Statistical Methods for Data Science

Semester 1 2021

DATA7202

Understanding statistical inference

Slava Vaisman

The University of Queensland

r.vaisman@uq.edu.au

1. Understanding statistical inference
2. Learning from data
3. The statistical learning framework
4. Train and Test sets
5. Basic Monte Carlo
6. Classification problems

Understanding statistical inference

In statistics, we make an assumption that there exists a (hidden) *data generation process*, and that we observe (possible) *outcomes* of this process. This is a convenient assumption that also helps to understand the connection between statistics and probability as shown in Figure 1.

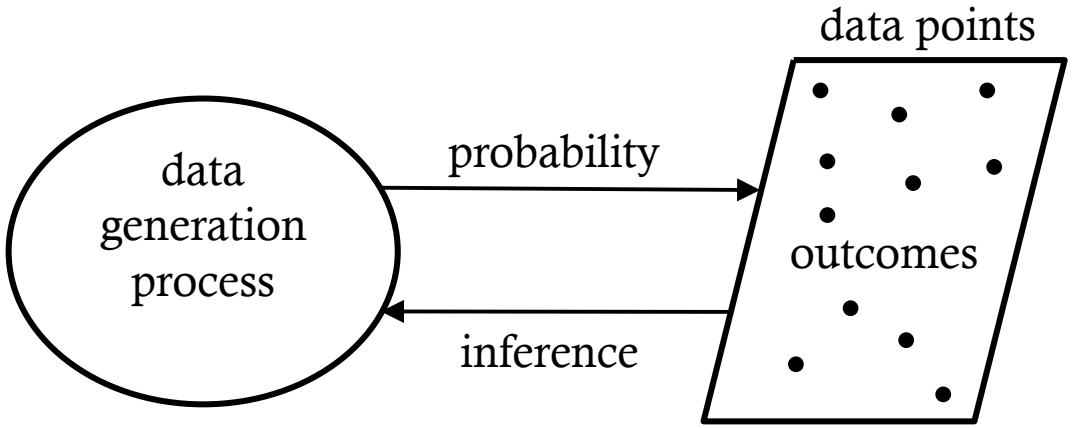


Figure 1: The theory of probability studies the scenario in which the data generation process is known and we are interested in the corresponding probability of an outcome. On the other hand, the inference task is to examine the unknown data generation process given the outcomes.

Statistics vs Machine learning (1)

- Some people believe that there is a difference between *statistics* and *machine learning*.
- It was probably true in the past, but now, it is basically the same discipline.
- If you like, you can call it *statistical inference*. Prediction, classification, clustering, and data mining, are all elements of *statistical inference*.

Let us examine a (very artificial) set of difference between the statistical and the machine learning model. In particular, we will see how the statistical model and the machine learning model look like.

Statistics vs Machine learning (2)

Statistics vs Machine learning

- A statistical model is generally *simple*, yet, capable to account for all important aspects of the problem. It might have a limited predictive accuracy, but, it is possible to use it for interpretations.
- A machine learning model is generally *complex*. It provides high predictive accuracy. However, it is usually hard or even impossible to interpret; that is, we deal with a black box.

In order to fully understand the difference, we continue with an illustrative example. Specifically, we examine a problem of sales maximization. In this example, we will use a synthetic dataset, namely, we will construct our own data generation process mechanism.

Example: Optimal advertisement budget (1)

- We consider the problem of optimal balance allocation in different advertisement domains.
- Specifically, a company can choose to invest in advertisement on the radio, television, and internet.
- The response variable is the number of sales. More formally, let x_1, x_2, x_3 be explanatory variables that correspond to the amount of resources that were allocated to the advertisement on radio, tv, and internet, respectively. As stated above, the response variable y stands for the number of sales.
- A typical portion of data is as follows.

id	x1 radio	x2 tv	x3 internet	y sales
1	0.417022	0.325810	0.576978	5.095785
2	0.720324	0.889827	0.875389	14.378409
3	0.000114	0.751708	0.608565	6.472246
4	0.302333	0.762632	0.251660	4.228842

...

Example: Optimal advertisement budget (2)

- The data was generated via the following data generation mechanism.
- $X_i \sim \mathrm{U}(0,1)$ for $i \in \{1,2,3\}$; here $\mathrm{U}(0,1)$ stands for the continuous uniform distribution over the $[0,1]$ set. In addition, the model for y is as follow:

$$Y = 0.5X_1 + 3X_2 + 5X_3 + 5X_2X_3 + 2X_1X_2X_3 + W, \qquad (1)$$

where $W \sim \mathrm{U}(-2,2)$. In real-life application, we do not know the generation mechanism, although, we might have some idea about it (such knowledge can be used for a better model construction).

The question is, what can we learn from (1)?

Example: Optimal advertisement budget (3)

1. First of all, the mechanism is not linear in x_1, x_2, and x_3 (see the $x_1x_2x_3$ term of

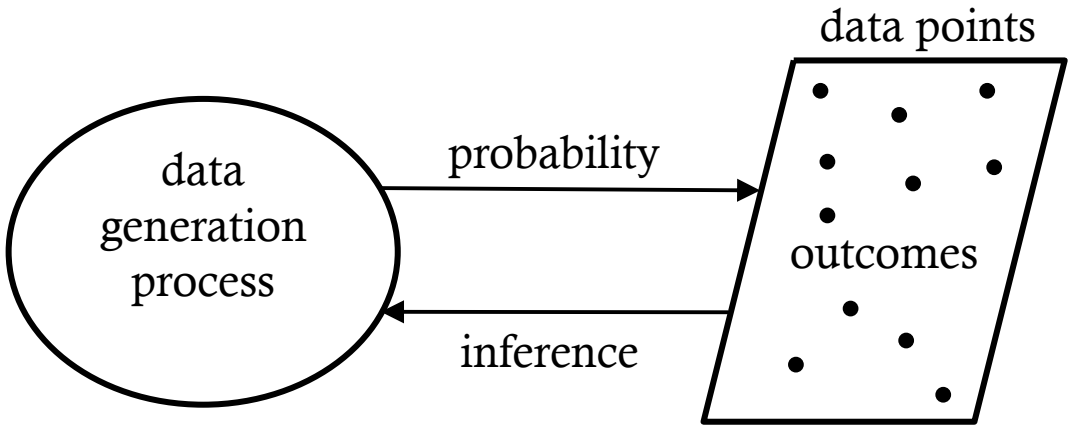
$$Y = 0.5X_1 + 3X_2 + 5X_3 + 5X_2X_3 + 2X_1X_2X_3 + W.$$

2. Intuitively, it seems like the radio advertisement (x_1) contributes less to the overall sales as compared to tv (x_2) and internet (x_3) advertisements.

Please make an important switch and consider a different reality

Please "forget" the equation (1) for now, and consider a predictor (function) $g : \mathbb{R}^3 \to \mathbb{R}$ that will approximate the model in (1). Our objective is to create a "good" predictor g based on the observed data. The next question is of course, what is "good" g.

Example: Optimal advertisement budget (4)



That is, now, we consider the direction:
`outcomes` $\Rightarrow$ `data generation process`.

id	x1 radio	x2 tv	x3 internet	y sales
1	0.417022	0.325810	0.576978	5.095785
2	0.720324	0.889827	0.875389	14.378409
3	0.000114	0.751708	0.608565	6.472246
4	0.302333	0.762632	0.251660	4.228842

...

Example: Optimal advertisement budget (5)

- We first start with linear regression, In particular, we fit the model

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$$

  to the data (here $\epsilon \sim \mathrm{N}(0, \sigma^2)$, where $\mathrm{N}(\mu, \sigma^2)$ is a Normal distribution with mean μ and variance of σ^2.
- By fitting the linear regression, we arrive at the following result.

```
==============================================================
            coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------
radio      0.0857      0.131      0.656      0.512      -0.171       0.342
tv         4.7897      0.126     38.128      0.000       4.543       5.036
internet   7.0184      0.133     52.641      0.000       6.757       7.280
==============================================================
```

Example: Optimal advertisement budget (6)

At this stage, please consider the `coef` and the `[0.025 0.975]` columns. These stand for the coefficients and the corresponding confidence intervals, respectively. We can see the following.

1. This is not clear if radio advertisement contributes to sales. In particular, note that the confidence interval is $(-0.171, 0.342)$. In other words, there is no statistically significant evidence that the radio coefficient is not zero.
2. In addition, the tv and the internet seem to contribute directly to sales. The corresponding confidence intervals suggest that these coefficients are positive (and the result is statistically significant).
3. Finally, regression coefficients suggest that the most beneficial domain for advertisement is the internet, the second on is tv, and the last one is radio.

Example: Optimal advertisement budget (7)

Attention!

We would like to emphasize that the linear model

$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$$

does not fully capture the real data generation mechanism in (1)

$$Y = 0.5X_1 + 3X_2 + 5X_3 + 5X_2X_3 + 2X_1X_2X_3 + W.$$

However, the inference that we made about the benefit of different advertisement types is very informative.

Example: Optimal advertisement budget – the prediction task (1)

- We still have to address the *prediction* task.
- Clearly, there should be a measure of predictive quality.
- In this case, we propose the measure the mean squared error of the prediction $\hat{y} = 0.0857x_1 + 4.7897x_2 + 7.0184x_3$, which is define via

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

where N is the number of data points. To ensure that the model can "generalize", we test the linear model on an unobserved data (we will extensively explain these ideas in the future).

The linear model MSE is equal to 1.9487.

Example: Optimal advertisement budget – the prediction task (2)

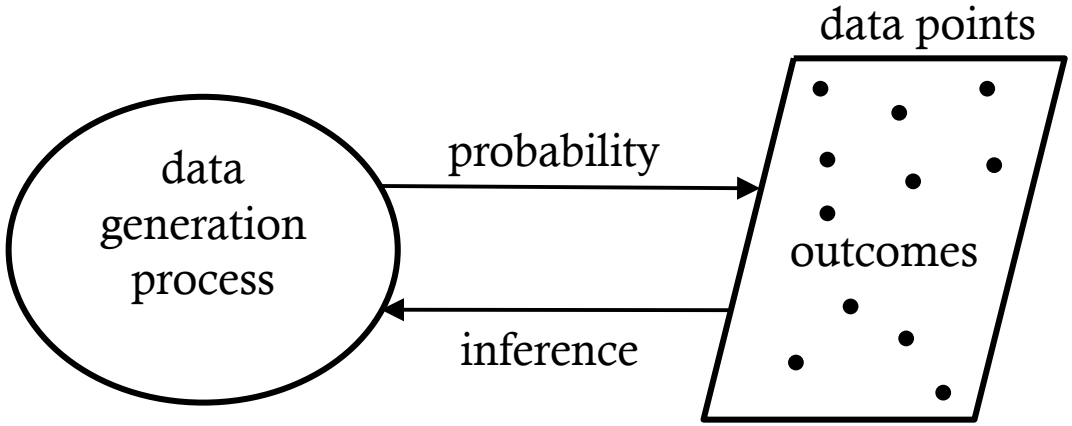
- We next consider the "Machine Learning" approach and train a *Random Forest* (RF) model with 500 trees.
- Do not worry if you are not familiar with random forests — this is a more complex model ($g : \mathbb{R}^3 \to \mathbb{R}$) that contains an ensemble of decision trees.
- This is not easy to interpret the RF model, but, it generally delivers a superior predictive ability. In particular, the following holds.

The RF model MSE is equal to 1.5326 (better than 1.9487 of the linear model).

The RF's MSE is better, but, we lost the interpretation ability.

Generating synthetic datasets (1)

A very important skill is to generate your own datasets.



That is, now, we consider the direction:
`data generation process` $\Rightarrow$ `outcomes`.

Generating synthetic datasets for the advertisement example)

```
def CreateDataset(fName, size, seed):
    np.random.seed(seed)
    N = size
    X1 = np.random.rand(N)
    X2 = np.random.rand(N)
    X3 = np.random.rand(N)

    Y = np.zeros((N,))
    # logic
    for i in range(N):
        if(np.random.rand()<0.5):
            noise = 2*np.random.rand()
        else:
            noise = -2*np.random.rand()
        Y[i] = 0.5*X1[i] + 3*X2[i] + 5*X3[i] + 5*X2[i]*X3[i] +
                           2*X1[i]*X2[i]*X3[i]  + noise
    data = np.array([X1, X2, X3 ,Y]).T
    df = pd.DataFrame(data,columns=['radio','tv','internet','sales'])
    #df.to_csv(fName, index=False)
    return df
```

Linear model and RF for the advertisement example

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn.ensemble import RandomForestRegressor

df_train = CreateDataset("synthetic_sales_train.csv", 1000, 1)
    df_test = CreateDataset("synthetic_sales_test.csv", 1000, 2)
    X_train = df_train[['radio','tv','internet']]
    y_train = df_train[['sales']].values.reshape(-1,)

    # Linear model
    lin_reg = sm.OLS(y_train, X_train).fit()
    print(lin_reg.summary())
    X_test = df_test[['radio','tv','internet']]
    y_test = df_test[['sales']].values.reshape(-1,)
    y_hat = lin_reg.predict(X_test)
    print("regression mean squared error: ",
          np.mean(np.power((y_test- y_hat).values,2)))
    # RF
    reg = RandomForestRegressor(500, random_state=0)
    reg.fit(X_train, y_train)
    y_hat = reg.predict(X_test)
    print("random forest loss: ", np.mean(np.power((y_test- y_hat),2)))
```

Train and Test datasets

- Please note that when we considered the precision of the prediction, we used a new (test) dataset.
- This makes sense since we would be interested to see how well the model generalizes (we will handle this in details later).
- Of course, it is possible to see how well the model makes the prediction on the training dataset. One can expect that the mean squared error should be smaller in this case, since the same data is used for both the training and the prediction.

```
y_hat = lin_reg.predict(X_train)
print("regression (training) mean squared error: ",
                          np.mean(np.power((y_train-y_hat).values,2)))
y_hat = reg.predict(X_train)
print("random forest loss: ", np.mean(np.power((y_train- y_hat),2)))
```

In this case the linear model MSE is $1.94038 < 1.9487$ and the RF MSE is $0.21955 < 1.5326$.

Learning

- When we say *learning*, we may think about the process of transforming a certain experience into knowledge.
- A learning algorithm will take a training data as an input, and output a certain expertise (knowledge). The output will be usually be a computer program.
- The described transformation of experience to expertise is summarized in Figure 2.

Learning — The general process flow of learning

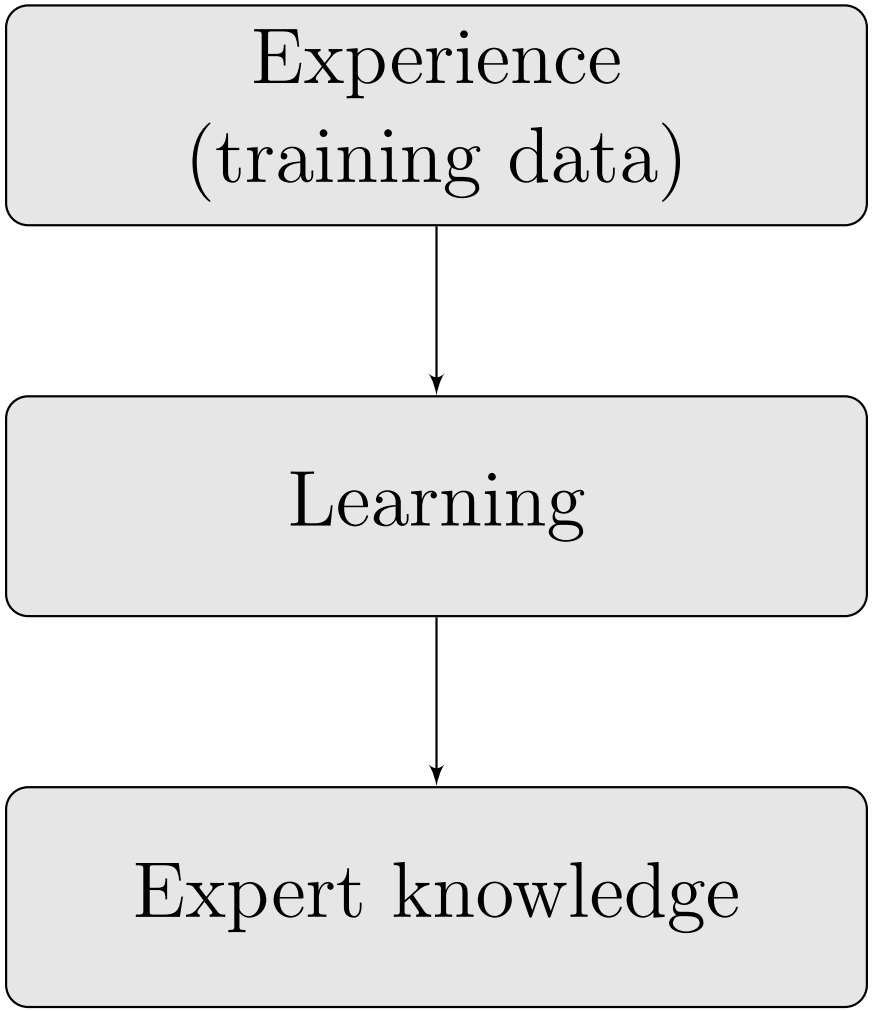


Figure 2: The general process flow of learning.

Why Statistical/Machine Learning? (1)

We need the Machine Learning (ML) capabilities in the following scenarios.

The problem can be solved by a human, but the program is very complex. Namely, there is no simple algorithm that can solve the problem. Examples of simple problems include multiplication of two numbers, sorting an array, or finding a shortest path in a GPS navigation system. Examples of complex problems are speech and image recognition, and driving a vehicle.

The problem cannot be solved by a human. For example, the data that should be processed to derive a meaningful insight is too large for a human to handle. Specifically, and especially in today's big-data world, we need to learn to detect meaningful patterns in large and complex data sets. These tasks appear in commerce, search engines, etc..

Why Statistical/Machine Learning? (2)

We need the Machine Learning (ML) capabilities in the following scenarios.

Program's adaptability. A normal computer program does not change its behavior over time. However, a learning algorithm should be able to adapt to a new input. For example, a voice or hand-writing recognition system in your smart-phone may introduce a better performance as soon as it learns the owner's patterns.

Relation to other fields

Since ML is an interdisciplinary field, it shares techniques with statistics, information theory, game theory, and optimization. Computer science is an important discipline for ML, since our ultimate goal is to create a computer program that is able to learn.

Types of Learning (Supervised/Unsupervised)

- In the Supervised learning, the learner receives tuples of explanatory and response variables.
- In the unsupervised setting, there is only an explanatory data (no response), and the learner should find an unusual behavior.
- For example, consider a spam-filter learning algorithm. The training data will consists of emails associated with their labels Spam/Not spam.
- Given a new email, the learner will decide (classify) if it is spam or not.
- On the other hand, given a set of messages without labels, an unsupervised learner will decide if there exist unusual messages.

Supervised learning

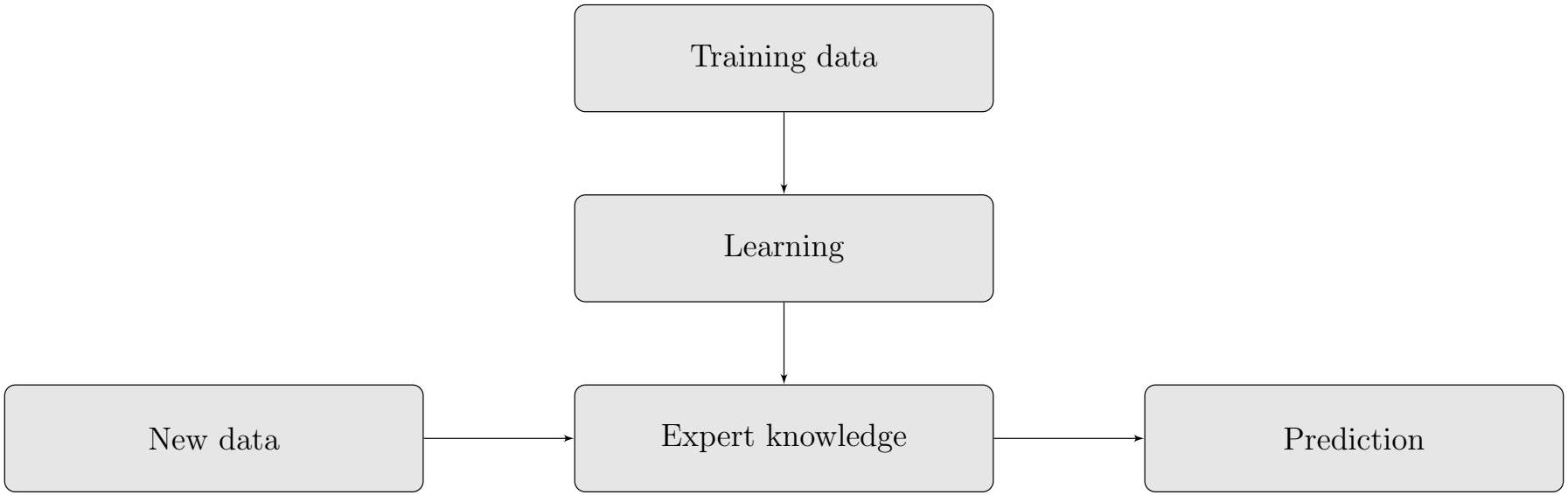


Figure 3: The supervised learning framework.

Unsupervised learning

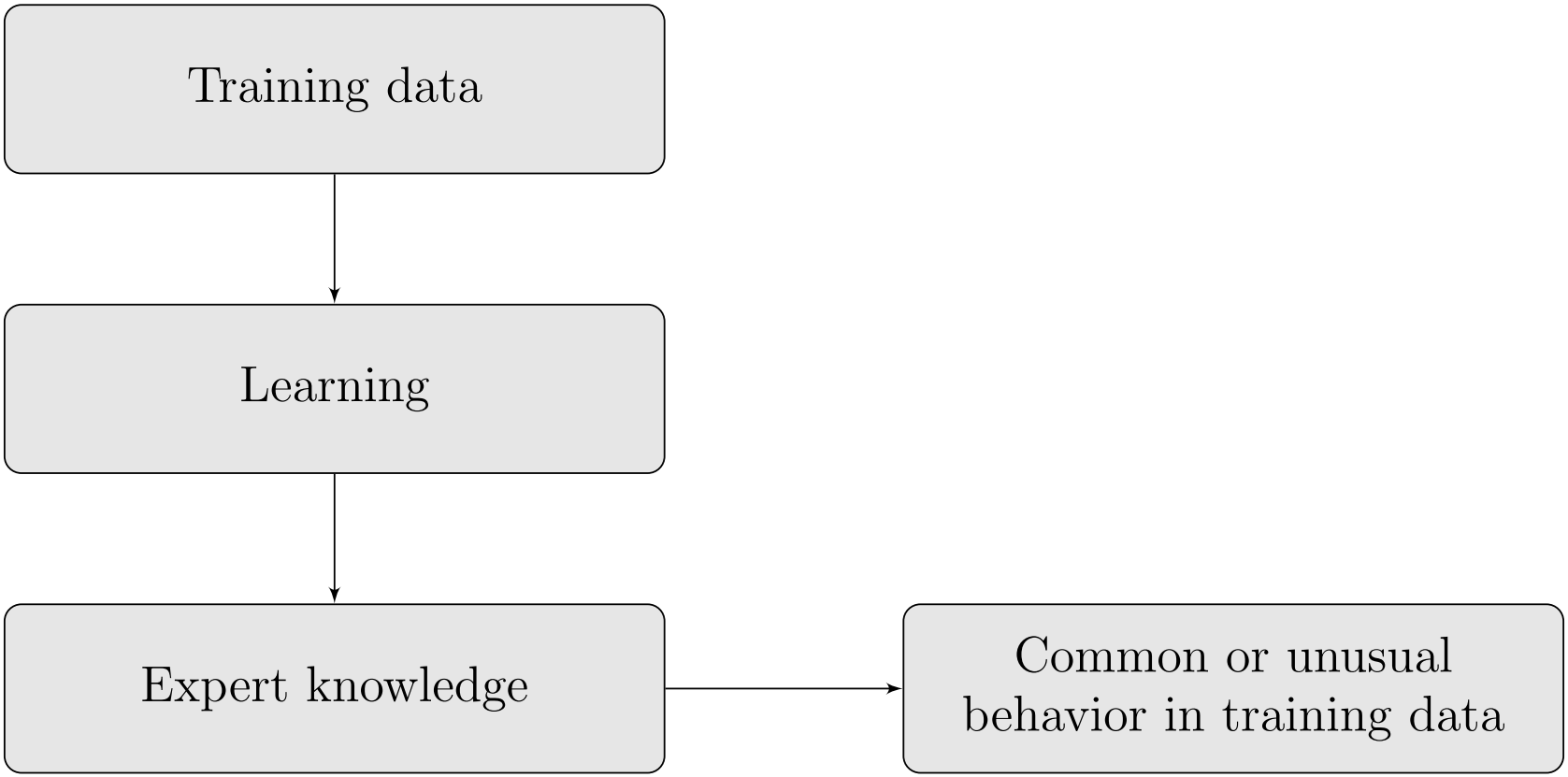


Figure 4: The unsupervised learning framework.

Types of Learning (The reinforcement learning (RL) framework)

- In this setting, a learner is feeded with a training data and learns how to take actions in an environment so as to maximize some notion of cumulative reward.
- For example, consider a chess game learner that needs to learn to play well (at every move). However, it is only given chess games and their corresponding outcomes.
- The RL framework lies in between the supervised and unsupervised frameworks.

Types of Learning (Active and passive learners)

- An active learner will interacts with the environment during the training process.
- Namely, it will introduce queries that might improve the learning. The passive learner is limited to an observation of the training data given by the environment.
- In the spam-filter example, the learner is passive.
- On the other hand, an active learner will introduce (self-generated) emails to the environment and ask it to label them. Such a behavior can improve the learner.

Types of Learning (The offline and the online learning/Different types of environment)

- The learning process can either take advantage of a large block of a training data, or, it required to learn and respond online.
- The former setting corresponds to the offline (or batch) learning, and the latter corresponds to the online learning paradigm.
- The environment (the teacher) can be helpful or even adversarial.
- A helpful teacher will supply the learner with the most helpful information.
- On the other hand, a passive environment such as the star shine, will generally disregard the needs of the learner.

The statistical learning framework — an illustrative example (1)

- We use the following "papaya" example. Specifically, we would like to learn how to predict whether a papaya is tasty or not.
- For simplicity, we will assume that two features of a papaya are of interest, namely, color ranging from 0 to 1, papaya's softness, also ranging from 0 to 1.
- The above features are explanatory variables, and the response variable is a label from the set $\{\mathtt{T}, \mathtt{N}\}$.

This is a supervised learning (classification) problem.

The statistical learning framework — an illustrative example (2)

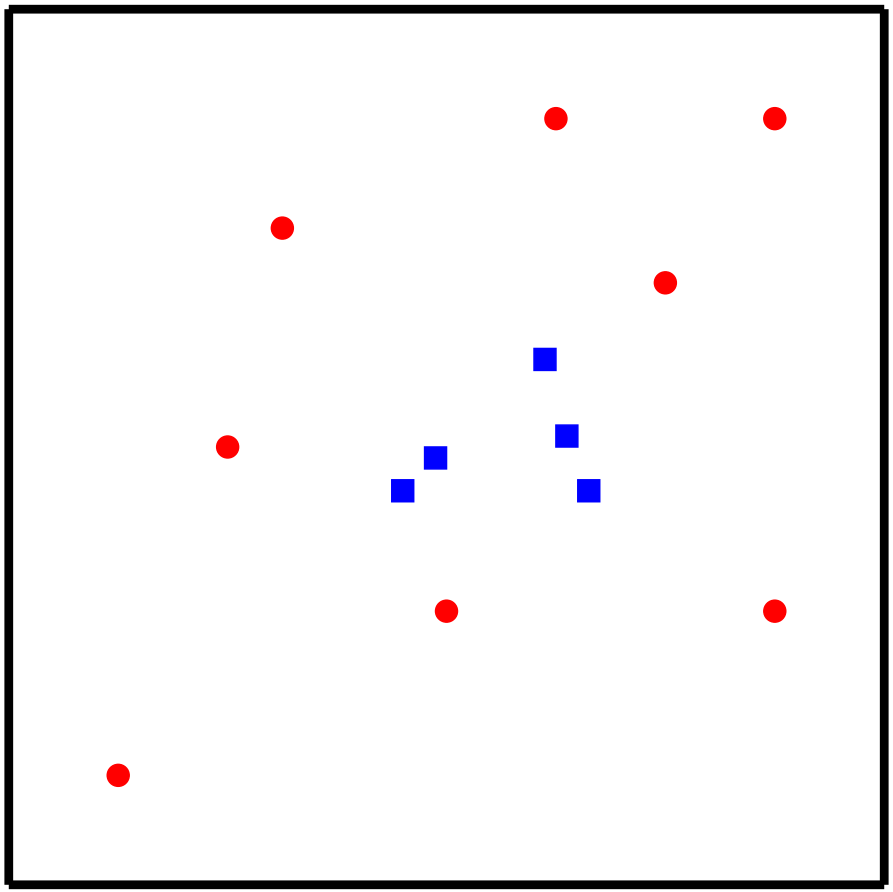


Figure 5: The papaya example. Squares and circles stand for tasty and non-tasty papayas in the training data set.

The statistical learning framework — formal notation (1)

- The **domain set**: Let $\mathcal{X}$ be the *domain set*. Namely, the set of explanatory variables. In the papaya example, $\mathcal{X} = [0,1]^2$.
- The **label set**: Let $\mathcal{Y}$ be the *label set*. Namely, the set of response variables. In the papaya example, $\mathcal{Y} = \{0,1\}$, where 0 and 1 stand for being not-tasty and tasty, respectively.
- The **training set**: Let $\tau = \{(\mathrm{x}_1, y_1), \ldots, (\mathrm{x}_m, y_m)\}$ be a finite sequence of tuples in $\mathcal{X} \times \mathcal{Y}$ be a training set. Here, $\mathrm{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ for $i = 1, \ldots, m$. In the papaya example, $\mathrm{x}_i = (x_{i,1}, x_{i,2}) \in [0,1]^2$ is a two-dimensional vector, and $y_i \in \{0,1\}$ for $i = 1, \ldots, m$.
- The **predictor**: The predictor (sometimes called hypothesis, or a classifier) is a function $g : \mathcal{X} \to \mathcal{Y}$. This function can be used to predict the label of new domain points. In the papayas example, it is a function (rule) that the learner will use to predict whether a new papaya he examines are going to be tasty or not. A predictor that was trained using the training set τ is denoted by g_τ, where $g_\tau : \mathcal{X} \to \mathcal{Y}$.

The statistical learning framework — formal notation (2)

- The **learner**: Given a training set τ, a learner is an algorithm ($\mathcal{A}$) that takes τ as an input and outputs a predictor g. That is,

$$\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^m \to \mathcal{H},$$

  where $\mathcal{H}$ is a predefined class of possible predictors (defining this class is not easy, and we will discuss in details), but for now, please think about $\mathcal{H}$ as say class of linear functions.
- The **training data generation distribution**: Let us assume that the training data is generate by some probability distribution over the $\mathcal{X}$ set, say $\mathcal{D}$. Recall that a probability distribution is a function that describes the likelihood of obtaining the possible values that a random variable can assume. In the papayas example, $\mathcal{D}$ can be a uniform distribution over the $[0, 1]^2$ set.

The statistical learning framework — formal notation (3)

- **The measure of success**: Let us define the error of a predictor as the probability that it does not predict the correct label on a random data point generated by the distribution $\mathcal{D}$. Here, we use the definition of Loss of the classifier, namely,

$$\text{Loss}_{\mathcal{D}}(g) \triangleq \mathbb{P}_{\mathsf{X}\sim\mathcal{D}}\left(g(\mathsf{X}) \neq g^*(\mathsf{X})\right) = \mathbb{E}_{\mathcal{D}}\mathbb{1}\{g(\mathsf{X}) \neq g^*(\mathsf{X})\},$$

  where g^* is the true labeling function (please note that we introduced a very strong assumption, namely, we assume that g^* exists).

Information available to the predictor: Note that the predictor does not "know" g^* and $\mathcal{D}$. That is, the predictor can only interact with the environment via the training set τ.

Empirical Risk Minimization

Since the predictor can only interact with the environment via the training set τ, it is useful to calculate the training loss:

$$\mathrm{Loss}_\tau(g) \triangleq \frac{\sum_{i=1}^{m} \mathbb{1}\{g(\mathsf{x}_i) \neq y_i\}}{m}$$

The training set is a sample from distribution $\mathcal{D}$, and it is the only thing that is available to the predictor, and therefore, it makes sense to search for a solution that works well on that data.

Definition

Finding a predictor g that minimizes $\mathrm{Loss}_\tau(g)$ is called Empirical Risk Minimization (ERM). Note that ERM is an algorithm, that is, it is a *learner*. That is:

$$\mathrm{ERM}_{\mathcal{H}}(\tau) = \operatorname*{argmin}_{g \in \mathcal{H}} \mathrm{Loss}_\tau(g).$$

Empirical Risk Minimization and Overfitting (1)

Consider the following predictor:

$$g'_{\mathcal{H}}(\mathrm{x}) = \begin{cases} y_i & \text{if there exists } (\mathrm{x}_i, y_i) \in \tau \text{ such that } \mathrm{x} = \mathrm{x}_i, \\ 1 & \text{otherwise.} \end{cases}$$

Regardless of the training set τ, $\mathrm{Loss}_\tau(g') = 0$, so this predictor may be chosen by an ERM algorithm. Note that no other predictor can have a smaller loss.

Essentially, we found a predictor with excellent performance on the training set!!!

Empirical Risk Minimization and Overfitting (2)

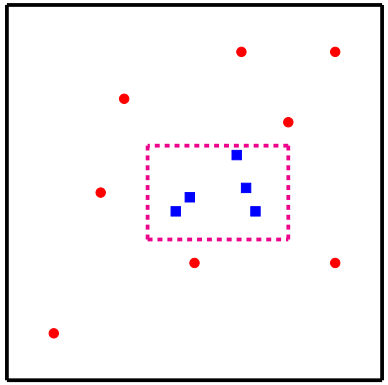


Figure 6: The papaya example. Squares and circles stand for tasty and non-tasty papayas in the training data set. Suppose that $\mathcal{D}$ is uniform distribution, and suppose that the area of the outer and the inner squares is 3 and 1, respectively.

It is not very hard to verify that despite that the empirical loss is zero, the true loss $\text{Loss}_{\mathcal{D}}(g'_\tau)$ is equal to $2/3$.

Essentially, we found a predictor with excellent performance on the training set, and with very bad performance on the true world. This common phenomenon is called *overfitting*.

How to resolve this problem? Learning in nature (1)

- Learning from examples is a strategy that is employed by many biological species.
- For example, when encountering a new type of food, a rat will generally eat a small amount.
- After a couple of hours, the rat will verify its condition in order to form an opinion about the new meal.
- Namely, if the rat feels well, it will continue with the food consumption and otherwise, it will retain from this particular food.
- This is the main reason to the hardness of poisoning rats.

How to resolve this problem? Learning in nature (2)

- The rat's behavior is an example of a successful learning.
- However, there exist bad examples too. Some scientists considered the following experiment.
- A number of pigeons were put into cage, and were delivered food via a random mechanism.
- After a while, pigeons started to associate their (random) behavior with the food delivery.
- This was further reinforced after a few iterations, since they tend to repeat the same activity that occurred during the first food distribution.

This reminds our empirical risk minimization discussion!

How to resolve this problem? Learning in nature (3)

- The rat experiment was revisited by other scientific group, in which the authors replaced the poison by unpleasant experiences such as electrical shock or noise. Surprisingly, the rats continued to consume the food. That is, it seems like the rats have a strong *prior knowledge* about the causal relationship of food and pain.
- Specifically, they "know" that there is no relationship between the food quality and electrical shock. That is, the electric shock did not cause a stomach pain!

To conclude, a difference between successful and unsuccessful learning may be the existence of prior knowledge mechanism that biases the learning process. This is also called an *inductive bias* (or *learning bias*), which is the set of assumptions of the learning algorithm, that is used to predict outputs given inputs that it has not encountered. In contrary, pigeons were willing to adopt any explanation for the occurrence of food.

Empirical Risk Minimization using Inductive Bias (1)

- To cope with the overfitting problem, we will perform learning over a restricted search space.
- Specifically, we should choose in advance (before seeing the data) a set of predictors. This set is denoted by $\mathcal{H}$ and is called a hypothesis class. Each $g \in \mathcal{H}$ is a function that maps $\mathcal{X}$ to $\mathcal{Y}$.
- Now, given g and a training set τ, we will use the ERM rule to choose a predictor $g \in \mathcal{H}$ with minimal $\mathrm{Loss}_\tau(g)$, namely:

$$\mathrm{ERM}_{\mathcal{H}}(\tau) = \operatorname*{argmin}_{h \in \mathcal{H}} \mathrm{Loss}_\tau(g).$$

- We restricted the learner to choose a predictor from $\mathcal{H}$, and thus we bias it.

Empirical Risk Minimization using Inductive Bias (2)

- This restriction is called an *inductive bias*. Since we choose the class before the learner saw the training data, it should ideally be based on some prior knowledge about the problem in hand.
- In the papaya example, we can choose the class $\mathcal{H}$ to be the set of predictors that are determined by axis aligned rectangles in $[0,1]^2$. The choice of this class is motivated by a prior knowledge that tasty papayas have similar colors and softness.

A more restrictive class will protect us from overfitting, however, it may cause a stronger inductive bias. This is a fundamental trade-of in the learning theory.

Some examples of Hypothesis Classes $\mathcal{H}$

You are already familiar with some Hypothesis Classes.

- $\mathcal{H} = \{\text{all linear predictors}\}$, that is,

$$\mathcal{H} = \{\mathrm{x}^\top \beta;\ \beta \in \mathbb{R}^m\},$$

  where m is the number of explanatory variables.
- $\mathcal{H} = \{\text{all CDFs}\}$.
- $\mathcal{H} = \{\text{all neural networks with 10 neurons}\}$.
- etc.

In general, for parametric models, consider

$$\mathcal{H} = \{f(\mathrm{x}, \theta);\ \theta \in \Theta\},$$

where θ is an unknown vector of parameters that take values in the parameter space Θ.

Example

- Let $X_1, \ldots, X_n$ be outcomes of the independent n coin tosses.
- It is reasonable to model the experiment via the Bernoulli distribution with success parameter p.
- That is,

$$\mathcal{H} = \{\text{Ber(p)};\ \text{p} \in [0,1]\}.$$

- Here $\theta = p$ and $\Theta = [0,1]$.

Fundamental Concepts in Inference (1)

- **Point estimation:** provide the best guess for the parameter θ
- **Confidence Sets:** provide a $1-\alpha$ confidence set for a parameter θ, namely, find a and b, such that

$$\mathbb{P}(\theta \in (a, b)) \geq 1 - \alpha, \quad \text{for all } \theta \in \Theta.$$

- **Hypothesis Testing:** make a hypothesis regarding θ and ask if the provided data is sufficient to reject the (null) hypothesis.

Fundamental Concepts in Inference (2)

Please note that in the advertisement example, we actually handled all three tasks by estimating the coefficients ($\theta = (radio, tv, internet)$) of the regression, providing confidence intervals, and considering the hypothesis ($radio = 0$,$tv = 0$, and $internet = 0$).

	coef	std err	t	P>|t|	[0.025	0.975]
radio	0.0857	0.131	0.656	0.512	-0.171	0.342
tv	4.7897	0.126	38.128	0.000	4.543	5.036
internet	7.0184	0.133	52.641	0.000	6.757	7.280

Various loss functions

- For now, we worked with mean squared error and binary classification, but other loss functions are possible.
- For example, one might consider a malty-label classification problem (say $\mathcal{Y} = \{0, 1, 2, 3, 4\}$), or, when $y \in \mathbb{R}$ (regression), we can work with the following loss functions.
  - $\ell(g, (\mathrm{x}, y)) = (g(\mathrm{x}) - y)^2$ — squared error loss
  - $\ell(g, (\mathrm{x}, y)) = |g(\mathrm{x}) - y|$ — absolute error loss
  - $\ell(g, (\mathrm{x}, y)) = |g(\mathrm{x}) - y|^p$ — L_p loss

Generalized Loss Functions

Definition

Let $\mathcal{H}$ be the set of predictors and let $\mathcal{Z}$ be a domain (for example, $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ for prediction problems). Let ℓ be a function from $\mathcal{H} \times \mathcal{Z}$ to the set of non-negative real numbers: $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$. We call such a function a *loss* function. Now, we define the risk function to be the expected loss of a classifier,

$$\text{Loss}_{\mathcal{D}}(g) = \mathbb{E}_{Z \sim \mathcal{D}} \ell(g, Z),$$

and similarly, we define the *empirical risk* to be the expected loss over a given sample $\tau = (z_1, \ldots, z_m) \in Z^m$, namely,

$$\text{Loss}_{\tau}(g) = \frac{1}{m} \sum_{i=1}^{m} \ell(g, z_i).$$

Generalized Loss Functions — Examples

For example, consider the above notation applied to the zero-one loss and to the square loss.

1. Zero-one loss:
$$\ell(g,(\mathrm{x},y)) \triangleq \begin{cases} 1 & g(\mathrm{x}) \neq y \\ 0 & g(\mathrm{x}) = y. \end{cases}$$
2. Square Loss:
$$\ell(g,(\mathrm{x},y)) \triangleq (g(\mathrm{x}) - y)^2.$$
3. Clustering (representing data by k coordinates): In this case $\mathcal{Z} \triangleq \mathbb{R}^d$, and the set of models class $\mathcal{H}$ is the set of all sets of size k in $\mathbb{R}^d$. Namely, if $g \in \mathcal{H}$, $g = \{\mathrm{c}_1, \ldots, \mathrm{c}_k\}$, where $c_i \in \mathbb{R}^d$ for $i = 1, \ldots, k$. Then,
$$\ell(g,\mathrm{z}) = \ell(\{\mathrm{c}_1, \ldots, \mathrm{c}_k\}, \mathrm{z}) = \min_{1 \leq i \leq k} \|\mathrm{c}_i - \mathrm{z}\|^2.$$

The Bias - variance decomposition of squared error

Recall that

$$\mathbb{E}\left[(Y - g(\mathsf{X}))^2\right] = (\mathrm{Bias}[g(\mathsf{X})])^2 + \mathrm{Var}(g(\mathsf{X})) + \sigma^2,$$

where

$$\mathrm{Bias}[g(\mathsf{X})] = \mathbb{E}\left[g(\mathsf{X}) - Y\right],$$

$$\mathrm{Var}(g(\mathsf{X})) = \mathbb{E}\left[(g(\mathsf{X}) - \mathbb{E}\left[g(\mathsf{X})\right])^2\right],$$

and σ^2 is the variance of the random error term.

- Note that the expectation is taken with respect to training sets (X, Y).
- The σ^2 term is irreducible.

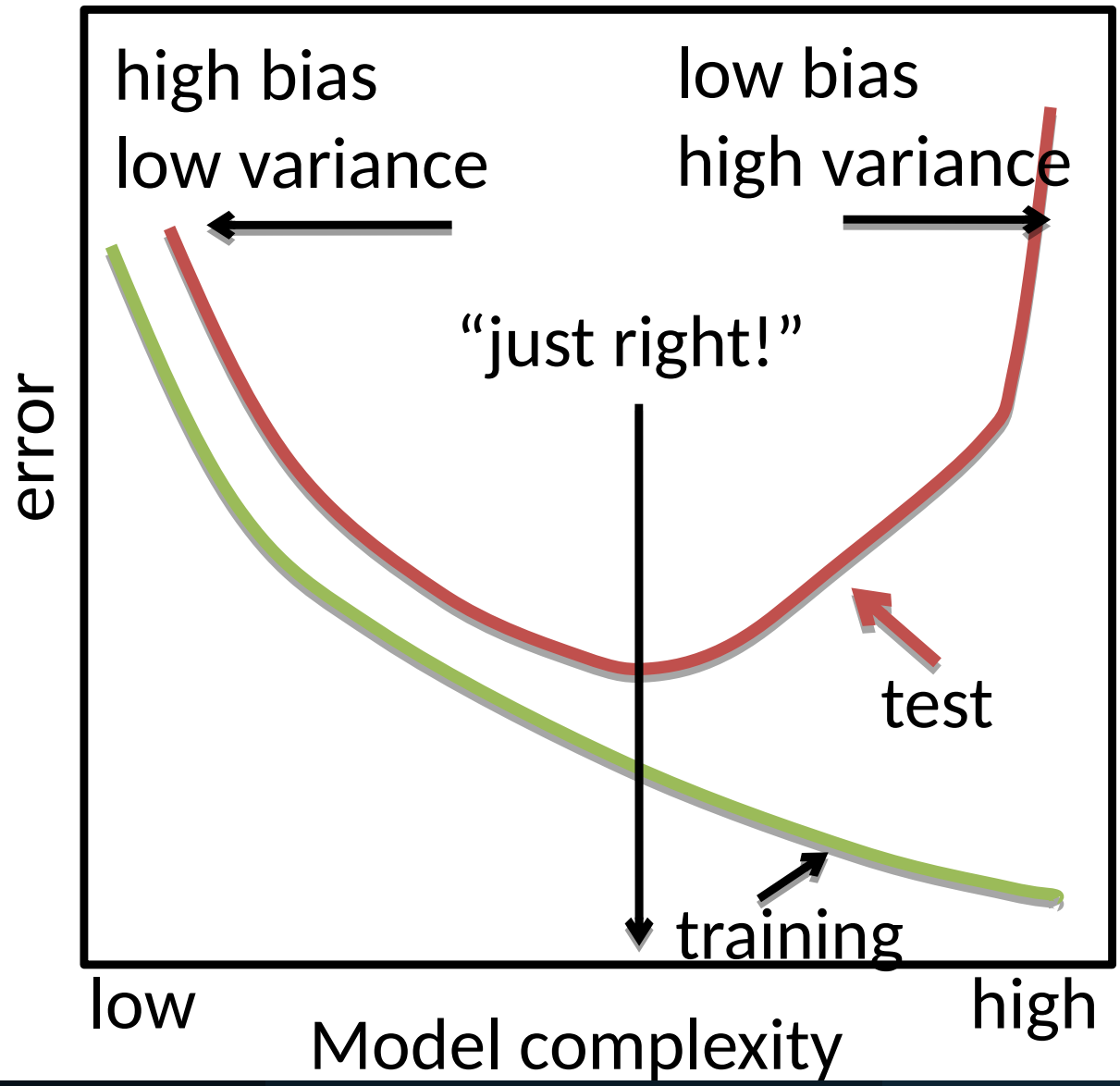
The bias and the variance

- The bias term $\mathrm{Bias}[g(\mathrm{X})] = \mathbb{E}\left[g(\mathrm{X}) - Y\right]$ measures the error caused by the simplifying assumptions built into the method.
- For example, if we model a polynomial of degree 7 with a polynomial of degree 1, we will encounter a high bias.

- The variance term $\mathrm{Var}(g(\mathrm{X})) = \mathbb{E}\left[\left(g(\mathrm{X}) - \mathbb{E}\left[g(\mathrm{X})\right]\right)^2\right]$, tells us about how (much) the model $g(\mathrm{x})$ will change if we will estimate it using a different dataset.
- For example, suppose that we have a dataset of with 10 points. Using a polynomial of degree 10, we can capture all the points. However, for a different dataset, the model will fail to generalize.

The bias and the variance problems are called under and over fitting. respectively. Our main objective is to find a model that will have both low bias and low variance.

A general overview on bias-variance trade-of



The major take home message

A more restrictive class will protect us from overfitting, however, it may cause a stronger inductive bias. This is a fundamental trade-of in the learning theory.

1. Linear regression — "weak model": good protection from overfitting but can have worse predictions.
2. Deep neural network — "powerful model" very powerful predictor but may experience overfitting.

- Give $\mathcal{H}$, the training is simple! Just minimize the empirical loss.
- How do we find a good class $\mathcal{H}$?

Modeling Data

- Assume that $\mathrm{X}_i = (X_1, \ldots, X_p)$. (Note that some of these variables can also be response variables.)
- Let $\mathrm{X}_1, \ldots, \mathrm{X}_n \sim F(\mathrm{x})$ be a data drawn randomly from some **unknown** distribution F.
- Assume that the data is independent and identically distributed (i.i.d), that is,
  1. X_is are independent, and
  2. $\mathrm{X}_i \sim F(\mathrm{x})$ for all $1 \leq i \leq n$.
- Every row x_i in the Table below is independent.

Id	variable 1	variable 2	$\cdots$	variable i	$\cdots$	variable p
1	.	.	.	.	.	.
2	.	.	.	.	.	.
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	.	.	.	.	.	.

Table 1: Data representation by the table

The supervised learning setting

- Suppose that $(X_1, Y_1), \ldots, (X_n, Y_n) \overset{\text{i.i.d}}{\sim} F(x, y)$.
- We would like to find a good prediction function

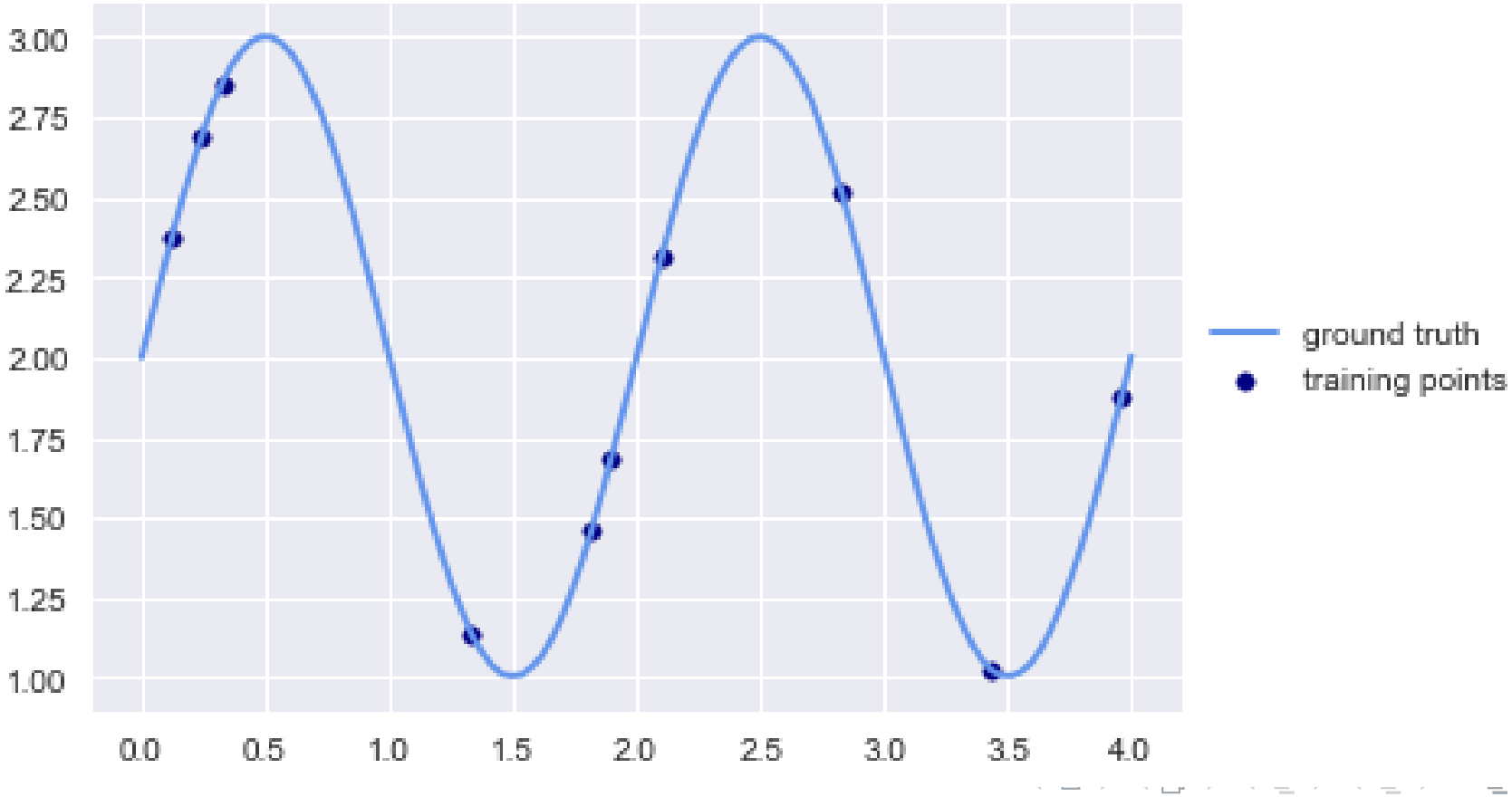
$$g^*(x) \approx y.$$

- Let $g_1(x), \ldots, g_k(x)$ be prediction functions.

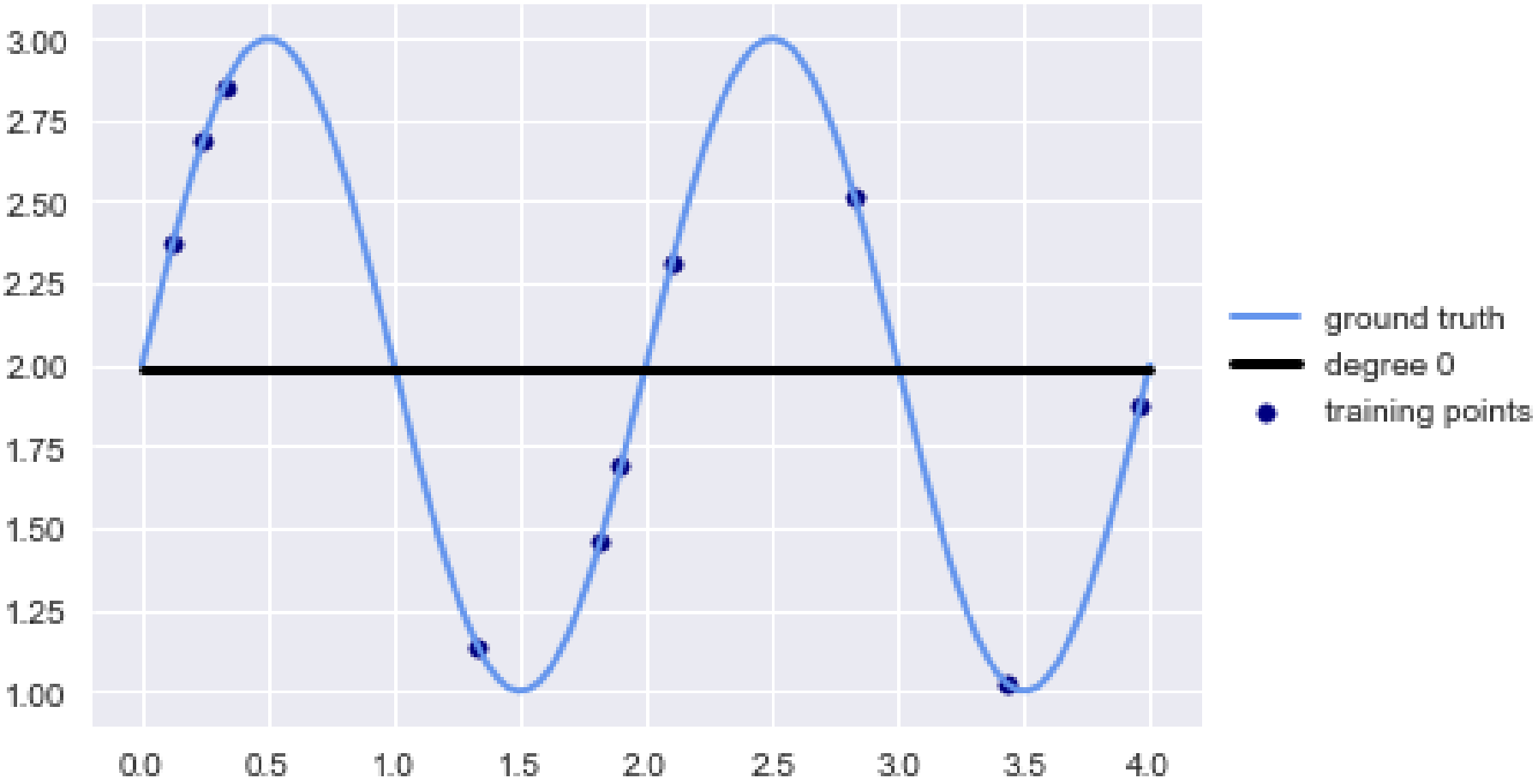
> Find the "best" prediction function $g^*(x)$ in the $\{g_1(x), \ldots, g_k(x)\}$ set.

Example: Polynomial regression

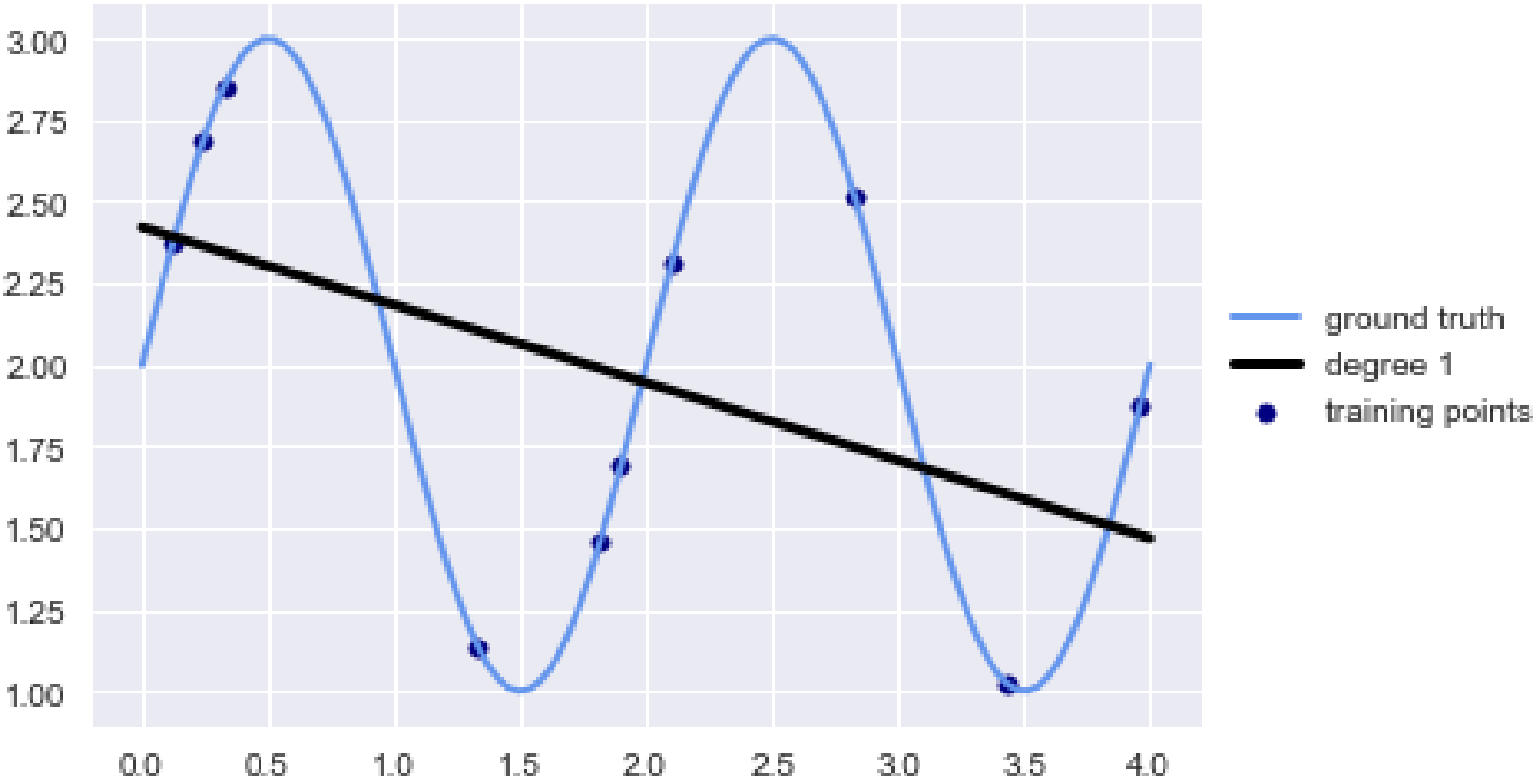
- Consider a function $f(x) = \sin(\pi x) + 2$. We will use 10 training points.
- We will fit 15 models $\{g_i(x)\}_{i=0}^{14}$, where $g_i(x)$ stands for a polynomial of degree i.



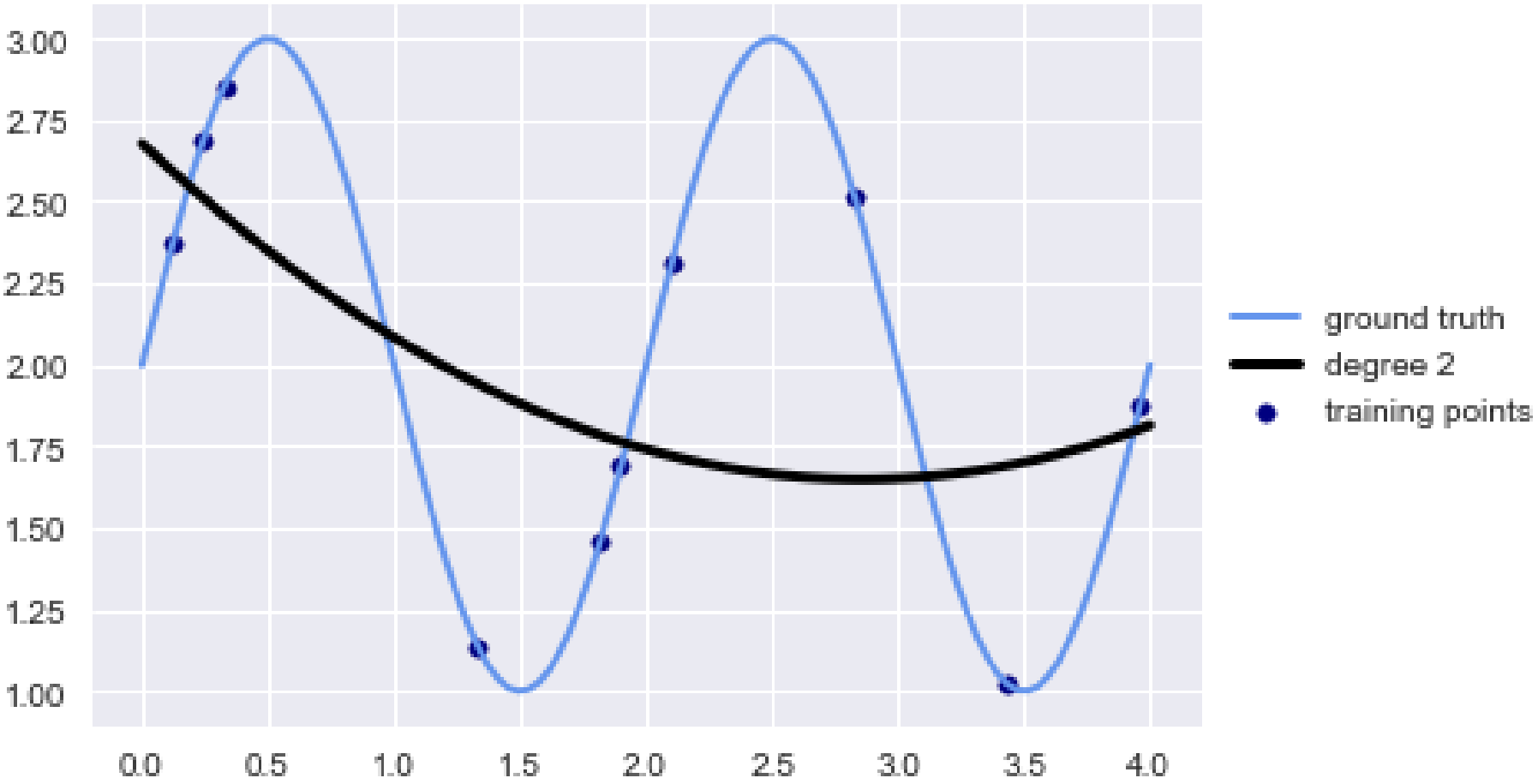
Example: Polynomial regression $g_0(x)$



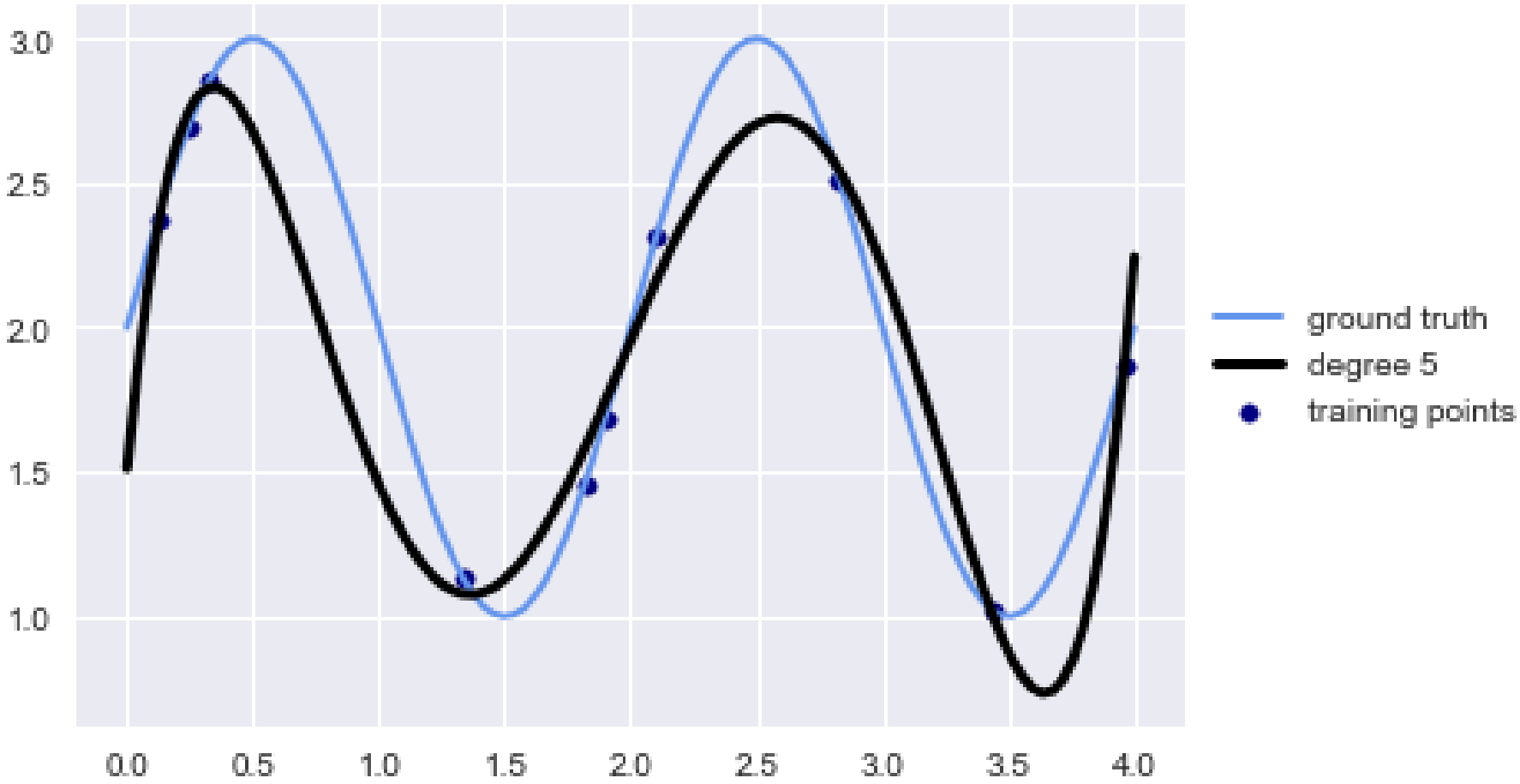
Example: Polynomial regression $g_1(x)$



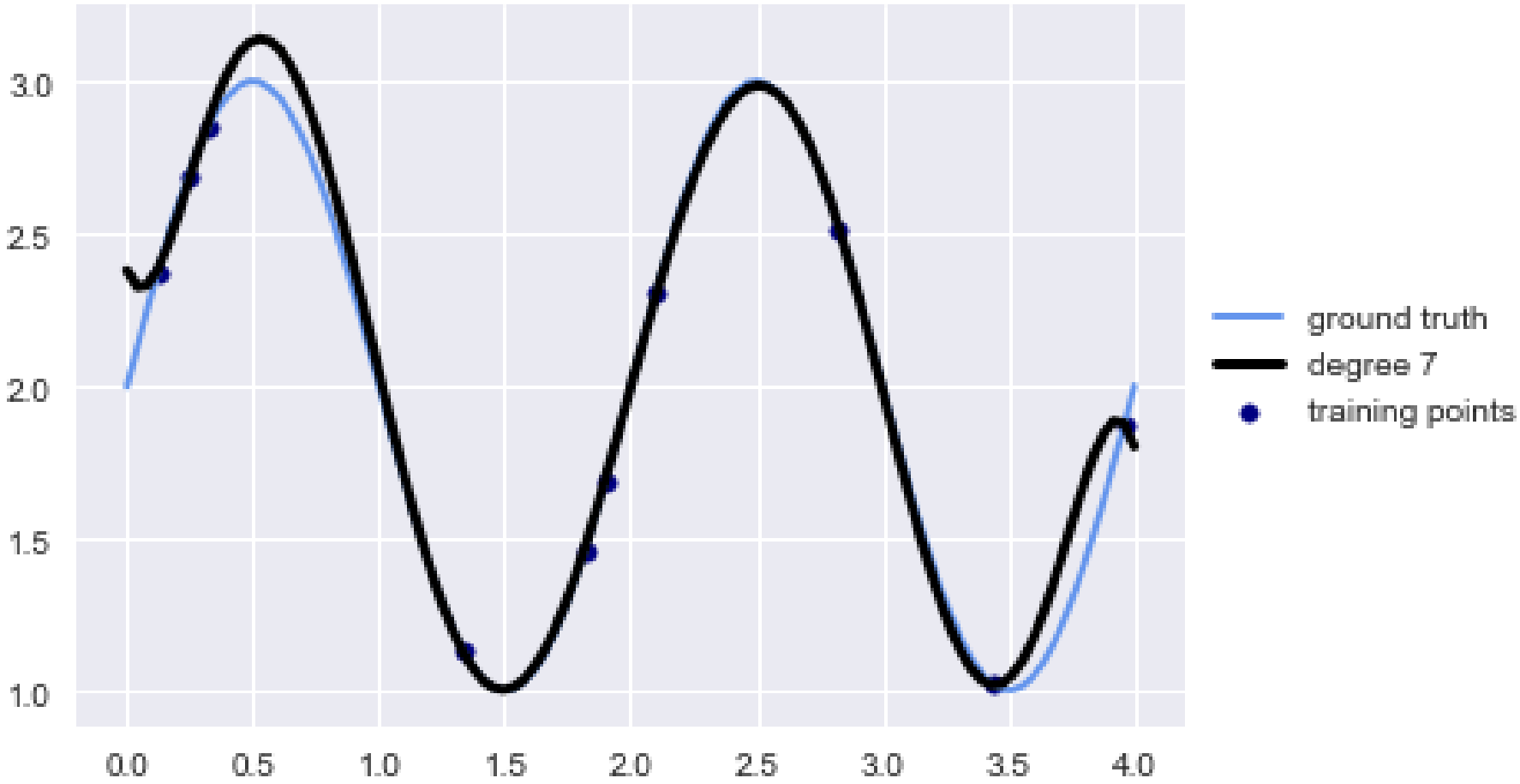
Example: Polynomial regression $g_2(x)$



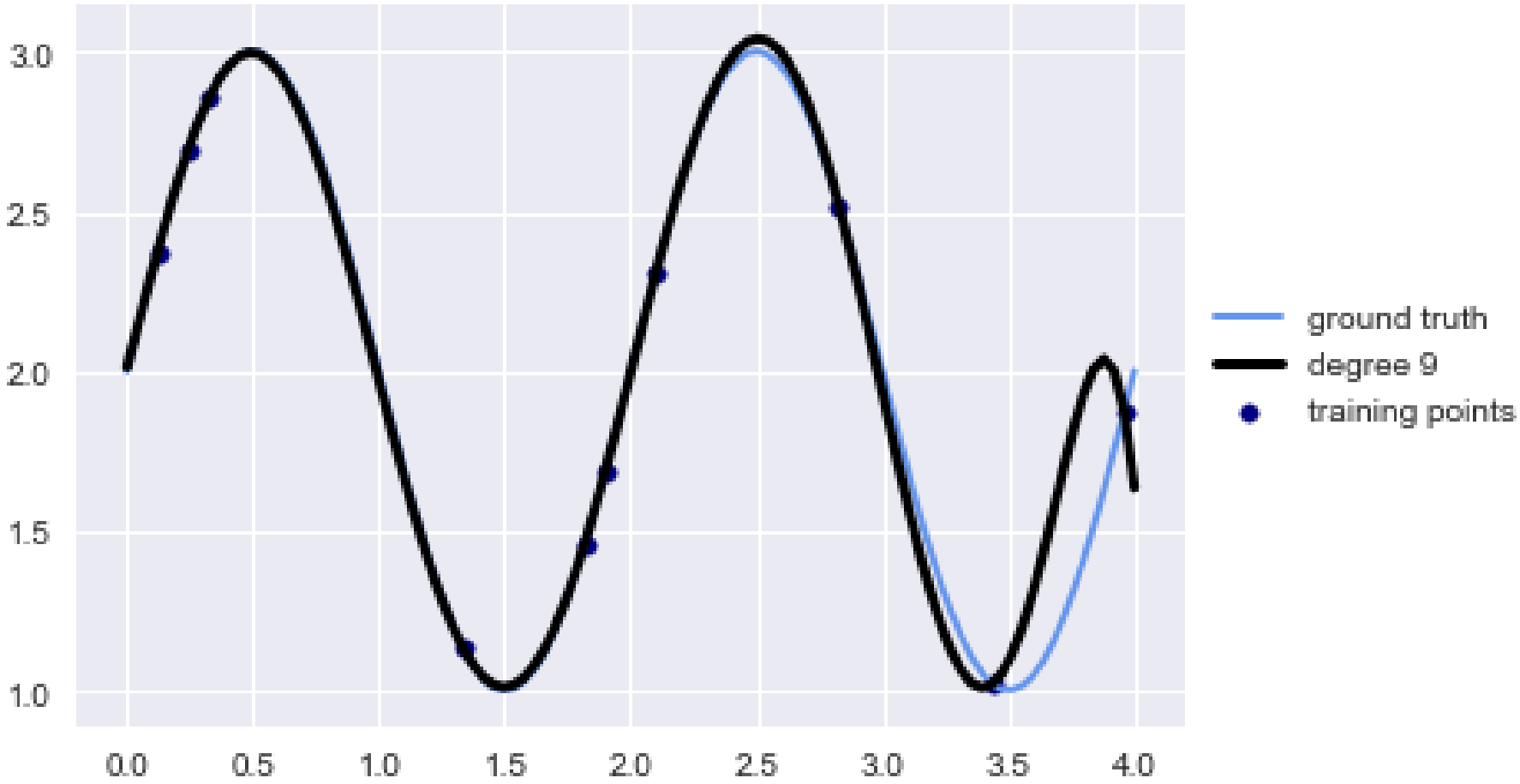
Example: Polynomial regression $g_5(x)$



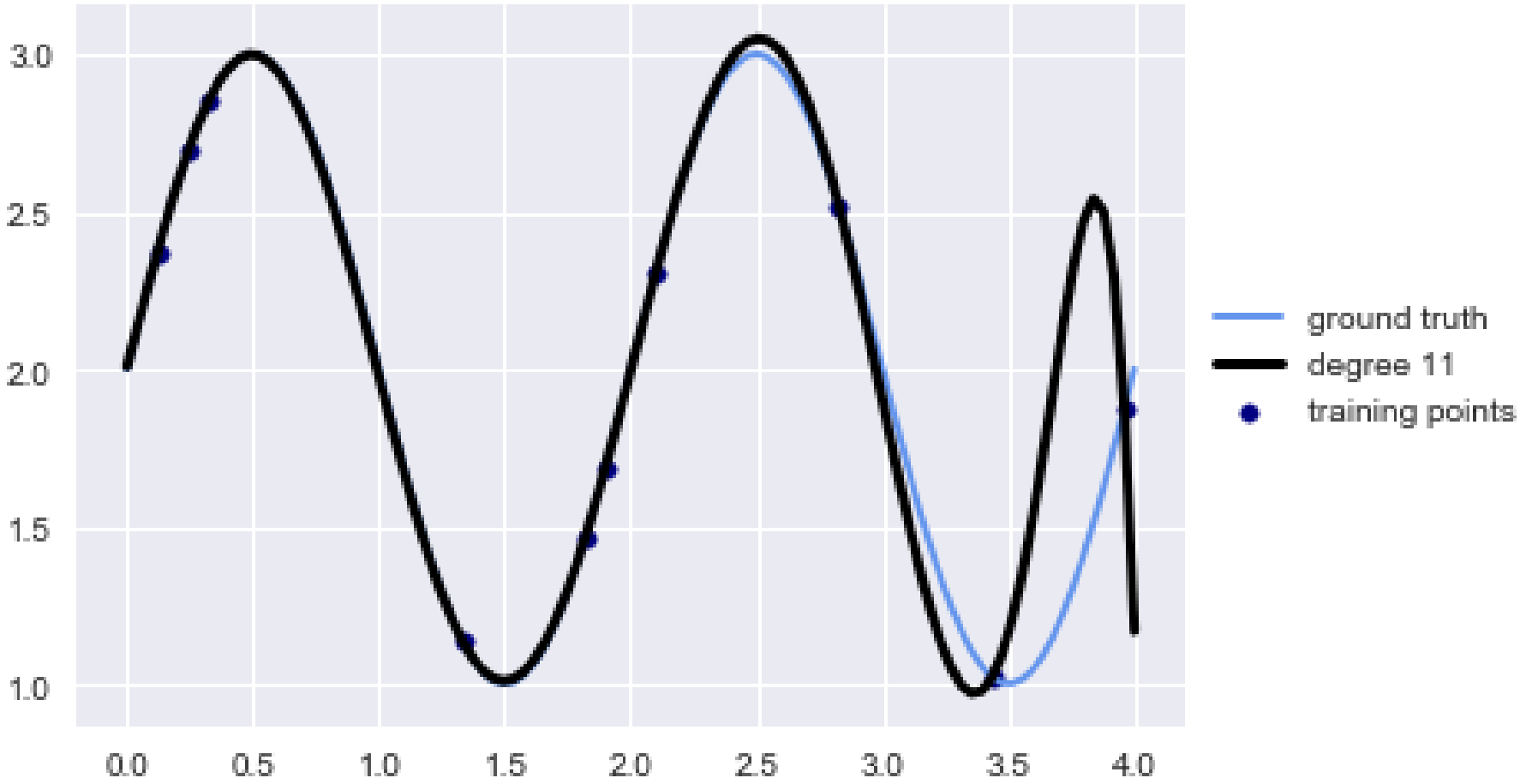
Example: Polynomial regression $g_7(x)$



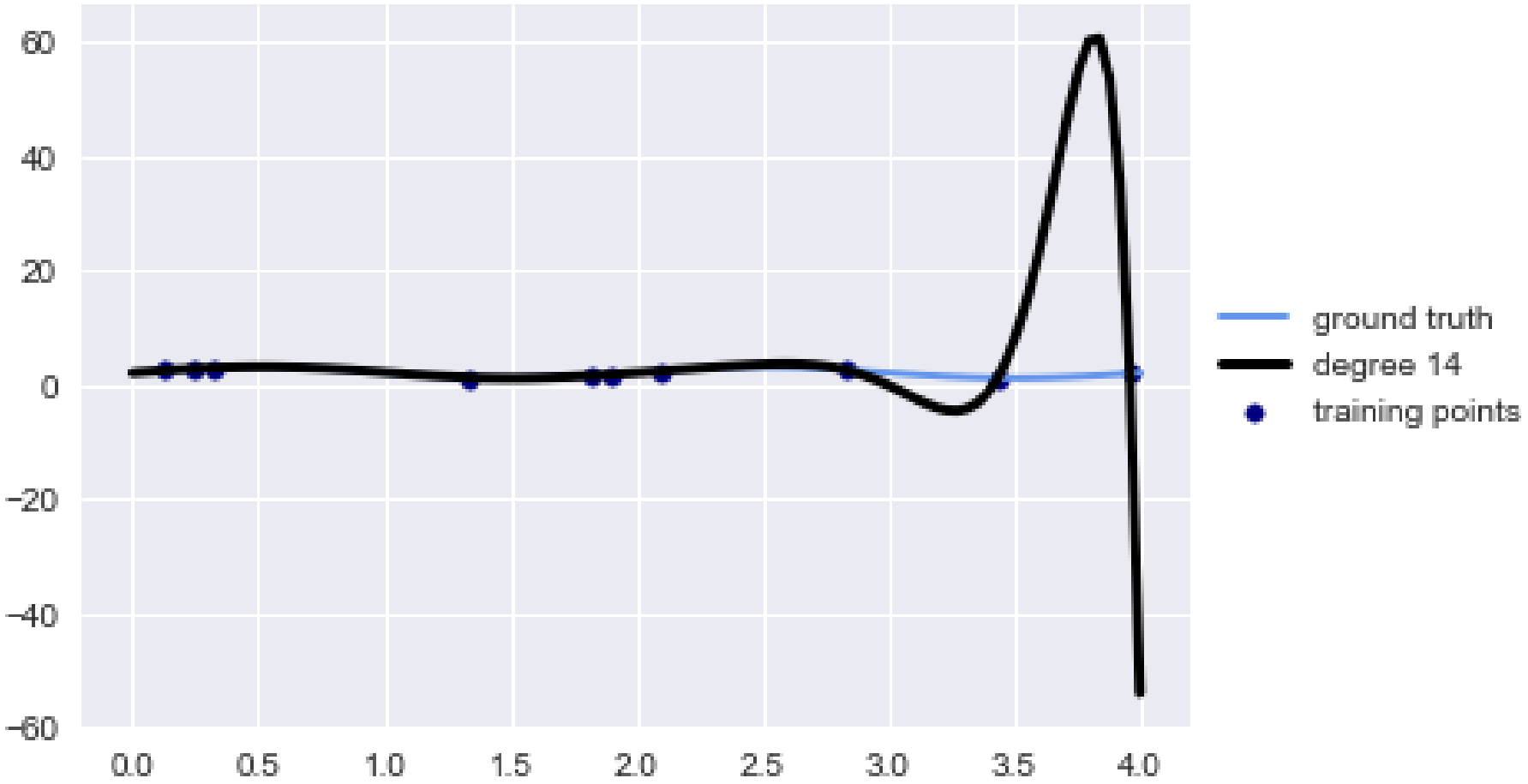
Example: Polynomial regression $g_9(x)$



Example: Polynomial regression $g_{11}(x)$

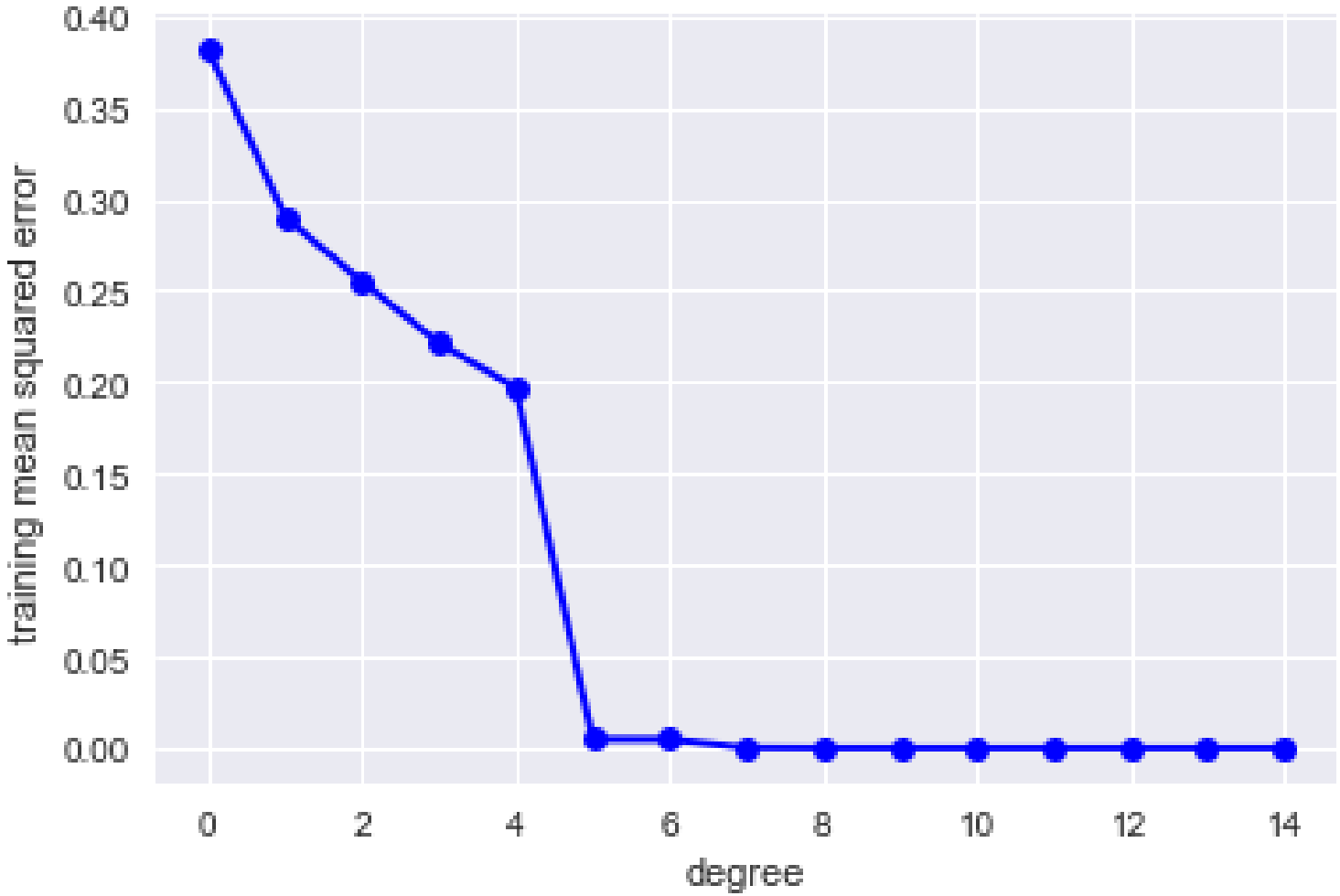


Example: Polynomial regression $g_{14}(x)$



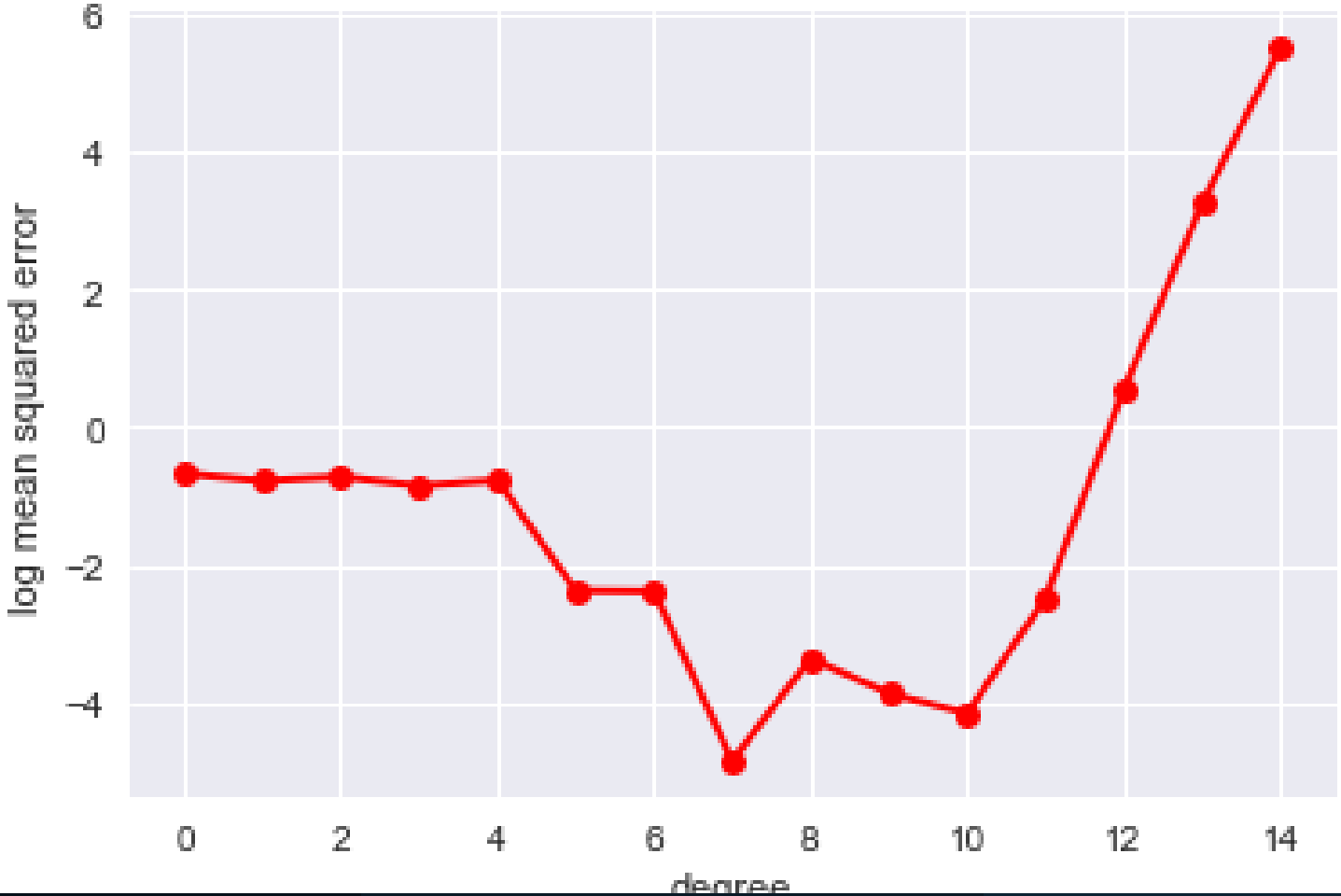
Training mean squared error

Recall that the training was based on 10 points; that is, a 10th degree polynomial should give the perfect match!

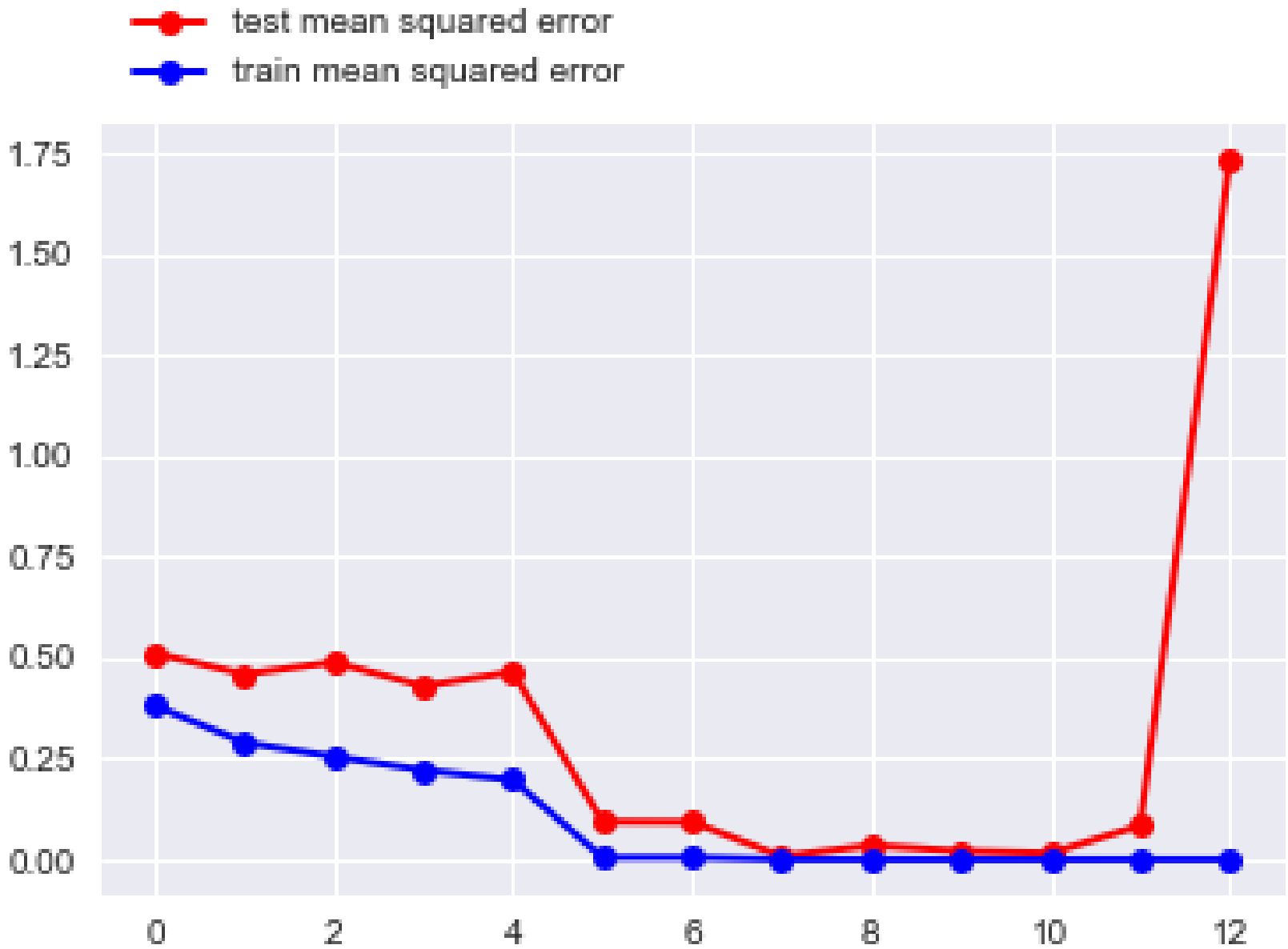


Example: Polynomial regression error

- The training was based on 10 points.
- We generated 90 additional points (from $\mathrm{U}(0, 4)$) and measured the mean square error for each model $\{g_i(x)\}_{i=0}^{14}$.



Example: Polynomial regression train vs. test errors



Example: Polynomial regression train vs. test errors

degree	mse train	mse test
0	0.381592382	0.511513562
1	0.289945476	0.459657388
2	0.254910753	0.489116399
3	0.221286972	0.430023669
4	0.196916864	0.464887448
5	0.004883227	0.093627273
6	0.004835895	0.092812472
7	0.000149973	0.007900633
8	4.69×10^{-8}	0.034999283
9	9.40×10^{-23}	0.021042974

Discussion and Conclusions from the Example

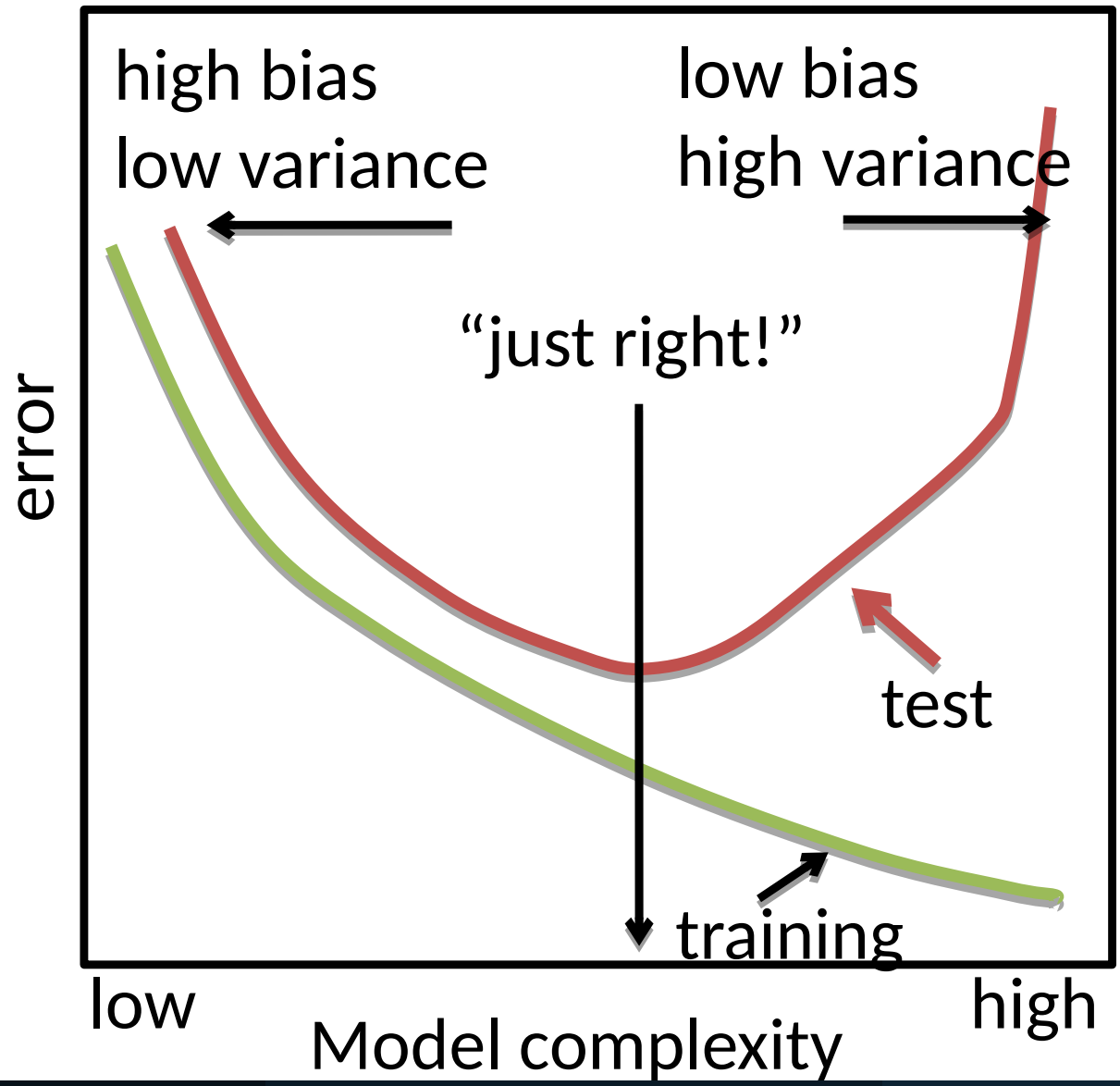
A more complicated model (higher degree polynomial), will not necessarily improve the prediction performance.

Is it reasonable to check the model performance on the unseen data?

Answer: Absolutely!

- Recall the setting. We assumed that $\mathrm{X}_i = (X_1, \ldots, X_p)$ to be a data drawn randomly (i.i.d) from some **unknown** distribution $F(\mathrm{x})$.
- In our case, $F(\mathrm{x}) = F(x, y)$ is the cdf from which we generated our samples.
- So, when we generated additional 90 points, it was from the unknown distribution $F(x, y)$, and, we could check the mean squared error on the unseen data.

A general overview on bias-variance trade-of



Discussion and Conclusions from the Example

- In fact, we could only estimate the model's mean square error.
- Specifically, given a set of new data-points $\{(x_i, y_i)\}_{i=1}^{90}$ we found an estimator

$$\hat{\mathrm{mse}} = \frac{1}{90}\sum_{j=1}^{90}(g_i(x_j) - y_j)^2, \quad i \in \{0, \ldots, 14\}.$$

- This estimator is an approximation of the **true** mse, which is given by

$$\mathbb{E}_F\left[(g_i(x) - y)^2\right], \quad i \in \{0, \ldots, 14\}.$$

But, we cannot generally obtain an additional data, right?

No problem. We can just divide the entire data-set into training and test data. The test data will be used for the estimation of the model's error.

- A typical split will be 60(or 50% and 50%)
- The training set is used to train the model.
- The test set is used to evaluate the model accuracy.

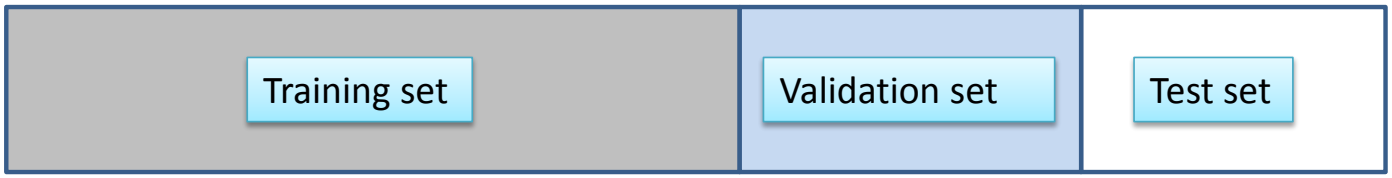
The above approach (Train/Test split), is good when we consider a single model only. It is a bad practice to use it for model selection. Why?

We should still be careful!

- Note that we used a test set to choose the best model.
- In our case, we choose the "best" degree of the polynomial.
- However, note that the test set was used to choose such a degree, and therefore, we might report an over-optimistic generalization error. That is:

> it is bad to choose the test set to evaluate the final model accuracy

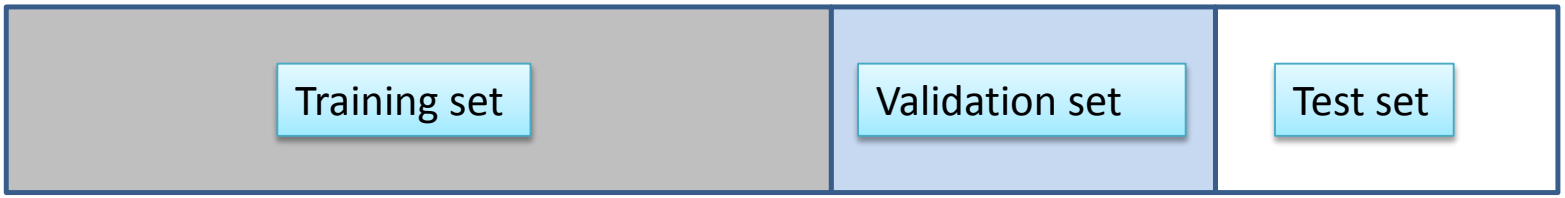
- Solution: the best approach will be to split the data into training, validation, and test sets. A typical split will be 50% for training and 25% for validation and test sets.



Train/Validate/Test

Given the models $\{g_i(x)\}_{i=1}^m$, the procedure of selecting the best model $g^*(x)$ is as follows.

- The training set is used to train the models.
- The validation set is used to select a single "best" model $g^*(x)$ from $\{g_i(x)\}_{i=1}^m$.
- Finally, the test set is used to evaluate the model accuracy of the selected model $g^*(x)$, (the generalization error of the final chosen model).



Cross validation

- The Train/Validate/Test approach works very well in the big data world.
- In case there is a limited amount data, we cannot afford to "waste" some of the data for the testing purposes. We need it for the training.

Suppose that we do not posses a large designated test set. In this case, a very popular technique called cross-validation can be used.

- Suppose we randomly partition a data set $\mathcal{T}$ of size n into K parts $\mathcal{C}_1, \ldots, \mathcal{C}_K$ of sizes $n_1, \ldots, n_K$ (that is, $n_1 + \cdots + n_K = n$).

Cross validation

- Let $\ell_{\mathcal{T}_{-k}}$ be the empirical test loss when using $\mathcal{C}_k$ as test data and all remaining data, denoted $\mathcal{T}_{-k}$, as training data.
- Each $\ell_{\mathcal{T}_{-k}}$ is an unbiased estimator of the generalization loss for training set $\mathcal{T}_{-k}$; that is, for $\ell(g_{\mathcal{T}_{-k}})$.

The K-fold cross-validation loss cross-validation loss is the weighted average of these loss estimators:

$$\begin{aligned} \mathrm{CV}_K &= \sum_{k=1}^{K} \frac{n_k}{n} \ell_{\mathcal{T}_{-k}} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in \mathcal{C}_k} \mathrm{loss}(g_{\mathcal{T}_{-k}}(\mathrm{x}_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^{n} \mathrm{loss}(g_{\mathcal{T}_{-\kappa(i)}}(\mathrm{x}_i), y_i), \end{aligned}$$

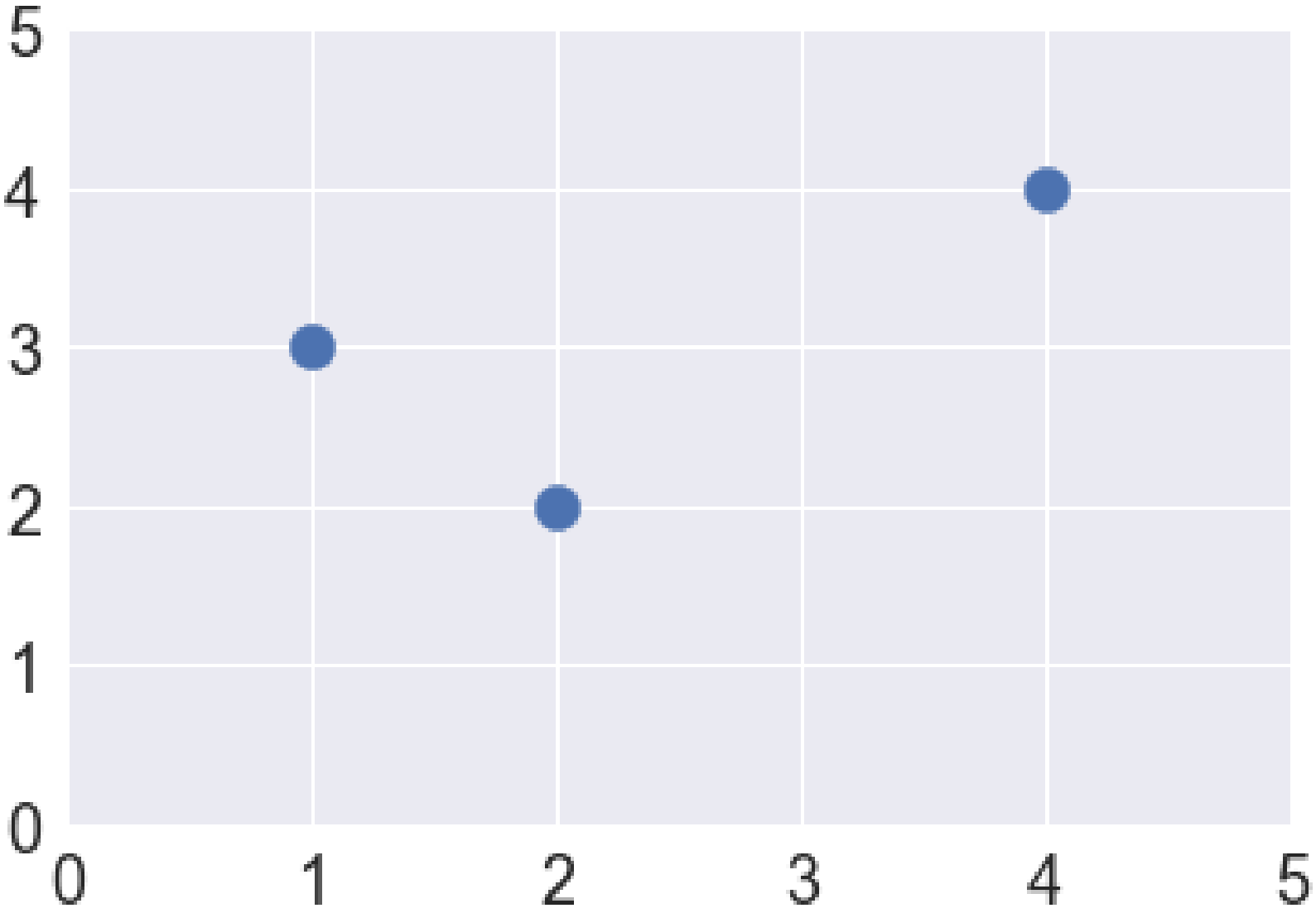
where the function $\kappa : \{1, \ldots, n\} \mapsto \{1, \ldots, K\}$ indicates to which of the K folds each of the n observations belong, and the loss can be defined (for example), as a squared error.

Cross validation

- A common choice is $K = 5$ or $K = 10$, but it generally depends on n. Specifically, since the cross-validation procedure requires the model to be fitted K times, (and this can be expensive), for larger datasets K $= 3$ can be used. If the dataset is small, $K = n$ (leave one out cross-validation) is a popular choice.

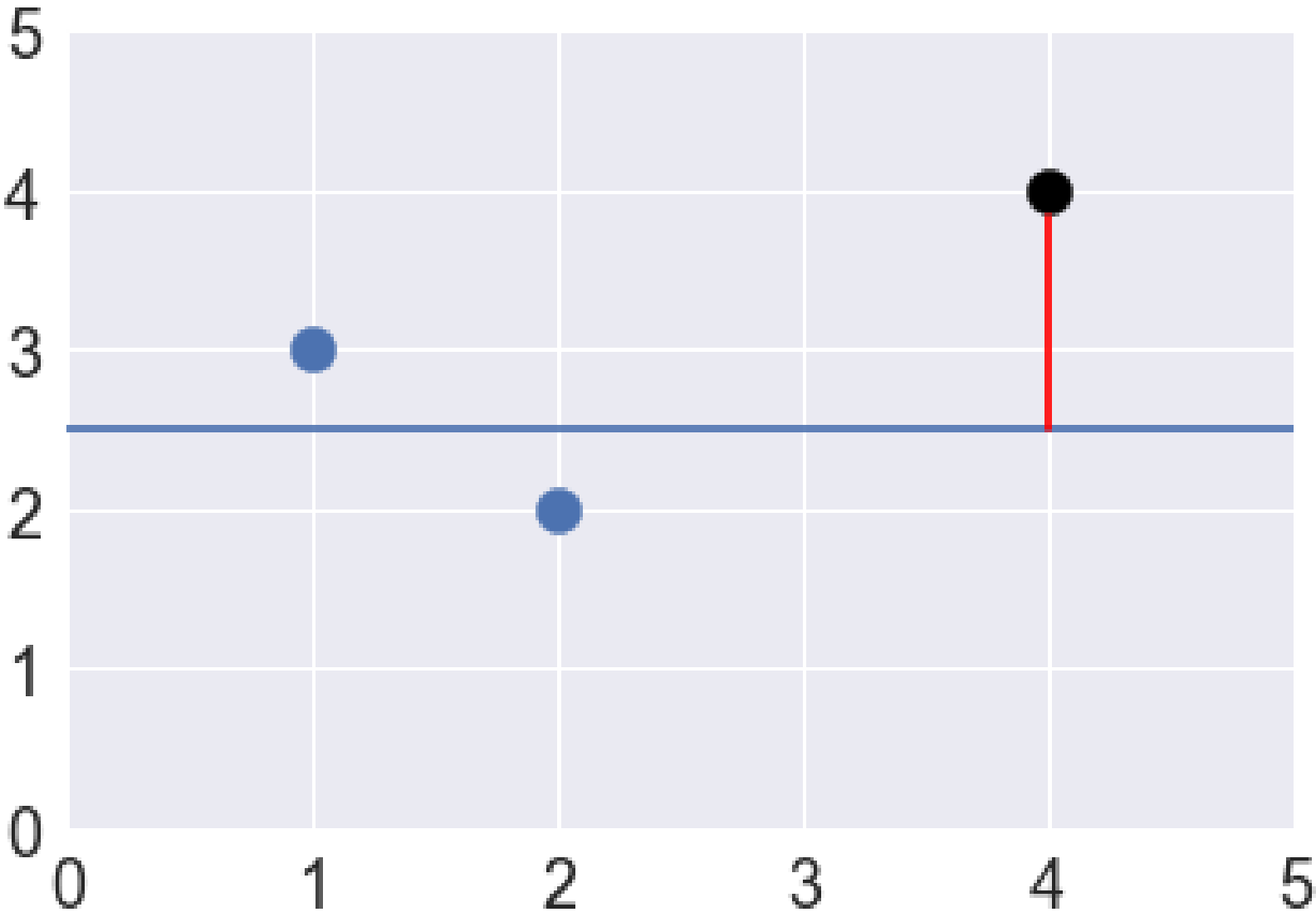
Simple example of leave one out Cross-Validation

Fitting a model $g(x) = \theta$, and estimating the model's error.



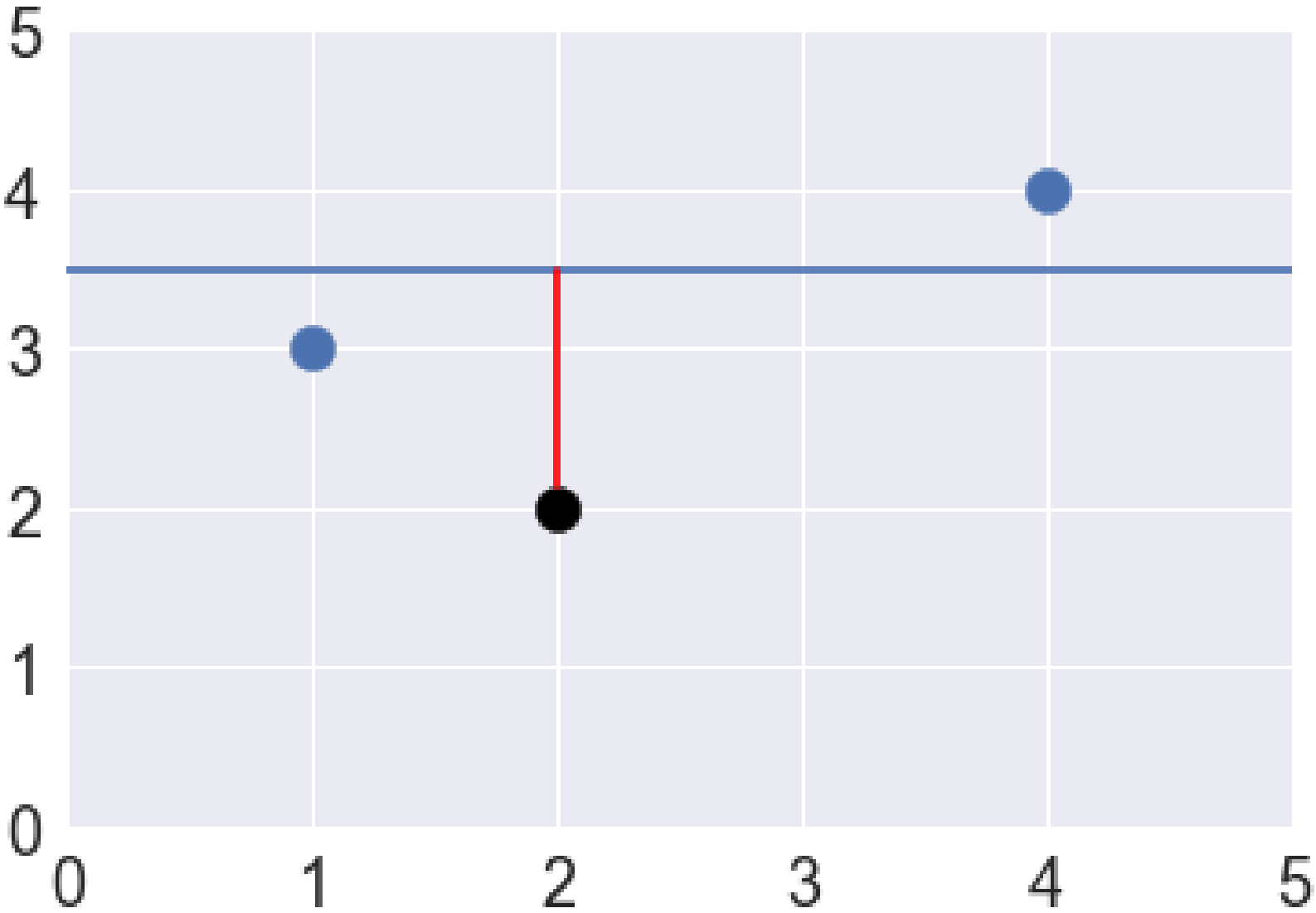
Simple example of leave one out Cross-Validation

The point $(4, 4)$ is in the test set, the loss is $(4 - 2.5)^2 = 1.5^2$.



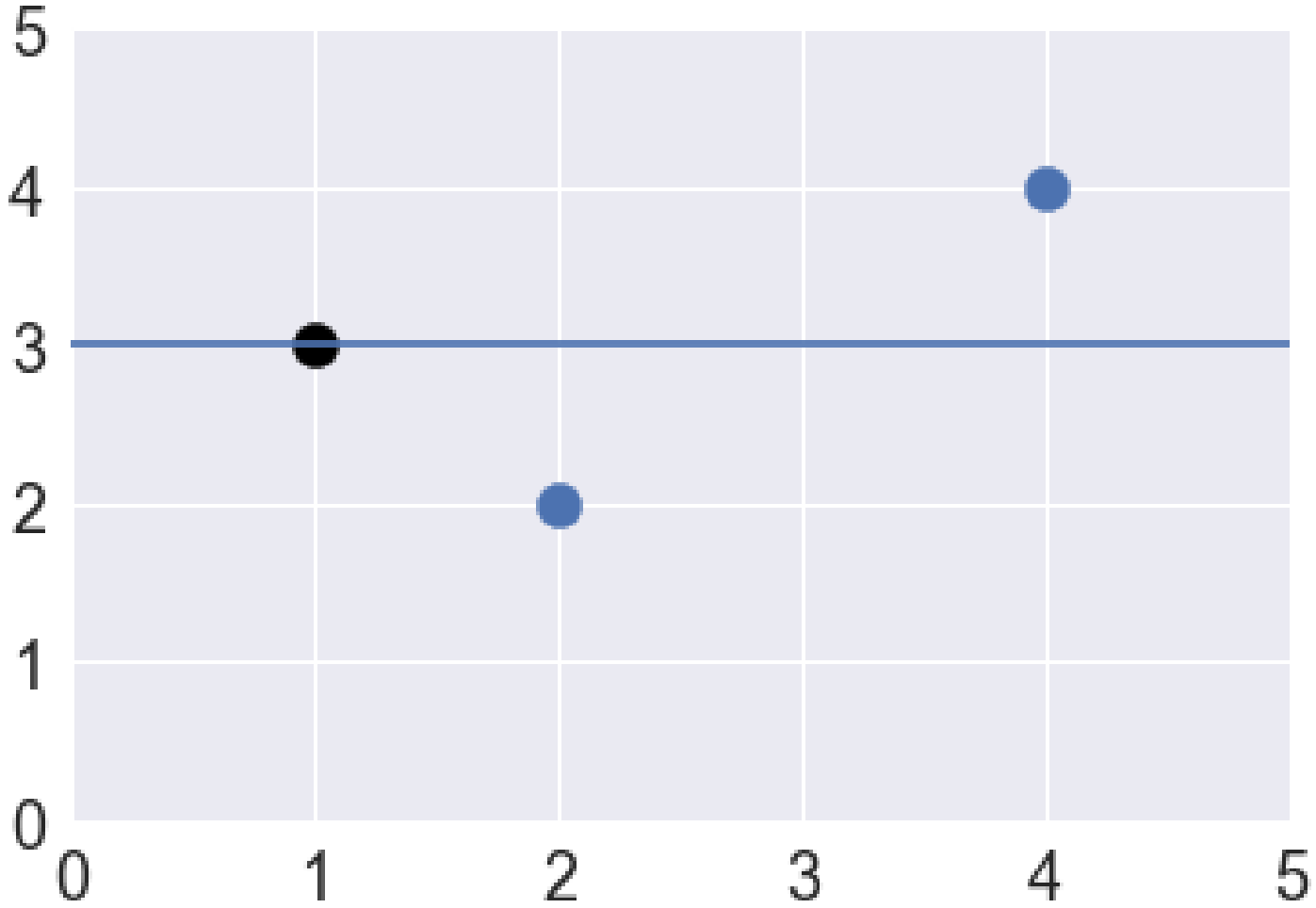
Simple example of leave one out Cross-Validation

The point $(2, 2)$ is in the test set, the loss is $(2 - 3.5)^2 = 1.5^2$.



Simple example of leave one out Cross-Validation

The point $(1, 3)$ is in the test set, the loss is $(3-3)^2 = 0^2 = 0$.



We conclude, that the cross-validation loss is $\frac{1}{3}\left(1.5^2 + 1.5^2 + 0\right)$.

CV bad example (1)

While the cross validation method will often work well in practice, it might fail. To see the, consider the following example of the cross validation procedure. Assume that we are dealing with $0-1$ classification problem, and suppose that the response label (0 or 1) are assigned to an explanatory variable x with probability $1/2$. Next, consider the following classifier:

$$g(\mathrm{x}) = \begin{cases} 0 & \mathrm{par}(\tau) = 0 \\ 1 & \mathrm{par}(\tau) = 1, \end{cases}$$

where $\mathrm{par}(\tau)$ is a parity function defined on the response labels $(y_1, \ldots, y_n)$ of the training set τ, namely

$$\mathrm{par}(\tau) = \left(\sum_{i=1} y_i \right) \mod 2.$$

CV bad example (2)

It is possible to show that the difference between the true loss and the leave-one-out cross-validation estimator is always $1/2$. To see this, consider two training datasets with the following labels $(1, 0, 0)$ and $(1, 1, 0)$. The following table provides the leave-one-out estimates (lous) and the corresponding zero-one loss for these datasets.

τ_1	lous	Loss	τ_2	lous	Loss
1	0	1	1	1	0
0	1	1	1	1	0
0	1	1	0	0	0

For the τ_1 dataset, the prediction is always wrong, and for the τ_2 dataset, the prediction is always correct.

CV bad example (3)

To prove this formally, let τ be any training set, and consider two cases: $\mathrm{par}(\tau) = 0$ and $\mathrm{par}(\tau) = 1$. In the first case, the number of ones is even, and for the second case, the number of ones is odd.

1. Suppose that $\mathrm{par}(\tau) = 0$, and let $\mathrm{par}(\tau_{-i})$ be the leave-one-out estimate for y_i for $i = 1, \ldots, n$.
   1. If $y_i = 0$, $\mathrm{par}(\tau_{-i}) = \mathrm{par}(\tau) = 0$. Thus, the prediction is correct and the loss is equal to 0.
   2. If $y_i = 1$, the number of ones in τ_{-i} is odd, and thus $\mathrm{par}(\tau_{-i}) = 1$; that is, the prediction is correct again, and thus the loss is equal to 0.
2. Suppose now that $\mathrm{par}(\tau) = 1$.
   1. If $y_i = 0$, $\mathrm{par}(\tau_{-i}) = \mathrm{par}(\tau) = 1$, and we conclude that prediction is wrong and thus the loss is equal to 1.
   2. For $y_i = 1$, it follows that $\mathrm{par}(\tau_{-i}) = 0$ (since the number of ones in τ_{-i} is even), and thus the prediction is wrong again. Consequently, the loss is equal to 1.

How many samples in validation/test sets revisited

1. the 50/25/25 percentage rule is OK, but...
2. We generally use some sort of metric, for example Mean square error, R^2, misclassification rate, etc.
3. To estimate these parameters, we can use concentration inequalities (that will give us an indication of the required sample size), such as:
   - Hoeffding's inequality
   - Chebyshev's inequality
   - Chernoff bounds
   - ...

An important topic - static estimation (1)

- Recall that we want to determine the expectation

$$\ell = \mathbb{E}[H(X)] = \int H(x) f(x) \mathrm{d}x,$$

  where ℓ is our statistics of interest (MSE, R^2,...).
- We can only use the validation/test set to calculate

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^{N} H(X_i),$$

  where $X_1, \ldots, X_N \overset{\text{iid}}{\sim} f(x)$.

Note that finding a point estimate $\hat{\ell}$ is generally not sufficient. We would like to find a confidence interval too.

An important topic - static estimation (2) — building the confidence interval

- Recall that by the central limit theorem, $\hat{\ell}$ has a $\mathrm{N}(\ell, \sigma^2/N)$ distribution.
- Despite that the σ^2 is generally unavailable, it can be estimated from Monte Carlo simulation via a calculation of a sample variance:

$$S^2 = \frac{1}{N-1} \sum_{i=1}^{N} \left(H(X_i) - \hat{\ell} \right)^2 .$$

As $N \to \infty$, $S^2 \to \sigma^2$.

Building the confidence interval (2)

Consequently, for large N, $\hat{\ell}$ has an approximately $\mathrm{N}(\ell, S^2/N)$ distribution. Thus,

$$\mathbb{P}\left(\hat{\ell} - z_{1-\alpha/2}\frac{S}{\sqrt{N}} \leq \ell \leq \hat{\ell} + z_{1-\alpha/2}\frac{S}{\sqrt{N}}\right) \approx 1-\alpha,$$

where z_γ denotes the γ quantile of the standard normal distribution. In particular, we constructed an approximate $(1-\alpha)100\%$ interval for ℓ, and it is equal to:

$$\left(\hat{\ell} \pm z_{1-\alpha/2}\frac{S}{\sqrt{N}}\right)$$

The above confidence interval can be used as a criterion for stopping the simulation, in particular, for an identification of the required sample size N.

The $z_{1-\alpha/2}$ values

- A $100(1-\alpha)\%$ confidence interval on ℓ is given by

$$\hat{\ell} - z_{1-\alpha/2}\frac{S}{\sqrt{n}} \leq \ell \leq \hat{\ell} + z_{1-\alpha/2}\frac{S}{\sqrt{n}}.$$

- Therefore,

$$\Phi(z_{1-\alpha/2}) - \Phi(-z_{1-\alpha/2}) = 1-\alpha \Rightarrow = 1 - \Phi(z_{1-\alpha/2}) + \Phi(-z_{1-\alpha/2}).$$

- Since $\alpha = 2\Phi(-z_{1-\alpha/2})$, we can choose $z_{1-\alpha/2}$ as follows:

1. $99\% \Rightarrow \alpha = 0.01 \Rightarrow \Phi(-z_{1-\alpha/2}) = 0.005 \Rightarrow z_{1-\alpha/2} = 2.57$
2. $98\% \Rightarrow\Rightarrow \alpha = 0.02 \Rightarrow \Phi(-z_{1-\alpha/2}) = 0.01 \Rightarrow z_{1-\alpha/2} = 2.32$
3. $95\% \Rightarrow\Rightarrow \alpha = 0.05 \Rightarrow \Phi(-z_{1-\alpha/2}) = 0.025 \Rightarrow z_{1-\alpha/2} = 1.96$
4. $90\% \Rightarrow\Rightarrow \alpha = 0.1 \Rightarrow \Phi(-z_{1-\alpha/2}) = 0.05 \Rightarrow z_{1-\alpha/2} = 1.64$

Determining the required sample size (1)

However, we will be in trouble when ℓ is very small. For example, suppose that $\ell = 10^{-10}$. In this case, a 0.05 confidence interval has a very little meaning. That is

$$\left(\hat{\ell} \pm 0.05\right)$$

carry very little information. In this case, we will use the concept of *relative* width of the confidence interval, defined by:

$$\left(\hat{\ell} \pm \frac{z_{1-\alpha/2}\frac{S}{\sqrt{N}}}{\hat{\ell}}\right).$$

In general, for small values of ℓ we will work with the so-called *relative error*, which is defined by:

$$\mathrm{RE} = \frac{\sqrt{\mathrm{Var}(\hat{\ell})}}{\mathbb{E}[\hat{\ell}]} = \frac{\sigma}{\ell\sqrt{N}}.$$

Determining the required sample size (2)

Consider the accuracy and hence efficiency of the Crude Monte Carlo (CMC) rare-event estimator. We would like to estimate the rare-event probability

$$\mathbb{P}(X \in \mathscr{X}^*) = \mathbb{E}[1_{\{X \in \mathscr{X}^*\}}] = \ell,$$

where X is sampled from $\mathscr{X}$, such that $\mathscr{X}^* \subseteq \mathscr{X}$, and $|\mathscr{X}^*| \ll |\mathscr{X}|$. Since $1_{\{X \in \mathscr{X}^*\}} \sim \text{Ber}(\ell)$, (that is $\text{Var}(1_{\{X \in \mathscr{X}^*\}}) = \ell(1-\ell)$) the RE of $\hat{\ell}_{\text{CMC}}$ is

$$\text{RE}\left(\hat{\ell}_{\text{CMC}}\right) = \frac{\sqrt{\text{Var}\left(\hat{\ell}_{\text{CMC}}\right)}}{\mathbb{E}\left(\hat{\ell}_{\text{CMC}}\right)} = \frac{\sqrt{\ell(1-\ell)/N}}{\ell}.$$

In the rare-event setting $\ell \ll 1$ so

$$\text{RE}\left(\hat{\ell}_{\text{CMC}}\right) \approx 1/\sqrt{N\ell}, \tag{2}$$

which imposes a serious challenge.

Determining the required sample size (3)

To see this, consider the rare-event probability $\ell \approx 10^{-12}$, and suppose that we are interested in a modest 10% RE. It is easy to verify from (2), that the required number of experiments N is about:

$$1/\sqrt{N\ell} = 0.1 \quad \Rightarrow \quad 1/\sqrt{N \times 10^{-12}} = 0.1 \quad \Rightarrow \quad N \geq 10^{14}.$$

Here, we discuss the mathematical intuition of our definition of the relative error. In particular, the RE is equal to the so -called coefficient of variation (CV). In theoretical computer science, we sometimes consider the squared coefficient of variation of the random variable X, that is given by:

$$\mathrm{CV}^2 = \frac{\mathrm{Var}(X)}{(\mathbb{E}[X])^2}.$$

Determining the required sample size (4)

Let us write $\mathrm{Var}(X) = \sigma^2$, $\mathbb{E}[X] = \mu$, and consider the Chebyshev's inequality.
Let X be a random variable with finite expected value μ and finite non-zero variance σ^2. Then for any real number $k > 0$,

$$\mathbb{P}\left(|X - \mu| \geq k\sigma\right) \leq \frac{1}{k^2}.$$

Setting $k = \epsilon\mu/\sigma$ (for some small ϵ, say $\epsilon = 0.05$), we arrive at:

$$\mathbb{P}\left(|X - \mu| \geq \epsilon\mu\right) \leq \frac{\sigma^2}{\epsilon^2\mu^2}.$$

Suppose now that we consider the Monte Carlo algorithm output $N^{-1}\sum_{i=1}^{N} X_i$, where $X_1, \ldots, X_N$ are independent realization of the random variable X. Then,

$$\mathbb{P}\left(\left|\frac{1}{N}\sum_{i=1}^{N} X_i - \mu\right| \geq \epsilon\mu\right) \leq \frac{\sigma^2/N}{\epsilon^2\mu^2} = \delta.$$

Determining the required sample size (5)

In this particular case, we also constructed a confidence interval, and we can say that

$$(1-\epsilon)\mu \leq \frac{1}{N}\sum_{i=1}^{N} X_i \leq (1+\epsilon)\mu,$$

with probability at least $1-\delta$.
If we know σ, we can also derive the required sample size, that is, we need to set:

$$\frac{\sigma^2/N}{\epsilon^2\mu^2} \leq \delta \quad \Rightarrow \quad N \geq \frac{1}{\epsilon^2\delta}\frac{\sigma^2}{\epsilon^2} = \frac{1}{\epsilon^2\delta}\mathrm{CV}^2.$$

Intuitively, we would like the δ to be small, say $\delta = 0.01$. This means that the CV^2 should also be small. In particular, the required sample size will be proportional to CV^2!

Classification problems

- Under the classification framework, we assume that there exists a set of objects, where each such object belongs to different class.

- For each object in the training data-set, we assume the knowledge of its true class value. More formally, let

$$\mathcal{T} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$$

  be a set of data entries.

- $\{X_i\}_{1 \leq i \leq n}$ assigned into categorical response variables $\{Y_i\}_{1 \leq i \leq n}$, where $Y_i \subseteq \{c_1, \ldots, c_k\}$ for $i = 1, \ldots, n$.

Binary Classification problems

- A *binary* classification problem, for which the response variable is to be classified into one of two classes, c_1 or c_2.

- For example, we might want to classify a satellite image as "cloudy" or "clear".

Multi-class Classification problems

- A *multi-class* classification problem, for which the response variable is to be classified into one of several classes, specifically, into $\{c_1, \ldots, c_k\}$, where $k > 2$. Note that the multi-class classification generalizes the binary case. As an example, consider a satellite image that should be classified as "cloudy", "clear", or "foggy".

Multi-label Classification problems

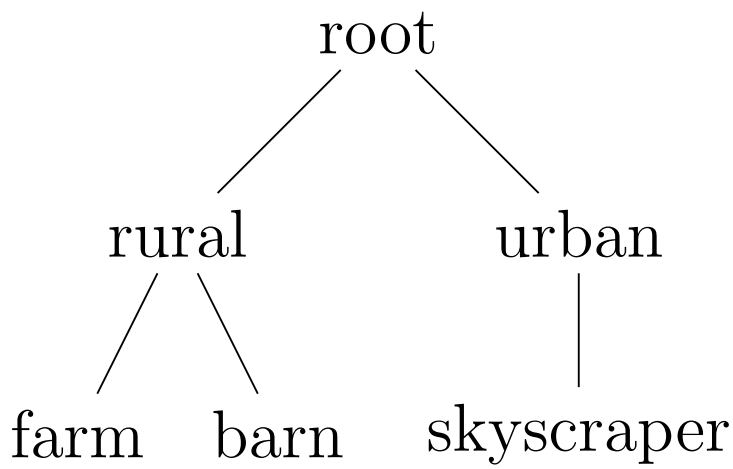
- A *multi-label* classification problem generalizes the multi-class classification task, by allowing multiple labels to be assigned to a data point X_i for all $i = 1, \ldots, n$. In this case, the response variable is a vector such that $Y_i \subseteq \{c_1, \ldots, c_k\}$.

- Consider the satellite image example and add two additional labels (classes), such as "road", and "river". Clearly, an image can contain both a road and a river. It can also be clear, cloudy, foggy, or all of the above.

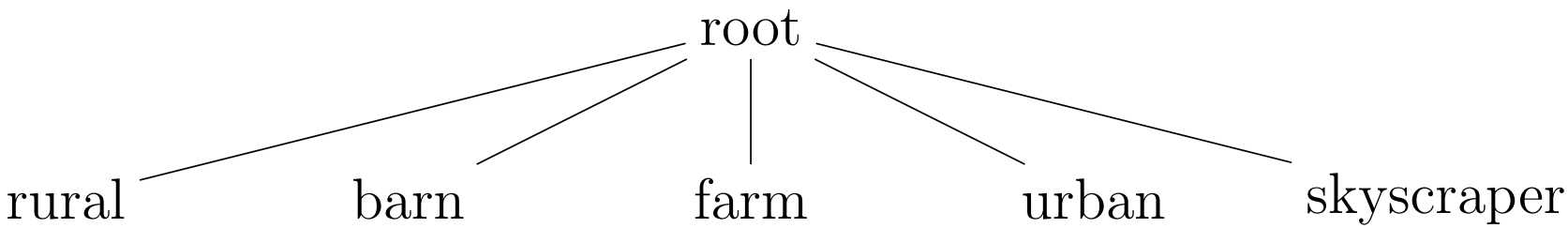
Hierarchical Classification problems

- In a *hierarchical* classification setting, we assume a relation among categories and take into account the corresponding hierarchical structure of classes during a classification process.
- The hierarchical classification problem generalizes all the above classification types and is common in text classification and bioinformatics.
- Usually, relations between classes are modeled via a tree or directed an acyclic graph.

Classification problems



(a) Hierarchical classification.



(b) Non-hierarchical classification

Figure 7: Hierarchical (a) and Non-hierarchical (b) classification schemes. Note that barns and farms are common in rural areas, while skyscrapers are generally located in cities. While this relation can be clearly observed in the hierarchical

The *confusion matrix*

The *confusion matrix* is a table that summarizes the performance of a classifier. Consider a binary classification problem with two classes, say "+" and "−". The corresponding confusion matrix skeleton is shown in the Table below.

	Actual class	
Predicted class	+	−
+	True positive	False positive
−	False negative	True negative

Table 2: Confusion matrix for a binary classification problem.

The *confusion matrix*

Example (Binary classification)

Consider a dataset consisting of 6 points and suppose that the true labels for $(X_1, \ldots, X_6)$ are $(+, +, -, -, +, -)$. Let us examine a certain classifier that outputs $(+, +, -, +, +, +)$ for $X_1, \ldots, X_6$, respectively.

	Actual class	
Predicted class	$+$	$-$
$+$	3	2
$-$	0	1

Table 3: Confusion matrix for Example 3.

The non-binary *confusion matrix*

The confusion matrix idea can be easily generalized to the non-binary case. Specifically, consider a classification problem with three classes. For example, suppose that we need to classify images with dogs, cats, and common brushtail possums. The Table shows the confusion matrix of some dog/cat/possum classifier.

	Actual class		
Predicted class	dog	cat	possum
dog	30	8	7
cat	2	22	4
possum	6	15	41

Table 4: Confusion matrix for three classes.

- Similar to the binary case, the diagonal elements represent the correct classifications and can be denoted as *true positive* entries. In this case the *true positive* counts for dogs, cats, and possums, are $30, 22$, and 41, respectively.

The non-binary *confusion matrix*

- Consequently, the *true negative* count for a class, is the sum of all matrix elements that does not belong to the raw and the column of this particular class.
- The *false positive* count for a specific class, is just the sum over the corresponding matrix raw (without the diagonal element count).
- Finally, the *false negative* count for a specific class, can be calculated by summing over the corresponding matrix column (again, without counting the diagonal element).

The *confusion matrix*

Formally, for each class $i \in \{1, \ldots, k\}$, let $\mathrm{tp}_i, \mathrm{tn}_i, \mathrm{fp}_i$, and fn_i be the true positive, true negative, false positive and false negative counts. Then, provided that the confusion matrix element in raw i and column j is denoted by $\mathrm{cm}(i, j)$, the following holds:

$$\mathrm{tp}_i = \mathrm{cm}(i, i),$$

$$\mathrm{fn}_i = \left(\sum_{j=1}^{k} \mathrm{cm}(j, i) \right) - \mathrm{cm}(i, i),$$

$$\mathrm{fp}_i = \left(\sum_{j=1}^{k} \mathrm{cm}(i, j) \right) - \mathrm{cm}(i, i),$$

$$\mathrm{tn}_i = \left(\sum_{j=1}^{k} \sum_{l=1}^{k} \mathrm{cm}(j, l) \right) - \mathrm{fn}_i - \mathrm{fp}_i - \mathrm{tp}_i.$$

The loss

A classifier can be evaluated using its confusion matrix. To begin with, define the binary classifier *accuracy* via

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}},$$

Suppose that we need to classify an instance into one of several classes, specifically, into $\{c_1, \ldots, c_k\}$, where $k > 2$. Then, the following metrics are of interest. Then, the following metrics are of interest.

- The *average accuracy* of a classifier:

$$\text{average accuracy} = \frac{\sum_{i=1}^{k} \frac{\text{tp}_i + \text{tn}_i}{\text{tp}_i + \text{tn}_i + \text{fp}_i + \text{fn}_i}}{k}$$

- The average per-class classification error (*error rate*):

$$\text{error rate} = \frac{\sum_{i=1}^{k} \frac{\text{fp}_i + \text{fn}_i}{\text{tp}_i + \text{tn}_i + \text{fp}_i + \text{fn}_i}}{k}$$

Many more possibilities for the classification loss

- The *precision*, or *positive predictive value* (already discussed above):

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}.$$

- The *recall*, or *sensitivity*, is the fraction of true positive instances among all "+" instances. The recall measures the effectiveness of a classifier to identify positive labels:

$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}.$$

- The *specificity* measures the effectiveness of a classifier to identify negative ("-") labels:

$$\text{specificity} = \frac{\text{tn}}{\text{fp} + \text{tn}}.$$

- The F_β score, is a combination of the precision and the recall and is used as a single measurement for a classifier performance:

$$F_\beta = \frac{(\beta^2 + 1)\,\text{tp}}{(\beta^2 + 1)\,\text{tp} + \beta^2\,\text{fn} + \text{fp}}.$$

Why loss should be application-dependent (1)

In some cases, the above average accuracy and the error rate are not sufficient for an adequate performance evaluation. To see this, consider the following classification problems based on a fingerprint detection system.

1. Fingerprint classification of authorized personnel in a top-secret military facility.
2. Fingerprint classification to get a discount in the local shop.

Both problems are binary classification problems. However, a miss-classification in the first problem is extremely dangerous, while a miss-classification in the second problem will make a customer happy.

Let us examine a classifier in the top-secret facility. The corresponding confusion matrix is given in Table below.

Why loss should be application-dependent (2)

	Actual class	
Predicted class	+	−
+	100	50
−	400	100,000

Table 5: Confusion matrix for fingerprinting secret-facility classification. The "+" and "-" stand for authorized and non-authorized personnel, respectively.

We conclude that the accuracy of classification is equal to

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}} = \frac{100 + 100,000}{100 + 100,000 + 50 + 400} \approx 99.55\%.$$

However, we can see that in this particular case, the accuracy is a problematic metric, since the algorithm allowed 50 non-authorized personnel to enter the facility. To handle this problem, we consider additional precision measures.