Machine Learning Basics Review

Akhil Vasvani

March 2019

1 Questions

Exercise 1. Can you state Tom Mitchell's definition of learning and discuss T, P and E?

Proof. Tom Mitchell provides the definition of learning: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." □

Exercise 2. What can be different types of tasks encountered in Machine Learning?

Proof. In this relatively formal definition of the word "task," the process of learning itself is not the task. Learning is our means of attaining the ability to perform the task. Machine learning tasks are usually described in terms of how the machine learning system should process an **example**. An example is a collection of **features** that have been quantitatively measured from some object or event that we want the machine learning system to process. We typically represent an example as a vector $\boldsymbol{x} \in \mathbb{R}^n$ where each entry x_i of the vector is another feature.

Here are some of the common machine learning tasks:

- **Classification:** In this type of task, the computer program is asked to specify which of k categories some input belongs to. To solve this task, the learning algorithm is usually asked to produce a function f: $\mathbb{R}^n \to \{1, ..., k\}$. When $y = f(\boldsymbol{x})$, the model assigns an input described by vector $boldsymbolx$ to a category identified by numeric code y. There are other variants of the classification task, for example, where f outputs a probability distribution over classes.

  Example. An example of a classification task is object recognition, where the input is an image (usually described as a set of pixel brightness values), and the output is a numeric code identifying the object in the image. For example, the Willow Garage PR2 robot is able to act as a waiter that can recognize different kinds of drinks and deliver them to people on command (Goodfellow et al., 2010). Modern object recognition is best accomplished with deep learning Krizhevsky et al. , 2012 ;Ioffe and Szegedy, 2015). Object recognition is the same basic technology that allows computers to recognize faces (Taigman et al., 2014), which can be used to automatically tag people in photo collections and allow computers to interact more naturally with their users.

- **Classification with missing inputs:** Classification becomes more challenging if the computer program is not guaranteed that every measurement in its input vector will always be provided. In order to solve the classification task, the learning algorithm only has to define a single function mapping from a vector input to a categorical output. When some of the inputs may be missing, rather than providing a single classification function, the learning algorithm must learn a *set* of functions. Each function corresponds to classifying $\boldsymbol{x}$ with a different subset of its inputs missing. This kind of situation arises frequently in medical diagnosis, because many kinds of medical tests are expensive or invasive. One way to efficiently define such a large set of functions is to learn a probability distribution over all of the relevant variables, then solve the classification task by marginalizing out the missing variables. With n input variables, we can now obtain all 2^n different classification functions needed for each possible set of missing inputs, but we only need to learn a single function describing the joint probability distribution.

- **Regression:** In this type of task, the computer program is asked to predict a numerical value given some input. To solve this task, the learning algorithm is asked to output a function $f\colon \mathbb{R}^n \to \mathbb{R}$. This type of task is similar to classification, except that the format of output is different.

  Example. An example of a regression task is the prediction of the expected claim amount that an insured person will make (used to set insurance premiums), or the prediction of future prices of securities. These kinds of predictions are also used for algorithmic trading.

- **Transcription:** In this type of task, the machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe it into discrete, textual form.

  Example. For example, in optical character recognition, the computer program is shown a photograph containing an image of text and is asked to return this text in the form of a sequence of characters (e.g., in ASCII or Unicode format). Google Street View uses deep learning to process address numbers in this way.

  Another example is speech recognition, where the computer program is provided an audio waveform and emits a sequence of characters or word ID codes describing the words that were spoken in the audio recording. Deep learning is a crucial component of modern speech recognition systems used at major companies including Microsoft, IBM and Google

- **Machine Translation:** In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language.

  Example. This is commonly applied to natural languages, such as translating from English to French.

- **Structured Output:** Structured output tasks involve any task where the output is a vector (or other data structure containing multiple values) with important relationships between the different elements. This is a broad category, and subsumes the transcription and translation tasks described above, but also many other tasks.

  Example. One example is parsing—mapping a natural language sentence into a tree that describes its grammatical structure and tagging nodes of the trees as being verbs, nouns, or adverbs, and so on.

  Another example is pixel-wise segmentation of images, where the computer program assigns every pixel in an image to a specific category. For example, deep learning can be used to annotate the locations of roads in aerial photographs (Mnih and Hinton , 2010). The output need not have its form mirror the structure of the input as closely as in these annotation-style tasks.

  For example, in image captioning, the computer program observes an image and outputs a natural language sentence describing the image.

- **Anomaly Detection:** In this type of task, the computer program sifts through a set of events or objects, and flags some of them as being unusual or atypical.

  Example. An example of an anomaly detection task is credit card fraud detection. By modeling your purchasing habits, a credit card company can detect misuse of your cards. If a thief steals your credit card or credit card information, the thief's purchases will often come from a different probability distribution over purchase types than your own. The credit card company can prevent fraud by placing a hold on an account as soon as that card has been used for an uncharacteristic purchase.

- **Synthesis and Sampling:** In this type of task, the machine learning algorithm is asked to generate new examples that are similar to those in the training data. Synthesis and sampling via machine learning can be useful for media applications where it can be expensive or boring for an artist to generate large volumes of content by hand.

  Example. For example, video games can automatically generate textures for large objects or landscapes, rather than requiring an artist to manually label each pixel (Luo et al. , 2013).

  In some cases, we want the sampling or synthesis procedure to generate some specific kind of output given the input. For example, in a speech synthesis task, we provide a written sentence and ask the program to emit an audio waveform containing a spoken version of that sentence. This is a kind of structured output task, but with the added qualification that there is no single correct output for each input, and we explicitly desire a large amount of variation in the output, in order for the output to seem more natural and realistic.

- **Imputation of Missing Values:** In this type of task, the machine learning algorithm is given a new example $\boldsymbol{x} \in \mathbb{R}^n$, but with some entries x_i of $\boldsymbol{x}$ missing. The algorithm must provide a prediction of the values of the missing entries.

- **Denoising:** In this type of task, the machine learning algorithm is given in input a *corrupted example* $\tilde{\boldsymbol{x}} \in \mathbb{R}^n$ obtained by an unknown corruption process from a *clean example* $\boldsymbol{x} \in \mathbb{R}^n$. The learner must predict the clean example $\boldsymbol{x}$ from its corrupted version $\tilde{\boldsymbol{x}}$, or more generally predict the conditional probability distribution $p(\boldsymbol{x} \mid \tilde{\boldsymbol{x}})$.

- **Density estimation or probability mass function estimation:** In the density estimation problem, the machine learning algorithm is asked to learn a function p_{model}: $\mathbb{R}^n \to \mathbb{R}$, where $p_{\text{model}}(\boldsymbol{x})$ can be interpreted as a probability density function (if $\mathbf{x}$ is continuous) or a probability mass function (if $\mathbf{x}$ is discrete) on the space that the examples were drawn from. To do such a task well (we will specify exactly what that means when we discuss performance measures P), the algorithm needs to learn the structure of the data it has seen. It must know where examples cluster tightly and where they are unlikely to occur. Most of the tasks described above require the learning algorithm to at least implicitly capture the structure of the probability distribution. Density estimation allows us to explicitly capture that distribution. In principle, we can then perform computations on that distribution in order to solve the other tasks as well.

  Example. For example, if we have performed density estimation to obtain a probability distribution $p(\boldsymbol{x})$, we can use that distribution to solve the missing value imputation task. If a value x_i is missing and all of the other values, denoted $\boldsymbol{x}_{-i}$, are given, then we know the distribution over it is given by $p(x_i \mid \boldsymbol{x}_{-i})$. In practice, density estimation does not always allow us to solve all of these related tasks, because in many cases the required operations on $p(\boldsymbol{x})$ are computationally intractable.

□

Exercise 3. What are supervised, unsupervised, semi-supervised, self-supervised, multi-instance learning, and reinforcement learning?

Proof. Machine learning algorithms can be broadly categorized as **unsupervised** or textbf-supervised by what kind of experience they are allowed to have during the learning process.

Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset. In the context of deep learning, we usually want to learn the entire probability distribution that generated a dataset, whether explicitly as in density estimation or implicitly for tasks like synthesis or denoising. Some other unsupervised learning algorithms perform other roles, like clustering, which consists of dividing the dataset into clusters of similar examples.

Supervised learning algorithms experience a dataset containing features, but each example is also associated with a **label** or **target**. For example, the Iris dataset is annotated with the species of each iris plant. A supervised learning algorithm can study the Iris dataset and learn to classify iris plants into three different species based on their measurements.

Roughly speaking, unsupervised learning involves observing several examples of a random vector $\mathbf{x}$, and attempting to implicitly or explicitly learn the probability distribution $p(\mathbf{x})$, or some interesting properties of that distribution, while supervised learning involves observing several examples of a random vector $\mathbf{x}$ and an associated value or vector $\mathbf{y}$, and learning to

predict $\mathbf{y}$ from $\mathbf{x}$, usually by estimating $p(\mathbf{y} \mid \mathbf{x})$. The term **supervised learning** originates from the view of the target $\mathbf{y}$ being provided by an instructor or teacher who shows the machine learning system what to do. In unsupervised learning, there is no instructor or teacher, and the algorithm must learn to make sense of the data without this guide.

Other variants of the learning paradigm are possible. For example, in **semi-supervised learning**, some examples include a supervision target but others do not. In **multi-instance learning**, an entire collection of examples is labeled as containing or not containing an example of a class, but the individual members of the collection are not labeled. Some machine learning algorithms do not just experience a fixed dataset. For example, **reinforcement learning** algorithms interact with an environment, so there is a feedback loop between the learning system and its experiences. Such algorithms are beyond the scope of this book. □

Exercise 4. Loosely how can supervised learning be converted into unsupervised learning and vice-versa?

Proof. Unsupervised learning and supervised learning are not formally defined terms. The lines between them are often blurred. Many machine learning technologies can be used to perform both tasks. For example, the chain rule of probability states that for a vector $\mathbf{x} \in \mathbb{R}^n$, the joint distribution can be decomposed as

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(\mathrm{x}_i \mid \mathrm{x}_1, ..., \mathrm{x}_{i-1}). \tag{1}$$

This decomposition means that we can solve the ostensibly unsupervised problem of modeling $p(\mathbf{x})$ by splitting it into n supervised learning problems.

Alternatively, we can solve the supervised learning problem of learning $p(y \mid \mathbf{x})$ by using traditional unsupervised learning technologies to learn the joint distribution $p(\mathbf{x}, y)$ and inferring

$$p(y \mid \mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')} \tag{2}$$

□

Exercise 5. Consider linear regression. What are T, P and E?

Proof. As the name implies, linear regression solves a regression problem.

In other words, our task T is to build a system that can predict the value of a scalar $y \in \mathbb{R}$ from taking a vector of features $\boldsymbol{x} \in \mathbb{R}^n$ as input via outputting $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$, where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of **parameters**. Think of $\boldsymbol{w}$ as a set of weights that determine how each feature affects the prediction — if a feature's weight is large in magnitude, then it has a large effect on the prediction; otherwise if a feature's weight is zero, it has no effect on the prediction.

The performance of the model, P, is measured by the **mean squared error** of the model on the test set. The mean squared error is given by:

$$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}^{(\mathrm{test})} - \boldsymbol{y}^{(\mathrm{test})})_i^2 = \frac{1}{m} ||\hat{\boldsymbol{y}}^{(\mathrm{test})} - \boldsymbol{y}^{(\mathrm{test})}||_2^2.$$

We need to design an algorithm that will improve the weights $\boldsymbol{w}$ in a way that reduces MSE_{test} when the algorithm is allowed to gain experience E by observing a training set $(\boldsymbol{X}^{(\text{train})}, \boldsymbol{y}^{(\text{train})})$:

$$\boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\right)^{-1}\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}.$$

□

Exercise 6. Derive the normal equation for linear regression.

Proof. To minimize ($\text{MSE}_{\text{train}}$), we can simply solve for where its gradient is $\mathbf{0}$:

$$\begin{aligned}
&\nabla_{\boldsymbol{w}}\text{MSE}_{\text{train}} = 0 \\
\Rightarrow\ &\nabla_{\boldsymbol{w}}\frac{1}{m}||\hat{\boldsymbol{y}}^{(\text{train})} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0 \\
\Rightarrow\ &\frac{1}{m}\nabla_{\boldsymbol{w}}||\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0 \\
\Rightarrow\ &\nabla_{\boldsymbol{w}}\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right)^\top\left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right) = 0 \\
\Rightarrow\ &\nabla_{\boldsymbol{w}}\left(\boldsymbol{w}^\top\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{w}^\top\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} + \boldsymbol{y}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}\right) = 0 \\
\Rightarrow\ &2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} = 0 \\
\Rightarrow\ &\boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\right)^{-1}\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}.
\end{aligned}$$

The system of equations whose solution is given above is known as the **normal equations**.

Note. *Matrix Differentiation*

If $\boldsymbol{a}, \boldsymbol{b}$ are $k \times 1$ vectors, then

$$\frac{\partial}{\partial \boldsymbol{b}}\left[\boldsymbol{a}^\top\boldsymbol{b}\right] = \frac{\partial}{\partial \boldsymbol{b}}\left[\boldsymbol{b}^\top\boldsymbol{a}\right] = \boldsymbol{a}.$$

If $\boldsymbol{A}$ is any symmetric matrix, then

$$\frac{\partial}{\partial \boldsymbol{b}}\left[\boldsymbol{b}^\top\boldsymbol{A}\boldsymbol{b}\right] = 2\boldsymbol{A}\boldsymbol{b} = 2\boldsymbol{b}^\top\boldsymbol{A}.$$

Note. Take a look at the normal equation that we just derived. It has the matrix inversion in it and inverting a matrix is an expensive operation. Our design matrix $\boldsymbol{X}$ has $k+1$ columns where k is the number of predictors $(x^{(1)}, x^{(2)}, x^{(3)}, ...)$ and m rows of samples. In most real life situations, k is easily greater than 1,000 and sample size will be greater than 100,000. Since the matrix inversion is $(O(n^3))$, inverting $\boldsymbol{X}^\top\boldsymbol{X}$ (1,000 by 1,000 matrix) will take a while to calculate. Hence, the reason why we would use Gradient Descent in Linear Regression is because it's computationally cheaper to find optima. However, if the sample size is small enough, just use the normal equations.

TL;DR. When we use Gradient Descent, we have to scale the data. When we use normal equation, we do not have to.

□

Exercise 7. What do you mean by affine transformation? Discuss affine vs. linear transformation.

Proof. It is worth noting that the term **linear regression** is often used to refer to a slightly more sophisticated model with one additional parameter—an intercept term b. In this model

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

so the mapping from parameters to predictions is still a linear function but the mapping from features to predictions is now an affine function. This extension to affine functions means that the plot of the model's predictions still looks like a line, but it need not pass through the origin. □

Exercise 8. Discuss training error, test error, generalization error, overfitting, and underfitting.

Proof. The central challenge in machine learning is that we must perform well on new, previously unseen inputs—not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called **generalization**.

Typically, when training a machine learning model, we have access to a training set, we can compute some error measure on the training set called the **training error**, and we reduce this training error. So far, what we have described is simply an optimization problem. What separates machine learning from optimization is that we want the **generalization error**, also called the **test error**, to be low as well. The generalization error is defined as the expected value of the error on a new input. Here the expectation is taken across different possible inputs, drawn from the distribution of inputs we expect the system to encounter in practice.

Of course, when we use a machine learning algorithm, we do not fix the parameters ahead of time, then sample both datasets. We sample the training set, then use it to choose the parameters to reduce training set error, then sample the test set. Under this process, the expected test error is greater than or equal to the expected value of training error. The factors determining how well a machine learning algorithm will perform are its ability to:

1. Make the training error small.

2. Make the gap between training and test error small.

These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.

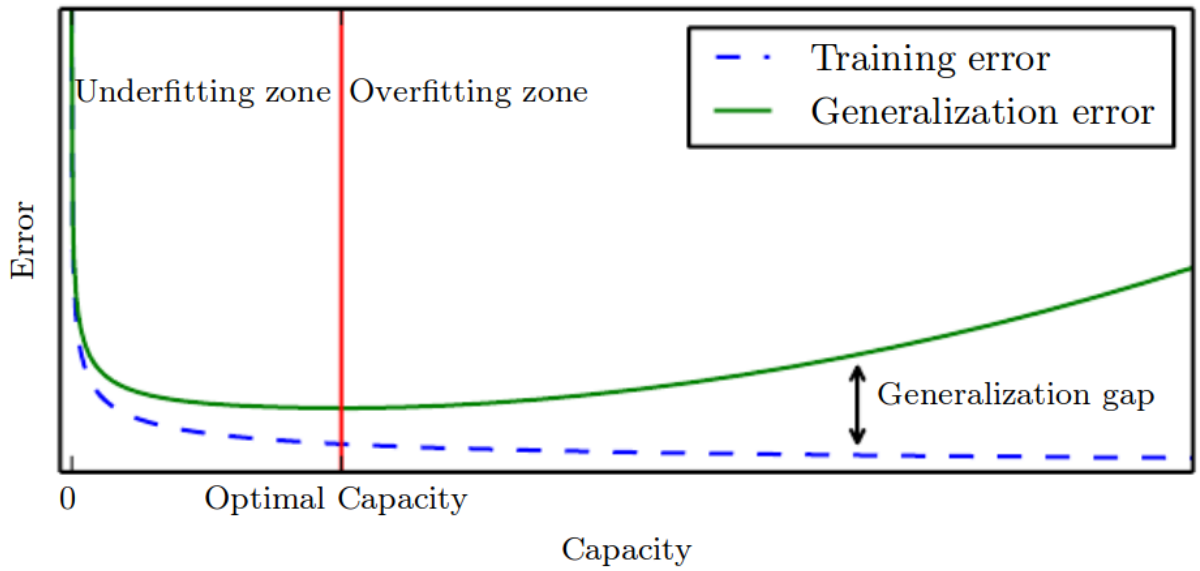

Figure 1: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the **underfitting regime**. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the **overfitting regime**, where capacity is too large, above the **optimal capacity**.

□

Exercise 9. Compare representational capacity vs. effective capacity of a model.

Proof. So far we have described only one way of changing a model's capacity: by changing the number of input features it has, and simultaneously adding new parameters associated with those features. There are in fact many ways of changing a model's capacity. Capacity is not determined only by the choice of model. The model specifies which family of functions the learning algorithm can choose from when varying the parameters in order to reduce a training objective. This is called the **representational capacity** of the model. In many cases, finding the best function within this family is a very difficult optimization problem. In practice, the learning algorithm does not actually find the best function, but merely one that significantly reduces the training error. These additional limitations, such as the imperfection of the optimization algorithm, mean that the learning algorithm's **effective capacity** may be less than the representational capacity of the model family. □

Exercise 10. Discuss VC dimension.

Proof. Statistical learning theory provides various means of quantifying model capacity. Among these, the most well-known is the **Vapnik-Chervonenkis dimension**, or **VC dimension**. The VC dimension measures the capacity of a binary classifier. The VC dimension is defined as being the largest possible value of m for which there exists a training set of m different $\boldsymbol{x}$ points that the classifier can label arbitrarily. □

Exercise 11. What are non-parametric models? What is non-parametric learning?

Proof. To reach the most extreme case of arbitrarily high capacity, we introduce the concept of **non-parametric** models. So far, we have seen only parametric models, such as linear regression. Parametric models learn a function described by a parameter vector whose size is finite and fixed before any data is observed. Non-parametric models have no such limitation.

Sometimes, non-parametric models are just theoretical abstractions (such as an algorithm that searches over all possible probability distributions) that cannot be implemented in practice. However, we can also design practical non-parametric models by making their complexity a function of the training set size. One example of such an algorithm is **nearest neighbor regression**. Unlike linear regression, which has a fixed-length vector of weights, the nearest neighbor regression model simply stores the $\boldsymbol{X}$ and $\boldsymbol{y}$ from the training set. When asked to classify a test point $\boldsymbol{x}$, the model looks up the nearest entry in the training set and returns the associated regression target. In other words, $\hat{y} = y_i$ where $i = \text{argmin}||\boldsymbol{X}_{i,:} - \boldsymbol{x}||_2^2$. The algorithm can also be generalized to distance metrics other than the L^2 norm, such as learned distance metrics (Goldberger et al., 2005). If the algorithm is allowed to break ties by averaging the y_i values for all $\boldsymbol{x}_{i,:}$ that are tied for nearest, then this algorithm is able to achieve the minimum possible training error (which might be greater than zero, if two identical inputs are associated with different outputs) on any regression dataset.

Finally, we can also create a non-parametric learning algorithm by wrapping a parametric learning algorithm inside another algorithm that increases the number of parameters as needed. For example, we could imagine an outer loop of learning that changes the degree of the polynomial learned by linear regression on top of a polynomial expansion of the input.

□

Exercise 12. What is an ideal model? What is Bayes error? What is/are the source(s) of Bayes error occur?

Proof. The ideal model is an oracle that simply knows the true probability distribution that generates the data. Even such a model will still incur some error on many problems, because there may still be some noise in the distribution. In the case of supervised learning, the mapping from $\boldsymbol{x}$ to y may be inherently stochastic, or y may be a deterministic function that involves other variables besides those included in $\boldsymbol{x}$. The error incurred by an oracle making predictions from the true distribution $p(\boldsymbol{x}, y)$ is called the **Bayes error**.

Example. The effect of the training dataset size on the train and test error, as well as on the optimal model capacity. We constructed a synthetic regression problem based on adding a moderate amount of noise to a degree-5 polynomial, generated a single test set, and then generated several different sizes of training set. For each size, we generated 40 different training sets in order to plot error bars showing 95 percent confidence intervals.

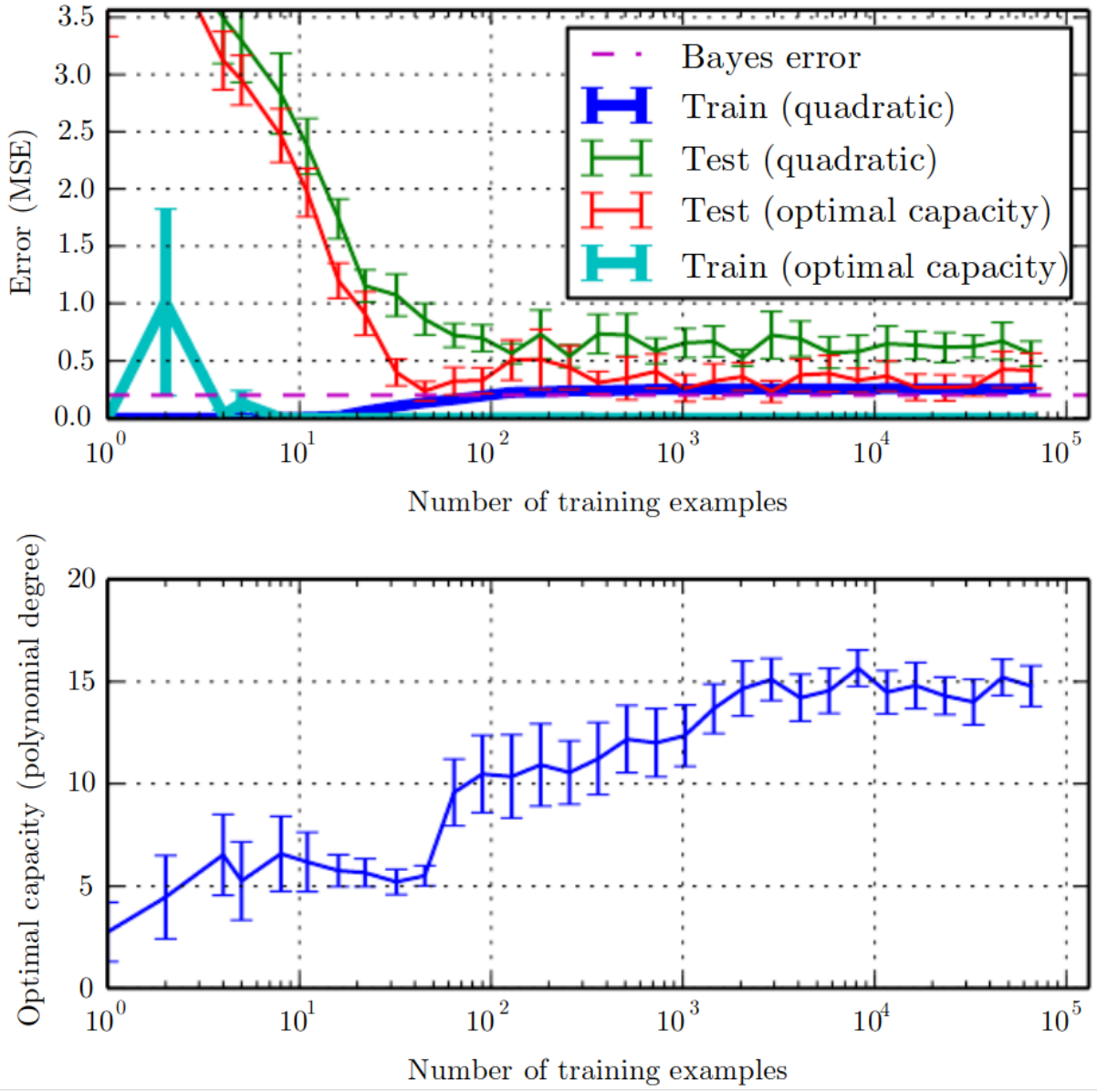

Figure 2: *(Top)* The MSE on the training and test set for two different models: a quadratic model, and a model with degree chosen to minimize the test error. Both are fit in closed form. For the quadratic model, the training error increases as the size of the training set increases. This is because larger datasets are harder to fit. Simultaneously, the test error decreases, because fewer incorrect hypotheses are consistent with the training data. The quadratic model does not have enough capacity to solve the task, so its test error asymptotes to a high value. The test error at optimal capacity asymptotes to the Bayes error. The training error can fall below the Bayes error, due to the ability of the training algorithm to memorize specific instances of the training set. As the training size increases to infinity, the training error of any fixed-capacity model (here, the quadratic model) must rise to at least the Bayes error. *(Bottom)* As the training set size increases, the optimal capacity (shown here as the degree of the optimal polynomial regressor) increases. The optimal capacity plateaus after reaching sufficient complexity to solve the task.

□

Exercise 13. What is the no free lunch theorem in connection to Machine Learning?

Proof. Learning theory claims that a machine learning algorithm can generalize well from a finite training set of examples. This seems to contradict some basic principles of logic because to logically infer a rule describing every member of a set, one must have information about every member of that set.

In part, machine learning avoids this problem by offering only probabilistic rules, rather than the entirely certain rules used in purely logical reasoning. Machine learning promises to find rules that are probably correct about most members of the set they concern.

Unfortunately, even this does not resolve the entire problem. The **no free lunch theorem** for machine learning (Wolpert, 1996) states that, averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points. In other words, in some sense, no machine learning algorithm is universally any better than any other. The most sophisticated algorithm we can conceive of has the same average performance (over all possible tasks) as merely predicting that every point belongs to the same class.

Note. Fortunately, these results hold only when we average over all possible data generating distributions. If we make assumptions about the kinds of probability distributions we encounter in real-world applications, then we can design learning algorithms that perform well on these distributions.

This means that the goal of machine learning research is not to seek a universal learning algorithm or the absolute best learning algorithm. Instead, our goal is to understand what kinds of distributions are relevant to the "real world" that an AI agent experiences, and what kinds of machine learning algorithms perform well on data drawn from the kinds of data generating distributions we care about.

TL;DR: The no free lunch theorem implies that we must design our machine learning algorithms to perform well on a specific task. We do so by building a set of preferences into the learning algorithm. When these preferences are aligned with the learning problems we ask the algorithm to solve, it performs better.

□

Exercise 14. What is weight decay? What is it added?

Proof. We can give a learning algorithm a preference for one solution in its hypothesis space to another. This means that both functions are eligible, but one is preferred. The unpreferred solution will be chosen only if it fits the training data significantly better than the preferred solution.

Example. For example, we can modify the training criterion for linear regression to include weight decay. To perform linear regression with **weight decay**, we minimize a sum comprising both the mean squared error on the training and a criterion $J(\boldsymbol{w})$ that expresses a preference for the weights to have smaller squared L^2 norm. Specifically,

$$J(\boldsymbol{w}) = \text{MSE}_{\text{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}, \tag{3}$$

where λ is a value chosen ahead of time that controls the strength of our preference for smaller weights. When $\lambda = 0$, we impose no preference, and larger λ forces the weights to become smaller. Minimizing $J(\boldsymbol{w})$ results in a choice of weights that make a tradeoff between fitting the training data and being small. This gives us solutions that have a smaller slope, or put weight on fewer of the features. As an example of how we can control a model's tendency to overfit or underfit via weight decay, we can train a high-degree polynomial regression model with different values of λ.

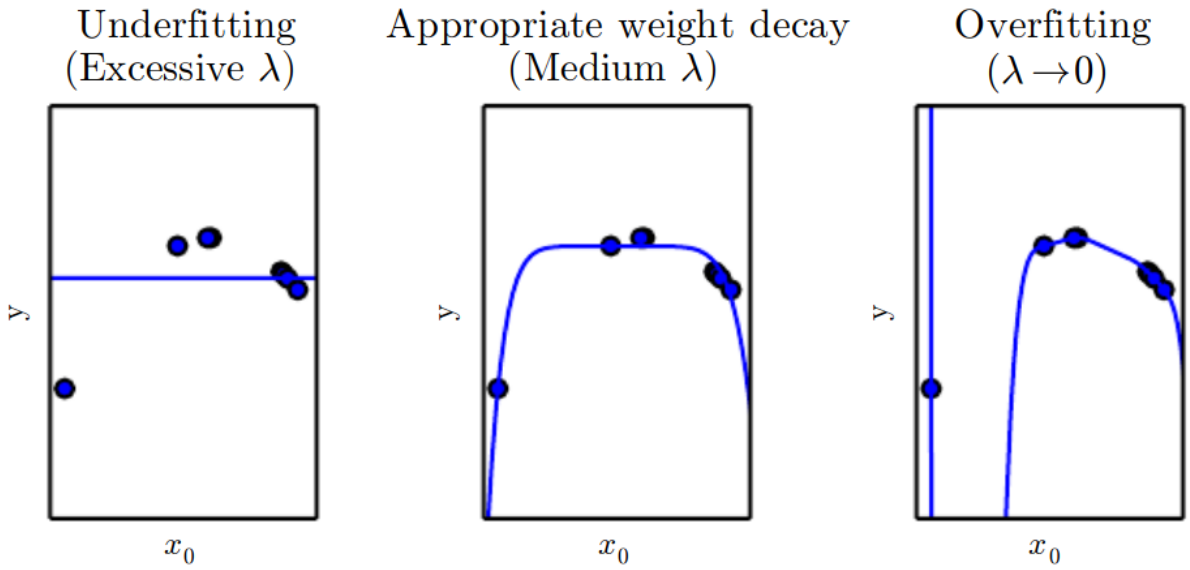

Figure 3: We fit a high-degree polynomial regression model to our example training set. The true function is quadratic, but here we use only models with degree 9. We vary the amount of weight decay to prevent these high-degree models from overfitting. *(Left)* With very large λ, we can force the model to learn a function with no slope at all. This underfits because it can only represent a constant function. *(Center)* With a medium value of λ, the learning algorithm recovers a curve with the right general shape. Even though the model is capable of representing functions with much more complicated shape, weight decay has encouraged it to use a simpler function described by smaller coefficients. *(Right)* With weight decay approaching zero (i.e., using the Moore-Penrose pseudoinverse to solve the underdetermined problem with minimal regularization), the degree-9 polynomial overfits significantly

TL;DR. Large weight decay results in underfitting, medium weight decay is just right, and small weight decay results in overfitting.

□

Exercise 15. What is regularization? Intuitively, what does regularization do during the optimization procedure? (expresses preferences to certain solutions, implicitly and explicitly)

Proof. More generally, we can regularize a model that learns a function $f(\boldsymbol{x}; \boldsymbol{\theta})$ by adding a penalty called a **regularizer** to the cost function. In the case of weight decay, the regularizer term is $\Omega(\boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{w}$.

In our weight decay example, we expressed our preference for linear functions defined with smaller weights explicitly, via an extra term in the criterion we minimize. There are many other ways of expressing preferences for different solutions, both implicitly and explicitly. Together, these different approaches are known as **regularization**. *Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.* Regularization is one of the central concerns of the field of machine learning, rivaled in its importance only by optimization.

□

Exercise 16. What is a hyperparameter? How do you choose which settings are going to be hyperparameters and which are going to be learnt?

Proof. Most machine learning algorithms have several settings that we can use to control the behavior of the learning algorithm. These settings are called **hyperparameters**. The values of hyperparameters are not adapted by the learning algorithm itself (though we can design a nested learning procedure where one learning algorithm learns the best hyperparameters for another learning algorithm).

Sometimes a setting is chosen to be a hyperparameter that the learning algorithm does not learn because it is difficult to optimize. More frequently, the setting must be a hyperparameter because it is not appropriate to learn that hyperparameter on the training set. This applies to all hyperparameters that control model capacity. If learned on the training set, such hyperparameters would always choose the maximum possible model capacity, resulting in overfitting.

Example. For example, we can always fit the training set better with a higher degree polynomial and a weight decay setting of $\lambda = 0$ than we could with a lower degree polynomial and a positive weight decay setting.

TL;DR. Hyperparameters are settings that we can use to control the behaviour of the learning algorithm. The setting must be a hyperparameter because it is either difficult to optimize or not appropriate to learn that hyperparameter for the training set. For instance, hyperparameter controlling model capacity where it would always choose to maximize the model capacity for the training set that results in overfitting.

□

Exercise 17. Why is a validation set necessary?

Proof. To solve this problem, we need a **validation set** of examples that the training algorithm does not observe.

A held-out test set, composed of examples coming from the same distribution as the training set, can be used to estimate the generalization error of a learner, after the learning process has completed. It is important that the test examples are not used in any way to make choices about the model, including its hyperparameters. For this reason, no example from the test set can be used in the validation set. Therefore, we always construct the validation set from the training data. Specifically, we split the training data into two disjoint subsets. One of these subsets is used to learn the parameters. The other subset is our validation set, used to estimate the generalization error during or after training, allowing for the hyperparameters

to be updated accordingly. The subset of data used to learn the parameters is still typically called the training set, even though this may be confused with the larger pool of data used for the entire training process. The subset of data used to guide the selection of hyperparameters is called the validation set. Typically, one uses about 80% of the training data for training and 20% for validation. Since the validation set is used to "train" the hyperparameters, the validation set error will underestimate the generalization error, though typically by a smaller amount than the training error. □

Exercise 18. What are the different types of cross-validation? When do you use which one?

Proof. Cross Validation is a very useful technique for assessing the effectiveness of your model, particularly in cases where you need to mitigate overfitting or underfitting. It is also of use in determining the hyper parameters of your model, in the sense that which parameters will result in lowest generalization error. There are two types of cross-validation: **exhaustive** and **non-exhaustive**. Exhaustive cross-validation methods are cross-validation methods which learn and test on all possible ways to divide the original sample into a training and a validation set. The following are exhaustive cross-validation techniques:

- **LOOCV:** Leave-one-out cross validation is k-fold cross validation taken to its logical extreme, with k equal to n, the number of data points in the set. That means that n separate times, the function approximator is trained on all the data except for one point and a prediction is made for that point. As before the average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error (LOO-XVE) is good, but at first pass it seems very expensive to compute. Fortunately, locally weighted learners can make LOO predictions just as easily as they make regular predictions. That means computing the LOO-XVE takes no more time than computing the residual error and it is a much better way to evaluate models.

- **LPOCV:** This approach is a generalization of the LOOCV method because it leaves p data points out of training data, i.e. if there are n data points in the original sample then, $n-p$ samples are used to train the model and p points are used as the validation set. This is repeated for all combinations in which original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness.

  Note. This method is exhaustive in the sense that it needs to train and validate the model for all possible combinations, and for moderately large p, it can become computationally infeasible.

Non-exhaustive cross validation methods do not compute all ways of splitting the original sample. In laymen's terms, you have to decide how many subsets need to be made. The following are non-exhaustive cross-validation techniques:

- **Holdout Method:** The holdout method is the simplest kind of cross validation. The data set is separated into two sets, called the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set (it has never seen these output values before). The errors it makes are accumulated as

before to give the mean absolute test set error, which is used to evaluate the model. The advantage of this method is that it is usually preferable to the residual method and takes no longer to compute. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made.

- **K-fold cross validation** K-fold cross validation is one way to improve over the holdout method. The data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other $k-1$ subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set $k-1$ times. The variance of the resulting estimate is reduced as k is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation.

  Note. k is usually set between 5-10.

- **Stratified K-fold Cross Validation:** Stratified k-fold cross validation splits the data into folds governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value.

- **Repeated K-fold Cross Validation:** This is a variant of where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over.

Now the question becomes when to use which one? Let's start with the exhaustive methods first.

- **LOOCV:** Good for less data and unbalanced dataset and target values.

- **LPOCV:** Good for less data and unbalanced dataset and target values.

- **Holdout:** The hold-out method is good to use when you have a very large dataset, you have a limited amount of time, or you are starting to build an initial model. Sometimes it is easier to start with a basic model as a reference point to compare with other more complex models. If the results (MSE or RMSE) are the same between the complex models and the simpler ones choose the simpler one (Occam's Razor).

- **k-fold Cross validation:** Cross-validation is usually the preferred method because it gives your model the opportunity to train on multiple train-test splits. This gives you a better indication of how well your model will perform on unseen data.

- **Stratified k-Fold Cross Validation:** In some cases, there may be a large imbalance in the response variables. For example, in dataset concerning price of houses, there might be large number of houses having high price. Or in case of classification, there might be several times more negative samples than positive samples. For such problems, a slight variation in the K Fold cross validation technique is made, such that each fold contains approximately the same percentage of samples of each target class as the complete set, or in case of prediction problems, the mean response value is approximately equal in all the folds.

- **Repeated k-Fold Cross Validation:** Good for huge data. If you are getting similar scores and optimal model's parameters with some of your iterations.

□

Exercise 19. What is the maximal likelihood of a parameter vector θ? Where does the log come from?

Proof. Consider a set of m examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\boldsymbol{x})$.

Let $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ be a parametric family of probability distributions over the same space indexed by $\boldsymbol{\theta}$. In other words, $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ maps any configuration $\boldsymbol{x}$ to a real number estimating the true probability $p_{data}(\boldsymbol{x})$.

The maximum likelihood estimator for θ is then defined as

$$\boldsymbol{\theta}_{\text{ML}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \tag{4}$$

$$= \underset{\boldsymbol{\theta}}{\text{argmax}} \prod_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{5}$$

This product over many probabilities can be inconvenient for a variety of reasons. For example, it is prone to numerical underflow. To obtain a more convenient but equivalent optimization problem, we observe that taking the logarithm of the likelihood does not change its arg max but does conveniently transform a product into a sum:

$$\boldsymbol{\theta}_{\text{ML}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}). \tag{6}$$

Because the arg max does not change when we rescale the cost function, we can divide by m to obtain a version of the criterion that is expressed as an expectation with respect to the empirical distribution $\hat{p}_{\text{data}}$ a defined by the training data:

$$\boldsymbol{\theta}_{\text{ML}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \tag{7}$$

Note. One way to interpret maximum likelihood estimation is to view it as minimizing the dissimilarity between the empirical distribution $\hat{p}_{\text{data}}$ a defined by the training set and the

model distribution, with the degree of dissimilarity between the two measured by the KL divergence. The KL divergence is given by

$$D_{\mathrm{KL}}(\hat{p}_{\mathrm{data}}||p_{\mathrm{model}}) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}}[\log \hat{p}_{\mathrm{data}}(\boldsymbol{x}) - \log p_{\mathrm{model}}(\boldsymbol{x})].$$

The term on the left is a function only of the data generating process, not the model. This means when we train the model to minimize the KL divergence, we need only minimize

$$-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}}[\log p_{\mathrm{model}}(\boldsymbol{x})],$$

which is of course the same as the maximization in equation 7.

□

Exercise 20. Prove that for linear regression MSE can be derived from maximal likelihood by proper assumptions.

Proof. To derive the same linear regression algorithm we obtained before, we define $p(y\,|\,\boldsymbol{x}) = \mathcal{N}(y; \hat{y}(\boldsymbol{x};\boldsymbol{w}), \sigma^2)$. The function $\hat{y}(\boldsymbol{x};\boldsymbol{w})$ gives the prediction of the mean of the Gaussian. In this example, we assume that the variance is fixed to some constant σ^2 chosen by the user. We will see that this choice of the functional form of $p(y\,|\,\boldsymbol{x})$ causes the maximum likelihood estimation procedure to yield the same learning algorithm as we developed before.

Since the examples are assumed to be i.i.d., the conditional log-likelihood is given by

$$\sum_{i=1}^{m} \log p(y^{(i)}\,|\,\boldsymbol{x}^{(i)};\boldsymbol{w}) = \sum_{i=1}^{m} \log \frac{e^{-\frac{||\hat{y}^{(i)}-y^{(i)}||^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} = -\sum_{i=1}^{m} \log\sigma - \sum_{i=1}^{m} \frac{1}{2}\log(2\pi) - \sum_{i=1}^{m} \frac{||\hat{y}^{(i)}-y^{(i)}||^2}{2\sigma^2}$$

$$\Rightarrow -m\log\sigma - \frac{m}{2}\log(2\pi) - \frac{1}{2\sigma^2}(m)\Big[\frac{1}{m}\sum_{i=1}^{m}||\hat{y}^{(i)}-y^{(i)}||^2\Big]$$

$$= -m\log\sigma - \frac{m}{2}\log(2\pi) - \frac{1}{2\sigma^2}(m)\mathrm{MSE}_{\mathrm{train}}$$

where $\boldsymbol{\theta} = \boldsymbol{w}$ and $\hat{y}^{(i)}$ is the output of the linear regression on the i-th input $\boldsymbol{x}^{(i)}$ and m is the number of the training examples.

Comparing the log-likelihood with the mean squared error we immediately see that maximizing the log-likelihood with respect to $\boldsymbol{w}$ yields the same estimate of the parameters $\boldsymbol{w}$ as does minimizing the mean squared error. The two criteria have different values but the same location of the optimum. This justifies the use of the MSE as a maximum likelihood estimation procedure. □

Exercise 21. What is consistency?

Proof. In particular, we usually wish that, as the number of data points m in our dataset increases, our point estimates converge to the true value of the corresponding parameters. More formally, we would like that

$$\mathrm{plim}_{m\to\infty}\hat{\theta}_m = \theta \tag{8}$$

The symbol plim indicates convergence in probability, meaning that for any $\epsilon > 0$,

$P(|\hat{\theta}_m - \theta| > \epsilon) \to 0$ as $m \to \infty$. The condition described in equation 7 is known as **consistency**. It is sometimes referred to as weak consistency, with strong consistency referring to the **almost sure** convergence of $\hat{\theta}$ to θ. **Almost sure convergence** of a sequence of random variables $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ...$ to a value $\boldsymbol{x}$ occurs when $p(\lim_{m\to\infty} \mathbf{x}^{(m)} = \boldsymbol{x}) = 1$.

Consistency ensures that the bias induced by the estimator diminishes as the number of data examples grows. However, the reverse is not true—asymptotic unbiasedness does not imply consistency. □

Exercise 22. What is statistic efficiency?

Proof. Consistent estimators can differ in their **statistic efficiency**, meaning that one consistent estimator may obtain lower generalization error for a fixed number of samples m, or equivalently, may require fewer examples to obtain a fixed level of generalization error.

Statistical efficiency is typically studied in the parametric case (like in linear regression) where our goal is to estimate the value of a parameter (and assuming it is possible to identify the true parameter), not the value of a function. A way to measure how close we are to the true parameter is by the expected mean squared error, computing the squared difference between the estimated and true parameter values, where the expectation is over m training samples from the data generating distribution. That parametric mean squared error decreases as m increases, and for m large, the Cramér-Rao lower bound (Rao, 1945; Cramér, 1946) shows that no consistent estimator has a lower mean squared error than the maximum likelihood estimator. □

Exercise 23. Why is maximal likelihood the preferred estimator in ML?

Proof. For these reasons (**consistency** and **efficiency**), maximum likelihood is often considered the preferred estimator to use for machine learning. □

Exercise 24. Under what conditions do the maximal likelihood estimator guarantee consistency?

Proof. Under appropriate conditions, the maximum likelihood estimator has the property of consistency, meaning that as the number of training examples approaches infinity, the maximum likelihood estimate of a parameter converges to the true value of the parameter. These conditions are:

- The true distribution p_{data} must lie within the model family $p_{\text{model}}(\cdot; \boldsymbol{\theta})$. Otherwise, no estimator can recover p_{data}.
- The true distribution p_{data} must correspond to exactly one value of $\boldsymbol{\theta}$. Otherwise, maximum likelihood can recover the correct p_{data}, but will not be able to determine which value of $\boldsymbol{\theta}$ was used by the data generating processing.

□

Exercise 25. What is cross-entropy of loss?

Proof. Many authors use the term "cross-entropy" to identify specifically the negative log-likelihood of a Bernoulli or softmax distribution, but that is a misnomer. Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model. For example, mean squared error is the cross-entropy between the empirical distribution and a Gaussian model.

Note. It is important to point out that while the loss does not depend on the distribution between the incorrect classes (only the distribution between the correct class and the rest), the gradient of this loss function does effect the incorrect classes differently depending on how wrong they are

□