Introduction to Data Science

dr Maciej Świtała
Ewa Weychert

Class 5: Introduction to machine learning

e.weychert@uw.edu.pl

Course Roadmap for Today (90 minutes)

0–8 min	What is ML? Why now? Examples.
8–18	Supervised vs. Unsupervised (with intuition).
18–38	Core algorithms: regression, classification, clustering (pros/cons).
38–55	ML workflow: data → split (train/val/test) → cross-validation → leakage.
55–72	Metrics: regression (MAE/MSE/RMSE/R^2), classification (Accuracy, Precision, Recall, F1, ROC-AUC, PR-AUC), imbalance.
72–82	Overfitting, regularization, bias–variance, learning curves.
82–90	Interpretability, ethics, pitfalls, next steps, Q&A.

Agenda

Definition & Intuition

Machine Learning enables systems to **learn patterns from data** and make predictions/decisions without explicit rules.

- Traditional approach: *Rules + Data* $\rightarrow$ *Output*
- ML: *Data + Output* $\rightarrow$ *Learn Rules (Model)*
- Goal: **generalize** well to unseen data.

ML vs. Statistics — Conceptual View

Statistics

- Explain relationships; estimate effects and uncertainty.
- Hypothesis testing, confidence intervals, causal inference.
- Emphasis on model assumptions and interpretability.
- Sampling design, experimental control (e.g., RCTs, A/B tests).

Machine Learning

- Predictive performance on unseen data.
- Flexible function approximators; automatic feature learning.
- Scales to large/high-dimensional data.
- Automation: pipelines, model selection, deployment.

Reality: There is strong overlap; both are complementary. Choose by *goal* (explain vs. predict), *data* (size/quality), and *constraints* (risk, cost, interpretability).

Objectives, Assumptions, Guarantees

Dimension	How it differs
Primary objective	Statistics: inference, explanation, uncertainty; ML: generalization error minimization (prediction).
Assumptions	Statistics: explicit (e.g., linearity, normality, iid errors). ML: weaker/implicit; capacity control via regularization/validation.
Validation	Statistics: theory-driven diagnostics, tests. ML: empirical via holdout/CV, learning curves.
Uncertainty	Statistics: CIs, p-values, posteriors (Bayes). ML: calibration, ensembling variance.
Causality	Statistics: designed experiments, DAGs, IVs. ML: typically predictive; causal ML requires extra design/assumptions.

When to Prefer Which? (Decision Heuristics)

Prefer Statistics if:

- You need **explanation** or **effect sizes** with uncertainty.
- Decisions require **causal** claims (policy, medicine).
- Data are limited; strong assumptions help stabilize estimates.
- Simplicity and transparency are paramount.

Prefer ML if:

- You need **accurate predictions** at scale.
- Relationship is complex/nonlinear, many features.
- You can afford empirical model selection (CV, tuning).
- Deployment/automation is required (APIs, pipelines).

Best practice: Start with interpretable statistical baselines; adopt ML where it adds *predictive* value without violating domain constraints.

Econometric vs. Machine Learning Questions

Same dataset — different perspectives and goals.

Econometric Questions (Inference-focused)

- **Q1.** Is X_1 *negatively associated* with Y, controlling for other classical variables? → *Main variable X_1 (e.g., media consumption or sentiment index) is tested for its statistical relationship with outcome Y.*
- **Q2.** Is the estimated effect of X_1 statistically significant and robust to alternative specifications? → *Focus on p-values, confidence intervals, multicollinearity, omitted variable bias.*
- **Other variables:** based on the literature — socio-economic controls, demographics, or macroeconomic indicators.

Machine Learning Questions (Prediction-focused)

- **Q1.** Can X_1 help *predict* Y using machine learning models, while controlling for classical variables? → *Emphasis on out-of-sample performance (cross-validation, RMSE, ROC-AUC).*
- **Q2.** What is the *relative importance* of X_1 in predicting Y, as measured by SHAP or feature importance? → *Interpret the contribution of X_1 to model prediction rather than causal effect.*
- **Q3.** How does predictive accuracy change if X_1 is excluded from the feature set? → *Quantifies the incremental predictive power of X_1.*

Key insight: Econometrics seeks to explain *why* something happens (causal, inferential). Machine Learning seeks to predict *what will happen* (pattern-based, predictive).

Common Ground & Integration

Shared foundations: probability, estimation, bias–variance, model checking.

- Statistical rigor improves ML: proper experimental design, leakage prevention, uncertainty reporting.
- ML strengthens statistics: flexible models, automation, scalability.
- Modern practice blends both: causal ML, probabilistic ML, Bayesian deep learning, conformal prediction.

Takeaway: Use *statistical thinking* to frame the problem and ensure validity; use *ML tooling* to reach performance and scale.

Where You Already See ML

- Recommenders (Netflix, Spotify)
- Email spam filtering
- Credit scoring & fraud detection
- Demand/price forecasting
- Speech & image recognition
- Translation, chat assistants
- Navigation ETA predictions
- Medical diagnosis support

Agenda

Supervised Learning — Overview and Regression

Goal: Given a set of **features** $\mathbf{X} = (x_1, x_2, \ldots, x_p)$ and known **labels** y, learn a function $f : \mathbf{X} \to y$ that can accurately predict y for unseen data.

Training data: $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ **Objective:** Find $f^*(\cdot)$ that minimizes a loss function $L(y, f(\mathbf{x}))$.

Two major subtypes:

- **Regression** — the output y is **numeric, continuous.** Examples:
  - Predicting house prices (y = price in PLN).
  - Forecasting temperature (y = average daily temperature).
  - Estimating GDP growth, demand, or sales volumes.

  Common algorithms: Linear Regression, Ridge/Lasso, Random Forest Regressor, XGBoost, Neural Networks.
  Typical loss: Mean Squared Error (MSE)

$$L_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Supervised Learning — Classification

Classification — the output y is **categorical or discrete.**

- **Goal:** Assign each observation to one of several possible classes.
- **Examples:**
  - Spam detection (y = spam / not spam).
  - Credit scoring (y = default / no default).
  - Image recognition (y = cat / dog / car).
  - Medical diagnosis (y = healthy / sick).
- **Common algorithms:** Logistic Regression, k-Nearest Neighbors (kNN), Support Vector Machines (SVM), Decision Trees, Random Forest, Neural Networks.
- **Typical loss:** Cross-Entropy (Log Loss)

$$L_{\log} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right]$$

Key idea: Supervised learning requires **labeled data** — models learn mappings from inputs to known outputs. Performance is measured via **train–validation–test splits** or **cross-validation**.

Unsupervised Learning — Overview

Goal: Learn hidden structure or patterns in data without labeled outcomes. No y values are given — only features $\mathbf{X} = (x_1, x_2, \ldots, x_p)$.

Training data: $\{\mathbf{x}_i\}_{i=1}^n$ **Objective:** Find structure in $\mathbf{X}$ — clusters, manifolds, or key dimensions.

Key characteristics:

- Discover relationships or patterns that were not pre-defined.
- Evaluate qualitatively or via indirect metrics (e.g., silhouette score, reconstruction error).
- Useful for exploration, feature engineering, anomaly detection, or compression.

Examples of use cases:

- Grouping customers by purchase behavior.
- Discovering topics in text data.
- Detecting unusual transactions or system failures.
- Visualizing high-dimensional data in 2D/3D.

Unsupervised Learning — Major Techniques

1. **Clustering:** Grouping similar data points based on a distance or similarity measure.
   - **K-Means:** partitions data into k clusters by minimizing within-cluster variance.
   - **DBSCAN:** density-based clustering — finds arbitrarily shaped clusters and noise points.
   - **Hierarchical Clustering:** builds a tree of clusters (dendrogram).
   - **Applications:** market segmentation, gene expression, document grouping.
2. **Dimensionality Reduction:** Compress data while preserving essential structure.
   - **PCA (Principal Component Analysis):** linear projection maximizing variance.
   - **t-SNE, UMAP:** nonlinear techniques for visualization and manifold learning.
   - **Applications:** visualization of embeddings, noise reduction, preprocessing before ML.
3. **Anomaly Detection:** Identify observations that deviate from the majority.
   - **Isolation Forest, One-Class SVM, Autoencoders.**
   - **Applications:** fraud detection, cybersecurity, quality control.

Beyond Supervision — Hybrid and Modern Approaches

1. **Semi-Supervised Learning**
   - Combines a small labeled dataset with a much larger unlabeled one.
   - Model learns structure from unlabeled data to improve performance on labeled examples.
   - **Examples:** image classification with few labeled examples, text classification with large corpora.
   - **Techniques:** pseudo-labeling, label propagation, consistency regularization.
2. **Self-Supervised Learning**
   - Model generates its own supervision signal from data itself — "labels without labeling."
   - Pretext tasks help learn general representations.
   - **Examples:**
     - Masked token prediction (BERT).
     - Predicting the next video frame or rotation angle (Vision Transformers).
   - **Applications:** NLP (GPT, BERT), computer vision, speech models.
3. **Reinforcement Learning**
   - Learning via interaction — agent receives rewards for actions in an environment.
   - **Goal:** maximize cumulative reward over time.
   - **Applications:** robotics, gaming (AlphaGo), resource allocation, trading systems.

Agenda

Agenda

Linear Regression (OLS)

Definition: Finds a linear relationship between input variables $x_1, x_2, \ldots, x_p$ and output y.

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$$

Goal: minimize residual sum of squares (RSS)

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Assumptions: linearity, independence, constant variance, no multicollinearity, normal errors.
Intuition: fits a straight (hyper)plane through the data points that best explains y.
Use cases: predicting prices, wages, demand, risk scores.
Limitations: poor with nonlinear relationships or outliers.

Regularized Linear Models (Ridge, Lasso, Elastic Net)

Purpose: prevent overfitting by penalizing large coefficients.

$$\min_{\beta} \sum (y_i - \hat{y}_i)^2 + \lambda \left[\alpha \sum |\beta_j| + (1-\alpha) \sum \beta_j^2 \right]$$

Ridge (L2): shrinks coefficients toward zero (no elimination). **Lasso (L1):** can shrink some β_j exactly to zero $\rightarrow$ feature selection. **Elastic Net:** combination of both penalties.
When to use:

- Ridge $\rightarrow$ multicollinearity, many correlated predictors.
- Lasso $\rightarrow$ sparse, interpretable models.
- Elastic Net $\rightarrow$ balance between both.

Example: predicting housing prices with 100+ correlated features.

Regression Trees and Random Forests

Decision Tree Regression:

- Splits data into regions by thresholding features.
- Each leaf contains the mean target value of that region.
- Nonlinear, interpretable, handles mixed data types.

Random Forest:

- Ensemble of many trees trained on random subsets (bagging).
- Final prediction = average of all trees.
- Reduces variance $\rightarrow$ more stable and accurate.

Pros: handles nonlinearity, robust to outliers, feature importance. **Cons:** less interpretable than single tree, slower for large datasets.
Example: predicting energy consumption or insurance claims.

Gradient Boosting (XGBoost, LightGBM, CatBoost)

Idea: build models sequentially — each new tree corrects errors of the previous ensemble.

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta \cdot f_t(\mathbf{x}_i)$$

where f_t is a shallow regression tree and η is the learning rate.
Key points:

- Combines many weak learners into a strong one.
- Handles missing data, categorical features, and nonlinear patterns.
- Tunable parameters: learning rate, tree depth, number of estimators.

Pros: top performance on structured/tabular data. **Cons:** sensitive to overfitting, requires careful tuning.
Applications: credit scoring, forecasting, marketing response models.

Support Vector Regression (SVR)

Concept: find a function $f(x)$ that approximates y with tolerance ϵ, minimizing model complexity.

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t. } |y_i - (w^T x_i + b)| \leq \epsilon$$

Kernel trick: allows nonlinear regression by mapping inputs to higher dimensions.
Pros: flexible via kernels (linear, RBF, polynomial), robust to outliers. **Cons:** computationally expensive, sensitive to scaling.
Typical uses: financial time series, temperature prediction, bioinformatics.

Neural Networks for Regression (MLP)

Architecture: layers of interconnected neurons transforming inputs through nonlinear activations:

$$\hat{y} = f(W_2 \cdot \sigma(W_1 \mathbf{x} + b_1) + b_2)$$

Training: optimize weights via backpropagation and gradient descent.
Properties:

- Can approximate complex nonlinear relationships.
- Require feature scaling and large data.
- Sensitive to architecture (layers, neurons, activations).

Pros: very flexible, supports interactions and nonlinearities. **Cons:** less interpretable, prone to overfitting, requires tuning.
Use cases: stock market regression, sales forecasting, sensor data prediction.

Agenda

Logistic Regression

Definition: models probability of belonging to class $y = 1$ using the logistic (sigmoid) function:

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta^T \mathbf{x})}}$$

Interpretation: coefficients β_j represent the log-odds effect of x_j.
Decision rule: predict class 1 if $P(y = 1|\mathbf{x}) > 0.5$ (or other threshold).
Pros: interpretable, fast, probabilistic output. **Cons:** assumes linear boundary, sensitive to multicollinearity.
Applications: churn prediction, credit scoring, medical diagnostics.

k-Nearest Neighbors (KNN)

Idea: classify a point based on the majority label among its k nearest neighbors.
Steps:

1. Compute distances between new point and all training points.
2. Choose the k closest.
3. Assign the most frequent class among them.

Pros: simple, non-parametric, adaptable. **Cons:** slow for large datasets, sensitive to scaling and irrelevant features.
Applications: pattern recognition, recommender systems, anomaly detection.

Support Vector Machine (SVM)

Goal: find a hyperplane that maximizes the margin between classes.

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t. } y_i(w^T x_i + b) \geq 1$$

Kernel trick: projects inputs into higher dimensions to separate non-linear data (RBF, polynomial).
Pros: effective in high dimensions, robust. **Cons:** not ideal for very large datasets, requires tuning.
Applications: text classification, bioinformatics, image recognition.

Decision Trees and Random Forests

Decision Tree:

- Splits features to maximize purity (e.g., Gini, entropy).
- Simple to interpret, visualizable.

Random Forest:

- Ensemble of trees on random feature and data subsets (bagging).
- Improves accuracy, reduces variance.

Pros: handles nonlinearities, robust, ranks feature importance. **Cons:** less interpretable when many trees.
Applications: fraud detection, credit risk, churn prediction.

Gradient Boosting for Classification

Concept: Sequentially train trees to correct previous errors.

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + \eta \cdot f_t(x)$$

where f_t minimizes the gradient of the loss function (e.g., log-loss).
Popular frameworks: XGBoost, LightGBM, CatBoost.
Pros: high accuracy, handles mixed features, feature importance. **Cons:** tuning required, less interpretable.
Applications: marketing response, customer segmentation, risk prediction.

Neural Networks for Classification

Architecture: layers of neurons transforming inputs into class probabilities.

$$P(y|\mathbf{x}) = \text{softmax}(W_2 \cdot \sigma(W_1\mathbf{x} + b_1) + b_2)$$

Common types:

- **MLP:** tabular data.
- **CNN:** image data.
- **RNN / Transformer:** sequential data (text, time series).

Pros: captures complex nonlinear patterns. **Cons:** requires large data, hyperparameter tuning, interpretability challenges.
Applications: NLP, vision, speech, fraud detection.

Agenda

The ML Workflow (High-Level)

1. Problem framing & success criteria
2. Data collection & rights/compliance
3. EDA & preprocessing (cleaning, encoding, scaling)
4. Split: train / validation / test
5. Model training & hyperparameter tuning
6. Evaluation & error analysis
7. Deployment & monitoring (concept drift)

Data Preparation Essentials

- **Missing values**: drop vs. impute (median/mean/knn/mice).
- **Categoricals**: one-hot vs. ordinal vs. target encoding (watch leakage).
- **Scaling**: standardization (z-score), min–max; required for KNN/SVM.
- **Outliers**: robust scalers, trimming/winsorization.
- **Text**: tokenization, TF–IDF, embeddings.
- **Time series**: lag features, rolling stats; use time-aware splits.

Train / Validation / Test

- **Train**: fit model parameters.
- **Validation**: tune hyperparameters, early stopping.
- **Test**: unbiased final estimate; use once.

Common splits: **70/15/15**, **80/10/10**; stratify for classification, **group splits** to avoid leakage across entities.

Cross-Validation (CV) Variants

Goal: Evaluate how well a model generalizes to unseen data by repeatedly training and testing on different data splits.

Why do we need CV?

- Provides a more **reliable estimate** of out-of-sample performance.
- Makes **better use of limited data** (every observation is used for both training and validation).
- **Reduces variance** of evaluation metrics compared to a single train–test split.

Main Cross-Validation Schemes:

- **K-Fold Cross-Validation:**
  - Data split into k equal folds (typically $k = 5$ or 10).
  - Model trained on $k - 1$ folds, validated on the remaining one.
  - Repeat k times $\rightarrow$ average the metric.
  - **Stratified K-Fold** ensures class balance (for classification).
- **Group K-Fold:**
  - Used when samples belong to the same group (e.g., patients, users, stores).
  - Ensures that all observations from the same group appear in only one fold.
  - Prevents **data leakage** across groups.

Data Leakage: What It Is & How to Avoid It

Definition: **Data leakage** occurs when information from outside the training data — especially from the validation or test set — is used (directly or indirectly) during model training. *As a result, the model performs unrealistically well during training but fails on truly unseen data.*

Why it's dangerous:

- Leads to overly optimistic performance metrics.
- Fails catastrophically in production, when real future data arrive.
- Can be subtle — often happens through preprocessing or feature engineering steps.

Common Types of Leakage:

- **1. Preprocessing Leakage:** When scaling, encoding, or imputing is fit on the full dataset before splitting. *Example:* Standardizing features using mean/SD from all data → future info leaks into train. **Fix:** Always fit preprocessing only on the training set, then apply to validation/test.
- **2. Target Leakage:** When a feature includes information derived from the target variable y. *Example:* Using "number of days overdue" to predict loan default — this value is only known *after* default. **Fix:** Remove or restrict features unavailable at prediction time.
- **3. Time Leakage:** When temporal order is ignored and future data are used to predict the past. *Example:* Randomly shuffling a time series before train–test split, so the model sees future patterns. **Fix:** Use time-based splits (`TimeSeriesSplit`), always preserve chronology.

Best Practices to Prevent Leakage:

Agenda

Agenda

Regression Metrics: Meaning & Interpretation

Goal: Quantify how close predicted values $\hat{y}$ are to the true outcomes y.

1. Mean Absolute Error (MAE) Average magnitude of errors, ignoring direction.

$$\text{MAE} = \frac{1}{n}\sum |y_i - \hat{y}_i|$$

Interpretation: "On average, my predictions are off by X units." *Notes:* easy to interpret, less sensitive to outliers.

2. Mean Squared Error (MSE) & Root MSE (RMSE) Square the errors before averaging $\rightarrow$ penalizes large deviations.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum (y_i - \hat{y}_i)^2}$$

Interpretation: emphasizes big mistakes; good for continuous forecasting (e.g., energy, prices). *Units:* same as target variable.

3. Coefficient of Determination (R^2) Fraction of variance in y explained by the model.

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

Interpretation: $R^2 = 0.8$ means 80% of the variability in y is captured. *Caution:* can be negative on test data $\rightarrow$ poor generalization.

Key insight: Choose metrics that reflect your real-world cost of error (e.g., absolute vs. squared losses).

Classification Metrics: Understanding the Confusion Matrix

Setup: Predictions fall into four possible outcomes:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Interpretation of Core Metrics:

- **Accuracy** — overall correctness *Proportion of all correct predictions. Caution:* misleading with imbalanced data.
- **Precision** — trust in positive predictions "When the model says positive, how often is it right?" *High precision → few false alarms.*
- **Recall (Sensitivity)** — completeness of positive detection "Of all real positives, how many did we catch?" *High recall → few missed cases.*
- **F1 Score** — balance between precision and recall Harmonic mean; useful when false positives/negatives are both costly.

Example: In medical screening, recall matters more (catch every patient); in spam filtering, precision matters more (don't flag legit mail).

ROC and Precision–Recall Curves

Why curves? Many classifiers output probabilities, not fixed labels — we can adjust the **threshold** to trade off precision vs. recall.

ROC Curve (Receiver Operating Characteristic):

- Plots True Positive Rate (Recall) vs. False Positive Rate.
- **AUC (Area Under Curve)** = probability that a random positive ranks above a random negative.
- **Interpretation:** 0.5 = random guessing, 1.0 = perfect ranking.
- **Use:** balanced datasets, general model ranking.

Precision–Recall (PR) Curve:

- Plots Precision vs. Recall as threshold varies.
- **PR-AUC:** average precision across thresholds.
- **Use:** when the positive class is rare (e.g., fraud, disease detection).
- **Interpretation:** focuses on performance for minority class.

Key takeaway: ROC-AUC evaluates ranking ability; PR-AUC focuses on positive detection quality. Always inspect both the curve *and* the chosen operating threshold.

Imbalanced Data: The Problem

Definition: An **imbalanced dataset** occurs when one class (usually the "positive" or "event" class) is much rarer than the other — e.g. fraud detection (1%), disease diagnosis (5%), or customer churn (10%).

Why it matters:

- A model can achieve **high accuracy** by always predicting the majority class — yet fail completely on the minority class.
- **Accuracy becomes misleading**; metrics should reflect business goals (e.g., recall for fraud detection).
- The model might never "learn" minority patterns because they contribute too little to the overall loss.

Example: If only 1% of transactions are fraudulent, a model that always predicts "no fraud" has 99% accuracy — but catches **zero** frauds. → High accuracy good model.

Intuition: The imbalance problem is not about math — it's about **signal vs. noise**. The model sees too few examples of the rare class to generalize well.

Imbalanced Data: Key Strategies

Goal: Help the model learn meaningful patterns from the minority class — not just increase numerical balance.

1. **Resampling Techniques**
   - **Oversampling** — duplicate or synthetically generate new minority class samples. *Example:* SMOTE (Synthetic Minority Oversampling Technique) creates new samples by interpolation between neighbors.
   - **Undersampling** — randomly remove majority samples to reduce imbalance. *Trade-off:* may discard valuable data, works best for large datasets.

2. **Class Weights in Loss Function**
   - Penalize misclassifying the minority class more heavily.
   - Implemented via `class_weight="balanced"` in scikit-learn models.
   - Forces algorithm to treat minority errors as more costly.

3. **Appropriate Evaluation Metrics**
   - Avoid accuracy — use **Precision, Recall, F1, ROC-AUC, PR-AUC, Recall@k**.
   - Match metric choice to your business cost function (e.g., fraud recall vs. false alarms).

4. **Data Splitting & Workflow Integrity**
   - Always use **stratified train/test splits** to preserve class ratios.
   - Apply resampling **only on the training set** to prevent leakage.
   - Integrate steps into a **Pipeline** for consistent cross-validation.

Key insight: The objective is not a 50/50 balance, but to make the model **recognize and learn minority signals effectively**.

Agenda

Underfitting vs. Overfitting

Key idea: Machine Learning models must balance **bias** (too simple) and **variance** (too complex). This trade-off determines how well the model generalizes to unseen data.

Underfitting — high bias, low variance

- Model is too simple to capture underlying patterns in the data.
- Both training and test errors are high.
- Example: fitting a straight line to data that's clearly quadratic.
- *Symptoms:* flat learning curve; training accuracy plateaus early.
- *Causes:* model too shallow, too few features, too strong regularization.
- *Fixes:* add complexity — more features, higher model capacity, reduce regularization.

Overfitting — low bias, high variance

- Model fits training data **too well**, including noise and random fluctuations.
- Training error is very low, but test error is high.
- Example: deep decision tree memorizing every training point.
- *Symptoms:* excellent train metrics, poor validation metrics.
- *Causes:* excessive model complexity, small dataset, data leakage.
- *Fixes:* regularization, pruning, dropout, early stopping, cross-validation.

Goal: Find the **"sweet spot"** — a model that's complex enough to learn true structure but simple enough to generalize well to new data.

Fixes for Overfitting

Overfitting recap: A model overfits when it learns not only the underlying patterns but also the **noise** in the training data. It performs very well on the training set but poorly on unseen (validation/test) data.

Symptoms:

- Training accuracy is much higher than validation accuracy.
- Loss continues to decrease on training data but increases on validation data.
- Model predictions fluctuate drastically with small data changes.

Main Strategies to Prevent or Fix Overfitting:

1. **Regularization — penalizing model complexity**
   - **L2 (Ridge):** adds a penalty $\lambda \sum \beta_j^2$ → shrinks coefficients smoothly toward zero.
   - **L1 (Lasso):** adds a penalty $\lambda \sum |\beta_j|$ → forces some coefficients to zero (feature selection).
   - **Elastic Net:** combines both — useful with correlated predictors.
   - For **neural networks:** use **dropout**, randomly deactivating neurons during training to prevent co-adaptation.
   - For **trees:** apply **pruning** — limit depth, minimum samples per split, or remove weak leaves.
2. **More Data or Data Augmentation**
   - The simplest and most effective cure for variance.
   - **More samples** → model learns the general pattern rather than memorizing few examples.
   - **Data augmentation:** synthetically increase dataset size (e.g., image flips, noise

Bias–Variance Trade-off

Goal: Understand the two main sources of prediction error and how they interact.

1. **Bias — Systematic Error (Underfitting side)**
   - Comes from overly simple assumptions about the data.
   - Model cannot capture the true relationship between features and target.
   - Predictions are consistently off in one direction — systematic mistakes.
   - *Example:* fitting a straight line to curved (nonlinear) data.
   - **High bias** → low model flexibility, poor training and test performance.

2. **Variance — Sensitivity to Noise (Overfitting side)**
   - Comes from fitting noise or random fluctuations in the training data.
   - Small changes in data → large changes in predictions.
   - *Example:* deep tree that perfectly fits training points but fails on new ones.
   - **High variance** → great training accuracy, poor generalization.

3. **The Trade-off**
   - Increasing model complexity usually **reduces bias** but **increases variance**.
   - Simpler models are stable but may miss structure; complex models capture detail but may memorize noise.
   - Optimal performance lies at the **"sweet spot"** — where total error (bias^2 + variance + irreducible noise) is minimal.

4. **How to Manage the Trade-off**
   - Use **cross-validation** to detect overfitting early.
   - Apply **regularization** (e.g., Lasso, Ridge, dropout).
   - Gather more data or add noise robust features

Agenda

Hyperparameter Tuning (Concept & Motivation)

Definition: **Hyperparameters** are model settings that control the learning process (e.g., tree depth, learning rate, regularization strength) and are **not learned from data** directly. They must be chosen or optimized externally.

Why tuning matters:

- Determines how well a model generalizes — wrong hyperparameters cause under/overfitting.
- Impacts training speed, stability, and performance.
- Often the difference between an "okay" model and a winning one.

Typical examples:

- Linear models — regularization strength λ (Ridge/Lasso).
- Tree models — max depth, number of trees, learning rate.
- Neural networks — number of layers, neurons, learning rate, dropout rate.

Key insight (preview): Hyperparameter tuning is about finding the **best bias–variance balance** for your model, not just the lowest validation error on one split.

Hyperparameter Tuning (Search & Validation Workflow)

Search Strategies for Hyperparameter Optimization:

- **Grid Search** — test every combination from a predefined grid. *Pros:* exhaustive and simple; *Cons:* computationally expensive. *Example:* try all (depth $\in [3, 5, 7]$) $\times$ (learning rate $\in [0.01, 0.1, 0.3]$).
- **Random Search** — sample combinations randomly within a range. *Pros:* faster, good for high-dimensional spaces (often near-optimal with fewer trials).

Cross-validation during tuning:

- Use **Nested Cross-Validation** to avoid overfitting the validation set — inner loop for tuning, outer loop for unbiased performance estimation.
- Always integrate preprocessing in a **Pipeline** to ensure no data leakage (scaling, encoding, or imputation must be fit only on the training folds).

Key insight: Hyperparameter tuning is about finding the **best bias–variance balance** for your model, not just the lowest validation error on one split.

Feature Engineering (Definition & Motivation)

Definition: **Feature Engineering** is the process of transforming raw data into meaningful input variables (*features*) that better represent the underlying patterns of the problem for the learning algorithm.

Why it matters:

- Well-designed features can dramatically improve model performance — sometimes more than switching algorithms.
- ML algorithms are only as good as the information encoded in the input variables.
- It injects domain knowledge, structure, and interpretability into the data.

1. Domain-Driven Features

- Use subject-matter expertise to create variables that reflect meaningful relationships.

  Examples:

  - In finance → ratios (debt-to-income, price-to-earnings).
  - In retail → recency/frequency/monetary (RFM) features.
  - In healthcare → BMI = weight / height^2.
- Domain-driven transformations often outperform complex model tweaks.

Key message: *Better features often beat better algorithms.*

Feature Engineering (Interactions & Polynomials)

2. Interaction and Polynomial Features

- Capture relationships between variables that linear models may miss. *Example:* adding $x_1 \times x_2$ or x_1^2 to capture interaction or nonlinearity.
- Use with caution — high-degree polynomials can lead to overfitting or unstable coefficients.
- Regularization (Ridge/Lasso) helps control complexity when adding such terms.

3. Encoding Categorical Variables

- **One-hot encoding** — for low-cardinality categorical variables.
- **Target / Mean encoding** — for high-cardinality features (replace each category with mean of target variable). **Important:** must be **cross-validation aware** — compute means using only the training fold to avoid leakage.

Pro tip: Categorical encodings can be a major source of leakage — always use pipelines to isolate training transformations.

Feature Engineering (Temporal, Text, Numeric Data)

4. **Handling Temporal, Text, and Numeric Data**
   - **Time-based features:** day of week, seasonality, lag features, moving averages.
   - **Text data:** TF-IDF, word embeddings, sentiment scores.
   - **Numeric transformations:** logs, scaling, normalization, winsorization (handle outliers).

Example applications:

- Forecasting sales → include lag sales, holiday dummy, and rolling mean.
- Sentiment analysis → extract word frequencies or sentiment polarity.
- Regression with skewed data → apply log transform to reduce scale dominance.

Practical insight: Different data types demand different preprocessing — combine numerical, categorical, and temporal pipelines carefully.

Feature Engineering (Leakage Prevention & Takeaways)

5. Leakage Prevention

- Never use information that would not be available at prediction time (e.g., future data, target-related stats).
- Common leakages: aggregations including the test fold, or target-encoded features computed on full data.
- Always perform transformations **inside pipelines** fitted on training data only.

Key insight: Good features turn simple models into powerful predictors — bad features can make even the most advanced algorithms fail.

Checklist for robust feature engineering:

- Think domain-first: what signals matter?
- Validate transformations only with training folds.
- Track feature importance and drop noisy ones.
- Document every step for reproducibility.

Remember: feature engineering is not a one-time task — it's an iterative process alongside model development.

Agenda

Interpretability & Explainability (Core Concepts)

Why interpretability matters:

- Understanding **why** a model made a certain prediction builds trust with users and stakeholders.
- Essential in **high-stakes domains** — finance, healthcare, law, public policy — where decisions must be explainable.
- Supports **debugging**, **bias detection**, and **regulatory compliance**.

Two main perspectives:

- **Global interpretability:** understanding the model's general behavior across the whole dataset. *Examples:*
  - Coefficients in a linear/logistic regression (direction and magnitude of effect).
  - Feature importances in tree-based models (e.g., Random Forest, XGBoost).
  - Partial Dependence Plots (PDP) — show how the average prediction changes with one variable.
  - Accumulated Local Effects (ALE) — similar to PDP but avoids feature correlation bias.
- **Local interpretability:** understanding individual predictions. *Examples:*
  - **LIME (Local Interpretable Model-agnostic Explanations):** approximates the model locally with a simple surrogate (e.g.,

Interpretability Tools (Examples & Visualization)

1. **Global understanding**
   - **Feature importance plot:** shows which variables contribute most to model predictions overall.
   - **Partial Dependence Plot (PDP):** reveals the marginal effect of a feature on the average prediction.
   - **SHAP summary plot:** combines global + local information; each dot represents a single observation's SHAP value.

2. **Local explanations**
   - **LIME:** perturb an instance's inputs, observe output changes, and fit a local surrogate model.
   - **SHAP:** compute contribution values per feature for one prediction; positive = pushes prediction up, negative = down.

Example scenario: Credit scoring model predicts loan default probability.

- Global: income and credit history are most important features.
- Local: for one applicant, high debt ratio (+) and short employment history (+) drive risk upward.

Key insight: Interpretability bridges the gap between **prediction** and **understanding**. It turns a "black box" into a transparent, accountable tool.

Fairness & Responsible ML (Essentials)

Why fairness matters:

- ML systems can unintentionally **amplify existing biases** in data.
- Biased outcomes can harm underrepresented groups — e.g., loan approval, hiring, or healthcare access.
- Fairness is not only ethical but often a **legal requirement**.

Sources of bias:

- **Data bias:** historical inequities or unbalanced representation in training data.
- **Measurement bias:** target labels reflect past prejudice (e.g., arrest records, not true crime).
- **Selection bias:** model trained on non-representative samples (e.g., only certain demographics).

Fairness metrics (context-specific):

- **Group metrics:** compare True Positive Rate (TPR), False Positive Rate (FPR), etc., across groups.
- **Equalized odds:** requires equal TPR and FPR across protected groups.
- **Demographic parity:** equal probability of positive prediction across groups (may conflict with accuracy).
- Always interpret metrics **in context** — fairness definitions can trade off with performance.

SHAP values

Use SHAP when: you need per-instance explanations with additivity & consistency.

- **Tree models** $\Rightarrow$ **TreeSHAP** (fast, exact for many objectives).
- **Other models** $\Rightarrow$ **KernelSHAP** (sampled, slower) or **DeepSHAP** (NNs).

Visuals to produce:

- **Summary (beeswarm)** for global importance & sign.
- **Dependence** (+ color by 2nd feature) for nonlinearity/interactions.
- **Force/Waterfall** for single-case narratives.

Common pitfalls:

- Correlations confound attribution — consider grouping or domain consolidation.
- Baseline choice changes narratives — pick a representative background.
- SHAP is not causality — pair with study design if you need causes.

Deliverables to stakeholders: 2–3 plots + short textual narrative per use-case.

Agenda

Mini-Example 1: Regression Flow

Goal: predict house prices.
Steps:

1. Train/val/test split (stratify by price quantiles optional).
2. Preprocess: impute, one-hot encode, scale numeric.
3. Baseline: median predictor $\rightarrow$ compare RMSE.
4. Linear model (Ridge/Lasso) vs. Random Forest vs. Gradient Boosting.
5. Tune hyperparameters via CV; check learning curves.
6. Error analysis by segments (size, location).

Mini-Example 2: Classification Flow

Goal: spam detection.
Steps:

1. Text preprocessing: tokenization, TF–IDF.
2. Split (stratified); consider imbalance.
3. Baseline: majority class.
4. Train Logistic Regression, SVM, Naïve Bayes.
5. Evaluate: ROC-AUC, PR-AUC; choose decision threshold for target Recall.
6. Calibrate probabilities if used for risk scoring.

Agenda

Common Pitfalls

- **Leakage** via preprocessing on full dataset.
- Using **test set for tuning**.
- Ignoring **class imbalance** or wrong metrics.
- **Data shift** between train and production.
- Over-interpreting feature importances as causality.

ML Project Checklist (Quick)

- Clear problem & metric aligned with impact.
- Data rights, privacy, consent verified.
- Pipeline with CV-safe preprocessing.
- Baselines + ablations (what truly helps).
- Robust evaluation, error analysis by segments.
- Documentation & monitoring plan.

Agenda

Key Takeaways

- ML learns patterns from data; generalization is king.
- Supervised vs. Unsupervised: different goals and tools.
- Workflow: data $\rightarrow$ split $\rightarrow$ CV $\rightarrow$ evaluate.
- Metrics matter; pick ones reflecting *costs*.
- Manage complexity (regularization), avoid leakage, validate properly.

Further Resources

- **Books**: Murphy (Probabilistic ML), Bishop (PRML), Géron (Hands-On ML).
- **Courses**: Andrew Ng (Coursera), fast.ai (practical DL).
- **Libraries**: scikit-learn docs; XGBoost/LightGBM; shap, lime.
- **Responsible AI**: Model cards, fairness guidelines (ACM/IEEE).

Questions?