Lecture 09

Model Evaluation 2: Confidence Intervals and Resampling Methods

STAT 451: Machine Learning, Fall 2020

Sebastian Raschka

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

1. Introduction

2. Holdout method for model evaluation

3. Holdout method for model selection

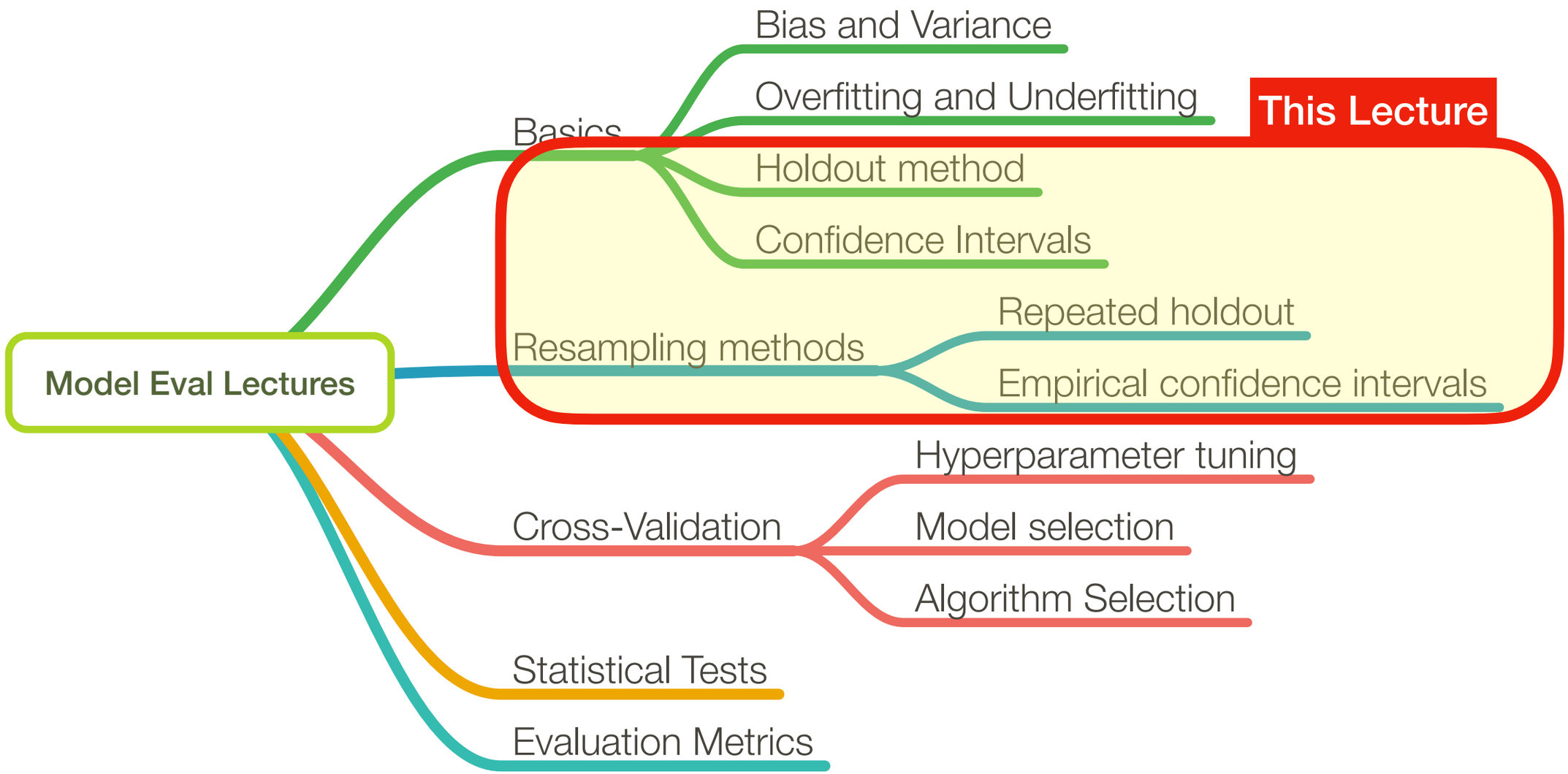
4. Confidence intervals -- normal approximation

5. Resampling & repeated holdout

6. Empirical confidence intervals via Bootstrap

7. The 0.632 and 0.632+ Bootstrap

Overview


Model Eval Lectures
Basics
Bias and Variance
Overfitting and Underfitting
Holdout method
Confidence Intervals
This Lecture
Resampling methods
Repeated holdout
Empirical confidence intervals
Cross-Validation
Hyperparameter tuning
Model selection
Algorithm Selection
Statistical Tests
Evaluation Metrics

Main points why we evaluate the predictive performance of a model:

Main points why we evaluate the predictive performance of a model:

1. Want to estimate the generalization performance, the predictive performance of our model on future (unseen) data.

Main points why we evaluate the predictive performance of a model:

1. Want to estimate the generalization performance, the predictive performance of our model on future (unseen) data.

2. Want to increase the predictive performance by tweaking the learning algorithm and selecting the best performing model from a given hypothesis space.

Main points why we evaluate the predictive performance of a model:

1. Want to estimate the generalization performance, the predictive performance of our model on future (unseen) data.

2. Want to increase the predictive performance by tweaking the learning algorithm and selecting the best performing model from a given hypothesis space.

3. Want to identify the ML 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.

Some unfortunate facts about test sets

- Training set error is an optimistically biased estimator of the generalization error

- Test set error is an unbiased estimator of the generalization error (test sample and hypothesis chosen independently)

- (In practice, the test set error is actually pessimistically biased; why?)

1. Introduction

2. Holdout method for model evaluation

3. Holdout method for model selection

4. Confidence intervals -- normal approximation

5. Resampling & repeated holdout

6. Empirical confidence intervals via Bootstrap

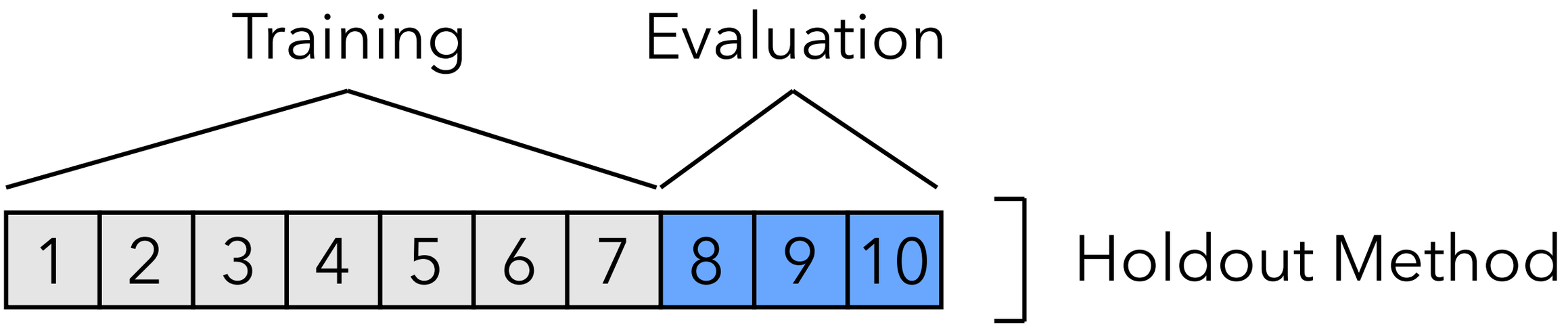
7. The 0.632 and 0.632+ Bootstrap

Some unfortunate facts about test sets

- Training set error is an optimistically biased estimator of the generalization error

- Test set error is an unbiased estimator of the generalization error (test sample and hypothesis chosen independently)

- (in practice, the test set error is actually pessimistically biased; why?)


Training
Evaluation
1
2
3
4
5
6
7
8
9
10
Holdout Method

Often, using the holdout method is not a good idea ...

Often using the holdout method is not a good idea ...

Test set error as generalization error estimator is ____imistically biased (not so bad)

But it does not account for variance in the training data (bad)

Why is pessimistic bias not "so bad"?

Suppose we have the following ranking based on accuracy:

h_2: 75% > h_1: 70% > h_3: 65%,

we would still rank them the same way if we add a 10% pessimistic bias:

h_2: 65% > h_1: 60% > h_3: 55%.

Test set errors can also be optimistically biased

Do CIFAR-10 Classifiers Generalize to CIFAR-10?

Benjamin Recht
UC Berkeley

Rebecca Roelofs
UC Berkeley

Ludwig Schmidt
MIT

Vaishaal Shankar
UC Berkeley

June 4, 2018

Abstract

Machine learning is currently dominated by largely experimental work focused on improvements in a few key tasks. However, the impressive accuracy numbers of the best performing models are questionable because the same test sets have been used to select these models for multiple years now. To understand the danger of overfitting, we measure the accuracy of CIFAR-10 classifiers by creating a new test set of truly unseen images. Although we ensure that the new test set is as close to the original data distribution as possible, we find a large drop in accuracy (4% to 10%) for a broad range of deep learning models. Yet, more recent models with higher original accuracy show a *smaller* drop and better overall performance, indicating that this drop is likely not due to overfitting based on adaptivity. Instead, we view our results as evidence that current accuracy numbers are brittle and susceptible to even minute natural variations in the data distribution.

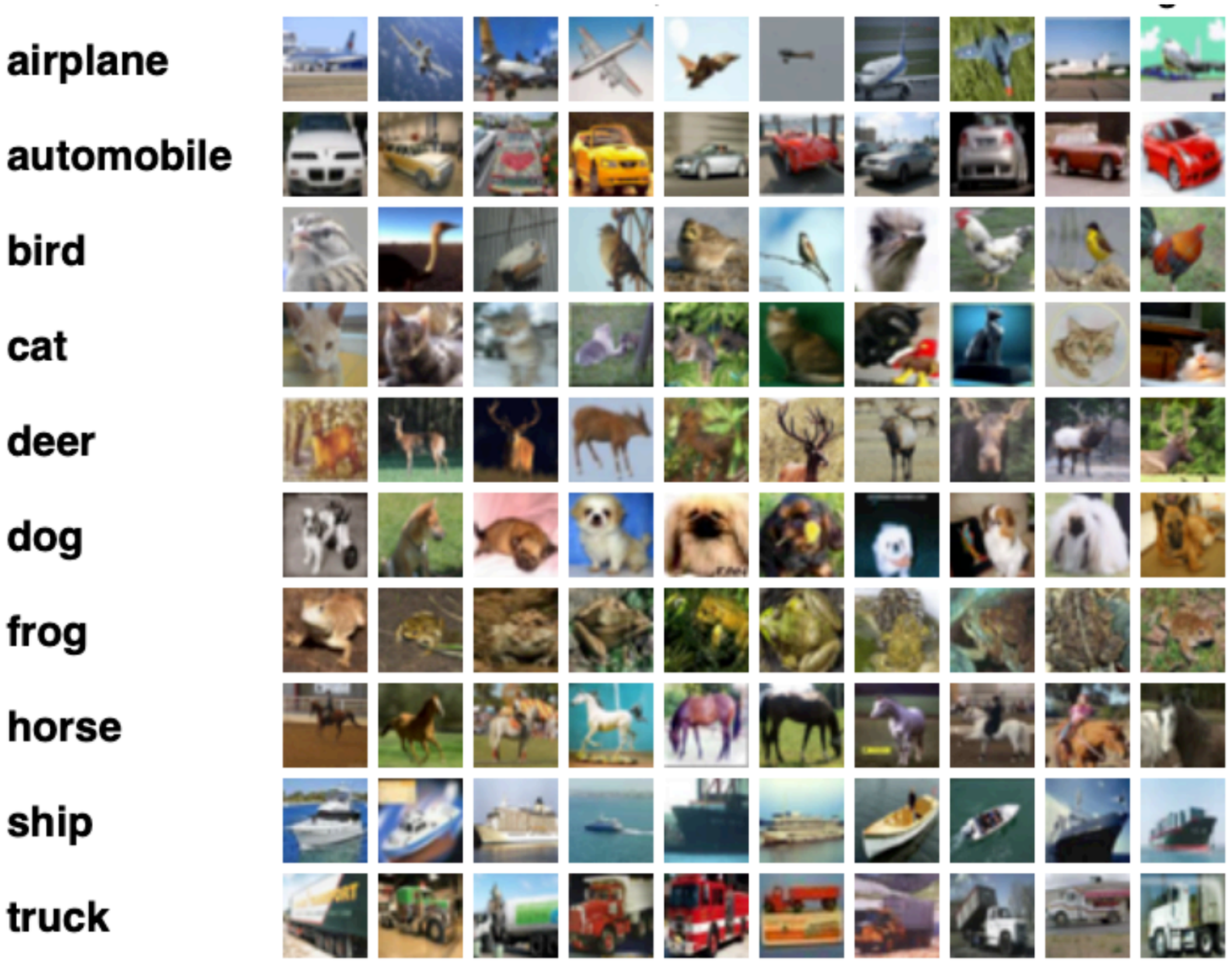
Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. (2018). Do CIFAR-10 classifiers generalize to CIFAR-10?. *arXiv preprint arXiv:1806.00451*.

https://arxiv.org/abs/1806.00451

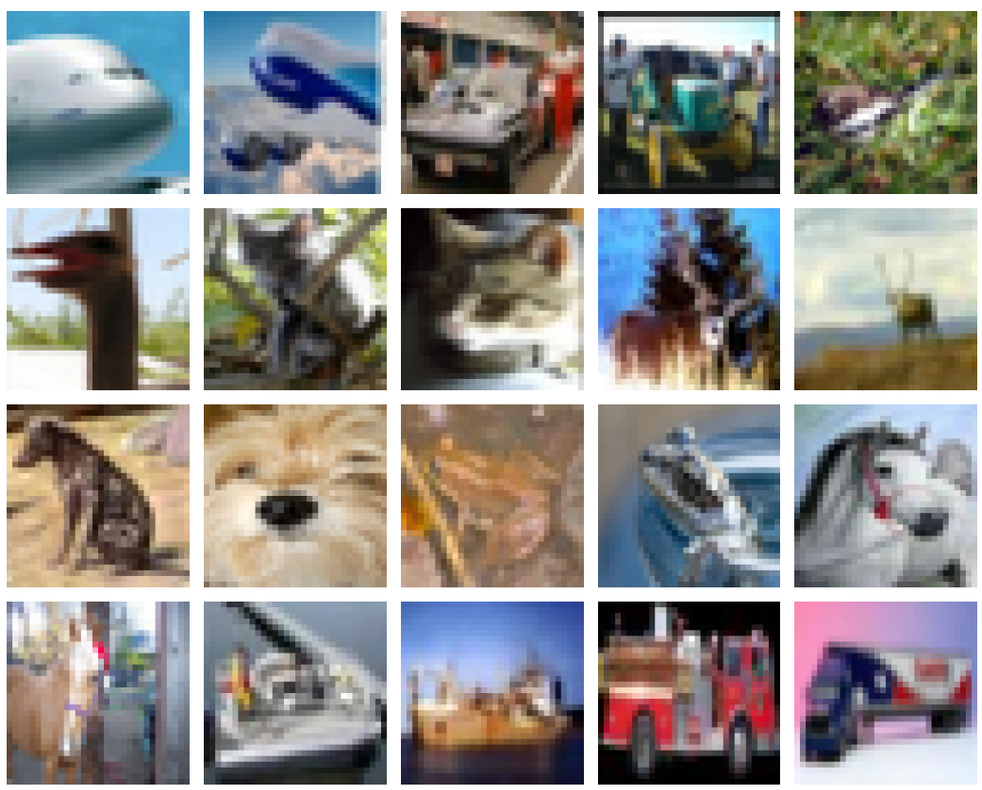
The CIFAR-10 dataset

CIFAR -> Canadian Institute For Advanced Research

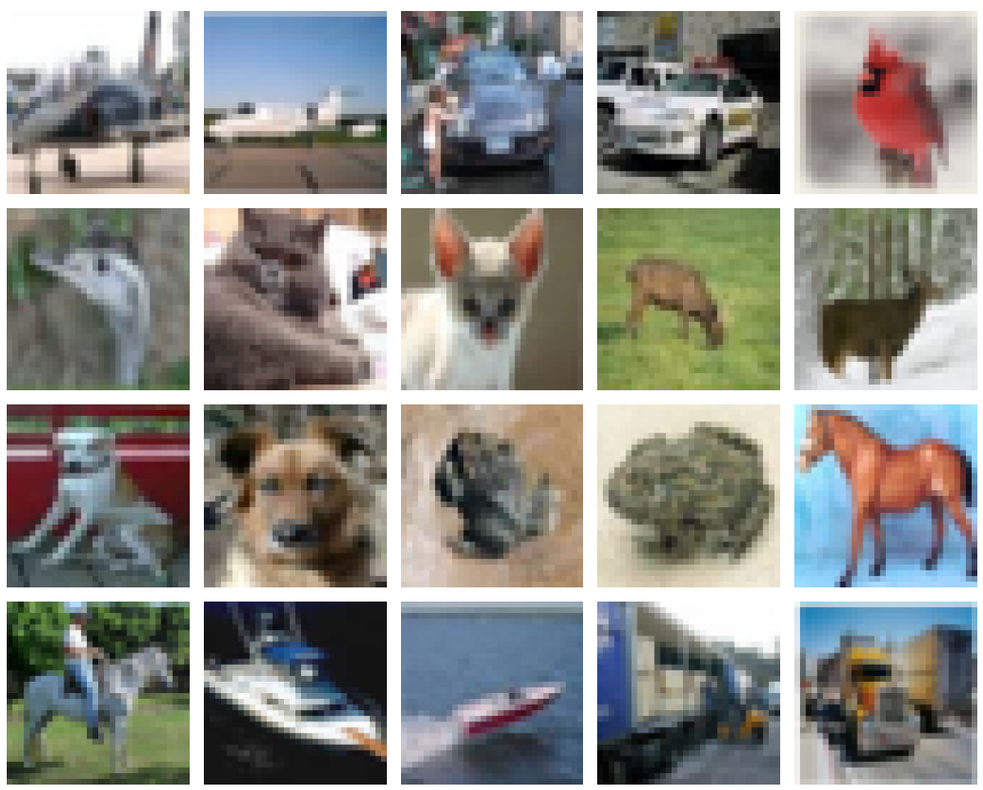
- 60,000 32x32 color images in 10 classes
- 6,000 images per class
- 50,000 training images and 10,000 test images



https://www.cs.toronto.edu/~kriz/cifar.html



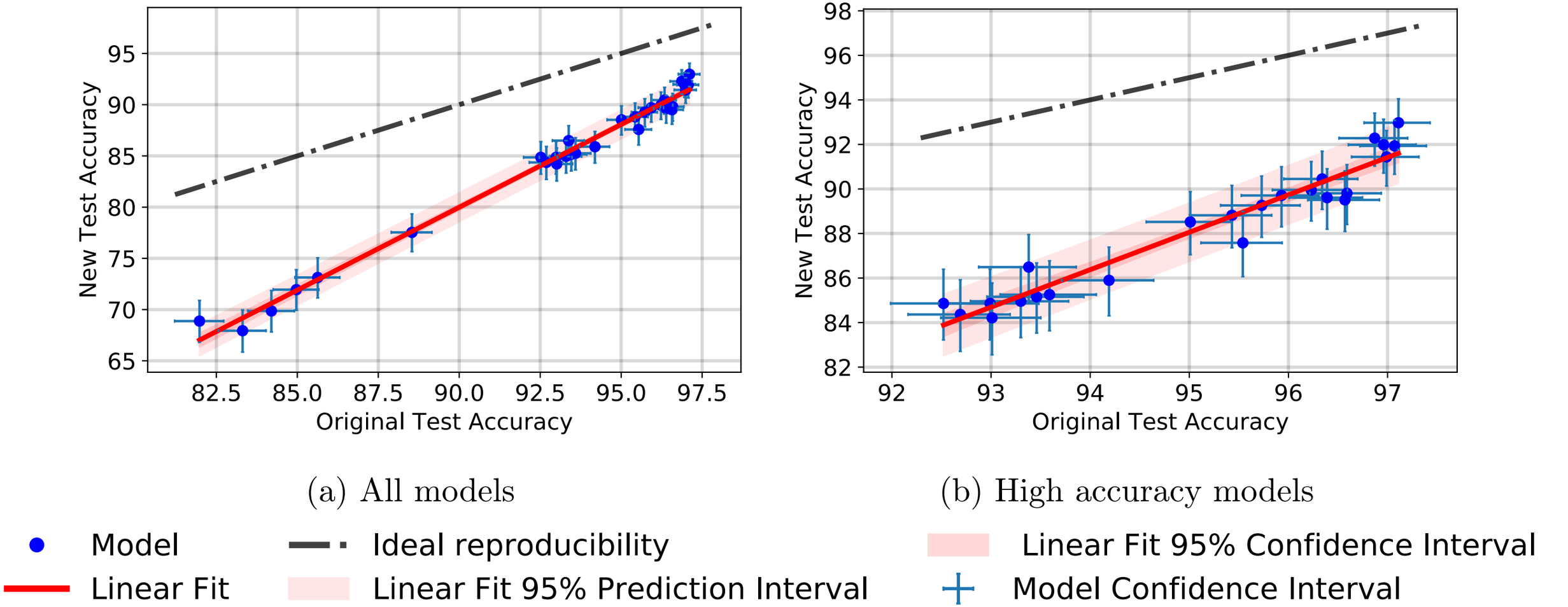
(a) Test Set A



(b) Test Set B

Figure 1: Class-balanced random draws from the new and original test sets.[1]

Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. (2018). Do CIFAR-10 classifiers generalize to CIFAR-10?. *arXiv preprint arXiv:1806.00451*.



(a) All models (b) High accuracy models

Figure 2: Model accuracy on new test set vs. model accuracy on original test set.

Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. (2018). Do CIFAR-10 classifiers generalize to CIFAR-10?. *arXiv preprint arXiv:1806.00451*.

Table 1: Model accuracy on the original CIFAR-10 test set and the new test set, with the gap reported as the difference between the two accuracies. Δ Rank is the relative difference in the ranking from the original test set to the new test set. For example, $\Delta\text{Rank} = -2$ means a model dropped in the rankings by two positions on the new test set.

	Original Accuracy	New Accuracy	Gap	Δ Rank
`shake_shake_64d_cutout` [3, 4]	97.1 [96.8, 97.4]	93.0 [91.8, 94.0]	4.1	0
`shake_shake_96d` [4]	97.1 [96.7, 97.4]	91.9 [90.7, 93.1]	5.1	-2
`shake_shake_64d` [4]	97.0 [96.6, 97.3]	91.4 [90.1, 92.6]	5.6	-2
`wide_resnet_28_10_cutout` [3, 22]	97.0 [96.6, 97.3]	92.0 [90.7, 93.1]	5	+1
`shake_drop` [21]	96.9 [96.5, 97.2]	92.3 [91.0, 93.4]	4.6	+3
`shake_shake_32d` [4]	96.6 [96.2, 96.9]	89.8 [88.4, 91.1]	6.8	-2
`darc` [11]	96.6 [96.2, 96.9]	89.5 [88.1, 90.8]	7.1	-4
`resnext_29_4x64d` [20]	96.4 [96.0, 96.7]	89.6 [88.2, 90.9]	6.8	-2
`pyramidnet_basic_110_270` [6]	96.3 [96.0, 96.7]	90.5 [89.1, 91.7]	5.9	+3
`resnext_29_8x64d` [20]	96.2 [95.8, 96.6]	90.0 [88.6, 91.2]	6.3	+3
`wide_resnet_28_10` [22]	95.9 [95.5, 96.3]	89.7 [88.3, 91.0]	6.2	+2
`pyramidnet_basic_110_84` [6]	95.7 [95.3, 96.1]	89.3 [87.8, 90.6]	6.5	0
`densenet_BC_100_12` [10]	95.5 [95.1, 95.9]	87.6 [86.1, 89.0]	8	-2
`neural_architecture_search` [23]	95.4 [95.0, 95.8]	88.8 [87.4, 90.2]	6.6	+1
`wide_resnet_tf` [22]	95.0 [94.6, 95.4]	88.5 [87.0, 89.9]	6.5	+1
`resnet_v2_bottleneck_164` [8]	94.2 [93.7, 94.6]	85.9 [84.3, 87.4]	8.3	-1
`vgg16_keras` [14, 18]	93.6 [93.1, 94.1]	85.3 [83.6, 86.8]	8.3	-1
`resnet_basic_110` [7]	93.5 [93.0, 93.9]	85.2 [83.5, 86.7]	8.3	-1
`resnet_v2_basic_110` [8]	93.4 [92.9, 93.9]	86.5 [84.9, 88.0]	6.9	+3
`resnet_basic_56` [7]	93.3 [92.8, 93.8]	85.0 [83.3, 86.5]	8.3	0
`resnet_basic_44` [7]	93.0 [92.5, 93.5]	84.2 [82.6, 85.8]	8.8	-3
`vgg_15_BN_64` [14, 18]	93.0 [92.5, 93.5]	84.9 [83.2, 86.4]	8.1	+1
`resnet_preact_tf` [7]	92.7 [92.2, 93.2]	84.4 [82.7, 85.9]	8.3	0
`resnet_basic_32` [7]	92.5 [92.0, 93.0]	84.9 [83.2, 86.4]	7.7	+3
`cudaconvnet` [13]	88.5 [87.9, 89.2]	77.5 [75.7, 79.3]	11	0
`random_features_256k_aug` [2]	85.6 [84.9, 86.3]	73.1 [71.1, 75.1]	12	0
`random_features_32k_aug` [2]	85.0 [84.3, 85.7]	71.9 [69.9, 73.9]	13	0
`random_features_256k` [2]	84.2 [83.5, 84.9]	69.9 [67.8, 71.9]	14	0
`random_features_32k` [2]	83.3 [82.6, 84.0]	67.9 [65.9, 70.0]	15	-1
`alexnet_tf`	82.0 [81.2, 82.7]	68.9 [66.8, 70.9]	13	+1

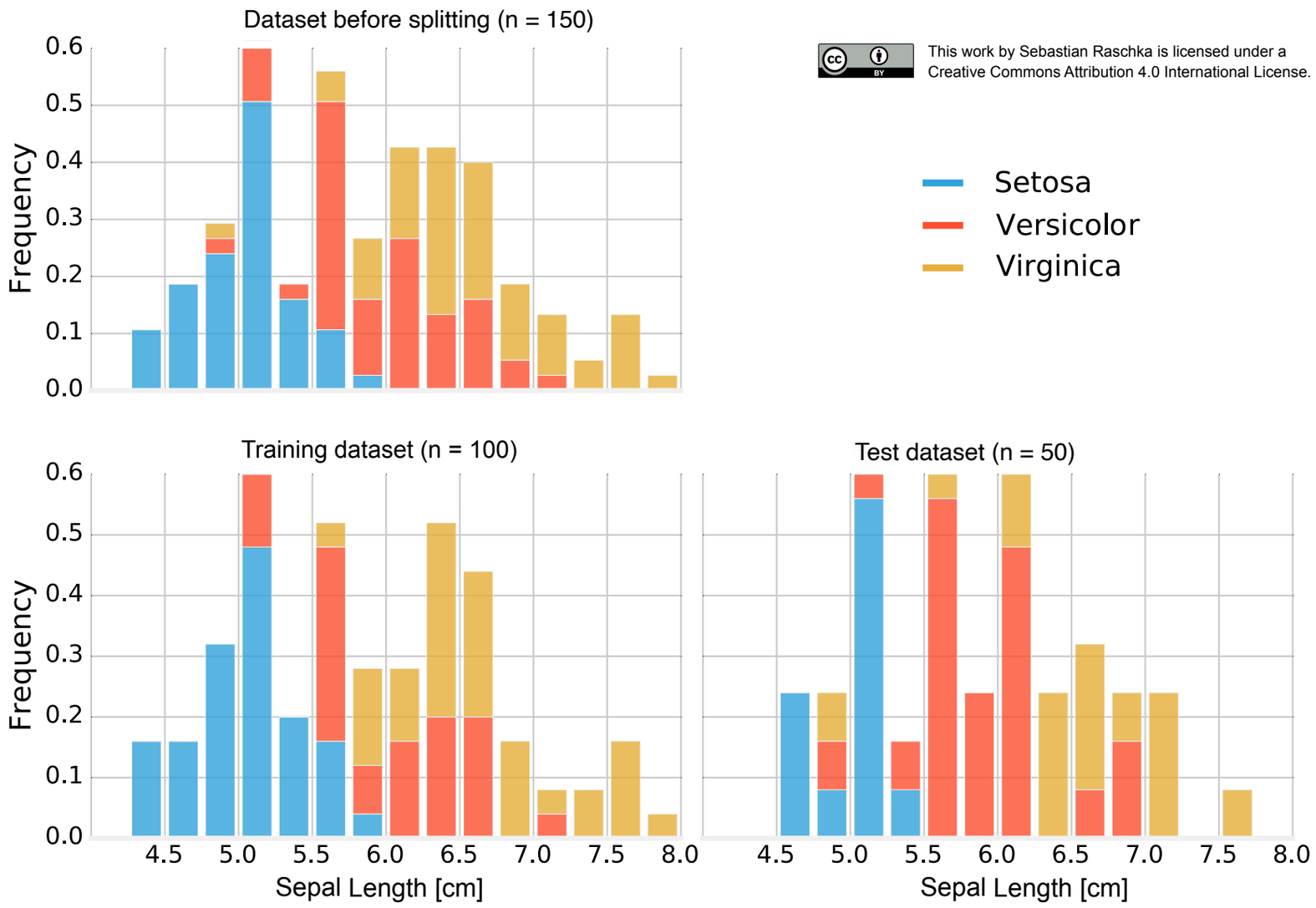
Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. (2018). Do CIFAR-10 classifiers generalize to CIFAR-10?. *arXiv preprint arXiv:1806.00451*.

Often using the holdout method is not a good idea ...

- Test set error as generalization error estimator is pessimistically biased (not so bad)

- Does not account for variance in the training data (bad)

Issues with Subsampling (Independence Violation)



The Iris dataset consists of 50 Setosa, 50 Versicolor, and 50 Virginica flowers; the flower species are distributed uniformly:

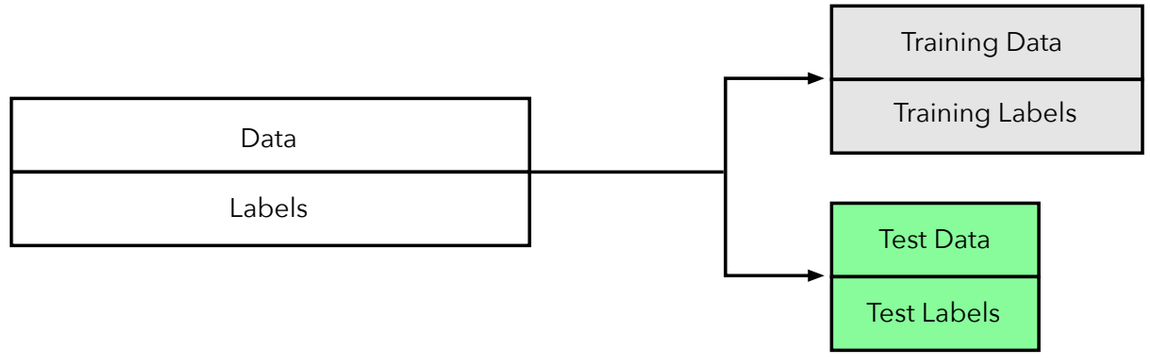
- 33.3% Setosa
- 33.3% Versicolor
- 33.3% Virginia

If our 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:

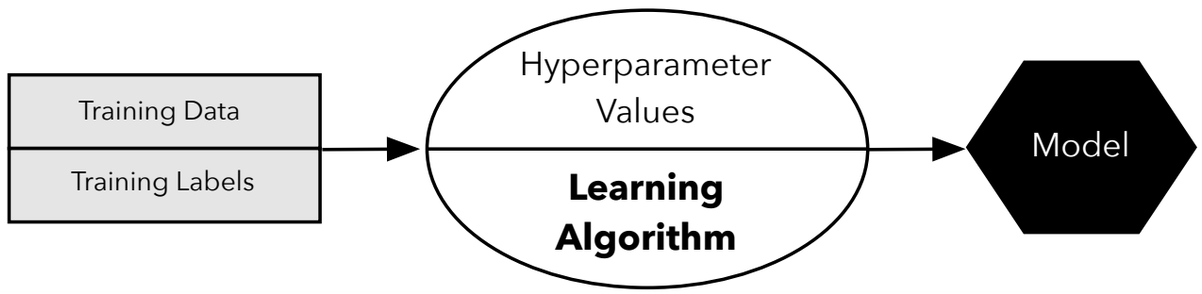
- training set → 38 x Setosa, 28 x Versicolor, 34 x Virginica
- test set → 12 x Setosa, 22 x Versicolor, 16 x Virginica

Holdout evaluation

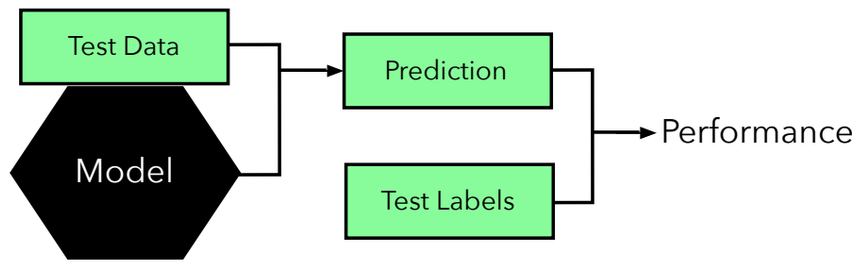
1



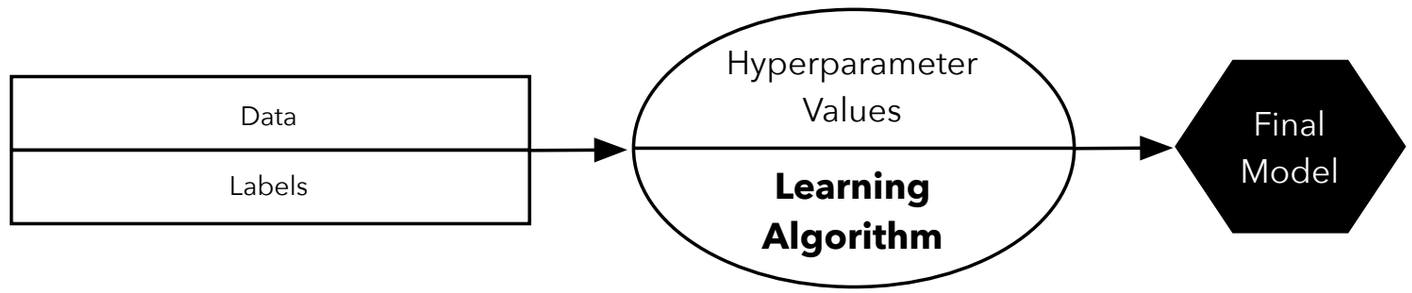
2



3



4



1. Introduction

2. Holdout method for model evaluation

3. Holdout method for model selection

4. Confidence intervals -- normal approximation

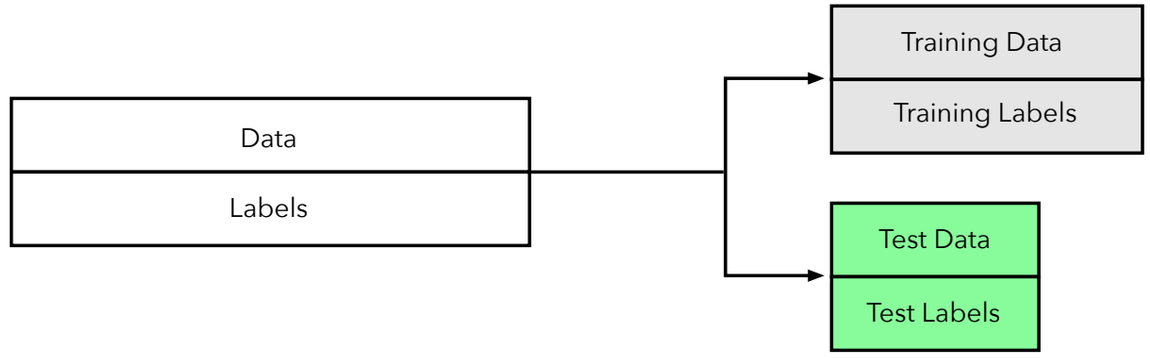
5. Resampling & repeated holdout

6. Empirical confidence intervals via Bootstrap

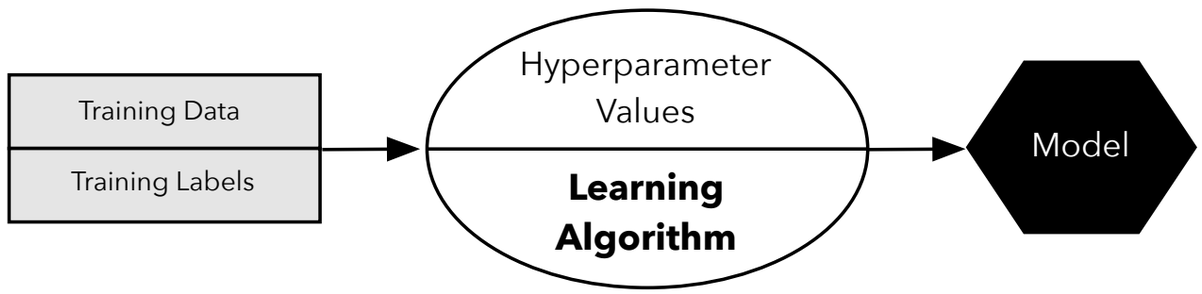
7. The 0.632 and 0.632+ Bootstrap

Holdout evaluation

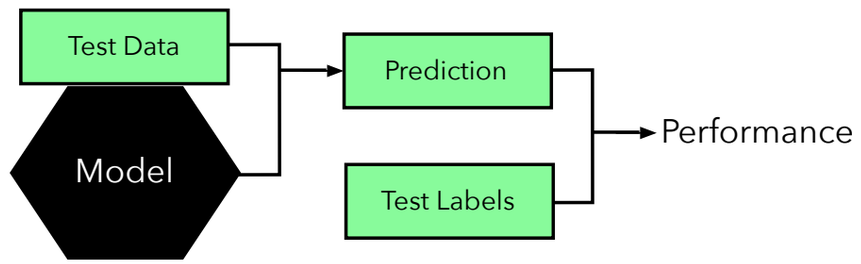
1



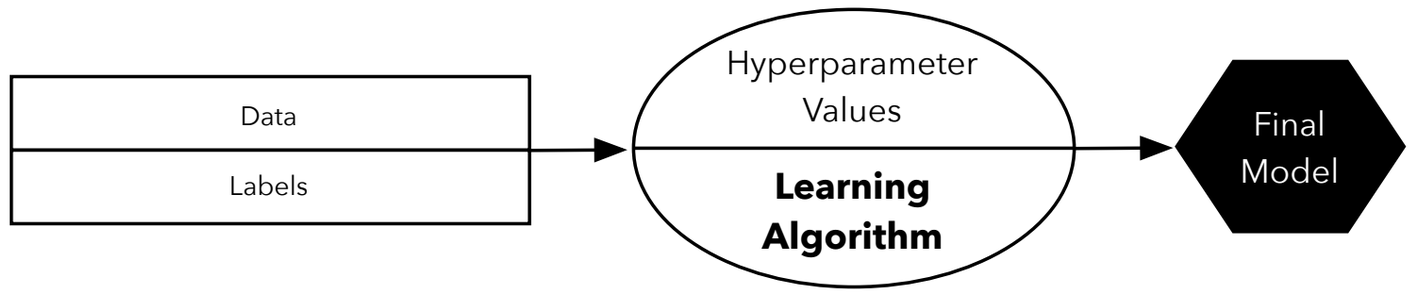
2



3

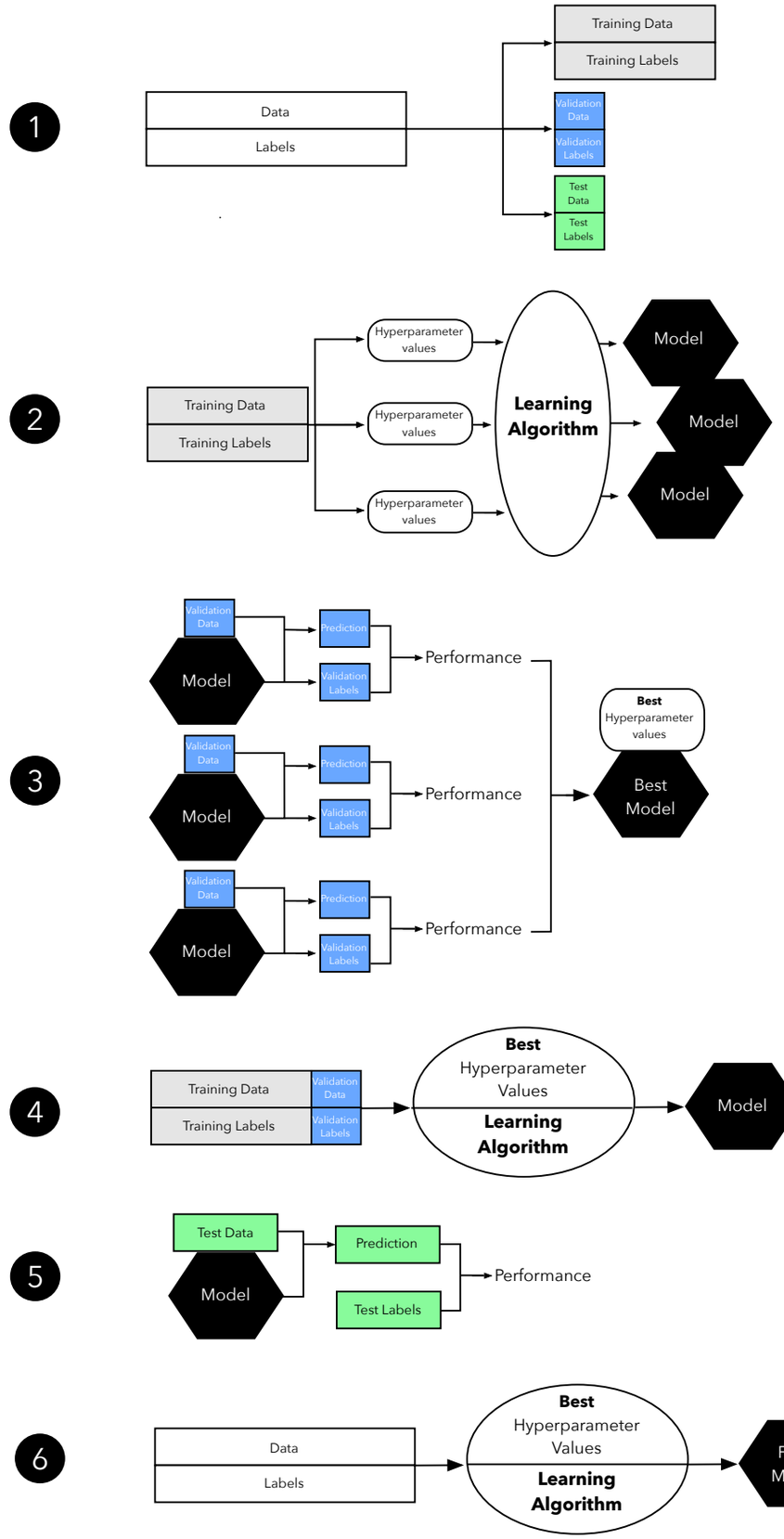


4



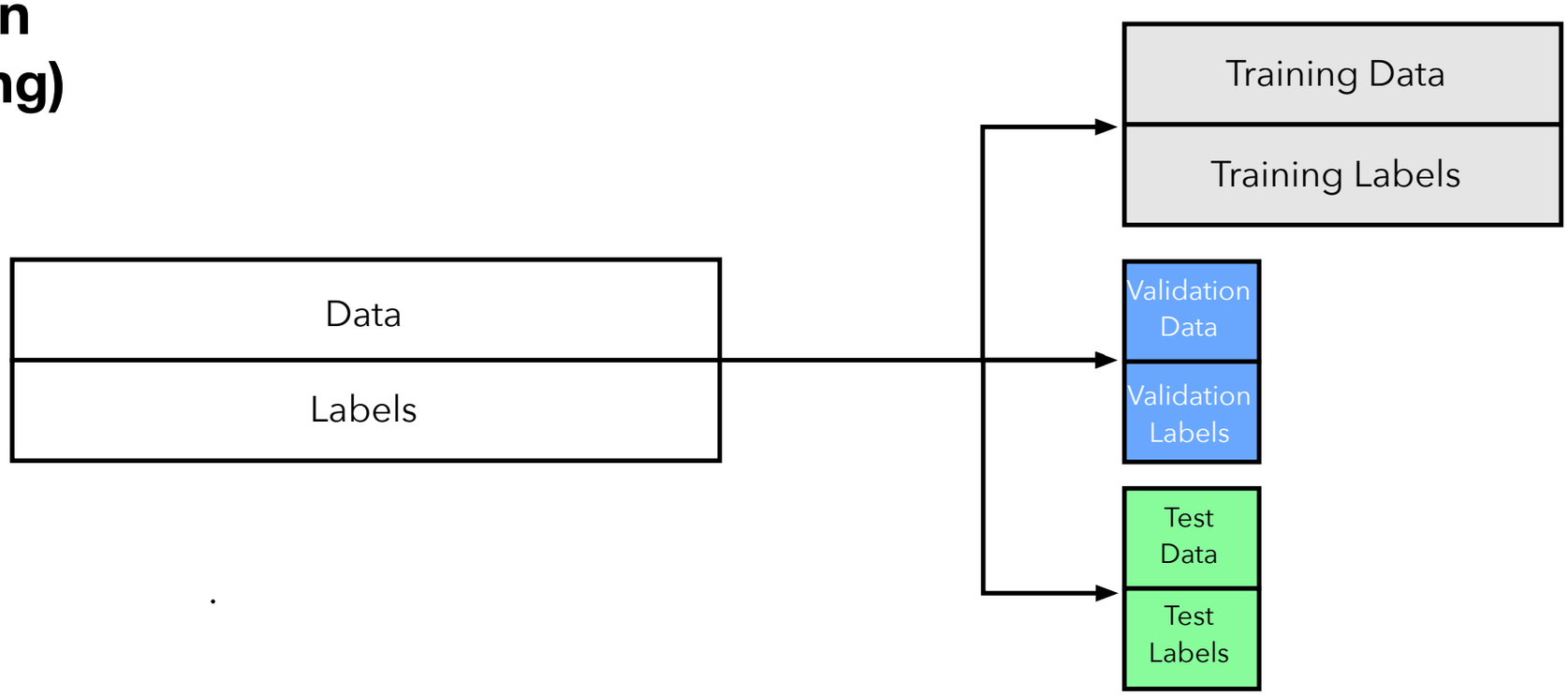
Can we use the holdout method for model selection?

Holdout validation (hyperparam. tuning)

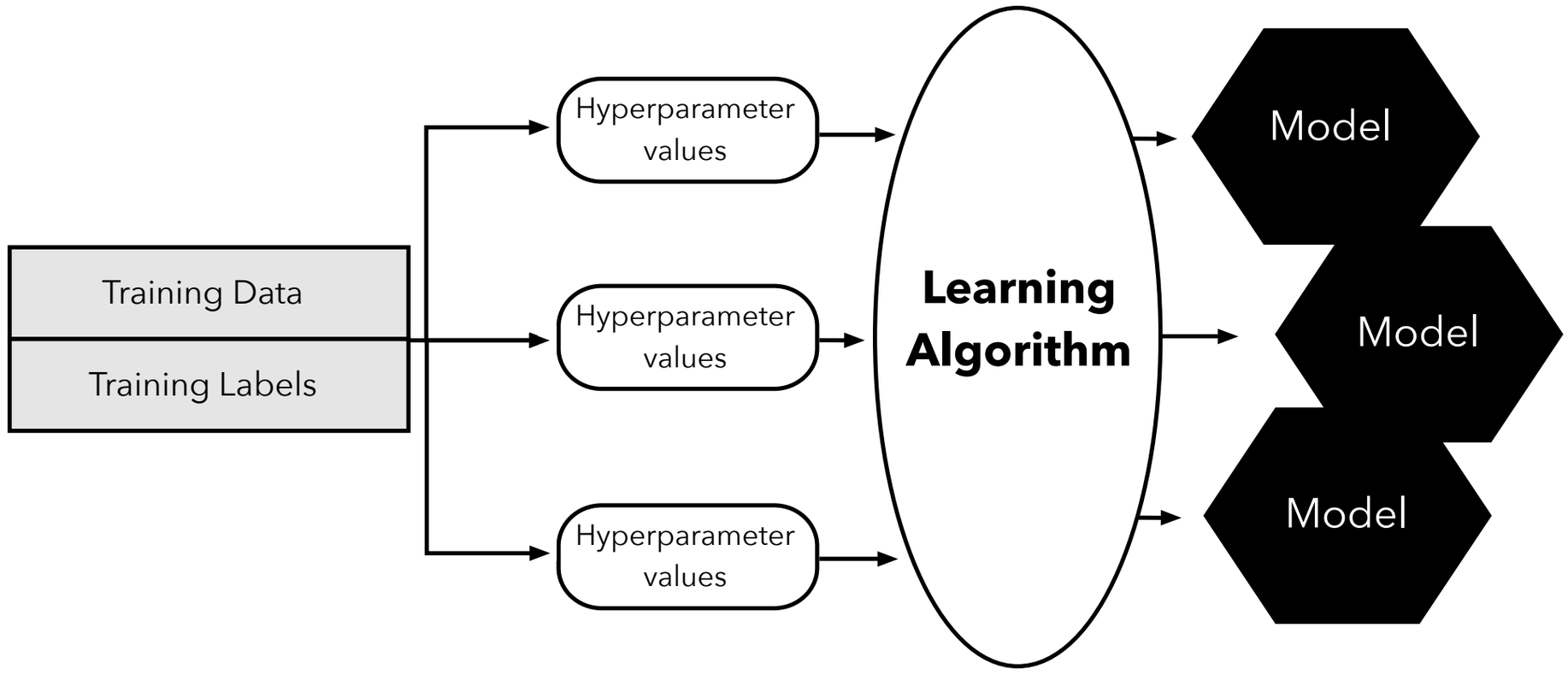


Holdout validation (hyperparam. tuning)

1

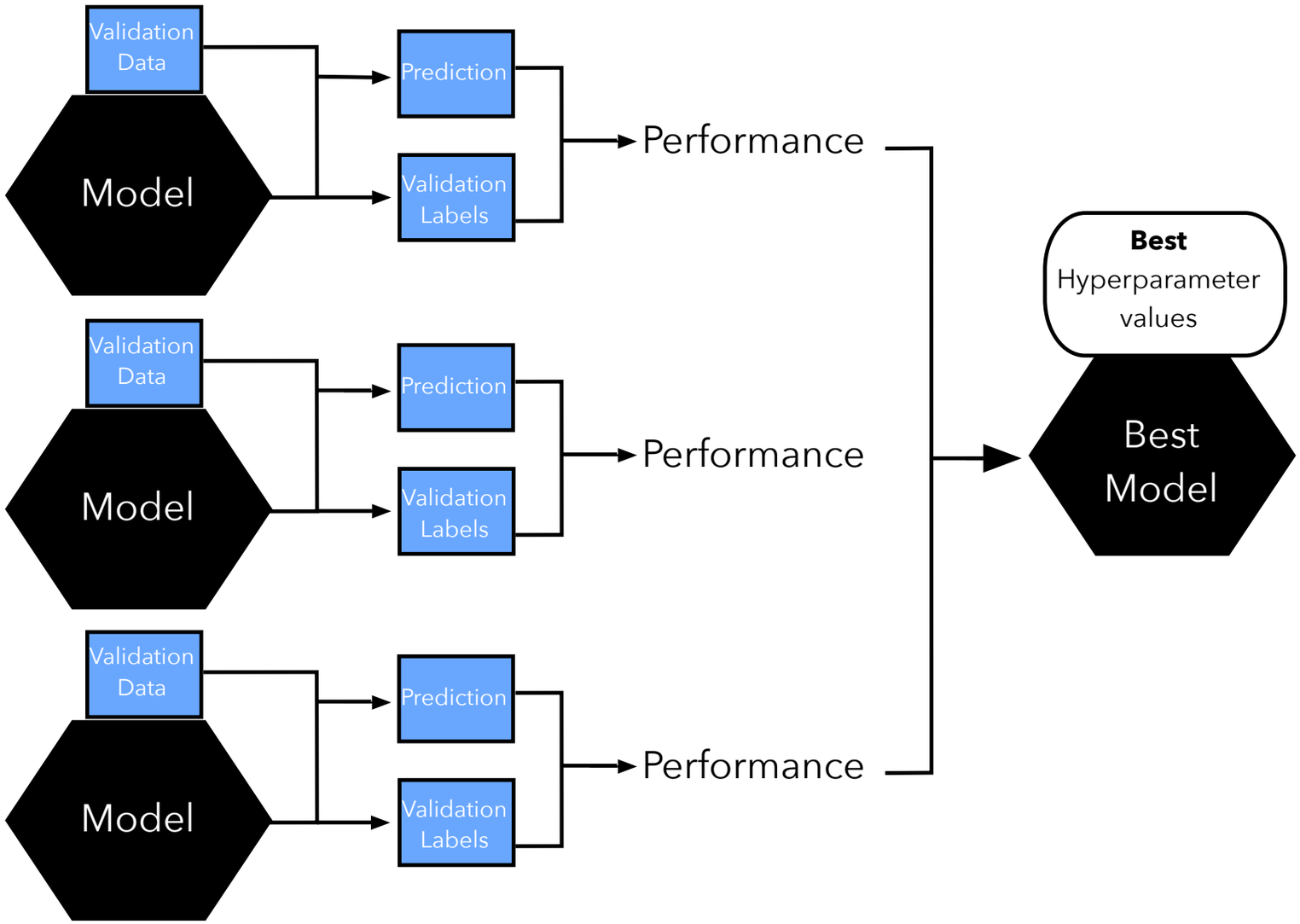


2

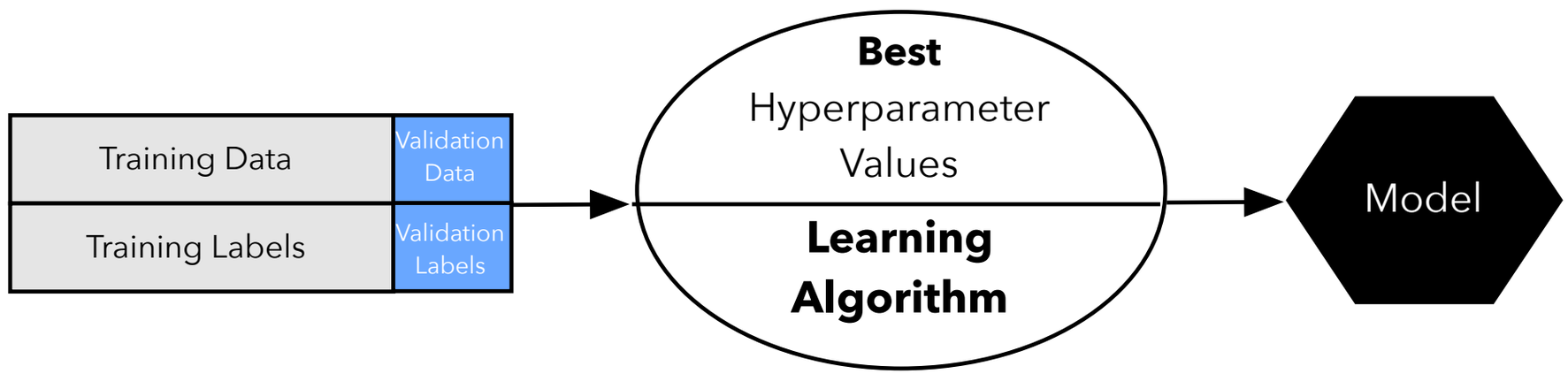


Holdout validation (hyperparam. tuning)

3

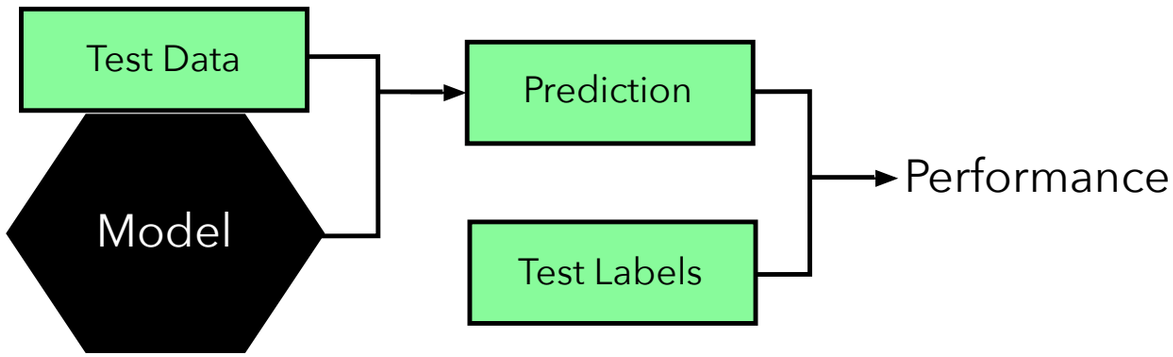


4

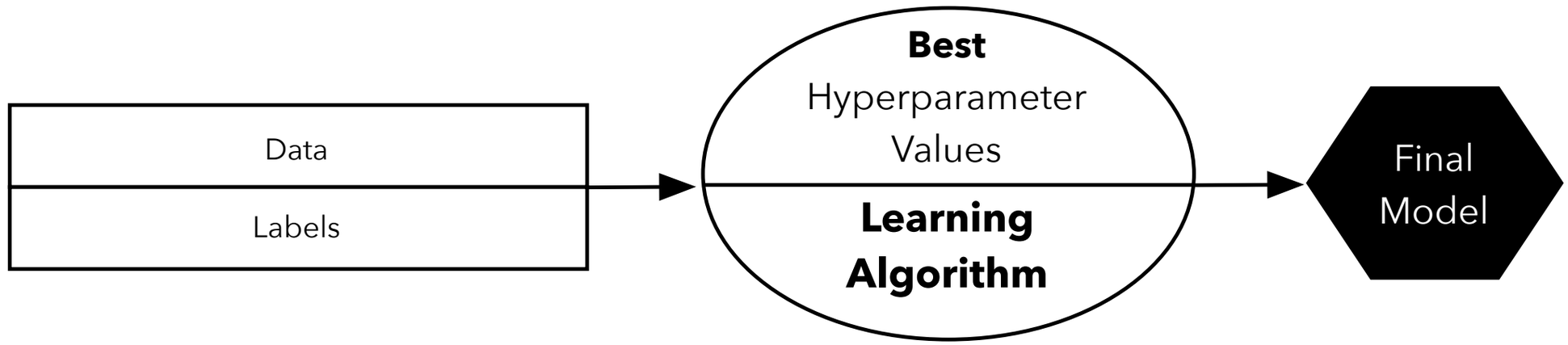


Holdout validation (hyperparam. tuning)

5



6



1. Introduction

2. Holdout method for model evaluation

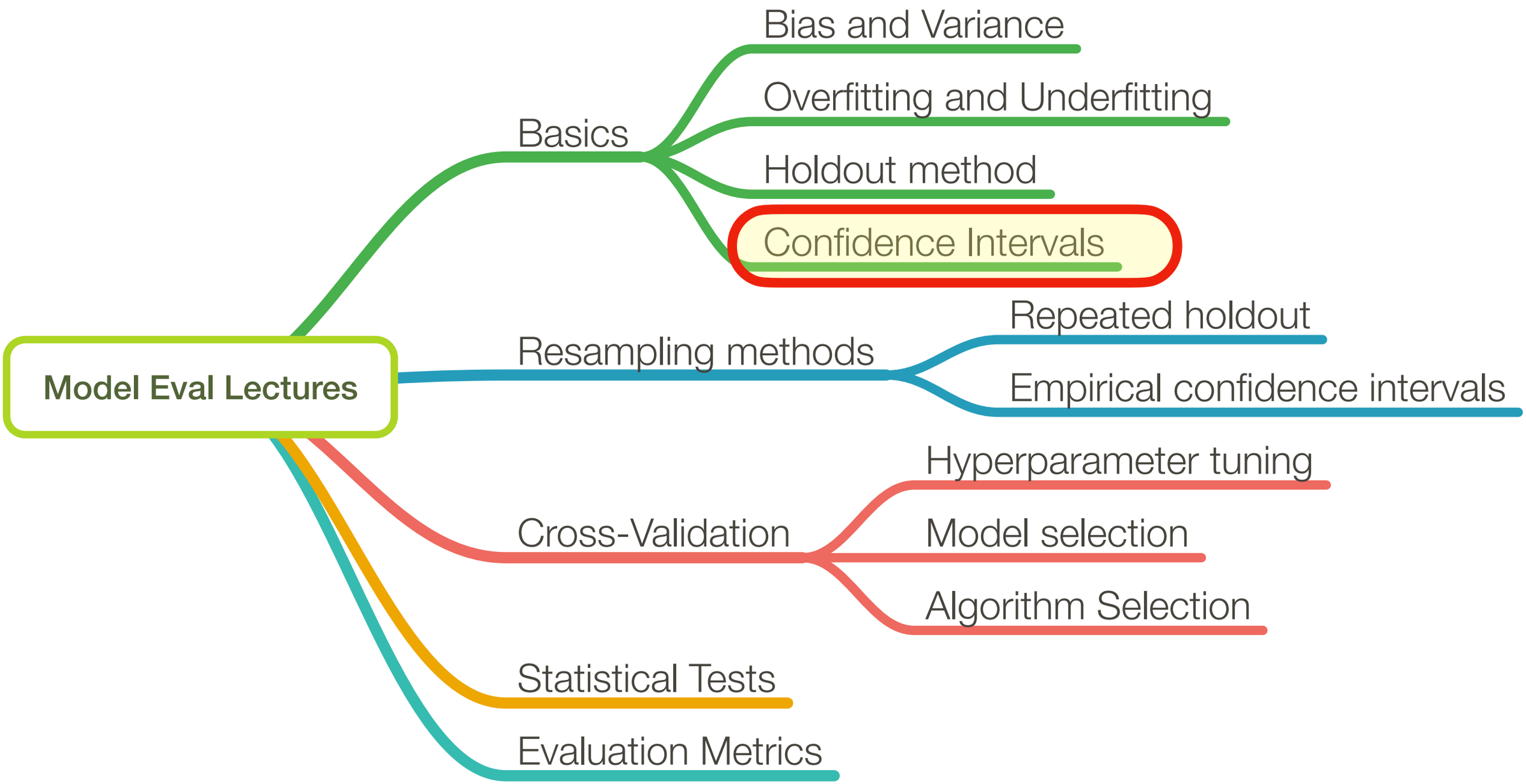
3. Holdout method for model selection

4. Confidence intervals -- normal approximation

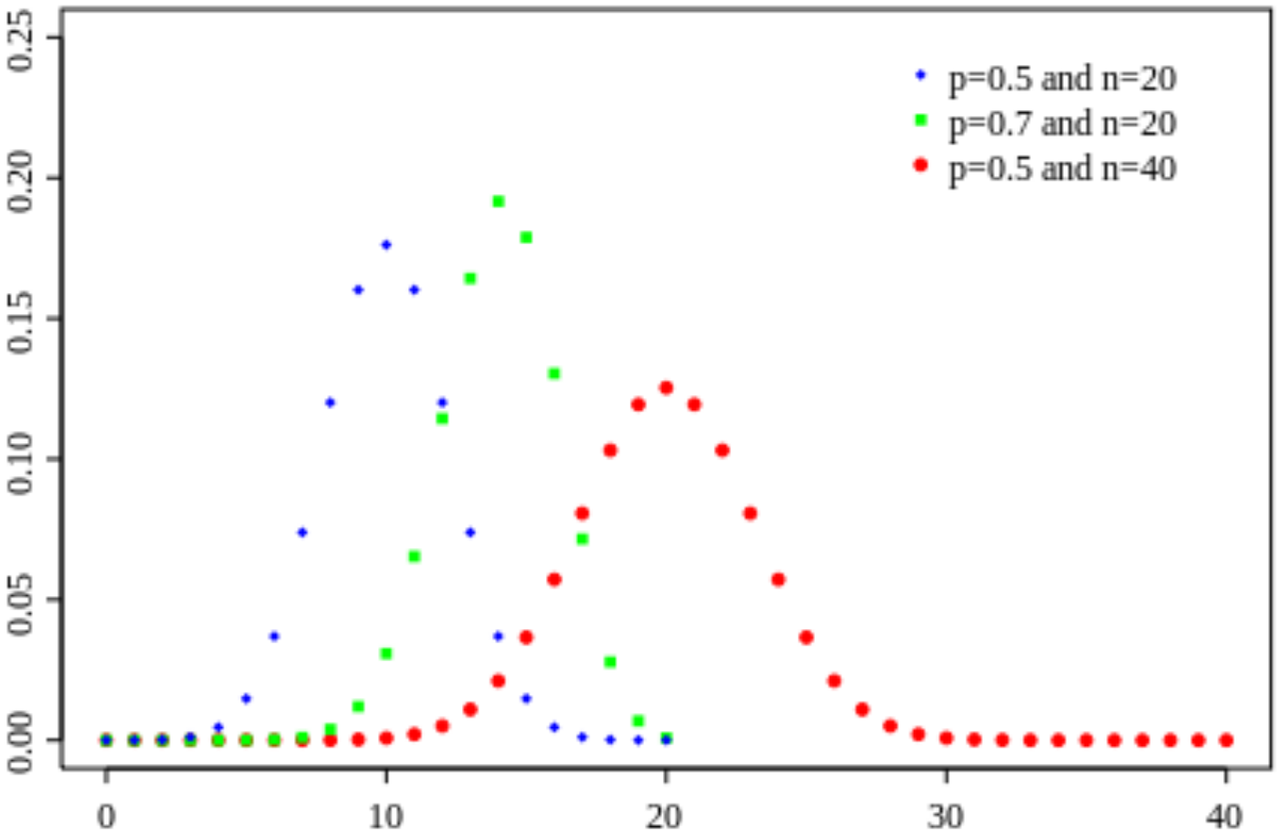
5. Resampling & repeated holdout

6. Empirical confidence intervals via Bootstrap

7. The 0.632 and 0.632+ Bootstrap


Model Eval Lectures
Basics
Bias and Variance
Overfitting and Underfitting
Holdout method
Confidence Intervals
Resampling methods
Repeated holdout
Empirical confidence intervals
Cross-Validation
Hyperparameter tuning
Model selection
Algorithm Selection
Statistical Tests
Evaluation Metrics

Binomial distribution



Notation	$B(n, p)$
Parameters	$n \in \mathbf{N}_0$ — number of trials $p \in [0,1]$ — success probability in each trial
Support	$k \in \{0, \ldots, n\}$ — number of successes
pmf	$\binom{n}{k} p^k (1-p)^{n-k}$
CDF	$I_{1-p}(n-k, 1+k)$
Mean	np
Median	$\lfloor np \rfloor$ or $\lceil np \rceil$
Mode	$\lfloor (n+1)p \rfloor$ or $\lceil (n+1)p \rceil - 1$
Variance	$np(1-p)$

(Image credit: Screenshot from https://en.wikipedia.org/wiki/Binomial_distribution)

Binomial distribution $Pr(k) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}.$

Coin Flip (Bernoulli Trial)

- coin lands on head ("success")
- probability of success p
- $\frac{k}{n}$, estimator of p
- mean, number of successes
  $\mu_k = np$

0-1 Loss

- example misclassified (0-1 loss)
- true error $ERR_{\mathcal{D}}(h) = \underset{x \in \mathcal{D}}{Pr} [f(x) \neq h(x)]$
- sample (test set) error

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

Binomial distribution $Pr(k) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}.$

Coin Flip (Bernoulli Trial)

- mean, number of successes
  $\mu_k = np$

- variance $\sigma_k^2 = np(1-p)$

- standard deviation $\sigma_k = \sqrt{np(1-p)}$

0-1 Loss

We are interested in proportions

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

$$\sigma^2 = p(1-p)$$

$$\sigma = \sqrt{p(1-p)}$$

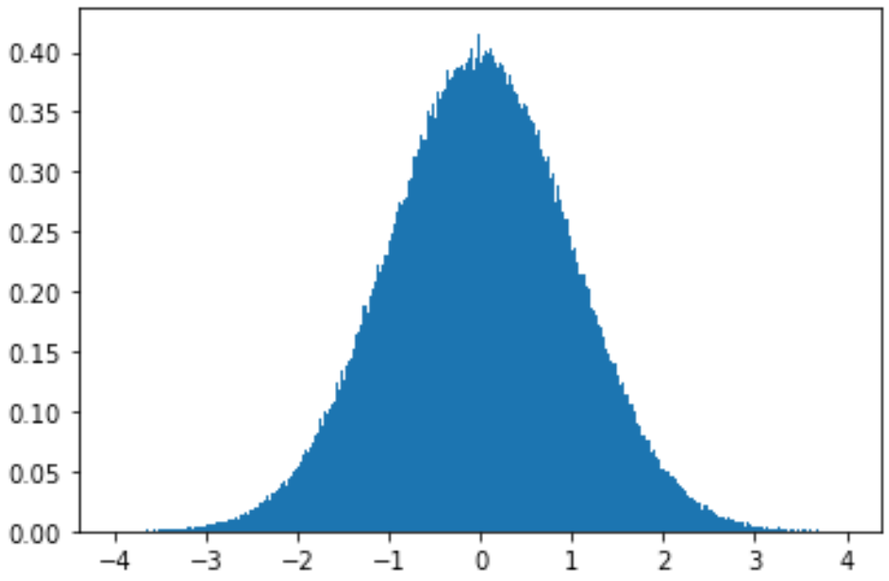
$$SE = \frac{\sqrt{ERR_S(1-ERR_S)}}{\sqrt{n}} = \sqrt{\frac{ERR_S(1-ERR_S)}{n}}$$

Confidence Intervals

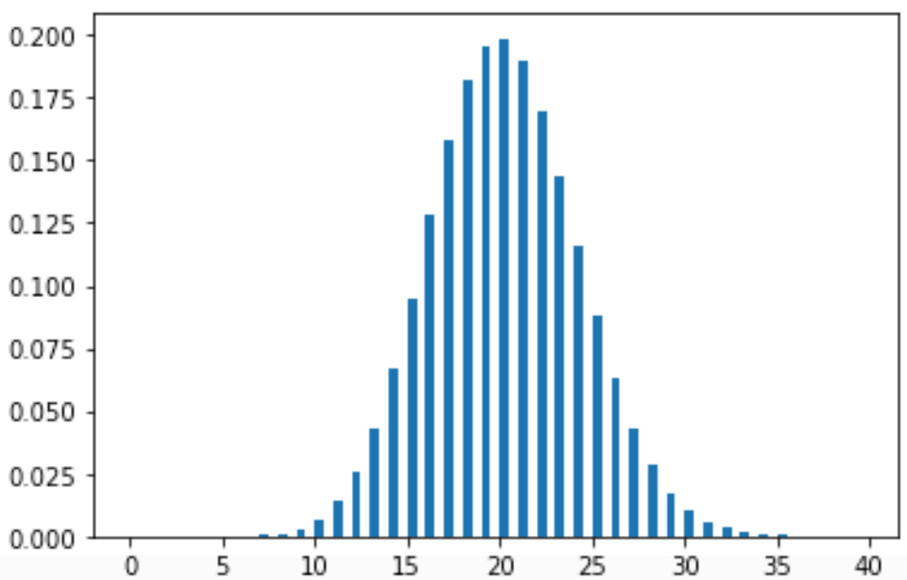
A XX% confidence interval of some parameter *p* is an interval that is expected to contain *p* with probability XX%.*

* the more precise definition is "In an infinite long series of trials in which repeated samples of n are taken from the same distribution, the 95% Confidence Interval is calculated using the same method, the proportion of intervals covering the true parameter p is xx%."

```
:
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(123)

std_norm_sample = np.random.randn(1000000)
plt.hist(std_norm_sample, bins=np.arange(-4, 4, 0.01), density=True)
plt.show()
```




```
:
binom_sample = np.random.binomial(n=100, p=0.2, size=1000000)
plt.hist(binom_sample, bins=np.arange(0, 40, 0.5), density=True)
plt.show()
```

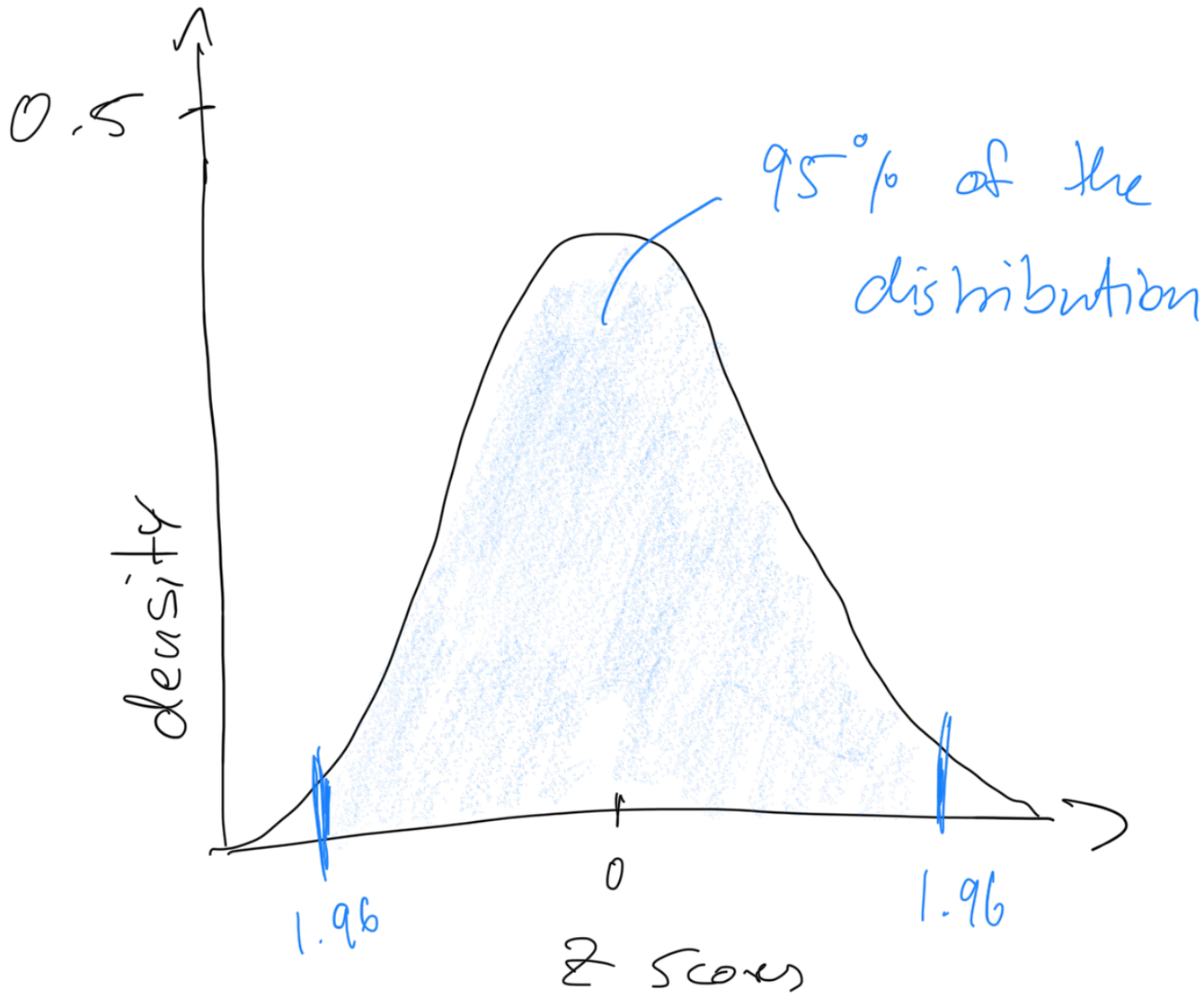


Normal Approximation Interval

- Less tedious than confidence interval for Binomial distribution and hence often used in (ML) practice for large *n*

- Rule of thumb: if *n* larger than 40, the Binomial distribution can be reasonably approximated by a Normal distribution; and *np* and *n*(1 - *p*) should be greater than 5

$$CI = ERR_S \pm z\sqrt{\frac{ERR_S(1 - ERR_S)}{n}}$$



The z constant for different confidence intervals:

- 99%: z=2.58
- 95%: z=1.96
- 90%: z=1.64

Code

```
clf.fit(X_train, y_train)
acc_test_na = clf.score(X_test, y_test)
ci_test_na = 1.96 * np.sqrt((acc_test_na*(1-acc_test_na)) / y_test.shape[0])

test_na_lower = acc_test_na-ci_test_na
test_na_upper = acc_test_na+ci_test_na

print(test_na_lower, test_na_upper)
```

```
0.8731774862637585 1.039865991997112
```

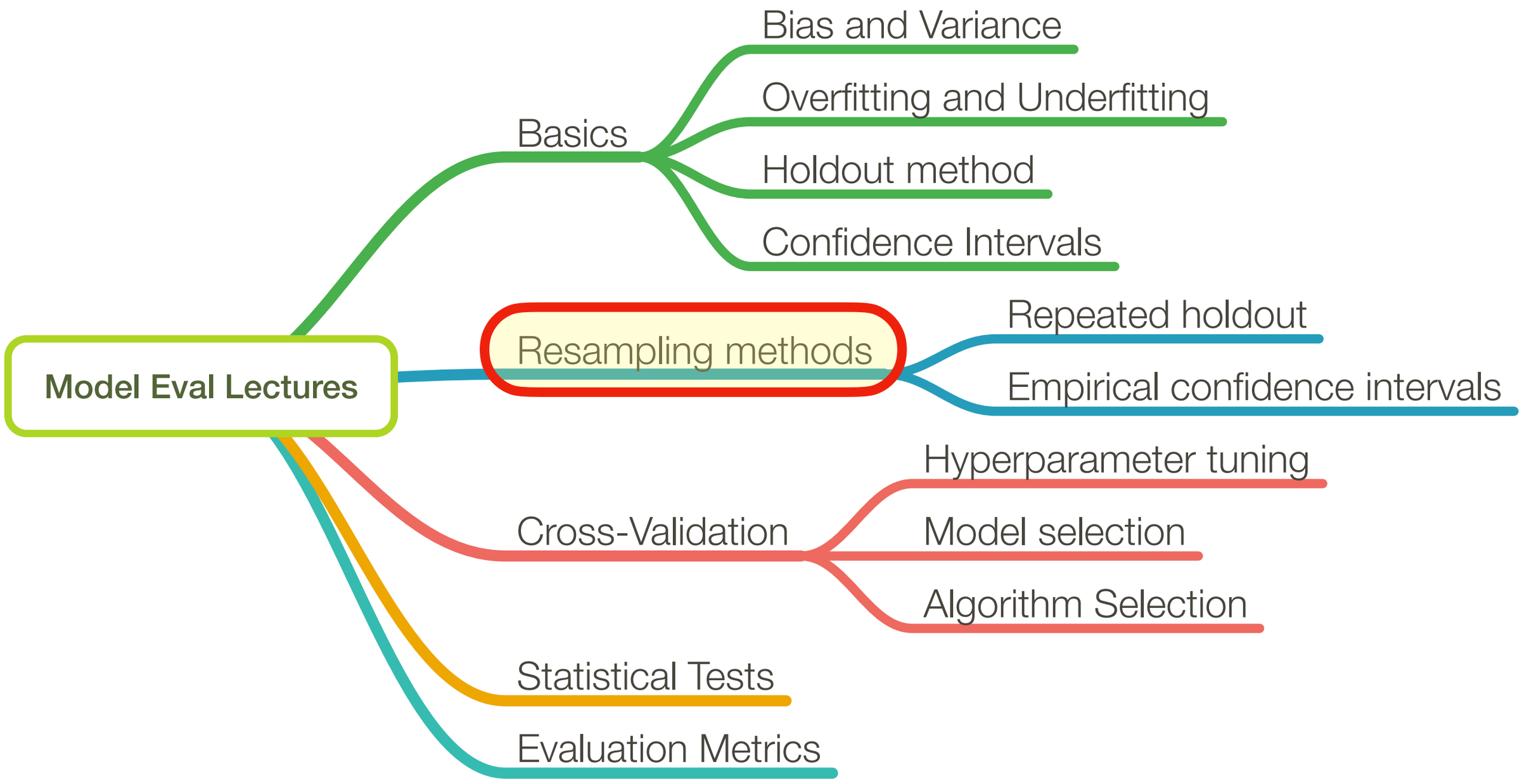
https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

Later in this lecture we will construct confidence intervals using different Bootstrap Techniques

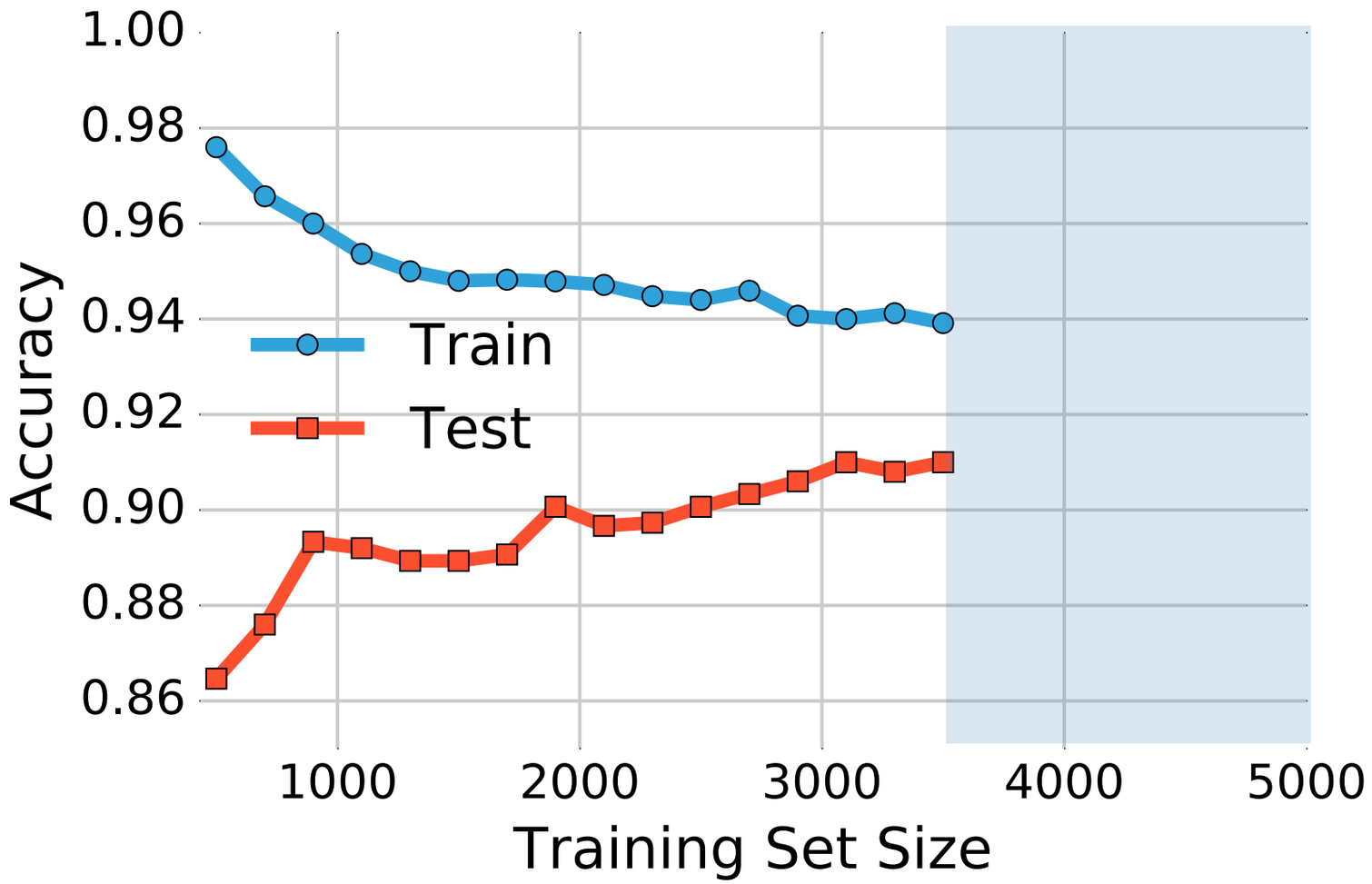
1. Introduction
2. Holdout method for model evaluation
3. Holdout method for model selection
4. Confidence intervals -- normal approximation

5. Resampling & repeated holdout

6. Empirical confidence intervals via Bootstrap
7. The 0.632 and 0.632+ Bootstrap


Model Eval Lectures
Basics
Bias and Variance
Overfitting and Underfitting
Holdout method
Confidence Intervals
Resampling methods
Repeated holdout
Empirical confidence intervals
Cross-Validation
Hyperparameter tuning
Model selection
Algorithm Selection
Statistical Tests
Evaluation Metrics

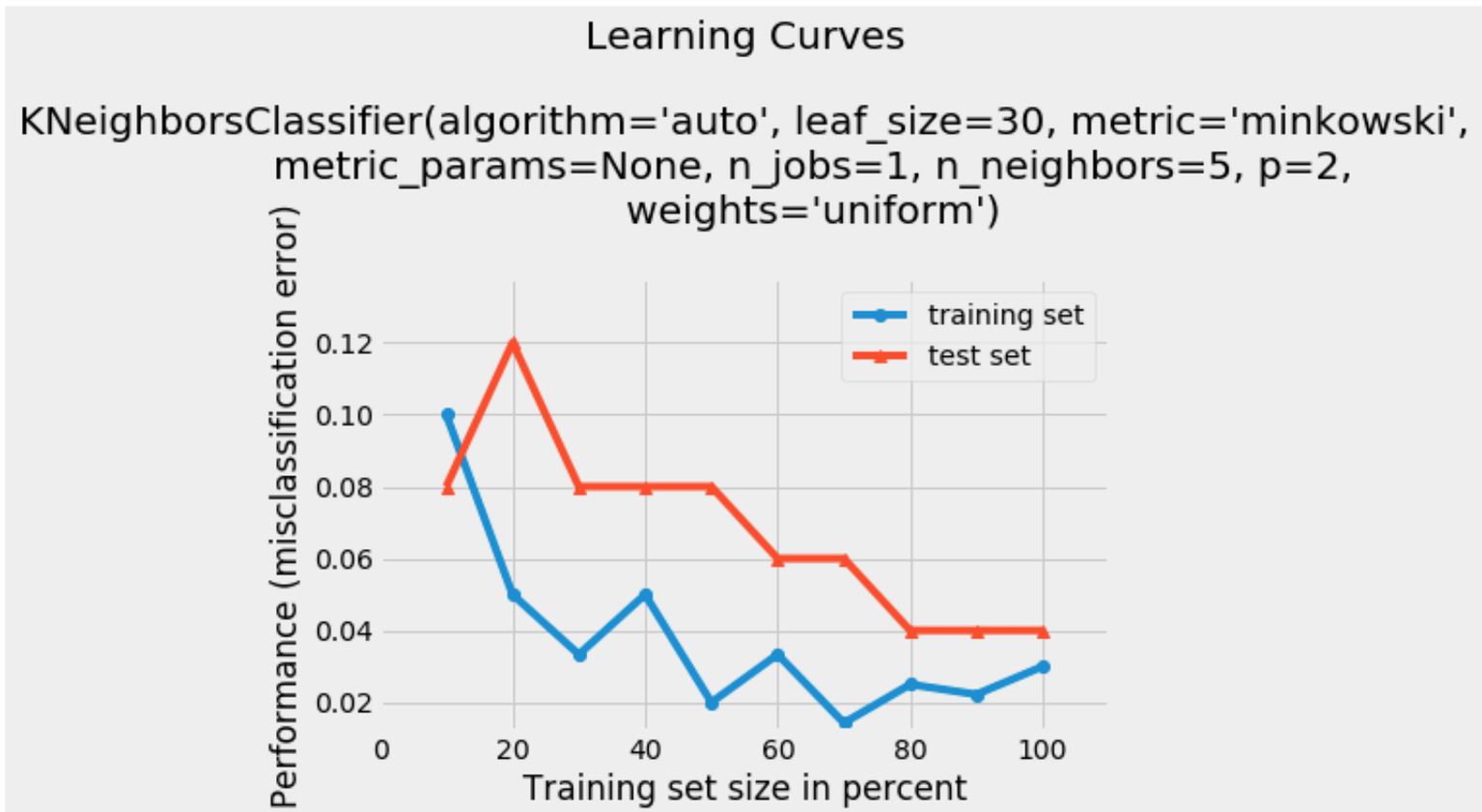
Proportionally large test sets increase the pessimistic bias if a model has not reached its full capacity, yet.



- To produce the plot above, I took 500 random samples of each of the ten classes from MNIST
- The sample was then randomly divided into a 3500-example training subset and a test set (1500 examples) via stratification.
- Even smaller subsets of the 3500-sample training set were produced via randomized, stratified splits, and I used these subsets to fit softmax classifiers and used the same 1500-sample test set to evaluate their performances; samples may overlap between these training subsets.

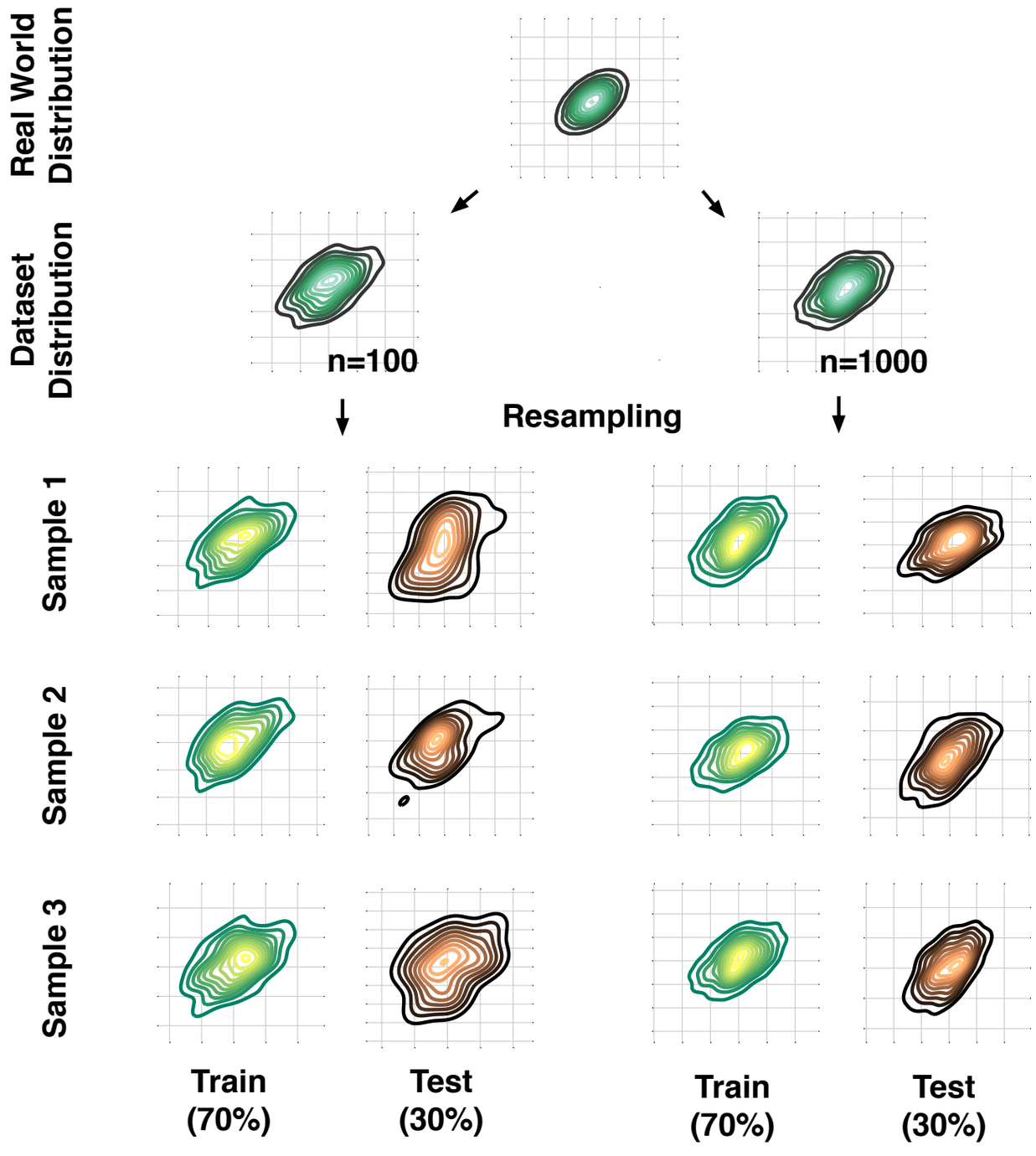
Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__3_pessimistic-bias-in-holdout.ipynb

```python
from mlxtend.plotting import plot_learning_curves
import matplotlib.pyplot as plt
from mlxtend.data import iris_data
from mlxtend.preprocessing import shuffle_arrays_unison
from sklearn.neighbors import KNeighborsClassifier
import numpy as np


# Loading some example data
X, y = iris_data()
X, y = shuffle_arrays_unison(arrays=[X, y], random_seed=123)
X_train, X_test = X[:100], X[100:]
y_train, y_test = y[:100], y[100:]

clf = KNeighborsClassifier(n_neighbors=5)

plot_learning_curves(X_train, y_train, X_test, y_test, clf)
plt.show()
```



http://rasbt.github.io/mlxtend/user_guide/plotting/plot_learning_curves/

Decreasing the size of the test set brings up another problem:
It may result in a substantial variance increase of our model's performance estimate.



Here, I repeatedly subsampled a two-dimensional Gaussian

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 our data, we alter the statistics of the distribution of the sample.

Repeated Holdout: Estimate Model Stability

(also called Monte Carlo Cross-Validation)

Average performance over *k* repetitions

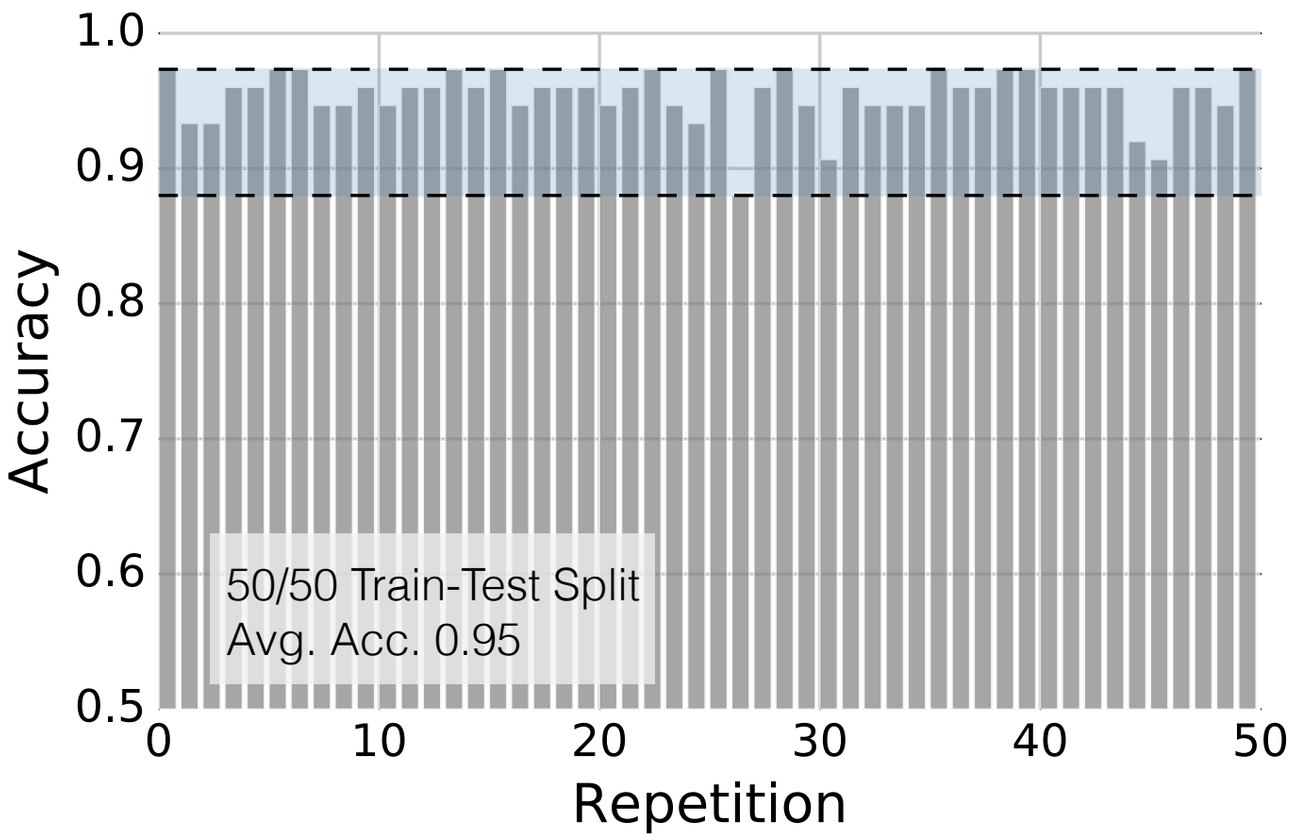
$$\text{ACC}_{avg} = \frac{1}{k} \sum_{j=1}^{k} ACC_j,$$

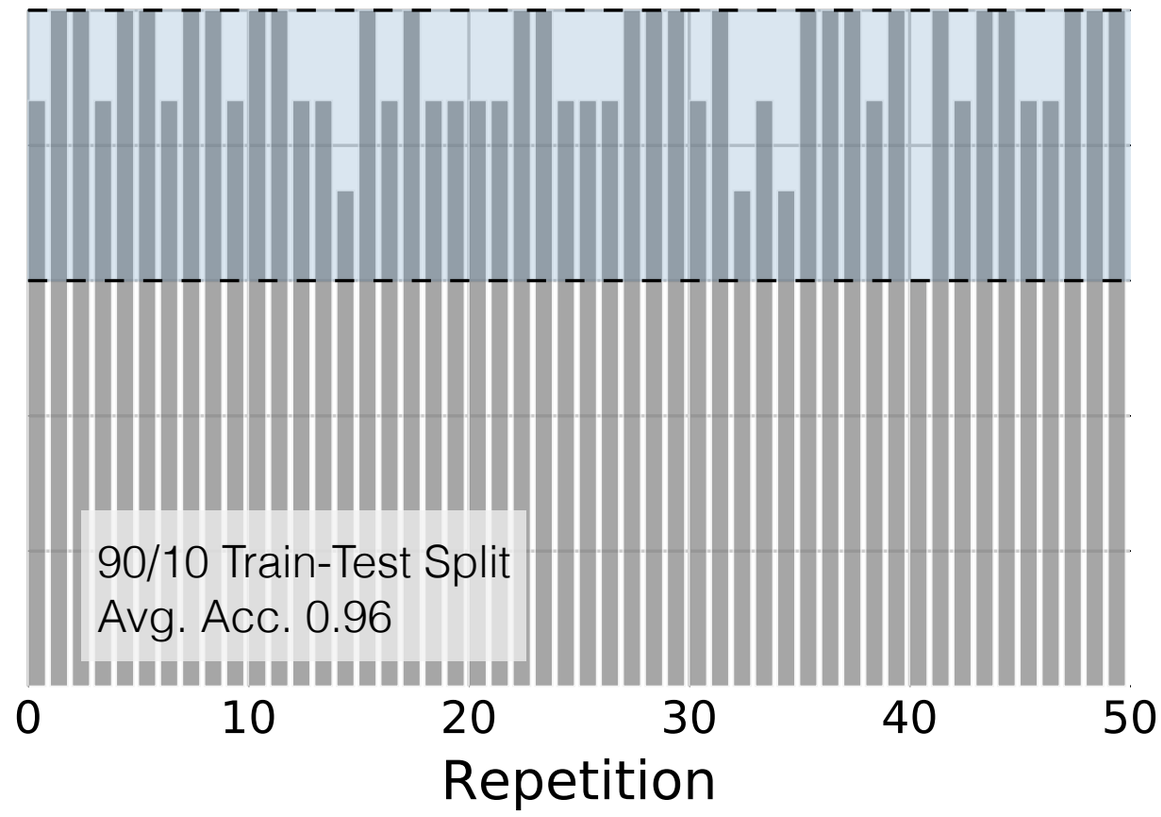
where ACC_j is the accuracy estimate of the *j*th test set of size *m*,

$$\text{ACC}_j = 1 - \frac{1}{n} \sum_{i=1}^{n} L\left(h(x^{[i]}), f(x^{[i]})\right) .$$

Repeated Holdout: Estimate Model Stability

How repeated holdout validation may look like for different training-test split using the Iris dataset to fit to 3-nearest neighbors classifiers:





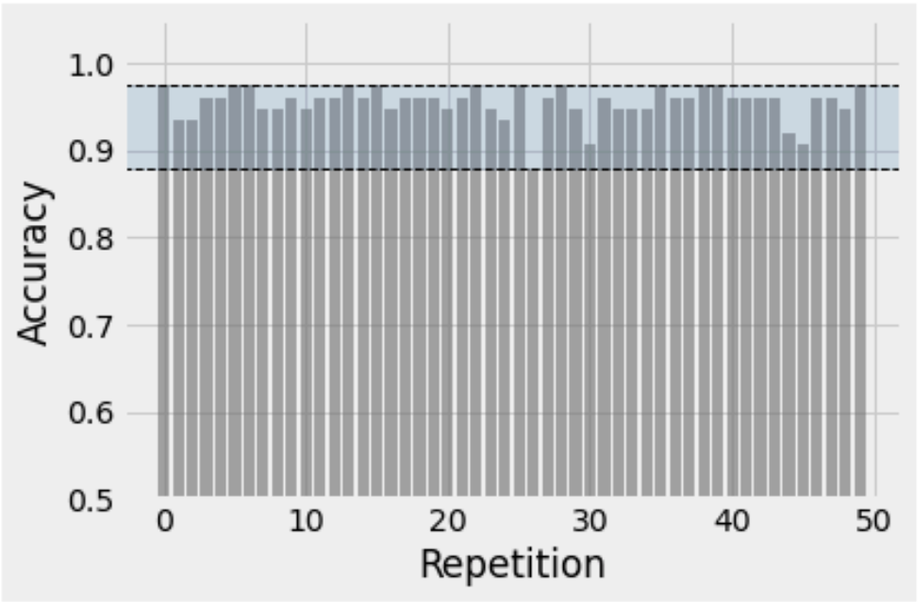
Left: I performed 50 stratified training/test splits with 75 samples in the test and training set each; a K-nearest neighbors model was fit to the training set and evaluated on the test set in each repetition.

Right: Here, I repeatedly performed 90/10 splits, though, so that the test set consisted of only 15 samples.

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__2_holdout-and-repeated-sampling.ipynb

```python
rng = np.random.RandomState(seed=12345)
seeds = np.arange(10**5)
rng.shuffle(seeds)
seeds = seeds[:50]

pred_2 = []

for i in seeds:
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size=0.5,
                                                        random_state=i,
                                                        stratify=y)
    y_pred_i = clf_1.fit(X_train, y_train).predict(X_test)
    y_pred_i_acc = np.mean(y_test == y_pred_i)
    pred_2.append(y_pred_i_acc)

pred_2 = np.asarray(pred_2)
print('Average: %.2f%%' % (pred_2.mean()*100))

with plt.style.context(('fivethirtyeight')):
    plt.bar(range(0, pred_2.shape[0]), pred_2, color='gray', alpha=0.7)
    plt.axhline(pred_2.max(), color='k', linewidth=1, linestyle='--')
    plt.axhline(pred_2.min(), color='k', linewidth=1, linestyle='--')
    plt.axhspan(pred_2.min(), pred_2.max(), alpha=0.2, color='steelblue')
    plt.ylim([0, pred_2.max() + 0.1])
    plt.xlabel('Repetition')
    plt.ylabel('Accuracy')
    plt.ylim([0.5, 1.05])
    plt.tight_layout()
    #plt.savefig('figures/model-eval-iris_0.svg')
    plt.show()
```

```
Average: 95.41%
```



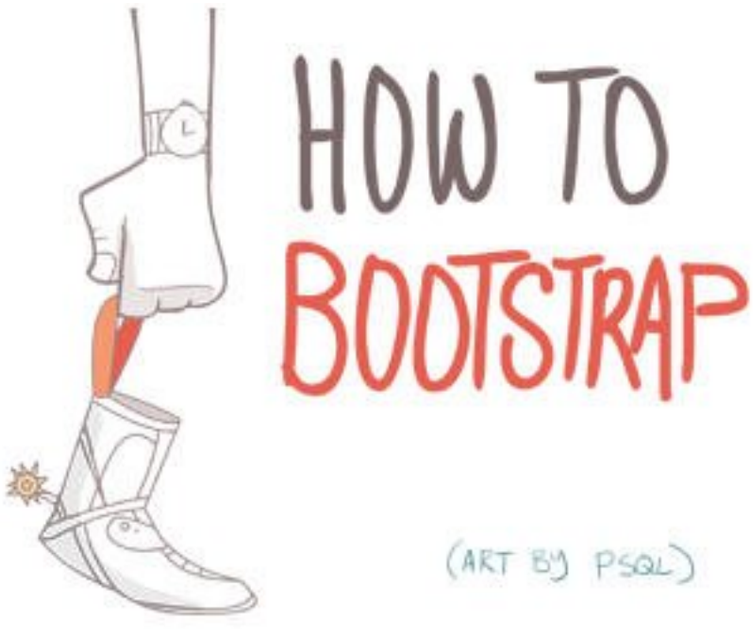
Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__2_holdout-and-repeated-sampling.ipynb

1. Introduction
2. Holdout method for model evaluation
3. Holdout method for model selection
4. Confidence intervals -- normal approximation
5. Resampling & repeated holdout
6. **Empirical confidence intervals via Bootstrap**
7. The 0.632 and 0.632+ Bootstrap

The Bootstrap Method and Empirical Confidence Intervals

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, "30. 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)



Source: https://memim.com/bootstrapping.html

The Bootstrap Method and Empirical Confidence Intervals

- The bootstrap method is a resampling technique for estimating a sampling distribution
- Here, we are particularly interested in estimating the uncertainty of our performance estimate
- The bootstrap method was introduced by Bradley Efron in 1979 [1]
- About 15 years later, Bradley Efron and Robert Tibshirani even devoted a whole book to the bootstrap, “An Introduction to the Bootstrap” [2]
- 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.

[1] Efron, Bradley. 1979. “Bootstrap Methods: Another Look at the Jackknife.” *The Annals of Statistics* 7 (1). Institute of Mathematical Statistics: 1–26. doi:10.1214/aos/1176344552.

[2] Efron, Bradley, and Robert Tibshirani. 1994. *An Introduction to the Bootstrap*. Chapman & Hall.

The Bootstrap Method and Empirical Confidence Intervals

Using the bootstrap, we can estimate sample statistics and compute the standard error of the mean and confidence intervals as if we have drawn a number of samples from an infinite population. In a nutshell, the bootstrap procedure can be described as follows:

1. Draw a sample with replacement
2. Compute the sample statistic
3. Repeat step 1-2 n times
4. Compute the standard deviation (standard error of the mean of the statistic)
5. Compute the confidence interval

The Bootstrap Method and Empirical Confidence Intervals For Evaluating Classifier Performance

1. We are given a dataset of size *n*.

2. For *b* bootstrap rounds:

    1. We draw one single instance from this dataset and assign it to our *j*th 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 so that certain samples may appear more than once in our 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

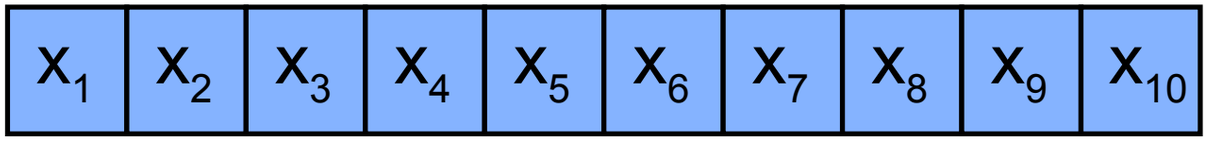
$$\text{ACC}_{boot} = \frac{1}{b}\sum_{j=1}^{b}\frac{1}{n}\sum_{i=1}^{n}\left(1 - L\left(h(x^{[i]}), f(x^{[i]})\right)\right.\text{.}$$

The Bootstrap Method and Empirical Confidence Intervals

- As we discussed previously, the resubstitution accuracy usually leads to an extremely optimistic bias, since a model can be overly sensible to noise in a dataset.
- Originally, the bootstrap method aims 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

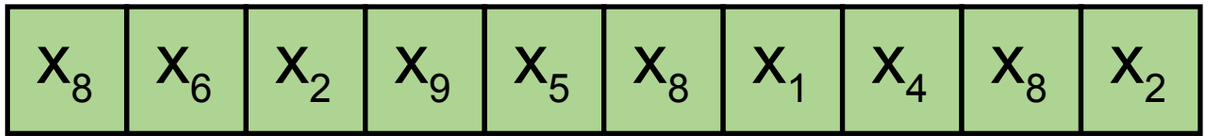
Bootstrap Sampling

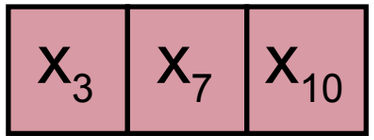
Original Dataset



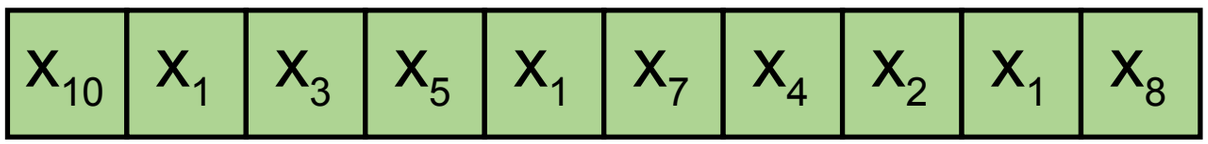
Out-of-bag samples

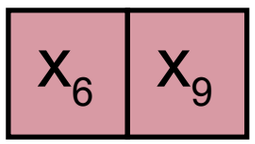
Bootstrap 1



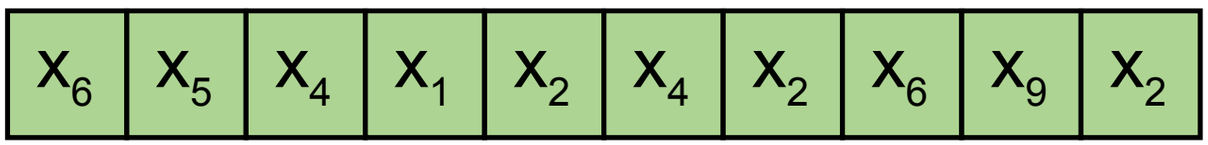


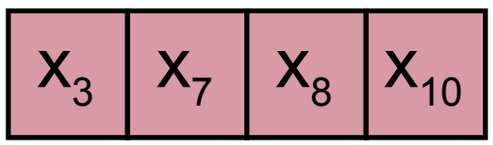
Bootstrap 2





Bootstrap 3





Training Sets

The Bootstrap Method and Empirical Confidence Intervals

We can compute the 95% confidence interval of the bootstrap estimate as

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

and use it to compute the standard error

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

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

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

For instance, given a sample with n=100, we find that $t_{\alpha=0.05,99} = 1.984$ (95% CI)

(In practice, at least 200 bootstrap rounds are recommended)

4.1 Iris Simulation

:
```
from mlxtend.data import iris_data
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split



X, y = iris_data()

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.15,
                                                    random_state=12345,
                                                    stratify=y)

clf = KNeighborsClassifier(n_neighbors=3,
                           weights='uniform',
                           algorithm='kd_tree',
                           leaf_size=30,
                           p=2,
                           metric='minkowski',
                           metric_params=None,
                           n_jobs=1)
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

4.1.1 Out-of-Bag (OOB) Bootstrap; Bootstrapping Training Sets -- Setup Step

- If you don't tune your model on the training set, you actually don't need a test set for this approach

:
```
import numpy as np


rng = np.random.RandomState(seed=12345)
idx = np.arange(y_train.shape[0])

bootstrap_train_accuracies = []

for i in range(200):

    train_idx = rng.choice(idx, size=idx.shape[0], replace=True)
    test_idx = np.setdiff1d(idx, train_idx, assume_unique=False)

    boot_train_X, boot_train_y = X_train[train_idx], y_train[train_idx]
    boot_test_X, boot_test_y = X_train[test_idx], y_train[test_idx]

    clf.fit(boot_train_X, boot_train_y)
    acc = clf.score(boot_test_X, boot_test_y)
    bootstrap_train_accuracies.append(acc)

bootstrap_train_mean = np.mean(bootstrap_train_accuracies)
bootstrap_train_mean
```

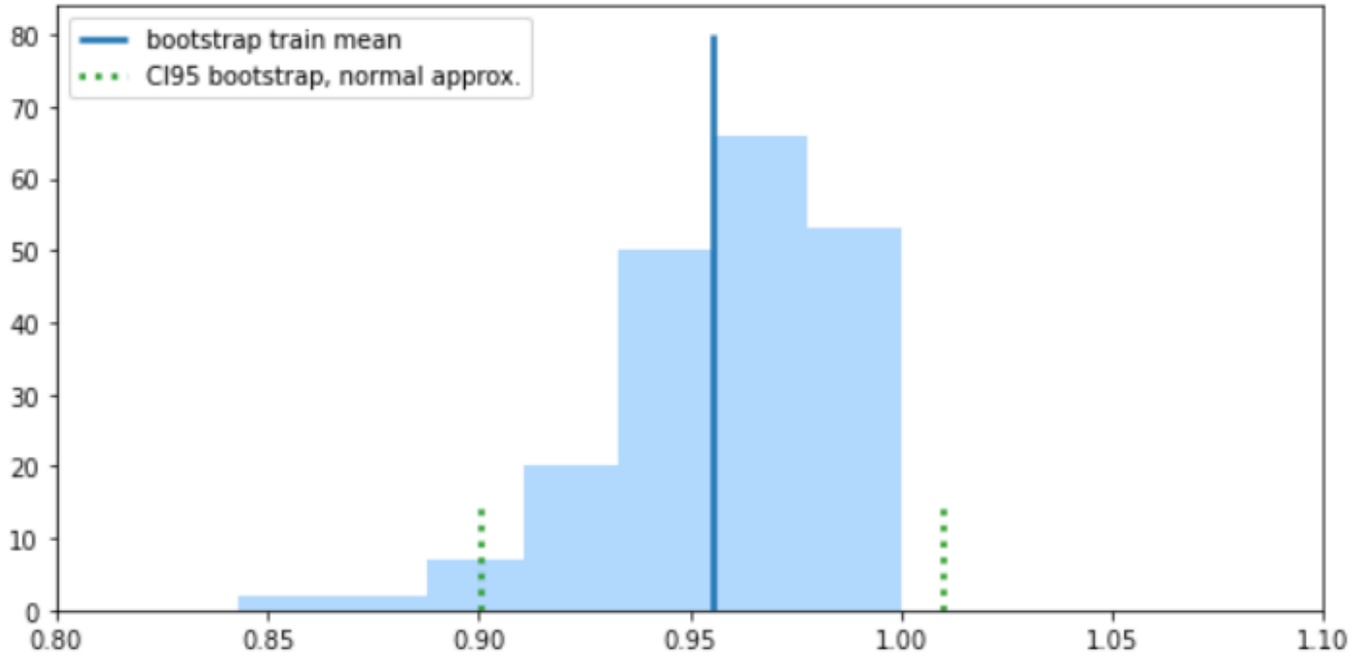
:
```
0.9552421700709434
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

4.1.2 Bootstrap 1-sample Confidence Interval (Based on OOB Estimates)

```
se = np.sqrt( (1. / (200-1)) * np.sum([(acc - bootstrap_train_mean)**2
                                       for acc in bootstrap_train_accuracies])
)
ci = 1.97 * se  # 1.97 based on T distribution

bootstrap_na_lower = bootstrap_train_mean-ci
bootstrap_na_upper = bootstrap_train_mean+ci

print(bootstrap_na_lower, bootstrap_na_upper)
```

```
0.900528306142954 1.009956033989329
```



Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

The Bootstrap Method and Empirical Confidence Intervals

And 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 (Efron, 1981). Here, we pick our lower and upper confidence bounds as follows:

$$\text{ACC}_{lower} = \alpha_1 th \text{ percentile of the } \text{ACC}_{boot} \text{ distribution}$$

$$\text{ACC}_{upper} = \alpha_1 th \text{ percentile of the } \text{ACC}_{boot} \text{ distribution}$$

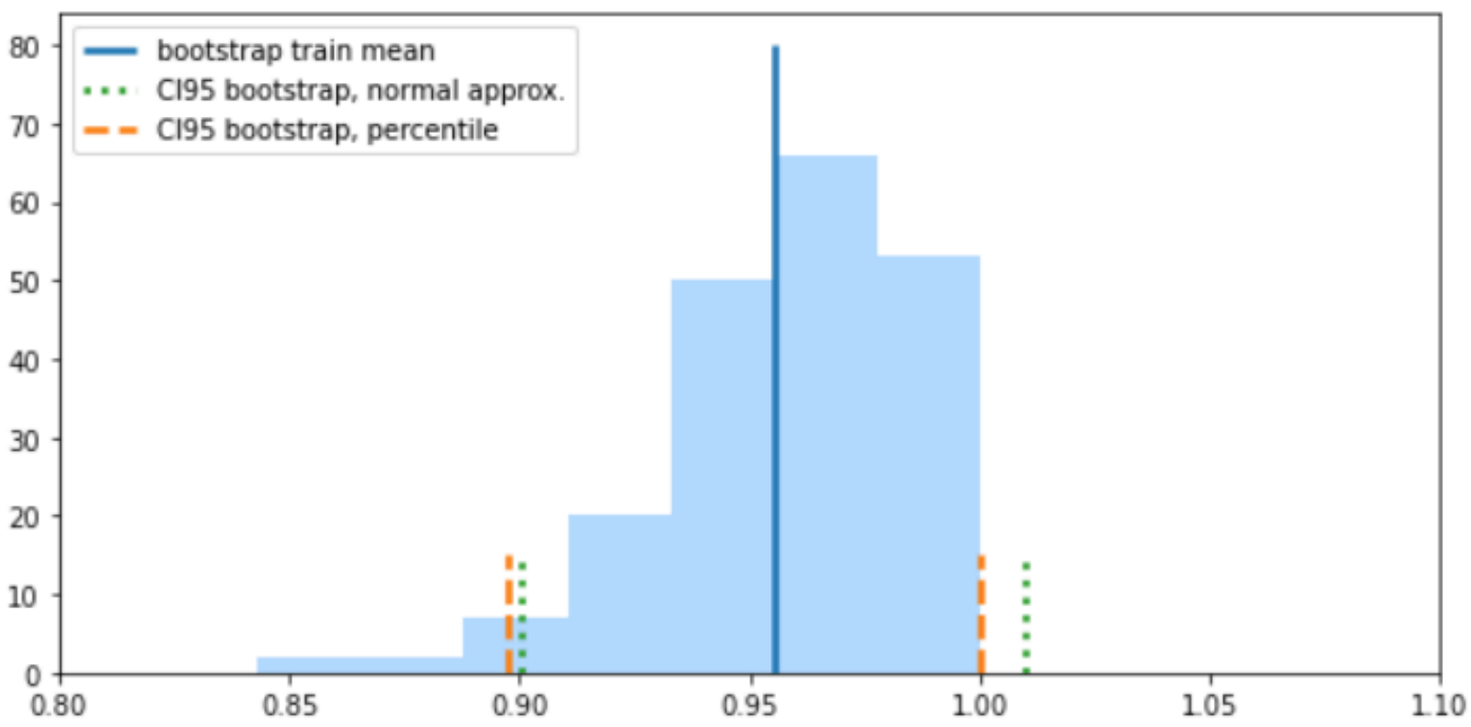
where $\alpha_1 = \alpha$ and $\alpha_2 = 1 - \alpha$ and α is our degree of confidence to compute the $100 \times (1 - 2 \times \alpha)$ confidence interval.

For instance, to compute a 95% confidence interval, we pick $\alpha = 0.025$ to obtain the 2.5th and 97.5th percentiles of the *b* bootstrap samples distribution as our upper and lower confidence bounds.

4.1.3 Bootstrap Percentile Method

```
bootstrap_percentile_lower = np.percentile(bootstrap_train_accuracies, 2.5)
bootstrap_percentile_upper = np.percentile(bootstrap_train_accuracies, 97.5)

print(bootstrap_percentile_lower, bootstrap_percentile_upper)
```

```
0.8977324263038549 1.0
```



Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

Bootstrapping the Test set

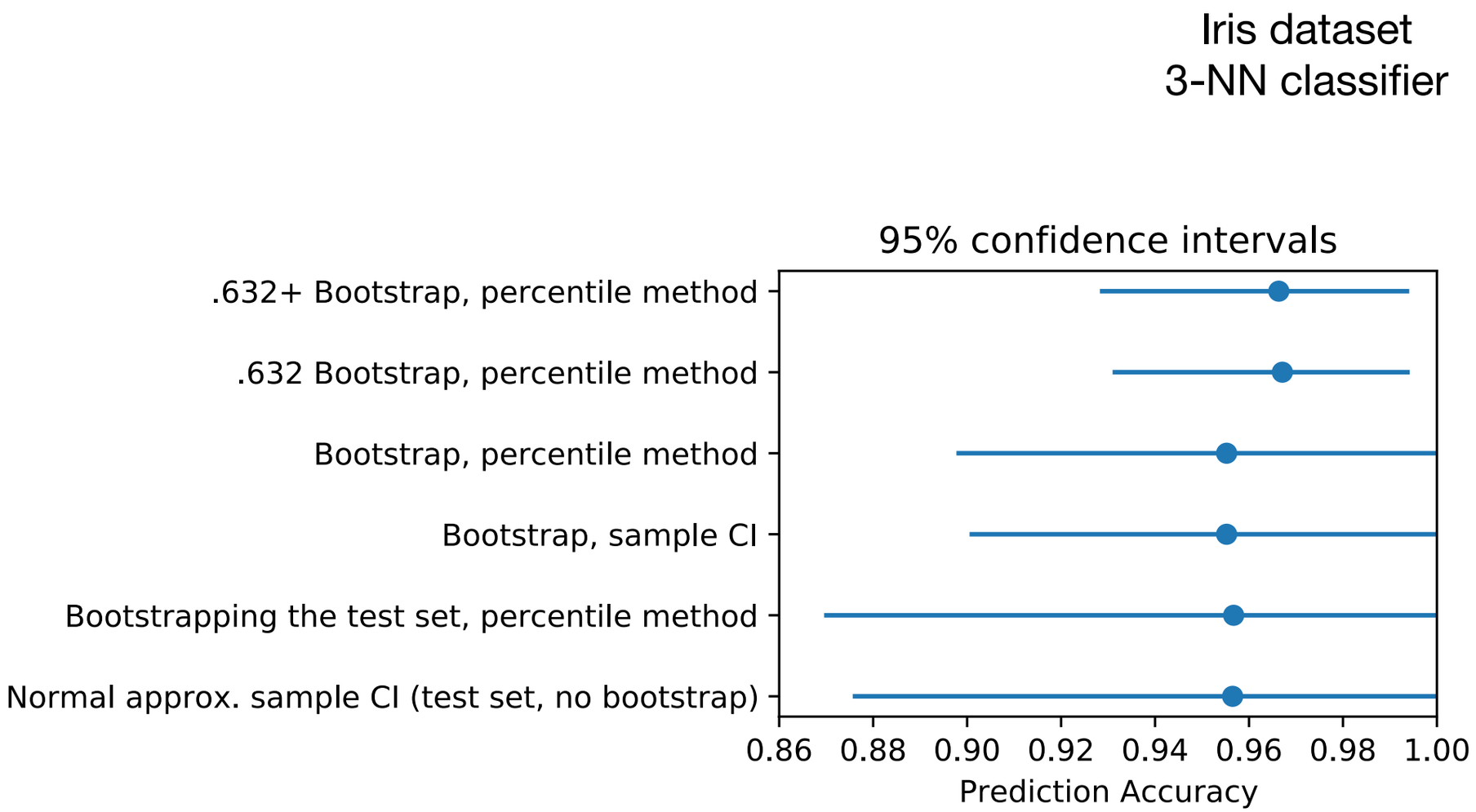
4.1.5 Bootstrapping the Test Set predictions

- Avoids retraining the model

:
```
clf.fit(X_train, y_train)

predictions_test = clf.predict(X_test)
acc_test = np.mean(predictions_test == y_test)

rng = np.random.RandomState(seed=12345)
idx = np.arange(y_test.shape[0])

test_accuracies = []

for i in range(200):

    pred_idx = rng.choice(idx, size=idx.shape[0], replace=True)
    acc_test_boot = np.mean(predictions_test[pred_idx] == y_test[pred_idx])
    test_accuracies.append(acc_test_boot)


mean_test_accuracies = np.mean(test_accuracies)
bootstrap_lower_test = np.percentile(test_accuracies, 2.5)
bootstrap_upper_test = np.percentile(test_accuracies, 97.5)

print(bootstrap_lower_test, bootstrap_upper_test)
```

```
0.8695652173913043 1.0
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

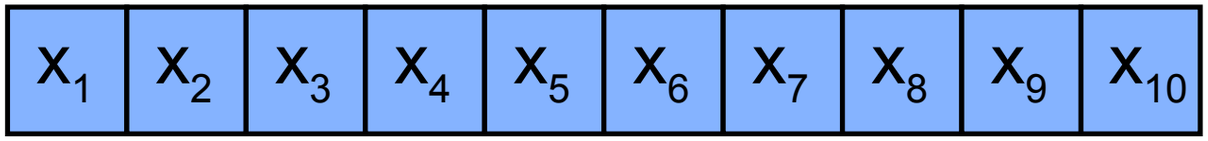


Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

1. Introduction
2. Holdout method for model evaluation
3. Holdout method for model selection
4. Confidence intervals -- normal approximation
5. Resampling & repeated holdout
6. Empirical confidence intervals via Bootstrap
7. **The 0.632 and 0.632+ Bootstrap**

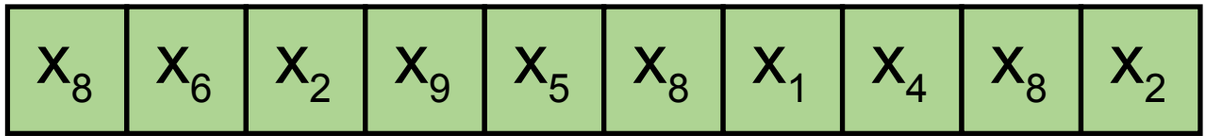
Bootstrap Sampling

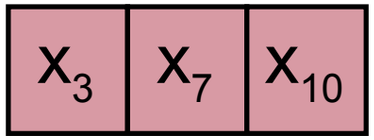
Original Dataset



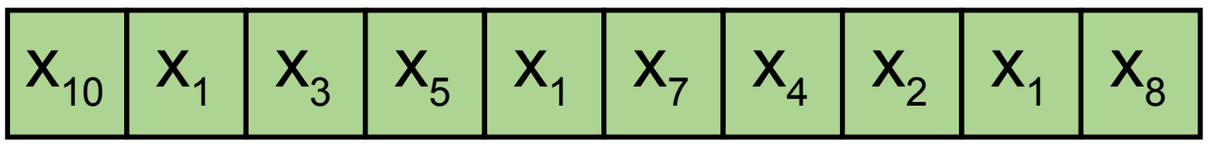
Out-of-bag samples

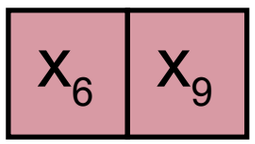
Bootstrap 1



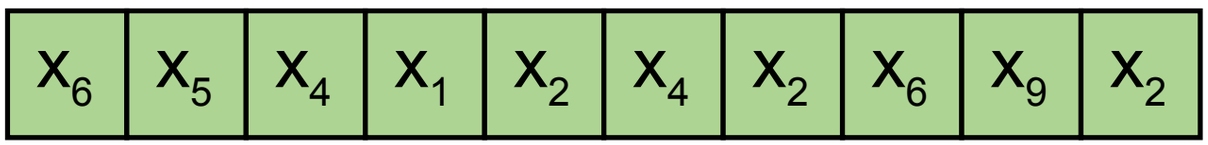


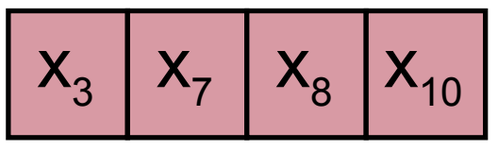
Bootstrap 2





Bootstrap 3





Training Sets

OOB Bootstrap
Object Oriented API

In this section, we are going to look at the OOB bootstrap method, which I recently implemented in mlxtend.

```
from mlxtend.evaluate import BootstrapOutOfBag
import numpy as np



oob = BootstrapOutOfBag(n_splits=3, random_seed=1)
for train, test in oob.split(np.array([1, 2, 3, 4, 5])):
    print(train, test)
```

```
[3 4 0 1 3] [2]
[0 0 1 4 4] [2 3]
[1 2 4 2 4] [0 3]
```

The reason why I chose a object-oriented implementation is that we can plug it into scikit-learn's `cross_val_score` function, which is super convenient.

https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__5.ipynb

OOB Bootstrap
Object Oriented API

```
from mlxtend.data import iris_data
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split


X, y = iris_data()

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=123, stratify=y)


model = DecisionTreeClassifier(random_state=123)
```

```
bootstrap_scores = \
    cross_val_score(model, X_train, y_train,
                    cv=BootstrapOutOfBag(n_splits=200, random_seed=123))

print('Mean Bootstrap score', np.mean(bootstrap_scores))
print('Score Std', np.std(bootstrap_scores))
```

```
Mean Bootstrap score 0.9483980861793887
Score Std 0.03981732245301400 4
```

```
lower = np.percentile(bootstrap_scores, 2.5)
upper = np.percentile(bootstrap_scores, 97.5)
print('95%% Confidence interval: [%.2f, %.2f]' % (100*lower, 100*upper))
```

```
95% Confidence interval: [83.33, 100.00]
```

https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__5.ipynb

OOB Bootstrap Functional API

4.1.5 OOB Bootstrap with Percentile Method

Same as "4.1.2 Normal Approximation-based Bootstrap Interval (Based on OOB Estimates)" but using mlxtend

```
from mlxtend.evaluate import bootstrap_point632_score

bootstrap_scores = bootstrap_point632_score(clf,
                                            X_train, y_train,
                                            n_splits=200,
                                            method='oob',
                                            random_seed=12345)

bootstrap_oob_mean = np.mean(bootstrap_scores)
print('Mean Bootstrap score', bootstrap_oob_mean)
print('Score Std', np.std(bootstrap_scores))

bootstrap_oob_percentile_lower = np.percentile(bootstrap_scores, 2.5)
bootstrap_oob_percentile_upper = np.percentile(bootstrap_scores, 97.5)
```

```
Mean Bootstrap score 0.9552421700709434
Score Std 0.027704014141397008
```

```
print(bootstrap_percentile_lower, bootstrap_percentile_upper)
```

```
0.8977324263038549 1.0
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

The 0.632 Bootstrap Method

- In 1983, Bradley Efron described the *.632 Estimate*, a further improvement to address the pessimistic bias of the bootstrap [1].

- 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 samples from the original dataset.

[1] Efron, Bradley. 1983. "Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation." *Journal of the American Statistical Association* 78 (382): 316. doi:10.2307/2288636.

Bootstrap Sampling

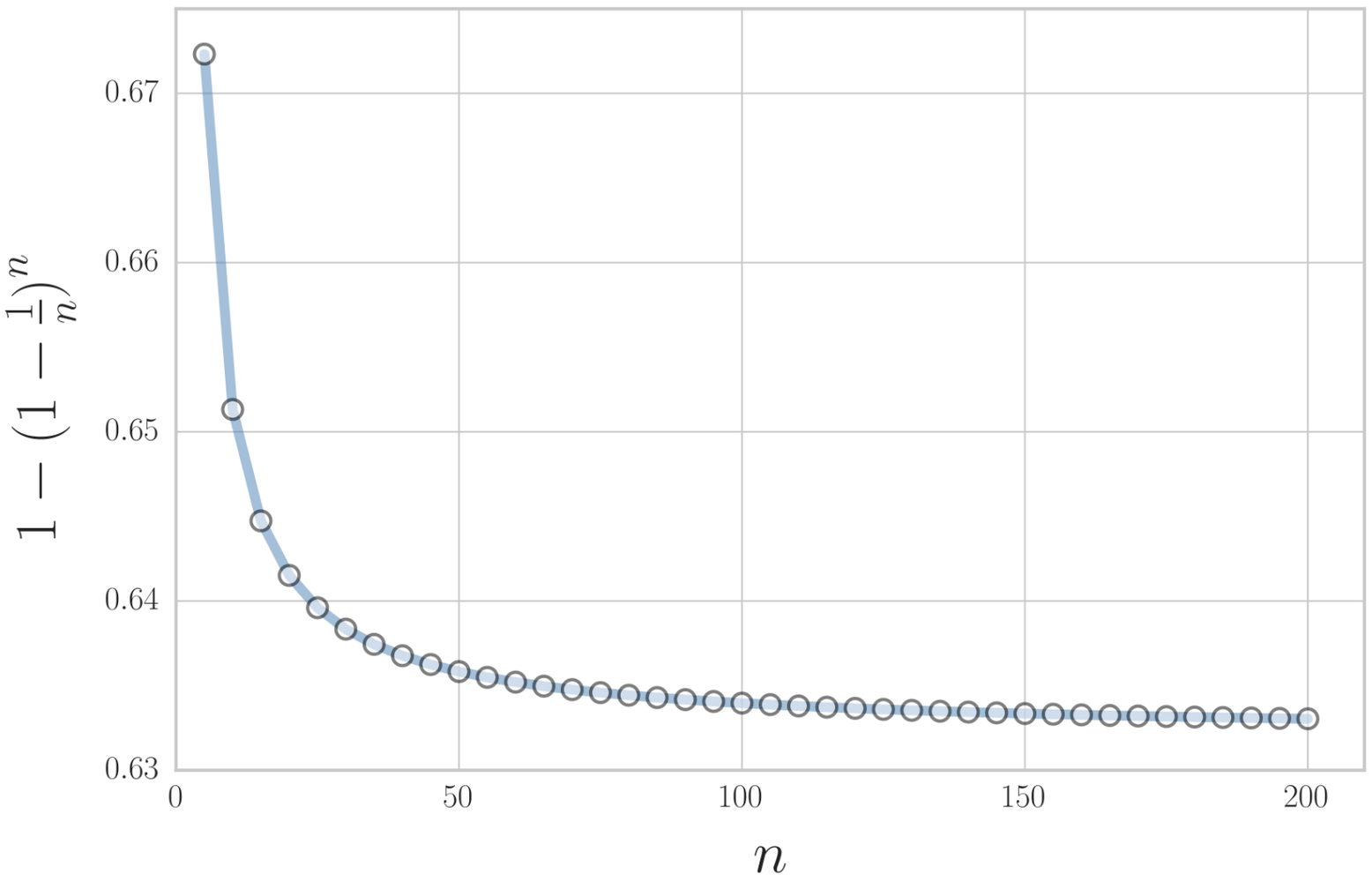
$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^n,$$

$$\frac{1}{e} \approx 0.368, \quad n \to \infty \, .$$

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

$$\frac{1}{e} \approx 0.368, \quad n \to \infty\,.$$

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



The .632 Bootstrap Method

The *.632 Estimate,* is computed via the following equation:

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

where

$\text{ACC}_{r,i}$ is the resubstitution accuracy

$\text{ACC}_{h,i}$ is the accuracy on the out-of-bag sample.

[1] Efron, Bradley. 1983. “Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation.” *Journal of the American Statistical Association* 78 (382): 316. doi:10.2307/2288636.

4.1.6 .632 Bootstrap

The .632 Bootstrap is the default setting of `bootstrap_point632_score`; it tends to be overly optimistic.

```
bootstrap_scores = bootstrap_point632_score(clf,
                                            X_train, y_train,
                                            n_splits=200,
                                            random_seed=12345)

bootstrap_632_mean = np.mean(bootstrap_scores)
print('Mean Bootstrap score', bootstrap_632_mean)
print('Score Std', np.std(bootstrap_scores))

bootstrap_632_percentile_lower = np.percentile(bootstrap_scores, 2.5)
bootstrap_632_percentile_upper = np.percentile(bootstrap_scores, 97.5)
```

```
Mean Bootstrap score 0.967105807390348
Score Std 0.01662536156386783 6
```

```
print(bootstrap_632_percentile_lower, bootstrap_632_percentile_upper)
```

```
0.9309740507436 57 0.9942047244094488
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

The .632+ Bootstrap Method

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 Efron and Robert Tibshirani proposed the *The .632+ Bootstrap Method* [1].

Instead of using a fixed "weight" $\omega = 0.632$ in

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

we compute the weight as $$\omega = \frac{0.632}{1 - 0.368 \times R},$$

where R is the *relative overfitting rate*

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

[1] Efron, Bradley, and Robert Tibshirani. 1997. "Improvements on Cross-Validation: The .632+ Bootstrap Method." *Journal of the American Statistical Association* 92 (438): 548. doi:10.2307/2965703.

The .632+ Bootstrap Method

R is the *relative overfitting rate*

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

Now, 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 the examples and
target class labels:

$$\gamma = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \left(1 - L\left(h(x^{[i]}), f(x^{[i]})\right) \, .$$

[1] Efron, Bradley, and Robert Tibshirani. 1997. “Improvements on Cross-Validation: The .632+ Bootstrap Method.” *Journal of the American Statistical Association* 92 (438): 548. doi:10.2307/2965703.

4.1.7 .632+ Bootstrap

The .632+ Boostrap method attempts to address the optimistic bias of the regular .632 Boostrap.

```
bootstrap_scores = bootstrap_point632_score(clf, X_train, y_train,
                                            n_splits=200,
                                            method='.632+',
                                            random_seed=12345)


bootstrap_632plus_mean = np.mean(bootstrap_scores)
print('Mean Bootstrap score', bootstrap_632plus_mean)
print('Score Std', np.std(bootstrap_scores))
```

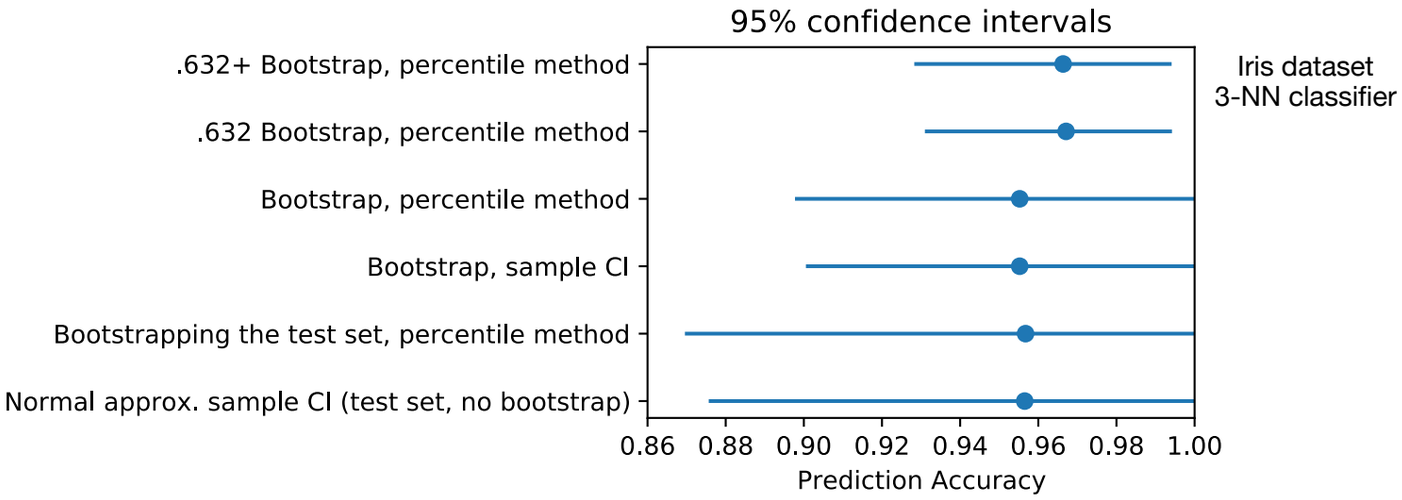
```
Mean Bootstrap score 0.9663500334312962
Score Std 0.01757293812762708
```

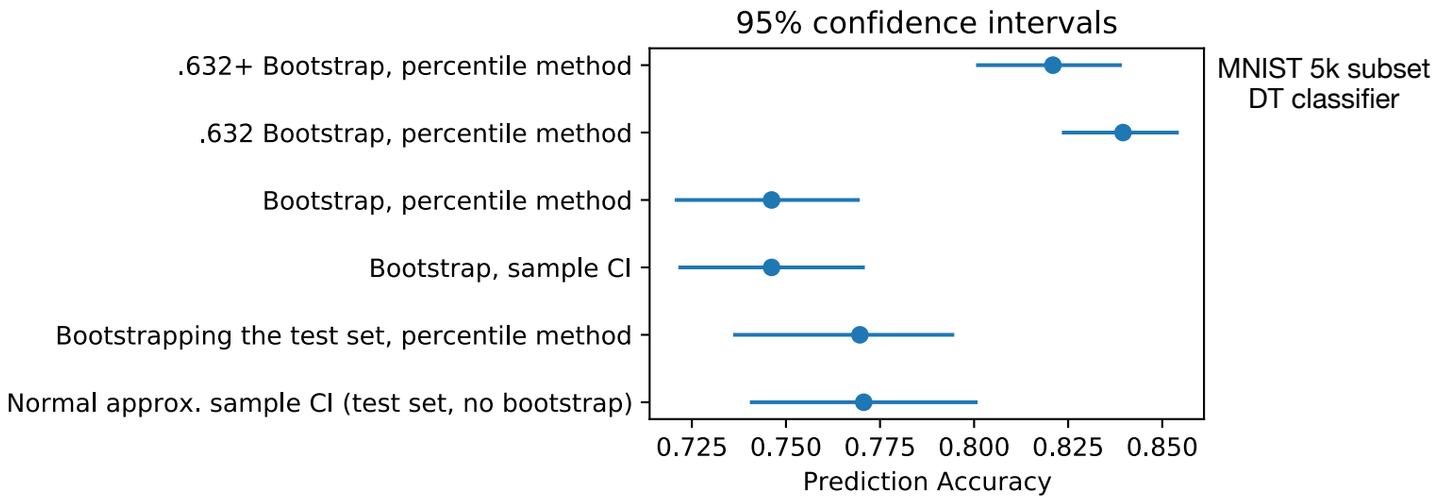
```
bootstrap_632plus_percentile_lower = np.percentile(bootstrap_scores, 2.5)
bootstrap_632plus_percentile_upper = np.percentile(bootstrap_scores, 97.5)
```

```
print(bootstrap_632plus_percentile_lower, bootstrap_632plus_percentile_upper)
```

```
0.9282626011381683 0.9941168892152473
```

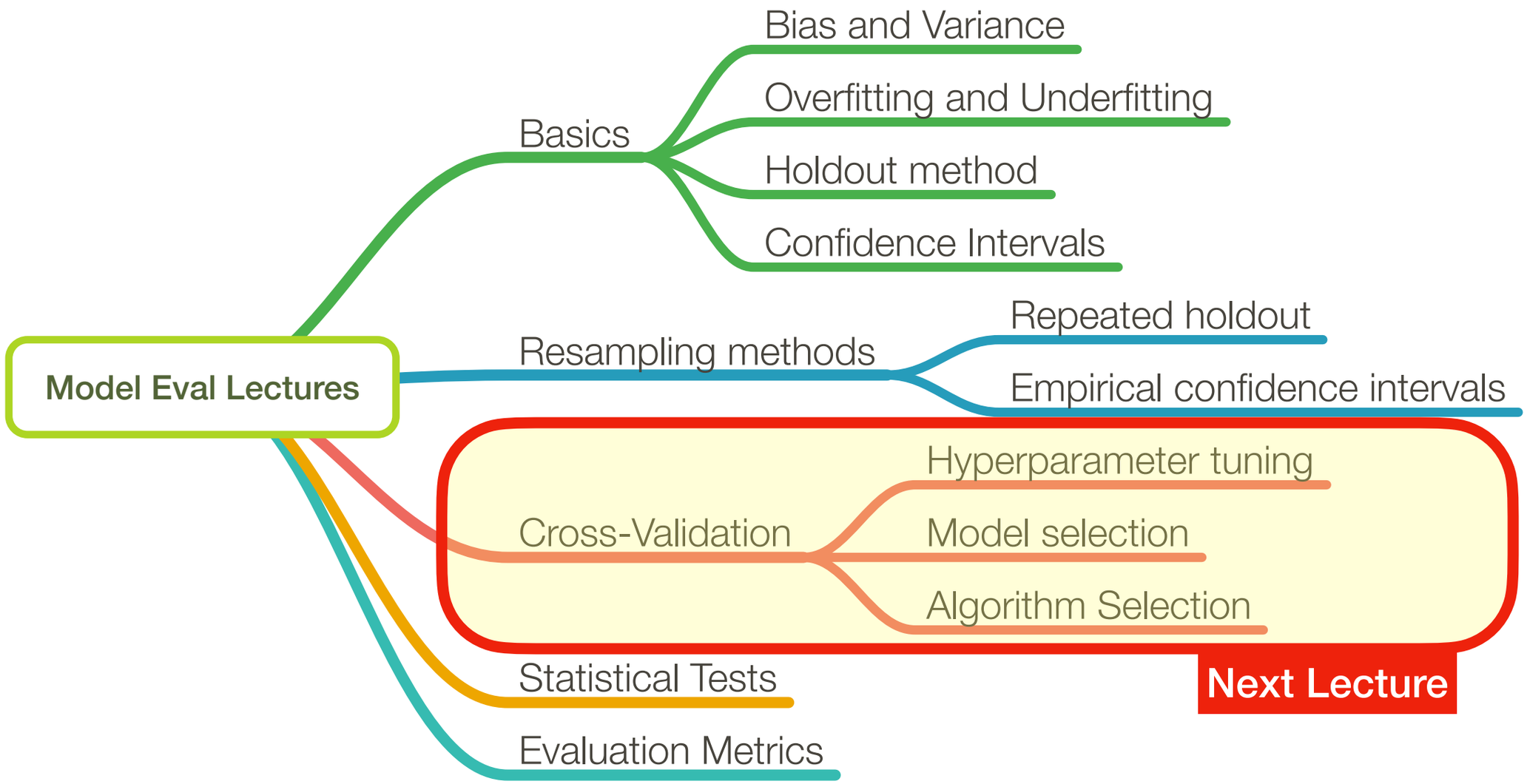
Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb





Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L09/code/09-eval2-ci__4-confidence-intervals_iris.ipynb

Overview


Model Eval Lectures
Basics
Bias and Variance
Overfitting and Underfitting
Holdout method
Confidence Intervals
Resampling methods
Repeated holdout
Empirical confidence intervals
Cross-Validation
Hyperparameter tuning
Model selection
Algorithm Selection
Next Lecture
Statistical Tests
Evaluation Metrics