Systematic Trading Strategies with Machine Learning Algorithms

Introduction to Unsupervised Learning Techniques

May 1, 2025

Outline

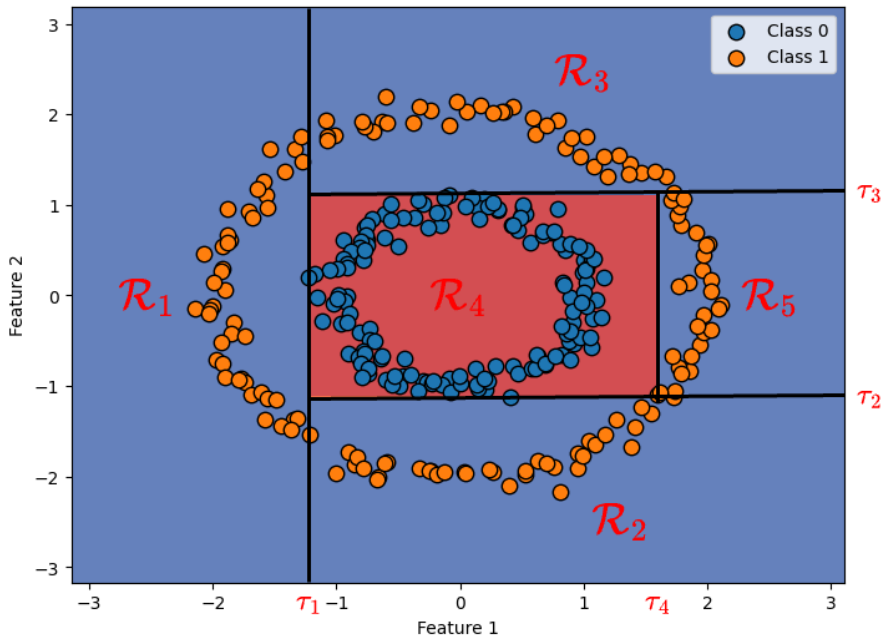
Outline

Outline

Decision Trees - A Visual Example

Decision trees create recursive binary partitioning of the feature space.

- Decision tree algorithm:
  - Divides the input space into regions
  - Each split based on a single feature
  - Predictions are made based on region assignment
- *The example shows classification on ring-shaped data using simple thresholds.*



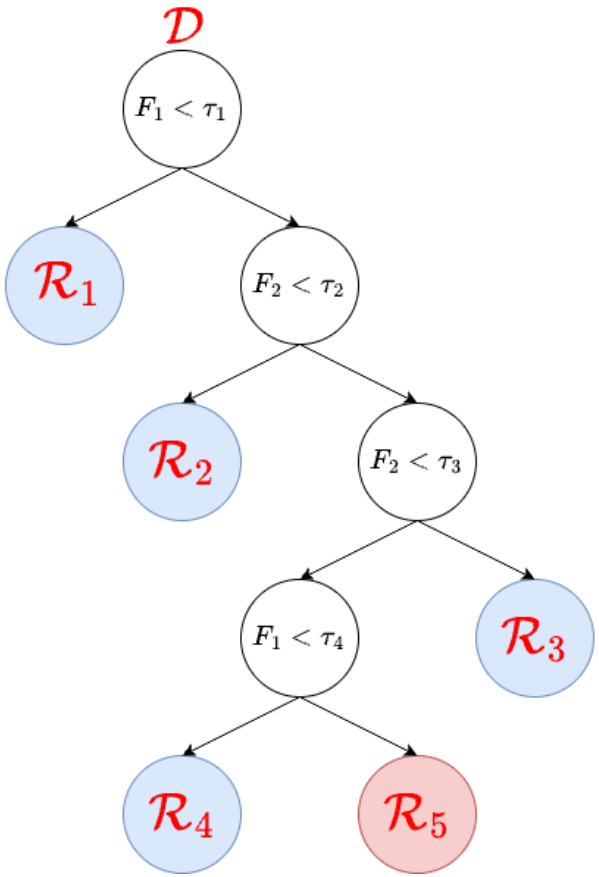
Imperial College Business School

How decision trees map inputs to outputs:

- A decision tree maps input $F \in \mathbb{R}^d$ to output y using binary decision rules:
  - Each node has a splitting rule
  - Each leaf node is associated with an output value
- Each splitting rule is of the form:

$$h(F) = \mathbf{1}\{F_j > \tau\}$$

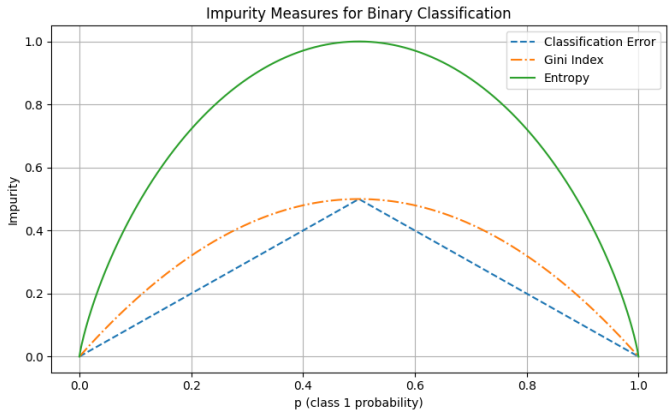
  for some dimension j of F and $\tau \in \mathbb{R}$
- Using these transition rules, a path to a leaf node gives the prediction
- The leaves define regions called $\mathcal{R}_m$



Imperial College Business School

Impurity measure

- For all $F \in \mathcal{R}$, let p_k be empirical fraction labeled k in this region.
- Measures of impurity for region $\mathcal{R}$:
  1. Classification error: $1 - \max_k p_k$
  2. Gini index: $\sum_{k=1}^{K} p_k(1 - p_k)$
  3. Entropy: $-\sum_{k=1}^{K} p_k \log p_k$
- *Impurity is maximized when classes are evenly distributed.*
- *Impurity is minimized when a region contains only one class.*



Information Gain and Decision Tree Algorithm

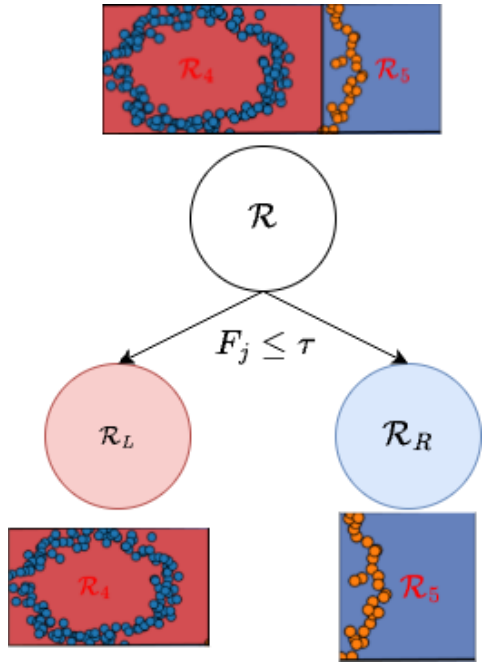
Imperial College Business School

- **Information Gain** for feature j and threshold τ:

$$IG(j,\tau) = I(\mathcal{R}) - \frac{|\mathcal{R}_L|}{|\mathcal{R}|} I(\mathcal{R}_L) - \frac{|\mathcal{R}_R|}{|\mathcal{R}|} I(\mathcal{R}_R)$$

Where:

  - $\mathcal{R}_L = \{F \in \mathcal{R} : F_j \leq \tau\}$
  - $\mathcal{R}_R = \{F \in \mathcal{R} : F_j > \tau\}$



- The DT algorithm:
  1. For each feature j and possible threshold τ, compute $IG(j,\tau)$
  2. Select feature j^* and threshold τ^* that maximize IG
  3. Split node and create child regions $\mathcal{R}_L$ and $\mathcal{R}_R$
  4. Recursively apply to each child node until stopping criteria met

Algorithm Decision Tree Learning Algorithm

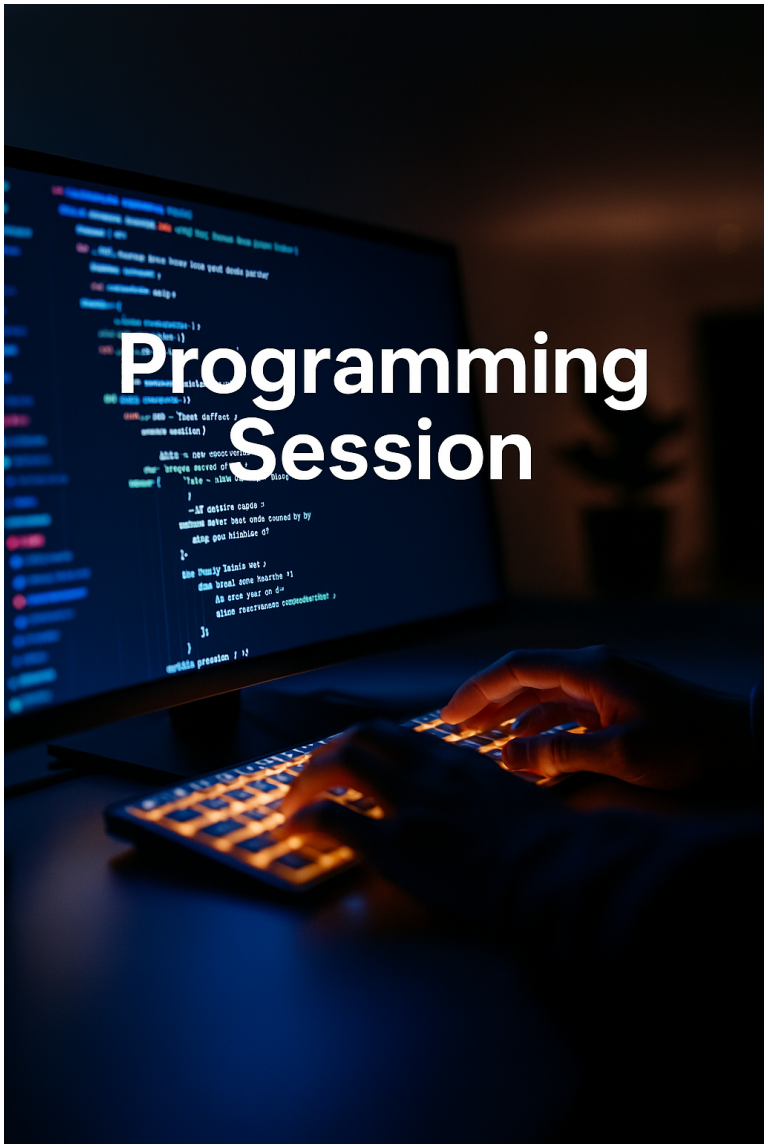
Require: Training data $\{(F_i, y_i)\}_{i=1}^{n}$, stopping criteria

Ensure: Decision tree T

Initialize tree with single root node containing all data
while nodes can be split and stopping criteria not met **do**
    for each leaf node with region $\mathcal{R}$ **do**
        Find (j^*, τ^*) that maximizes:
        $IG(j, \tau) = I(\mathcal{R}) - \frac{|\mathcal{R}_L|}{|\mathcal{R}|} I(\mathcal{R}_L) - \frac{|\mathcal{R}_R|}{|\mathcal{R}|} I(\mathcal{R}_R)$
        Where $\mathcal{R}_L = \{F \in \mathcal{R} : F_j \leq \tau\}$ and $\mathcal{R}_R = \{F \in \mathcal{R} : F_j > \tau\}$
        Split node using rule $F_{j^*} > \tau^*$
    end for
end while
Assign prediction to each leaf node (majority class)
return T

Quiz Time!

Click here to take the quiz



Optional Programming Session: Implementation of the DT algorithm

- *Click here to access the programming session*

Content:

- Build a Decision Tree classifier from scratch.
- Recursively split data; leaf nodes store class predictions.
- Generate non-linear data and compare Logistic Regression vs. Decision Tree decision boundaries.

Outline

Mean Decrease Impurity (MDI)

Imperial College Business School

Feature Importance Analysis in Random Forests

- Introduced by *Breiman [2001]* for measuring feature importance in tree-based models.
- Based on the accumulated decrease in impurity across all trees.

Pros

- Fast computation (calculated during training)
- Default method in most libraries
- Provides interpretable values (0-1)

Cons

- In-sample measure only.
- Can assign importance to noise.
- Biased toward high cardinality features

Algorithm Mean Decrease Impurity (MDI) for Random Forests

Require: Trained random forest model with T trees

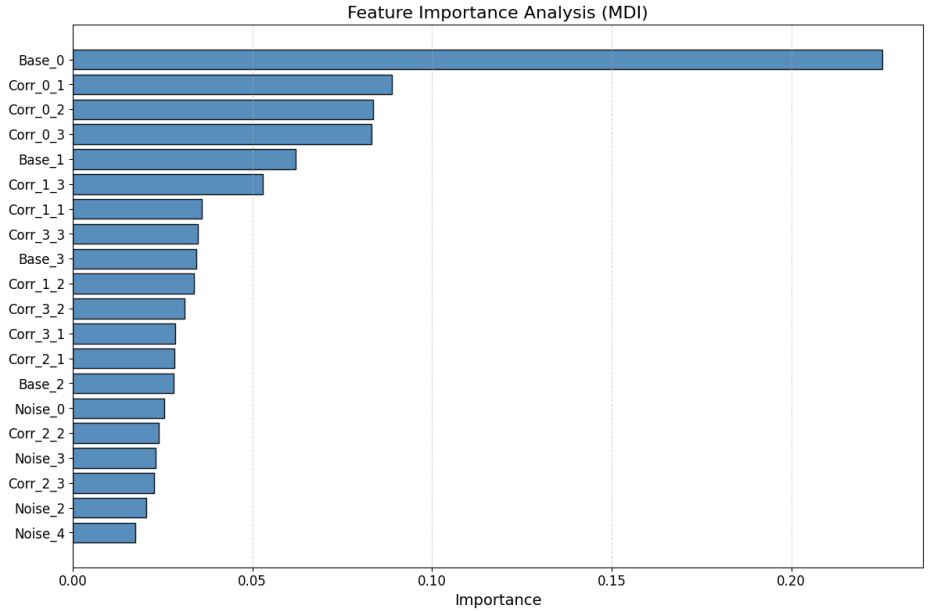
Ensure: Feature importance scores $\{\text{MDI}_j\}_{j=1}^{d}$

Initialize importance scores: $\text{IMP}_j = 0$ for all features j
for each tree $t = 1$ to T in the forest **do**
    for each internal node n in tree t **do**
        Identify the feature F_j used for splitting at node n
        Let w_n be the proportion of samples reaching node n
        Calculate information gain $IG_n(j, \tau_n)$
        Update importance: $\text{IMP}_j = \text{IMP}_j + w_n \cdot IG_n(j, \tau_n)$
    end for
end for
Compute $\text{MDI}_j = \frac{1}{T} \cdot \frac{\text{IMP}_j}{\sum_{k=1}^{d} \text{IMP}_k}$ for all j
return $\{\text{MDI}_j\}_{j=1}^{d}$

Imperial College Business School

Mean Decrease Impurity

- Generated dataset with 1000 samples
- 4 feature clusters with 4 features each:
  - One base feature per cluster (Base_i)
  - Three correlated features per cluster (Corr_i_j)
- 5 pure noise features (Noise_i)
- Ground truth: Only "Base" features directly influence the target



- **MDI Interpretation:**
- Importance is split among correlated features in each cluster
- Some noise features receive non-zero importance

Outline

Permutation Feature Importance

Imperial College Business School

Out-of-Sample Feature Importance

- Measures the decrease in model performance when a feature is randomly shuffled.
- Intuition: If shuffling a feature decreases performance significantly, that feature is important.

Pros

- **Out-of-sample**: Evaluates on validation data.
- **Model-agnostic**: Works with any machine learning model.
- **Metric-flexible**: Compatible with any performance metric.

Cons

- Computationally expensive.
- Results vary with different random permutations.
- Sensitive to feature correlation.

Permutation Feature Importance Algorithm

Imperial College Business School

Algorithm Permutation Feature Importance (PFI)

Require: Fitted model m, validation data D, repetitions K

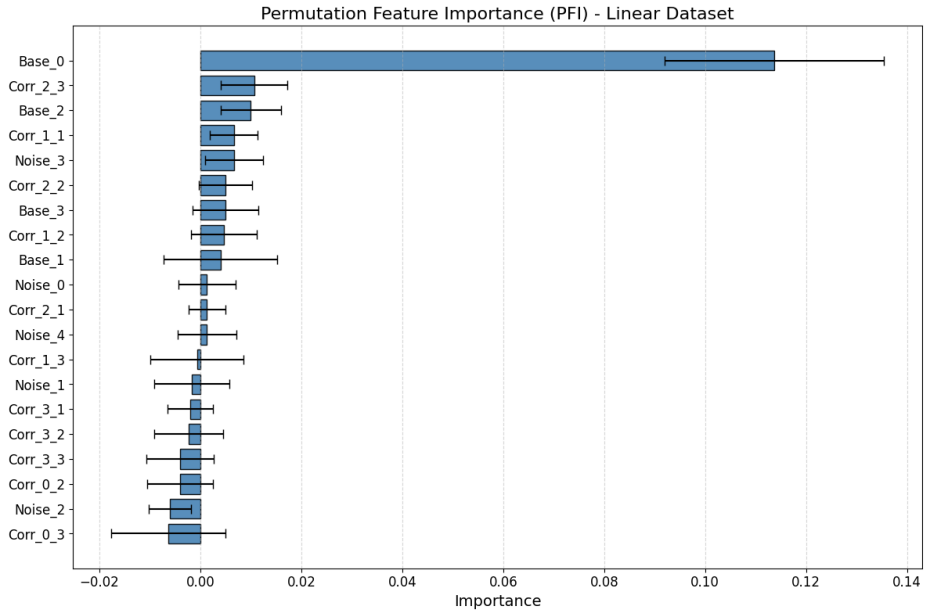
Ensure: Feature importance scores $\{PFI_j\}_{j=1}^{d}$ with stds $\{\sigma_j\}_{j=1}^{d}$

1: Compute reference score s of model m on data D
2: **for** each feature F_j (column of D) **do**
3: Initialize array $scores_j$ of length K
4: **for** each repetition k in $1, \ldots, K$ **do**
5: Randomly shuffle column j of dataset D to generate corrupted version $\tilde{D}_{k,j}$
6: Compute score $s_{k,j}$ of model m on corrupted data $\tilde{D}_{k,j}$
7: Store in array: $scores_j[k] = s - s_{k,j}$
8: **end for**
9: Compute mean importance PFI_j and standard deviation σ_j from array $scores_j$
10: **end for**
11: **return** $\{PFI_j\}_{j=1}^{d}$ and $\{\sigma_j\}_{j=1}^{d}$

Imperial College Business School

Permutation Feature Importance Analysis

- Generated dataset with 1000 samples
- 4 feature clusters with 4 features each:
  - One base feature per cluster (Base_i)
  - Three correlated features per cluster (Corr_i_j)
- 5 pure noise features (Noise_i)
- Ground truth: Only "Base" features directly influence the target

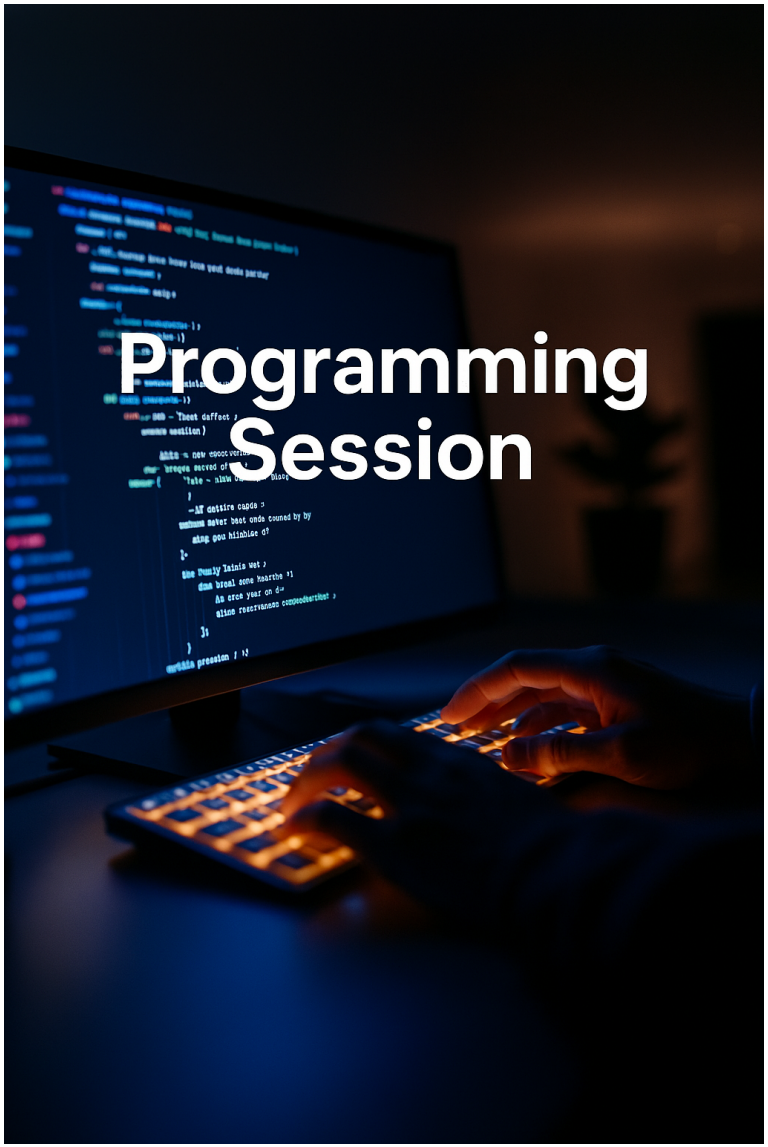


- **PFI Interpretation:**
- More robust to noise features than MDI.
- Importance is underestimated for all correlated features

The Correlation Problem in Feature Importance

Imperial College Business School

- **Issue:** When features are correlated, permuting one doesn't fully remove its information (it's still available through the other).
- This results in smaller performance drops, leading to underestimated importance scores.
- Consequence: correlated features may appear unimportant, even when jointly they matter.
- **Relevance to finance:** Feature are often highly correlated, making this a critical issue in systematic trading strategies.
- **Mitigation strategies:**
  - **Clustering:** group correlated features, select representatives, and perform feature importance at the cluster level.
  - **Dimensionality reduction:** use PCA or autoencoders to extract uncorrelated components before computing importance.



Programming Session 2: Sections 1 and 2

- Section 1: Generating Synthetic Data.
- Section 2: Feature Importance Analysis on Correlated Features.
- *Click here to access the programming session*

Solution will be posted tonight on the GitHub page.

- *Click here to access the GitHub Page*

Feedback Poll

Click here to participate in the poll

Outline

Outline

Motivation for Clustering

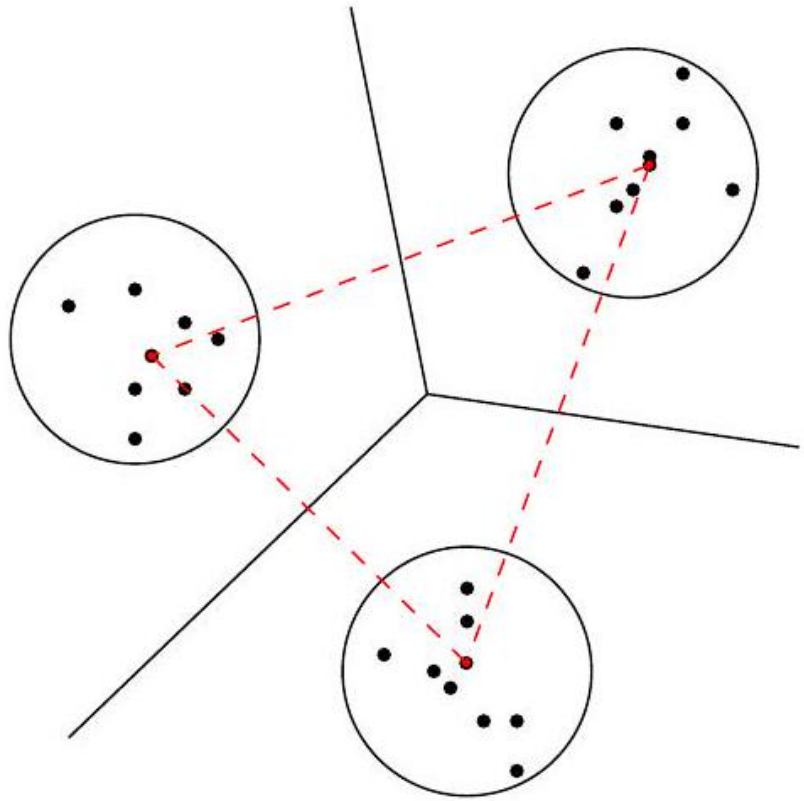
Imperial College Business School

Given a data set
$X = (x_1, \ldots, x_n) \in \mathbb{R}^{n \times p}$ where:

- n is the number of observations
- p is the number of features

Goal: Separate data into K clusters by learning:

- Centroids of each cluster
  $\{c_1, \ldots, c_K\} \in \mathbb{R}^{p \times K}$
- Assignment function
  $\Psi : \{x_1, \ldots, x_n\} \to \{1, \ldots, K\}$
- Meaning: sample x_i belongs to class $\Psi(x_i)$



A simple representation of clustering
$(n = 25, p = 2, K = 3)$

Algorithm The K-means Algorithm

Require: A data set $X = \{x_1, \ldots, x_n\}$ $(x_i \in \mathbb{R}^p)$

Ensure: An assignment function Ψ^* and the associated centroids $c_1^*, \ldots, c_K^*$.

Initialization: Choose $c_1, \ldots, c_K$ in X at random
repeat
  for $i = 1 \ldots n$ **do**
    $\Psi(x_i) \leftarrow \arg\min_{k \in \{1,\ldots,K\}} \|x_i - c_k\|^2$
  end for
  for $k = 1 \ldots K$ **do**
    $c_k \leftarrow \frac{1}{\sum_{i=1}^{n} \mathbb{1}(\Psi(x_i)=k)} \sum_{i=1}^{n} \mathbb{1}(\Psi(x_i) = k)\, x_i$
  end for
until convergence
return $\Psi^*, c_1^*, \ldots, c_K^*$

K-means: Theoretical Properties

Imperial College Business School

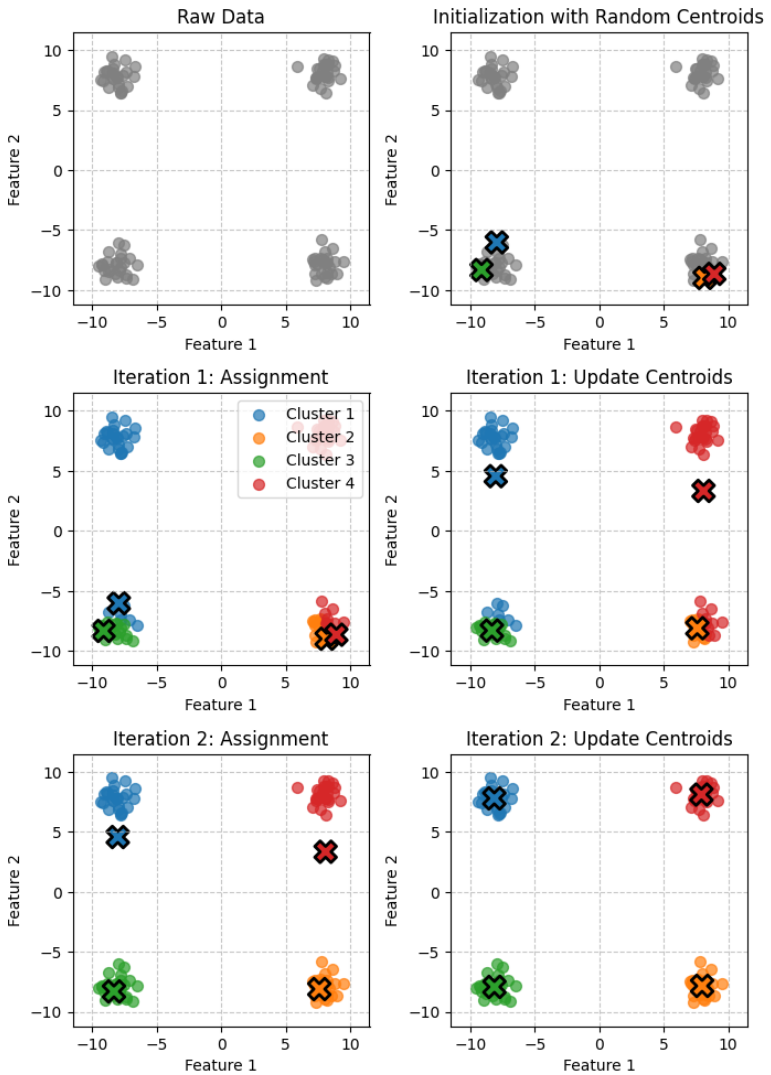
Distortion Measure:

Let Ψ be an assignment function and $c = (c_1, \ldots, c_K)$ be centroids.

$$J(\Psi, c) = \frac{1}{n} \sum_{i=1}^{n} \left\| x_i - c_{\Psi(x_i)} \right\|^2$$

Key Properties:

- K-means monotonically decreases the distortion
- This guarantees convergence to a local minimum
- The algorithm stops after a finite number of steps
- *Optional*: *Click here for the proof*



K-means visualization

Finding the Optimal Number of Clusters

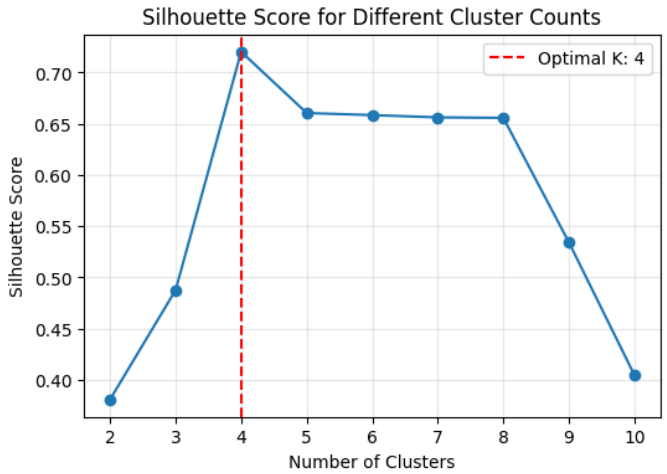
Imperial College Business School

- Determining the appropriate number of clusters (K) is a critical challenge in cluster analysis
- Several validation metrics help identify the optimal K:
  - **Silhouette Score** - Measures cohesion and separation
  - **Calinski-Harabasz Index** - Ratio of between-cluster to within-cluster variance
  - **Davies-Bouldin Index** - Ratio of within-cluster scatter to between-cluster separation
- Approach: Run K-means with different K values, evaluate metrics, select optimal K

Silhouette Score

Principle: Measures how well-separated clusters are
For each sample i:

- $a(i)$ = average distance to other points in the same cluster
- $b(i)$ = average distance to points in the nearest different cluster
- Silhouette value $s(i) = \frac{b(i)-a(i)}{\max(a(i),b(i))}$
- **Selection criteria:** *Higher is better*



Silhouette score visualization for different values of K

Interpretation:

- Range: $[-1, 1]$
- Close to 1: *Well-clustered*, Close to 0: *On cluster boundary*, Close to -1: *Not Well-clustered*.

Outline

Gaussian Mixture Models (GMMs)

Imperial College Business School

Goal:

- Represent complex probability distributions as a mixture of Gaussians

Key advantages:

- Captures multimodal distributions that single Gaussians cannot model
- Provides probabilistic (soft) assignments to clusters
- Adapts to clusters of varying shapes, sizes, and densities

Applications:

- Market regimes in finance
- Clustering with soft assignments

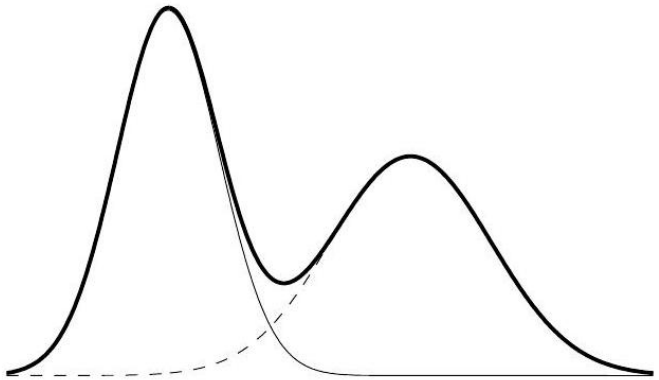
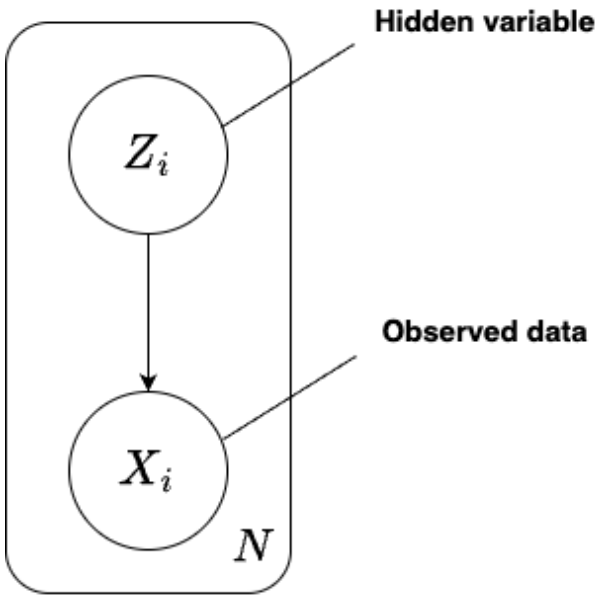


Figure: Density represented as a mixture of two Gaussians

Parameterization

Mathematical Formulation:

- n observations: $X_1, \ldots, X_n \in \mathbb{R}^d$
- Hidden variables: $Z_1, \ldots, Z_n \in \{1, \ldots, K\}$
- For a mixture of K Gaussians:
  - $Z_i \sim \mathcal{M}(1, \pi_1, \ldots, \pi_K)$
  - $(X_i | Z_i = k) \sim \mathcal{N}_d(\mu_k, \Sigma_k)$
  - Parameters $\theta = (\pi, \mu, \Sigma)$



Density function (via marginalization):

$$p_\theta(x_i) = \sum_{z_i} p_\theta(x_i, z_i) = \sum_{z_i} p_\theta(x_i | z_i) p_\theta(z_i) = \sum_{k=1}^{K} p_\theta(x_i | z_i = k) p_\theta(z_i = k)$$

$$= \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i | \mu_k, \Sigma_k)$$

Parameter Estimation:

- Need to learn $\theta = (\pi, \mu, \Sigma)$ using the **Expectation-Maximization algorithm**.
- Introduce a lower bound $\mathcal{L}(q, \theta)$ on the log-likelihood
- EM alternates between two steps:
  - Maximize wrt q (E-step)
  - Maximize wrt θ (M-step)
- Iterative process that guarantees improvement at each step

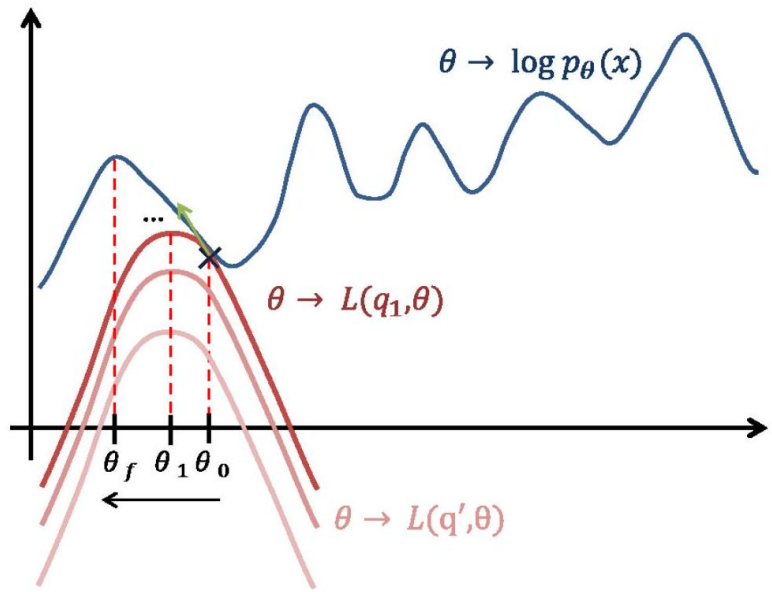


Figure: EM algorithm converging to a local maximum

Lower Bound via Jensen's Inequality:

For n observations with latent variables $z_1, \ldots, z_n$:

$$\begin{aligned}
\log p_\theta(x) &= \sum_{i=1}^{n} \log \left(\sum_{z_i} p_\theta(x_i, z_i) \right) \\
&= \sum_{i=1}^{n} \log \left(\sum_{z_i} q(z_i) \frac{p_\theta(x_i, z_i)}{q(z_i)} \right) \\
&\geq \sum_{i=1}^{n} \sum_{z_i} q(z_i) \log \left(\frac{p_\theta(x_i, z_i)}{q(z_i)} \right) = \mathcal{L}(q, \theta)
\end{aligned}$$

Key Idea:

- **E-step:** Maximize $\mathcal{L}(q, \theta)$ with respect to q with fixed θ
- **M-step:** Maximize $\mathcal{L}(q, \theta)$ with respect to θ with fixed q

Algorithm EM Algorithm

Require: Data set $X = \{x_1, \ldots, x_n\}$

Ensure: Optimal θ

1: **Initialization:** Choose initial parameters $\theta^{(0)}$.

2: **while** not converged **do**

3: **E-step:** Update q to maximize the lower bound wrt q.

$$q_{t+1} \in \arg\max_{q} \left(\mathcal{L}(q, \theta_t)\right)$$

4: **M-step:** Update θ to maximize the lower bound wrt θ.

$$\theta_{t+1} \in \arg\max_{\theta} \left(\mathcal{L}(q_{t+1}, \theta)\right)$$

5: **end while**

6: **return** Optimized parameters θ^*.

The E-Step

Finding optimal q:

- The gap between log-likelihood and lower bound is:

$$\begin{aligned} d &= \log p_\theta(x) - \mathcal{L}(q, \theta) \\ &= \sum_{i=1}^{n} D_{\mathrm{KL}}\left(q(z_i) \,\|\, p_\theta(z_i|x_i)\right) \end{aligned}$$

- Gap is minimized when $q(z_i) = p_\theta(z_i|x_i)$ for all i
- The E-step is equivalent to setting q to be the posterior distribution:

$$q(z_i) = p_\theta(z_i|x_i)$$

- *Optional*: *Click here for the proof*

Key insight: We maximize the lower bound by setting q to match the posterior distributions under current parameter estimates

Rewriting the lower bound:

$$\mathcal{L}(q, \theta) = \sum_{i=1}^{n} \left(\sum_{z_i} q(z_i) \log p_\theta(x_i, z_i) - \sum_{z_i} q(z_i) \log q(z_i) \right)$$

For optimization with respect to θ:

- The second term $\sum_{z_i} q(z_i) \log q(z_i)$ doesn't depend on θ
- Maximizing $\mathcal{L}(q, \theta)$ with respect to θ is equivalent to:

$$\max_\theta \sum_{i=1}^{n} \sum_{z_i} q(z_i) \log p_\theta(x_i, z_i) = \max_\theta \mathbb{E}_{q(z)}[\log p_\theta(x, z)]$$

Key insight: The M-step maximizes the expected complete log-likelihood with respect to the posterior distribution

The EM Algorithm (Final Form)

Imperial College Business School

Algorithm EM algorithm

Require: Observations $x_1, \ldots, x_n$
Ensure: Optimal θ
1: Initialize $\theta^{(0)}$
2: **while** not converged **do**
3: **E-step:** $q(z) = p(z|x; \theta^{(i-1)})$
4: **M-step:** $\theta^{(i)} = \arg\max_\theta \mathbb{E}_q[\log p(x, z; \theta)]$
5: $i \leftarrow i + 1$
6: **end while**

Properties:

- Guaranteed to increase log-likelihood at each iteration
- Converges to a local maximum.

Exercise: EM for GMMs

Imperial College Business School

Problem: Use the EM algorithm to estimate parameters of a Gaussian Mixture Model.

- Given n observations $x_1, \ldots, x_n \in \mathbb{R}^p$
- Assume latent variables $z_1, \ldots, z_n$ with:
  - $z_i \sim \mathcal{M}(1, \pi_1, \ldots, \pi_K)$ (multinomial with K outcomes)
  - $(x_i \mid z_i = j) \sim \mathcal{N}(\mu_j, \Sigma_j)$ (conditional Gaussian)
- Parameters to estimate: $\theta = (\pi, \mu, \Sigma)$ where:
  - $\pi = (\pi_1, \ldots, \pi_K)$ are mixture weights
  - $\mu = (\mu_1, \ldots, \mu_K)$ are component means
  - $\Sigma = (\Sigma_1, \ldots, \Sigma_K)$ are covariance matrices

Task: Derive the E-step and M-step updates for this model.

Results of the exercise: *Click here for the detailed solution*

- **E-step:** Posterior probabilities $\tau_i^j = p_\theta(z_i = j|x_i)$

$$\tau_i^j = \frac{\pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}{\sum_{j'=1}^{K} \pi_{j'} \mathcal{N}(x_i|\mu_{j'}, \Sigma_{j'})}$$

- **M-step:** Update parameters

$$\pi_{j,t+1} = \frac{1}{n} \sum_{i=1}^{n} \tau_i^j$$

$$\mu_{j,t+1} = \frac{\sum_i \tau_i^j x_i}{\sum_i \tau_i^j}$$

$$\Sigma_{j,t+1} = \frac{\sum_i \tau_i^j (x_i - \mu_{j,t+1})(x_i - \mu_{j,t+1})^T}{\sum_i \tau_i^j}$$

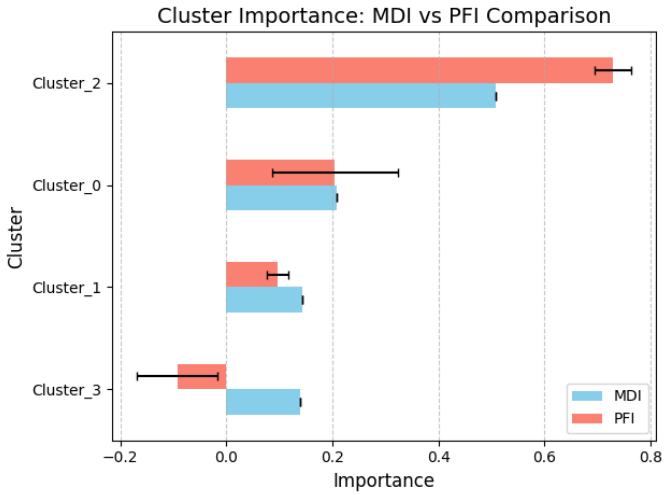
Quiz Time!

Click here to take the quiz

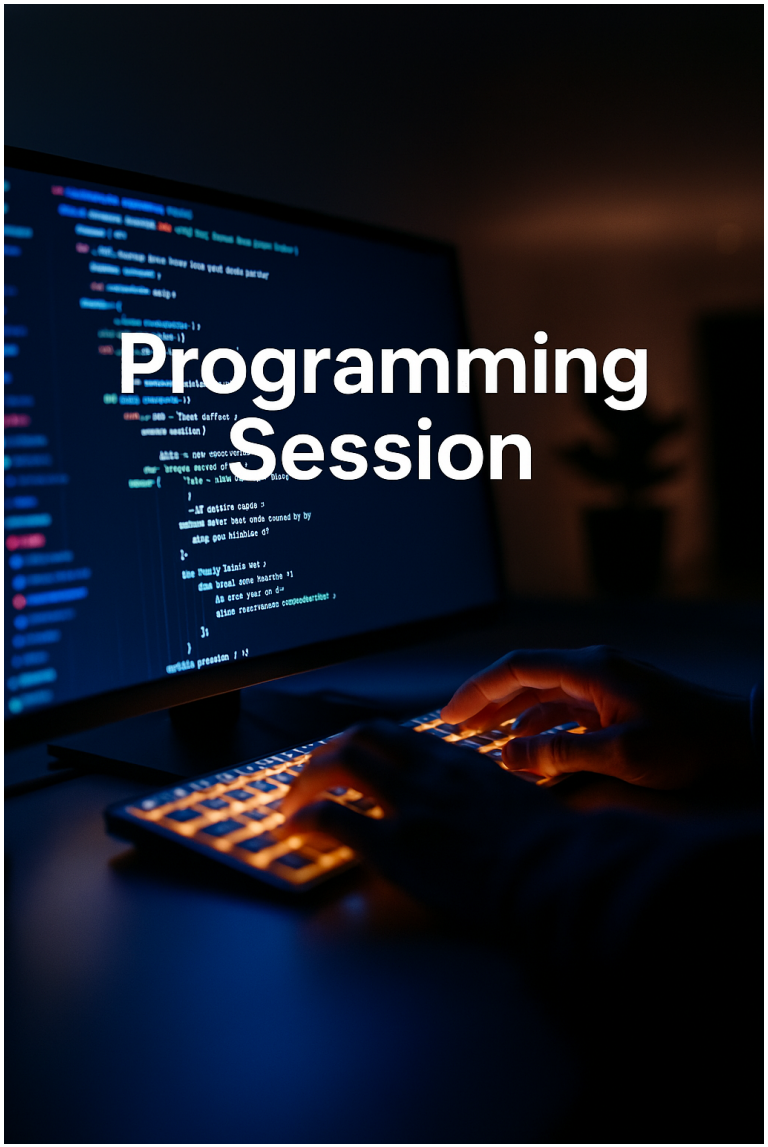
Imperial College Business School

Feature Importance Analysis on a cluster level

- Generated dataset with 1000 samples
- 4 feature clusters with 4 features each:
  - One base feature per cluster (Base_i)
  - Three correlated features per cluster (Corr_i_j)
- 5 pure noise features (Noise_i)
- Ground truth: Only "Base" features directly influence the target



- **Interpretation:**
- Cluster-level analysis correctly identifies noise clusters with negative importance.
- Feature importance at cluster level prevents underestimation of correlated features.



Programming Session 2: Section 3

- Section 3: Cluster-level Feature Importance Analysis
- *Click here to access the programming session*

Solution will be posted tonight on the GitHub page.

- *Click here to access the GitHub Page*

Feedback Poll

Click here to participate in the poll

Thank you for your attention