STAT 479: Machine Learning
Lecture Notes

Sebastian Raschka
Department of Statistics
University of Wisconsin–Madison

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2019/

Fall 2019

Contents

STAT 479: Machine Learning
Lecture Notes

Sebastian Raschka
Department of Statistics
University of Wisconsin–Madison

http://stat.wisc.edu/~sraschka/teaching/stat479-fs2019/

Fall 2019

10 Model Evaluation 3: Model Selection and Cross-Validation

10.1 Overview

Almost every machine learning algorithm comes with a large number of settings that we, the machine learning researchers and practitioners, need to specify. These tuning knobs, the so-called hyperparameters, help us control the behavior of machine learning algorithms when optimizing for performance, finding the right balance between bias and variance. Hyperparameter tuning for performance optimization is an art in itself, and there are no hard-and-fast rules that guarantee best performance on a given dataset. The previous lecture covered holdout and bootstrap techniques for estimating the generalization performance with and the construction of confidence intervals. The bias-variance trade-off was introduced as well, two lectures ago, and also discussed in the context of computing the uncertainty of performance estimates (last lecture). This third lecture in the model evaluation series now focuses on different methods of cross-validation for model evaluation and model selection. It covers cross-validation techniques to rank models from several hyperparameter configurations and estimate how well these generalize to independent datasets.

10.2 About Hyperparameters and Model Selection

Previously, the *holdout method* and different flavors of the *bootstrap* were introduced to estimate the generalization performance of our predictive models. We split the dataset into two parts: a training and a test dataset. After the machine learning algorithm fit a model to the training set, we evaluated it on the independent test set that we withheld from the machine learning algorithm during model fitting. While we were discussing challenges such as the bias-variance trade-off, we used fixed hyperparameter settings in our learning algorithms, such as the number of k in the *k-nearest neighbors algorithm*. We defined hyperparameters as the parameters of the learning algorithm itself, which we have to specify a priori – before model fitting. In contrast, we referred to the parameters of our resulting model as the *model parameters*.

So, what are hyperparameters, exactly? Considering the k-nearest neighbors algorithm, one example of a hyperparameter is the integer value of k (Figure 1). If we set k=3, the k-nearest neighbors algorithm will predict a class label based on a majority vote among the 3-nearest neighbors in the training set. The distance metric for finding these nearest neighbors is yet another hyperparameter of this algorithm.

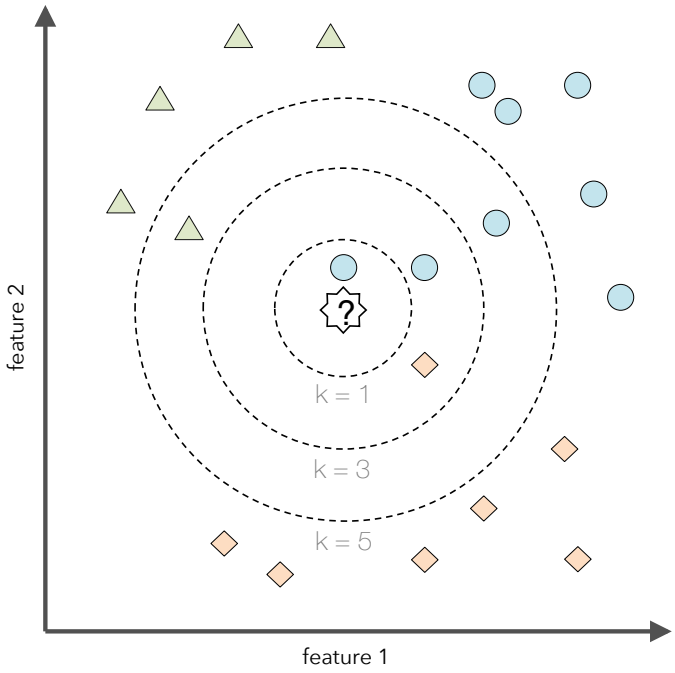

Figure 1: Illustration of the k-nearest neighbors algorithm with different choices for k.

Now, the k-nearest neighbors algorithm may not be an ideal choice for illustrating the difference between hyperparameters and model parameters, since it is a *lazy learner* and a nonparametric method. In this context, lazy learning (or instance-based learning) means that there is no training or model fitting stage: A k-nearest neighbors model literally stores or memorizes the training data and uses it only at prediction time. Thus, each training instance represents a parameter in the k-nearest neighbors model. In short, nonparametric models are models that cannot be described by a fixed number of parameters that are being adjusted to the training set. The structure of parametric models is not decided by the training data rather than being set *a priori*; nonparametric models do not assume that the data follows certain probability distributions unlike parametric methods (exceptions of nonparametric methods that make such assumptions are Bayesian nonparametric methods). Hence, we may say that nonparametric methods make fewer assumptions about the data than parametric methods.

In contrast to k-nearest neighbors, a simple example of a parametric method is logistic regression, a generalized linear model with a fixed number of model parameters: a weight coefficient for each feature variable in the dataset plus a bias (or intercept) unit. These weight coefficients in logistic regression, the model parameters, are updated by maximizing a log-likelihood function or minimizing the logistic cost (negative log-likelihood loss). For fitting a model to the training data, a hyperparameter of a logistic regression algorithm could be the number of iterations or passes over the training set (epochs) in gradient-based optimization. Another example of a hyperparameter would be the value of a regularization parameter such as the lambda-term in L2-regularized logistic regression (Figure 2). Note that we have not covered logistic regression in the lectures, yet. It will be covered in

the introductory parts of the deep learning class next Spring (Stat 479: Deep Learning). However, I assume many students are already familiar with logistic regression from other statistics courses. If you are not familiar with it, do not worry about it for now (and maybe think of linear regression as a parametric model instead – logistic regression is somewhat similar, but it is a classification model in contrast to linear regression)[1].

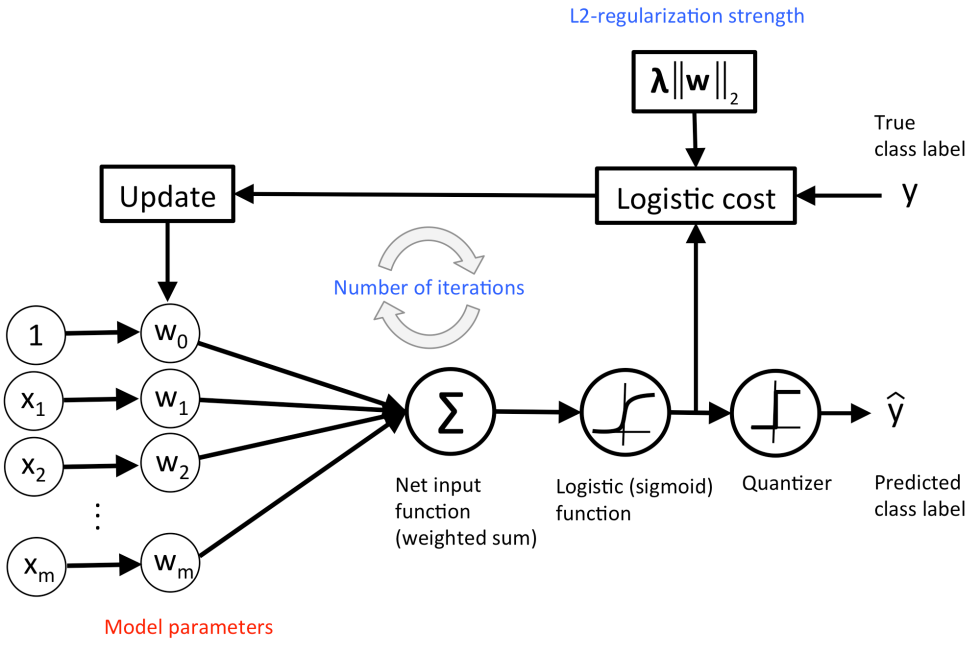

Figure 2: Conceptual overview of logistic regression.

Changing the hyperparameter values when running a learning algorithm over a training set may result in different models. The process of finding the best-performing model from a set of models that were produced by different hyperparameter settings is called *model selection*. The next section introduces an extension to the holdout method that is useful when carrying out this selection process.

10.3 The Three-Way Holdout Method for Hyperparameter Tuning

In an earlier lecture, we established that resubstitution validation is a bad approach for estimating of the generalization performance, because it will be overly optimistically biased. Since we want to know how well our model generalizes to new data, we used the holdout method to split the dataset into two parts, a training set and an independent test set. Can we use the holdout method for hyperparameter tuning? The answer is "yes." However, we have to make a slight modification to our initial approach, the "two-way" split, and split the dataset into three parts: a training, a validation, and a test set[2].

The process of hyperparameter tuning (or hyperparameter optimization) and model selection can be regarded as a meta-optimization task. While the learning algorithm optimizes an objective function on the training set (with exception to lazy learners), hyperparameter optimization is yet another task on top of it; here, we typically want to optimize a performance metric such as classification accuracy or the area under a Receiver Operating

[1] If you are interested, you can already read more about logistic regression in Python Machine Learning, 2nd edition, chapter 3: pages 51-76

[2] We actually already discussed this briefly in the previous lecture.

Characteristic (ROC) curve[3]. After the tuning stage, selecting a model based on the test set performance seems to be a reasonable approach. However, reusing the test set multiple times would introduce a bias and the final performance estimate and likely result in overly optimistic estimates of the generalization performance – one might say that "the test set leaks information." To avoid this problem, we could use a three-way split, dividing the dataset into a training, validation, and test dataset. Having a training-validation pair for hyperparameter tuning and model selections allows us to keep the test set "independent" for model evaluation. Once more, let us recall the "3 goals" of performance estimation that we introduced in the last lecture:

1. We want to estimate the generalization accuracy, the predictive performance of a model on future (unseen) data.
2. We want to increase the predictive performance by tweaking the learning algorithm and selecting the best-performing model from a given hypothesis space.
3. We want to identify the machine learning algorithm that is best-suited for the problem at hand; thus, we want to compare different algorithms, selecting the best-performing one as well as the best-performing model from the algorithm's hypothesis space.

The "three-way holdout method" is one way to tackle points 1 and 2. Though, if we are only interested in point 2, selecting the best model, and do not care so much about an "unbiased" estimate of the generalization performance, we could stick to the two-way split for model selection. Thinking back of our discussion about learning curves and pessimistic biases in the previous lecture, we noted that a machine learning algorithm often benefits from more labeled data; the smaller the dataset, the higher the pessimistic bias and the variance – the sensitivity of a model towards the data is partitioned.

"There ain't no such thing as a free lunch." The three-way holdout method for hyperparameter tuning and model selection is not the only – and certainly often not the best – way to approach this task. Later sections, will introduce alternative methods and discuss their advantages and trade-offs. However, before we move on to the probably most popular method for model selection, k-fold cross-validation (or sometimes also called "rotation estimation" in older literature), let us have a look at an illustration of the 3-way split holdout method in Figure 3.

Let us walk through Figure 3 step by step.

Step 1. We start by splitting our dataset into three parts, a training set for model fitting, a validation set for model selection, and a test set for the final evaluation of the selected model.

Step 2. This step illustrates the hyperparameter tuning stage. We use the learning algorithm with different hyperparameter settings (here: three) to fit models to the training data.

Step 3. Next, we evaluate the performance of our models on the validation set. This step illustrates the model selection stage; after comparing the performance estimates, we choose the hyperparameters settings associated with the best performance. Note that we often merge steps two and three in practice: we fit a model and compute its performance before moving on to the next model in order to avoid keeping all fitted models in memory.

[3]We will talk more about performance metrics, including ROC curves, in the upcoming lectures

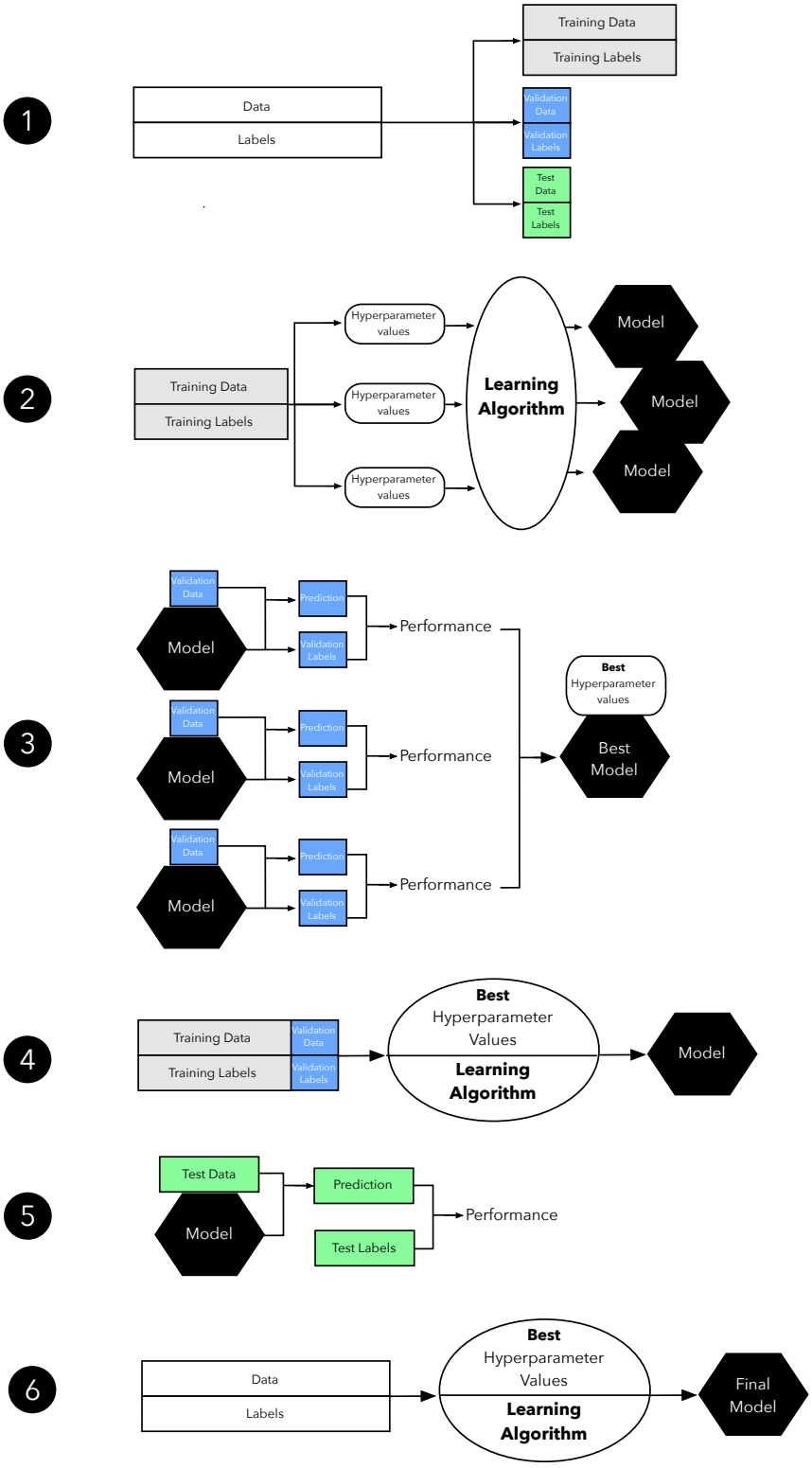

Figure 3: Illustration of the three-way holdout method for hyperparameter tuning.

Step 4. As discussed earlier, the performance estimates may suffer from pessimistic bias if the training set is too small. Thus, we can merge the training and validation set after model

selection and use the best hyperparameter settings from the previous step to fit a model to this larger dataset.

Step 5. Now, we can use the independent test set to estimate the generalization performance our model. Remember that the purpose of the test set is to simulate new data that the model has not seen before. Re-using this test set may result in an overoptimistic bias in our estimate of the model's generalization performance.

Step 6. Finally, we can make use of all our data – merging training and test set– and fit a model to all data points for real-world use.

Note that fitting the model on all available data might yield a model that is likely slightly different from the model evaluated in Step 5. However, in theory, using all data (that is, training and test data) to fit the model should only improve its performance. Under this assumption, the evaluated performance from Step 5 might slightly underestimate the performance of the model fitted in Step 6. (If we use test data for fitting, we do not have data left to evaluate the model, unless we collect new data.) In real-world applications, having the "best possible" model is often desired – or in other words, we do not mind if we slightly underestimated its performance. In any case, we can regard this sixth step as optional.

10.4 Introduction to K-Fold Cross-Validation

After covering the holdout method in great detail, it is about time that we talk more about the probably most common technique for model evaluation and model selection in machine learning practice: k-fold cross-validation. The term cross-validation is used loosely in literature, where practitioners and researchers sometimes refer to the train/test holdout method as a cross-validation technique. However, it might make more sense to think of cross-validation as a crossing over of training and validation stages in successive rounds. Here, the main idea behind cross-validation is that each sample in our dataset has the opportunity of being tested. K-fold cross-validation is a special case of cross-validation where we iterate over a dataset set k times. In each round, we split the dataset into k parts: one part is used for validation, and the remaining $k - 1$ parts are merged into a training subset for model evaluation as shown in Figure 4 , which illustrates the process of 5-fold cross-validation.

Just as in the "two-way" holdout method, we use a learning algorithm with fixed hyperparameter settings to fit models to the training folds in each iteration when we use the k-fold cross-validation method for *model evaluation*. In 5-fold cross-validation, this procedure will result in five different models fitted; these models were fit to distinct yet partly overlapping training sets and evaluated on non-overlapping validation sets. Eventually, we compute the cross-validation performance as the arithmetic mean over the k performance estimates from the validation sets.

We saw the main difference between the "two-way" holdout method and k-fold cross validation: k-fold cross-validation uses all data for training and testing. The idea behind this approach is to reduce the pessimistic bias by using more training data in contrast to setting aside a relatively large portion of the dataset as test data. And in contrast to the repeated holdout method, test folds in k-fold cross-validation are not overlapping. In repeated holdout, the repeated use of samples for testing results in performance estimates that become dependent between rounds; this dependence can be problematic for statistical comparisons, which we will be discussed in the next lecture. Also, k-fold cross-validation guarantees that each sample is used for validation in contrast to the repeated holdout-method, where some samples may never be part of the test set.

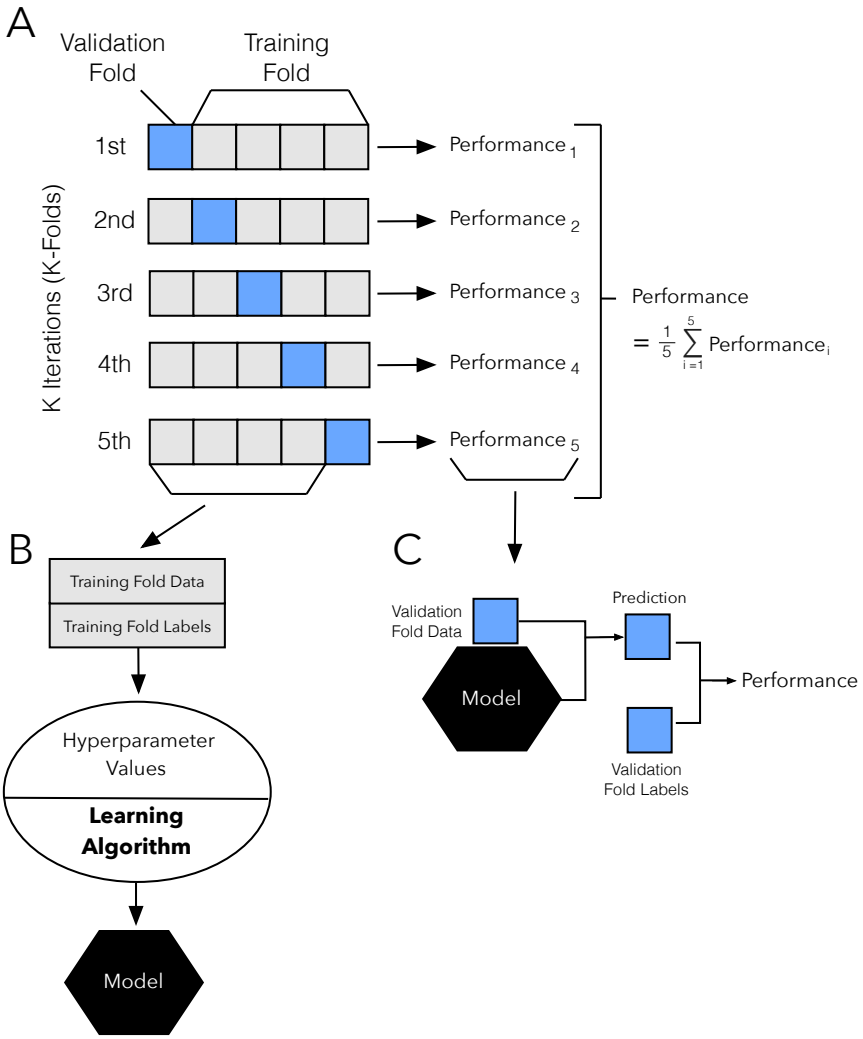

Figure 4: Illustration of the k-fold cross-validation procedure.

This section introduced k-fold cross-validation for *model evaluation*. In practice, however, k-fold cross-validation is more commonly used for model selection or algorithm selection. K-fold cross-validation for model selection is a topic that we will be covered in the next sections, and algorithm selection will be discussed in the next lecture then.

10.5 Special Cases: 2-Fold and Leave-One-Out Cross-Validation

At this point, you may wonder why $k = 5$ was chosen to illustrate k-fold cross-validation in the previous section. One reason is that it makes it easier to illustrate k-fold cross-validation compactly. Moreover, $k = 5$ is also a common choice in practice, since it is computationally less expensive compared to larger values of k. If k is too small, though, the pessimistic bias of the performance estimate may increase (since less training data is available for model fitting), and the variance of the estimate may increase as well since the model is more sensitive to how the data was split (in the next sections, experiments will be discussed that suggest $k = 10$ as a good choice for k).

In fact, there are two prominent, special cases of k-fold cross validation: $k = 2$ and $k =$

n. Most literature describes 2-fold cross-validation as being equal to the holdout method. However, this statement would only be true if we performed the holdout method by rotating the training and validation set in two rounds (for instance, using exactly 50% data for training and 50% of the examples for validation in each round, swapping these sets, repeating the training and evaluation procedure, and eventually computing the performance estimate as the arithmetic mean of the two performance estimates on the validation sets). Given how the holdout method is most commonly used though, here, we consider the holdout method and 2-fold cross-validation as two different processes as illustrated in Figure 5.

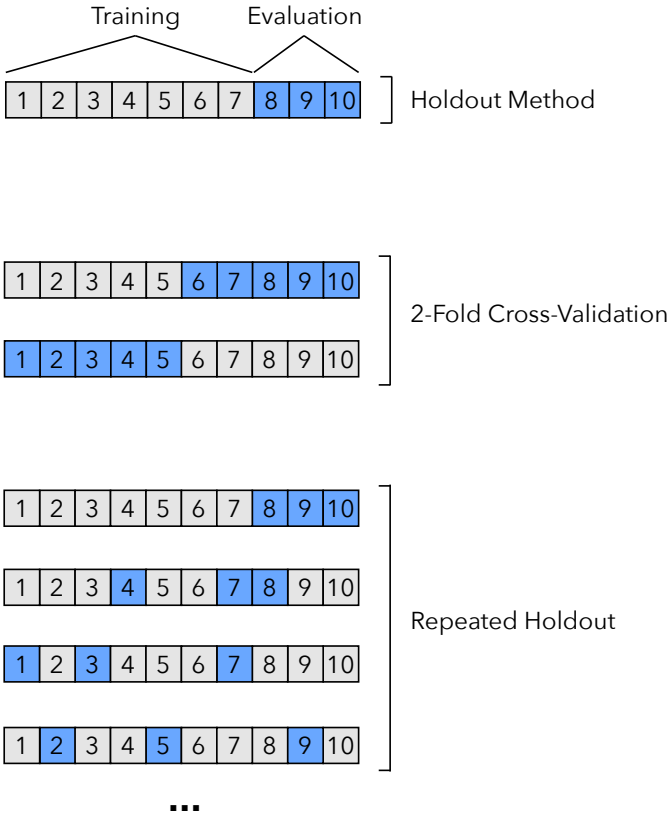

Figure 5: Comparison of the holdout method, 2-fold cross-validation, and the repeated holdout method.

Now, if we set $k = n$, that is, if we set the number of folds as being equal to the number of training instances, we refer to the k-fold cross-validation process as *Leave-One-Out Cross-Validation* (LOOCV), which is illustrated in Figure 6. In each iteration during LOOCV, we fit a model to $n - 1$ samples of the dataset and evaluate it on the single, remaining data point. Although this process is computationally expensive, given that we have n iterations, it can be useful for small datasets, cases where withholding data from the training set would be too wasteful.

Several studies compared different values of k in k-fold cross-validation, analyzing how the choice of k affects the variance and the bias of the estimate. Unfortunately, there is "No unbiased estimator of the variance of k-fold cross-validation:"

> The main theorem shows that there exists no universal (valid under all distributions) unbiased estimator of the variance of K-fold cross-validation.[4]

[4] **bengio2004no**.

Figure 6: Illustration of leave-one-out cross-validation.

However, we may still be interested in finding a "sweet spot," a value for k that seems to be a good trade-off between variance and bias in most cases, and the bias-variance trade-off discussion will be continued in the next section. For now, let us conclude this section by looking at an interesting research project where Hawkins and others compared performance estimates via LOOCV to the holdout method and recommend the LOOCV over the latter – if computationally feasible:

> [...] where available sample sizes are modest, holding back compounds for model testing is ill-advised. This fragmentation of the sample harms the calibration and does not give a trustworthy assessment of fit anyway. It is better to use all data for the calibration step and check the fit by cross-validation, making sure that the cross-validation is carried out correctly. [...] The only motivation to rely on the holdout sample rather than cross-validation would be if there was reason to think the cross-validation not trustworthy – biased or highly variable. But neither theoretical results nor the empiric results sketched here give any reason to disbelieve the cross-validation results.[5]

The conclusions in the previous quotation are partly based on the experiments carried out in this study using a 469-sample dataset, and Table 1 summarizes the findings in a comparison of different Ridge Regression models evaluated by LOOCV and the holdout method[6]. The first row corresponds to an experiment where the researchers used LOOCV to fit regression models to 100-example training subsets. The reported "mean" refers to the averaged difference between the true coefficients of determination (R^2) and the coefficients obtained via LOOCV (here called q^2) after repeating this procedure on different 100-example training

[5] **hawkins2003assessing**.
[6] **hawkins2003assessing**.

Table 1: Summary of the findings from the LOOCV vs. holdout comparison study conducted by Hawkins *and others* **hawkins2003assessing**. See text for details.

Experiment	Mean	Standard deviation
True R^2 — q^2	0.010	0.149
True R^2 — hold 50	0.028	0.184
True R^2 — hold 20	0.055	0.305
True R^2 — hold 10	0.123	0.504

sets and averaging the results. In rows 2-4, the researchers used the holdout method for fitting models to the 100-example training sets, and they evaluated the performances on holdout sets of sizes 10, 20, and 50 samples. Each experiment was repeated 75 times, and the *mean* column shows the average difference between the estimated R^2 and the true R^2 values. As we can see, the estimates obtained via LOOCV (q^2) are the closest to the true R^2 on average. The estimates obtained from the 50-example test set via the holdout method are also passable, though. Based on these particular experiments, we may agree with the researchers' conclusion:

> Taking the third of these points, if you have 150 or more compounds available, then you can certainly make a random split into 100 for calibration and 50 or more for testing. However it is hard to see why you would want to do this.
>
> **hawkins2003assessing**

One reason why we may prefer the holdout method may be concerns about computational efficiency, if the dataset is sufficiently large. As a rule of thumb, we can say that the pessimistic bias and large variance concerns are less problematic the larger the dataset. Moreover, it is not uncommon to repeat the k-fold cross-validation procedure with different random seeds in hope to obtain a "more robust" estimate. For instance, if we repeated a 5-fold cross-validation run 100 times, we would compute the performance estimate for 500 test folds report the cross-validation performance as the arithmetic mean of these 500 folds. (Although this is commonly done in practice, we note that the test folds are now overlapping.) However, there is no point in repeating LOOCV, since LOOCV always produces the same splits.

10.6 K-fold Cross-Validation and the Bias-Variance Trade-off

Based on the study by Hawkins *and others* **hawkins2003assessing** discussed in Section 10.5 we may prefer LOOCV over single train/test splits via the holdout method for small and moderately sized datasets. In addition, we can think of the LOOCV estimate as being approximately unbiased: the pessimistic bias of LOOCV ($k = n$) is intuitively lower compared $k < n$-fold cross-validation, since almost all (for instance, $n - 1$) training samples are available for model fitting.

While LOOCV is almost unbiased, one downside of using LOOCV over k-fold cross-validation with $k < n$ is the large variance of the LOOCV estimate. First, we have to note that LOOCV is *defect* when using a discontinuous loss-function such as the 0-1 loss in classification or even in continuous loss functions such as the mean-squared-error. It is said that LOOCV "[LOOCV has] high variance because the test set only contains one sample"[7] and "[LOOCV] is highly variable, since it is based upon a single observation (x1, y1)"[8]. These statements are certainly true if we refer to the variance between folds. Remember that if we use the 0-1 loss function (the prediction is either correct or not), we could consider each prediction as a

[7] **tan2005datamining**.
[8] **gareth2013stat**.

Bernoulli trial, and the number of correct predictions X if following a binomial distribution $X \approx B(n, p)$, where $n \in \mathbb{N}$ and $p \in [0, 1]$; the variance of a binomial distribution is defined as

$$\sigma^2 = np(1-p) \tag{1}$$

We can estimate the variability of a statistic (*here*: the performance of the model) from the variability of that statistic between subsamples. Obviously though, the variance between folds is a poor estimate of the variance of the LOOCV estimate – the variability due to randomness in our training data. Now, when we are talking about the variance of LOOCV, we typically mean the difference in the results that we would get if we repeated the resampling procedure multiple times on different data samples from the underlying distribution. In this context interesting point has been made by Hastie, Tibshirani, and Friedman:

> With $k = n$, the cross-validation estimator is approximately unbiased for the true (expected) prediction error, but can have high variance because the n "training sets" are so similar to one another.
>
> **hastie2009**

Or in other words, we can attribute the high variance to the well-known fact that the mean of highly correlated variables has a higher variance than the mean of variables that are not highly correlated. Maybe, this can intuitively be explained by looking at the relationship between covariance (cov) and variance (σ^2):

$$\text{cov}_{X,X} = \sigma^2_X. \tag{2}$$

or

$$\text{cov}_{X,X} = E\left[(X - \mu)^2\right] = \sigma^2_X \tag{3}$$

if we let $\mu = E(X)$.

And the relationship between covariance $\text{cov}_{X,Y}$ and correlation $\rho_{X,Y}$ (X and Y are random variables) is defined as

$$\text{cov}_{X,Y} = \rho_{X,Y}\ \sigma_X \sigma_Y, \tag{4}$$

where

$$\text{cov}_{X,Y} = E[(X - \mu_X)(Y - \mu_Y)] \tag{5}$$

and

$$\rho_{X,Y} = E[(X - \mu_X)(Y - \mu_Y)]/(\sigma_X \sigma_Y). \tag{6}$$

The large variance that is often associated with LOOCV has also been observed in empirical studies **kohavi1995**.

Now that we established that LOOCV estimates are generally associated with a large variance and a small bias, how does this method compare to k-fold cross-validation with other choices for k and the bootstrap method? In the last lecture, we discussed the pessimistic

bias of the standard bootstrap method, where the training set asymptotically (only) contains 63.2% of the examples from the original dataset; 2- or 3-fold cross-validation has about the same problem (the .632 bootstrap that was designed to address this pessimistic bias issue). However, Kohavi also observed in his experiments[9] that the bias in bootstrap was still extremely large for certain real-world datasets (now, optimistically biased) compared to k-fold cross-validation. Eventually, Kohavi's experiments on various real-world datasets suggest that 10-fold cross-validation offers the best trade-off between bias and variance. Furthermore, other researchers found that repeating k-fold cross-validation can increase the precision of the estimates while still maintaining a small bias[10].

Before moving on to model selection in the next section, the following points shall summarize the discussion of the bias-variance trade-off, by listing the general trends when increasing the number of folds or k:

- The bias of the performance estimator decreases (more accurate)
- The variance of the performance estimators increases (more variability)
- The computational cost increases (more iterations, larger training sets during fitting)
- Exception: decreasing the value of k in k-fold cross-validation to small values (for example, 2 or 3) also increases the variance on small datasets due to random sampling effects.

10.7 Model Selection via K-fold Cross-Validation

Previous sections introduced k-fold cross-validation for model *evaluation*. Now, this section discusses how to use the k-fold cross-validation method for model *selection*. Again, the key idea is to keep an independent test dataset, that we withhold from during training and model selection, to avoid the leaking of test data in the training stage. This procedure is outlined in Figure 7.

Although Figure 7 might seem complicated at first, the process is quite simple and analogous to the "three-way holdout" workflow that we discussed at the beginning of this article. The following paragraphs will discuss Figure 7 step by step.

Step 1. Similar to the holdout method, we split the dataset into two parts, a training and an independent test set; we tuck away the test set for the final model evaluation step at the end (Step 4).

Step 2. In this second step, we can now experiment with various hyperparameter settings; we could use Bayesian optimization, randomized search, or grid search, for example. For each hyperparameter configuration, we apply the k-fold cross-validation method on the training set, resulting in multiple models and performance estimates.

Step 3. Taking the hyperparameter settings that produced the best results in the k-fold cross-validation procedure, we can then use the complete training set for model fitting with these settings.

Step 4. Now, we use the independent test set that we withheld earlier (Step 1); we use this test set to evaluate the model that we obtained from Step 3.

[9] **kohavi1995**.
[10] **molinaro2005prediction**; **kim2009estimating**.

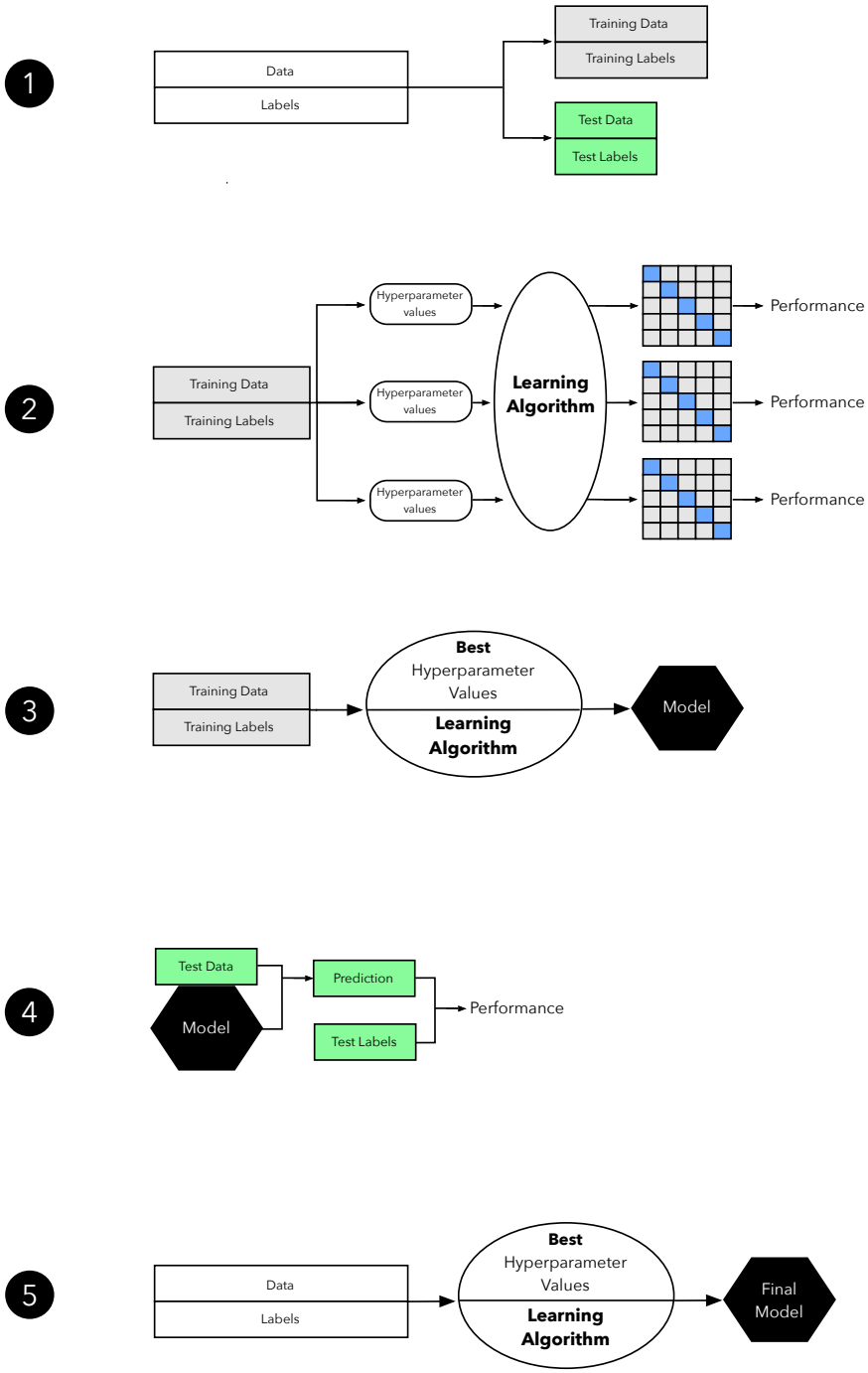

Figure 7: Illustration of k-fold cross-validation for model selection.

Step 5. Finally, after we completed the evaluation stage, we can optionally fit a model to all data (training and test datasets combined), which could be the model for (the so-called) deployment.

10.8 A Note About Model Selection and Large Datasets

When we browse the deep learning literature, we often find that that the 3-way holdout method is the method of choice when it comes to model evaluation; it is also common in older (non-deep learning literature) as well. As mentioned earlier, the three-way holdout may be preferred over k-fold cross-validation since the former is computationally cheap in comparison. Aside from computational efficiency concerns, we only use deep learning algorithms when we have relatively large sample sizes anyway, scenarios where we do not have to worry about high variance – the sensitivity of our estimates towards how we split the dataset for training, validation, and testing – so much. Thus, it is fine to use the holdout method with a training, validation, and test split over the k-fold cross-validation for model selection if the dataset is relatively large.

10.9 A Note About Feature Selection During Model Selection

Note that if we normalize data or select features, we typically perform these operations inside the k-fold cross-validation loop in contrast to applying these steps to the whole dataset upfront before splitting the data into folds. Feature selection inside the cross-validation loop reduces the bias through overfitting, since it avoids peaking at the test data information during the training stage. However, feature selection inside the cross-validation loop may lead to an overly pessimistic estimate, since less data is available for training. A more detailed discussion on this topic, whether to perform feature selection inside or outside the cross-validation loop, can be found in Refaeilzadeh's "On comparison of feature selection algorithms"[11].

10.10 The Law of Parsimony

Now that we discussed model selection in the previous section, let us take a moment and consider the Law of Parsimony, which is also known as Occam's Razor: "Among competing hypotheses, the one with the fewest assumptions should be selected." In model selection practice, Occam's razor can be applied, for example, by using the *one-standard error method*[12] as follows:

1. Consider the numerically optimal estimate and its standard error.
2. Select the model whose performance is within one standard error of the value obtained in step 1.

Although, we may prefer simpler models for several reasons, Pedro Domingos made a good point regarding the performance of "complex" models. Here is an excerpt from his article, "Ten Myths About Machine Learning[13]:"

> Simpler models are more accurate. This belief is sometimes equated with Occam's razor, but the razor only says that simpler explanations are preferable, not why. They're preferable because they're easier to understand, remember, and reason with. Sometimes the simplest hypothesis consistent with the data is less accurate for prediction than a more complicated one. Some of the most powerful learning algorithms output models that seem gratuitously elaborate – sometimes even continuing to add to them after they've perfectly fit the data – but that's how they beat the less powerful ones.

[11] **refaeilzadeh2007comparison**.
[12] **breiman1984classification**.
[13] https://medium.com/@pedromdd/ten-myths-about-machine-learning-d888b48334a3

Again, there are several reasons why we may prefer a simpler model if its performance is within a certain, acceptable range – for example, using the one-standard error method. Although a simpler model may not be the most "accurate" one, it may be computationally more efficient, easier to implement, and easier to understand and reason with compared to more complicated alternatives.

To see how the one-standard error method works in practice, let us consider a simple toy dataset: 300 training examples, concentric circles, and a uniform class distribution (150 samples from class 1 and 150 samples from class 2).

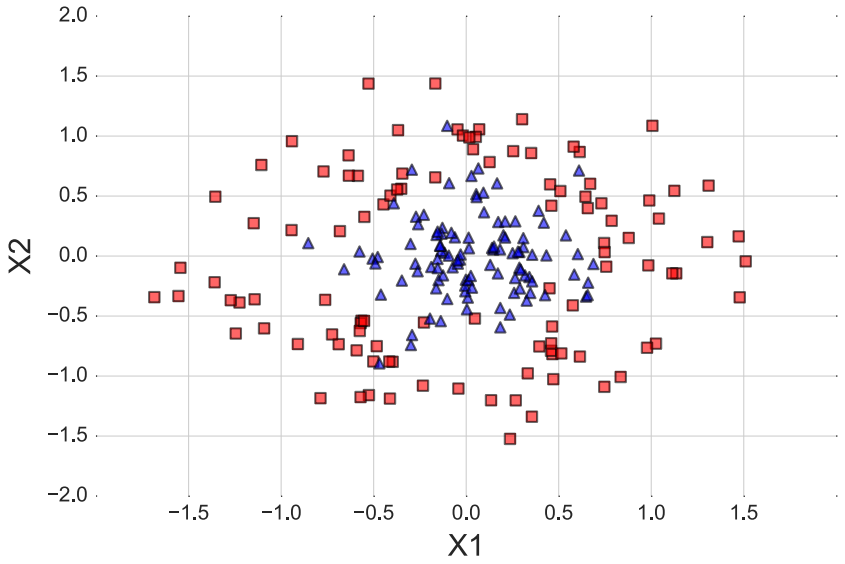

Figure 8: Concentric circles dataset with 210 training examples and uniform class distribution.

The concentric circles dataset is then split into two parts, 70% training data and 30% test data, using stratification to maintain equal class proportions. The 210 samples from the training dataset are shown in Figure 8.

Now, let us assume that our goal is to optimize the γ (gamma) hyperparameter of a Support Vector Machine (SVM) with a non-linear Radial Basis Function-kernel (RBF-kernel), where γ is the free parameter of the Gaussian RBF:

$$K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2), \gamma > 0. \tag{7}$$

(Intuitively, we can think of γ as a parameter that controls the influence of single training samples on the decision boundary.)

After running the RBF-kernel SVM algorithm with different γ values over the training set, using stratified 10-fold cross-validation, the performance estimates shown in Figure 9 were obtained, where the error bars are the standard errors of the cross-validation estimates.

As shown in Figure 9, Choosing γ values between 0.1 and 100 resulted in a prediction accuracy of 80% or more. Furthermore, we can see that $\gamma = 10.0$ resulted in a fairly complex decision boundary, and $\gamma = 0.001$ resulted in a decision boundary that is too simple to separate the two classes in the concentric circles dataset. In fact, $\gamma = 0.1$ seems like a good trade-off between the two aforementioned models ($\gamma = 10.0$ and $\gamma = 0.1$) – the performance of the corresponding model falls within one standard error of the best performing model with $\gamma = 0$ or $\gamma = 10$.

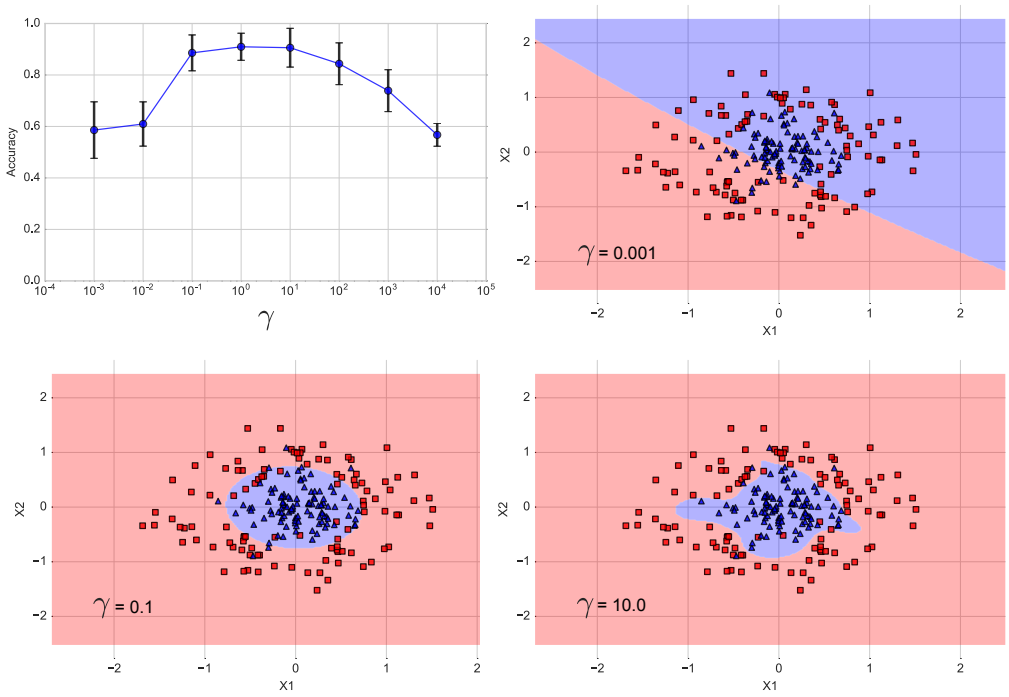

Figure 9: Performance estimates and decision regions of an RBF-kernel SVM with stratified 10-fold cross-validation for different γ values. Error bars represent the standard error of the cross-validation estimates.

10.11 Summary

There are many ways for evaluating the generalization performance of predictive models. So far, this lecture covered the holdout method, different flavors of the bootstrap approach, and k-fold cross-validation. Using the holdout method is absolutely fine for model evaluation when working with relatively large sample sizes. For hyperparameter optimization, we may prefer 10-fold cross-validation, and Leave-One-Out cross-validation is a good option when working with small sample sizes. When it comes to model selection, again, the "three-way" holdout method may be a good choice if the dataset is large, if computational efficiency is a concern; a good alternative is using k-fold cross-validation with an independent test set.

The next lecture will introduce some statistical methods/tests for comparing the performance of different models as well as empirical cross-validation approaches for comparing different machine learning algorithms.