STAT 451: Introduction to Machine Learning Lecture Notes

Sebastian Raschka
Department of Statistics
University of Wisconsin–Madison

http://stat.wisc.edu/~sraschka/teaching/stat451-fs2020/

Fall 2020

Contents

STAT 451: Introduction to Machine Learning Lecture Notes

Sebastian Raschka
Department of Statistics
University of Wisconsin–Madison

http://stat.wisc.edu/~sraschka/teaching/stat451-fs2020/

Fall 2020

9 Model Evaluation 2: Confidence Intervals and Resampling Techniques

9.1 Introduction: Essential Model Evaluation Terms and Techniques

Machine learning has become a central part of our life – as consumers, customers, and hopefully as researchers and practitioners. Whether we are applying predictive modeling techniques to our research or business problems, I believe we have one thing in common: We want to make "good" predictions. Fitting a model to our training data is one thing, but how do we know that it generalizes well to unseen data? How do we know that it does not simply memorize the data we fed it and fails to make good predictions on future samples, samples that it has not seen before? And how do we select a good model in the first place? Maybe a different learning algorithm could be better-suited for the problem at hand?

Model evaluation is certainly not just the end point of our machine learning pipeline. Before we handle any data, we want to plan ahead and use techniques that are suited for our purposes. In this article, we will go over a selection of these techniques, and we will see how they fit into the bigger picture, a typical machine learning workflow.

9.1.1 Performance Estimation: Generalization Performance Vs. Model Selection

Let us consider the obvious question, "How do we estimate the performance of a machine learning model?" A typical answer to this question might be as follows: "First, we feed the training data to our learning algorithm to learn a model. Second, we predict the labels of our test set. Third, we count the number of wrong predictions on the test dataset to compute the model's prediction accuracy." Depending on our goal, however, estimating the performance of a model is not that trivial, unfortunately. Maybe we should address the previous question from a different angle: "Why do we care about performance estimates at all?" Ideally, the estimated performance of a model tells how well it performs on unseen data – making predictions on future data is often the main problem we want to solve in applications of machine learning or the development of new algorithms. Typically, machine learning involves a lot of experimentation, though – for example, the tuning of the internal knobs of a learning algorithm, the so-called hyperparameters. Running a learning algorithm over a training dataset with different hyperparameter settings will result in different models. Since we are typically interested in selecting the best-performing model from this set, we need to find a way to estimate their respective performances in order to rank them against each other.

Going one step beyond mere algorithm fine-tuning, we are usually not only experimenting with the one single algorithm that we think would be the "best solution" under the given circumstances. More often than not, we want to compare different algorithms to each other, oftentimes in terms of predictive *and* computational performance. Let us summarize the main points why we evaluate the predictive performance of a model:

1. We want to estimate the generalization performance, the predictive performance of our 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.

Although these three sub-tasks listed above have all in common that we want to estimate the performance of a model, they all require different approaches. We will discuss some of the different methods for tackling these sub-tasks in this and the following lectures.

Of course, we want to estimate the future performance of a model as accurately as possible. However, we should keep in mind that that biased performance estimates can be okay in the context of model selection and algorithm selection **if the bias affects all models equally**. In other words, if we rank different models or algorithms against each other in order to select the best-performing one, we only need to know their "relative" performance. For example, if **all** performance estimates are pessimistically biased, and we underestimate their performances by 10%, it will not affect the ranking order. Let us assume we know the true model accuracies:

$$h_2(\mathbf{x}) : 75\% > h_1(\mathbf{x}) : 70\% > h_3(\mathbf{x}) : 65\%,$$

we would still rank them the same way if all apparent accuracies were measured with a 10% pessimistic bias:

$$h_2(\mathbf{x}) : 65\% > h_1(\mathbf{x}) : 60\% > h_3(\mathbf{x}) : 55\%.$$

However, note that if we reported the generalization (future prediction) accuracy of the best ranked model ($h_2(\mathbf{x})$ to be 65%, this would obviously be quite inaccurate – given that the true accuracy is 75 %. Estimating the absolute performance of a model is probably one of the most challenging tasks in machine learning.

9.1.2 Assumptions and Terminology

Model evaluation is certainly a complex topic. To make sure that we do not diverge too much from the core message, let us make certain assumptions and go over some of the technical terms that we will use throughout this lecture.

i.i.d. We assume that the training examples are i.i.d (independent and identically distributed), which means that all examples have been drawn from the same probability distribution and are statistically independent from each other. A scenario where training examples are not independent would be working with temporal data or time-series data[1].

Supervised learning and classification. The lecture focuses on supervised learning. Although many concepts also apply to regression analysis, we will focus on classification, the assignment of categorical target labels to the training and test examples.

0-1 loss and prediction accuracy. In this and the following lectures, we will focus on the classification accuracy and and misclassification error. Classification accuracy is defined as the number of all correct predictions divided by the number of examples in the dataset. We compute the prediction accuracy as the number of correct predictions divided by the number of examples n. Or in more formal terms, we define the prediction accuracy ACC as

$$\mathrm{ACC} = 1 - \mathrm{ERR}, \tag{1}$$

where the prediction error, ERR, is computed as the expected value of the 0-1 loss over n examples in a dataset S:

$$\mathrm{ERR}_S = \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}^{[i]}, y^{[i]}), \tag{2}$$

with y being the true class label ($f(\mathbf{x}) = y$) and $\hat{y}$ being the predicted class label ($h(\mathbf{x}) = \hat{y}$) as usual.

Remember, the 0-1 loss $L(\cdot)$ is simply a Kronecker function, defined as

$$L(\hat{y}^{[i]}, y^{[i]}) = \begin{cases} 0 & \text{if } \hat{y}^{[i]} = y^{[i]} \\ \\ 1 & \text{if } \hat{y}^{[i]} \neq y^{[i]}, \end{cases} \tag{3}$$

where $y^{[i]}$ is the ith true class label and $\hat{y}^{[i]}$ the ith predicted class label,

$$h(\mathbf{x}^{[i]}) = \hat{y}^{[i]}.$$

Our objective is to learn a model/hypothesis h that has a good generalization performance. Such a model maximizes the prediction accuracy or, vice versa, minimizes the probability, $C(h)$, of making a wrong prediction:

$$C(h) = \Pr_{(\mathbf{x},y)\sim\mathcal{D}} [h(\mathbf{x}) \neq y]. \tag{4}$$

Here, $\mathcal{D}$ is the generating distribution the dataset has been drawn from, $\mathbf{x}$ is the feature vector of a training example with class label y.

[1]We will not discuss time series data in this course, but if you choose a project where the i.i.d. assumption is violated, you are still encouraged to work on it and research relevant techniques. For example, see http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

Bias. Throughout this article, the term bias refers to the *statistical bias* (in contrast to the bias in a machine learning system[2]). In general terms, the bias of an estimator $\hat{\beta}$ is the difference between its expected value $E[\hat{\beta}]$ and the true value of a parameter β being estimated:

$$\text{Bias} = E[\hat{\beta}] - \beta. \tag{5}$$

Thus, if $\text{Bias} = E[\hat{\beta}] - \beta = 0$, then $\hat{\beta}$ is an unbiased estimator of β. More concretely, we compute the prediction bias as the difference between the expected prediction accuracy of a model and its true prediction accuracy. For example, if we computed the prediction accuracy on the training set, this would be an optimistically biased estimate of the absolute accuracy of a model since it would overestimate its true accuracy.

Variance. The variance is simply the statistical variance of the estimator $\hat{\beta}$ and its expected value $E[\hat{\beta}]$:

$$\text{Variance} = E\Big[\big(\hat{\beta} - E[\hat{\beta}]\big)^2\Big]. \tag{6}$$

The variance is a measure of the variability of a model's predictions if we repeat the learning process multiple times with small fluctuations in the training set. The more sensitive the model-building process is towards these fluctuations, the higher the variance.

Finally, let us disambiguate the terms model, hypothesis, classifier, learning algorithms, and parameters:

Target function. In predictive modeling, we are typically interested in modeling a particular process; we want to learn or approximate a specific, unknown function. The target function $f(x) = y$ is the true function $f(\cdot)$ that we want to model.

Hypothesis. A hypothesis is a certain function that we believe (or hope) is similar to the true function, the target function $f(\cdot)$ that we want to model. In context of *spam* classification, it would be a classification rule we came up with that allows us to separate spam from non-spam emails.

Model. In the machine learning field, the terms *hypothesis* and *model* are often used interchangeably. In other sciences, these terms can have different meanings: A hypothesis could be the "educated guess" by the scientist, and the model would be the manifestation of this guess to test this hypothesis.

Learning algorithm. Again, our goal is to find or approximate the target function, and the learning algorithm is a set of instructions that tried to model the target function using a training dataset. A learning algorithm comes with a hypothesis space, the set of possible hypotheses it can explore to model the unknown target function by formulating the final hypothesis.

Hyperparameters. Hyperparameters are the *tuning parameters* of a machine learning algorithm – for example, the value of k (number of nearest neighbors in a k-Nearest Neighbor algorithm, or a value for setting the maximum depth of a decision tree classifier. In contrast,

[2] **We also called this "machine learning bias" the "inductive bias" of an algorithm.**.

model parameters are the parameters that a learning algorithm fits to the training data – the parameters of the model itself. For example, the weight coefficients (or slope) of a linear regression line and its bias term (*here:* y-axis intercept) are model parameters.

9.2 Resubstitution Validation and the Holdout Method

The holdout method is inarguably the simplest model evaluation technique; it can be summarized as follows. First, we take a labeled dataset and split it into two parts: A training and a test set. Then, we fit a model to the training data and predict the labels of the test set. The fraction of correct predictions, which can be computed by comparing the predicted labels to the ground truth labels of the test set, constitutes our estimate of the model's prediction accuracy. Here, it is important to note that we do not want to train and evaluate a model on the same training dataset (this is called *resubstitution validation* or *resubstitution evaluation*), since it would typically introduce a very optimistic bias due to overfitting. In other words, we cannot tell whether the model simply memorized the training data, or whether it generalizes well to new, unseen data. (On a side note, we can estimate this so-called *optimism bias* as the difference between the training and test accuracy.)

Typically, the splitting of a dataset into training and test sets is a simple process of *random subsampling*. We assume that all data points have been drawn from the same probability distribution (with respect to each class). And we randomly choose 2/3 of these samples for the training set and 1/3 of the samples for the test set. Note that there are two problems with this approach, which we will discuss in the next sections.

9.2.1 Stratification

We have to keep in mind that a dataset represents a random sample drawn from a probability distribution, and we typically assume that this sample is representative of the true population – more or less. Now, further subsampling without replacement alters the statistic (mean, proportion, and variance) of the sample. The degree to which subsampling without replacement affects the statistic of a sample is inversely proportional to the size of the sample. Let us have a look at an example using the *Iris* dataset [3], which we randomly divide into 2/3 training data and 1/3 test data as illustrated in Figure 1.

When we randomly divide a labeled dataset into training and test sets, we violate the assumption of *statistical independence*. The Iris datasets consists of 50 Setosa, 50 Versicolor, and 50 Virginica flowers; the flower species are distributed uniformly:

- 33.3% Setosa
- 33.3% Versicolor
- 33.3% Virginica

If a random function assigns 2/3 of the flowers (100) to the training set and 1/3 of the flowers (50) to the test set, it may yield the following (also shown in Figure 1):

- training set $\rightarrow$ 38 $\times$ Setosa, 28 $\times$ Versicolor, 34 $\times$ Virginica
- test set $\rightarrow$ 12 $\times$ Setosa, 22 $\times$ Versicolor, 16 $\times$ Virginica

Assuming that the Iris dataset is representative of the true population (for instance, assuming that iris flower species are distributed uniformly in nature), we just created two

[3] https://archive.ics.uci.edu/ml/datasets/iris

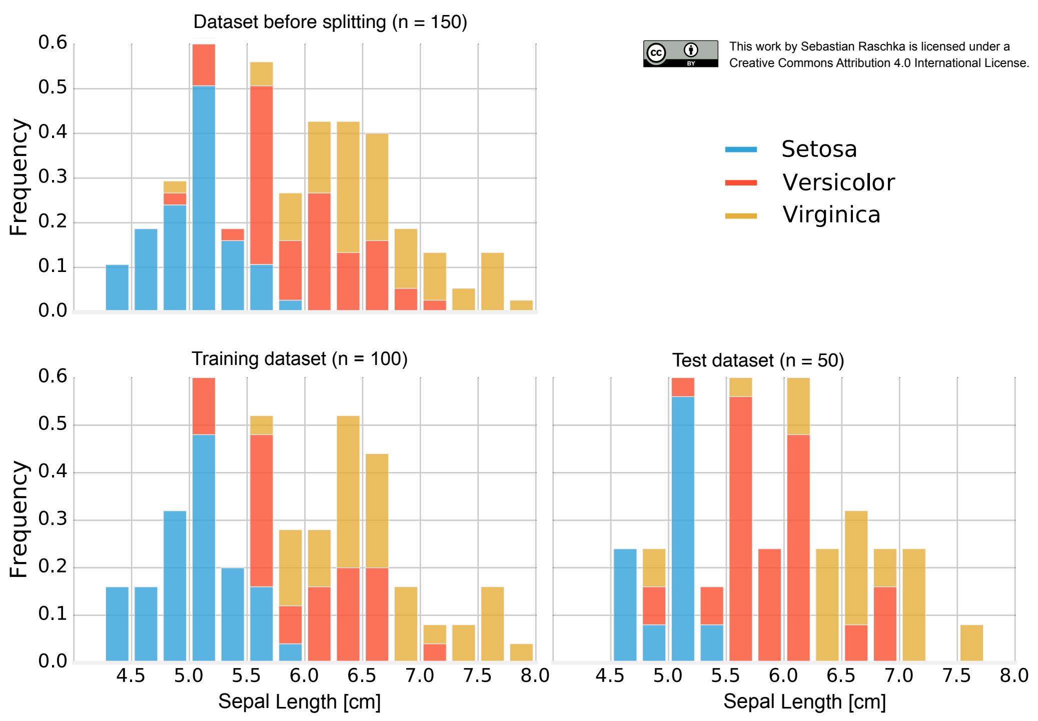

Figure 1: Distribution of *Iris* flower classes upon random subsampling into training and test sets.

imbalanced datasets with non-uniform class distributions. The class ratio that the learning algorithm uses to learn the model is "38% / 28% / 34%." The test dataset that is used for evaluating the model is imbalanced as well, and even worse, it is balanced in the "opposite" direction: "24% / 44% / 32%." Unless the learning algorithm is completely insensitive to these perturbations, this is certainly not ideal. The problem becomes even worse if a dataset has a high class imbalance upfront, prior to the random subsampling. In the worst-case scenario, the test set may not contain any instance of a minority class at all. Thus, a recommended practice is to divide the dataset in a stratified fashion. Here, *stratification* simply means that we randomly split a dataset such that each class is correctly represented in the resulting subsets (the training and the test set) – in other words, stratification is an approach to maintain the original class proportion in resulting subsets.

It shall be noted that random subsampling in non-stratified fashion is usually not a big concern when working with relatively large and balanced datasets. However, in my opinion, stratified resampling is usually beneficial in machine learning applications. Moreover, stratified sampling is incredibly easy to implement, and Ron Kohavi provides empirical evidence[4] that stratification has a positive effect on the variance and bias of the estimate in k-fold cross-validation, a technique that we will revisit once more later for a more detailed discussion.

9.3 Holdout Validation

Before diving deeper into the pros and cons of the holdout validation method, Figure 2 provides a visual summary of this method that will be discussed in the following text.

[4]Ron Kohavi. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *International Joint Conference on Artificial Intelligence* 14.12 (1995), pp. 1137–1143.

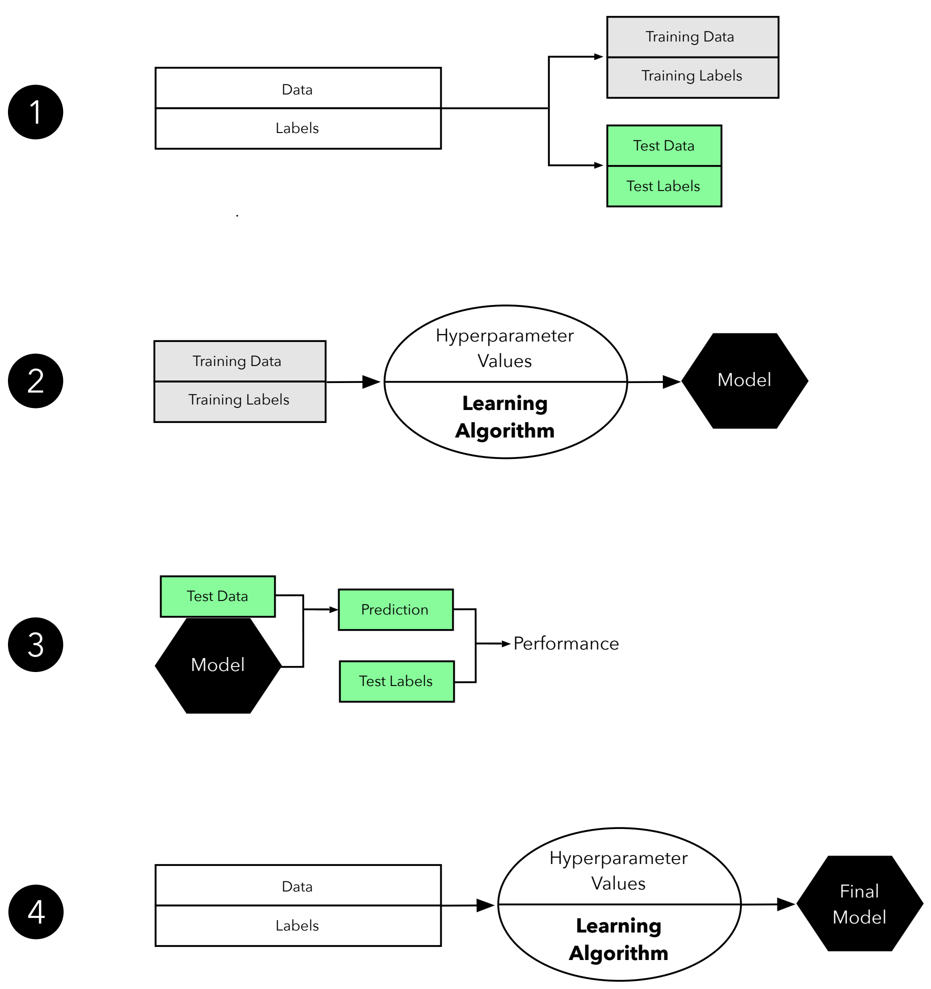

Figure 2: Visual summary of the holdout validation method.

Step 1. First, we randomly divide our available data into two subsets: a training and a test set. Setting test data aside is a work-around for dealing with the imperfections of a non-ideal world, such as limited data and resources, and the inability to collect more data from the generating distribution. Here, the test set shall represent new, unseen data to the model; it is important that the test set is only used once to avoid introducing bias when we estimating the generalization performance. Typically, we assign 2/3 to the training set and 1/3 of the data to the test set. Other common training/test splits are 60/40, 70/30, or 80/20 – or even 90/10 if the dataset is relatively large.

Step 2. After setting test examples aside, we pick a learning algorithm that we think could be appropriate for the given problem. As a quick reminder regarding the *Hyperparameter Values* depicted in Figure 2, hyperparameters are the parameters of our learning algorithm, or meta-parameters. And we have to specify these hyperparameter values manually – the learning algorithm does not learn these from the training data in contrast to the actual model parameters. Since hyperparameters are not learned during model fitting, we need some sort of "extra procedure" or "external loop" to optimize these separately – this holdout approach is ill-suited for the task. So, for now, we have to go with some fixed hyperparameter values – we could use our intuition or the default parameters of an off-the-shelf algorithm if we are using an existing machine learning library.

Step 3. After the learning algorithm fit a model in the previous step, the next question is: How "good" is the performance of the resulting model? This is where the independent test set comes into play. Since the learning algorithm has not "seen" this test set before, it should provide a relatively unbiased estimate of its performance on new, unseen data. Now, we take this test set and use the model to predict the class labels. Then, we take the predicted class labels and compare them to the "ground truth," the correct class labels, to estimate the models generalization accuracy or error.

Step 4. Finally, we obtained an estimate of how well our model performs on unseen data. So, there is no reason for with-holding the test set from the algorithm any longer. Since we assume that our samples are i.i.d., there is no reason to assume the model would perform worse after feeding it all the available data. As a rule of thumb, the model will have a better generalization performance if the algorithms uses more informative data – assuming that it has not reached its capacity, yet.

9.3.1 Pessimistic Bias

Section 9.2 (Resubstitution Validation and the Holdout Method) referenced two types of problems that occur when a dataset is split into separate training and test sets. The first problem that occurs is the violation of independence and the changing class proportions upon subsampling (discussed in Section 9.2.1). Walking through the holdout validation method (Section 9.3) touched upon a second problem we encounter upon subsampling a dataset: Step 4 mentioned *capacity* of a model, and whether additional data could be useful or not. To follow up on the capacity issue: If a model has *not* reached its capacity, the performance estimate would be pessimistically biased. This assumes that the algorithm could learn a better model if it was given more data – by splitting off a portion of the dataset for testing, we withhold valuable data for estimating the generalization performance (for instance, the test dataset). To address this issue, one might fit the model to the whole dataset after estimating the generalization performance (see Figure 2 step 4). However, using this approach, we cannot estimate its generalization performance of the refit model, since we have now "burned" the test dataset. It is a dilemma that we cannot really avoid in real-world application, but we should be aware that our estimate of the generalization performance may be pessimistically biased if only a portion of the dataset, the training dataset, is used for model fitting (this is especially affects models fit to relatively small datasets).

9.4 Confidence Intervals via Normal Approximation

Using the holdout method as described in Section 9.3, we computed a point estimate of the generalization performance of a model. Certainly, a confidence interval around this estimate would not only be more informative and desirable in certain applications, but our point estimate could be quite sensitive to the particular training/test split (for instance, suffering from high variance). A simple approach for computing confidence intervals of the predictive accuracy or error of a model is via the so-called *normal approximation*. Here, we assume that the predictions follow a normal distribution, to compute the confidence interval on the mean on a single training-test split under the central limit theorem. The following text illustrates how this works.

As discussed earlier, we compute the prediction accuracy as follows:

$$ACC_S = \frac{1}{n}\sum_{i=1}^{n}\delta(L(\hat{y}^{[i]}, y^{[i]})), \tag{7}$$

where $L(\cdot)$ is the 0-1 loss function (Equation 3), and n denotes the number of samples in the test dataset. Further, let $\hat{y}^{[i]}$ be the predicted class label and $y^{[i]}$ be the ground truth class label of the ith test example, respectively. So, we could now consider each prediction as a Bernoulli trial, and the number of correct predictions X is following a binomial distribution $X \sim (n, p)$ with n test examples, k trials, and the probability of success p, where $n \in \mathbb{N}$ and $p \in [0, 1]$:

$$f(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{8}$$

for $k = 0, 1, 2, ..., n$, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}. \tag{9}$$

(Here, p is the probability of success, and consequently, $(1 - p)$ is the probability of failure – a wrong prediction.)

Now, the expected number of successes is computed as $\mu = np$, or more concretely, if the model has a 50% success rate, we expect 20 out of 40 predictions to be correct. The estimate has a variance of

$$\sigma^2 = np(1-p) = 10 \tag{10}$$

and a standard deviation of

$$\sigma = \sqrt{np(1-p)} = 3.16. \tag{11}$$

Since we are interested in the *average* number of successes, not its absolute value, we compute the variance of the accuracy estimate as

$$\sigma^2 = \frac{1}{n} ACC_S(1 - ACC_S), \tag{12}$$

and the respective standard deviation as

$$\sigma = \sqrt{\frac{1}{n} ACC_S(1 - ACC_S)}. \tag{13}$$

Under the normal approximation, we can then compute the confidence interval as

$$ACC_S \pm z\sqrt{\frac{1}{n} ACC_S\,(1 - ACC_S)}, \tag{14}$$

where α is the error quantile and z is the $1 - \frac{\alpha}{2}$ quantile of a standard normal distribution. For a typical confidence interval of 95%, ($\alpha = 0.05$), we have $z = 1.96$.

In practice, however, I would rather recommend repeating the training-test split multiple times to compute the confidence interval on the mean estimate (for instance, averaging the individual runs). In any case, one interesting take-away for now is that having fewer samples in the test set increases the variance (see n in the denominator above) and thus widens the confidence interval. Confidence intervals and estimating uncertainty will be discussed in more detail in the next section, Section 9.5.

9.5 Overview of the Next Sections on Resampling

In a previous section (Section 9.1, Introduction: Essential Model Evaluation Terms and Techniques), I introduced the general ideas behind model evaluation in supervised machine learning. We then discussed the holdout method, which helps us to deal with real world limitations such as limited access to new, labeled data for model evaluation. Using the holdout method, we split our dataset into two parts: A training and a test set. First, we provide the training data to a supervised learning algorithm. The learning algorithm builds a model from the training set of labeled observations. Then, we evaluate the predictive performance of the model on an independent test set that shall represent new, unseen data. Also, we briefly introduced the normal approximation, which requires us to make certain assumptions that allow us to compute confidence intervals for modeling the uncertainty of our performance estimate based on a single test set, which we have to take with a grain of salt.

This section introduces some of the advanced techniques for model evaluation. We will start by discussing techniques for estimating the uncertainty of our estimated model performance as well as the model's variance and stability. And after getting these basics under our belt, we will look at cross-validation techniques for model selection in the next lecture. As we remember from Section 9.1, there are three related, yet different tasks or reasons why we care about model evaluation:

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. Hence, we want to compare different algorithms, selecting the best-performing one as well as the best-performing model from the algorithm's hypothesis space.

9.6 Resampling

The first section of these lecture notes document introduced the prediction accuracy or error measures of classification models. To compute the classification error or accuracy on a dataset S, we defined the following equation:

$$\text{ERR}_S = \frac{1}{n}\sum_{i=1}^{n} L\big(\hat{y}^{[i]}, y^{[i]}\big) = 1 - \text{ACC}_S. \tag{15}$$

Here, $L(\cdot)$ represents the 0-1 loss, which is computed from a predicted class label ($\hat{y}^{[i]}$) and a true class label ($y^{[i]}$) for $i = 1, ..., n$ in dataset S:

$$L(\hat{y}^{[i]}, y^{[i]}) = \begin{cases} 0 & \text{if } \hat{y}^{[i]} = y^{[i]} \\ \\ 1 & \text{if } \hat{y}^{[i]} \neq y^{[i]}. \end{cases} \tag{16}$$

In essence, the classification error is simply the count of incorrect predictions divided by the number of samples in the dataset. Vice versa, we compute the prediction accuracy as the number of correct predictions divided by the number of samples.

Note that the concepts presented in this section also apply to other types of supervised learning, such as regression analysis. To use the resampling methods presented in the following sections for regression models, we swap the accuracy or error computation by, for example, the mean squared error (MSE):

$$\text{MSE}_S = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{[i]} - y^{[i]})^2. \tag{17}$$

As we learned in Section 9.1, performance estimates may suffer from bias and variance, and we are interested in finding a good trade-off. For instance, the resubstitution evaluation (fitting a model to a training set and using the same training set for model evaluation) is heavily optimistically biased. Vice versa, withholding a large portion of the dataset as a test set may lead to pessimistically biased estimates. While reducing the size of the test set may decrease this pessimistic bias, the variance of a performance estimates will most likely increase. An intuitive illustration of the relationship between bias and variance is given in Figure 3. This section will introduce alternative resampling methods for finding a good balance between bias and variance for model evaluation and selection.

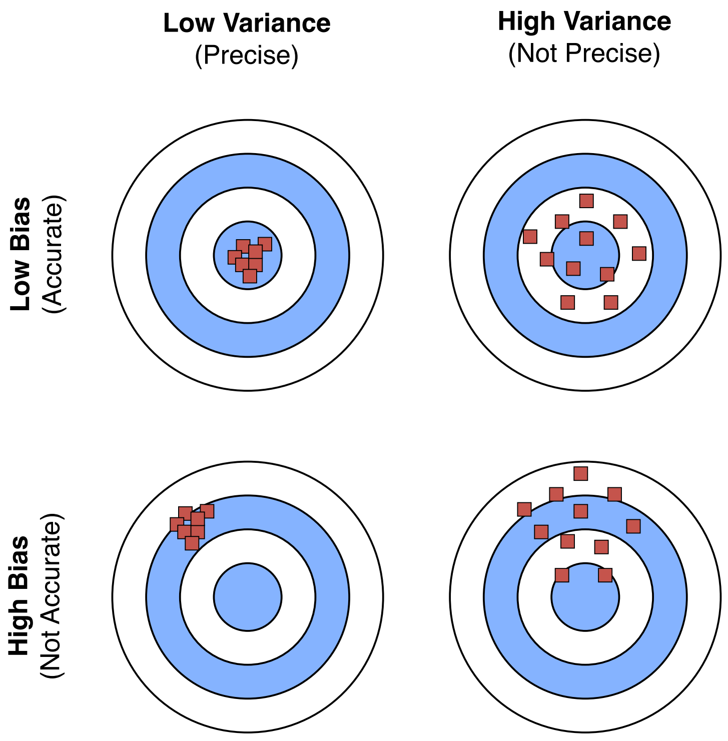

Figure 3: Illustration of bias and variance.

The reason why a proportionally large test sets increase the pessimistic bias is that the model may not have reached its full capacity, yet. In other words, the learning algorithm could have formulated a more powerful, more generalizable hypothesis for classification if it had seen more data. To demonstrate this effect, Figure 4 shows learning curves of a softmax classifiers[5], which were fitted to small subsets of the MNIST[6] dataset.

[5]A softmax classifier is another term for multinomial logistic regression; if we have time, we will discuss it later in this course. I initially picked it for computational simplicity here, but for alternative examples with random forests and k-Nearest Neighbors, please see the accompanying code notebook.

[6]http://yann.lecun.com/exdb/mnist

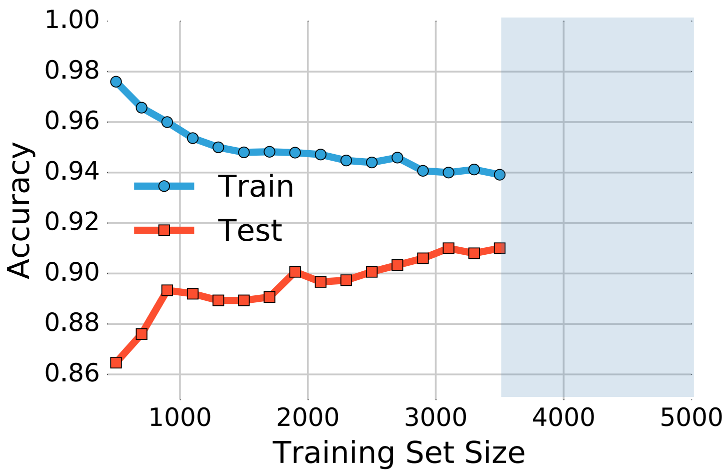

Figure 4: Learning curves of softmax classifiers fit to MNIST.

To generate the learning curves shown in Figure 4, 500 random samples of each of the ten classes from MNIST – instances of the handwritten digits 0 to 9 – were drawn. The 5000-sample MNIST subset was then randomly divided into a 3500-sample training subset and a test set containing 1500 samples while keeping the class proportions intact via stratification. Finally, even smaller subsets of the 3500-sample training set were produced via randomized, stratified splits, and these subsets were used to fit softmax classifiers and the same 1500-sample test set was used to evaluate their performances (samples may overlap between these training subsets). Looking at the plot above, we can see two distinct trends. First, the resubstitution accuracy (training set) declines as the number of training samples grows. Second, we observe an improving generalization accuracy (test set) with an increasing training set size. These trends can likely be attributed to a reduction in overfitting. If the training set is small, the algorithm is more likely picking up noise in the training set so that the model fails to generalize well to data that it has not seen before. This observation also explains the pessimistic bias of the holdout method: A training algorithm may benefit from more training data, data that was withheld for testing. Thus, after we evaluated a model, we may want to run the learning algorithm once again on the complete dataset before we use it in a real-world application.

Now, that we established the point of pessimistic biases for disproportionally large test sets, we may ask whether it is a good idea to decrease the size of the test set. Decreasing the size of the test set brings up another problem: It may result in a substantial variance of our model's performance estimate. The reason is that it depends on which instances end up in training set, and which particular instances end up in test set. Keeping in mind that each time we resample a dataset, we alter the statistics of the distribution of the sample. Most supervised learning algorithms for classification and regression as well as the performance estimates operate under the assumption that a dataset is representative of the population that this dataset sample has been drawn from. As discussed in Section 9.2.1, stratification helps with keeping the sample proportions intact upon splitting a dataset. However, the change in the underlying sample statistics along the features axes is still a problem that becomes more pronounced if we work with small datasets, which is illustrated in Figure 5.

9.7 Repeated Holdout Validation

One way to obtain a more robust performance estimate that is less variant to how we split the data into training and test sets is to repeat the holdout method k times with different random seeds and compute the average performance over these k repetitions:

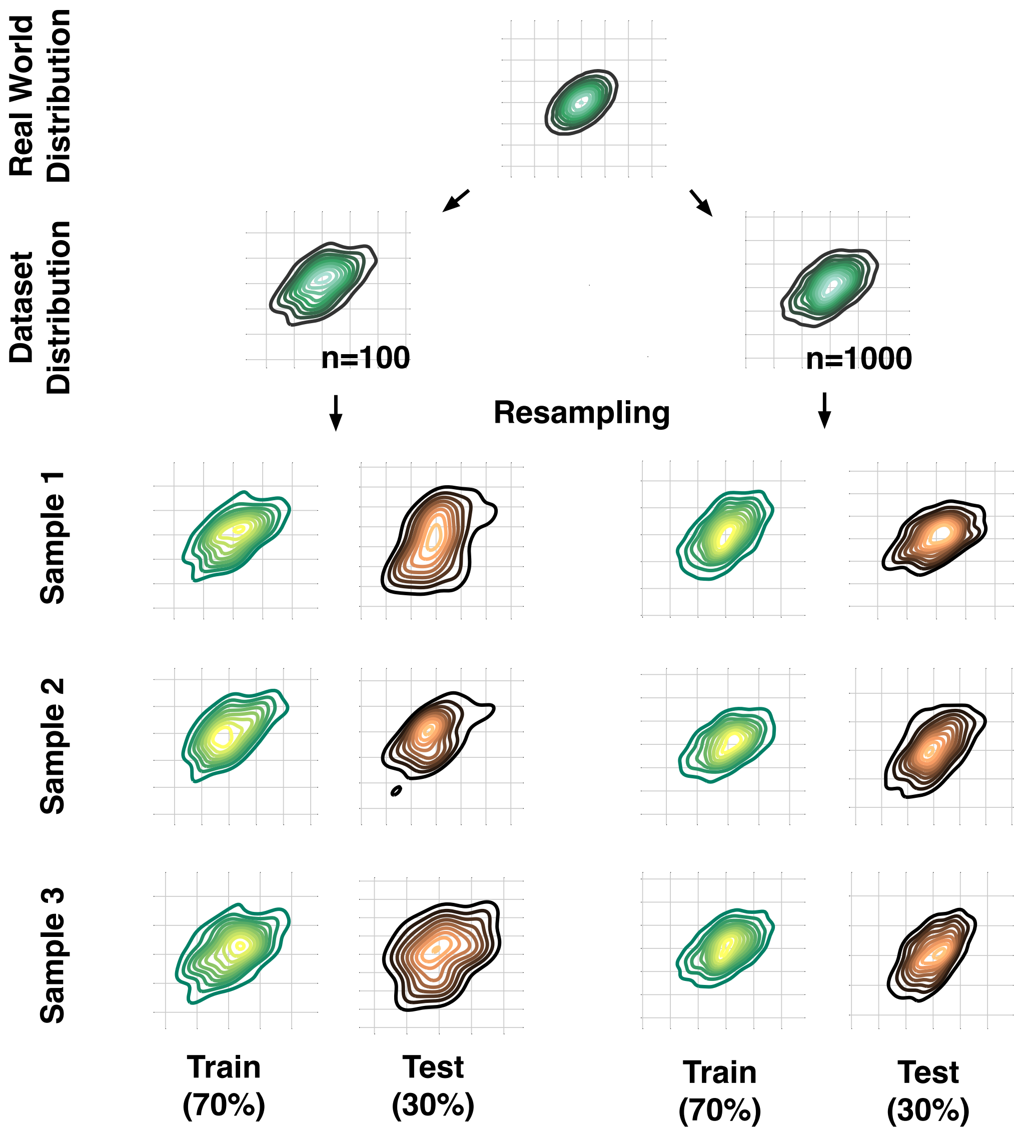

Figure 5: Repeated subsampling from a two-dimensional Gaussian distribution.

$$\text{ACC}_{avg} = \frac{1}{k} \sum_{j=1}^{k} ACC_j, \tag{18}$$

where ACC_j is the accuracy estimate of the jth test set of size m,

$$\text{ACC}_j = 1 - \frac{1}{m} \sum_{i=1}^{m} L\big(\hat{y}^{[i]}, y^{[i]}\big). \tag{19}$$

This repeated holdout procedure, sometimes also called Monte Carlo Cross-Validation, provides a better estimate of how well our model may perform on a random test set, compared to the standard holdout validation method. Also, it provides information about the model's stability – how the model, produced by a learning algorithm, changes with different training set splits. Figure 6 shall illustrate how repeated holdout validation may look like for different training-test split using the Iris dataset to fit to 3-nearest neighbors classifiers.

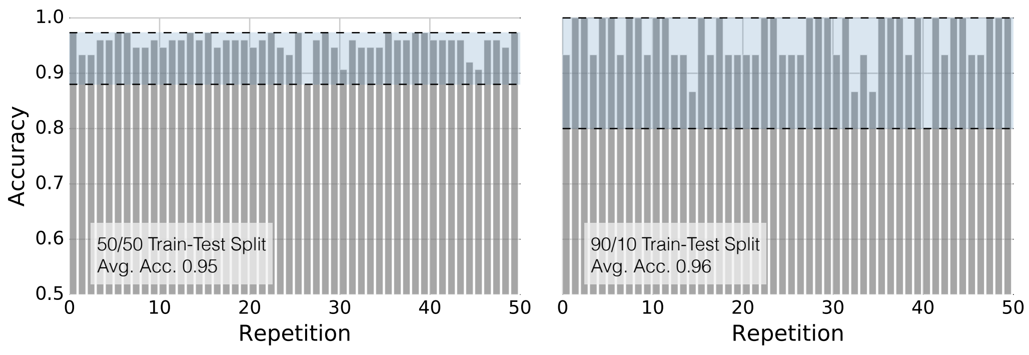

Figure 6: Repeated holdout validation with 3-nearest neighbor classifiers fit to the Iris dataset.

The left subplot in Figure 6 was generated by performing 50 stratified training/test splits with 75 samples in the test and training set each; a 3-nearest neighbors model was fit to the training set and evaluated on the test set in each repetition. The average accuracy of these 50 50/50 splits was 95%. The same procedure was used to produce the right subplot in Figure 6. Here, the test sets consisted of only 15 samples each due to the 90/10 splits, and the average accuracy over these 50 splits was 96%. Figure 6 demonstrates two of the points that were previously discussed. First, we see that the variance of our estimate increases as the size of the test set decreases. Second, we see a small increase in the pessimistic bias when we decrease the size of the training set – we withhold more training data in the 50/50 split, which may be the reason why the average performance over the 50 splits is slightly lower compared to the 90/10 splits.

The next section introduces an alternative method for evaluating a model's performance; the next section will go over different flavors of the bootstrap method that are commonly used to infer the uncertainty of a performance estimate.

9.8 The Bootstrap Method and Empirical Confidence Intervals

The previous examples of Monte Carlo Cross-Validation may have convinced us that repeated holdout validation could provide us with a more robust estimate of a model's performance on random test sets compared to an evaluation based on a single train/test split via holdout validation (Section 9.3). In addition, the repeated holdout may give us an idea about the stability of our model. This section explores an alternative approach to model evaluation and for estimating uncertainty using the bootstrap method.

Let us assume that we would like to compute a confidence interval around a performance estimate to judge its certainty – or uncertainty. How can we achieve this if our sample has been drawn from an unknown distribution? Maybe we could use the sample mean as a point estimate of the population mean, but how would we compute the variance or confidence intervals around the mean if its distribution is unknown? Sure, we could collect multiple, independent samples; this is a luxury we often do not have in real world applications, though. Now, the idea behind the bootstrap is to generate "new samples" by sampling from an empirical distribution. As a side note, the term "bootstrap" likely originated from the phrase "to pull oneself up by one's bootstraps:"

> Circa 1900, to pull (oneself) up by (one's) bootstraps was used figuratively of an impossible task (Among the "practical questions" at the end of chapter one

of Steele's "Popular Physics" schoolbook (1888) is "Why can not a man lift himself by pulling up on his boot-straps?". By 1916 its meaning expanded to include "better oneself by rigorous, unaided effort." The meaning "fixed sequence of instructions to load the operating system of a computer" (1953) is from the notion of the first-loaded program pulling itself, and the rest, up by the bootstrap.

[Source: Online Etymology Dictionary[7]]

The bootstrap method is a resampling technique for estimating a sampling distribution, and in the context of this lecture, we are particularly interested in estimating the uncertainty of a performance estimate – the prediction accuracy or error. The bootstrap method was introduced by Bradley Efron in 1979[8]. About 15 years later, Bradley Efron and Robert Tibshirani even devoted a whole book to the bootstrap method, "An Introduction to the Bootstrap"[9], which is a highly recommended read for everyone who is interested in more details on this topic. In brief, the idea of the bootstrap method is to generate new data from a population by repeated sampling from the original dataset with replacement – in contrast, the repeated holdout method can be understood as sampling without replacement. Walking through it step by step, the bootstrap method works like this:

1. We are given a dataset of size n.
2. For b bootstrap rounds:

   We draw one single instance from this dataset and assign it to the jth bootstrap sample. We repeat this step until our bootstrap sample has size n – the size of the original dataset. Each time, we draw samples from the same original dataset such that certain examples may appear more than once in a bootstrap sample and some not at all.
3. We fit a model to each of the b bootstrap samples and compute the resubstitution accuracy.
4. We compute the model accuracy as the average over the b accuracy estimates (Equation 20):

$$\mathrm{ACC}_{boot} = \frac{1}{b}\sum_{j=1}^{b}\frac{1}{n}\sum_{i=1}^{n}\Big(1 - L\big(\hat{y}^{[i]}, y^{[i]}\big)\Big). \tag{20}$$

Originally, the bootstrap method aimed to determine the statistical properties of an estimator when the underlying distribution was unknown and additional samples are not available. So, in order to exploit this method for the evaluation of predictive models, such as hypotheses for classification and regression, we may prefer a slightly different approach to bootstrapping using the so-called Leave-One-Out Bootstrap (LOOB) technique. Here, we use out-of-bag samples as test sets for evaluation instead of evaluating the model on the training data. Out-of-bag samples are the unique sets of instances that are not used for model fitting as shown in Figure 7.

Figure 7 illustrates how three random bootstrap samples drawn from an exemplary ten-sample dataset $(x_1, x_2, ..., x_{10})$ and how the out-of-bag sample might look like. In practice, Bradley Efron and Robert Tibshirani recommend drawing 50 to 200 bootstrap samples as being sufficient for producing reliable estimates[10].

[7] https://www.etymonline.com/word/bootstrap

[8] Bradley Efron. "Bootstrap methods: another look at the jackknife". In: *Breakthroughs in Statistics*. Springer, 1992, pp. 569–593.

[9] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[10] Efron and Tibshirani, *An introduction to the bootstrap*.

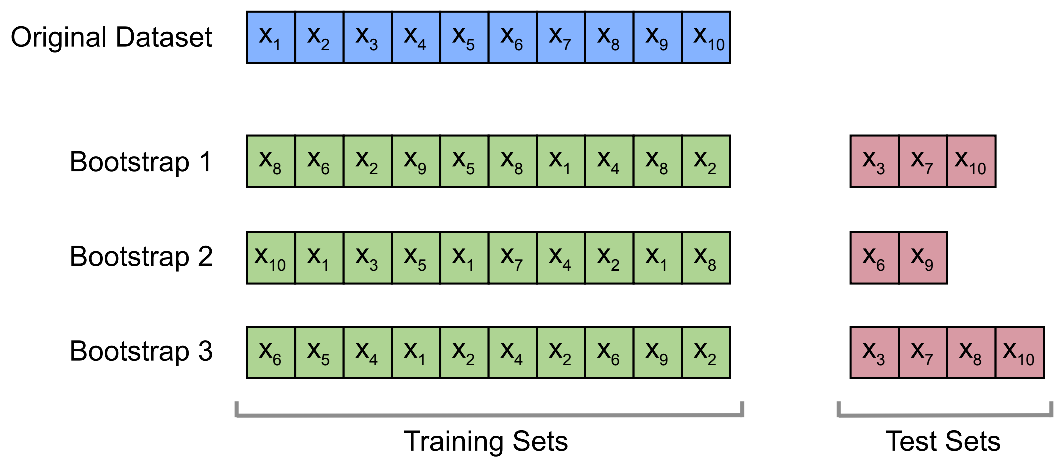
Original Dataset
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
Bootstrap 1
x8 x6 x2 x9 x5 x8 x1 x4 x8 x2
x3 x7 x10
Bootstrap 2
x10 x1 x3 x5 x1 x7 x4 x2 x1 x8
x6 x9
Bootstrap 3
x6 x5 x4 x1 x2 x4 x2 x6 x9 x2
x3 x7 x8 x10
Training Sets
Test Sets

Figure 7: Illustration of training and test data splits in the Leave-One-Out Bootstrap (LOOB).

Taking a step back, let us assume that a sample that has been drawn from a normal distribution. Using basic concepts from statistics, we use the sample mean $\bar{x}$ as a point estimate of the population mean μ:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x^{[i]}. \tag{21}$$

Similarly, the variance σ^2 is estimated as follows:

$$\text{VAR} = \frac{1}{n-1} \sum_{i=1}^{n} (x^{[i]} - \bar{x})^2. \tag{22}$$

Consequently, the standard error (SE) is computed as the standard deviation's estimate (SD $\approx \sigma$) divided by the square root of the sample size:

$$\text{SE} = \frac{\text{SD}}{\sqrt{n}}. \tag{23}$$

Using the standard error we can then compute a 95% confidence interval of the mean according to

$$\bar{x} \pm z \times \frac{\sigma}{\sqrt{n}}, \tag{24}$$

such that

$$\bar{x} \pm t \times \text{SE}, \tag{25}$$

with $z = 1.96$ for the 95 % confidence interval. Since SD is the standard deviation of the population (σ) estimated from the sample, we have to consult a t-table to look up the actual value of t, which depends on the size of the sample – or the *degrees of freedom* to be precise. For instance, given a sample with $n = 100$, we find that $t_{95} = 1.984$.

Similarly, we can compute the 95% confidence interval of the bootstrap estimate starting with the mean accuracy,

$$\text{ACC}_{boot} = \frac{1}{b}\sum_{i=1}^{b}\text{ACC}_i, \tag{26}$$

and use it to compute the standard error

$$\text{SE}_{boot} = \sqrt{\frac{1}{b-1}\sum_{i=1}^{b}(\text{ACC}_i - \text{ACC}_{boot})^2}. \tag{27}$$

Here, ACC_i is the value of the statistic (the estimate of ACC) calculated on the ith bootstrap replicate. And the standard deviation of the values $\text{ACC}_1, \text{ACC}_1, ..., \text{ACC}_b$ is the estimate of the standard error of ACC[11].

Finally, we can then compute the confidence interval around the mean estimate as

$$\text{ACC}_{boot} \pm t \times \text{SE}_{boot}. \tag{28}$$

Although the approach outlined above seems intuitive, what can we do if our samples do not follow a normal distribution? A more robust, yet computationally straight-forward approach is the percentile method as described by B. Efron[12]. Here, we pick the lower and upper confidence bounds as follows:

- $\text{ACC}_{lower} = \alpha_1$th percentile of the ACC_{boot} distribution
- $\text{ACC}_{upper} = \alpha_2$th percentile of the ACC_{boot} distribution,

where $\alpha_1 = \alpha$ and $\alpha_2 = 1 - \alpha$, and α is the degree of confidence for computing the $100 \times (1 - 2 \times \alpha)$ confidence interval. For instance, to compute a 95% confidence interval, we pick $\alpha = 0.025$ to obtain the the 2.5th and 97.5th percentiles of the b bootstrap samples distribution as our upper and lower confidence bounds.

In practice, if the data is indeed (roughly) following a normal distribution, the "standard" confidence interval and percentile method typically agree as illustrated in the Figure 8.

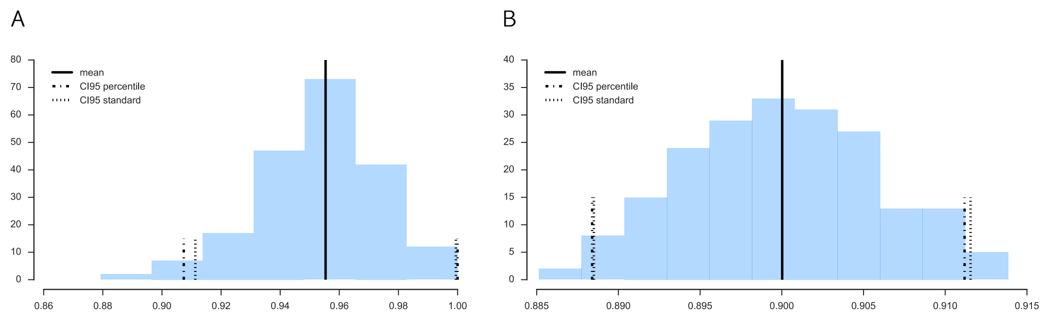

Figure 8: Comparison of the standard and percentile method for computing confidence intervals from leave-one-out bootstrap samples. Subpanel A evaluates 3-nearest neighbors models on Iris, and sublpanel B shows the results of softmax regression models on MNIST.

[11] Efron and Tibshirani, *An introduction to the bootstrap*.

[12] Bradley Efron. "Nonparametric standard errors and confidence intervals". In: *Canadian Journal of Statistics* 9.2 (1981), pp. 139–158.

In 1983, Bradley Efron described the *.632 Estimate*, a further improvement to address the pessimistic bias of the bootstrap cross-validation approach described above[13]. The pessimistic bias in the "classic" bootstrap method can be attributed to the fact that the bootstrap samples only contain approximately 63.2% of the unique examples from the original dataset. For instance, we can compute the probability that a given example from a dataset of size n is not drawn as a bootstrap sample as follows:

$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^n, \tag{29}$$

which is asymptotically equivalent to $\frac{1}{e} \approx 0.368$ as $n \to \infty$.

Vice versa, we can then compute the probability that a sample *is* chosen as

$$P(\text{chosen}) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632 \tag{30}$$

for reasonably large datasets, so that we select approximately $0.632 \times n$ unique examples as bootstrap training sets and reserve $0.382 \times n$ out-of-bag examples for testing in each iteration, which is illustrated in Figure 9.

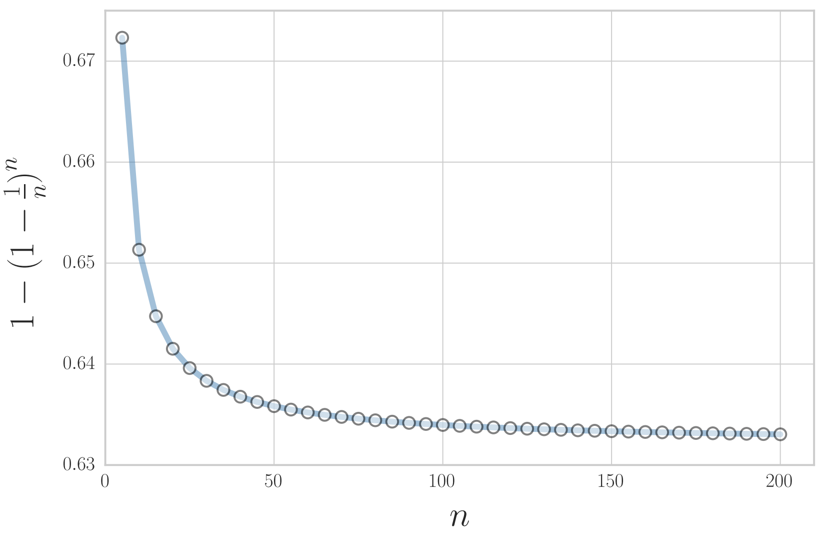

Figure 9: Probability of including an example from the dataset in a bootstrap sample for different dataset sizes n.

Now, to address the bias that is due to this the sampling with replacement, Bradley Efron proposed the *.632 Estimate* mentioned earlier, which is computed via the following equation:

$$\text{ACC}_{boot} = \frac{1}{b} \sum_{i=1}^{b} \big(0.632 \cdot \text{ACC}_{h,i} + 0.368 \cdot \text{ACC}_{r,i}\big), \tag{31}$$

where $\text{ACC}_{r,i}$ is the resubstitution accuracy, and $\text{ACC}_{h,i}$ is the accuracy on the out-of-bag sample. Now, while the .632 Boostrap attempts to address the pessimistic bias of the estimate, an optimistic bias may occur with models that tend to overfit so that Bradley

[13]Bradley Efron. "Estimating the error rate of a prediction rule: improvement on cross-validation". In: *Journal of the American Statistical Association* 78.382 (1983), pp. 316–331.

Efron and Robert Tibshirani proposed *The .632+ Bootstrap Method*[14]. Instead of using a fixed weight $\omega = 0.632$ in

$$ACC_{\text{boot}} = \frac{1}{b} \sum_{i=1}^{b} \big(\omega \cdot \text{ACC}_{h,i} + (1 - \omega) \cdot \text{ACC}_{r,i} \big), \tag{32}$$

we compute the weight ω as

$$\omega = \frac{0.632}{1 - 0.368 \times R}, \tag{33}$$

where R is the *relative overfitting rate*:

$$R = \frac{(-1) \times (\text{ACC}_{h,i} - \text{ACC}_{r,i})}{\gamma - (1 - \text{ACC}_{h,i})}. \tag{34}$$

(Since we are plugging ω into Equation 32 for computing $\text{ACC}_{b}oot$ that we defined above, $\text{ACC}_{h,i}$ and $\text{ACC}_{r,i}$ still refer to the resubstitution and out-of-bag accuracy estimates in the ith bootstrap round, respectively.)

Further, we need to determine the *no-information rate* γ in order to compute R. For instance, we can compute γ by fitting a model to a dataset that contains all possible combinations between samples $x'^{[i]}$ and target class labels $y^{[i]}$ – we pretend that the observations and class labels are independent:

$$\gamma = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{i'=1}^{n} L(y_i, f(x_{i'})). \tag{35}$$

Alternatively, we can estimate the no-information rate γ as follows:

$$\gamma = \sum_{k=1}^{K} p_k (1 - q_k), \tag{36}$$

where p_k is the proportion of class k examples observed in the dataset, and q_k is the proportion of class k examples that the classifier predicts in the dataset.

9.9 Conclusions

This lecture continued the discussion around biases and variances in evaluating machine learning models in more detail. Further, it introduced the repeated hold-out method that may provide us with some further insights into a model's stability. Then, we looked at the bootstrap method; a technique borrowed from the field of statistics. We explored different flavors of this bootstrap method that help us estimate the uncertainty of our performance estimates. After covering the basics of model evaluation in this and the previous section, the next lecture introduces hyperparameter tuning and model selection.

[14]Bradley Efron and Robert Tibshirani. "Improvements on cross-validation: the .632+ bootstrap method". In: *Journal of the American Statistical Association* 92.438 (1997), pp. 548–560.