Elements of Machine Learning and Data Science

Part I: Data Science — Exam Notes (Living Document)

Emir Pisirici

January 30, 2026

Exam likelihood: High (overall Data Science part)

This document is structured to match the lecture topics exactly and is designed for adding **exam-style notes**, **common traps**, and **visual summaries**.

Contents

1 Introduction to Data Science **3**
- 1.1 Introduction . . . 3
- 1.2 Tabular Data . . . 3
- 1.3 Data Science Process . . . 3
  - 1.3.1 ETL vs ELT (Definitions + Differences) . . . 3
  - 1.3.2 CRISP-DM . . . 3
  - 1.3.3 PDCA . . . 4
  - 1.3.4 DMAIC . . . 4
- 1.4 Data Types . . . 4
- 1.5 Descriptive Statistics . . . 4
- 1.6 Basic Visualizations . . . 5
- 1.7 Feature Transformations . . . 6
- 1.8 “How to lie with statistics” . . . 6

2 Decision Trees **7**
- 2.1 Introduction to Decision Trees . . . 7
- 2.2 Entropy and Information Gain . . . 7
- 2.3 ID3 Algorithm . . . 9
- 2.4 Quantifying Information Gain . . . 12
- 2.5 Pruning . . . 13
- 2.6 Continuous Data (Threshold splits) . . . 14
- 2.7 Ensembles (Bagging/Random Forest/Boosting) . . . 17

3 Clustering **18**
- 3.1 Introduction to Unsupervised Learning . . . 18
- 3.2 Introduction to Clustering . . . 18
- 3.3 Similarity and Dissimilarity . . . 19
- 3.4 K-means and K-medoids . . . 20
- 3.5 Agglomerative Clustering . . . 21
- 3.6 DBSCAN . . . 23
- 3.7 Closing . . . 25

4 Frequent Itemsets **26**
4.1 Introduction 26
4.2 Properties of Frequent Itemsets 27
4.3 Apriori Algorithm 28
4.4 FP-Growth Algorithm 30

5 Association Rules **32**
5.1 Introduction 32
5.2 Generating Association Rules 32
5.3 Evaluation (support, confidence, lift, conviction) 32
5.4 Applications 32
5.5 Simpson's Paradox 32

6 Time Series **33**
6.1 Temporal Data 33
6.2 Introduction to Time Series 33
6.3 Analysis 33
6.4 Forecasting 33

1 Introduction to Data Science

1.1 Introduction

1.2 Tabular Data

1.3 Data Science Process

Exam likelihood: High

Framework questions are easy to grade and strongly test “big picture” understanding.

Examiner favorite (what they love to ask)

Typical asks: **ETL vs ELT**, **CRISP-DM phases**, and mapping a scenario to the correct phase. Also: where data leakage/bias lives (data understanding + evaluation).

1.3.1 ETL vs ELT (Definitions + Differences)

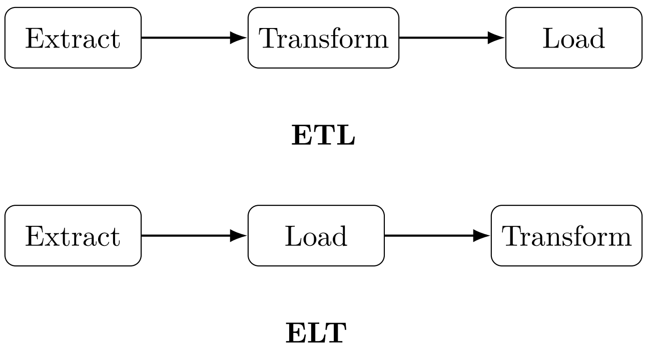
Cheat sheet / must-memorize

ETL: Extract → Transform → Load (transform before target).
ELT: Extract → Load → Transform (transform inside target platform).
Key contrast: where transformations happen; governance vs flexibility; raw history availability.

Common pitfall

People confuse “ELT = no cleaning”. Wrong. It means cleaning happens *after loading*, often in warehouse/lakehouse layers (staging → curated).

Visual

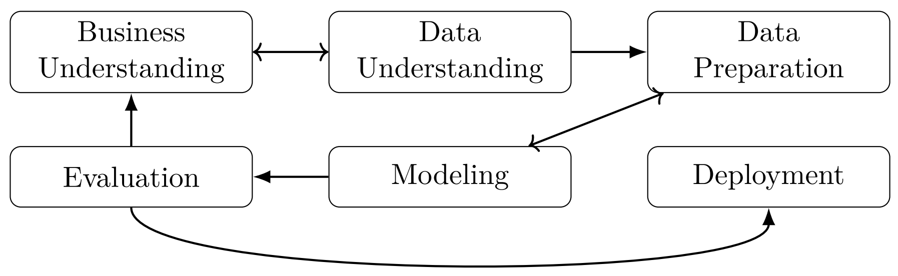


1.3.2 CRISP-DM

Cheat sheet / must-memorize

CRISP-DM: Business Understanding → Data Understanding → Data Preparation → Modeling → Evaluation → Deployment (iterative loops).

Visual



1.3.3 PDCA

Cheat sheet / must-memorize

PDCA: Plan → Do → Check → Act (continuous improvement loop).

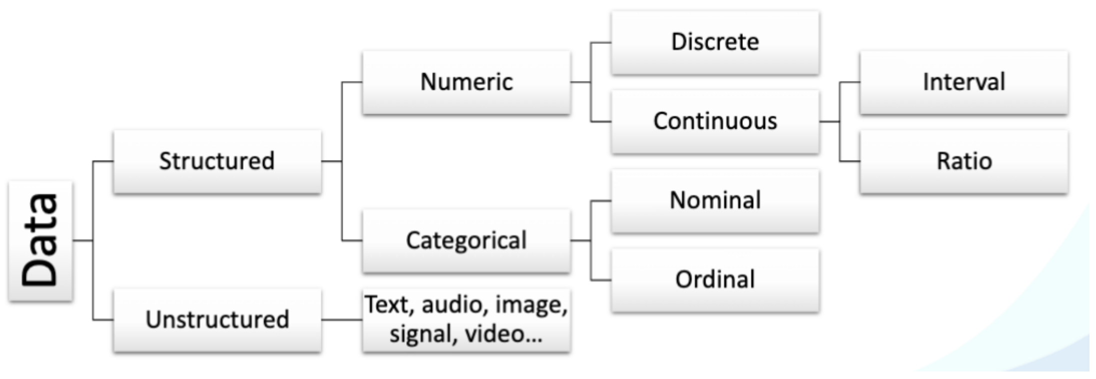
1.3.4 DMAIC

Cheat sheet / must-memorize

DMAIC: Define → Measure → Analyze → Improve → Control. Often used for process/quality improvement + monitoring and part of the Six Sigma methodology.

1.4 Data Types

Visual



Advantages: clarifies variable handling and measurement choices.
Limitations: real data can mix types or sit between categories.

1.5 Descriptive Statistics

Exam likelihood: High

Frequent: compute variance/STD/covariance/correlation by hand; read a correlation matrix. Answer: apply sample formulas, compute values, and interpret sign/magnitude; matrix is symmetric with 1s on the diagonal.

Examiner favorite (what they love to ask)

Explain why covariance depends on units, and why correlation is normalized in $[-1, 1]$. Answer: covariance scales with units; correlation divides by SDs so it is unitless and bounded.

Why (motivation): Quantify spread and association between variables.
What (definition): Variance/STD measure spread; covariance/correlation measure linear association.
How (procedure/usage): Compute formulas, then interpret sign/magnitude and check the correlation matrix.

Cheat sheet / must-memorize

- **Variance (sample):** $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
- **Std dev:** $s = \sqrt{s^2}$
- **Covariance (sample):** $\mathrm{cov}(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$
- **Correlation:** $r = \frac{\mathrm{cov}(X,Y)}{s_X s_Y}$ (unitless, -1 to 1)
- **Correlation matrix:** table of pairwise correlations; symmetric with 1s on the diagonal.

Common pitfall

Correlation $\neq$ causation; a strong correlation can be driven by a confounder or Simpson's paradox.

Advantages: quick, compact summaries of spread and association.
Limitations: can hide distribution shape and outliers.

Visual

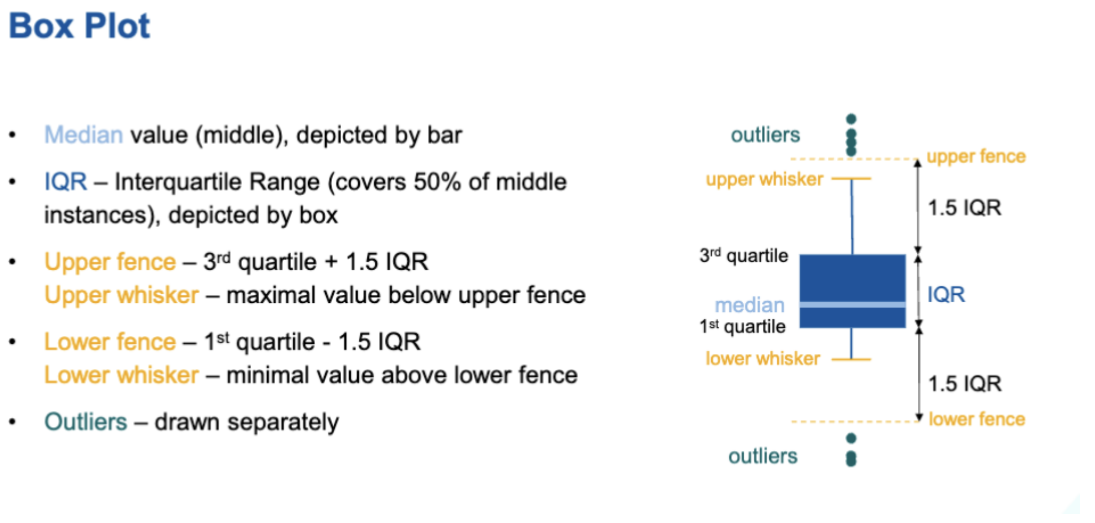
1	r_{12}	r_{13}
r_{21}	1	r_{23}
r_{31}	r_{32}	1

Correlation matrix

Key takeaways: Know formulas + interpretations; correlation matrix is symmetric with 1s on the diagonal.

1.6 Basic Visualizations

Visual



Box Plot

- Median value (middle), depicted by bar
- IQR – Interquartile Range (covers 50% of middle instances), depicted by box
- Upper fence – 3rd quartile + 1.5 IQR
  Upper whisker – maximal value below upper fence
- Lower fence – 1st quartile - 1.5 IQR
  Lower whisker – minimal value above lower fence
- Outliers – drawn separately

Advantages: makes distribution and outliers visible at a glance.
Limitations: can hide multimodality or sample size differences.

1.7 Feature Transformations

Exam likelihood: High

Typical: pick the right transform (scale, log, encode) and explain why. Answer: choose encoding by category type and scaling/log for skew; justify the choice.

Examiner favorite (what they love to ask)

Identify data leakage in preprocessing; name the correct order for train/test transformations. Answer: fit transforms on training data only, then apply to validation/test.

Why (motivation): Turn raw categorical/continuous variables into model-ready features.
What (definition): Encoding or discretizing features without changing the target meaning.
How (procedure/usage): Choose encoding by category type; choose binning by distribution.

Cheat sheet / must-memorize

- **One-hot encoding:** create a 0/1 column per category (nominal).
- **Binary encoding:** represent categories as binary digits (compact one-hot).
- **Ordinal encoding:** map ordered categories to ranks (only if order is real).
- **Binning:** convert continuous to categories.
- **Equal-width binning:** fixed interval sizes across the range.
- **Equal-frequency binning:** same number of samples per bin.

Common pitfall

Fitting transforms on the full dataset (leakage). Always fit on training data, then apply to validation/test.

Advantages: improves model performance and stability.
Limitations: can reduce interpretability or introduce leakage if misapplied.

Key takeaways: Use one-hot for nominal, ordinal for ordered labels, and binning for simplification.

1.8 "How to lie with statistics"

2 Decision Trees

2.1 Introduction to Decision Trees

Exam likelihood: High

Intro questions often ask you to explain how trees split data and what leaves represent. Answer: describe root/internal/leaf roles and how splits partition the feature space.

Examiner favorite (what they love to ask)

Draw a small tree from a toy dataset or explain interpretability vs overfitting. Answer: build simple if-then splits and note that deeper trees overfit without pruning.

Why (motivation): Learn a function from labeled training instances to make predictions.
What (definition): A tree partitions the feature space by sequential if-then splits; leaves output a class or value.
How (procedure/usage): Choose splits to improve class purity or reduce prediction error.

Cheat sheet / must-memorize

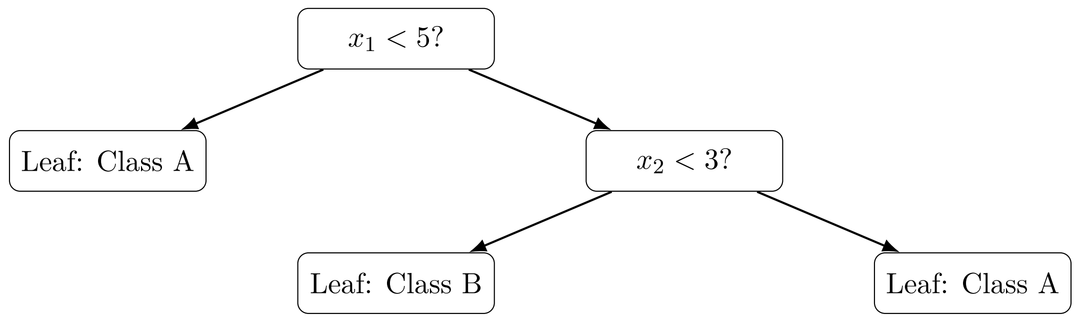
- **Goal:** learn a function $f(X)$ from labeled data to predict labels/values.
- **Tree structure:** root node, internal (non-leaf) nodes, leaf nodes.
- **Split rule:** one feature + threshold; paths are if-then rules.
- **Leaf meaning:** prediction (class/value) for that region of the space.

Common pitfall

Overly deep trees memorize training data; control with max depth, min samples, or pruning.

Advantages: interpretable rules and fast inference.
Limitations: high variance and prone to overfitting without pruning.

Visual



Key takeaways: Trees learn f from labeled data using splits; nodes/leaf roles are core.

2.2 Entropy and Information Gain

Exam likelihood: Very High

Almost guaranteed: compute entropy and information gain for candidate splits. Answer: compute parent entropy, weighted child entropies, then IG.

Examiner favorite (what they love to ask)

Given a small labeled dataset, compare splits and pick the one with highest information gain. Answer: calculate IG for each split and select the largest.

Why (motivation): Choose splits that make child nodes as pure as possible.
What (definition): Entropy measures impurity; information gain is impurity reduction.
How (procedure/usage): Compute parent entropy, child entropies, then IG = parent - weighted children.

Cheat sheet / must-memorize

- **Entropy:** $H(S) = -\sum_c p_c \log_2 p_c$ (define $0 \log 0 = 0$).
- **Weighted child entropy:** $\sum_k \frac{|S_k|}{|S|} H(S_k)$.
- **Information gain:** $\mathrm{IG}(S, \mathrm{split}) = H(S) - \sum_k \frac{|S_k|}{|S|} H(S_k)$.
- **Goal:** choose split with highest IG (most impurity reduction).

Common pitfall

Forgetting to weight child entropies by subset size; using raw entropy sums gives wrong IG.

Advantages: principled split criterion that increases purity.
Limitations: IG is biased toward many-valued attributes.

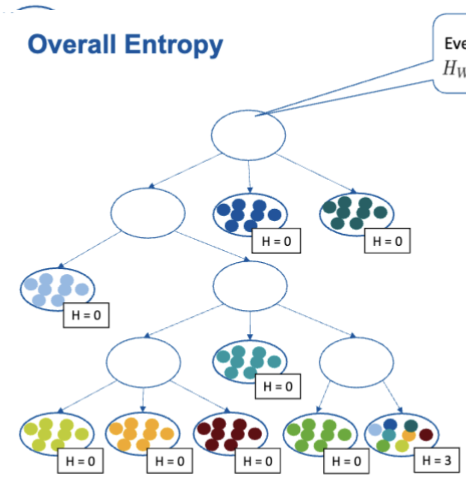
Visual

Entropy - Formula

$$H(t) = -\sum_{k=1}^{K}(P(t=k)\cdot log_s(P(t=k)))$$

$$H(color) = -(\tfrac{7}{14}\cdot log_2(\tfrac{7}{14}) + \tfrac{3}{14}\cdot log_2(\tfrac{3}{14}) + \tfrac{4}{14}\cdot log_2(\tfrac{4}{14})) \approx 1.49$$

Overall Entropy



Even distribution of 8 colors over 72 balls:
$H_W(color) = \frac{72}{72}\cdot\left(-\sum_{k=1}^{8}\left(\frac{9}{72}\cdot log_2(\frac{9}{72})\right)\right) = \log_2(8) = 3$

Overall entropy H_W is the weighted average of the individual entropies:

$$H_W(t) = \sum_{node\in nodes}(\frac{|node|}{N}\cdot H^{node}(t))$$

Example: $N = 72, K = 8$

$$H_W(color) = \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 0 + \frac{8}{72}\cdot 3 = \frac{24}{72} \approx 0.33$$

Information Gain – Another Flight Example

$H(\text{delayed}) = 1$

Weather	Traffic	Night flight	Flight delayed
Cloudy	No	No	**Yes**
Cloudy	Yes	No	**Yes**
Cloudy	Yes	No	**Yes**
Clear	Yes	Yes	**No**
Clear	No	Yes	**No**
Clear	No	No	**No**

$H^{cloudy}(\text{delayed}) = 0$
$H^{clear}(\text{delayed}) = 0$

$H_W^{weather}(\text{delayed}) = 0$

Weather	Flight delayed
Cloudy	**Yes**
Cloudy	**Yes**
Cloudy	**Yes**
Clear	**No**
Clear	**No**
Clear	**No**

$H^{traffic\_yes}(\text{delayed}) = 0.92$
$H^{traffic\_no}(\text{delayed}) = 0.92$

$H_W^{traffic}(\text{delayed}) = 0.92$

Traffic	Flight delayed
No	**Yes**
Yes	**Yes**
Yes	**Yes**
Yes	**No**
No	**No**
No	**No**

$H^{night\_yes}(\text{delayed}) = 0$
$H^{night\_no}(\text{delayed}) \approx 0.81$

$H_W^{night\_flight}(\text{delayed}) \approx 0.54$

Night flight	Flight delayed
No	**Yes**
No	**Yes**
No	**Yes**
Yes	**No**
Yes	**No**
No	**No**

Y N Y N Y N
Y Y Y
N N N
N Y Y
Y N N
N N
N Y Y Y

Key takeaways: Compute entropy, weight children, pick split with highest IG.

Exam likelihood: Very High

Almost guaranteed: compute entropy / information gain on a small dataset.

2.3 ID3 Algorithm

Exam likelihood: Very High

Common: list the ID3 steps or run one iteration to choose the best split. Answer: check stopping, compute IG per attribute, split on max IG, recurse.

Examiner favorite (what they love to ask)

Explain stopping conditions and why ID3 prefers high information gain. Answer: stop if pure/no attributes; IG maximizes impurity reduction.

Why (motivation): Build a decision tree that best separates labeled data.
What (definition): ID3 is a greedy, top-down tree induction algorithm using information gain.
How (procedure/usage): Compute IG for each attribute, split on the best, and recurse.

Cheat sheet / must-memorize

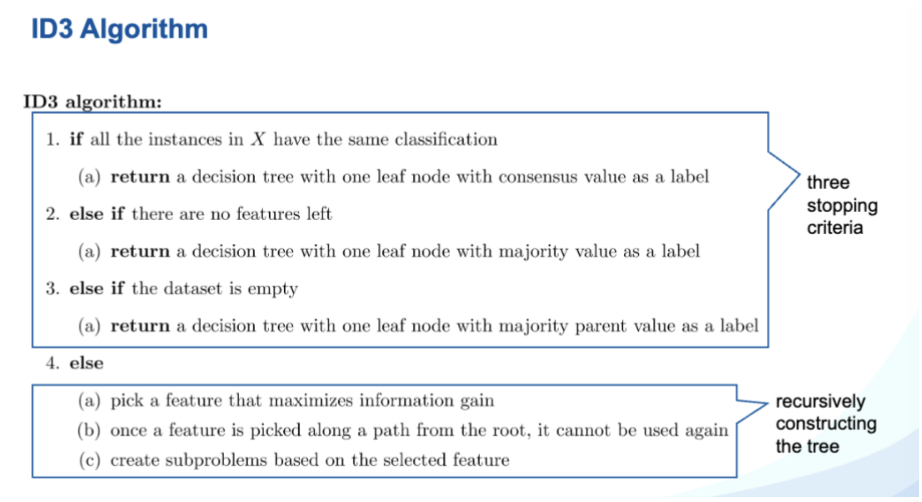
- **Input:** labeled dataset S with categorical attributes (ID3 original).
- **Step 1:** if all labels same $\rightarrow$ make a leaf.
- **Step 2:** if no attributes left $\rightarrow$ leaf with majority class.
- **Step 3:** choose attribute with highest IG.
- **Step 4:** split S by attribute values and recurse.
- **Output:** a decision tree; leaves store class label.

Common pitfall

ID3 favors attributes with many values; without corrections (e.g., gain ratio) it can overfit.

Advantages: simple, fast, and easy to explain.
Limitations: greedy (not optimal), biased to many-valued attributes.

Visual



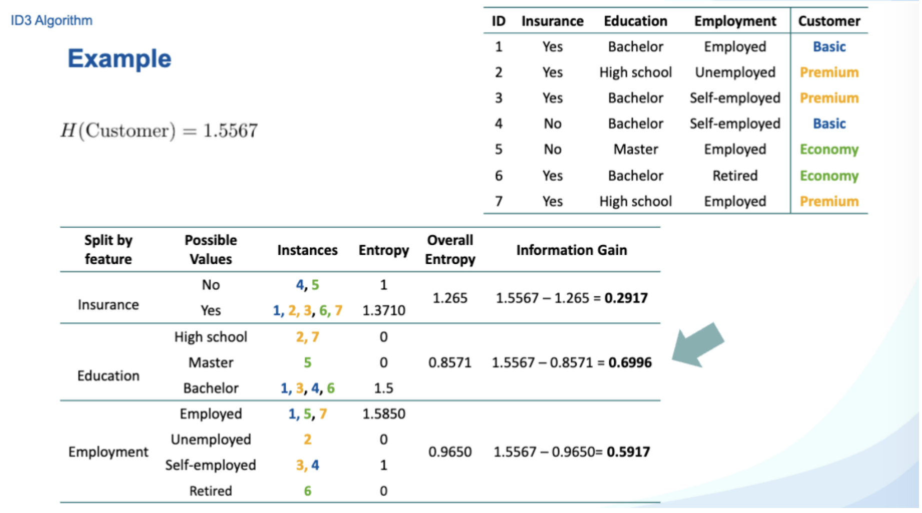
ID3 Algorithm

ID3 algorithm:

```
1. if all the instances in $X$ have the same classification
    (a) return a decision tree with one leaf node with consensus value as a label
2. else if there are no features left
    (a) return a decision tree with one leaf node with majority value as a label
3. else if the dataset is empty
    (a) return a decision tree with one leaf node with majority parent value as a label
4. else
    (a) pick a feature that maximizes information gain
    (b) once a feature is picked along a path from the root, it cannot be used again
    (c) create subproblems based on the selected feature
```

three stopping criteria

recursively constructing the tree



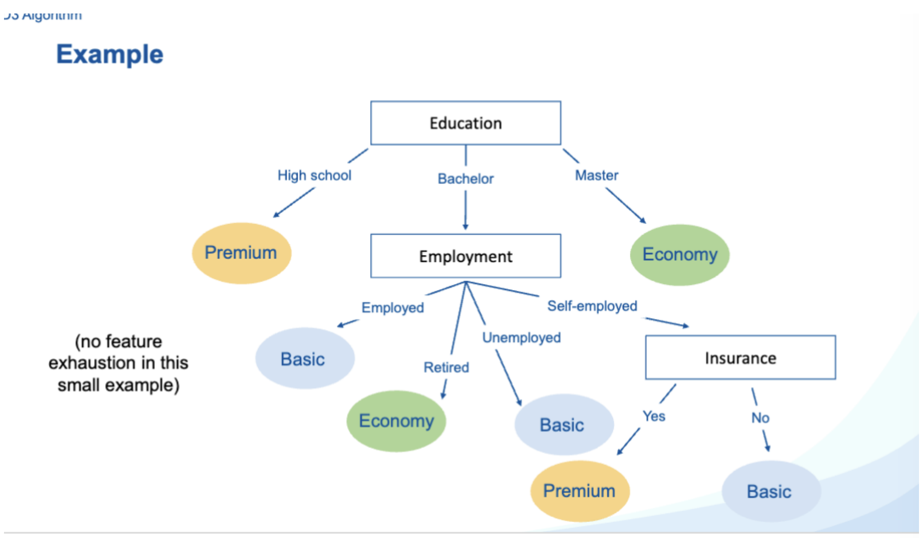
ID3 Algorithm

Example

$H(\text{Customer}) = 1.5567$

ID	Insurance	Education	Employment	Customer
1	Yes	Bachelor	Employed	Basic
2	Yes	High school	Unemployed	Premium
3	Yes	Bachelor	Self-employed	Premium
4	No	Bachelor	Self-employed	Basic
5	No	Master	Employed	Economy
6	Yes	Bachelor	Retired	Economy
7	Yes	High school	Employed	Premium

Split by feature	Possible Values	Instances	Entropy	Overall Entropy	Information Gain
Insurance	No	4, 5	1	1.265	1.5567 – 1.265 = **0.2917**
	Yes	1, 2, 3, 6, 7	1.3710		
Education	High school	2, 7	0	0.8571	1.5567 – 0.8571 = **0.6996**
	Master	5	0		
	Bachelor	1, 3, 4, 6	1.5		
Employment	Employed	1, 5, 7	1.5850	0.9650	1.5567 – 0.9650= **0.5917**
	Unemployed	2	0		
	Self-employed	3, 4	1		
	Retired	6	0		



ID3 Algorithm

Example

(no feature exhaustion in this small example)

Key takeaways: ID3 is greedy; compute IG, split, recurse, stop with pure/majority leaves.

2.4 Quantifying Information Gain

Exam likelihood: Very High

Often compute IG for a specific split and compare candidate attributes. Answer: show parent entropy, weighted children, and IG (optionally gain ratio).

Examiner favorite (what they love to ask)

Show all intermediate steps: parent entropy, each child entropy, weighted sum, IG. Answer: compute each step explicitly and report the final IG (and GR if asked).

Why (motivation): Convert "best split" into a concrete, comparable number.
What (definition): IG = parent entropy minus weighted child entropies; Split Info measures how evenly the split divides data; Gain Ratio normalizes IG.
How (procedure/usage): Compute IG, then divide by split info to get gain ratio.

Cheat sheet / must-memorize

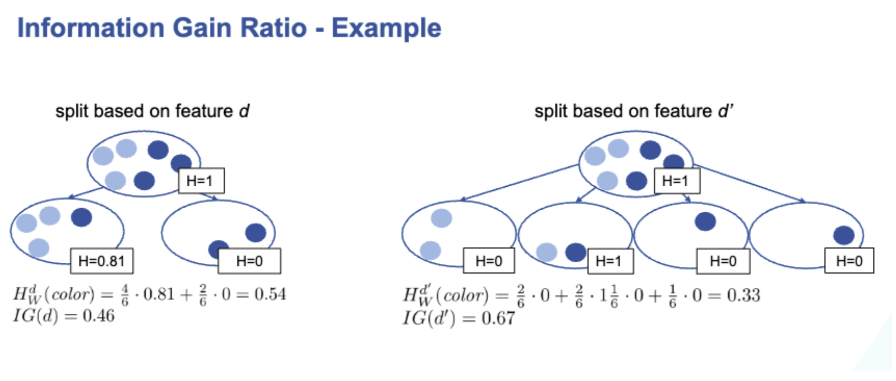
- **Step 1:** compute parent entropy $H(S)$.
- **Step 2:** split by attribute values.
- **Step 3:** compute each child entropy $H(S_k)$.
- **Step 4:** compute weighted sum $\sum_k \frac{|S_k|}{|S|} H(S_k)$.
- **Step 5:** IG $= H(S) - \sum_k \frac{|S_k|}{|S|} H(S_k)$.
- **Split info (lecture: $H(d)$):** entropy of split proportions (how evenly data is partitioned).
  $H(d) = SI = -\sum_k \frac{|S_k|}{|S|} \log_2 \frac{|S_k|}{|S|}$.
- **Gain ratio:** $GR = \frac{IG}{H(d)}$ (same as IG/SI; penalizes many-valued attributes).

Common pitfall

Information gain is biased toward attributes with many values; use gain ratio to correct. Also: weight by subset size and keep log base consistent.

Advantages: makes split comparisons explicit and quantitative.
Limitations: IG can favor attributes with many values without normalization.

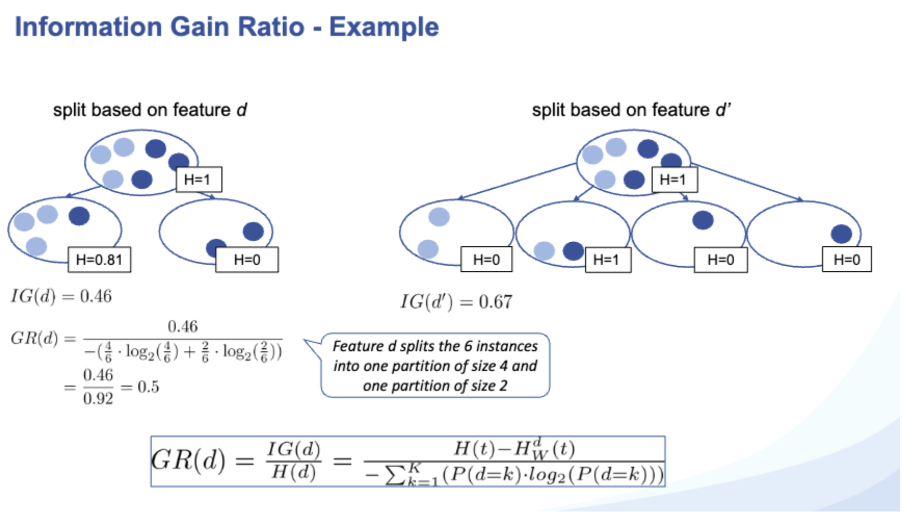
Visual



Information Gain Ratio - Example

$H_W^d(color) = \frac{4}{6} \cdot 0.81 + \frac{2}{6} \cdot 0 = 0.54$
$IG(d) = 0.46$

$H_W^{d'}(color) = \frac{2}{6} \cdot 0 + \frac{2}{6} \cdot 1\frac{1}{6} \cdot 0 + \frac{1}{6} \cdot 0 = 0.33$
$IG(d') = 0.67$



Information Gain Ratio - Example

$IG(d) = 0.46$

$IG(d') = 0.67$

$$GR(d) = \frac{0.46}{-(\frac{4}{6} \cdot \log_2(\frac{4}{6}) + \frac{2}{6} \cdot \log_2(\frac{2}{6}))} = \frac{0.46}{0.92} = 0.5$$

Feature d splits the 6 instances into one partition of size 4 and one partition of size 2

$$\boxed{GR(d) = \frac{IG(d)}{H(d)} = \frac{H(t) - H_W^d(t)}{-\sum_{k=1}^{K}(P(d=k) \cdot log_2(P(d=k)))}}$$

Key takeaways: IG is a weighted impurity reduction; higher is better.

2.5 Pruning

Exam likelihood: High

Often: explain why pruning reduces overfitting and name pre- vs post-pruning. Answer: pruning removes low-value branches to reduce variance; name pre/post pruning.

Examiner favorite (what they love to ask)

Given a tree, identify which branches to prune using validation error or complexity. Answer: prune branches that do not improve validation performance or reduce cost-complexity.

Why (motivation): Reduce overfitting by simplifying a deep tree.
What (definition): Pruning removes splits/branches that do not improve generalization.
How (procedure/usage): Stop early (pre-pruning) or cut back after training (post-pruning).

Cheat sheet / must-memorize

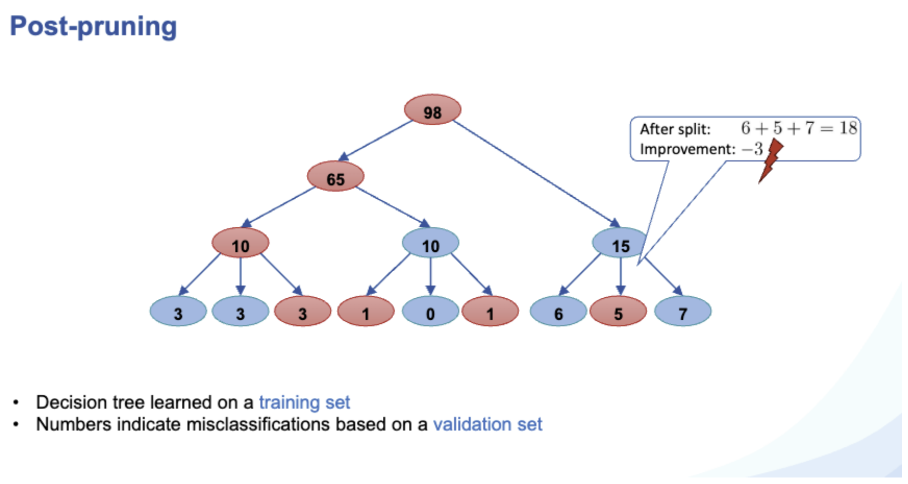
- **Pre-pruning:** stop splitting early using rules like: max depth, min samples per node, min impurity decrease, min samples per leaf.
- **Post-pruning:** grow full tree, then prune using validation error or cost-complexity.
- **Goal:** simpler tree with similar or better validation performance.

Common pitfall

Pruning too aggressively can underfit; always tune on validation data, not test data.

Advantages: improves generalization by reducing variance.
Limitations: too much pruning increases bias.

Visual


Post-pruning

- Decision tree learned on a training set
- Numbers indicate misclassifications based on a validation set

Key takeaways: Pruning trades depth for generalization; use validation to choose.

2.6 Continuous Data (Threshold splits)

Exam likelihood: High

Common: show how a continuous feature is split using a threshold and compute the best split score. Answer: sort values, test midpoints, and choose the split with highest IG (or variance reduction).

Examiner favorite (what they love to ask)

Given sorted values, test candidate thresholds and pick the one with maximum information gain. Answer: evaluate IG at valid midpoints and choose the maximum.

Why (motivation): Many real-world features are continuous; trees need a simple yes/no split.
What (definition): A threshold split picks a value t so that $x_j \le t$ goes left and $x_j > t$ goes right.
How (procedure/usage): Sort values, test candidate thresholds (midpoints between distinct neighbors), compute split score, choose best, recurse.

Variance in a node/leaf (regression trees): For a node with targets $y_1, \dots, y_n$ and mean $\bar{y}$, the node variance (impurity) is the average squared deviation from the mean; a good split minimizes the weighted variance of the child nodes (equivalently, maximizes variance reduction).

Mini example (regression): Data $(x, y) = (1, 2), (2, 2), (3, 3), (4, 10), (5, 11), (6, 12)$.
Parent variance ≈ 19.22. Try two thresholds:

- $t = 3.5$: left $y = \{2, 2, 3\}$ var ≈ 0.22, right $y = \{10, 11, 12\}$ var ≈ 0.67. Weighted variance $= (3/6) \cdot 0.22 + (3/6) \cdot 0.67 \approx 0.45$.
- $t = 1.5$: left $y = \{2\}$ var $= 0$, right $y = \{2, 3, 10, 11, 12\}$ var ≈ 17.84. Weighted variance $\approx (5/6) \cdot 17.84 = 14.87$.

So $t = 3.5$ is preferred because it gives much lower weighted variance.

Cheat sheet / must-memorize

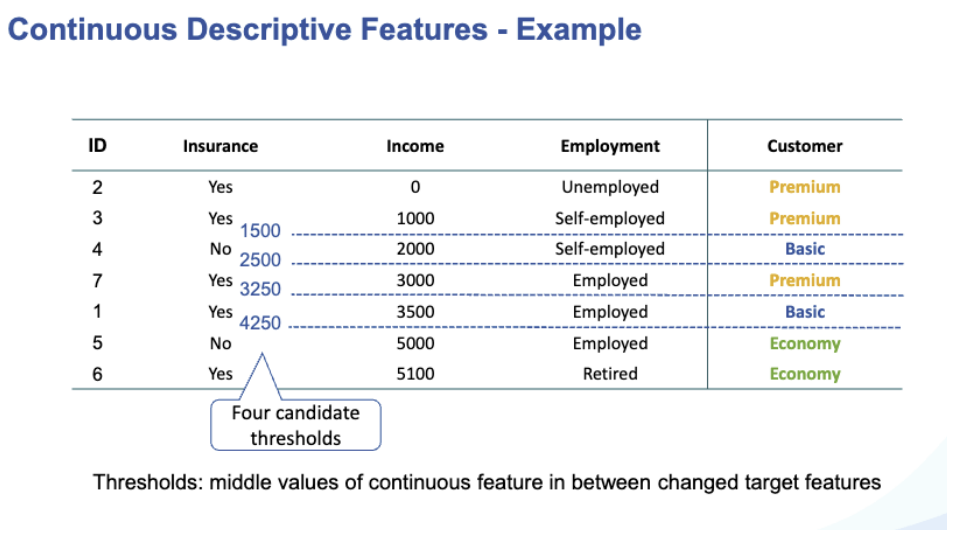
- **Split rule:** $x_j \leq t$ vs $x_j > t$ (binary split).
- **Candidates:** midpoints between sorted unique values; often only where class label changes.
- **Score (classification):** maximize information gain (IG).
- **Variance in a node:** $\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$ (SSE/variance impurity).
- **Score (regression):** minimize weighted child variance / MSE (maximize variance reduction).
- **Efficiency:** sort once ($O(n \log n)$), then sweep thresholds.

Common pitfall

Using thresholds that create empty or nearly empty children (e.g., duplicates without midpoints); always use distinct values and check child sizes.

Advantages: enables trees to handle continuous features.
Limitations: many candidate splits; sensitive to noise/outliers.

Visual

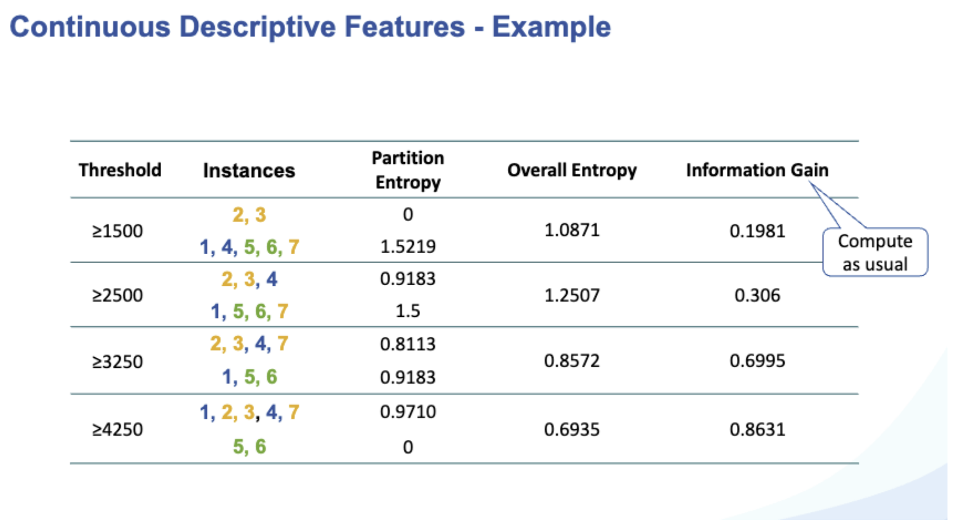


Continuous Descriptive Features - Example

ID	Insurance	Income	Employment	Customer
2	Yes	0	Unemployed	Premium
3	Yes	1000	Self-employed	Premium
4	No	2000	Self-employed	Basic
7	Yes	3000	Employed	Premium
1	Yes	3500	Employed	Basic
5	No	5000	Employed	Economy
6	Yes	5100	Retired	Economy

1500
2500
3250
4250
Four candidate thresholds

Thresholds: middle values of continuous feature in between changed target features



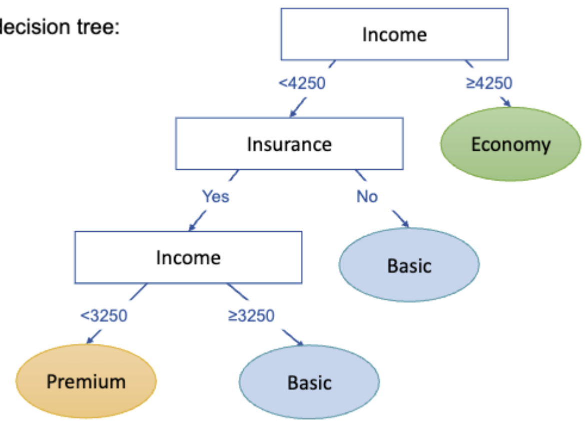
Continuous Descriptive Features - Example

Threshold	Instances	Partition Entropy	Overall Entropy	Information Gain
≥1500	2, 3	0	1.0871	0.1981
	1, 4, 5, 6, 7	1.5219		
≥2500	2, 3, 4	0.9183	1.2507	0.306
	1, 5, 6, 7	1.5		
≥3250	2, 3, 4, 7	0.8113	0.8572	0.6995
	1, 5, 6	0.9183		
≥4250	1, 2, 3, 4, 7	0.9710	0.6935	0.8631
	5, 6	0		

Compute as usual

Continuous Descriptive Features - Example

Resulting decision tree:



The same continuous feature can now be used multiple times!

Key takeaways: Continuous features are split by thresholds; evaluate candidate midpoints and pick the best impurity reduction.

2.7 Ensembles (Bagging/Random Forest/Boosting)

Exam likelihood: High

Short theory questions about why ensembles help and how they are built are common. Answer: averaging many trees reduces variance and stabilizes predictions.

Examiner favorite (what they love to ask)

Explain bias–variance intuition: averaging many trees reduces variance and improves stability. Answer: averaging cancels noise, lowering variance and improving generalization.

Why (motivation): Single trees are high-variance; ensembles stabilize predictions and improve accuracy.
What (definition): An ensemble combines many base models (often trees) into one predictor.
How (procedure/usage): Train multiple trees (e.g., on resampled data) and aggregate their outputs by vote or average.

Cheat sheet / must-memorize

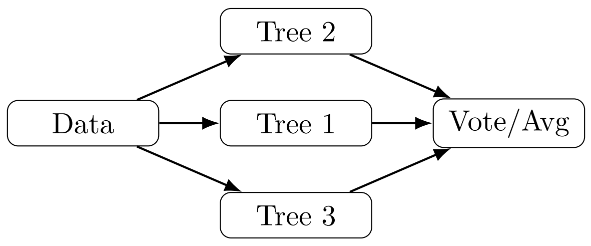
- **Goal:** reduce variance and improve generalization.
- **Combine:** predictions are averaged (regression) or majority-voted (classification).
- **Intuition:** many weak/unstable trees $\rightarrow$ one strong, stable predictor.

Common pitfall

Thinking more trees always fix bias; ensembles mainly reduce variance, not poor features or labels.

Advantages: strong accuracy and stability.
Limitations: reduced interpretability and higher compute.

Visual



Key takeaways: Ensembles combine many trees to reduce variance and stabilize predictions.

3 Clustering

3.1 Introduction to Unsupervised Learning

Exam likelihood: High

Often asked: define unsupervised learning and contrast it with supervised learning. Answer: unsupervised has no labels; it discovers structure like clusters.

Examiner favorite (what they love to ask)

Explain what “no labels” means and give a concrete task like clustering or dimensionality reduction. Answer: only X is given; tasks include clustering and dimensionality reduction.

Why (motivation): Labels are expensive or unavailable; we still want structure in the data.
What (definition): Unsupervised learning finds patterns, groups, or low-dimensional structure without target labels.
How (procedure/usage): Choose a goal (grouping, structure, anomaly detection), define a similarity measure, then apply an algorithm.

Cheat sheet / must-memorize

- **No labels:** only X (features), no y.
- **Main tasks:** clustering, dimensionality reduction, anomaly detection.
- **Key idea:** “similar” points should end up close or in the same group.

Common pitfall

Interpreting clusters as ground truth classes without validation; clusters depend on distance metrics and scaling.

Advantages: finds structure without labels.
Limitations: results can be subjective and metric-dependent.

Key takeaways: Unsupervised learning discovers structure without labels; clustering groups similar points.

3.2 Introduction to Clustering

Exam likelihood: High

Expect definitions and a short compare/contrast of clustering goals or algorithms. Answer: clustering groups similar points; results depend on metric and method.

Examiner favorite (what they love to ask)

Explain what a cluster is and why distance/similarity choice matters. Answer: clusters maximize within-group similarity; metric choice changes the grouping.

Why (motivation): We want to group similar observations when no labels exist.
What (definition): Clustering partitions data into groups so points in the same cluster are more similar to each other than to points in other clusters.
How (procedure/usage): Choose a similarity/distance metric, pick a clustering method, then evaluate/interpret the groups.

Cheat sheet / must-memorize

- **Goal:** high within-cluster similarity, low between-cluster similarity.
- **Distance matters:** Euclidean, Manhattan, cosine, etc., change the result.
- **Outputs:** cluster labels (hard) or membership scores (soft).

Common pitfall

Clustering is not "ground truth"; different scaling or distance metrics can change clusters drastically.

Advantages: reveals hidden group structure.
Limitations: sensitive to metric, scale, and algorithm choice.
Key takeaways: Clustering groups similar points; metric choice and scaling drive the result.

3.3 Similarity and Dissimilarity

Exam likelihood: High

Common: compute a distance/similarity and explain when to use each metric. Answer: apply the correct formula and justify the metric for the data type.

Examiner favorite (what they love to ask)

Compare Euclidean vs Manhattan vs cosine; note the role of scaling/normalization. Answer: Euclidean (L2) vs Manhattan (L1) for continuous data, cosine for direction; scale features first.

Why (motivation): Clustering depends on "closeness"; wrong metric gives wrong groups.
What (definition): Similarity measures how alike points are; dissimilarity (distance) measures how far apart they are.
How (procedure/usage): Choose a metric that matches data type and scale; normalize features when needed.
Goal statement: Maximize similarity within the same group and maximize dissimilarity between different groups.

Cheat sheet / must-memorize

- **Jaccard (nominal/binary):** $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$; distance $d_J = 1 - J$.
- **Minkowski (continuous):** $d_p(x,y) = \left(\sum_i |x_i - y_i|^p\right)^{1/p}$.
- **Manhattan:** $d_1 = \sum_i |x_i - y_i|$ (Minkowski with $p = 1$).
- **Euclidean:** $d_2 = \sqrt{\sum_i (x_i - y_i)^2}$ (Minkowski with $p = 2$).
- **Chebyshev:** $d_\infty = \max_i |x_i - y_i|$ (Minkowski as $p \to \infty$).
- **Scaling:** standardize/normalize when units differ.

Common pitfall

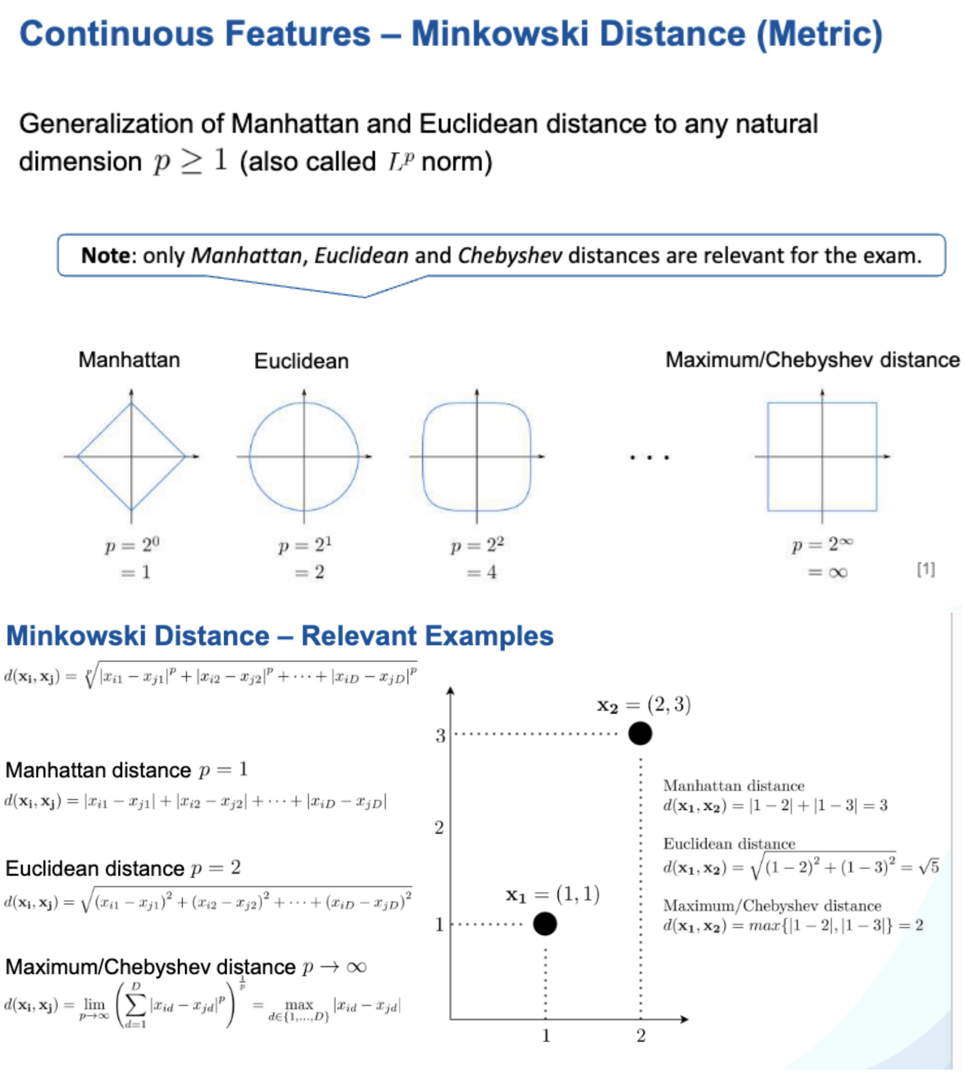
Using Euclidean on unscaled features (one large-unit feature dominates).

Advantages: lets you tailor similarity to data type.
Limitations: wrong metric or scaling yields misleading clusters.
Which to use (rule of thumb):

- **Jaccard:** nominal/binary attributes or set-like data (presence/absence).

- **Manhattan:** high-dimensional data or when you want axis-aligned distance.
- **Euclidean:** continuous features with comparable scale and spherical clusters.
- **Chebyshev:** when the maximum coordinate difference is what matters (strict tolerance).

Visual



Continuous Features – Minkowski Distance (Metric)

Generalization of Manhattan and Euclidean distance to any natural dimension $p \geq 1$ (also called L^p norm)

Note: only *Manhattan*, *Euclidean* and *Chebyshev* distances are relevant for the exam.

Manhattan
Euclidean
Maximum/Chebyshev distance
$p = 2^0 = 1$
$p = 2^1 = 2$
$p = 2^2 = 4$
$\cdots$
$p = 2^\infty = \infty$
[1]

Minkowski Distance – Relevant Examples

$$d(\mathbf{x_i}, \mathbf{x_j}) = \sqrt[p]{|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \cdots + |x_{iD} - x_{jD}|^p}$$

Manhattan distance $p = 1$

$$d(\mathbf{x_i}, \mathbf{x_j}) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{iD} - x_{jD}|$$

Euclidean distance $p = 2$

$$d(\mathbf{x_i}, \mathbf{x_j}) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{iD} - x_{jD})^2}$$

Maximum/Chebyshev distance $p \to \infty$

$$d(\mathbf{x_i}, \mathbf{x_j}) = \lim_{p \to \infty} \left(\sum_{d=1}^{D} |x_{id} - x_{jd}|^p \right)^{\frac{1}{p}} = \max_{d \in \{1,\dots,D\}} |x_{id} - x_{jd}|$$

Manhattan distance
$d(\mathbf{x_1}, \mathbf{x_2}) = |1 - 2| + |1 - 3| = 3$

Euclidean distance
$d(\mathbf{x_1}, \mathbf{x_2}) = \sqrt{(1 - 2)^2 + (1 - 3)^2} = \sqrt{5}$

Maximum/Chebyshev distance
$d(\mathbf{x_1}, \mathbf{x_2}) = max\{|1 - 2|, |1 - 3|\} = 2$

Key takeaways: Metric choice and scaling control the notion of similarity and the resulting clusters.

3.4 K-means and K-medoids

Exam likelihood: High

Often: list K-means steps and compare K-means vs K-medoids. Answer: K-means = init/assign/update/repeat; K-medoids uses medoids and is more robust.

Examiner favorite (what they love to ask)

Run 1 iteration of K-means by hand (assign & update) or explain why K-medoids is more robust to outliers. Answer: compute distances, assign clusters, recompute means; medoids resist outliers.

Why (motivation): Need a simple, scalable way to cluster continuous data.
What (definition): K-means uses centroids (means) to minimize within-cluster SSE; K-medoids uses

actual data points as centers and minimizes total within-cluster distance.
How (procedure/usage): Initialize k centers, assign each point to nearest center, update centers, repeat until convergence.

Comparison: K-means is faster but sensitive to outliers and scaling; K-medoids is more robust but slower.
Difference: K-means centers are means (can be non-data points); K-medoids centers are actual data points.

Cheat sheet / must-memorize

- **K-means objective:** minimize $\sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$.
- **K-means steps:** (1) choose K and initialize μ_k; (2) assign each point to nearest μ_k; (3) update μ_k to the mean of its cluster; (4) repeat 2–3 until assignments stop changing.
- **K-means optimization:** alternating minimization (assignment step + centroid update); decreases SSE each iteration.
- **K-medoids center:** medoid m_k is a data point minimizing total distance in its cluster.
- **K-medoids steps:** (1) choose K medoids m_k; (2) assign each point to nearest medoid; (3) for each cluster, swap medoid candidate to reduce total distance; (4) repeat 2–3 until no improvement.
- **K-medoids optimization:** minimize $\sum_k \sum_{x_i \in C_k} d(x_i, m_k)$ via swap heuristics (e.g., PAM); monotonic improvement, not guaranteed global optimum.
- **Robustness:** K-medoids less sensitive to outliers than K-means.

Common pitfall

K-means assumes roughly spherical clusters, is sensitive to scaling/initialization, and handles outliers poorly. K-medoids is more robust but slower on large datasets.

Advantages: K-means is fast; K-medoids is robust.
Limitations: K-means sensitive to outliers; K-medoids is slower.

Key takeaways: K-means is fast but sensitive; K-medoids is more robust but costlier.

3.5 Agglomerative Clustering

Exam likelihood: High

Common: list agglomerative steps and compute one merge with a given linkage. Answer: start with singletons, merge closest by linkage, update distances, repeat.

Examiner favorite (what they love to ask)

Given a small distance matrix, perform one or two agglomerative merges and state the linkage used. Answer: compute linkage distances and merge the smallest pair step by step.

Why (motivation): We want a hierarchy of clusters and do not want to pre-specify K.
What (definition): Agglomerative hierarchical clustering starts with each point as its own cluster and repeatedly merges the closest clusters.
How (procedure/usage): Choose a distance metric + linkage rule, then merge until one cluster remains (cut the dendrogram to pick K).

Cheat sheet / must-memorize

- **Algorithm (simplified):** start with n singleton clusters; compute inter-cluster distances; merge closest pair; update distances; repeat.
- **Single linkage:** $d(A, B) = \min_{x \in A, y \in B} d(x, y)$.
- **Complete linkage:** $d(A, B) = \max_{x \in A, y \in B} d(x, y)$.
- **Average linkage:** $d(A, B) = \frac{1}{|A||B|} \sum_{x \in A} \sum_{y \in B} d(x, y)$.
- **Centroid linkage:** $d(A, B) = \|\mu_A - \mu_B\|$.
- **Ward:** merge that yields the smallest increase in within-cluster SSE.

Common pitfall

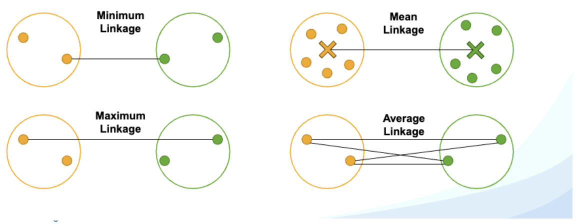
Single linkage can "chain" clusters; results depend heavily on scaling and the chosen linkage.

Advantages: produces a full hierarchy; no need to choose K upfront. Limitations: $O(n^2)$ memory/time; early merges are irreversible.

Visual

Linkage Measures

- The distance between clusters is otherwise known as linkage measure
- Four widely used linkage measures:



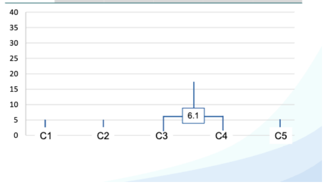
Simplistic Agglomerative Clustering – Example

Cluster	Country	Meat & Fish	Plants	Eggs & Milk
1	Albania	11.7	50.1	9.4
2	Bulgaria	15	65.7	9.9
3+4	Austria	25	37.2	24.2
	Belgium	27.3	38.4	21.6
5	Czechoslovakia	23.1	44.4	15.3

Cluster	1	2	3	4	5
1	-	19.4	41	39.5	23
2		-	52.8	51.3	34.8
3			-	6.1	18
				-	16.5
5					-

Merge these clusters

- Compare countries based on sources of protein
- Use mean linkage between clusters (compare centroids)
- Use Manhattan distance between instances



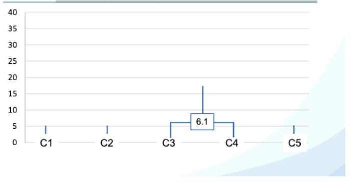
Simplistic Agglomerative Clustering – Example

Cluster	Country	Meat & Fish	Plants	Eggs & Milk
1	Albania	11.7	50.1	9.4
2	Bulgaria	15	65.7	9.9
3+4	Austria Belgium	26.15	37.8	22.9
5	Czechoslovakia	23.1	44.4	15.3

Cluster	1	2	3+4	5
1	-	19.4	40.25	23
2		-	52.05	34.8
3+4			-	17.25
5				-

Compute new linkage values

- Compare countries based on sources of protein
- Use mean linkage between clusters (compare centroids)
- Use Manhattan distance between instances



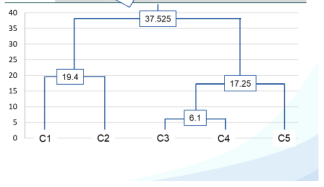
Simplistic Agglomerative Clustering – Example

Cluster	Country	Meat & Fish	Plants	Eggs & Milk
1+2	Albania Bulgaria	13.35	57.9	9.65
3+4+5	Austria Belgium Czechoslovakia	24.625	41.1	19.1

Cluster	1+2	3+4+5
1+2	-	37.525
3+4+5		-

Merge to final cluster

- Compare countries based on sources of protein
- Use mean linkage between clusters (compare centroids)
- Use Manhattan distance between instances



Key takeaways: Agglomerative builds a dendrogram using a linkage rule; choose the cut for K.

3.6 DBSCAN

Exam likelihood: High

Often: define ε and MinPts, classify core/border/noise points. Answer: core has $\geq$ MinPts in ε-neighborhood; border within ε of a core; noise otherwise.

Examiner favorite (what they love to ask)

Given a small plot, label core/border/noise and explain why DBSCAN finds non-spherical clusters. Answer: use density reachability; clusters can follow arbitrary shapes.

Why (motivation): K-based methods struggle with arbitrary shapes and noise.
What (definition): DBSCAN groups dense regions using ε-neighborhoods and MinPts; points in sparse regions become noise.
How (procedure/usage): Pick ε and MinPts, mark core points, expand clusters via density reachability, label remaining as noise/border.

Cheat sheet / must-memorize

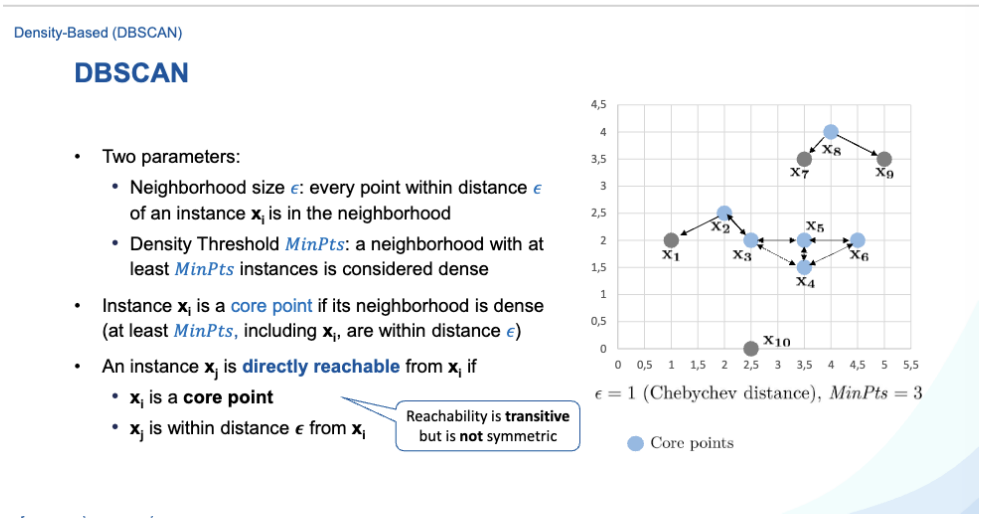
- **Parameters:** ε (radius), MinPts (minimum neighbors).
- **Core point:** has at least MinPts points in its ε-neighborhood.
- **Border point:** within ε of a core but not itself core.
- **Noise:** not reachable from any core.
- **Density-reachable/connected:** chains of core points link a cluster.

Common pitfall

Bad ε/MinPts choices: too small $\rightarrow$ many noise points; too large $\rightarrow$ merged clusters.

Advantages: no need to choose K; finds arbitrary-shaped clusters; handles noise well.
Limitations: struggles with varying density; parameter sensitive; distance metrics degrade in high dimensions.

Visual



Density-Based (DBSCAN)

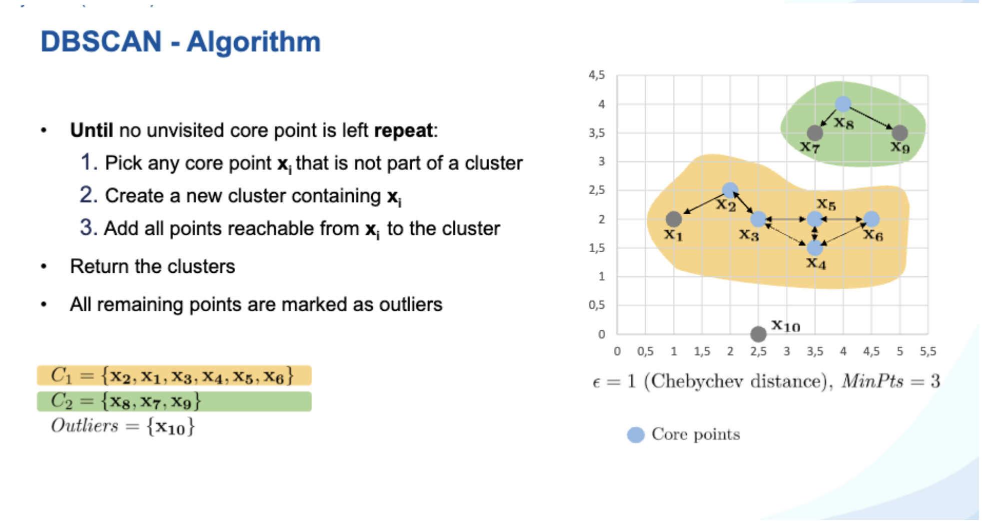
DBSCAN

- Two parameters:
  - Neighborhood size ϵ: every point within distance ϵ of an instance $\mathbf{x}_i$ is in the neighborhood
  - Density Threshold $MinPts$: a neighborhood with at least $MinPts$ instances is considered dense
- Instance $\mathbf{x}_i$ is a core point if its neighborhood is dense (at least $MinPts$, including $\mathbf{x}_i$, are within distance ϵ)
- An instance $\mathbf{x}_j$ is **directly reachable** from $\mathbf{x}_i$ if
  - $\mathbf{x}_i$ is a **core point**
  - $\mathbf{x}_j$ is within distance ϵ from $\mathbf{x}_i$

Reachability is **transitive** but is **not** symmetric

$\epsilon = 1$ (Chebychev distance), $MinPts = 3$

● Core points



DBSCAN - Algorithm

- **Until** no unvisited core point is left **repeat**:
  1. Pick any core point $\mathbf{x}_i$ that is not part of a cluster
  2. Create a new cluster containing $\mathbf{x}_i$
  3. Add all points reachable from $\mathbf{x}_i$ to the cluster
- Return the clusters
- All remaining points are marked as outliers

$C_1 = \{\mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$
$C_2 = \{\mathbf{x}_8, \mathbf{x}_7, \mathbf{x}_9\}$
$Outliers = \{\mathbf{x}_{10}\}$

$\epsilon = 1$ (Chebychev distance), $MinPts = 3$

● Core points

Key takeaways: DBSCAN clusters by density and naturally flags noise; choose ε and MinPts carefully.

3.7 Closing

4 Frequent Itemsets

4.1 Introduction

Exam likelihood: High

Often: define itemset, support, and why frequent itemsets matter. Answer: itemset is a set of items; support is fraction of transactions; frequent if above minsup.

Examiner favorite (what they love to ask)

Compute support for a small transaction set and identify frequent itemsets above a threshold. Answer: count transactions containing the itemset, divide by total, compare to minsup.

Why (motivation): Discover co-occurrence patterns in transaction data (e.g., market baskets).
What (definition): An itemset is a set of items; a frequent itemset appears in at least a minimum fraction of transactions.
How (procedure/usage): Count itemset support across transactions and keep those meeting a minimum support threshold.

Cheat sheet / must-memorize

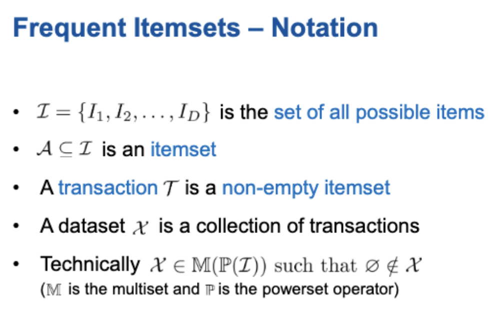
- **Transaction DB:** each transaction is a set of items.
- **Itemset:** a subset of items (e.g., $\{A, B\}$).
- **Support:** $\mathrm{supp}(X) = \frac{\#\text{transactions containing } X}{\#\text{transactions}}$.
- **Frequent:** $\mathrm{supp}(X) \geq \text{min\_support}$.
- **Apriori idea:** if an itemset is frequent, all its subsets are frequent.

Common pitfall

Mixing up support count (absolute) vs support (fraction); always state which one you use.

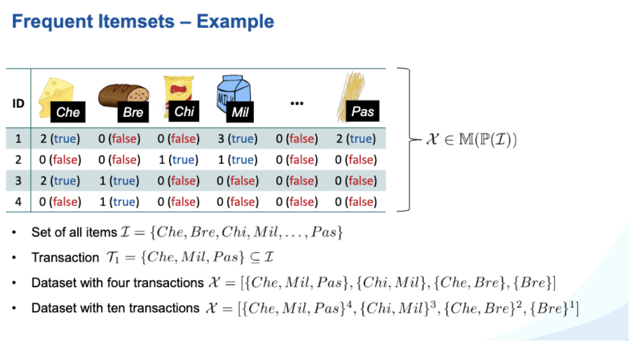
Advantages: highlights common co-occurrence patterns.
Limitations: can generate many itemsets; sensitive to min support.

Visual



Frequent Itemsets – Notation

- $\mathcal{I} = \{I_1, I_2, \ldots, I_D\}$ is the set of all possible items
- $\mathcal{A} \subseteq \mathcal{I}$ is an itemset
- A transaction $\mathcal{T}$ is a non-empty itemset
- A dataset $\mathcal{X}$ is a collection of transactions
- Technically $\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$ such that $\varnothing \notin \mathcal{X}$
  ($\mathbb{M}$ is the multiset and $\mathbb{P}$ is the powerset operator)

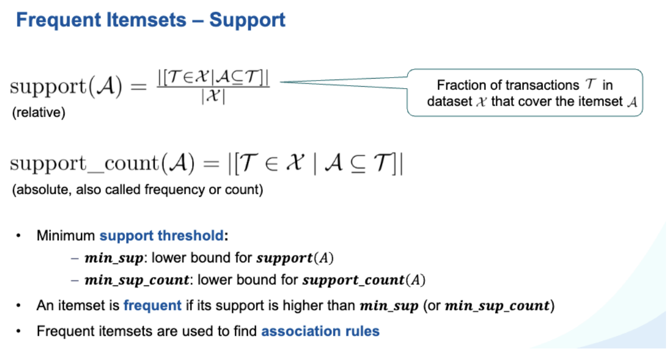


Frequent Itemsets – Example

ID	Che	Bre	Chi	Mil	...	Pas
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	0 (false)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$

- Set of all items $\mathcal{I} = \{Che, Bre, Chi, Mil, \ldots, Pas\}$
- Transaction $\mathcal{T}_1 = \{Che, Mil, Pas\} \subseteq \mathcal{I}$
- Dataset with four transactions $\mathcal{X} = [\{Che, Mil, Pas\}, \{Chi, Mil\}, \{Che, Bre\}, \{Bre\}]$
- Dataset with ten transactions $\mathcal{X} = [\{Che, Mil, Pas\}^4, \{Chi, Mil\}^3, \{Che, Bre\}^2, \{Bre\}^1]$



Frequent Itemsets – Support

$$\text{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|}$$

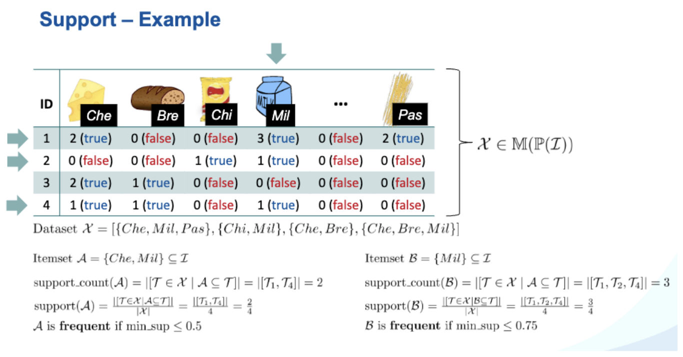
(relative)

Fraction of transactions $\mathcal{T}$ in dataset $\mathcal{X}$ that cover the itemset $\mathcal{A}$

$$\text{support\_count}(\mathcal{A}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]|$$

(absolute, also called frequency or count)

- Minimum support threshold:
  - ***min_sup***: lower bound for ***support***(A)
  - ***min_sup_count***: lower bound for ***support_count***(A)
- An itemset is frequent if its support is higher than ***min_sup*** (or ***min_sup_count***)
- Frequent itemsets are used to find association rules



Support – Example

ID	Che	Bre	Chi	Mil	...	Pas
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	1 (true)	1 (true)	0 (false)	1 (true)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$

Dataset $\mathcal{X} = [\{Che, Mil, Pas\}, \{Chi, Mil\}, \{Che, Bre\}, \{Che, Bre, Mil\}]$

Itemset $\mathcal{A} = \{Che, Mil\} \subseteq \mathcal{I}$

$\text{support\_count}(\mathcal{A}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]| = |[\mathcal{T}_1, \mathcal{T}_4]| = 2$

$\text{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|} = \frac{|[\mathcal{T}_1, \mathcal{T}_4]|}{4} = \frac{2}{4}$

$\mathcal{A}$ is **frequent** if min_sup ≤ 0.5

Itemset $\mathcal{B} = \{Mil\} \subseteq \mathcal{I}$

$\text{support\_count}(\mathcal{B}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]| = |[\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_4]| = 3$

$\text{support}(\mathcal{B}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{B} \subseteq \mathcal{T}]|}{|\mathcal{X}|} = \frac{|[\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_4]|}{4} = \frac{3}{4}$

$\mathcal{B}$ is **frequent** if min_sup ≤ 0.75

Key takeaways: Frequent itemsets reveal common co-occurrences; support is the key measure.

4.2 Properties of Frequent Itemsets

Exam likelihood: High

Often: define closed vs maximal itemsets and compare them with frequent itemsets. Answer: closed has no superset with same support; maximal has no frequent superset.

Examiner favorite (what they love to ask)

Given a small set of frequent itemsets, identify which are closed and which are maximal. Answer: maximal = no frequent superset; closed = no superset with equal support.

Why (motivation): Frequent itemsets can be many; closed/maximal summarize them compactly.
What (definition): Closed itemsets keep full support information; maximal itemsets keep only the largest frequent ones.
How (procedure/usage): Mine frequent itemsets first, then filter for closed or maximal conditions.

Cheat sheet / must-memorize

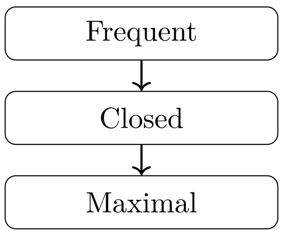
- **Frequent:** supp(X) $\geq$ min_support.
- **Closed:** no proper superset of X has the *same* support.
- **Maximal:** no proper superset of X is frequent.
- **Relationship:** Maximal $\subset$ Closed $\subset$ Frequent.
- **Info loss:** Maximal loses support counts; closed preserves supports for all frequent itemsets.

Common pitfall

Confusing "closed" with "maximal": closed allows frequent supersets as long as support drops; maximal allows no frequent supersets.

Advantages: closed/maximal reduce output size.
Limitations: maximal loses support info; closed can still be large.

Visual



Key takeaways: Closed keeps support info; maximal is a compact summary but loses support counts.

4.3 Apriori Algorithm

Exam likelihood: High

Common: outline Apriori steps and generate C_k and L_k for a tiny dataset. Answer: build L_1, generate/prune C_k, count supports, repeat to get L_k.

Examiner favorite (what they love to ask)

Show candidate generation and pruning using the Apriori property. Answer: join L_{k-1} to form C_k, prune any candidate with an infrequent subset.

Why (motivation): Brute-force itemset search is exponential; Apriori prunes early.
What (definition): Apriori uses the downward-closure property: all subsets of a frequent itemset are frequent.
How (procedure/usage): Iteratively generate candidates and prune using frequent subsets.

Cheat sheet / must-memorize

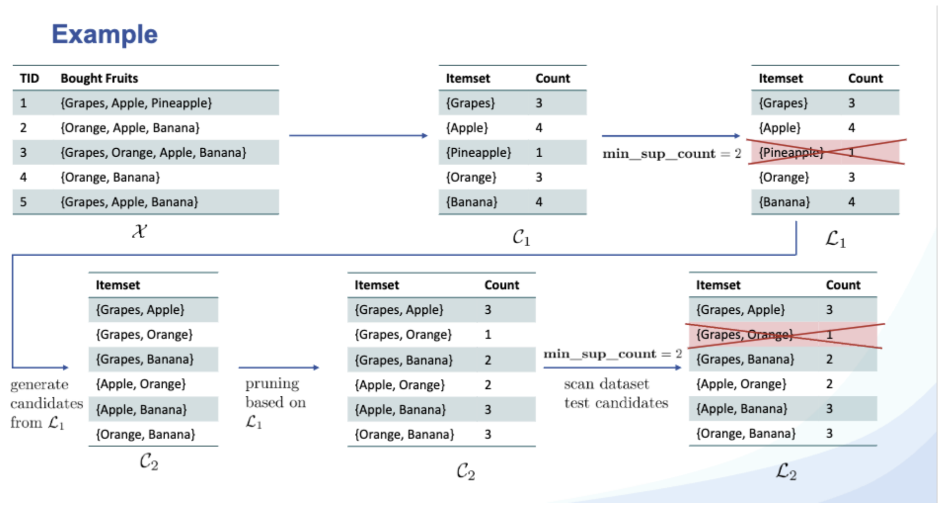
- **Step 1:** find frequent 1-itemsets L_1.
- **Step 2:** generate candidate k-itemsets C_k from L_{k-1} (join step).
- **Step 3:** prune any candidate in C_k with an infrequent $(k-1)$-subset.
- **Step 4:** scan DB to count supports; keep L_k (frequent candidates).
- **Stop:** when L_k is empty.

Common pitfall

Forgetting the prune step (using the Apriori property) leads to too many candidates.

Advantages: strong pruning via downward-closure; easy to implement and explain.
Limitations: challenging candidate generation; each candidate must be tested against the whole dataset.

Visual



Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

Itemset	Count
{Grapes}	3
{Apple}	4
{Pineapple}	1
{Orange}	3
{Banana}	4

$\mathcal{C}_1$

min_sup_count = 2

Itemset	Count
{Grapes}	3
{Apple}	4
~~{Pineapple}~~	~~1~~
{Orange}	3
{Banana}	4

$\mathcal{L}_1$

generate candidates from $\mathcal{L}_1$

Itemset
{Grapes, Apple}
{Grapes, Orange}
{Grapes, Banana}
{Apple, Orange}
{Apple, Banana}
{Orange, Banana}

$\mathcal{C}_2$

pruning based on $\mathcal{L}_1$

Itemset	Count
{Grapes, Apple}	3
{Grapes, Orange}	1
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

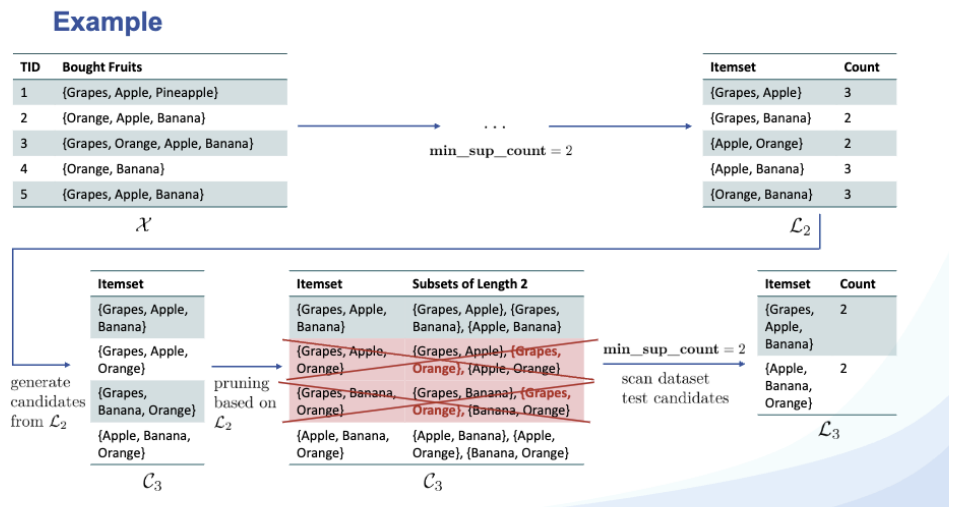
$\mathcal{C}_2$

min_sup_count = 2

scan dataset test candidates

Itemset	Count
{Grapes, Apple}	3
~~{Grapes, Orange}~~	~~1~~
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

$\mathcal{L}_2$



Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

. . .

min_sup_count = 2

Itemset	Count
{Grapes, Apple}	3
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

$\mathcal{L}_2$

generate candidates from $\mathcal{L}_2$

Itemset
{Grapes, Apple, Banana}
{Grapes, Apple, Orange}
{Grapes, Banana, Orange}
{Apple, Banana, Orange}

$\mathcal{C}_3$

pruning based on $\mathcal{L}_2$

Itemset	Subsets of Length 2
{Grapes, Apple, Banana}	{Grapes, Apple}, {Grapes, Banana}, {Apple, Banana}
~~{Grapes, Apple, Orange}~~	~~{Grapes, Apple}, {Grapes, Orange}, {Apple, Orange}~~
~~{Grapes, Banana, Orange}~~	~~{Grapes, Banana}, {Grapes, Orange}, {Banana, Orange}~~
{Apple, Banana, Orange}	{Apple, Banana}, {Apple, Orange}, {Banana, Orange}

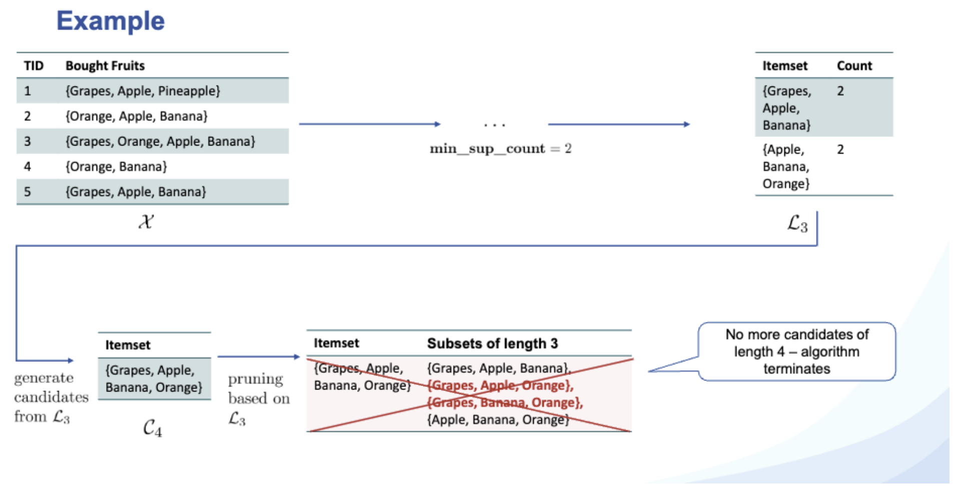
$\mathcal{C}_3$

min_sup_count = 2

scan dataset test candidates

Itemset	Count
{Grapes, Apple, Banana}	2
{Apple, Banana, Orange}	2

$\mathcal{L}_3$



Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

. . .

min_sup_count = 2

Itemset	Count
{Grapes, Apple, Banana}	2
{Apple, Banana, Orange}	2

$\mathcal{L}_3$

generate candidates from $\mathcal{L}_3$

Itemset
{Grapes, Apple, Banana, Orange}

$\mathcal{C}_4$

pruning based on $\mathcal{L}_3$

Itemset	Subsets of length 3
~~{Grapes, Apple, Banana, Orange}~~	~~{Grapes, Apple, Banana}, {Grapes, Apple, Orange}, {Grapes, Banana, Orange}, {Apple, Banana, Orange}~~

No more candidates of length 4 – algorithm terminates

Key takeaways: Apriori alternates candidate generation and pruning using downward-closure.

4.4 FP-Growth Algorithm

Exam likelihood: High

Often: explain why FP-Growth is faster than Apriori and outline its steps. Answer: no candidate generation; build FP-tree in two passes and mine with DFS.

Examiner favorite (what they love to ask)

State the two DB passes and describe the FP-tree + conditional pattern base idea. Answer: pass 1 counts and orders items; pass 2 builds FP-tree; mine conditional bases/trees by DFS.

Why (motivation): Apriori's candidate explosion and repeated full scans are costly.
What (definition): FP-Growth mines frequent itemsets by building an FP-tree and doing DFS on conditional pattern bases (no candidate generation).
How (procedure/usage): Make two passes, build the FP-tree, then recursively mine conditional FP-trees (depth-first).

Cheat sheet / must-memorize

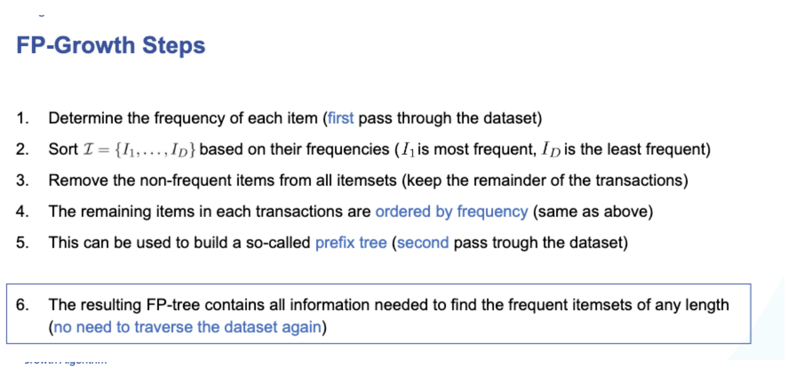
- **Pass 1:** count item supports; keep frequent items and order them.
- **Pass 2:** build FP-tree by inserting ordered transactions.
- **DFS mining:** for each item, build its conditional pattern base and conditional FP-tree; recurse.
- **Output:** frequent itemsets without candidate generation.

Common pitfall

Not ordering items by global frequency before building the FP-tree (tree becomes large and inefficient).

Advantages: avoids candidate generation; only two DB passes; efficient for dense datasets.
Limitations: FP-tree can be large in very sparse data; more complex to implement.

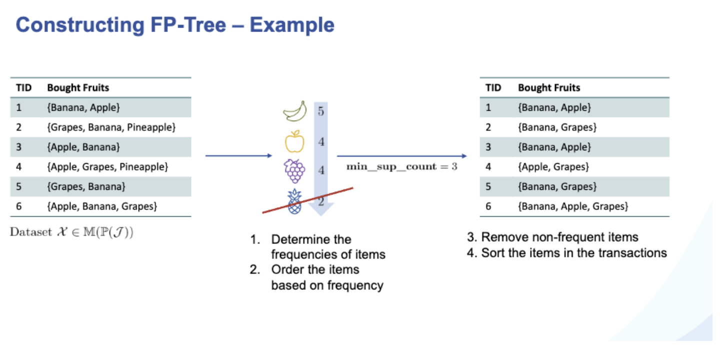
Visual



FP-Growth Steps

1. Determine the frequency of each item (first pass through the dataset)
2. Sort $\mathcal{I} = \{I_1, \ldots, I_D\}$ based on their frequencies (I_1 is most frequent, I_D is the least frequent)
3. Remove the non-frequent items from all itemsets (keep the remainder of the transactions)
4. The remaining items in each transactions are ordered by frequency (same as above)
5. This can be used to build a so-called prefix tree (second pass trough the dataset)

6. The resulting FP-tree contains all information needed to find the frequent itemsets of any length (no need to traverse the dataset again)



Constructing FP-Tree – Example

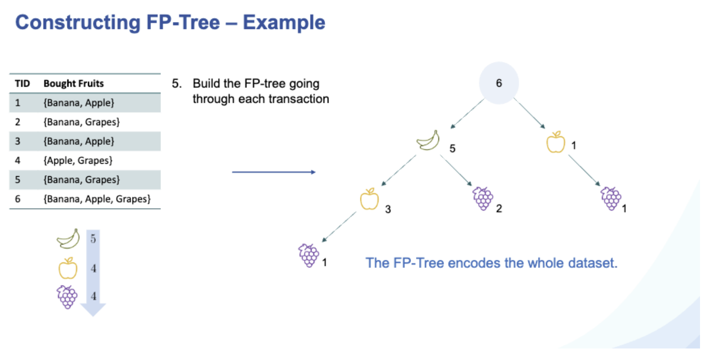
TID	Bought Fruits
1	{Banana, Apple}
2	{Grapes, Banana, Pineapple}
3	{Apple, Banana}
4	{Apple, Grapes, Pineapple}
5	{Grapes, Banana}
6	{Apple, Banana, Grapes}

Dataset $\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{J}))$

1. Determine the frequencies of items
2. Order the items based on frequency

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

3. Remove non-frequent items
4. Sort the items in the transactions

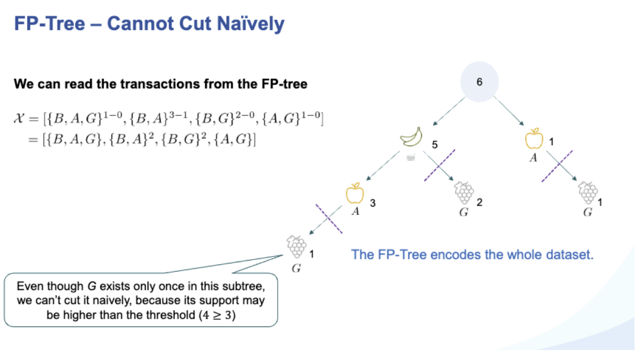


Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

5. Build the FP-tree going through each transaction

The FP-Tree encodes the whole dataset.



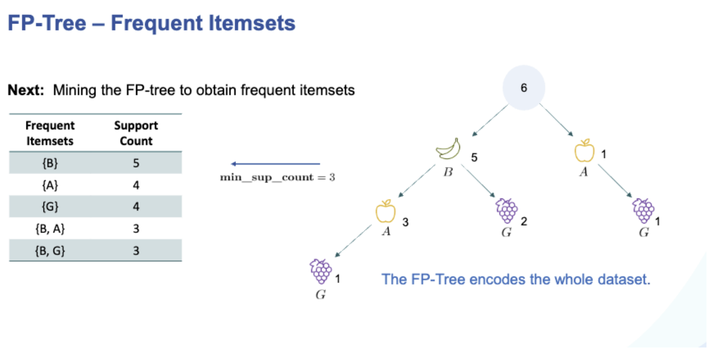
FP-Tree – Cannot Cut Naïvely

We can read the transactions from the FP-tree

$$\mathcal{X} = [\{B,A,G\}^{1-0}, \{B,A\}^{3-1}, \{B,G\}^{2-0}, \{A,G\}^{1-0}]$$
$$= [\{B,A,G\}, \{B,A\}^2, \{B,G\}^2, \{A,G\}]$$

The FP-Tree encodes the whole dataset.

Even though G exists only once in this subtree, we can't cut it naively, because its support may be higher than the threshold (4 ≥ 3)



FP-Tree – Frequent Itemsets

Next: Mining the FP-tree to obtain frequent itemsets

Frequent Itemsets	Support Count
{B}	5
{A}	4
{G}	4
{B, A}	3
{B, G}	3

The FP-Tree encodes the whole dataset.

Key takeaways: FP-Growth uses an FP-tree and DFS to mine patterns with only two passes.

5 Association Rules

5.1 Introduction

5.2 Generating Association Rules

5.3 Evaluation (support, confidence, lift, conviction)

5.4 Applications

5.5 Simpson's Paradox

6 Time Series

6.1 Temporal Data

6.2 Introduction to Time Series

6.3 Analysis

6.4 Forecasting