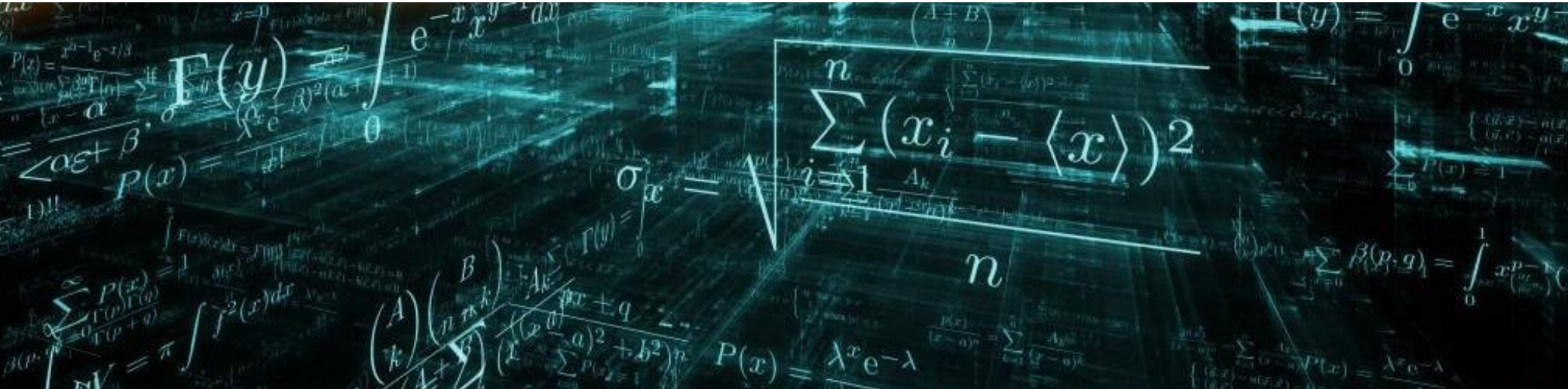
Universität Basel

Computer Science / 15731-01 / 2023

Multimedia Retrieval

Chapter 11: Machine Learning Methods

Dr. Roger Weber, roger.weber@gmail.com



11.1 Machine Learning Process

- In the upcoming chapters, we will explore a range of machine learning (ML) approaches, starting from simple methods like Naïve Bayes to more advanced techniques like transformers for language processing. While the focus of this course is on retrieval techniques, understanding and correctly applying ML methods are crucial. Mastering the retrieval problem requires a comprehension of the strengths and limitations of these underlying methods. Therefore, we cover specific ML methods when they are first needed. The final chapter of this course provides a comprehensive description of these methods, rather than cluttering them throughout all chapters.
- In this introductory section, our emphasis is on the machine learning process in general. We introduce key learning concepts and discuss potential pitfalls such as underfitting and overfitting, which can hinder the successful application of ML methods. In modern data science, the concept of MLOps has gained significance aiming to structure and organize the data preparation, training, deployment, and operations of ML functions more effectively.
- In his 1997 book, "Machine Learning," Mitchell defined the machine learning problem as follows:

  > 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 [Mitchell 1997]

  - Mitchell considers the task to be anything that we want the system to perform, ranging from simple classification tasks to complex scenarios like self-driving cars, as we will explore with examples in the following pages.
  - To evaluate system performance for a given task, Mitchell introduces the notion of a performance measure. This measure not only helps in improving the system's task performance but also allows us to compare two systems. In the next chapter "Evaluation“, we delve into benchmarking systems and comparing them with each other. On the next pages, we present two commonly used performance measures in machine learning.
  - Lastly, the term "experience" refers to the input provided to the system and its ability to utilize that input to enhance its performance. In supervised learning, we provide numerous examples to demonstrate how to perform a task correctly. In reinforcement learning, we offer feedback through a reward function, guiding the system towards better models. In unsupervised learning, we provide only data without labels or a reward function. The system must learn to describe the underlying distribution, either by clustering objects or detecting anomalies.

11.1.1 Tasks

The following provides a summary of the most common learning tasks that we will encounter throughout this course. It's important to note that there are many more learning tasks in various domains where machine learning is applied.

- **Classification**: The task involves mapping input features to a set of K categories. Typically, this means finding a function f that maps an M-dimensional vector $\boldsymbol{x}$ to a category represented by a discrete value y. Another variant of classification involves assigning a probability distribution $P(y)$ over all classes y, which sum up to 1 over all classes y. Applications of classification include object recognition in images, text categorization, spam filtering, handwriting and speech recognition, credit scoring, pattern recognition, and more.

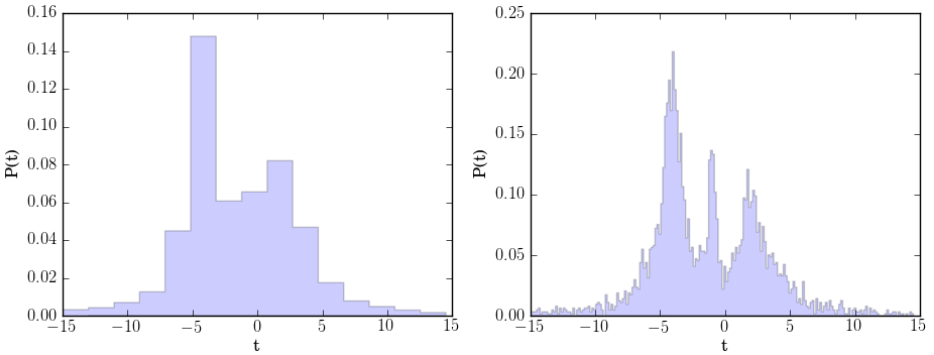
Sample	fixed acidity	volatile acidity	citric acid	pH	alcohol	quality
#1	8.5	0.28	0.56	3.3	10.5	7
#2	8.1	0.56	0.28	3.11	9.3	5
#3	7.4	0.59	0.08	3.38	9	4
#4	7.9	0.32	0.51	3.04	9.2	6
#5	8.9	0.22	0.48	3.39	9.4	6

- **Classification with missing input:** This task is similar to classification but allows for missing input values. Instead of a single function f, a set of functions is required to map different subsets of inputs to a category y (or a distribution $P(y)$). Alternatively, learning probability distributions over relevant features and marginalizing out the missing ones can be a better approach. All tasks have a generalization that accommodates missing inputs.

- **Regression**: The task involves predicting a numerical value based on the input features. The learning algorithm must find a function f that maps an M-dimensional vector $\boldsymbol{x}$ to a numeric value. Unlike classification, regression aims to produce a real number as the output and does not provide distribution functions over all possible values. Applications of regression include predictions/extrapolations to the future, statistical analysis, algorithmic trading, expected claim estimation in insurance, financial risk assessment, cost restrictions, budgeting, data mining, pricing (and its impact on sales), and correlation analysis.

- **Clustering** divides a set of inputs into groups. Unlike classification, the number of groups is not known in advance, and the machine learning algorithm must discover them. Since the output is unknown during training, this task is referred to as "unsupervised," while the previous tasks are labeled as "supervised" (we provided expected outputs). Applications of clustering include human genetic clustering, market segmentation (customer groups), social network analysis (communities), image segmentation, anomaly detection, and crime analysis.

- **Density estimation (probability mass function estimation)** entails constructing an estimate of an unknown probability density function based on the input features. In its simplest form, the algorithm learns a function $p: \mathbb{R}^M \rightarrow \mathbb{R}$ where $p(\boldsymbol{x})$ is interpreted as a probability density function (or a probability mass function for discrete $\boldsymbol{x}$). An example of basic density estimation is shown using histogram-based density estimation with different numbers of bins. Applications of density estimation include age estimation for countries, modeling complex patterns, feature extraction, and simplification of models.

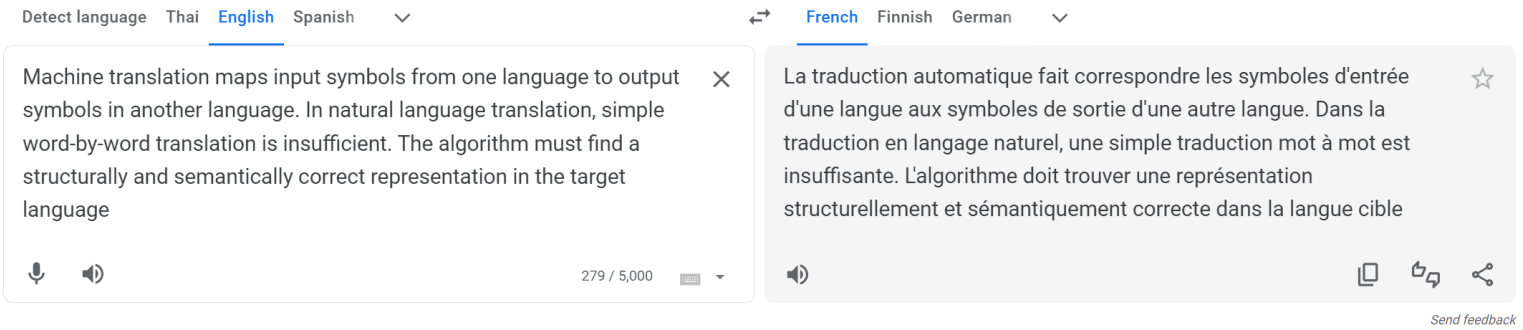


- **Imputation of missing** values involves replacing (estimating/guessing) missing data with substituted values. Given a new example $\boldsymbol{x} \in \mathbb{R}^M$ with some missing x_i, the algorithm must provide a prediction for the missing values. Applications of imputation of missing values include incomplete sensing data, demographics (incomplete personal data), medical analysis (incomplete or expensive test data), and signal restoration after data loss.

- **Anomaly detection** requires the algorithm to identify unusual, incorrect, or atypical events or data points. The output can be a simple binary flag (0 or 1, indicating an anomaly) or a probability of an anomaly. Supervised anomaly detection requires a training set with labels for "normal" (0) and "abnormal" (1) instances. Unsupervised anomaly detection involves the algorithm describing normal behavior (e.g., using density estimation) and automatically detecting outliers. Applications of anomaly detection include credit card fraud detection, intrusion detection (cybersecurity), elimination of outliers for statistical analysis, change detection, system health monitoring, event detection, and fault detection.

- **Generative AI** involves generating new data instances that resemble the existing training data. It captures underlying patterns to create samples with similar characteristics, such as images, text, music, and other content. Generative AI is an unsupervised learning task where algorithms generate output not tied to specific inputs. Applications include image synthesis, text generation, artistic style transfer, data augmentation, chatbots, computer-assisted coding, artefact creation for games, and text summarization.

- **Machine translation** maps input symbols from one language to output symbols in another language. In natural language translation, simple word-by-word translation is insufficient. The algorithm must find a structurally and semantically correct representation in the target language.
  - Example with Google Translate

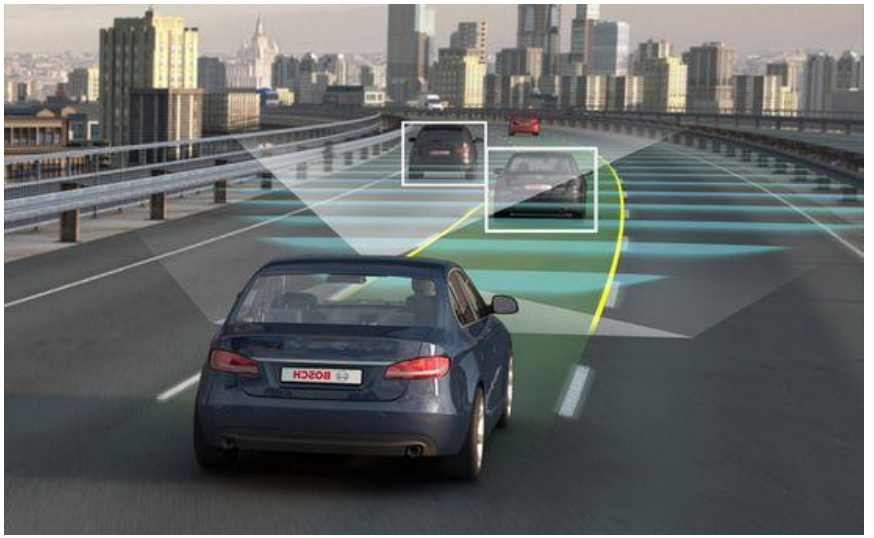


- **Transcription** involves converting unstructured data into a discrete, often textual form. Optical character recognition (OCR) and speech recognition are well-known transcription applications.

- **Dimensionality reduction** simplifies input vectors by transforming them into a lower-dimensional space. The output is interpreted as topics, concepts or embeddings, making it easier for the machine to find documents with similar topics. Dimensionality reduction is commonly used for data mining, latent semantic analysis, principal component analysis, statistical analysis, data reduction, and compression.

- **Reasoning** involves generating conclusions from knowledge using logical techniques like deduction and induction. Knowledge-based systems, including expert systems written in Prolog, have been used for the past 30 years. These systems used facts and rules to prove or disprove statements within a closed world. Modern approaches utilize machine learning for theorem proving or constraint solving. Cognitive reasoning and cognitive AI have recently improved the performance of chatbots and speech recognition.

- **Autonomous robots** employ reinforcement learning, where they adjust their behavior based on incentives and penalties from the environment. Autonomous driving has presented new challenges in reinforcement learning, particularly in machine ethics. Robots must make decisions in unforeseen scenarios where programmers cannot anticipate or hard-code the behavior. For instance, when faced with an inevitable collision with either an animal or a person, should the machine risk an evasive maneuver that endangers its passengers or accept the potential harm to the animal or person on the street?
  - While the field is relatively young, recent progress has been accelerated by deep learning techniques. Tesla claims that its autopilot is ten times safer than the average driver.
  - Laws and societal acceptance of robots are still in their early stages. Concerns regarding safety, privacy, and car hacking are raised, and insurance issues regarding who is liable for mistakes made by robots remain as further obstacles.



11.1.2 Performance

- Performance measures evaluate the effectiveness of a system in performing a task, with each task having its own interpretation of what "good" means. These measures can be categorized broadly whether they support supervised, unsupervised, and reinforcement learning.
- In **supervised learning**, the main tasks are regression and classification
  - In **regression tasks**, performance is measured using the mean squared error (MSE), which calculates the squared difference between the actual values (vector $\boldsymbol{y} \in \mathbb{R}^N$) and the predicted values (vector $\hat{\boldsymbol{y}} \in \mathbb{R}^N$).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 = \frac{1}{N}\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2$$

Note that the factor $1/N$ does not change the solution $\boldsymbol{\theta}^*$, hence we can omit it below

Regression models, represented by a function f with parameters $\boldsymbol{\theta}$, map M input values x_i to N output values y_i, where $f: \mathbb{R}^M \rightarrow \mathbb{R}^N$ and $\boldsymbol{\theta} \in \mathbb{R}^D$. The number of parameters, D, depends on the chosen function. The goal is to find the best solution $\boldsymbol{\theta}^*$ that minimizes the MSE, which involves finding the values of $\boldsymbol{\theta}$ where the gradient is zero.

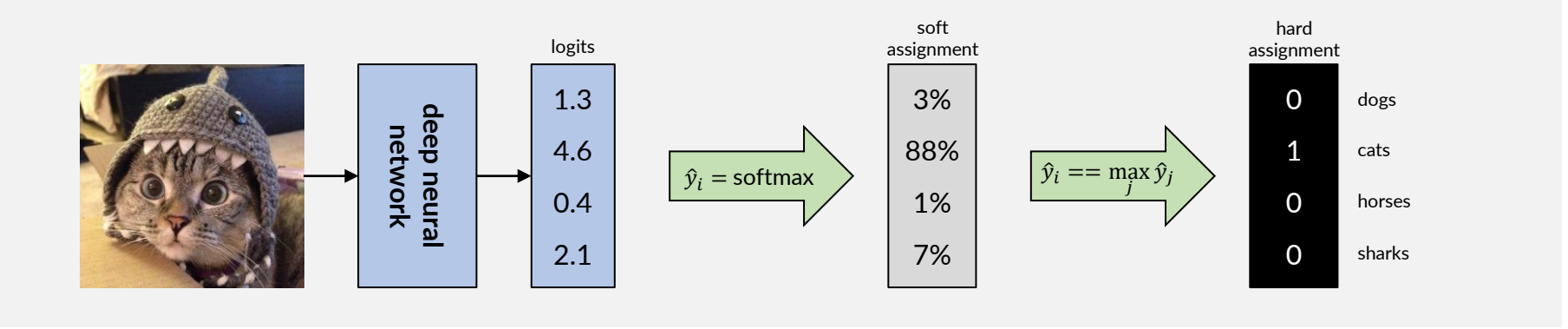
$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\|\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 \quad \Leftrightarrow \quad \nabla_{\boldsymbol{\theta}}\|\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2 = 0$$

For simple regression models, an exact solution can be found using calculus. In more complex cases, a numerical solution using gradient descent is often used, even if it only finds a local minimum (approximate result). The use of squared error simplifies the gradient calculations significantly. Backpropagation, employed in neural networks, utilizes a similar approach to train the weights in the network through stochastic gradient descent.

- **Classification tasks** can be categorized based on two dimensions:

  1) **Binary vs. multi-class**: Binary classification distinguishes between two output values, while multi-class deals with multiple output values. In the next chapter "Evaluation“, we will explore the assessment of binary classification in more detail, including precision, recall, accuracy, and confusion matrix. Tasks often involve classifying against hundreds of different labels, where the confusion matrix helps assess model accuracy.

  2) **Hard vs. soft assignments**: To understand this dimension, let's consider an example of detecting different animals in pictures (dogs, cats, horses, sharks). With hard assignments, a neural network can use an output bin for each animal where the highest value represents the predicted class (1) and the other bins are set to 0. With soft assignments, the network outputs probabilities or likelihoods for each class, forming a probability distribution across the classes. Soft assignments require mapping the network's logits (unnormalized output of the neural network) to meaningful probabilities that should align with the 0/1 values of the training set.



With hard assignments, we can compare the output of a neural network directly with the labels in the training set and assess the accuracy of the predictions (see next chapter on “Evaluation” for more details). In contrast, with soft assignments, we first need an effective method to map the logits (network outputs) to meaningful probabilities for each class. Secondly, we require performance measures that evaluate how well these probabilities align with the 0/1 values of the training set.

Consider the above example: the logits are converted into probabilities (the next page will introduce the softmax function). For instance, the bin representing "cats" receives an 88% probability, indicating that the picture likely contains a cat (hard assignment). However, a system that generates only a 79% likelihood for "cats" performs comparatively worse, even though it still results in the same hard assignment (“it’s a cat”).

With soft assignments, machine learning models produce K output values $o_k \in \mathbb{R}$, known as logits, where K is the number of classes. The range of values for o_k can vary based on the model and activation functions used in deep neural networks, making it challenging to control or predict. We have to convert these values into probabilities p_k, maintaining the relative order among o_k values and ensuring $\sum p_k = 1$ for a valid probability distribution across the K classes. There are many options for such a mapping, but the "softmax" function is a widely used approach.

$$\hat{y}_k = \frac{\exp(o_k)}{\sum_j \exp(o_j)}$$

If we do not state otherwise exp/log always refers to the natural base e. However, for our purpose, the base is irrelevant as it only scales the result but does not change order

In information theory, **cross-entropy** measures the accuracy of a model distribution p in matching the true distribution q over a set of events ε. For classification, the true distribution is often represented as a 'one-hot' vector $\boldsymbol{y}$, with only one component with value 1, and all others with value 0. The cross-entropy loss is then:

$$J(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_k y_k \log \hat{y}_k$$

softmax

$$J(\boldsymbol{y}, \boldsymbol{o}) = -\sum_k y_k \log \frac{\exp(o_k)}{\sum_j \exp(o_j)}$$

By plugging in the softmax definition, we arrive at the formula on the right, which can be further simplified to:

$$J(\boldsymbol{y}, \boldsymbol{o}) = \sum_k y_k \log\left(\sum_j \exp(o_j)\right) - \sum_k y_k o_k \quad = \log\left(\sum_j \exp(o_j)\right) - \sum_j y_j o_j$$

Similar to regression earlier, our goal is to find a model that minimizes this loss function using gradient descent search. To do this, we need to compute the partial derivatives of the loss function for each logit o_k, which simplifies to the difference between the softmax value $\hat{y}_k$ and the true label y_k.

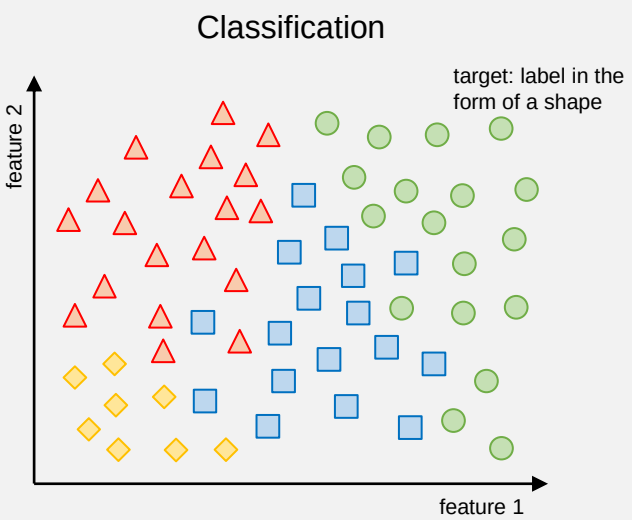
$$\frac{\partial J}{\partial o_k} = \frac{\partial}{\partial o_k}\left(\log\left(\sum_j \exp(o_j)\right) - \sum_j y_j o_j\right) = \frac{\exp(o_k)}{\sum_j \exp(o_j)} - y_k = \hat{y}_k - y_k$$

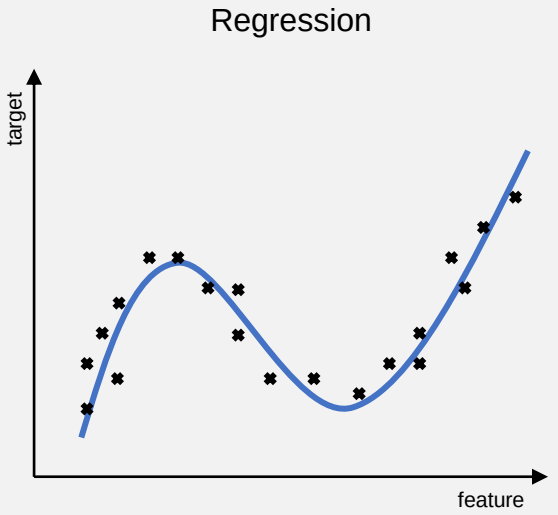
Recall the chain rule from calculus:
$F(x) = f(g(x))$
$F'(x) = f'(g(x)) \cdot g'(x)$

- **Unsupervised learning** encompasses tasks where no ground truth is available, making comparison methods from the previous pages inapplicable. Instead, we evaluate unsupervised tasks based on the model's effectiveness in tasks like identifying "good" clusters and detecting "true" outliers.
  - **Clustering** presents a key challenge in selecting the appropriate number of clusters and achieving well-defined cluster shapes. Having too many clusters may cause coherent regions to split, while too few clusters might result in insufficient distinction among data items, causing everything to blur together. Common methods to address this challenge include the elbow method, Silhouette Score, Davies-Bouldin Index, and Adjusted Rand Index (ARI).
  - **Dimensionality reduction** methods, like Principle Component Analysis (PCA) or auto-encoders, simplify and compress data representation. Performance measures evaluate the model's ability to reconstruct the original data from the reduced representation. Often, dimensionality reduction approximates results for main tasks, like vector search for text retrieval. This requires balancing faster execution with potential loss of quality due to approximation when compared to methods without compression.
- **Reinforcement learning** agents evaluate actions in an environment to maximize cumulative rewards. The tasks are broad and studied in various fields like game theory, control theory, autonomous driving, simulations, and genetic algorithms. Unlike supervised learning, reinforcement learning doesn't have known input/output correlations. The focus is on balancing exploration (of unknown situations) and exploitation (of current knowledge). The agent interacts with the environment in discrete time steps, observing potential rewards, choosing actions, and receiving rewards for transitions. The goal is to maximize cumulative rewards. A few examples:
  - Autonomous Driving: The reward function encourages safe and efficient driving. The agent (self-driving car) receives positive rewards for following traffic rules, staying on the road, and avoiding collisions. It receives negative rewards for breaking rules, driving too fast or erratically, and causing accidents.
  - Game Playing: The reward function focuses on gaining strategic advantages that lead to winning the game. In chess, this includes capturing opponent pieces, controlling strategic cells in the center of the board, and avoiding losing own pieces.
  - Financial Trading: The reward function balances trade profitability with risk exposure. If the agent (trade bot) focuses too much on highly profitable trades, it may deviate from the defined risk profile of the portfolio owner. Conversely, a low-risk trading strategy could result in missed opportunities. The trading strategy must withstand unpredicted market changes while aiming for profitable outcomes.

11.1.3 Experience

- **Supervised Learning** algorithms observe a dataset with features and corresponding target labels. The objective is to learn a generic rule that maps features to target labels, allowing the algorithm to predict outcomes for new data instances. The term "supervised" comes from the idea that the target labels are provided by an instructor or teacher. For example, in classification tasks, each data example consists of features and a corresponding label. The "teacher" provides instructions on how the features should be mapped to labels, and the algorithm learns this mapping rule.
  - In the next section, the task is not merely to "learn" the mappings in the training set, but to create a generic model that performs well for new data. The teacher typically provides both the labels and a performance measure, assessing how effectively the generalization worked compared to exact replication of the training data
  - Generating labels for training sets is a laborious and expensive task, similar to obtaining metadata for data objects. New approaches aim to avoid the need for explicit labeling while still gaining the advantages of supervised learning. An example is Generative Adversarial Networks (GANs), where a generator creates fake data (e.g., images) and a competing discriminator evaluates whether a given sample is real (drawn from a given data set) or generated. The generator attempts to deceive the discriminator by maximizing the number of “real” outcomes, while the discriminator optimizes its model to better distinguish between real drawn from the given set and generated data.





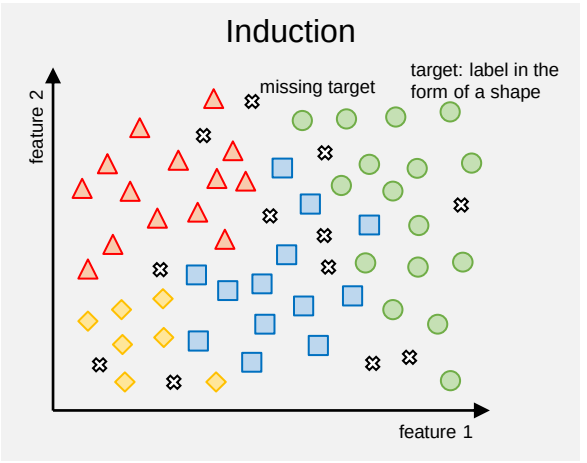
- **Semi-Supervised Learning** is a variation of supervised learning. The algorithm receives both features and targets, but some of features or label may be missing in the training data (incomplete observation). Depending on the task, the algorithm needs to either fill in the missing features or predict targets for new data sets.
  - **Missing targets** occur when some objects in the training set lack targets (or labels) due to the expensive or labor-intensive labeling process. For instance, in credit card fraud detection, only a small subset of transactions is labeled as "fraud" or "no fraud" based on investigations. The vast majority remains unlabeled. Algorithms dealing with such data make assumptions to learn effectively.
    1) Smoothness: points in close proximity share the same label. Hence, we assume that the distribution function is smooth and continuous.
    2) Cluster: data tends to form clusters and all objects in the same cluster share the same label
    3) Manifold: often, features are high-dimensional but the data is more likely to lie on a low dimensional manifold

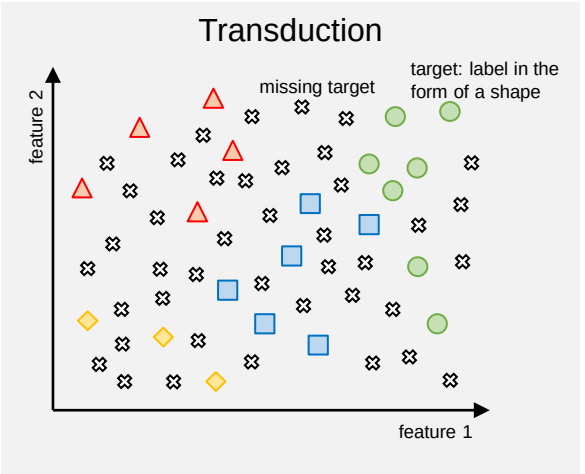
  - **Induction**: When only a few labels are missing, a useful approach is to learn the distribution from the labeled data using supervised learning. We can then use this knowledge to predict the missing labels. However, this method becomes ineffective when a large number of objects lack labels, as the training set becomes inadequate to capture the true label distribution. Consequently, this approach disregards a significant portion of the data, leading to information loss.

  - **Transduction:** To utilize all data points, transductive algorithms identify clusters in the dataset and assign the same label to all objects within each cluster. One approach is the partitioning transduction method:

    1. Start with a single cluster containing all objects.
    2. While a cluster has two objects with different labels, partition the cluster to resolve the conflict.
    3. Assign the same label to all objects within each cluster.

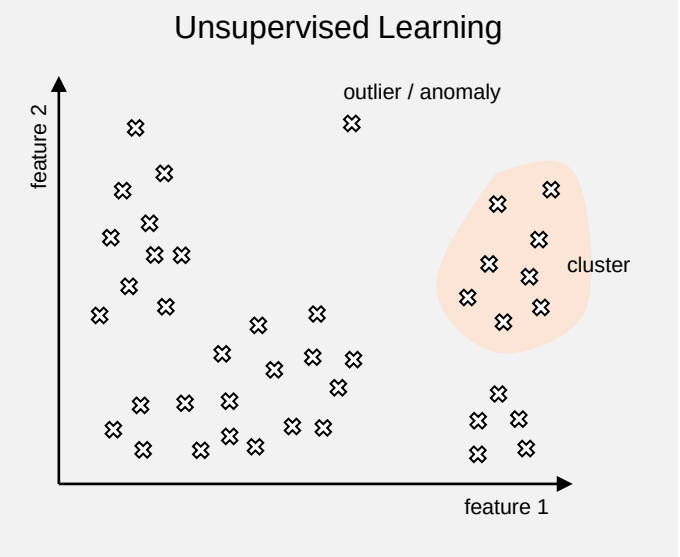
    Various other variants exist for developing these clusters.





- **Missing features:** The training set has complete targets, but some items lack some of the features. For newly presented data, potentially with missing features, the algorithm must predict the target. A good example is disease prediction where the target ("healthy", "has disease") must be predicted from a set of test results. Laboratory tests are expensive and thus not all features (test results) are available.
  - Naïve Bayes is a simple technique for building classifiers using conditional probabilities. With K classes C_k and M features x_i, the best class k^* is determined by $k^* = \underset{k}{\operatorname{argmax}} P(C_k) \prod_i P(x_i|C_k)$. The probabilities $P(C_k)$ and $P(x_i|C_k)$ are learned from the training data (ignoring missing features x_i). When predicting the class for a new object with missing features, we simply ignore those features in the Naïve Bayes optimization.
  - If we have learned the distribution function for all features, we can simply "integrate" or "average" over the missing features. This means assuming that the missing features follow the distribution of the training set and approximating them with an expected value.

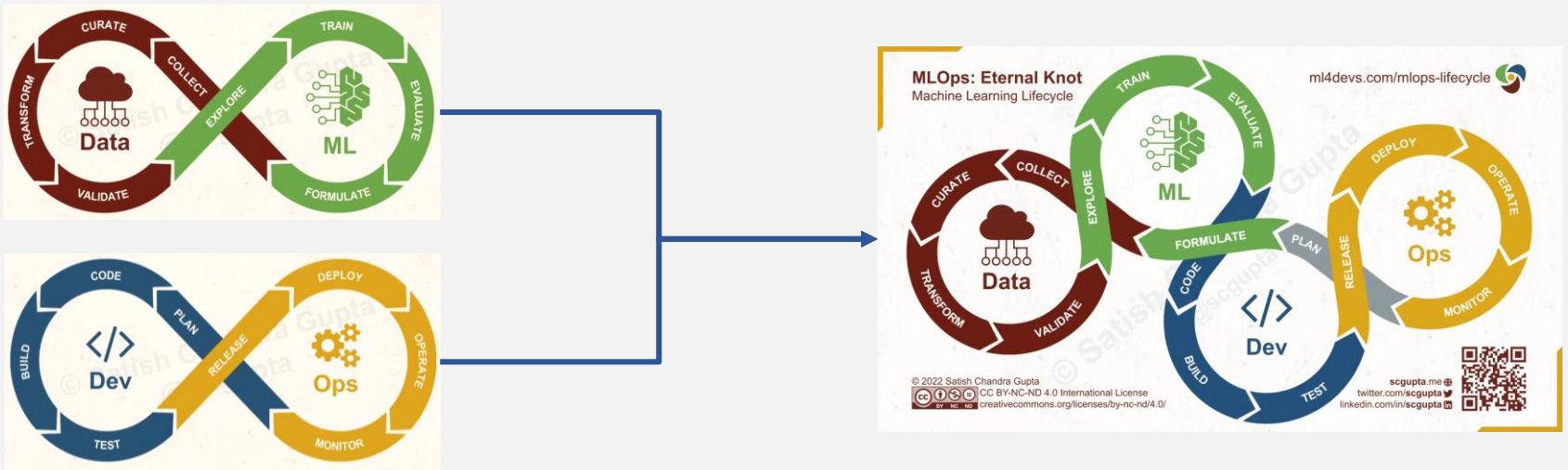
- **Unsupervised Learning** algorithms analyze a data set without targets and derive a function that captures the underlying structure or distribution of the data. The goal is to discover meaningful patterns and gain insights into the data's structure. Unlike supervised learning, there is no instructor or teacher providing targets or evaluating the model's performance. The algorithm must learn and make discoveries independently.
  - Clustering: Identifying groups of objects that are similar based on a distance function. The number of clusters is often unknown.
  - Outlier/Anomaly Detection: Learning the "normal" behavior and identifying outliers that significantly deviate from the rest. The training data may also contain outliers.
  - Density Function: Describing the data with an appropriate density function. Gaussian approximation is a method, while more complex ones optimize for the best fit among different distribution functions.
  - Dimensionality Reduction: Extracting core concepts from high-dimensional features using techniques like Principal Component Analysis for a simpler yet accurate view of the data.
  - Self-Organizing Maps (SOM): Mapping high-dimensional data to 2-dimensional presentation using a competitive learning approach.



11.1.4 Training

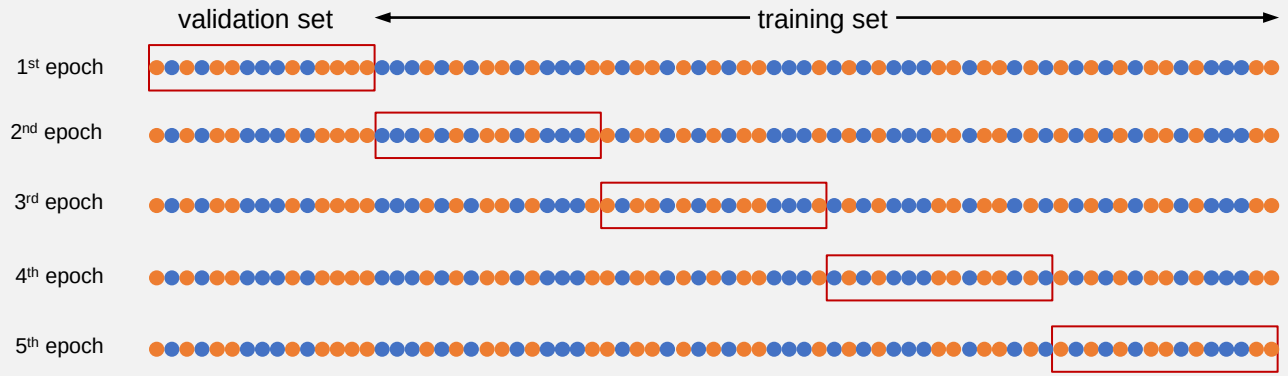
- Training is a vital step in the broader "Model Development" process, which is now integrated with DevOps to form a comprehensive machine learning lifecycle known as MLOps. This process encompasses four main cycles:
  - The **data** cycle collects, curates, and generates datasets for learning. For example, this involves gathering images, filtering, and labeling them for an object recognition task, and creating training, test, and validation subsets.
  - The **machine learning** cycle selects an appropriate model, optimizes hyperparameters, trains the model, and evaluates its performance to choose the best option. In the image recognition example, we might use a ResNet architecture, adjust the number of layers and other parameters to achieve the highest accuracy on the test set.
  - The **development** cycle integrates the model into the application architecture or business process and conducts end-to-end testing of the (business) use case. For the image example, this could mean embedding the trained model in a container with an API accessible by other parts of the architecture.
  - The **operations** cycle extends the application operations to monitor the performance of the machine learning model and trigger feature requests if its performance falls below the desired level. In the image example, this might include a feedback mechanism where end-users can report issues with objects not properly identified.

source: ml4devs.com, Satish Chandra Gupta



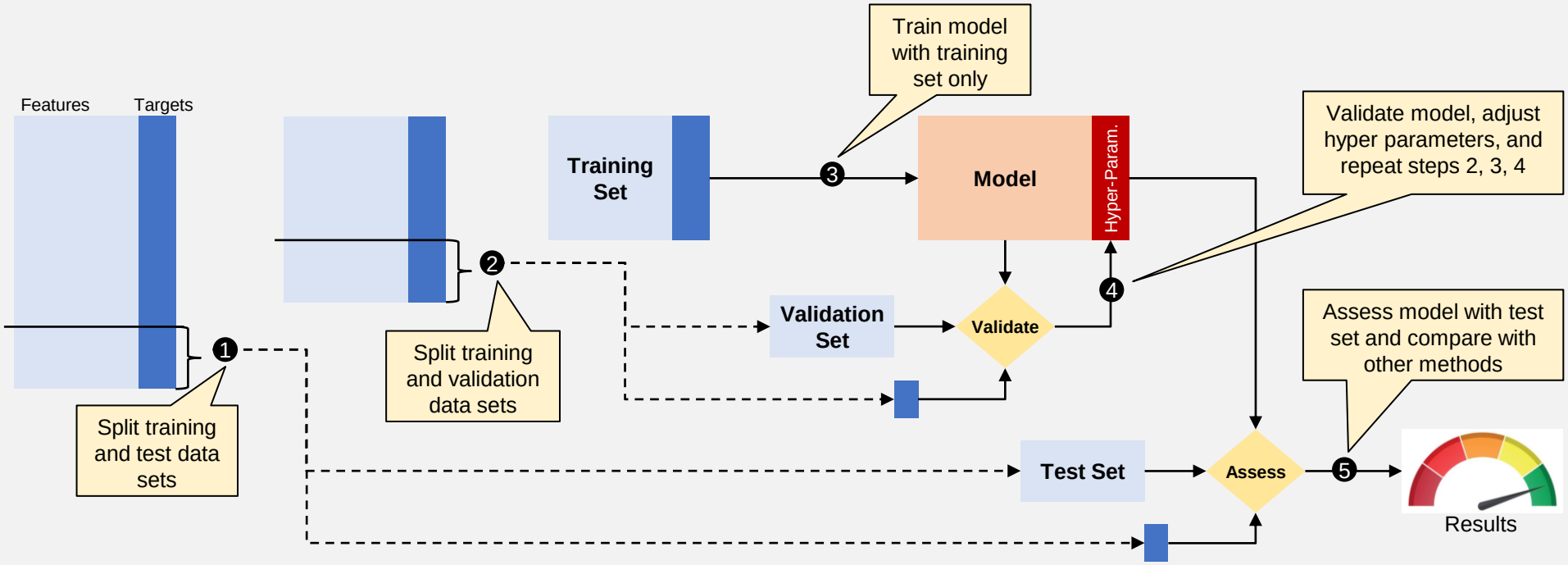
- While we aim for good performance on the training data, our primary goal is to ensure that the model excels with new, unseen data. Memorizing the training data is undesirable since it leads to poor performance on unfamiliar data. Instead, we seek a model that not only performs well on the training data but also generalizes its learnings, allowing it to achieve equally impressive results on new, previously unseen data.
- Earlier in this chapter, we introduced cross-entropy as a loss function for classification tasks. During model training, our goal is to minimize this loss. However, the best model is not solely determined by the lowest loss on the training data. It is the model that achieves the highest accuracy on new, unseen data. To estimate this accuracy, we split the data into two parts: roughly 80% for training, and the remaining for final evaluation without being used in training. In machine learning competitions, participants do not event have access to the test data. They can optimize their models with training data, but final evaluation is with unknown new data to identify the best-performing models.
- Most models have hyperparameters, like ResNet in PyTorch with varying layer numbers. Self-developed models may have other hyperparameters like activation functions, optimization algorithms, regularization features, and different number representations (int16, fp16, fp32). This adds an extra optimization loop to the training process to find the best hyperparameter set.
  - Once more, our goal is to find the best hyperparameters that perform well on future predictions. While minimizing the training loss is essential, we do not want the model to memorize it; instead, it should learn to generalize based on the chosen hyperparameters. We select the hyperparameters that enables the model to generalize most effectively.
  - However, using the test data for this purpose would "leak" information into the training process that compromises the accuracy of future predictions. Even model developers should refrain from inspecting the test data to diagnose model failures, as it could lead to over-optimization for the training and test sets, rather than improving the model's ability to generalize.
  - Instead, we further divide the training data: approximately 80% for model training, and the remaining portion for validating the chosen hyperparameters. After finding a good set of hyperparameters, we can then use the test set to assess the final model performance.
- To avoid bias towards the training set, modify the validation set in each iteration when searching for optimal hyperparameters. Remember, we want to find models that generalize well and not models that memorize training and/or validation data

- In cases where we have limited training examples due to the double splitting of the data, we can use the **k-fold cross-validation** approach. This method not only ensures effective model training but also prevents overfitting by presenting different data to the model in each epoch.
  - As before, it is crucial to keep the test data separate from the training data. Never use any part of the test data during the training process, even if you are running low on examples.
  - However, we vary the validation set in each epoch. An epoch is a full iteration over all the training data, and during this iteration, we evaluate the model on the validation set. For each epoch, we split the training and validation data differently using the following approach illustrated below.
    - The training data, after splitting the test data, is divided into k folds. For smaller datasets, using a larger value for k can lead to better results, with typical values ranging between 5 and 10.
    - In each epoch, one fold is used for validation, while the other folds are used for training. As we iterate over epochs, different folds are used for training, reducing the risk of bias towards the training data. However, this increases the variance of our accuracy estimates.
    - k is a hyperparameter itself, and you have to optimize it for each learning process again. Try varying k to observe which value yields the best validation results (do not use the test data for this, as discussed before).
    - Sometimes datasets are imbalanced, with some labels being rare while others are frequent. To improve the stability of the learning process, we need to ensure that the folds contain a representative number of each class.
  - If more epochs are required, we can restart with the first fold and continue iterating until the model's performance has converged or we decide to stop the current run and begin again with new hyperparameters.



The training process involves the following steps (see picture below)

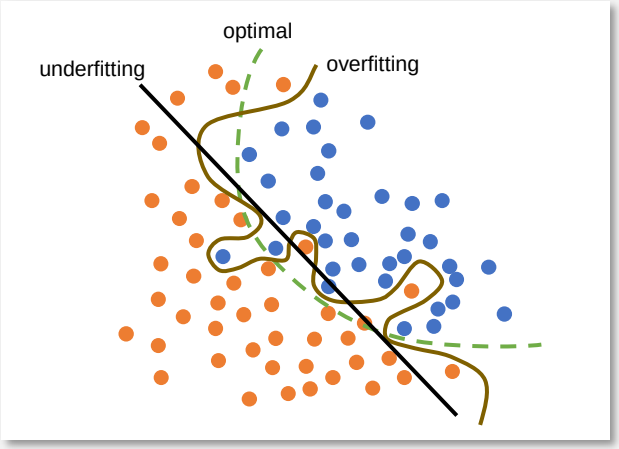
1. Split the data into training and test sets. Keep about 20% test data for final assessment, but avoid using or looking at the test data during training. In competitions, the test data is usually not shared with contestants
2. Split the training data again into training and validation sets. Allocate about 20% for validating hyperparameters. Consider k-fold cross-validation for limited data or to prevent bias towards the training data
3. Train the model with the training set and iterate over epochs until performance converges or it has become evident that the current hyperparameters are sub-optimal
4. Adjust hyperparameters based on validation results. Retrain the model with the optimal set of hyperparameters using more epochs if needed
5. Finally, evaluate the model's performance using the test data and compare it with other approaches. If you iterate the training process, use a different test data set to avoid bias. In competitions, the contestants only receive the final assessment results but cannot inspect where and why the model has failed

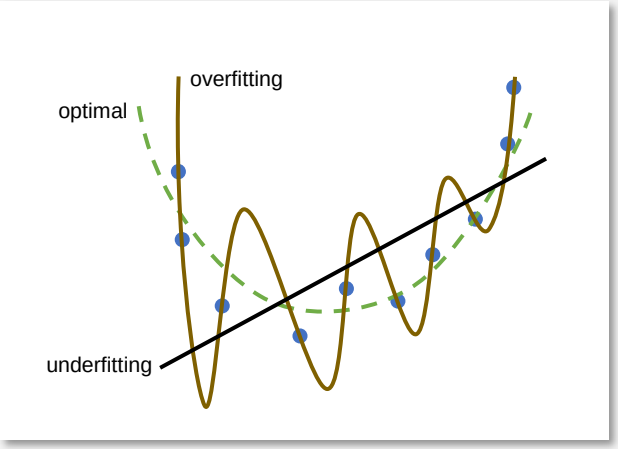


11.1.5 Over- and Underfitting

- During the training process, we mentioned the bias towards the training set without fully explaining its meaning and how to detect it. A fundamental concept in machine learning deals with the balance between two types of errors that can occur during the training process: bias and variance
  - Bias is the error introduced by using an overly simplistic model to approximate the data. A model with high bias is not able to capture the essential patterns and relationships in the data
  - Variance is the sensitivity of the model to small changes in the data, often caused by a too complex model. A model with high variance has good performance on the training data but poor performance on new, unseen data
  - A tradeoff is necessary because reducing bias can increase variance, and vice versa. The goal is to strike a balance by finding a model that generalizes well to unseen data. Thus, the goal is to minimize the sum of bias and variance
- Model complexity is a key factor influencing bias and variance. Different machine learning structures have different facets of complexity. For instance, a linear regression model is simpler with fewer parameters to adjust $y = a \cdot x + b$, while a polynomial model has more parameters, giving it greater adaptability to different datasets. Capacity, in this context, refers to a model's ability to adjust itself through structural changes and model parameters.
- But what is the right complexity: if the model is too simple (low complexity), we risk a high bias; if the model too complex, we risk a high variance like memorizing the training data rather than generalizing the observed patterns
  - An example for a too simplistic model: “if the sun is out, it is warm”
  - An example for an overly complex model: “if the sun is out and it is a summer month and you are on the north side or it is a winter month and you are on the south side or you are equatorial or you are in a dessert and it is not an ice dessert and it is not cloudy or raining or snowing and there is not a strong wind and there is not a sun eclipse and there is not a volcano eruption and you are not in the water or in a cave or in the shadows or in a house with air conditioning or in a car with air conditioning or in a freezer … then it is warm”
- Our brain is excellent in finding the right level of abstraction. Consider the following examples:
  - “birds can fly” but wait, not all birds can fly → we us a simple model and learn the exceptions
  - “describe what makes a chair a chair” → no simple model, so we employ abstract concepts (“you can sit on it”)
  - “horse” → narrow variety of forms and what is accepted as a horse (e.g., donkey, zebra, giraffe)
  - “dog” → wide variety of forms that count as dogs yet we recognize them immediately

- **Overfitting and underfitting** are common issues in machine learning. Overfitting happens when the model becomes overly complex, trying to match the training data too closely. This often occurs when the model has too many parameters relative to the available training data, leading to poor predictive performance when applied to new data. Underfitting, on the other hand, occurs when the model is too simplistic to capture the underlying data trends. For example, fitting a linear model to a non-linear data distribution will result in high training error and inadequate predictive performance.
  - As shown below, **overfitting** occurs when the model is optimized for the training data using too many parameters. Such a model may display a small training loss, indicating good adaptation to the training data, but it fails to predict new data points effectively
  - On the other hand, **underfitting** exhibits large errors on the training data and poor prediction performance for new data points. It clearly fails to capture the true essence of the distribution
  - To control overfitting and underfitting, we can adjust the model's capacity. The optimal capacity is achieved when the model shows small errors on both the training set and the validation set
- To recognize overfitting and underfitting during training, we use a loss function like mean squared error or cross-entropy loss. Underfitting is indicated by high losses on both the training and validation sets, while overfitting is characterized by small losses on the training data but significantly higher losses on the validation set. Note that the loss on validation sets is typically higher, but a large gap is a key sign of overfitting.

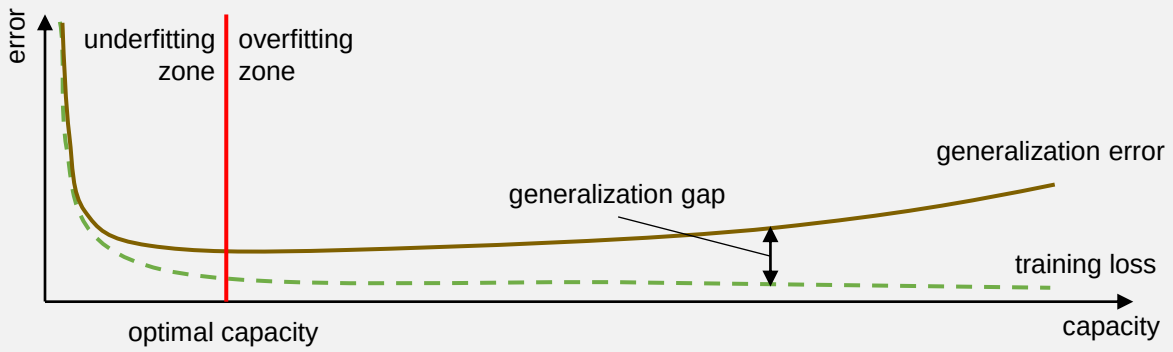




- How can we optimize the model's capacity to strike a balance between bias and variance?
  - **Occam's razor** is an intuitive heuristic, first stated by William of Ockham (c. 1287-1347). It has been further refined, especially in the 20th century, for statistical learning purposes.

    Numquam ponenda est pluralitas sine necessitate (Plurality must never be posited without necessity

    In a modern interpretation, Occam's razor suggests that when multiple hypotheses explain observations equally well, we should prefer the simplest one. This means we aim for models with low complexity and only increase it if needed to perform well for the specific task. We can also find examples for Occam's razor in physics, such as Newton's laws, which offer a simplified yet effective approximation of the real-world in many situations.
  - Simpler models are better at generalizing, but we must avoid models that are overly simplistic and have high training loss. As we increase the model's capacity, the training loss decreases. However, if the capacity becomes too high, the model loses its ability to generalize (e.g., memorizing data), and the gap between training loss and validation loss widens. The illustration below shows the underfitting and overfitting zones, divided by the optimal model capacity that reduces training loss and enables good generalization at the same time.



  - Modern language models use deep learning architectures with billions of parameters. Handling such a large number of parameters presents new challenges as we must prevent the model to memorize and instead encourage it to generalize. Later in the course, we will explore different regularization techniques, which aim to penalize complex models and strike a balance between bias and tradeoff. For instance, L2 regularization in deep learning introduces a penalty in the loss function to discourage the utilization of large parameter values. This motivates the model to use fewer connections in the network and prevents overfitting

11.2 Literature

- Tom M. Mitchell, **Machine Learning**, 1997, McGraw-Hill Science/Engineering/Math, ISBN: 0070428077
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, **Deep Learning**, 2016, MIT Press, online version: https://www.deeplearningbook.org/
- Various authors, **Dive into Deep Learning**, 2023, to be published at Cambridge University Press, online version: https://d2l.ai/
- **MLOps: Machine Learning Life Cycle**, 2022, https://www.ml4devs.com/articles/mlops-machine-learning-life-cycle/