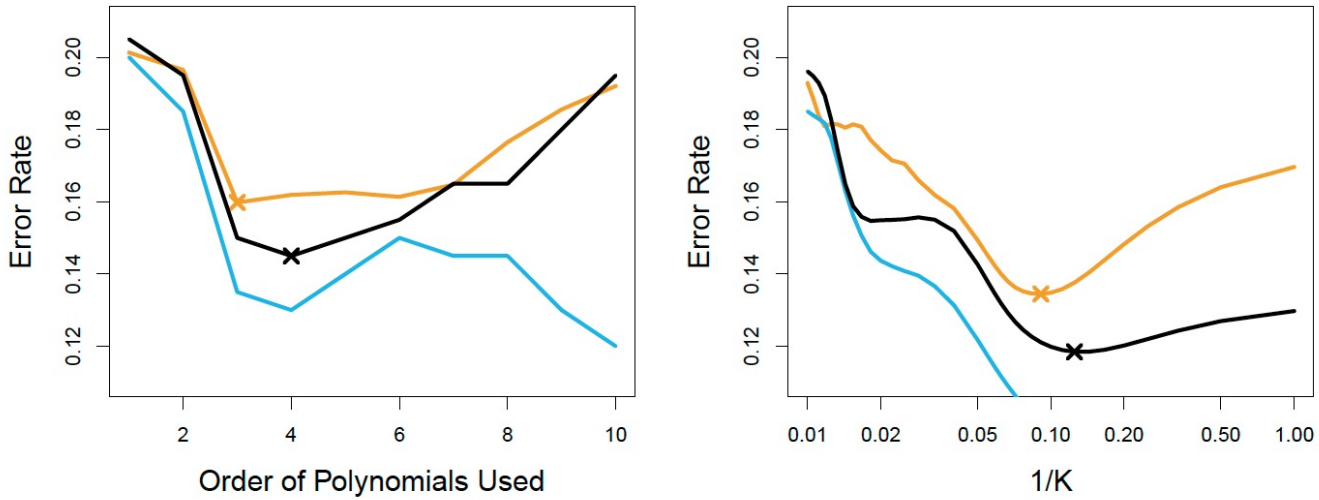
Cross-Validation on Classification Problems

In a classification problem, cross-validation works similarly except that rather than using MSE to quantify test error, we instead use the number of misclassified observations. For instance, in the classification setting, the LOOCV error rate takes the form

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i = \frac{1}{n}\sum_{i=1}^{n} 1\{y_i \neq \hat{y}_i\}.$$



CV: Logistic Polynomial and KNN



Bootstrap

The bootstrap is a widely applicable and extremely powerful statistical tool that can be used to quantify the uncertainty associated with a given estimator or statistical learning method. As a simple example, the bootstrap can be used to estimate the standard errors of the coefficients from a linear regression fit.

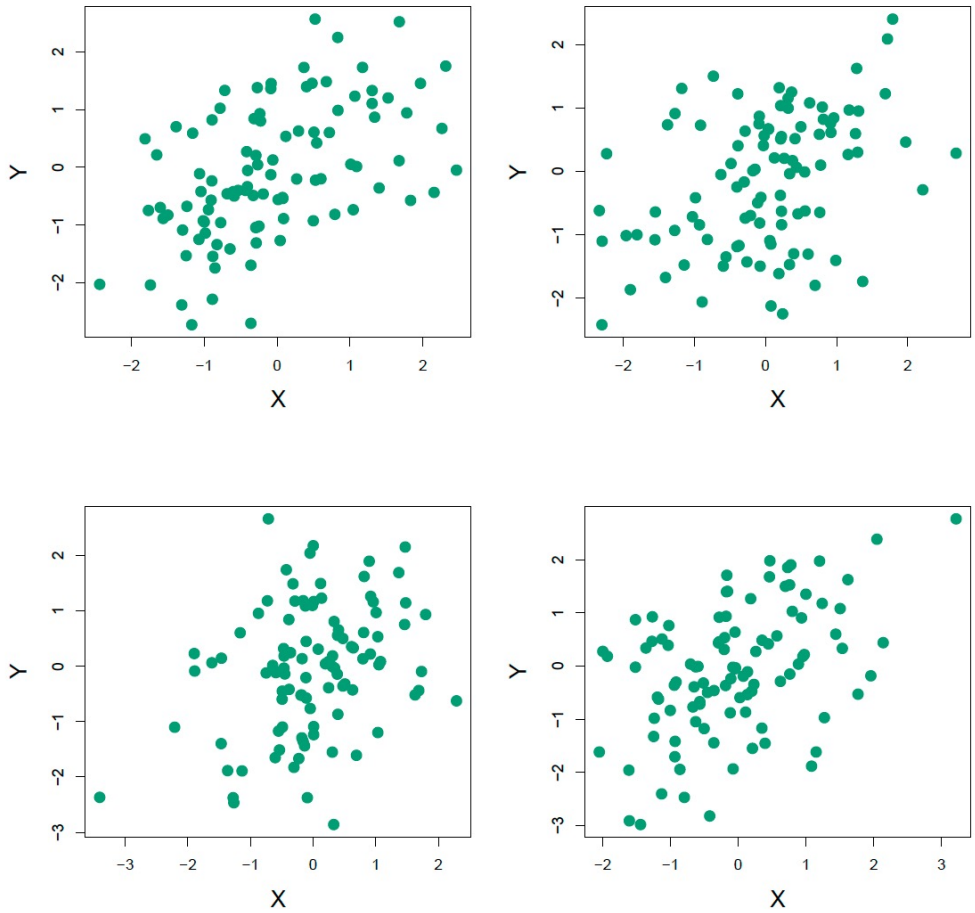
Toy Example

Suppose that we wish to invest a fixed sum of money in two financial assets that yield returns of X and Y, respectively. We will invest a fraction α of our money in X, and will invest the remaining $1-\alpha$ in Y. Since there is variability associated with the returns on these two assets, we wish to choose α to minimize the total risk, or variance, of our investment. In other words, we want to minimize $Var(\alpha X + (1-\alpha)Y)$. One can show that the value that minimizes the risk is given by

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}} \tag{1}$$

Bootstrap

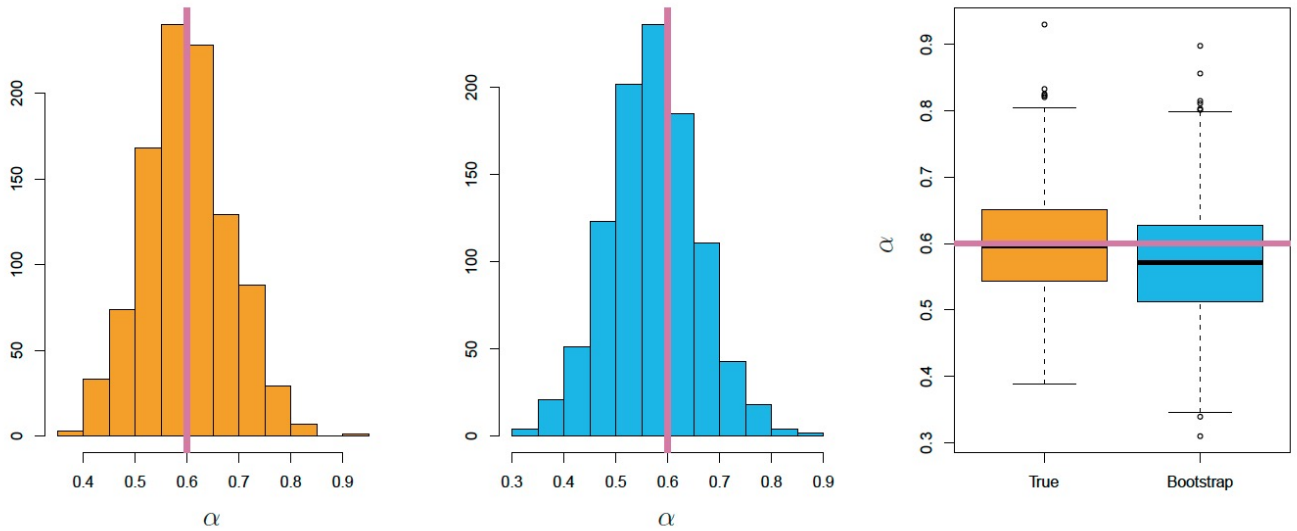
In reality, the quantities σ_X^2, σ_Y^2, and σ_{XY}^2 are unknown. We can compute estimates for these quantities, $\hat{\sigma}_X^2$, $\hat{\sigma}_Y^2$, and $\hat{\sigma}_{XY}^2$, using a data set that contains past measurements for X and Y.



From left to right and top to bottom, the resulting estimates for α are 0.576, 0.532, 0.657, and 0.651.

Bootstrap

It is natural to wish to quantify the accuracy of our estimate of α. To estimate the standard deviation of $\hat{\alpha}$, we repeated the process of simulating 100 paired observations of X and Y, and estimating α using Equation (1), 1,000 times. We thereby obtained 1,000 estimates for α, denoted as $\hat{\alpha}_1, \cdots, \hat{\alpha}_{1000}$.



Bootstrap

In practice, however, the procedure for estimating $SE(\hat{\alpha})$ outlined above cannot be applied, because for real data we cannot generate new samples from the original population. However, the bootstrap approach allows us to use a computer to emulate the process of obtaining new sample sets, so that we can estimate the variability of $\hat{\alpha}$ without generating additional samples. We obtain distinct data sets by repeatedly sampling observations from the original data set.