Machine Learning Cheat Sheet

1. ML Basics & Problem Setup

- **Machine Learning (ML):** Field of study giving computers ability to learn without explicit programming.
- **AI vs. ML:** AI for fundraising (sci-fi), ML for hiring (engineering). ML is a subset of AI.
- **ML in a Nutshell:** Program learns from **Experience (E)**, with respect to a **Task (T)**, as measured by **Performance (P)**, if its performance at tasks in T, as measured by P, improves with E.
    - *Example:* T: Recognize human face, P: Accuracy, E: Dataset of faces.
- **Machine Learning Model / Classifier / Hypothesis (h):** Learned program/function mapping inputs to outputs/predictions.
- **Target Function (f):** The true function $f: X \to Y$ we want to model, mapping data points (instances) to targets/labels.
- **Hypothesis Class (H):** Set of possible functions $h: X \to Y$ that can be learned.
    - *Problem:* H is usually very large, making random selection or exhaustive search impractical.
- **Objective of ML:** Discover patterns in data, make predictions on unseen data (test set) based on learned experience. Find $h \in H$ that minimizes error on *training data* and unseen *future examples*.
- **Loss Functions:** Used to quantify errors. Higher loss = worse performance; zero loss = no errors.
    - **Zero-One Loss:** Simplest, counts number of mistakes an h makes on training set.
    - **Overfitting Issue (Memorizer):** A model $h(x) = yi$ if x is in training data D, 0 otherwise, has **zero training error** but performs *horribly* on unseen data, failing to generalize.
- **Generalization:** We want minimum error for all $(\sim x, y)$ drawn from the true underlying distribution P (which is unknown). Test loss approximates generalization loss.
    - **Why $\varepsilon_{TE} \to \varepsilon$ as $|D_{TE}| \to +\infty$?** Weak law of large numbers: empirical average converges to its mean.

2. Dataset Understanding & Preprocessing

- **Dataset / Training Set / Database:** Set of data with features and the target to predict.

- **Instance / Data Point / Example / Observation:** A row in the dataset, consists of feature values.

- **Features / Attributes:** Inputs used for prediction/classification. A column in the dataset. An individual measurable property or characteristic of an observed phenomenon.
    - **Feature Vector:** `n` features describing one observation: `<x_i1, x_i2, ..., x_in> ∈ R^n`.
    - **Good Features:** Make it easy for the classifier to decide/learn between classes (enhance inter-class variation, minimize intra-class variation).
- **Feature Types:**
    - **Numerical:** Expressed in numbers.
        - **Continuous:** Can take any real value (e.g., temperature, weight).
        - **Discrete:** Count from a set of distinct whole values (e.g., number of rooms).
    - **Categorical:** Stored in groups or categories.
        - **Ordinal:** Values can be logically ordered/ranked (e.g., short, tall).
        - **Nominal:** Values cannot be logically ordered (e.g., color, names). Numerical nominal values have no mathematical interpretation (e.g., labeling people 1, 2, 3).
- **Preprocessing:** Transformations applied to raw data before feeding to algorithm, converting it into a clean, suitable format.
    - **Motivation:** Remove redundant info/outliers, noise removal, meet specific algorithm format requirements (e.g., Random Forest doesn't support nulls, PCA needs zero mean/unit variance).
    - **Techniques:**
        - **Data Scaling / Normalization:** Transforms features to fit a specific scale (e.g., 0-1 or 0-100) or distribution (e.g., Gaussian with mean 0, std 1 for PCA).
            - **Motivation for Scaling:** Features with wide range can dominate cost functions (e.g., Euclidean distance), and algorithms (e.g., Gradient Descent) converge faster. PCA results biased without scaling.
            - **Min-max normalization:** `x_scaled = (x - min(x)) / (max(x) - min(x))`. Scales to ``.
            - **Mean normalization:** `x_scaled = (x - x¯) / (max(x) - min(x))`.
            - **Standardization:** `x_scaled = (x - x¯) / σ`. Makes features zero-mean and unit-variance.
        - **Outlier Removal:** Outliers are data points very different from the mean, can cause large errors or bias (e.g., AdaBoost, linear/logistic regression, K-NN if K is small).
            - **Detection:** Box-Plot for visualization.
            - **Dealing with Outliers:**

        - Discard if small number and large dataset.
        - **Winsorizing:** Set extreme values to a specified percentile value.
        - **Log-Scale Transformation:** Reduce data variability.
        - Adopt robust cost functions.
    - **Missing Data / Null Value Handling:**
        - Discard feature vectors (if large dataset, rare missing values).
        - "Complete" by zeros, mean, or customized function (imputation).

3. Model Evaluation Workflow & Metrics

- **Workflow for Supervised Learning (Classification):**
    - Data preprocessing and feature extraction.
    - Training phase: `{<x_i, y_i>} → algorithm`.
    - Evaluation phase: Provides feedback to improve model accuracy. *The training process repeats until desired accuracy is achieved.*
- **Model Evaluation:** Crucial for gauging performance on *unseen data* (generalization ability).
    - **Common Mistake:** Never evaluate performance on training data; conclusion would be biased.
    - Good training accuracy can be achieved by memorizing training data (overfitting), hence evaluating on unseen data is vital.
- **Dataset Partitioning Techniques:**
    - **Hold-out Cross Validation:** Data randomly partitioned into train and test sets. Often, three sets are used: training, validation (for hyperparameter tuning), and testing.
    - **K-fold Cross Validation:** Dataset divided into `k` equal subsets. `k-1` subsets for training, 1 for testing. Process repeated `k` times, each subset used once for testing. `k` estimations are averaged.
    - **Bootstrap (Bagging):** Sampling *with replacement* to form training set. Generates `m` new datasets, each size `n' < n`. Some entries can be repeated.
    - **Leave-One-Out Cross-Validation (LOOCV):** `N` experiments, using `N-1` samples for training and 1 for testing. Errors are averaged.
- **Key Principle:** More training data gives better generalization; more test data gives better estimate for classification error probability.
- **Measures of Classification Performance:**
    - **Classification Accuracy:** `Accuracy = N_cc / D_ts = (Tp + Tn) / (Tp + Fp + Tn + Fn)`.

        - **Drawback:** Can be misleading with **class imbalance**. E.g., 98% class A, model predicts all as A, gets 98% accuracy but is useless.
    - **Confusion Matrix:** Diagnostic/visualization tool showing actual vs. predicted classifications. Provides insight into *where misclassifications occurred*.
        - **True Positive (Tp):** Classifier predicted positive, actual is positive.
        - **True Negative (Tn):** Classifier predicted negative, actual is negative.
        - **False Positive (Fp / Type I error):** Classifier predicted positive, actual is negative.
        - **False Negative (Fn / Type II error):** Classifier predicted negative, actual is positive.
    - **Sensitivity / Recall:** Ratio of positive class correctly detected. How good model is at recognizing positive class..
        - `Recall = Tp / (Tp + Fn)`.
        - *Useful when cost of False Negatives is high* (e.g., Cancer Detection).
    - **Specificity:** Ratio of actual negatives predicted as negative.
        - `Specificity = Tn / (Fp + Tn)`.
    - **Precision:** Ratio of correctly classified positive examples to total predicted positive examples.
        - `Precision = Tp / (Tp + Fp)`.
        - *Useful when cost of False Positives is high* (e.g., Spam Email Identification – legitimate emails shouldn't be spam).
    - **Precision vs. Recall Trade-off:** Increasing recall often lowers precision, and vice-versa. Not possible to maximize both simultaneously.
    - **F-Measure (F-score):** Harmonic mean of precision and recall.
        - `Fα = (1 + α^2) * (precision * recall) / (α^2 * precision + recall)`.
        - **F1-score (α=1):** `F1 = 2 * (precision * recall) / (precision + recall)`.
- **Beyond Numerical Metrics (Important Considerations):**
    - **Interpretability:** Critical if model needs explanation for transparency.
    - **Complexity / Simplicity:** Simplicity is important for practical reasons (easier to tune).
    - **Scalability:** Model needs to scale with data size or parameters.
    - **Fast Prototyping:** Speed of delivery for production vs R&D.

4. Machine Learning Algorithms

4.1. K-Nearest Neighbors (KNN)

- **Definition:** Supervised ML algorithm for classification and regression.
- **How it Works (Classification):**
    - Find the `k` training examples closest to a test example `x`.

    - Predict the most frequent label in that set.
- **"Lazy Learning":** No model training phase; essentially "saves the data".
- **Prediction:** Relatively slow compared to linear classifiers or decision trees.
- **Decision Boundary (1-NN):** Voronoi Diagram, meaning it's non-linear.
- **Distance Metrics:**
    - **Euclidean Distance:** `||x1 - x2||`.
    - **Manhattan Distance:** (mentioned as option for K-Means, also applicable).
    - **Cosine Distance:** `1 - <x1, x2> / (||x1|| * ||x2||)`.
- **Choosing K:** Small K sensitive to noise; Large K might ignore local patterns. No perfect K, use cross-validation.
- **Strengths:** Simple, intuitive, no training phase, effective with small datasets.
- **Weaknesses:** Computationally slow for large datasets, sensitive to irrelevant/unscaled features, doesn't work well with high-dimensional data.

4.2. K-Means Clustering

- **Definition:** Most popular unsupervised learning algorithm for clustering.
- **Goal:** Group objects into `K` clusters.
- **Algorithm Steps (Iterative):**
    - **Initialization:** Set `K` seed points (randomly).
    - **Assignment Step:** Assign each object to the cluster of the *nearest* seed point using a distance metric (e.g., Euclidean distance).
    - **Update Step:** Compute new seed points as the *centroids* (mean point) of the current clusters.
    - **Repeat:** Go back to Step 1 (Assignment) until no more new assignments occur (i.e., membership no longer changes, convergence).
- **Partitioning:** Partitions the whole data space into `K` mutually exclusive subspaces.
- **Efficiency:** `O(tKn)`, where `n` = #objects, `K` = #clusters, `t` = #iterations (`K, t << n`).
- **Issues / Weaknesses:**
    - **Local Optimum:** Sensitive to initial seed points, may converge to an unwanted local optimum.
    - **Requires K:** Need to specify the number of clusters `K` in advance.
    - **Noisy Data & Outliers:** Unable to handle them well (K-Medoids algorithm is more resistant).
    - **Mean Dependent:** Applicable only when mean is defined (K-Mode for categorical data).
- **Application Example:** Color-Based Image Segmentation.

4.3. Support Vector Machines (SVM)

- **Definition:** Classification method for both linear and nonlinear data.
- **Core Idea:** Transforms original training data into a higher dimension (using a nonlinear mapping) to find a *linear optimal separating hyperplane* (decision boundary).
- **Key Components:**

    - **Support Vectors:** Training tuples that fall on the hyperplanes defining the margin; they are the "essential" or "critical" training examples closest to the decision boundary.
    - **Margin:** Distance between the hyperplane and the closest data points (support vectors). SVM searches for the hyperplane with the **largest margin** (Maximum Marginal Hyperplane - MMH).
- **Hyperplane:** `W • X + b = 0`. For 2D, a line; for 3D, a plane; in higher dimensions, a hyperplane. `W` is weight vector, `b` is bias.
- **Why Effective on High-Dimensional Data:** Complexity of trained classifier is characterized by the *number of support vectors*, not the dimensionality of data. Small number of support vectors implies good generalization even in high dimensions.
- **Linearly Inseparable Data:** Handled by transforming data into a higher-dimensional space where a linear separation becomes possible. This is achieved via **Kernel Functions** ("Kernel Trick").
    - **Typical Kernel Functions:** Polynomial, Radial Basis Function (RBF).
- **Scalability Issues:** SVM is *not scalable* for very large datasets in terms of training time and memory usage.
    - **CB-SVM (Clustering-Based SVM):** Uses hierarchical micro-clustering to reduce the number of points for SVM, then de-clusters near "candidate vectors" for accuracy.
- **SVM vs. Neural Networks:**
    - **SVM:** Deterministic, good generalization properties, hard to learn (batch mode, quadratic programming), uses kernels for complex functions.
    - **NN:** Non-deterministic, generalizes well (but less mathematical foundation), easier incremental learning, needs multi-layer perceptron for complex functions.
- **Hyperparameters:**
    - `C`: Trades off misclassification of training samples against decision surface simplicity. Low `C` = smooth surface; High `C` = aims for perfect training classification (more support vectors).
    - `Gamma`: Defines influence of a single training example. Low `Gamma` = 'far' influence; High `Gamma` = 'close' influence (inverse of support vector radius of influence).

4.4. Naive Bayes Classifier

- **Definition:** A probabilistic classifier based on **Bayes' Rule**. It's a **generative model** (learns `P(x|c)`), meaning it models data within each class.
- **Bayes Rule:** `P(Hypothesis | Data) = (P(Data | Hypothesis) * P(Hypothesis)) / P(Data)`.
    - **Prior Probability `P(H)`:** Probability of hypothesis before seeing data.
    - **Conditional Probability `P(D|H)` (Likelihood):** Probability of data given the hypothesis.
    - **Posterior Probability `P(H|D)`:** Probability of hypothesis given the data.

- **"Naive" Assumption:** All input features are **class conditionally independent**. This simplifies learning the joint probability `P(x1, ..., xn | c)`.
    - `P(x | c) = P(x1 | c) * P(x2 | c) * ... * P(xn | c)`.
- **Classification Rule: Maximum A Posteriori (MAP)** rule: Assign `x` to the class `c` that maximizes `P(c | x)` (which is proportional to `P(x | c) * P(c)`).
- **Handling Continuous-Valued Features:** Conditional probability `P(x_j | c)` is often modeled with a **Normal (Gaussian) Distribution** (requires mean and standard deviation for each feature per class).
- **Zero Conditional Probability Problem:** If no example contains a specific feature value, its conditional probability can be zero, causing the entire product for that class to become zero.
    - **Remedy (m-estimate):** Re-estimate probabilities by adding `m` "virtual" examples to smooth out counts. `P(a_jk | c_i) = (n_ijk + mp) / (n_i + m)`.
- **Strengths:** Training and testing are very efficient. Works well for text classification (e.g., spam filtering). Competitive performance even if independence assumption is violated. Good base learner for ensemble learning.

4.5. Decision Trees

- **Definition:** Flowchart-like tree structure where each internal node tests an attribute, each branch represents an outcome, and each leaf node holds a class label.
- **Key Advantage: Understandability** – simulates human decision-making and is easy to interpret.
- **Tree Induction (Learning):** Greedy strategy; recursively splits records based on an attribute test that optimizes a certain criterion.
- **How to Specify Test Condition (Splitting):**
    - Depends on **attribute types:** Nominal, Ordinal, Continuous.
    - Depends on **number of ways to split:** 2-way (binary) split or Multi-way split.
    - **Continuous Attributes:** Often handled by binary splits (e.g., `Attribute < v`). Split points can be halfway between values; sorted instances can speed up computation.
- **Attribute Selection Criteria ("Impurity Measures"):** Choose attribute that produces the "purest" nodes (least mixed classes).
    - **Entropy:** Measures information (in bits) required to predict an event given a probability distribution. `Entropy(S) = - Σ (pi * log2(pi))`.
    - **Information Gain:** `Gain(S, A) = Entropy(S) - Σ (|Sv|/|S|) * Entropy(Sv)`. Expected reduction in entropy from knowing attribute `A`'s value.
        - **Disadvantage:** Biased towards attributes with a large number of distinct values (e.g., ID code), as they tend to create many small, pure partitions, resulting in high gain.
    - **Gain Ratio:** Modifies Information Gain to reduce its bias for high-branch attributes by normalizing it with "Intrinsic Information" (entropy of the split distribution).

        - `IntrinsicInfo(A, S) = - Σ (|Si|/|S|) * log2(|Si|/|S|)`.
        - `GainRatio(A, S) = Gain(S, A) / IntrinsicInfo(A, S)`.
        - **Problem:** Can overcompensate, choosing attributes with very low intrinsic information.
    - **Gini Index (CART):** `Gini(T) = 1 - Σ (p_j)^2`. Minimized if classes in `T` are skewed (pure). Favors tests that result in equal-sized partitions and purity in both partitions. Used for binary splits.
    - **Chi-Square (CHAID):** Measures statistical significance between differences in sub-nodes and parent node. Works with categorical target, can perform two or more splits. `Chi-square = ((Actual − Expected)^2 / Expected)^1/2`.
- **Overfitting:** Induced tree may be too complex, reflecting anomalies/noise, leading to poor accuracy on unseen samples.
- **Tree Pruning (Goal: Prevent Overfitting):**
    - **Pre-pruning (Early Stopping):** Stops tree growth prematurely when information becomes unreliable. Faster than post-pruning. *Drawback:* May fail to capture complex relationships (e.g., XOR problems) if individual attributes don't show strong association early.
    - **Post-pruning:** Build full tree first, then prune it. Shows all attribute interactions. Operations include Subtree Replacement and Subtree Raising. Prune only if estimated error reduces.
- **Extracting Rules:** Decision trees can be converted to IF-THEN classification rules by tracing paths from root to leaves.

4.6. Random Forests

- **Definition:** An **ensemble approach** that combines multiple decision trees ("weak learners") to form a "strong learner".
- **How it Works (Training):**
    1. Generate `T` new datasets by **sampling with replacement** (`~66%` of total data) – this is **Bagging**.
    2. For each tree (at each node): Randomly select `m` predictor variables from all available variables.
    3. Choose the predictor variable from these `m` that provides the *best split* (based on objective function).
    4. Grow the tree.
- **Prediction:** New input runs down all trees; result is average/weighted average (regression) or voting majority (categorical) of terminal nodes.
- **Strengths:** Fast runtimes, deals with unbalanced and missing data.
- **Weaknesses:** Cannot predict beyond training data range for regression; may overfit noisy datasets.

4.7. Principal Component Analysis (PCA)

- **Definition:** An unsupervised dimensionality reduction technique.
- **Purpose of Dimensionality Reduction:**
    - Avoid the **curse of dimensionality** (data becomes sparse in high dimensions).
    - Reduce time and memory requirements for algorithms.
    - Enable visualization (project to 2D/3D).
    - Eliminate irrelevant features or reduce noise.
- **PCA Goal:** Find a projection that captures the **largest amount of variation (variance)** in data. This is equivalent to finding a projection with **minimum reconstruction error**.
- **Algorithm Steps:**
    - **Center the data:** Subtract the mean from each feature/row.
    - **Compute Covariance Matrix (S):** `S = (1/N) * Xc^T * Xc` (where Xc is centered data).
    - **Find Eigenvectors and Eigenvalues:** of the covariance matrix `S`.
    - **Sort:** Eigenvectors are sorted by their **descending eigenvalues** (larger eigenvalue = more important eigenvector/component).
    - **Select Principal Components:** Choose the top `k` eigenvectors with the highest eigenvalues.
    - **Project Data:** Project the centered data onto these `k` principal components.
- **Scaling Up:** For large covariance matrices, use **Singular Value Decomposition (SVD)** to find `k` eigenvectors efficiently. `X = U * S * V^T`.
- **Assumptions of PCA:**
    - Means and covariance matrix are sufficient to describe predictor distributions (true for multivariate Normal distribution, but often works approximately).
    - High variance indicates importance (must be checked, as this may not always hold true).
    - **When assumptions fail:** Linear methods may fail if data lies on a nonlinear manifold; **Kernel PCA** can be used for nonlinear dimensionality reduction.
- **Applications:** Image Embedding, Eigenfaces (face recognition), Image Compression.

4.8. DBSCAN Clustering

- **Definition:** An unsupervised, density-based spatial clustering algorithm that identifies clusters based on density and is effective for arbitrary-shaped clusters and noise. It does not require input for the number of clusters.
- **Parameters:**
    - **`eps` (epsilon):** Maximum distance between two data points to be considered neighbors (defines the neighborhood radius).
        - *Too small:* large part of data considered outliers.
        - *Too large:* clusters merge, most data in same cluster.
        - **Determining `eps`:** K-distance graph (look for point of maximum curvature).

    - **`min_samples` (MinPts):** Minimum number of neighbors (data points) within `eps` radius for a point to be considered a core point (and thus part of a dense region/cluster).
        - Larger datasets require larger MinPts. General rule: `MinPts >= D + 1` (where D is dimensionality), minimum value `3`.
    - **`metric`:** Distance metric (e.g., Euclidean distance).
- **Types of Data Points:**
    - **Core Point:** Has `>= MinPts` points within its `eps` neighborhood. Interior of a cluster.
    - **Border Point:** Has `< MinPts` within `eps`, but is in the neighborhood of a core point.
    - **Noise / Outlier Point:** Neither a core point nor a border point.
- **DBSCAN Process:**
    - Find all neighbor points within `eps` for every point, identify core points.
    - For each core point not yet assigned to a cluster, create a new cluster.
    - Recursively find all its **density-connected** points (core points reachable through a chain of directly density-reachable core points, or border points within `eps` of a core point) and assign them to the same cluster.
    - Points that do not belong to any cluster are noise.
- **Strengths:**
    - Can discover clusters of **arbitrary shape and size**.
    - Automatically finds the **number of clusters**.
    - Can **separate clusters from surrounding noise**.
- **Weaknesses:**
    - Cannot handle **varying densities** well.
    - Sensitive to parameter settings (though `eps` can be guided by k-distance graph).

4.9. Regression

- **Definition:** A supervised learning task where the output is a **continuous (real number)** value. (Opposite of Classification, which has discrete outputs).
- **Goal:** Create a (linear) relationship between two or more variables to enable prediction.
    - Target: `y = f(x1, x2, ...)`.
- **Forms of Regression:**
    - **Linear Regression:** `y = b0 + b1x` (single predictor).
    - **Multiple Regression:** `y = b0 + b1x1 + b2x2 + b3x3 + ...` (multiple predictors).
    - **Polynomial Regression:** `y = b0 + b1x + b2x^2 + b3x^3 + ...` (nonlinear relation).
- **"Best" Regression:** Finds the curve (parameters) that minimizes the **Sum of Squared Errors (SSE)**. This is the cost function that is minimized.
- **Evaluation Metric: R-squared (`R^2` value):**

    - Statistical measure of how close the data are to the fitted regression line.
    - Ranges from 0% to 100%.
    - 0% = model explains none of variability. 100% = model explains all variability.
    - Higher `R^2` generally means better fit.

4.10. Logistic Regression

- **Definition:** Used for binary classification (predicts a probability between 0 and 1). Considered a **discriminative model** (distinguishes between classes).
- **Mathematical Basis:** Applies a **logit transformation** (natural logarithm of odds) to model the probability of success.
    - `Logit(p_i) = 1 / (1 + exp(-p_i))`.
    - `ln(p_i / (1 - p_i)) = Beta_0 + Beta_1*X_1 + ... + B_k*X_k`.
- **Parameter Estimation:** Beta parameters (coefficients) are typically estimated via **Maximum Likelihood Estimation (MLE)** or, in machine learning, by minimizing the **negative log likelihood** using **Gradient Descent**.
- **Binary Classification Rule:** Probability less than `0.5` predicts `0`, greater than `0.5` predicts `1`.
- **Goodness of Fit:** Assessed using methods like the **Hosmer–Lemeshow (HL) test**.
- **Odds Ratio (OR):** Exponentiating beta estimates transforms results into OR for easier interpretation.
    - `OR > 1`: event associated with higher odds of specific outcome.
    - `OR < 1`: event associated with lower odds of specific outcome.
- **Types:** Multinomial (3+ unordered outcomes), Ordinal (3+ ordered outcomes).
- **Overfitting:** Can be prone to overfitting, especially with many predictors. **Regularization** is used to penalize large coefficients.

4.11. Reinforcement Learning (RL)

- **Definition:** ML method where an agent receives a **delayed reward** from an environment to evaluate its previous action.
- **RL Setup Components:**
    - **Agent:** The RL algorithm itself.
    - **Environment:** The object the agent acts on.
- **RL Process:** Environment sends a **state** to agent → Agent takes an **action** → Environment sends **next state** and **reward** → Agent updates its knowledge → Loop continues until **terminal state**.
- **Key Terminology:**
    - **Action (A):** Possible moves agent can take.
    - **State (S):** Current situation returned by environment.
    - **Reward (R):** Immediate return from environment for last action.
    - **Policy (π):** Strategy agent uses to determine next action based on current state.
    - **Value (V):** Expected long-term return with discount. `Vπ(s)` is expected long-term return of state `s` under policy `π`.

    - **Q-value / Action-value (Q):** Expected long-term return of state `s` taking action `a` under policy `π`. `Qπ(s, a)`.
- **Model-free vs. Model-based:**
    - **Model-based:** Learns transition probability `T(s'|s,a)` (how likely to enter `s'` given `s` and `a`). Impractical for large state/action spaces.
    - **Model-free:** Relies on trial-and-error to update knowledge; does not require storing all state-action combinations.
- **On-policy vs. Off-policy:**
    - **On-policy:** Agent learns value based on its *current policy's actions*.
    - **Off-policy:** Agent learns value based on actions obtained from *another policy* (e.g., greedy policy in Q-learning).
- **Q-Learning:**
    - **Q-table (matrix):** `[state, action]` shape, initialized to zero. Becomes reference for best action.
    - **Explore vs. Exploit:**
        - **Exploit:** Select action with max value from Q-table.
        - **Explore:** Select a random action (important to discover new states). Balance with `epsilon (ε)` parameter.
    - **Transition Rule (Bellman Equation for Q-Learning):** `Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]`
        - **Learning Rate (`lr` / `alpha` / `α`):** How much new value is accepted vs. old.
        - **Gamma (`γ`):** Discount factor (0.8 to 0.99) for balancing immediate and future rewards. Future rewards are discounted. Larger `γ` means future rewards have more impact.
- **Deep Q Network (DQN):** Addresses Q-learning's lack of generality for unseen states by introducing Neural Networks instead of a Q-table.

5. Markov Decision Processes (MDPs)

- **Definition:** A framework to cope with **uncertainty** in sequential decision-making.
- **Components:**
    - **States (S):** Set of possible situations.
    - **Actions (A):** Possible choices in a state. `Actions(s)`: possible actions from state `s`.
    - **Transition Probabilities `T(s'|s, a)`:** Probability of ending in state `s'` if action `a` is taken in state `s`. (Contrast with deterministic `Succ(s,a)` in search problems).
    - **Reward `Reward(s, a, s')`:** Reward for the transition `(s, a, s')`.
    - **`IsEnd(s)`:** Whether `s` is a terminal state.
    - **Discount Factor `γ`:** `0 <= γ <= 1`. Discounts future rewards.

- **Solution to an MDP:** A **Policy (π)**, which is a mapping from each state `s ∈ States` to an action `a ∈ Actions(s)`. (Contrast with a "path" in search problems).
- **Objective Functions:** Map infinite sequences of rewards to a single utility value.
    - **Discounting:** Most common; a reward `n` steps away is discounted by `γ^n`.
        - **Why Discounting?** Models mortality (you may die), preference for shorter solutions, inflation.
- **Value Function `Vπ(s)`:** Represents the expected discounted sum of future rewards (utility) obtained by following policy `π` from state `s`.
- **Q-value `Qπ(s, a)`:** Expected utility of taking action `a` from state `s`, and then following policy `π`.
- **Bellman Equations:** Relate the value function to itself via the problem dynamics.
    - **For `Vπ(s)`:** `Vπ(s) = R(s, π(s)) + Σ_s' T(s, π(s), s') * γ * Vπ(s')` (if not end state).
    - **For `V*(s)` (Optimal Value):** `V*(s) = max_a [R(s, a) + Σ_s' T(s, a, s') * γ * V*(s')]` (if not end state).
- **Optimal Value `V*(s)`:** The maximum value attained by any policy. All optimal policies have the same `V*`.
- **Optimal Policy `π*(s)`:** `π*(s) = arg max_a Q*(s,a)`.
- **Algorithms:**
    - **Policy Evaluation:** Iterative algorithm to compute `Vπ(s)` for a *given* policy `π`. Starts with arbitrary values and repeatedly applies Bellman equations until convergence.
    - **Value Iteration:** Iterative algorithm to compute the **optimal value `V*(s)`** and thus the **optimal policy `π*(s)`**.
        - **Convergence:** Guaranteed if `γ < 1` or the MDP graph is acyclic.

6. MLOps (Machine Learning Operations)

- **Definition:** Set of practices focused on **rapidly and reliably developing, deploying, and maintaining ML models in production**. Fosters communication/collaboration.

- **Machine Learning Deployment Gap:** Bridging the gap between building models and deploying them is challenging (repetitive manual work, messy experiment tracking, lack of versioning, avoiding testing, narrow focus on single metric).

- **Main Challenge:** ML systems are an entanglement of data, models, and code. Data and business needs change constantly, leading to the "Changing Anything Changes Everything Principle".

- **MLOps Principles ("Continuous X"):**

    - **Continuous Integration (CI):** Frequently merging code, automated builds and tests.
    - **Continuous Delivery (CD):** Automatically building, testing, and deploying ML models.
    - **Continuous Training (CT):** Constantly training different models for comparison/benchmarking.
    - **Continuous Monitoring (CM):** Constantly monitoring aspects of ML systems in production.
    - **Continuous Documentation:** Automatically generating and updating documentation.
    - **Version Everything:** Artifacts (data, code, models, environments, docs) should be stored and versioned to enable:
        - **Reproducibility:** Rerunning experiments with same results.
        - **Traceability:** Tracing experiments to code/data.
        - **Collaboration:** Multiple users working with same artifacts.
    - **Automate (whenever allowed):** Remove manual, error-prone tasks.
    - **If it isn't tested, it is broken:** Enforce passing tests, test entire system, different artifacts, various approaches.
    - **Reusability:** Ability to use model/code for multiple purposes/environments (modularity, agnostic tools, discoverable artifacts).
    - **Collaboration and Communication:** Frequent communication, transparency.
- **Model Deployment Strategies:**

    - **Batch Prediction:** Periodically run model, cache results in a database. Suitable for small input universes.
    - **Model as a Serverless Function:** Deploy model as an on-demand function.
    - **Model-in-Service:** Deploy model on an application server.
- **Monitoring ML Models:** More central to ML than traditional software, as bugs can cause "silent depredations in performance".

    - **Signals to Monitor:**
        - System performance.
        - Business metrics.
        - Model metrics (e.g., accuracy).
        - Model inputs and predictions.
    - **Measuring Distribution Change (Data Drift/Model Drift):**
        - Select a **measurement window** (problem-dependent).
        - Compare windows using **distance metrics**.
            - **Rule-based:** Min/max/mean in range, sufficient data points, low missing values.
            - **Statistical:**
                - **Kullback-Leibler (KL) Divergence:** Sensitive to tails, not interpretable.

            - **Kolmogorov-Smirnov (KS) Statistic:** Max distance between CDFs, easy to interpret, widely used.
            - **D1 distance:** Sum of distances between CDFs, used at Google.
        - **Multi-dimensional data:** Open problem; can use Maximum Mean Discrepancy (MMD) or a bunch of 1D comparisons.
        - **Projections:** Analytical (mean pixel value), Random, Statistical (Autoencoder, PCA/T-SNE).
    - **Closing the Data Flywheel:** Monitoring feeds back into data collection, cleaning, training, and deployment processes to continuously improve models.
