Supervised Learning: Find a mapping f from input features x to target outputs t using **labelled** training data (input-output pairs).
Unsupervised Learning: Find interesting patterns in the data, e.g., find clusters, where **no target labels are given** (only inputs are given).
Reinforcement Learning: Find a sequence of actions that produce a high reward signal in an environment.

Training Set: Used to learn the (*parameters* of) the mapping f from input features to the predicted output
Validation Set: Used th choose *hyperparameters* or model settings that are not provided by by the learning algorithm
Test Set: Used to estimate how well the model will *generalize* to new data and how well the model will perform when deployed

---

Training:
k-Nearest Neighbours: No training necessary; just store the data
Decision Trees: Greddy method: find the optimum split at each node, where "optimum" is determined by a loss, e.g., information gain
Linear Regression and Classification: Find weights w that minimize a cost function, e.g., mean square error or average cross entropy loss across the training set

---

$x_j^{(i)}$ represents the j-th feature of the i-th training example
Hypothesis (model, predictor) is a function f that maps input x to output prediction y
Voroni diagram partitions space into regions closest to each data
kNN: k too small may overfit (sensitive to noise), k too large would underfit (fail to capture regularities)
$k < \sqrt{n}$, choose k with highest validation accuracy
We use test data to estimate generalization error 60% training, 20% validation and test
We normalize each dimension to zero mean and unit variance
Curse of dimensionality: data not dense in high dimensions, and hard to calculate distances. Most points are approximately equally distanced, squared Euclidean distance becomes dominated by the average behavior, making distances converge
Overfitting: High training accuracy, poor test accuracy; model too complex, small data set, noise in data; regularization, better hyperparameters, better model
Underfitting: Low training and test accuracy; model too simple, bad hyperparameters, bad model; adjust hyperparameters, regularization, normalization

---

Decision Tree Output: Most common value for discrete classification, or mean value for regression
Use greedy heustic to construct decision tree based on (highest) information gain
More certain outcome has lower entropy, i.e. not uniformly distributed
Entropy characterizes uncertainty of random variable
Conditional Entrypy characterizres uncertainty of random variable given another random variable
Information Gain measures informativeness of variable
Process: Pick a feature, compare all possible splits, choose the one with highest IG, repeat recursively
We may minimize squared error instead: $SE = \sum_{i=1}^N (f(x^{(i)}) - t^{(i)})^2$

A model is a set of assumptions (restrictions) about the structure of f
The model or architecture restricts the hypothesis f space
Goal of Linear Regression: find w (weights) and b (bias) minimizing $\mathcal{E}(w, b) = \mathcal{E}(\mathbf{w})$
Let X be matrix with $N \times (D+1)$ shape with each $x^{(i)}$, then $\mathbf{y} = \mathbf{X}\mathbf{w}$
Gradiant Descent is more general than matrix inversion for finding $\mathbf{w}$
Each GD update costs $O(ND)$, $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}) = \mathbf{w} - \frac{\alpha}{N} \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t})$, or $w_i \leftarrow w_i - \alpha \frac{\partial \mathcal{E}}{\partial w_i}$, where initially $\alpha = 0.002, w_i = b = 0$
If α too small, training is slow, if α large, diverges

We may generalize linear regression by mapping input feature to another space, by feature mapping (or basis expansion): $\psi(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^d$, and treat $\mathbb{R}^d$ as input space, e.g. polynomial $\psi(x) = (1, x, ..., x^M)^T$, then $y = \psi(x)^T \mathbf{w}$
Adjust the degree of polynomial to avoid underfitting or overfitting
regularizer or **penalty** is function that qualifies how much we prefer one hypothesis (penalize large weights)
Both polynomial degree M and λ are hyperparameters
λ large cause underfit, λ small cause overfit
We want each feature's values to be indicators or orderable
ML Algorithm Design: 1. Choose model; 2. Define loss function; 3. Choose regularizer; 4. Fit model by minimizing regularized cost, using **optimization algorithm**.
$t = 1$ for positive example, $t = 0$ or -1 for negative example
For **Linear Binary Classification Model**, let $z = \mathbf{w}^T x$, then $y = 1$ if $z \geq 0$, else $y = 0$, decision boundary being hyperplane $z = 0$
Given small training set, go over each data point to find restrictions of w_i, if resion in weight space satisfying all constraints is non-empty, then this **feasible region** makes the problem (in)-**festible** and the training set **linearly separable**
Use square error loss function would cause correctly classified points with high loss; use indicator loss function would cause gradiant be zero e.w.; use sigmoid plus square loss won't update much for very wrongly classified example, i.e. poor gradient signal
We use sigmoid to smooth the output of each choice of $\mathbf{w}$ between 0 and 1, then use cross entropy loss function to penalize the model for being wrong

---

$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$, same as linear regression
hypothesis space $\mathcal{H}$ set of all possible h a model can represent, i.e. all linear functions for linear regression
Members of $\mathcal{H}$ with algorithm's preference determine **inductive bias**
Algorithm is **parametric** if $\mathcal{H}$ is defined by a set of parameters (linear and logistic regression, neutral network; counter-example kNN)
Previously we used **full batch gradient descent** of $\frac{\partial \mathcal{E}}{\partial w}$, we may estimate $\frac{\partial \mathcal{E}}{\partial w}$ using subset of data if N too large, via **stochastic gradient descent**: computing average of a small number of examples called **mini-batch** with a **batch size**.
inner loop is **iteration** (each mini-batch), outer is **epoch** (one update)
num_iter = num_epoch $*$ N/batch_size

Accuracy: $\frac{1}{N} \sum_{i=1}^N \mathbb{I}[t^{(i)} = y^{(i)}]$
Error: $1 -$ Accuracy

Euclidean: $\left|\left|x^{(a)} - x^{(b)}\right|\right|_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$
Cosine Similarity: $\text{cosine}(x^{(a)}, x^{(b)}) = \frac{x^{(a)} \cdot x^{(b)}}{\left|\left|x^{(a)}\right|\right|_2 \left|\left|x^{(b)}\right|\right|_2}$

Mean: $\bar{x} = \frac{1}{N} \sum_{i=1}^N x^{(i)} = \mu$
Variance: $\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x^{(i)} - \bar{x})^2$
Standard Deviation: $\sigma = \sqrt{\sigma^2}$
Normalization: $\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$

Entropy:

$$H(X) = -\mathbb{E}_{X \sim p}[\log_2 p(X)]$$
$$= -\sum_{x \in X} p(x) \log_2 p(x)$$
$$H(X, Y) = -\sum_{\substack{x \in X \\ y \in Y}} p(x, y) \log_2 p($$
$$H(Y|x) = -\sum_{y \in Y} p(y|x) \log_2 p($$
$$H(Y|X) = \sum_{x \in X} p(x) H(Y|x)$$
$$= -\sum_{\substack{x \in X \\ y \in Y}} p(x, y) \log_2 p($$
$$H(X, Y) = H(Y, X) = H(X|Y) +$$
$$H \geq 0, H(Y|Y) = 0, H(Y|X) \leq$$
$$H(Y|X) = H(Y) \text{ if } X \text{ and } Y \text{ ind.}$$

Information Gain: $IG(Y|X) = H(Y) - H(Y|X)$

Linear Model: $y = f(x) = \sum_{j=1}^D w_j x_j + b = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$
where $\tilde{\mathbf{w}}_1 = b, \tilde{\mathbf{w}}_i = w_{i-1}, \tilde{\mathbf{x}}_1 = 1, \tilde{\mathbf{x}}_i = x_{i-1}$
Squared error loss function: $\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$
$y - t$ is called residual
Cost (average loss) function:

$$\mathcal{E}(w, b) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y^{(i)}, t^{(i)})$$
$$= \frac{1}{2N} (\mathbf{y} - \mathbf{t})^T (\mathbf{y} - \mathbf{t})$$
$$= \frac{1}{2N} ||\mathbf{X}\mathbf{w} - \mathbf{t}||_2^2$$

$\frac{\partial \mathcal{E}}{\partial w} = \frac{1}{N} \sum_{j=1}^N (y^{(i)} - t^{(i)}) x^{(i)}$
$\frac{\partial \mathcal{E}}{\partial b} = \frac{1}{N} \sum_{j=1}^N (y^{(i)} - t^{(i)})$
$\frac{\partial \mathcal{L}}{\partial w_j} = (y - t) x_j$
$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = (y - t)\mathbf{x}$

$$\nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{($$
$$= \frac{1}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{t})$$

$\nabla_{\mathbf{w}}\mathcal{E}(\mathbf{w}) = 0$: $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$

L^2 **Regularizer**:
$\mathcal{R}(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||_2^2 = \frac{1}{2}\sum_{j=1}^{D} w_j^2$
Regularized cost function:
$\mathcal{E}_{\text{reg}}(\mathbf{w}) = \mathcal{E}(\mathbf{w}) + \lambda\mathcal{R}(\mathbf{w})$

Logistic / Sigmoid function:
$y = \sigma(z) = \frac{1}{1+e^{-z}}$
Both sigmoid and square loss:
$\frac{\partial\mathcal{L}}{\partial w_j} = (y-t)y(1-y)x_j$
Cross Entropy Loss:
$\mathcal{L}_{\text{CE}}(y,t)$

$$= \begin{cases} -\log(y) & \text{if } t = 1 \\ -\log(1-y) & \text{if } t = 0 \end{cases}$$

$= -t\log(y) - (1-t)\log(1-y)$
$\frac{\partial\mathcal{L}_{\text{CE}}}{\partial w_j} = (y-t)x_j$

3rd column cont., batch size too large = expensive, too small = noisy
center and normalize to avoid ravines

Graph of kNN: Draw tangent line at each midpoint, connect regions with same class
Graph of Decision Tree: Draw axis-aligned lines to split regions
IG of Decision Tree: First calculate the entropy $H(Y)$ with probability (percentages) of each class, similarly for $H(Y|\text{right}), H(Y|\text{left})$, then $IG = H(Y) - p(\text{right})H(Y|\text{right}) - p(\text{left})H(Y|\text{left})$
Shape: # of rows $\times$ # of columns

Linear Regression: Square error loss function, with average loss cost function
Logistic Regression: Sigmoid function, with cross entropy loss function
Decision Tree: Greedy method, with information gain loss function
kNN: No training, with Euclidean distance for lower dimension D