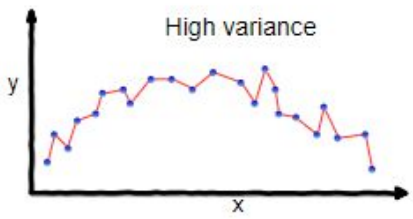
BIAS / VARIANCE & CROSS VAL

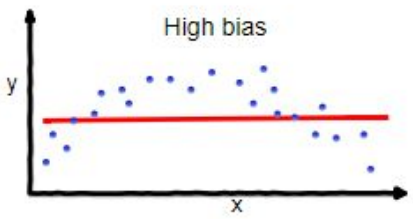
Chyld @ Galvanize

BIAS / VARIANCE

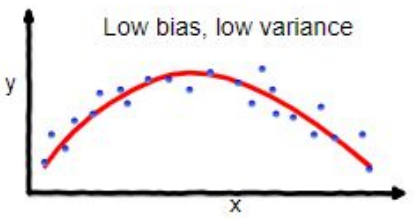
Bias / Variance Graphical Overview



overfitting



underfitting



Good balance

MODELING A SINE WAVE WITH LINEAR FUNCTIONS

$f : [-1, 1] \rightarrow \mathbb{R}$ $\quad f(x) = \sin(\pi x)$

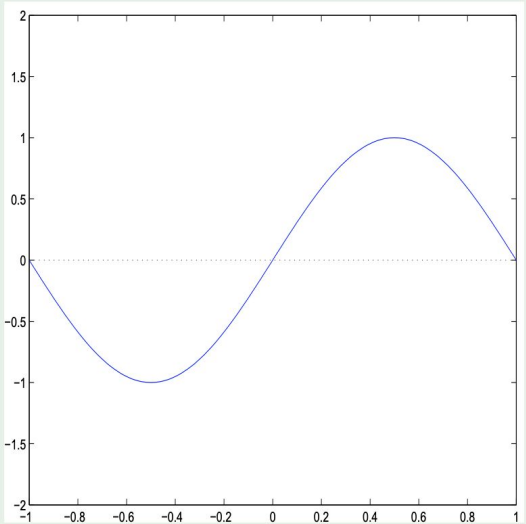
Only two training examples! $\quad N = 2$

Two models used for learning:

$\mathcal{H}_0$: $\quad h(x) = b$

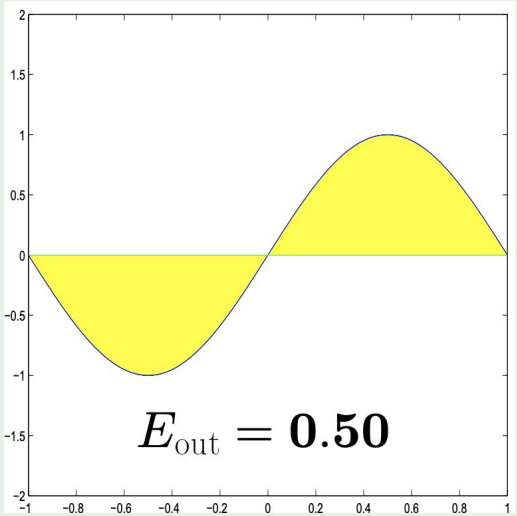
$\mathcal{H}_1$: $\quad h(x) = ax + b$

Which is better, $\mathcal{H}_0$ or $\mathcal{H}_1$?

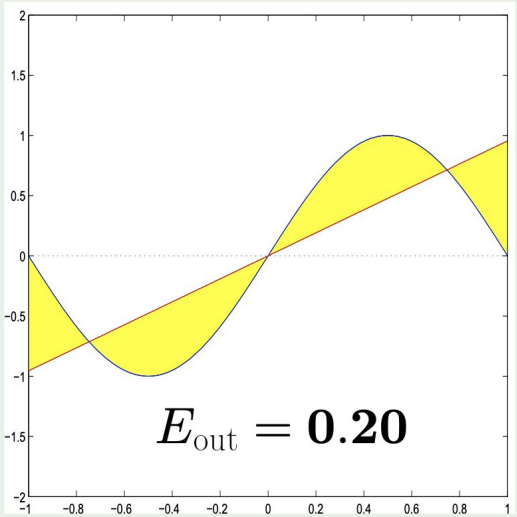


COMPUTING ERROR IF TARGET FUNCTION IS KNOWN

$\mathcal{H}_0$

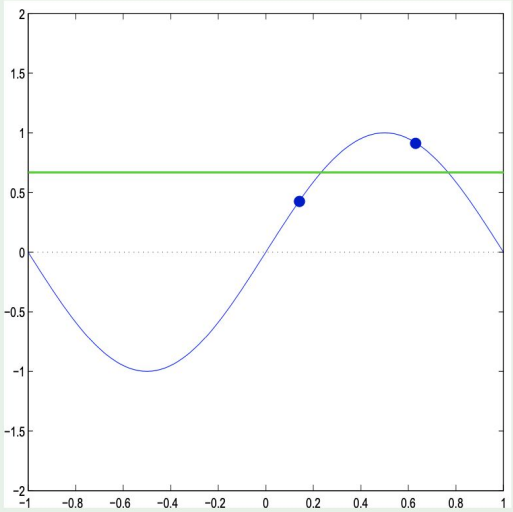


$\mathcal{H}_1$

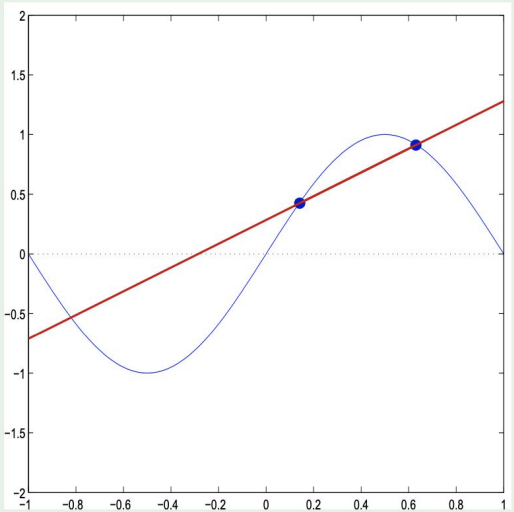


RANDOMLY DRAWING 2 SAMPLES AND FITTING CURVES

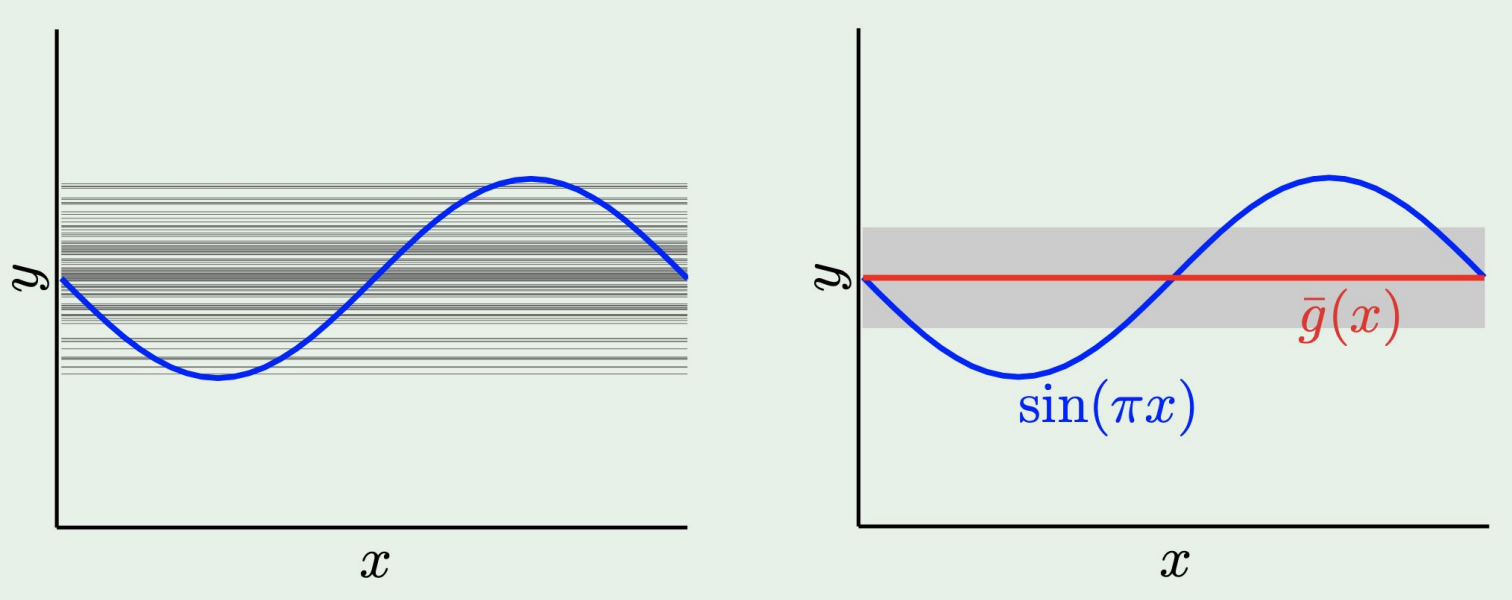
$\mathcal{H}_0$



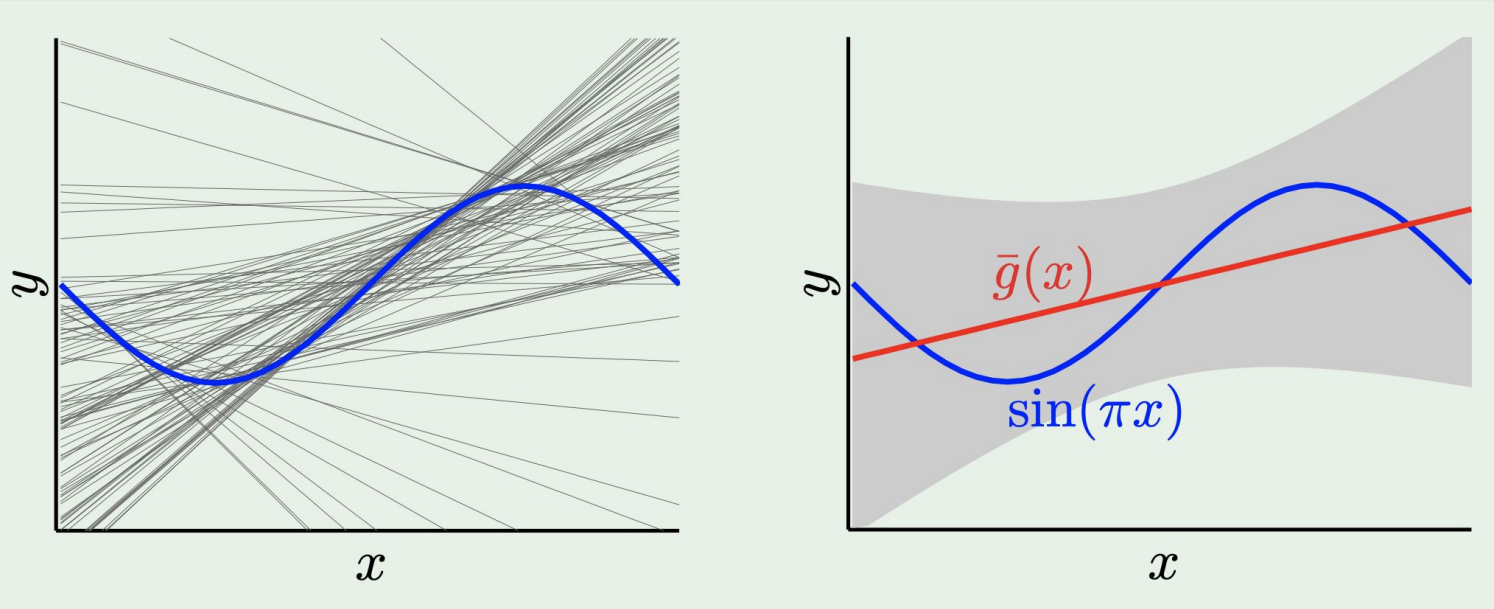
$\mathcal{H}_1$



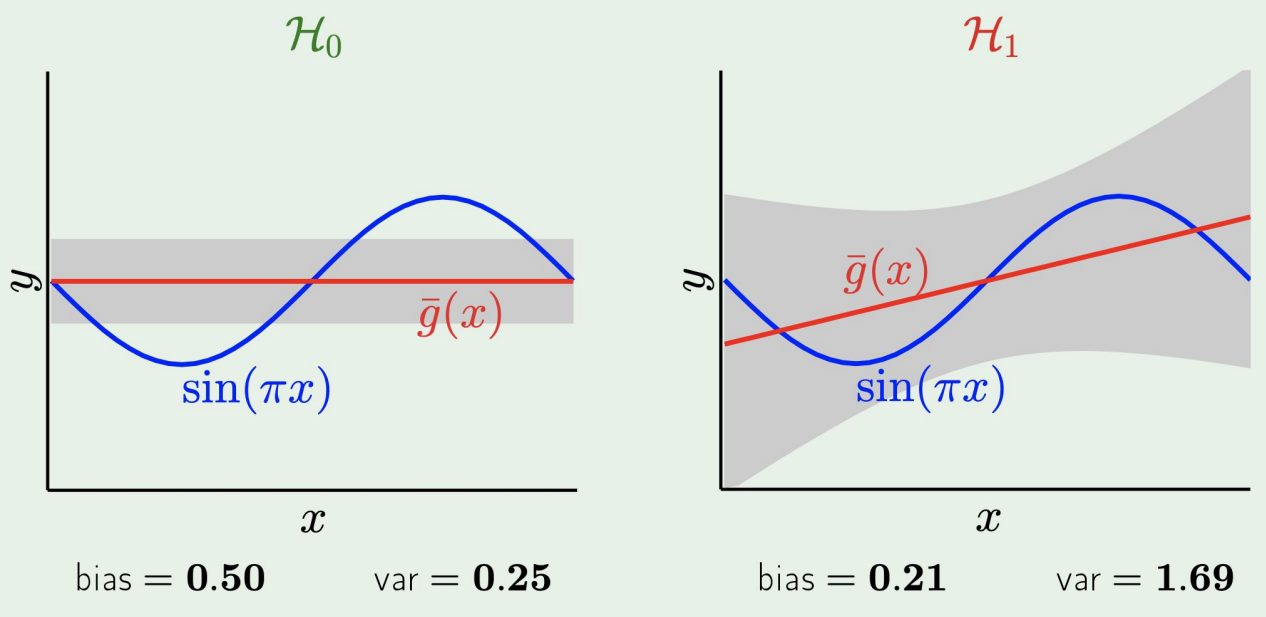
PRODUCING MANY HO MODELS FROM DIFFERENT DATA



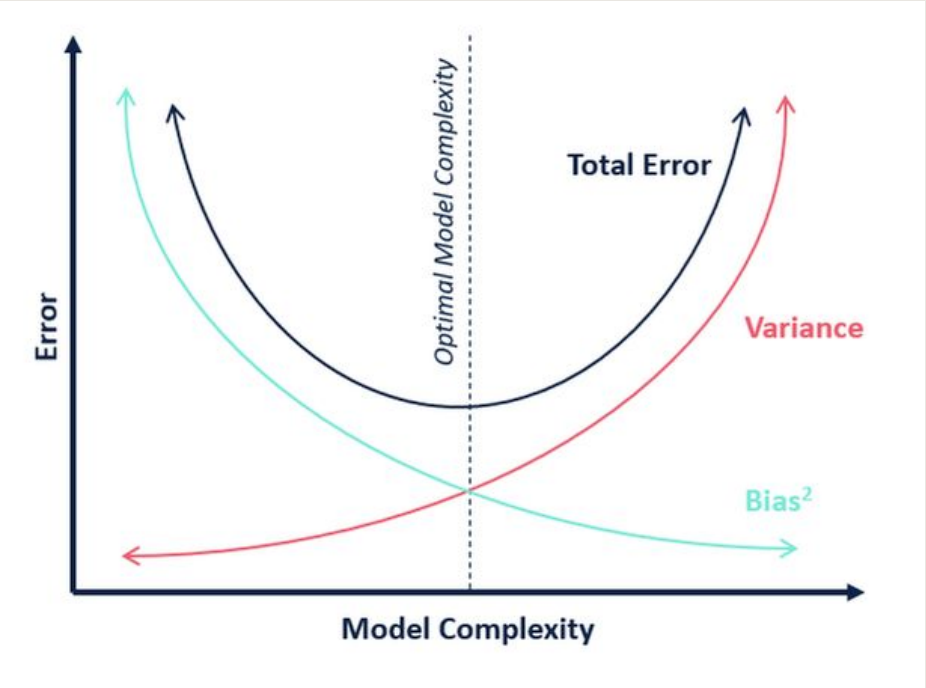
PRODUCING MANY H1 MODELS FROM DIFFERENT DATA

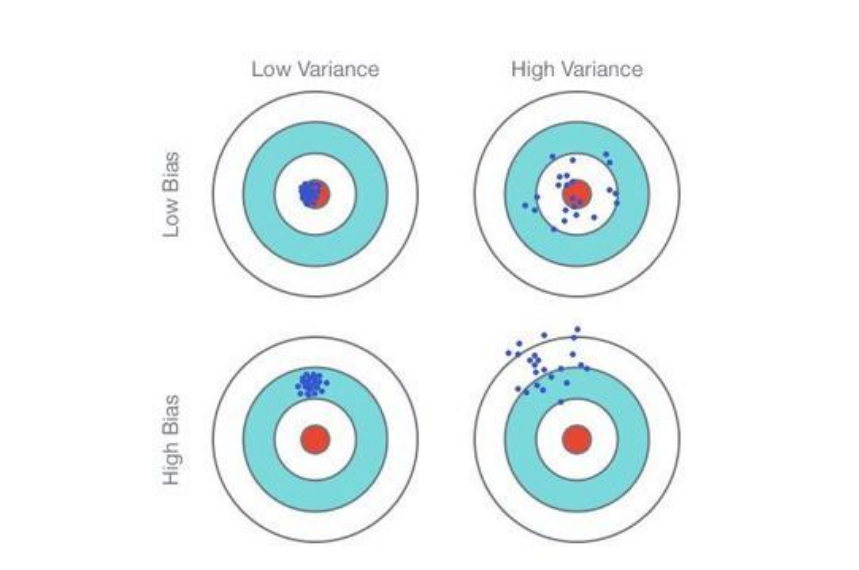


HO IS BEST OVERALL MODEL; IT HAS LOWEST TOTAL ERROR

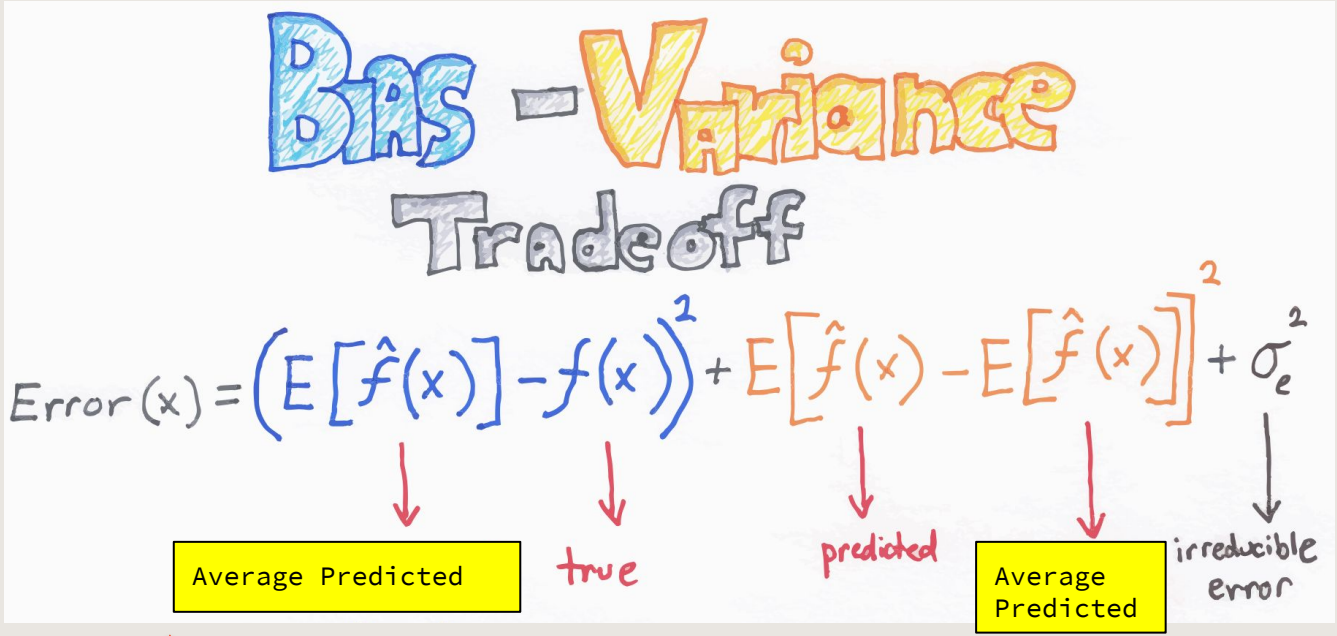


BIAS / VARIANCE TRADEOFF





BIAS / VARIANCE FORMULA



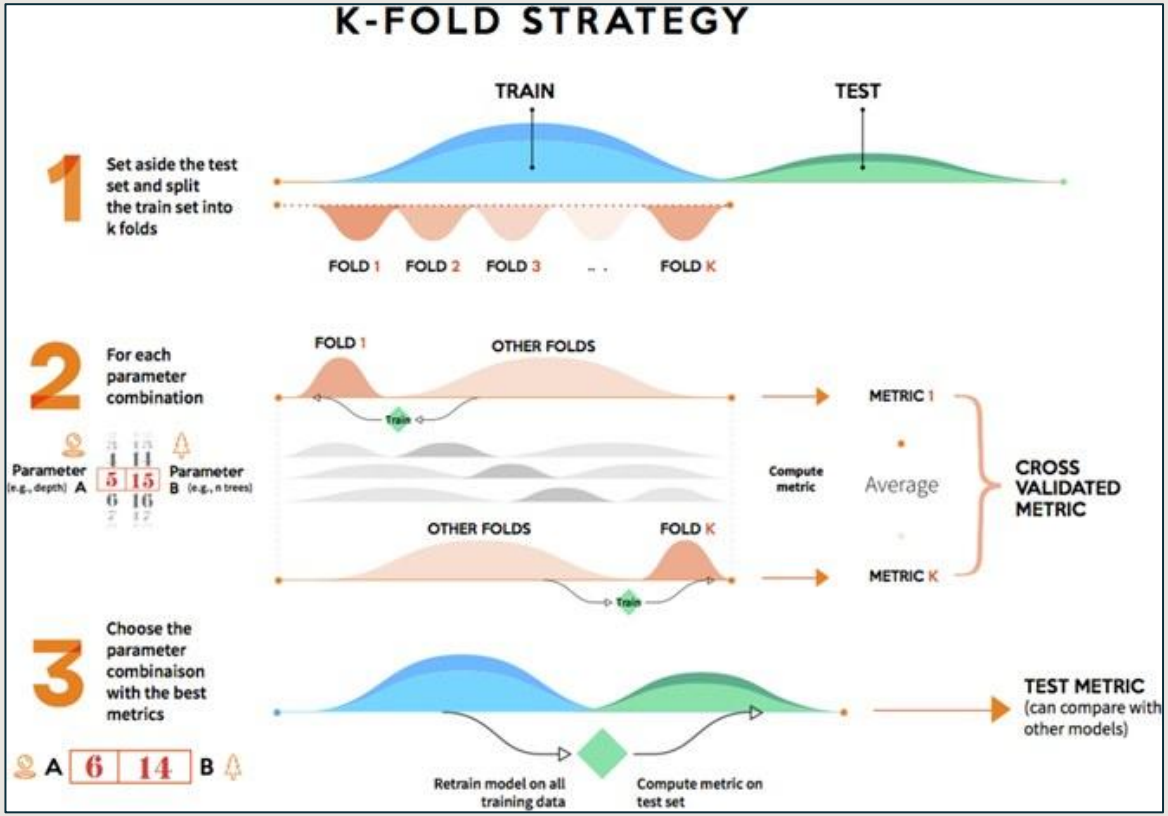
CROSS VALIDATION

PURPOSE OF CROSS VALIDATION

"Cross-validation ... is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set."
- Wikipedia

We use cross-validation for two things:

1) Attempting to quantify how well a model (of some given complexity) will predict on an unseen data set
2) Tuning hyperparameters of models to get best predictions.



K-FOLD STRATEGY

1. Split all data into 70% train, 30% test
    a. holdout set same name as test
2. Let's use K=3 as example
3. The 70% train will be split into 3 sets
    a. train1, train2, train3
4. You will build three models with the same hyperparameters
    a. model1
        i. trains on train1, train2
        ii. evaluate model on train3
    b. model2
        i. trains on train1, train3
        ii. evaluate model on train2
    c. model3
        i. trains on train2, train3
        ii. evaluate on train1
5. Let's say you care about accuracy, so you average the accuracy of model1, model2 and model3
6. Repeat steps 4 and 5 for many different hyperparameters
7. Pick the hyperparameters with the best average metric
8. Train final model with chosen hyperparameters on entire train data set and evaluate with test or holdout set.

SKLEARN API

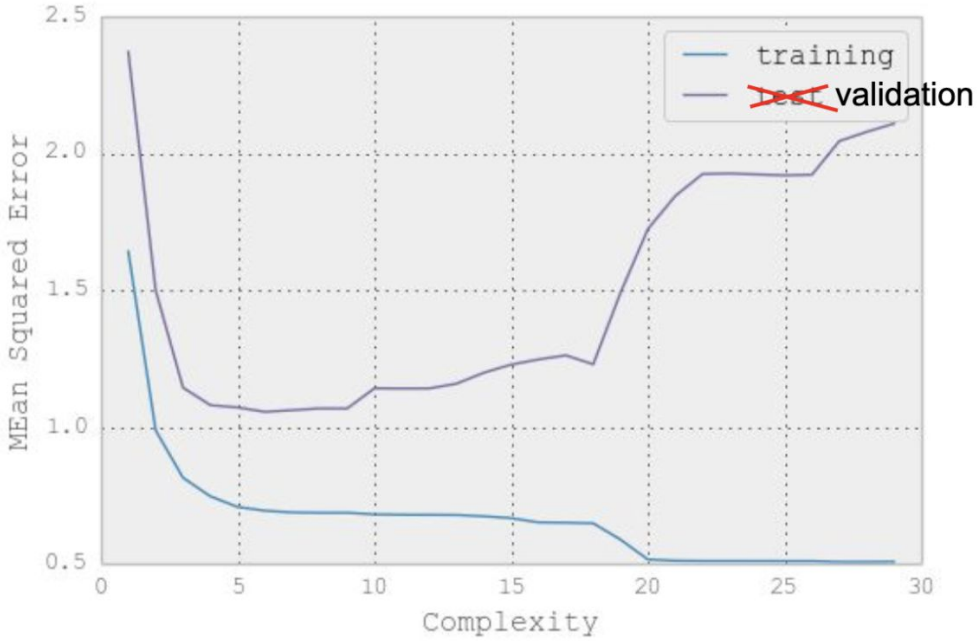
sklearn.model_selection.train_test_split

sklearn.model_selection.**train_test_split**(**arrays*, ***options*) [source]

sklearn.model_selection.KFold

class sklearn.model_selection.**KFold**(*n_splits=5*, *shuffle=False*, *random_state=None*) [source]

CHOOSING THE RIGHT MODEL



Which error is most important to minimize? Why?

What model complexity is best? How did you decide?

At the optimum model complexity, what is the bias? What is the variance?

Is it possible for the test error to be lower than the training error?

Number of features, interaction between features, order of features

WHAT IF YOUR MODEL IS OVERFITTING

1. **Get more data…** (not usually possible/practical)
2. **Subset Selection:** keep only a subset of your predictors (i.e, dimensions)
3. **Regularization:** restrict your model's parameter space (Wednesday)
4. **Dimensionality Reduction:** project the data into a lower dimensional space (later in course)

SUBSET SELECTION

Best subset: Try every model. Every possible combination of *p* predictors

- Computationally intensive. 2^p possible subsets of *p* predictors
- High chance of finding a "good" model by random chance.

  … A sort-of monkeys-Shakespeare situation …

Stepwise: Iteratively pick predictors to be in/out of the final model.

- Forward, backward, forward-backward strategies
  - Forward: starting with just one and adding more features, one-by-one
  - Backward: starting with them all, and removing one-by-one
- Sklearn features only backward recursive elimination.

Imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold

%matplotlib inline
```

Load Iris Data

```
iris = load_iris()
X = iris.data
y = iris.target
```

JUPYTER DEMO

Display Data

```
X[:3]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2]])
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```