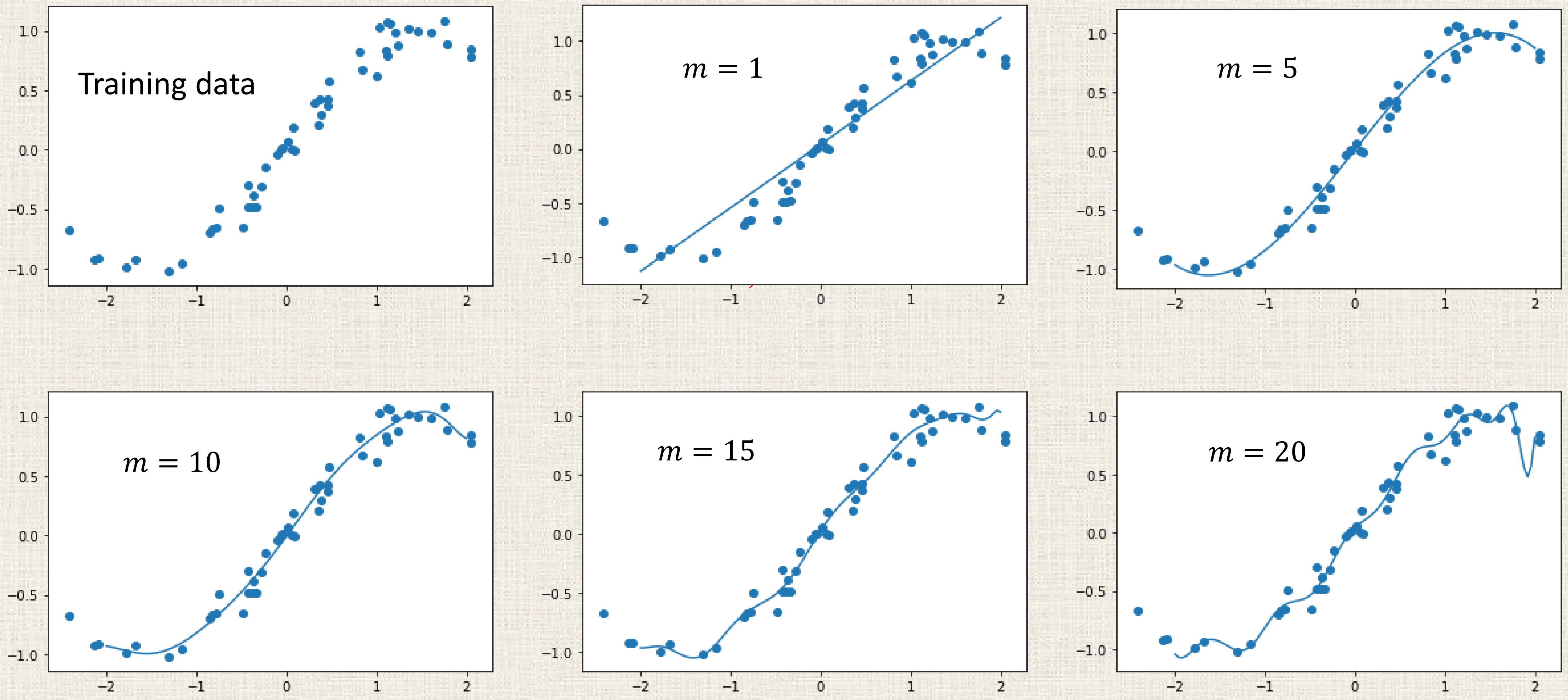
Performance Evaluation and Model Selection

Underfitting and Overfitting Models



- Underfitting: the model used is too simple to catch the trend behind the given data set;
- Overfitting: the model is so powerful; it not only catch the trend but also the noise in the data set;

- **Model selection:** which model should we use to achieve the best performance?

- **Performance evaluation:** Can we use the training error (SSE in the example) to measure the performance of the trained models (fitted curves)?

- Do we have a procedure to find the “best” model for a given data set?

Parameters in the supervised machine learning problems

$$\boldsymbol{w}_{Ridge} = \underset{\boldsymbol{w}}{argmin}\{\|\boldsymbol{\Phi w} - \boldsymbol{y}\|_{\boldsymbol{2}}{}^{2} + \lambda\|\boldsymbol{w}\|_{2}{}^{2}\} = \left(\boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{T}\boldsymbol{y}$$

Assuming polynomial model is used in this ridge regression problem.

- The weight vector, $\boldsymbol{w}$, is called the ***model parameter***, it is the ***decision variable***, a parameter to be optimized.

- The order of polynomial model, m, and the regularization weight, λ, are not decision variables, they are called ***hyperparameters***.

    - m is a ***model hyperparameter***, since it determines the structure of the model.

    - λ is not part of the model. Rather, it is an aspect of the optimization process used to fit the model. It is called an ***optimization hyperparameter***.

Types of Error (true error vs training error)

- Assume that the data are distributed according to some (unknown) distribution $p(\boldsymbol{x}, y)$, and that we have a ***loss function*** l, which is to measure the error between the true output and our estimate $\hat{y} = h_{\boldsymbol{w}}(\boldsymbol{x})$.

- The ***true error*** of a particular hypothesis $h \in H$ is the ***expected loss over the whole data distribution*** (over the entire ***population***):

$$R(\boldsymbol{w}) = E_{\boldsymbol{x},y}[l(h_{\boldsymbol{w}}(\boldsymbol{x}), y)]$$

- Ideally, we would find the hypothesis that minimizes the ***true error***, i.e.,

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{argmin}\, R(\boldsymbol{w})$$

- However, computing this expectation is impossible because we do not have access to the true data distribution.

- Rather, we have access to samples $(\boldsymbol{x}_i, y_i)$ that follows the same distribution.

- These enable us to approximate the real problem by minimizing the ***training error (in-sample error)***:

$$\hat{R}_{Train}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} l(h_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i)$$

- But since we have a finite number of samples, ***the hypothesis that performs the best on the training data is not necessarily the best on the whole data distribution***.

Training set vs Test set

- A common solution is to set aside some portion (say 30%) of the data (***held-out samples***), to be called the ***test set***, which is ***disjoint*** from the training set and ***not allowed to be used when fitting the model***:

- We can use this test set to estimate the **true error** by the ***test error*** (***out-of-sample error***) :

$$\hat{R}_{test}(\boldsymbol{w}) = \frac{1}{M}\sum_{i=1}^{M} l(h_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i)$$

- ***Never touch the test set until the final fitted model is decided***.

For example, let's solve the OLS problem on the Iris data set.

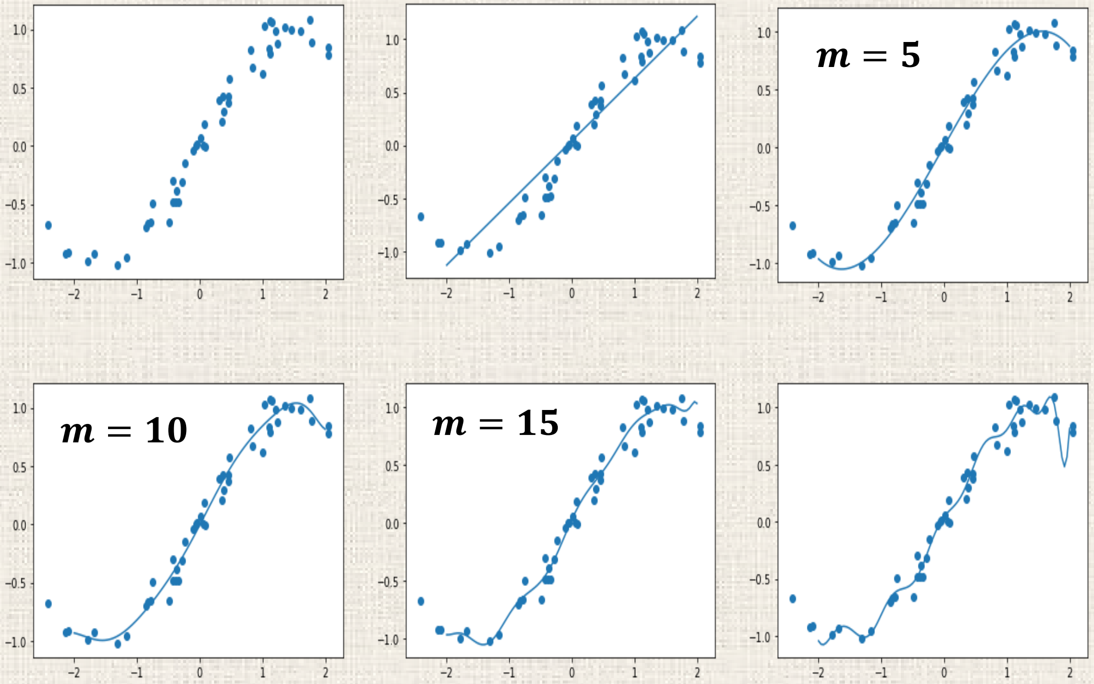
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	**5.1**	**3.5**	**1.4**	**0.2**	**0**
1	**4.9**	**3**	**1.4**	**0.2**	**0**
2	**4.7**	**3.2**	**1.3**	**0.2**	**0**
3	**4.6**	**3.1**	**1.5**	**0.2**	**0**
4	**5**	**3.6**	**1.4**	**0.2**	**0**
…					
50	**7**	**3.2**	**4.7**	**1.4**	**1**
51	**6.4**	**3.2**	**4.5**	**1.5**	**1**
52	**6.9**	**3.1**	**4.9**	**1.5**	**1**
53	**5.5**	**2.3**	**4**	**1.3**	**1**
54	**6.5**	**2.8**	**4.6**	**1.5**	**1**
…					
144	**6.7**	**3.3**	**5.7**	**2.5**	**2**
145	**6.7**	**3**	**5.2**	**2.3**	**2**
146	**6.3**	**2.5**	**5**	**1.9**	**2**
147	**6.5**	**3**	**5.2**	**2**	**2**
148	**6.2**	**3.4**	**5.4**	**2.3**	**2**
149	**5.9**	**3**	**5.1**	**1.8**	**2**

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{argmin}\{L(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2{}^2\}$$
$$= \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

- The entire data set is split into training set (100 data points) and test set (50 data points)
- The training set is used to find $\boldsymbol{w}^*$
- The test set is used to evaluate the performance (true error) of $\boldsymbol{w}^*$

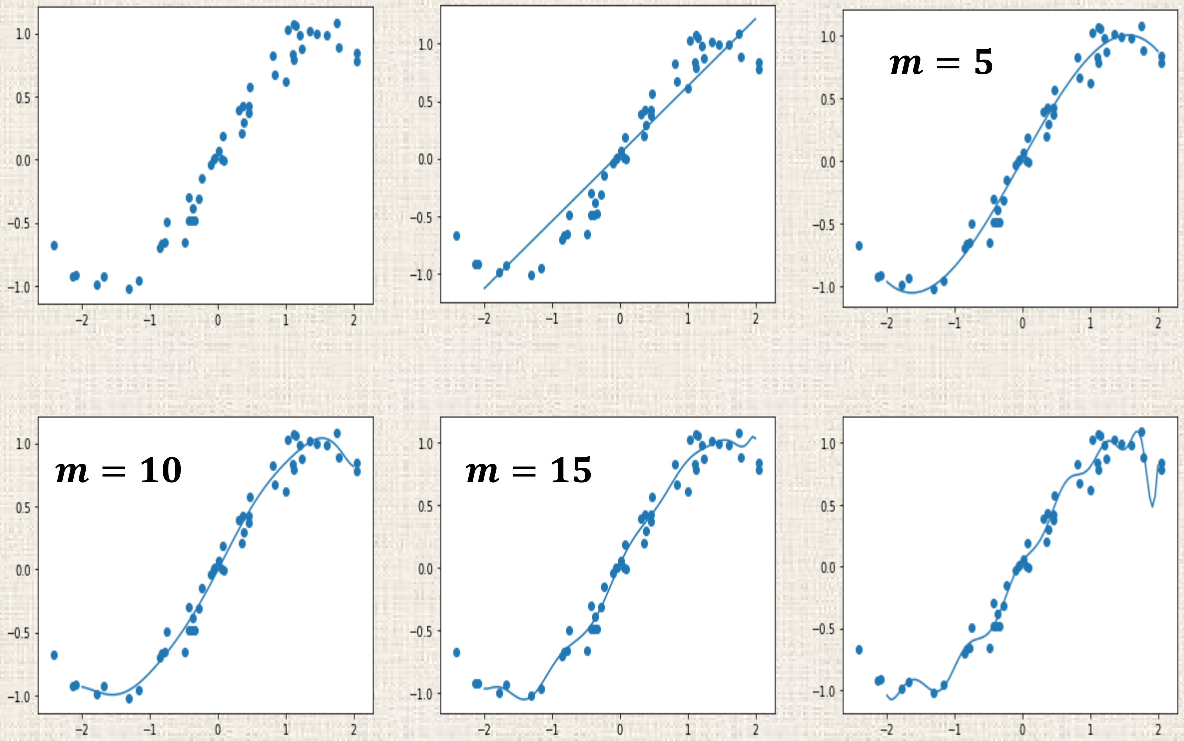
Training set vs validation set

- **To tune hyperparameters**, we need to evaluate the performance of the fitted model.
- For example, we want to find out which m value (out of 5,10, 15) in the polynomial models fit the given data set the best?



- First, the given data set is split into two parts: the training set (70%), and the test set (30%).
- Then, the following is carried out:
  - The polynomial model with $m = 5$ is trained on the training set and find the optimal weight vector $\boldsymbol{w}_1^*$ (fitted model #1)
  - The polynomial model with $m = 10$ is trained on the training set and find the optimal weight vector $\boldsymbol{w}_2^*$ (fitted model #2)
  - The polynomial model with $m = 15$ is trained on the training set and find the optimal weight vector $\boldsymbol{w}_3^*$ (fitted model #3)
- What are the differences among $\boldsymbol{w}_1^*, \boldsymbol{w}_2^*, and\ \boldsymbol{w}_3^*$?

- To compare the performance of these three models, corresponding ***training errors*** can not be used.
- The ***test set*** is reserved to evaluate the performance of the finalized model.
- What should we do?



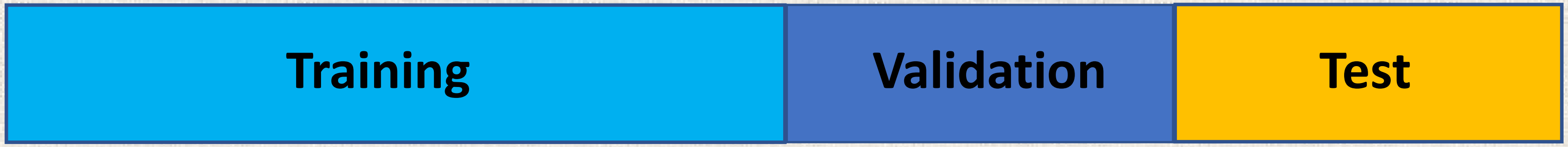
- Understanding that the test set will be only used to evaluate the finalized model, this evaluation can only be carried out on the training set.

- Hence, in many cases, ***we need to split the training set into two parts:***
  - one part (called training set) will be used to training the fitting model
  - the other part (called ***validation set***) will be used for evaluating the fitted model with certain hyperparameter values.
  - We can use this validation set to estimate the true error by the ***validation error***:

$$\hat{R}_{VAL}(\boldsymbol{w}) = \frac{1}{K}\sum_{i=1}^{K} l(h_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i)$$

- With this estimate, we have a simple method for choosing hyperparameter values: ***try a bunch of configurations of the hyperparameters and choose the one that yields the lowest validation error***

- Note that the ***purpose of validation set is to determine the best values of the hyperparameters! (used for hyper-parameters tuning)***
- Then, the tuned model (the finalized model) will be tested using the test data set.
- Hence, in general the original data is split into the following:

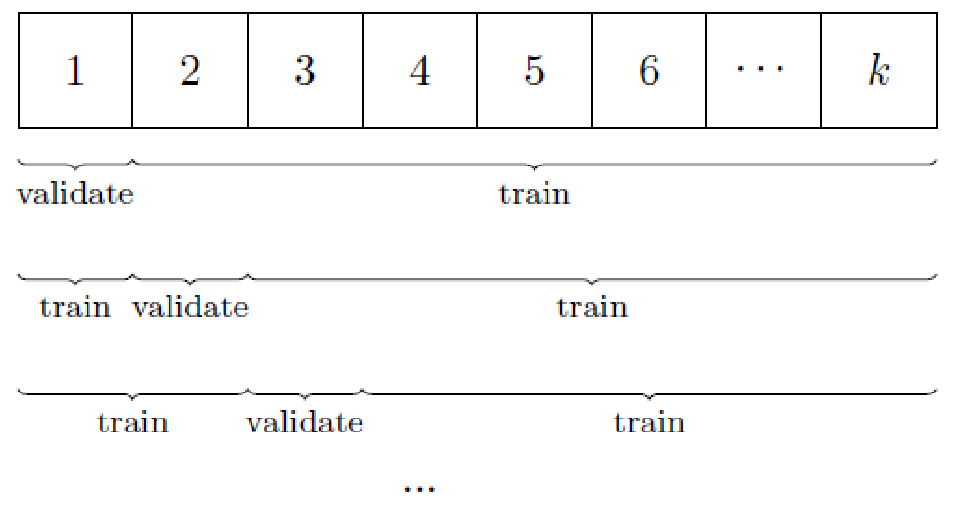


- ***Setting aside a validation set works at a cost!***
- We can not use the validation data for training.
- This is especially important when we have little data to begin with or collecting more data is expensive.

Cross-validation

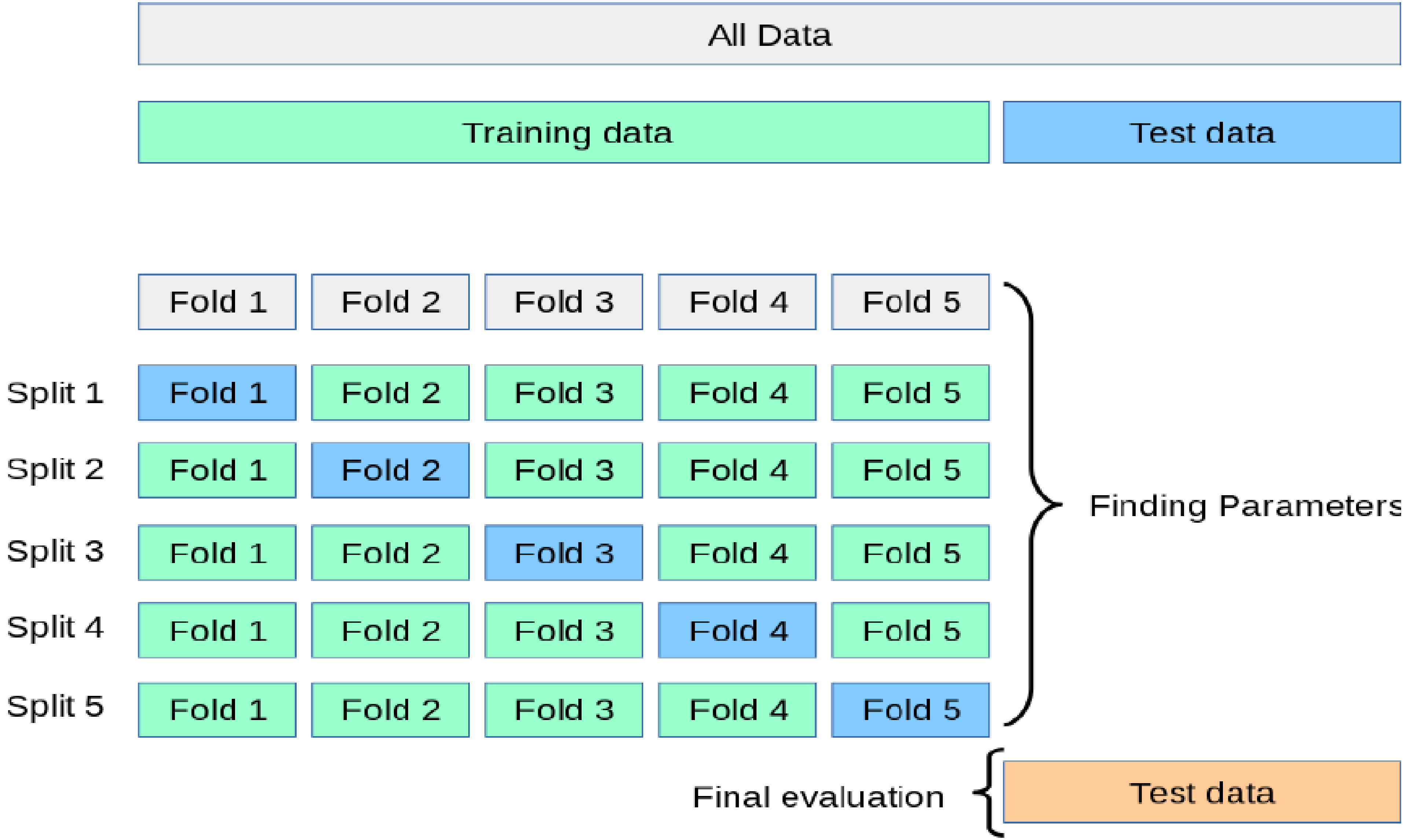
- ***Cross-validation*** is an alternative to having a dedicated validation set.

- ***k-fold cross-validation*** works as follows:
  1. Shuffle the training data and partition it into k equally-sized blocks.
  2. For $i = 1, \ldots, k$
     - Train the model on all the data except block i
     - Evaluate the model (i.e., compute the validation error) using block i
  3. Average the k validation errors; this is the final estimate of the true error.



Cross-validation

- Note that this process (except for the shuffling and partitioning) must be repeated for every hyperparameter configuration we wish to test.
- This is roughly k times as much computation required using the held-out validation set.
- This can be very ***expensive*** when the model takes a long time to train.


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

Training Data

(optimize the model parameters)

Validation Data

(optimize the model's hyperparameters)

Test Data

(evalute the optimized model)

Tuning the Hyperparameters

- **Hyperparameters are not model parameters, and they can not be directly trained from the data.**
- **The general process of tuning hyper-parameters:**
  - Define a model
  - Define the range of possible values of all hyperparameters
  - Define a method for sampling hyperparameter values
  - Define an evaluation criteria to judge the model
  - Define a cross-validation method

- **Grid Search:** an exhaustive sampling of the hyperparameter space. i.e., we simply build a model for each possible combination of all the hyperparameter values provided, evaluate each model and select the structure which produces the best results.

- For example: Consider a ***random forests classifier***, which is an ***ensemble model*** comprised of a collection of ***decision trees***. This model has two hyperparameters:

  N_trees: the number of decision trees included, with values $[10,50,100]$
  M_depth: maximum depth allowed for each decision tree model, with values $[3,5]$

  Performing grid search would generate the following models to be evaluated:

  Random_forest_classifier(N_tree=10,M_depth=3)

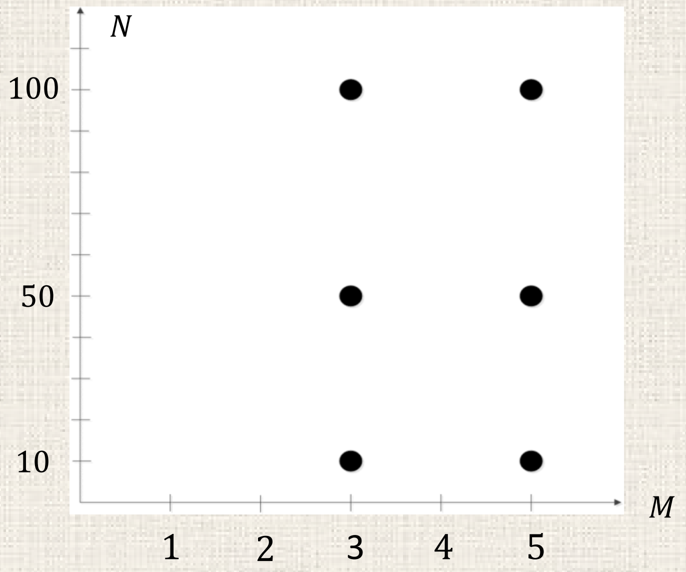
  Random_forest_classifier(N_tree=10,M_depth=5)

  Random_forest_classifier(N_tree=50,M_depth=3)

  Random_forest_classifier(N_tree=50,M_depth=5)

  Random_forest_classifier(N_tree=100,M_depth=3)

  Random_forest_classifier(N_tree=100,M_depth=5)



- Each of these model will then be evaluated on a validation set or using cross-validation method.

- **Random Search:** tries random combinations of a range of values of hyperparameters.

  - It is good in testing a wide range of values and normally it reaches a very good combination very fast, but it does not guarantee to give the best parameter combination.

  - Random search has a probability of 95% of finding a combination of parameters within the 5% optima with only 60 iterations.

- Consider any distribution in a hyperparameter space with a finite maximum.

- Let's randomly sample points from this space and check if any of them lands in the 5% interval within the maximum. Each trial will have 5% chance that the sample point is within that interval.

- If the sampling within each trial is independent, then, the probability that all n trials missed the desired interval is $(1-0.05)^n$.

- So, the probability that at least one trial hit the desired interval after n trials is $1-(1-0.05)^n$. If we want this probability to be larger than 95%, i.e.,

$$1-(1-0.05)^n > 0.95$$

- The number of trials can be found as $n \geq 60$.

Goodness-of-fit (performance metrics of regression model)

The following are the popular performance metrics used to evaluate a trained regression model:

- **Mean Squared Error (MSE):** (mean_squared_error in sk-learn)

  Let $\hat{y}_i$ be the predicted value of the i^{th} sample, and y_i be the corresponding target value. Then the mean squared error estimated over n samples is defined as:

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

- **Mean Absolute Error (MAE):** (mean_absolute_error in sk-learn)

  Let $\hat{y}_i$ be the predicted value of the i^{th} sample, and y_i be the corresponding target value. Then the mean absolute error estimated over n samples is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **R^2 score: coefficient of determination** (r2_score in sk-learn)

Let $\hat{y}_i$ be the predicted value of the i^{th} sample, and y_i be the corresponding target value. Then the R^2 score estimated over n samples is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Where,

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

R^2 is a statistical measure of how well the regression predictions approximate the real data points.

For more detailed information on performance evaluation of regression models using sk-learn, visit the website:

https://scikit-learn.org/stable/model_selection.html

Bias-Variance Decomposition of Model Prediction Error

- Recall from the regression problem, our goal is to estimate a hypothesis model $h(\boldsymbol{x})$ to approximate the unknown model $f(\boldsymbol{x})$ governing the sampled data set.
- One question is: ***how exactly can we measure the effectiveness of a hypothesis model(model prediction error)?***
- Let's make some assumptions:
  - We have a training data set: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, $\boldsymbol{x}_i$ and y_i's are values sampled from random variables X_i and Y_i, that is, the training data set is a random sample from a random variable $\boldsymbol{\mathcal{D}}$, consisting of random variables X_i and Y_i.
  - For some arbitrary test input $\boldsymbol{x}$, we use the trained model to estimate corresponding output, $h(\boldsymbol{x}; \boldsymbol{\mathcal{D}})$, which depends on the random variable $\boldsymbol{\mathcal{D}}$ that was used to train h. Since $\boldsymbol{\mathcal{D}}$ is random, we will have a slightly different hypothesis model $h(\boldsymbol{x}; \boldsymbol{\mathcal{D}})$ every time we use a new training dataset.
  - Note that $\boldsymbol{x}$ and $\boldsymbol{\mathcal{D}}$ are completely independent from one another: $\boldsymbol{x}$ is a test point, while $\boldsymbol{\mathcal{D}}$ consists of the training data.

- Our objective is to, for a fixed test point $\boldsymbol{x}$, evaluate how closely the trained model can estimate (predict) the noisy observation Y corresponding to $\boldsymbol{x}$.
- We express our metric as the ***expected squared error*** between the prediction and the observation $Y = f(\boldsymbol{x}) + Z$:

$$\epsilon(\boldsymbol{x}, h) = E[(h(\boldsymbol{x}, \boldsymbol{\mathcal{D}}) - Y)^2]$$

The expectation here is over two random variables, $\boldsymbol{\mathcal{D}}\ and\ Y$.

- We need the following facts to decompose the error:
  - First, let's find the expectation and variance of Y, assuming $E[Z] = 0$:

$$E[Y] = E[f(\boldsymbol{x}) + Z] = f(\boldsymbol{x}) + E[Z] = f(\boldsymbol{x})$$

$$Var(Y) = Var(f(\boldsymbol{x}) + Z) = Var(Z)$$

  - Also, notice that for any random variable X, we have,

$$Var(X) = E[X^2] - (E[X])^2 \Rightarrow E[X^2] = Var(X) + (E[X])^2$$

- Let's use these facts to decompose the error:

$$\epsilon(\boldsymbol{x}, h) = E[(h(\boldsymbol{x}, \boldsymbol{\mathcal{D}}) - Y)^2]$$

$$= E[(h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})^2 - 2h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})Y + Y^2)] = E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})^2] - 2E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})Y] + E[Y^2]$$

$$= \left\{Var\left(h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})\right) + (E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})])^2\right\} - 2E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})]E[Y] + \{Var(Y) + (E[Y])^2\}$$

$$= \{(E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})])^2 - 2E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})]E[Y] + (E[Y])^2\} + Var\left(h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})\right) + Var(Y)$$

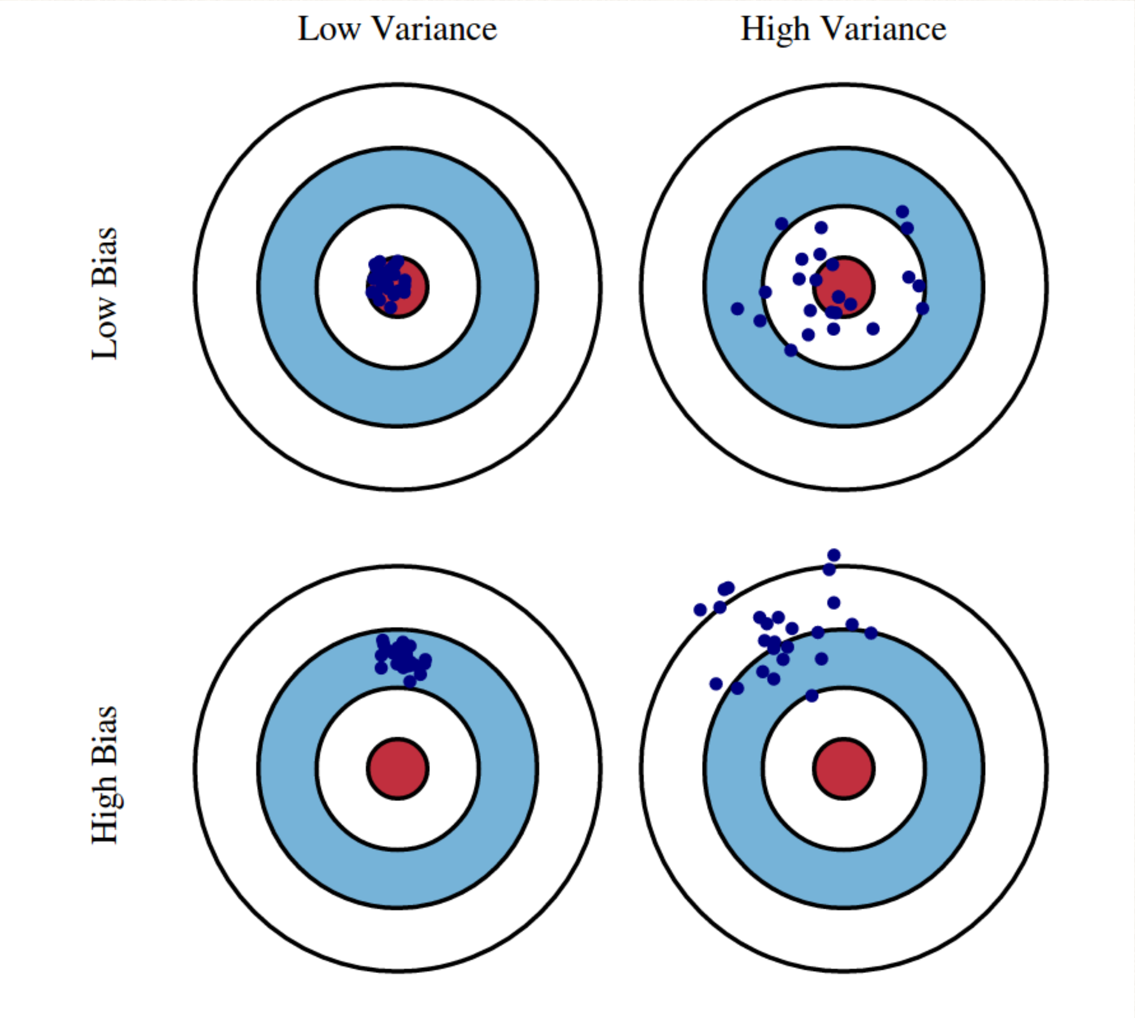
$$= \{E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})] - E[Y]\}^2 + Var\left(h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})\right) + Var(Y)$$

$$= \underbrace{\{E[h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})] - f(\boldsymbol{x})\}^2}_{\textbf{Bias}} + \underbrace{Var\left(h(\boldsymbol{x}, \boldsymbol{\mathcal{D}})\right)}_{\text{Variance}} + \underbrace{Var(Z)}_{\text{Noise}}$$

Now that the error is decomposed into three parts, where,

- $\{E[h(\boldsymbol{x},\boldsymbol{\mathcal{D}})] - f(\boldsymbol{x})\}^2$ is called the ***bias of model***: which measures how well the average hypothesis (over all possible training sets) can come close to the true underlying value $f(\boldsymbol{x})$, for a fixed value of $\boldsymbol{x}$.
  - A low bias means that on average the fitted model $h(\boldsymbol{x})$ accurately estimates $f(\boldsymbol{x})$.
- $Var\big(h(\boldsymbol{x},\boldsymbol{\mathcal{D}})\big)$ is the ***variance of model***: which measures the variance of the hypothesis (over all possible training sets), for a fixed value of $\boldsymbol{x}$.
  - A low variance means that the prediction does not change much as the training set varies.
- $Var(Z)$ is the ***irreducible error***: this is the error in the model that we can not control or eliminate, because it is due to errors inherent in the noisy observation Y.

$$expected\ squared\ error = bias + variance + noise$$



An experiment on Bias-Variance tradeoff

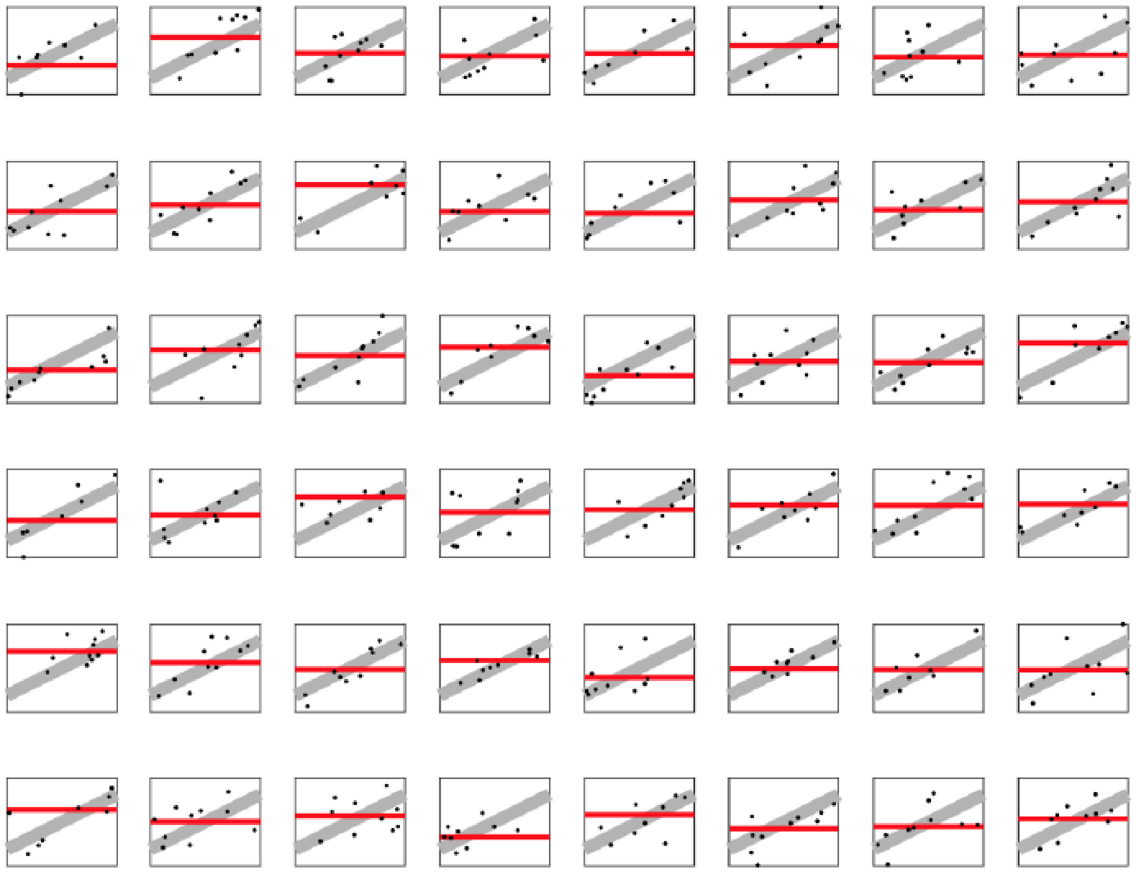
$$\boldsymbol{expected\ squared\ error = bias + variance + noise}$$

- Let's confirm the result of bias-variance decomposition with an experiment that measures the bias and variance for polynomial regression with 0 degree, 1^{st} degree, and 2^{nd} degree polynomials.

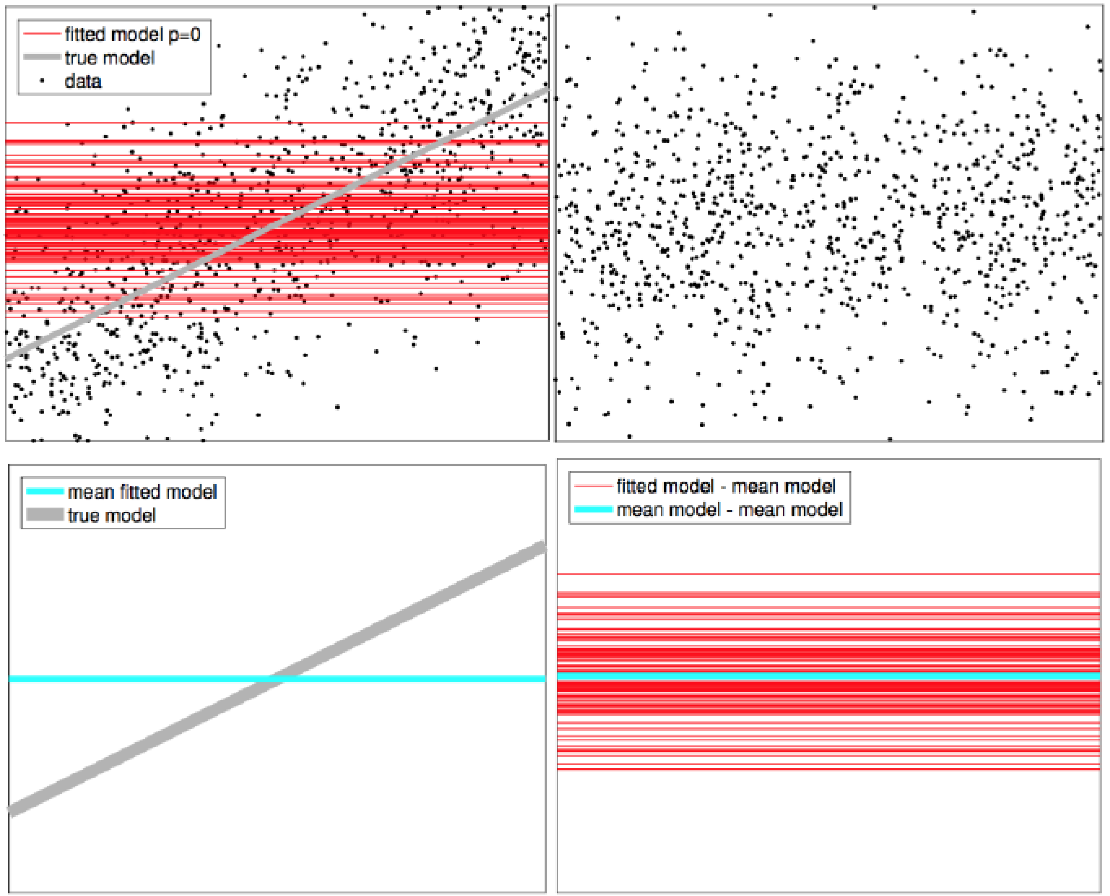
- In our experiment, we will repeatedly fit our hypothesis model to a random training set of 10 points drawn from a linear model $y = a_0 + a_1 x + z$, where $z \sim N(0, \sigma^2)$.

An experiment on Bias-Variance tradeoff

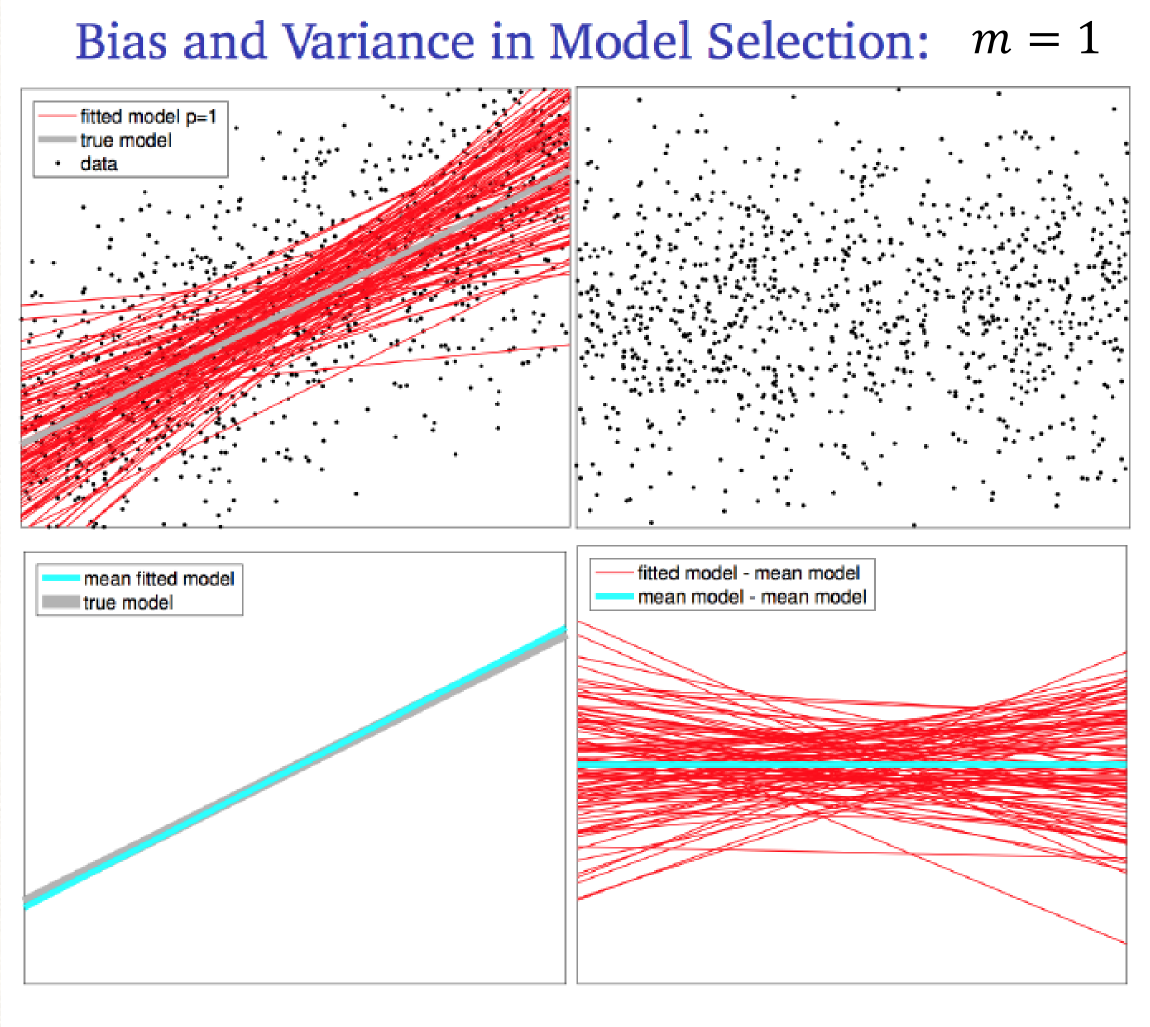
Fitting A Model over Multiple Datasets: $m = 0$



Bias and Variance in Model Selection: $m = 0$

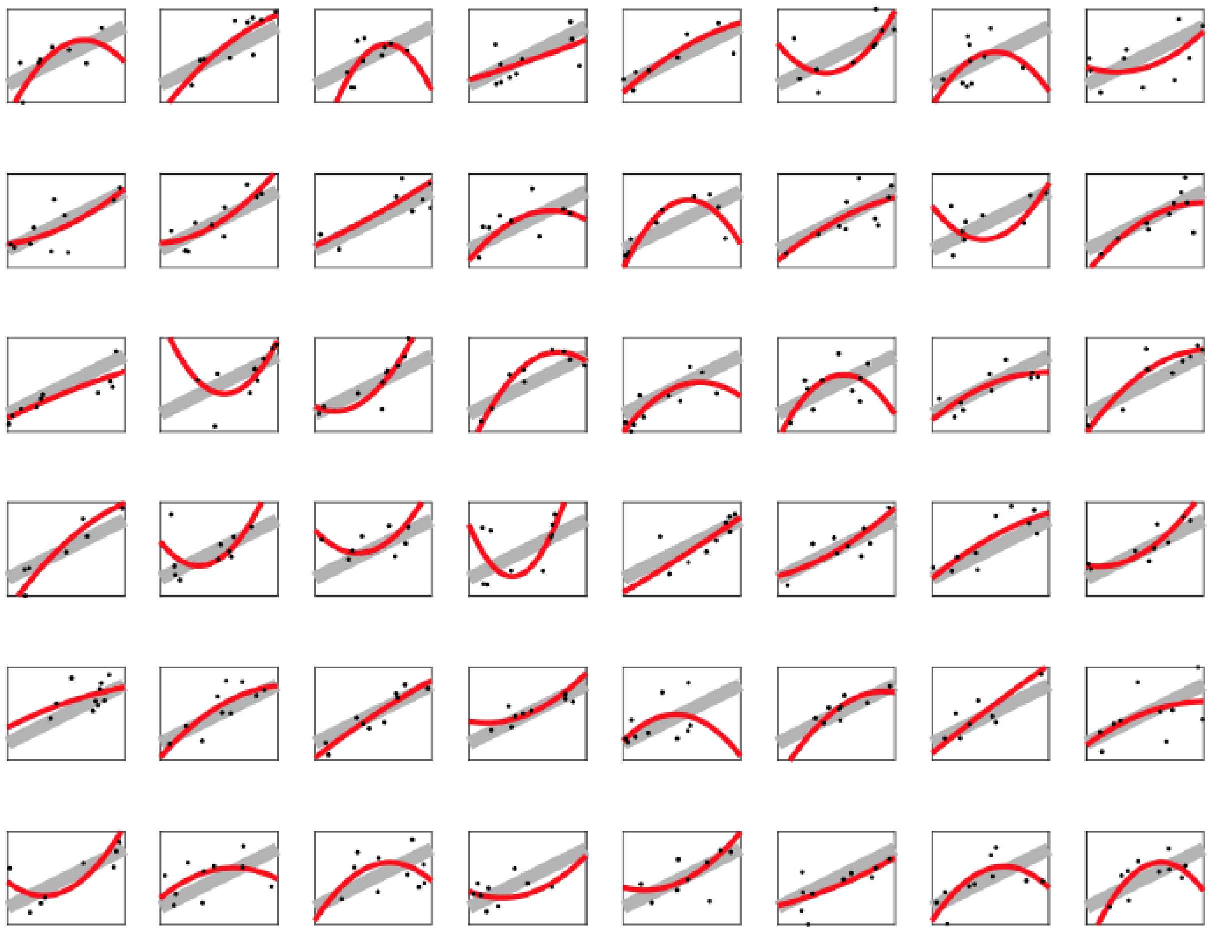


An experiment on Bias-Variance tradeoff

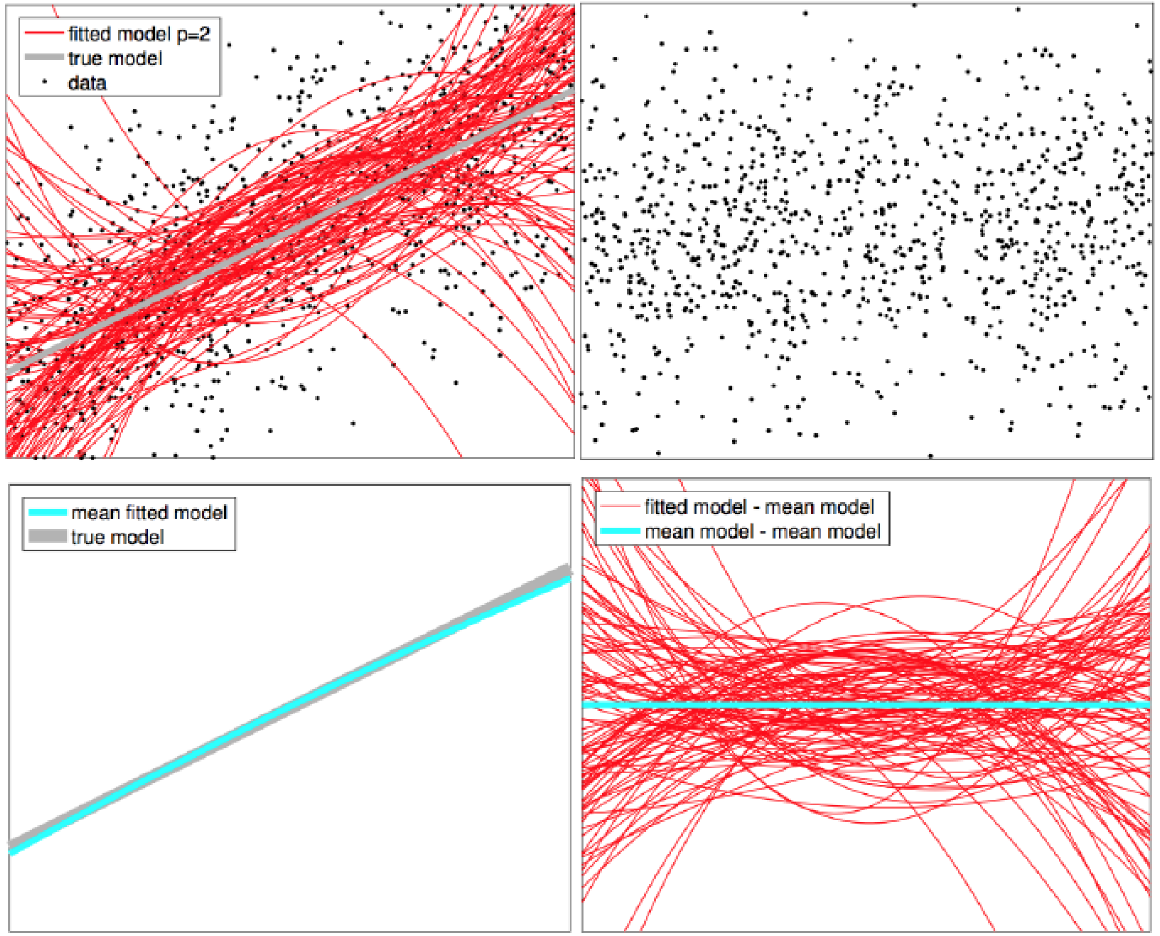


An experiment on Bias-Variance tradeoff

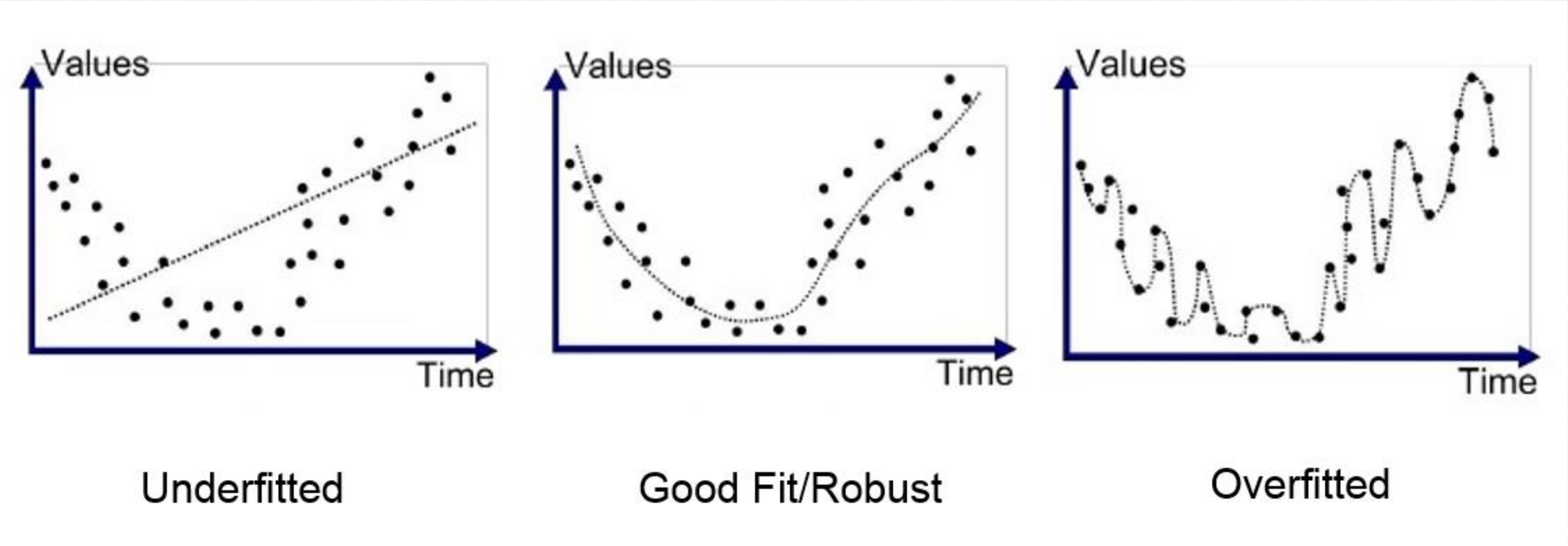
Fitting A Model over Multiple Datasets: $m = 2$



Bias and Variance in Model Selection: $m = 2$

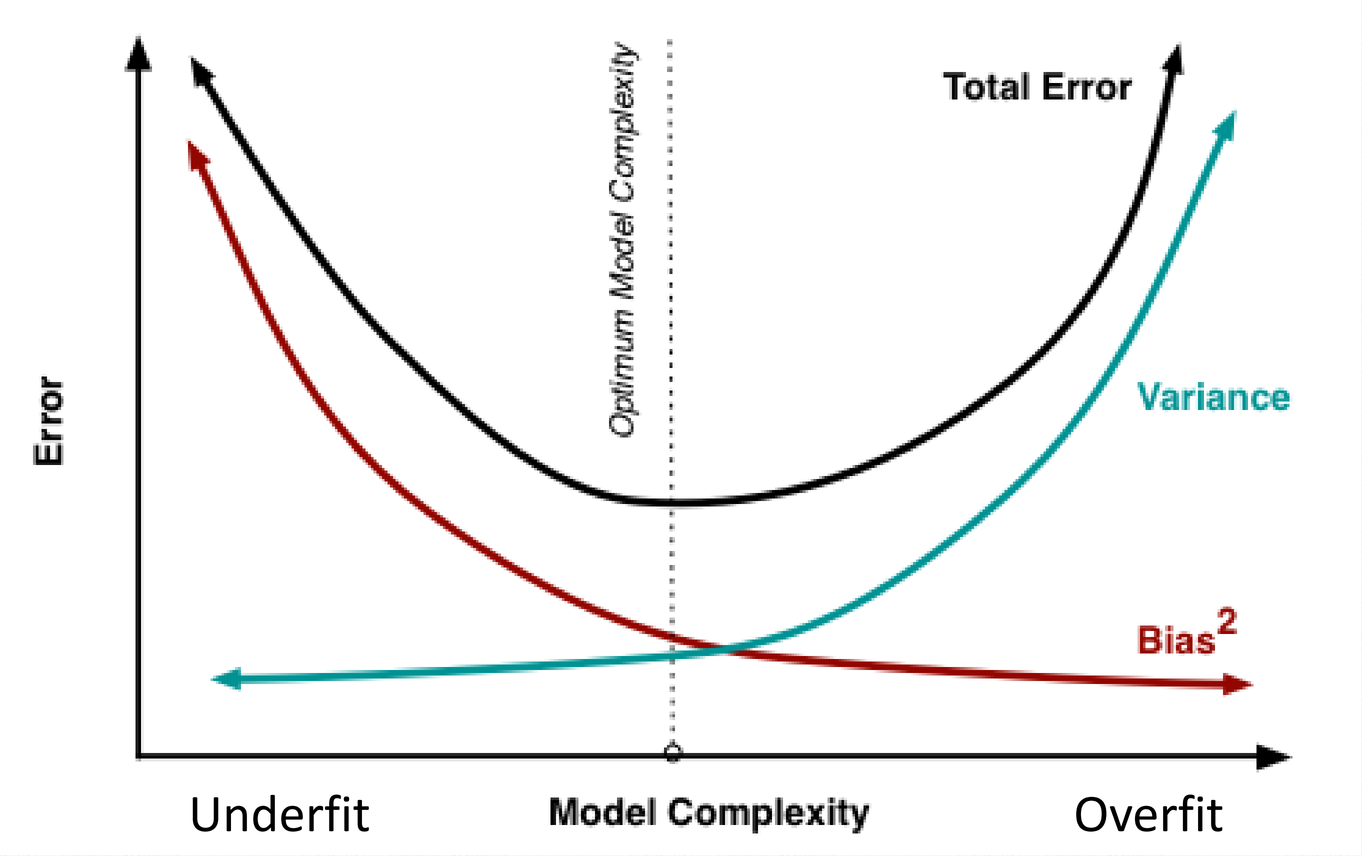


Underfitting and Overfitting



- ***Overfitting*** occurs when a statistical model or machine learning algorithm captures the noise of the data.
  - Intuitively, overfitting occurs when the model or algorithm fits the data too well.
  - Specifically, ***overfitting model or algorithm shows low bias but high variance***
  - Overfitting is often a result of an ***excessively complicated model***.
- ***Underfitting*** occurs when a statistical model or machine learning algorithm can not capture the underlying trend of the data.
  - Intuitively, underfitting occurs when the model or algorithm does not fit the data well enough.
  - Specifically, ***underfitting model or algorithm shows low variance but high bias***.
  - Underfitting is often a result of an ***excessively simple model***.
- ***Both underfitting and overfitting lead to poor prediction on new data sets.***

Bias-Variance tradeoff



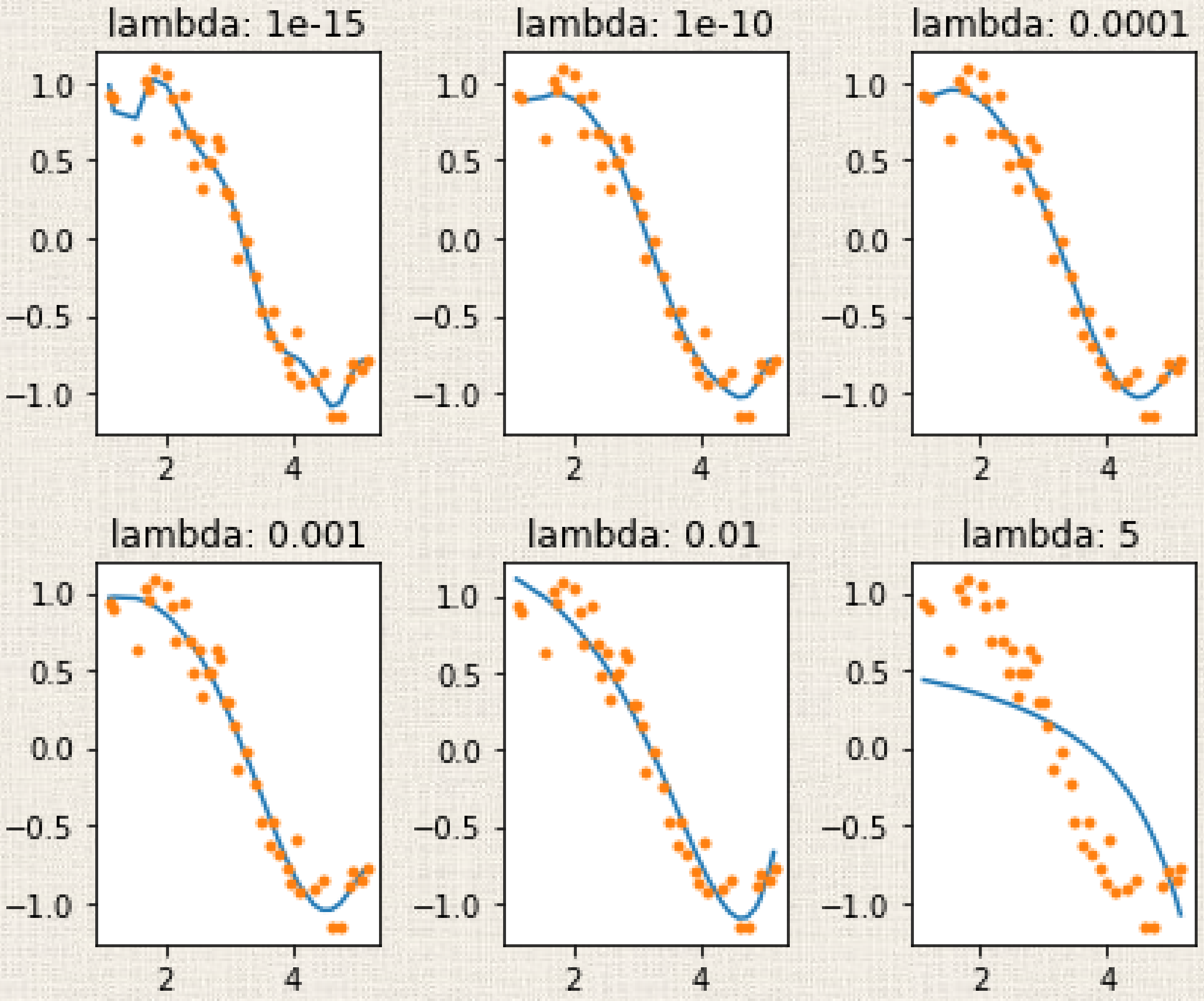
- ***Training error reflects bias but not variance. Test error reflects both***.
- ***In practice, if the training error is much smaller than test error, then there is overfitting***
- ***Decreasing the bias will increase the variance***
- ***Decreasing the variance will increase the bias***
- There is ***no escaping*** this relationship between bias and variance in a given supervised machine learning problem.
- There is a tradeoff between these two concerns for your problems.
- Use validation or cross-validation to ***tune some hyperparameters*** of your algorithm to achieve the trade-off

Regularization to overcome Overfitting

Loss function:(L2 regularization)

$$L(\boldsymbol{w}) = ||\boldsymbol{\Phi w} - y||^2 + \lambda||\boldsymbol{w}||^2$$

$m = 15$



Examples of Bias and Variance

- Consider a picture classification task. An ideal classifier (such as a human) may achieve nearly perfect performance in this task.

- Suppose your algorithm perform as following:
  - Training error = 1%
  - Test error = 11%

- Consider a picture classification task. An ideal classifier (such as a human) may achieve nearly perfect performance in this task.

- Suppose your algorithm perform as following:
  - Training error = 15%
  - Test error = 16%

- Consider a picture classification task. An ideal classifier (such as a human) may achieve nearly perfect performance in this task.

- Suppose your algorithm perform as following:
  - Training error = 15%
  - Test error = 30%

- Consider a picture classification task. An ideal classifier (such as a human) may achieve nearly perfect performance in this task.

- Suppose your algorithm perform as following:
  - Training error = 0.5%
  - Test error = 1%

- Consider a speed recognition task. Suppose that 14% of the audio clips have so much background noise that even a human can not recognize what was said. Hence the ***optimal error rate*** in this case is set as 14%.

- Suppose your algorithm perform as following:
  - Training error = 15%
  - Test error = 30%

- Unavoidable bias = 14%
- Avoidable bias = 1%
- Variance = 15%

- Consider a speed recognition task. Suppose that 14% of the audio clips have so much background noise that even a human can not recognize what was said. Hence the optimal error rate in this case is set as 14%.

- Suppose your algorithm perform as following:
  - Training error = 15%
  - Test error = 16%

Techniques to reduce avoidable bias

- Increase the model size (complexity)
- Reduce or eliminate regularization

Techniques to reduce variance

- Decrease the model size
- Add regularization
- Add more training data
- Feature selection to reduce number/type of input features
- Add early stop during the training process