Data & Pre-Processing

Numerical Data: Discrete / Continuous
Categorical Data: Nominal / Ordinal
Data Analysis: min, max, quantiles, median, mean, stddev, outliers, correlation matrix (dependency of features each other)
Pre-Processing: delete/impute, discretize (grouping e.g. by age), scaling; re-label (small→0), one-hot-encode, drop unimportant feat.
Standardization (z-score) $x'_j = \frac{x_j - \bar{x}_j}{\sigma}$, where $\bar{x}_j = \frac{1}{|X|}\sum x_j$
Minkowski Distance: $d(a,b) = \left(\sum_{i=1}^{m} |a_i - b_i|^p\right)^{1/p}$
Levenshtein (Edit) Distance: Min number of edits required.

Core Concepts of Machine Learning

Instance space X, Label space $\mathcal{Y}$. Dataset $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times \mathcal{Y})^m$. Model $h : X \to \mathcal{Y}$. Loss function $\mathcal{L}$. Data space $\mathcal{D}$.
Supervised Learning: Minimize **Empirical risk** $R_S = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(h, (x_i, y_i))$. In other words $h^* \in \text{argmin}_{h \in \mathcal{H}}(R_S(h))$.
Classification: Is goat or sheep? Loss $= 0$ if $x = y$ else Loss $= 1$.
Regression: How does it weigh? Loss $= (h(x) - y)^2$.
True risk: $R_{\mathcal{D}} = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(h, (x, y))]$
Gradient descent: Walk down slope, η learning rate, ∇f gradient.
Underfitting: High bias, low variance.
Overfitting: Low bias, high variance.
Regularization: Add penalty to loss for high complexity, $L1$ (when few features important) and $L2$ (when all features important).

Basic Algorithms I

Gradient $\nabla\mathcal{L}$: $\nabla\mathcal{L}(w_1, \cdots, w_n) = \begin{bmatrix} \mathcal{L}' \text{ by } w_1(w_1, \cdots, w_n) \\ \vdots \\ \mathcal{L}' \text{ by } w_n(w_1, \cdots, w_n) \end{bmatrix}$
Linear model: $h(x) = w_1 \cdot x + w_0$. Optimal (closed form) solution:

$$w_1 = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{m}(x_i - \bar{x})^2}, \quad w_0 = \bar{y} - w_1\bar{x}$$

Polynomial model: $h(x) = \sum_{k=0}^{p} w_k x^k$. To fit use

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}, \text{ where } \mathbf{X} = \begin{bmatrix} \mathbf{1}^T \\ (\mathbf{x}^1)^T \\ \vdots \\ (\mathbf{x}^p)^T \end{bmatrix}, \text{ e.g. } \begin{bmatrix} 1 & 1 \\ {x_1}^1 & {x_2}^1 \\ & \vdots \\ {x_1}^p & {x_2}^p \end{bmatrix}$$

Multiple Polynomial model: $h(x) = \sum_{k=1}^{p} w_k^T x^k + w_0$. Fit is the same as Polynomial model, but $(x^k)^T = \begin{bmatrix} x_{11}^k & \cdots & x_{m1}^k \\ \vdots & \ddots & \vdots \\ x_{1d}^k & \cdots & x_{md}^k \end{bmatrix}$

Basic Algorithms II

Linear Classifier: $h(x) = \text{sign}(x^T \cdot a - b)$, or tanh instead of sign. Solve the same as linear regression, but use $w = \begin{bmatrix} -b \\ a \end{bmatrix}$.
Perceptron Algorithm: Initialize $\mathbf{w}_0 = \mathbf{0}$. For each $\mathbf{x}_i$: If $\text{sign}(\mathbf{w}_{i-1}^T\mathbf{x}_i) \neq y_i$: $\mathbf{w}_i = \mathbf{w}_{i-1} + y_i\mathbf{x}_i$.
Logistic Regression: $p(x) = \sigma(w^T x)$, $\sigma(z) = \frac{1}{1+e^{-z}}$. Use empirical risk $R_S(\mathbf{w}) = \sum_{i=1}^{m} \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i))$.
Decision Tree: Disjunctive Normal Form over features (e.g. $(x_1 \wedge \neg x_2) \vee \cdots$). Train by choosing split (feature=true | feature=false) with least impurity I. Stopping: Depth limit, min samples per leaf, or pure node.
Ensemble Methods: Multiple models, classify with majority voting.
Bagging: Split training data for each model. E.g. **Random Forest**.
Boosting: Train second model on errors made by first model.

Experiment Design & Evaluation

Hyperparameter: model parameter (e.g. tree depth, #leaves).
Grid Search: Search for optimal Hyperparam. in grid exhaustively.
Random Search: Randomly try out Hyperparam. n times.
Random Seed: Not a Hyperparam.. Set to fixed value.
Dataset splitting: Train, Validation (for Hyperparameters), Test

- **Holdout:** Split e.g. 80% Train, 10% Val, 10% Test.
- k**-fold:** Split k parts, one part Test, rest Train. Train k models.
- **Stratification:** Equally distribute labels across splits.

Regression Metrics (Loss): *M*ean **A**verage **E**rror, (**R**oot) **M**ean **S**quared **E**rror.
Acc. $\frac{TP+TN}{TP+TN+FP+FN}$ **Pre.** $\frac{TP}{TP+FP}$ **Rec.** $\frac{TP}{TP+FN}$ **F1** $2 \cdot \frac{\text{Pre.}\cdot\text{Rec.}}{\text{Pre.}+\text{Rec.}}$
Baseline model: Dummy model to compare trained model to.

ML Theory & PAC Learning

Hypothesis Class $\mathcal{H}$: Class of all possible hypotheses.
i.i.d. Assumption: Independent identically distributed samples.
Realizability: Does a perfect h in $\mathcal{H}$ exist.
$\mathcal{H}$ **PAC-Learnable:** $\exists$Algorithm A, error at most ε, prob δ to fail satisfying min error, min required samples "sample complexity" $m(\varepsilon, \delta) \geq \frac{1}{\varepsilon}(\ln(|\mathcal{H}|) + ln(1/\delta))$.
$\mathcal{H}$ **shatters** X if all labellings of X have a correct $h \in \mathcal{H}$.
Vapnik-Chervonenkis (vc) Dimension: Size of largest X, where $\mathcal{H}$ shatters X. $\text{vc}(\mathcal{H}) \leq \log_2|\mathcal{H}|$.
If $\text{vc}(\mathcal{H}) = d < \infty$ then $m(\varepsilon, \delta) \leq \mathcal{O}(\frac{1}{\varepsilon}(d\ln(1/\varepsilon) + \ln(1/\delta))$.

SVM & Kernel Methods

Support Vector Machine (SVM): computes Hyperplane, based on support vectors.
Hyperplane: Defined by $w^T x = b$, for $x_i \in$ support vectors we can calculate $y_i(w^T x_i + b) = 1$. **Margin** $\gamma = 1/||w||$.
Kernel Function: Computes similarity between two data points, can be used as new "dimension".
Positive Semi-Definite: $\forall c : c^T Ac \geq 0$. Calculate by using $c^T = (x, y, z)$.

Probabilistic ML

Bayesian Inference: Let Θ be hypothesis, $\mathcal{X}$ be data.

$$\underbrace{P(\Theta|\mathcal{X})}_{\text{Posterior}} = \frac{\overbrace{P(\mathcal{X}|\Theta)}^{\text{Likelihood}}\overbrace{P(\Theta)}^{\text{Prior}}}{\underbrace{\sum_{\theta'} P(\mathcal{X}|\Theta = \theta')P(\Theta = \theta')}_{\text{Marginal Likelihood (Evidence)}}} = \frac{P(\mathcal{X}|\Theta)P(\Theta)}{P(\mathcal{X})}$$

Max Likelihood Est.: $\theta^*_{MLE} = \arg\max_\theta P(\mathcal{X}|\Theta = \theta)$
E.g. $P(Tails|A) = 0.1$, $P(Tails|B) = 0.9$, therefore $MLE = 0.9$.
Max A Posteriori Est.: $\theta^*_{MAP} = \arg\max_\theta P(\mathcal{X}|\Theta = \theta)P(\Theta = \theta)$
E.g. $P(Tails|A) = 0.1$, $P(Tails|B) = 0.9$, $P(A) = 0.99$, $P(B) = 0.01$ therefore $MAP = 0.99 * 0.1 = 0.099$
Factorized Probability Distribution Example with X_2 depends on X_1, X_3 depends on X_2, X_4 depends on X_2:
$P(X_1, X_2, X_3, X_4) = P(X_1) \cdot P(X_2|X_1) \cdot P(X_3|X_2) \cdot P(X_4|X_2)$

Dim. Reduction & Distance Algorithms

Principal Component Analysis (PCA): Reduce number of dimensions, by projecting data X onto direction w. Maximize variance $\text{Var}(Xw) \Leftrightarrow$ Minimize reconstruction error $||X - Xww^T||^2$. $w =$ eigenvector of C with largest eigenvalue, $C = \frac{1}{n-1}X^T X$.
Random Projection (Johnson-Lindenstrauss): faster than PCA, but some error expected
t-SNE: alt. to PCA, but non-(linear, deterministic, parameter-free), tries to preserve similar data points being close to each other.
k-NN: Classify point based on majority class of k nearest neighbors.
k-Means: Iteratively find k clusters with least distance from center.
Spectral Clustering: Works on non-circular shapes, unlike k-Means.
Hierarchical Clustering: Tree of clusters, arbitrary # of clusters.

Deep Neural Networks

Multi-layer perceptron (MLP): Stacking of multiple perceptrons. $L =$ # of layers. $a_i^l =$ node i in layer l. $w_{i,j}^l =$ weight from node j in layer l to node i in layer $l+1$. $s^l =$ pre-activation vector in layer l. $a^l =$activation vector in layer l. W^l weights of layer l. $b^l =$ bias.
Activation Function: Non-linear function $\sigma \Rightarrow$ non-linear network. $a^l = \sigma(s^l) = \sigma(W^{l-1}a^{l-1})$. $s_j^l = \sum_i a_i^{l-1}w_{i,j}^{l-1}$. σ e.g. tanh, ReLU.
Universal Approx. Theorem: Network with >1 hidden layer, non-linear activation can approximate continuous function $[0,1]^n \to [0,1]$.
Forward Pass: Calculation of output nodes based on input nodes.
Backpropagation: Minimize Loss $\mathcal{L}(\hat{y}, y)$. Error at layer l is δ^l. $\delta^{l-1} = (W^l)^T\delta^l \odot \sigma'(s^{l-1})$ ($\odot$... component wise multiplication).

Gradient $\nabla W^l = \delta^l (a^{l-1})^T$, gradient of bias $\nabla b^l = \delta^l$.
Example with 1 hidden layer:
$\nabla W^2_{new} = \delta^2 (a^1)^T$. $\nabla W^1 = \delta^1 (a^0)^T = ((W^2)^T \delta^2 \odot \sigma'(s^1))(a^0)^T$

Bias and Fairness in AI

Fairness through Unawareness: removing protected attribute A fails because proxies (e.g., ZIP code) correlate with A.

1. Statistical Parity (Naïve): Assign labels at equal rate to groups.
Con: If different groups have different rates, fails.

2. Calibration: Outcome independent of group given score.
Con: Error distribution can differ.

3. Error Rate Balance: Equal error rates across groups.
Con: Cannot be combined with calibration.

4. Individual Fairness: Treat similar individuals similarly (measure their distance in the data, risk proportional to distance).
Con: Difficult to define distance measures.

Reinforcement Learning

An **Agent** interacts with an **Environment**. At time step t, agent observes **State** S_t, selects **Action** A_t based on **Policy** π, and receives **Reward** R_{t+1} and a new state S_{t+1} from the environment.
State: Representation of the current situation of the environment.
Action: The decision or move made by the agent.
Reward: Scalar feedback indicating immediate success of an action.
Policy: A mapping (function) from states to actions (or probabilities of actions) defining the agent's behavior.
Return (G_t): The cumulative sum of discounted future rewards, defined as $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.
Discount Factor (γ): A value $\in [0, 1]$ that determines the importance of future rewards compared to immediate rewards.
Value Function: The expected return starting from a state (and action) following a specific policy.
Goal: The objective is to find an optimal policy π that maximizes the expected value of the return $\mathbb{E}_\pi[G_t | S_t = s]$.