Measurement Error

- Error Rate: # of errors / # of instances = (FN+FP)/N
- Recall: # found positives / # actual positives = TP/(TP+FN)
- Precision: # found positives / # predicted positives = TP/(TP+FP)
- Specificity = TN/(TN+FP)
- False Alarm Rate = FP/(FP+TN) = 1 - Specificity

Association Rule Mining

For finding **frequent patterns**, associations, corr., causal structures.

Format: $X \Rightarrow Y$[support, confidence]

- ***Support***: $P(X \cap Y)$
  - prob. that transaction **contains both X & Y**
  - indicates **statistical independece** of association rule
- ***Confidence***: $P(Y|X) = \frac{P(X \cap Y)}{P(X)} = \frac{P(X|Y)P(Y)}{P(X)}$
  - prob. that transaction **having X also has Y**
  - the degree of correlation. Measures **rule's strength**

Apriori Algorithm

```
# C[k]: Candidate itemset of size k
# L[k]: Frequent itemset of size k
L[1] = [frequent single items]; k = 2
while L[k-1] is not empty:
  C[k] = generate_candidates_from(L[k-1])
  C[k] = prune_candidates(C[k])
  L[k] = count_support(C[k])
```

Step 1: Generate Candidates

Generate candidates for k-itemset from $(k-1)$-itemset:

```
n = len(L[k-1])
for i in range(n):
  for j in range(i+1, n):
    itemset_i, itemset_j = L[k-1][i], L[k-1][j]
    shared_items = intersection(itemset_i, itemset_j)
    if len(shared_items) == k-2:
      candidate = join(itemset_i, itemset_j)
      C[k].append(candidate)
```

Step 2: Prune Candidates

Prune candidates that do not meet the minimum support threshold:

```
# given k-itemset, prune if any (k-1) subset not in L[k-1]
for cand in C[k]:
  for split in range(len(cand)):
    subset = cand[:split] + cand[split+1:]
    if subset not in L[k-1]:
      C[k].remove(cand)
      break
```

Step 3: Counting

```
cnt = {cand: 0 for cand in C[k]}
for entry in database:
  for cand in C[k]:
    if cand in entry:
      cnt[cand] += 1
L[k] = [cand for cand in C[k] if cnt[cand] >= min_support]
```

Repeat Step 1 & 2 until `L[k]` is empty

Rule Generation

- **For** each *frequent itemset* L, gen all non-empty subsets of L
- **For** each non-empty subset $s \in L$, gen rule $s \Rightarrow (L - s)$

Other Measures

Objective Measures: support & confidence

Subjective Measures: Interest (lift)

$$\text{Lift} = \frac{\text{Conf}(X \Rightarrow Y)}{P(Y)} = \frac{P(X \cap Y)}{P(X)P(Y)}$$

- **Intuition 1**: *Performace (above baseline) Test*
  - $\text{Conf}(X \Rightarrow Y)$ given X happened, prob of Y also happening;
  - $P(Y)$, prob of Y happening in general **(baseline)**
- **Intuition 2**: *Independence Test*
  - $P(X \cap Y)$ **actual observed** prob. of X&Y happening together;
  - $P(X)P(Y)$ expected prob. of X&Y happening together if they were completely independent. **(the independence test)**

Sequential Rule Mining

Step1-Sort Phase: First group data by customer ID. For each customer, sort their transaction by timestamp.

Step2-Freq Itemset: Find all freq/large itemset L (using Apriori).

- Counting rule: number of customers whose sequence contains the itemset at least once

Step3-Transform: Transform each sequence to frequent itemset

- Given: Freq. Itemsets (from Step 2): $\{(A), (B), (A, B), (C)\}$
- Trans: $< (A, B), (C), (D) > \rightarrow < \{(A), (B), (A, B)\}, \{(C)\} >$,
  - (A, B) became $\{(A), (B), (A, B)\}$
  - (C) became $\{(C)\}$
  - (D) is removed, because it is not in Freq. Itemsets

Step4-Sequence: Find sequences using frequent itemsets (AprioriAll)

- Given: transformed seq (from step 3) and freq 1-itemset (from step 2)
- Find frequent k-sequences, by joining $(k-1)$-sequences with the same $(k-2)$ prefix. Prune if any $(k-1)$-subseq is not frequent.
- Count support for each k-sequence, and remove if below threshold.
- Given candidate seq `ABC`, transaction $< (A, B), (C), (D) >$ supports `ABC`; transformed version helps make this counting easier: $< \{(A), (B), (A, B)\}, \{(C)\} >$ **This is why Step3 is needed!!!**

Step5-Maximal Phase:

Sequence s is maximal if s is **not** contained in any other sequences

```
for k in range(max_k, 1, -1):
  for seq in L[k]: for subseq in seq:
      if subseq in L[k-1]: L[k-1].remove(subseq)
```

AprioriAll ALogirhtm

```
L[1] = [frequent 1-itemsets]; k = 2
while L[k-1] is not empty:
  C[k] = generate_candidates_from(L[k-1])
  L[k] = prune_candidates(C[k])
```

Diff. compared to Apriori

`generate_candidates_from`:

- given two $(k-1)$-sequences A and B, generate k-sequence C,
- if `A[:-1] == B[:-1]`, we get: `A + [B[-1]]` and `B + [A[-1]]`
- A and B **must** share first $(k-2)$ items!
- **Example**: permutate last item of $(k-1)$-sequence
  - $< A, B > + < A, C > \Rightarrow < A, B, C >$ and $< A, C, B >$

`prune_candidates`:

- given a k-seq, prune if **any** of its $(k-1)$-subseqs is not frequent.
- **Example**: `L[3] = {abc, abd}`, generate `abdc` and `abcd`
  - Prune `abdc` because `adc` and `bdc` are not in `L[3]`

Decision Tree (DT)

Two Stages

1. Tree construction
2. Tree pruning (rm branch that reflects outliers and noise)

Tree Construction

ID3/C4.5: **information gain**

- Attributes assumed categorical
- Can be modified for continuous-valued attributes

IBM IntelligentMiner: **gini index**

- Attributes assumed continuous
- Need other tools to get possible split values
- Can be modified for categorical attributes

Information Gain

Definition:

- How much did information entropy **decrease by**?
- Information Entropy $\equiv$ impurity
- $\text{Gain}(D, a)$: Information gain if we were to split samples D by attribute a into V subsets $\{D_1, D_2, ..., D_V\}$.

Equations

$$\text{Information} : I(x_i) = -\log_2(p(x_i))$$

$$\begin{aligned}\text{Information Entropy} : \text{Ent}(X) &= E(I(X)) \\ &= \textstyle\sum_{i=1}^{n} p(x_i) I(x_i) \\ &= \textstyle\sum_{i=1}^{n} p(x_i)(-\log_2(p(x_i))) \\ &= -\textstyle\sum_{i=1}^{n} p(x_i)(\log_2(p(x_i)))\end{aligned}$$

$$\text{Information Gain} : \text{Gain}(D, a) = \text{Ent}(D) - \textstyle\sum_{v=1}^{V} \frac{|D_v|}{|D|} \text{Ent}(D_v)$$

$$\text{Impurity After Split} : -\textstyle\sum_{v=1}^{V} \frac{|D_v|}{|D|} \sum_{x_i \in D_v} p(x_i) \log_2 p(x_i)$$

Everything Together:

$$\begin{aligned}&\text{Gain}(D, a) = \\ &= \text{Ent}(D) - \sum_{v=1}^{V} \frac{|D_v|}{|D|} \text{Ent}(D_v) \\ &= -\left(\sum_{i=1}^{n} p(x_i) \log p(x_i) - \sum_{v=1}^{V} \frac{|D_v|}{|D|} \sum_{x_i \in D_v} p(x_i) \log p(x_i) \right)\end{aligned}$$

- For categorical values V **is the number of distinct values of** a
- $|D_v|$ is the size v'th split. $|D|$ is the size of the entire dataset

Regressoin Tree

- Prediction is computed as **avergae** of sample values in the leaf node.
- Impurity measured as **sum of squared deviations** from leaf mean
- Performance measured by **root mean squared error** (RMSE)

Tree Pruning & Enhancements

- Pre-pruning:
  - Halt tree construction early
  - Halt before *goodness measure* falls below certain threshold.
  - (hard to choose appropriate threshold)
- Post-pruning:
  - Removes branches from a "fully grown" tree.
  - Progressively remove branches.
  - Use testing set to decide the best pruned tree.

Other Enhancements

Continuous-valued Attributes:

- Treat each unique continuous attribute value in the dataset as discrete and identify all possible split points between them.
- e.g. [21.6, 22.0, 23.0]. Try splits at: [21.8, 22.5]
- Time Complexity: $O(N)$ after sort. $O(N \log N)$ in total.

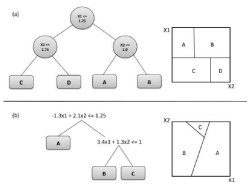
Missing Values:

- Method 1: Assign most common value of that attribute
- Method 2: Assign probability to each of the possible values

Attribute Construction:

- Create new attribute on existing ones (e.g., spare ones)
- Purpose: Reduce fragmentation, repetition, replication

Non-axis Parallel Split



Divide & Conquer

Internal Decision Nodes:

- Univariate: Use single attribute a_i
  - Numeric: Binary split
  - Discrete n-way split, for all n possible values
- Multivariate: use all attributes $a_i \in A$

Leaves:

- Classification: class labels
- Regression: Numeric - avg of leaf nodes / local fit (fit a more sophisticated model locally, e.g., linear regression)

Pseudocode

```
def GenerateTree(x):
  if NodeEntropy(x) < θ_I /* (See I_m on slide 29)
      Create leaf labelled by majority class in x
      return
  i ← SplitAttribute(x)
  for each branch of x[i]:
      Find x[i] falling in branch
      GenerateTree(x[i])

def SplitAttribute(x):
    MinEnt ← MAX
    for all attributes i = 1, ..., d:
        if x[i] is discrete with n values:
            Split x into x[1], ..., x[n] by x[i]
            e ← SplitEntropy(x[1], ..., x[n])
            if e < MinEnt:
                MinEnt ← e
                bestf ← i
        else: # x[i] is numeric
            for all possible splits
                Split x into x[1], x[2] on x[i]
                e ← SplitEntropy(x[1], x[2])
                if e < MinEnt:
                    MinEnt ← e
                    bestf ← i
    return bestf
```

Exam Tips (Rule Extraction Format)

```
R1:  IF (age > 38.5) AND (years-in-job > 2.5) THEN y = 0.8
R2:  IF (age > 38.5) AND (years-in-job ≤ 2.5) THEN y = 0.6
R3:  IF (age ≤ 38.5) AND (job-type = 'A') THEN y = 0.4
R4:  IF (age ≤ 38.5) AND (job-type = 'B') THEN y = 0.3
R5:  IF (age ≤ 38.5) AND (job-type = 'C') THEN y = 0.2
```

Bayes Classification

Given sample X and class C:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)} \text{ aka. posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

Idea: assign sample X the best C such that $P(C|X)$ is maximized.

- $P(X)$ unknown, but constant for all C. So we can ignore it.
- $P(C)$ unknown, but reasonably estimated from dataset
- $P(X|C)$ unknown and infeasible.
  - Need to know the joint probability $P(x_1, x_2, ..., x_n \mid C)$.
  - If each x_i takes 2 values, we can have 2^n possible combinations.
  - Probability that any such sample already exist in our dataset ≈ 0

Naive Bayes Classifier

Assumes attributes are **conditionally independent**:

$$P(C|X) \propto P(X) \prod_{i=1}^{n} P(x_i \mid C)$$

How to estimate $P(x_i|C)$?

- If x_i categorical: relative frequency of x_i in class C in dataset
- If x_i continuous: gaussian density via all i-th attr. in class C

KNN (K-Nearest Neighbors)

Instance-Based Methods *(a.k.a memory-based)*

- Instance-based classification: Store training examples, delay processing until a new instance must be classified **("Lazy Evaluation")**

Examples:

- k-nearest neighbors: instances as points in Euclidean space.
- Case-based reasoning: symbolic representation and knowledge-based inference

Procedure

- Time Complexity: $O(n \log k)$ via min-heap of size k

1. Compute distance to all other training records.
2. Sort the distance, and find k nearest neighbors
3. Use the k nearest neighbors to determine the class label of the unknown sample (e.g., through majority voting)

- Increasing k: Less sensitive to noise
- Decreasing k: Capture finer structure

Distance Metrics

- Continuous Attributes: Euclidean Distance

- **normalize** each dimension by standard deviation
- Discrete Data: use hamming distance

Curse of Dimensionality

Definition: Prediction accuracy degrade quickly when number of attributes grow, because

- When many irrelevant attributes involved, relevant attributes are shadowed, and distance measurements are unreliable

Solution:

- Remove irrelevant attributes in pre-processing (e.g., dimensionality reduction such as PCA)

Ensemble Methods

Advantages: Improved accuracy; other types of classifiers can be directly included; Easy to implement; Not too much parameter tuning
Disadvantage: Black box; Not compact representation.

Who Minimize Variance:

- Bagging
- Random Forest

Who Minimize Bias:

- Boosting
- Ensemble Selection

Proof

Consider binary classification problem $y \in \{-1, +1\}$, with ground truth function f, base classifier h_i, and base classifier error rate ε:

- Base Classifier Error Rate: $P\big(h_{i(x)} \neq f(x)\big) = \varepsilon$
- Ensemble Classifier: $H(x) = \text{sign}\big(\sum_i^T h_{i(x)}\big)$

Assume independence of h_i error rates. From *Hoeffding Inequality*:

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\varepsilon)^k \varepsilon^{T-k}$$
$$\leq \exp\Big(-\frac{1}{2}T(1-2\varepsilon)^2\Big)$$

Therefore, as the number of base classifiers T increase, the probability of misclassification decrease exponentially

Bagging

Bagging = **B**ootstrap **Agg**regation

Traditional Bagging

1. Takes original dataset D with N training examples
2. Creates M **different** copies $\{D_m\}_{m=1}^M$ via
   - Sampling from D with replacement
   - Each D_m has same number of examples as D
3. Train base classifiers $h_1, ..., h_M$ using $D_1, ..., D_m$
4. Average / Vote model as the final ensemble $h = \frac{1}{M}\sum_{m=1}^M h_m$

Random Forest

1. Takes original dataset D with N training examples
2. Creates M **different** copies $\{D_m\}_{m=1}^M$ via
   - Sampling from D with replacement
   - Each D_m has same number of examples as D
3. Train **decision trees** $t_1, ..., t_M$ using $D_1, ..., D_m$
   - suppose there are k features
   - **randomly sample k' ($\leq k$) features** at each split
   - when $k' = k$ its indifferent to traditional DTs
   - usually choose $k' = \log_2 k$ (or $\sqrt{k}$ according to Korris)
4. Average / Vote model as the final ensemble $h = \frac{1}{M}\sum_{m=1}^M h_m$

Intuition: By randomizing not just samples (step 2), but also features (step 3), we introduce even greater variety compared to *Bagging*, reducing correlation. (The entire premise of ensemble method is independence between classifiers)

Boosting

Basic Idea

$$H(x) = \sum_{t=1}^T \alpha_t h_t(x)$$

- Train T weak learners (through T iterations)
- Each weak learner:
  - **only** have to be slightly better than random
  - Focuses on difficult data points

Procedure

- Initialize **weights** for the N samples.
  - $D_1 = \{w_{11}, ...w_{1N}\}$ is uniform distribution ($\forall w_{1i} = 1/N$)
  - D_t means the weighted dataset at t iteration
- **For $t = 1, 2, ..., T$ do:**

1. Train h_t on weighted dataset D_t, and compute error of ε_t

$$\varepsilon_t = \sum_{i=1}^N P_{x\sim D_t}(h_t(x_i) \neq y_i) = \sum_{h_{t(x_i)} \neq y_i} w_{ti}$$

2. Use h_t's error rate ε_t to determine its weight α_t in the ensemble

$$\alpha_t = \frac{1}{2}\log\frac{1-\varepsilon_t}{\varepsilon_t}$$

   - if $\varepsilon_t > 0.5$, then $\alpha_t < 0$ and exponentially decrease.
   - if $\varepsilon_t < 0.5$, then $\alpha_t > 0$ and exponentially increases.

3. Use h_t's weight α_t to determine the **next** weighted dataset D_{t+1}

$$D_{t+1} = \{w_{t+1,1}, ..., w_{t+1,N}\}$$

$$w_{t+1,i} = \begin{cases} w_{ti} \times \exp(\alpha_t) & \text{if } h_{t(x_i)} \neq y_i \text{ (incorrect prediction)} \\ w_{ti} \times \exp(-\alpha_t) & \text{if } h_{t(x_i)} = y_i \text{ (correct prediction)} \end{cases}$$

   - Wrong $\rightarrow w_{t+1,i} \uparrow$; correct $\rightarrow w_{t+1,i} \downarrow$
   - Change magnitude $\propto \exp(\alpha_t)$
   - because high h_t importance α_t requires:
     - ... much higher weights for misclassified samples
     - ... much lower weights for correct samples
     - to balance error impact

4. Normalize D_{t+1} by so that it sums to 1

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^N w_{t+1,j}}$$

- Output "boosted" ensemble $H(x) = \text{sign}\big(\sum_{t=1}^T \alpha_t h_t(x)\big)$

XGBoost

- All of the advantages of gradient boosting
- CPU parallelism by default
- Low runtime, high model performance

Boosting vs Bagging

- Bagging computationally efficient
- Both reduce variance
- Bagging can't reduce bias, but boosting can
- Bagging is better when we don't have high bias, and only want to reduce variance. (e.g., when overfitting)

Clustering

Requirements for clustering in data mining:

- Scalability
- Ability to deal with different types of attributes
- Discovery of clusters with arbitrary shape
- Minimal req. for domain knowledge to determine input parameters
- Able to deal with noise and outliers
- Insensitive to order of input records
- High dimensionality
- Incorporation of user-specified constraints
- Interpretability and usability

Distance Metrics (Continuous Variables)

Minkowski Distance (L_q distance general form)

$$d(\vec{i}, \vec{j}) = \sqrt[q]{\sum_{k=1}^p |x_{ik} - x_{jk}|^q} = \sqrt[q]{|x_{i1} - x_{j1}|^q + ... + |x_{ip} - x_{jp}|^q}$$

Manhattan Distance (L_1 distance)

$$d(\vec{i}, \vec{j}) = \sum_{k=1}^p |x_{ik} - x_{jk}| = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + ... + |x_{ip} - x_{jp}|$$

Distance Metrics (Binary Variables)

	1	0	sum
1	a	b	$a+b$
0	c	d	$c+d$
sum	$a+c$	$b+d$	p

Simple Matching:
$d(i,j) = (b+c)/(a+b+c+d)$
Jaccard Coefficient:
$d(i,j) = b/(a+b+c)$

Distance Metrics (Categorical Variables)

Simple Matching

$$d(\vec{i}, \vec{j}) = (\text{total features} - \text{matched features}) / \text{ total features}$$

1-Hot Encoding: Convert categorical variable into 1-hot encoding, then use aforementioned binary variable metrices

Distance Metrics (Transactional Data)

$$\text{Sim}(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} \quad \text{E.g., Sim}(\{a,b,c\}, \{c,d,e\}) = \frac{|\{c\}|}{|\{a,b,c,d,e\}|} = \frac{1}{5}$$

K-means Algorithm

1. **Initialization**: Partition objects into k nonempty subsets
2. **Mean-op**: Compute seed points as centroids of the clusters of the current partition. The centroid is the mean point of the cluster.
3. **Nearest_Centroid-op**: Assign objs to cluster w/ nearest centroid.
4. **Go back to the step 2, stop when no more new assignment.**

Strengths

- Relatively efficient: $O(tkn)$, where n is # objects, k is # clusters, and t is # iterations. Normally, $k, t \ll n$.
- Often terminates at local optimum. Global optimum may be found using: deterministic annealing and genetic algorithms

Weaknesses

- Applicable only when mean is defined, then what about categorical data? What is the mean of red, orange and blue?
- Need to specify k, the number of clusters, in advance
- Unable to handle noisy data and outliers
- Not suitable to discover clusters with non-convex shapes; the basic cluster shape is spherical (convex shape)

Handling Categorical Data

- Replacing means of clusters with modes
- Using new dissimilarity measures to deal with categorical objects
- Using a frequency-based method to update modes of clusters
- A mixture of categorical and numerical data: k-prototype method

K-medoids Algorithm

Same as K-means, but instead of using virtual mean, use real data points as centroids. **Strength**: medians less sensitive to outliers

How To Choose Medoids?

In ***PAM*** (Partitioning Around Medoids) algorithm, enumerate all possible medoids, and choose one w/ least total distance to other points.

In ***CLARA*** (Clustering Large Applications) algorithm, random sample a **fixed subset** from the whole dataset. Then use it to perform PAM.

Hierarchical Clustering

- **Agglomerative (ANGES)**: Bottom-up approach
- **Divisive (DIANA)**: Top-down approach

Agglomerative (ANGES)

1. Initialize: 1 sample per cluster
2. Merge a pair of clusters with **least dissimilarity**
3. Decrement # of clusters by one
4. **UNTIL** only 1 cluster left.

Weaknesses:

- Do not scale well: time complexity of at least $O(n^2)$, where n is # total objects (need to compute the similarity or dissimilarity of each pair of objects)
- Can never undo what was done previously
- Hierarchical methods are biased towards finding 'spherical' clusters even when the data contain clusters of other shapes.
- Partitions are achieved by 'cutting' a dendrogram or selecting one of the solutions in the nested sequence of clusters that comprise the hierarchy.
- Deciding of appropriate number of clusters for the data is difficult.

Divisive (DIANA)

1. Initialize: 1 cluster containing all samples
2. Split a cluster into 2 **most dissimilar** sub-clusters
3. Decrement # of clusters by one

Linkages

distance measured by...
Single Linkage: **closest pair** of samples from each cluster
Complete Linkage: **furthest pair** of samples from each cluster
Centroid Linkage: **mean** (centroids) of each cluster $\frac{1}{n}\sum_{i=1}^n x_i$

Standards

- Intra-cluster high similarity, inter-cluster low similarity
- **Choose K**: Elbow method

DBSCAN Algorithm

Short for: Density Based Spatial Clustering of Applications with Noise

- ε: radius for the neighborhood of point
- `MinPts`: minimum number of points in the ε-neighborhood
  - **include** itself, and **include** all visited points.

- **核心点 (Core Point)**：“大佬”。
  - 他的圈子（ϵ）里至少有 MinPts 个人。
  - *作用*：只有大佬有资格发展下线，以此驱动聚类的扩张。
- **边界点 (Border Point)**：“跟班”。
  - 他自己的圈子里人不够（少于 MinPts），但是他正好在大佬的圈子里。
  - *作用*：他是聚类的边缘，属于这个帮派，但不能继续向外扩张（因为他人微言轻，密度不够）。
- **噪声点 (Noise/Outlier)**：“孤独者”。
  - 既不是大佬，也不是大佬的跟班。
  - *作用*：被算法无情抛弃。

Procedure:

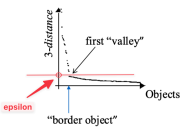
1. Start w/ any **unvisited** point P.
2. Is P core point? (has at least `MinPts` points in ε sphere)
   - If **yes**: Mark P as core.
   - If **no**: Mark P as noise (temporarily). Back to Step 1.
3. **Expansion**: Add all P's neighbors to the cluster
   - If neighbor Q is core point, add Q's neighbors to the cluster
   - If neighbor Q is border point, skip.
   - Repeat above until all reachable points are border points.

Weaknesses:

- Cannot handle varying densities.
- Sensitive to hyperparameters.

How to Find ε and `MinPts`?

1. Fix `MinPts` (usually $2 \times d - 1$), where d is dimension of data space
2. For each p, compute its distance to its k-th nearest neighbor $d_k(P)$
   - For p in dense clusters, the value of $d_k(p)$ is relatively small.
   - For outliers p, the value of $d_k(p)$ is relatively large.
3. **Sort all $d_k(p)$ in descending order**, and plot $d_k(p)$
4. Find the "elbow" point, and use it as ε.



In this example: k is chosen as 3, hence the "3-distance" y-axis.
left: high $d_k(p) \rightarrow$ outliers sparse points (varying, high value)
right: low $d_k(p) \rightarrow$ intra-cluster points (smooth, low value)

Data Preprocessing

Stage 1: Data Cleaning

Garbage In, Garbage Out. Thus we need preprocessing. Real data is:

- Incomplete: lack attribute values, lack certain attributes of interest, or containing only aggregate data
- Noisy: containing errors, outliers
- Inconsistent: conflicting discrepencies in codes or names

Missing Value ***(Worst to Best)***

1. **worst**: Ignore Tuple. Unless missing label, this loss information.
2. **okay**: Manual imputation, fill constant (e.g., unknown) or mean.
3. **best**: Inference. Association Analysis to infer missingness.

Noisy Data

1. **Binning**: Discretize the variable
   - Equal-width (distance) partitioning: Divide range into N intervals of equal size. If the lowest-highest values of the attribute is A and B, interval width = $(B-A)/N$
     - Stragihtforward. But **sensitive to outliers** or **skewness** .
   - Equal-depth (frequency) partitioning: Divide range into N intervals, each containing (about) the same number of samples.
     - Good data scaling. Managing categorical variable can be tricky.
2. **Clustering**: Use distance-based clustering to find outliers
3. **Regression**: fit linear model. Data points far from fit are outliers.

Smoothing:

1. Binning: smooth by bin means or bin boundary
2. Clustering: replace noisy data with cluster centroid
3. Regression: replace noisy data with predicted value (project to line)

Stage 2: Data Transformation

1. Min-Max: $v' = \frac{v-\min}{\max-\min}(\max_{\text{new}} - \min_{\text{new}}) + \min_{\text{new}}$
   - $[\min_{\text{new}}, \max_{\text{new}}]$ is the **new range** of the variable (e.g., $[-1, 1]$)
2. Z-score: $v' = \frac{v-\text{mean}}{\sigma}$
3. Decimal Scaling: $v' = \frac{v}{10^j}$, where j is smallest integer such that $\max(|v'|) \leq 1$

Stage 3: Data Reduction

1. Reduction

- *Feature Selection*: choose $k < d$ important features, ignore rest.
  - Forward Search: start with $\emptyset$ features. Each iteration, add best new feature. Essentially, **Greedy Hillclimbing**
  - Backward Search: start with all features. Each iteration, remove worst feature.
  - Decision Tree Induction: Use only the features used by DT.
- *Feature Extraction*: project d-dimensional data to $k < d$ dimension
  - Principal Component Analysis (PCA)
  - Linear Discriminant Analysis (LDA)
  - Factor Analysis
- *Numerosity Reduction*: reduce data vol. w/ compact representation
  - Parametric: Fit data w/ model. Store just parameters, discard data.
  - Non-parametric: histograms, clustering, (stratified) sampling.

2. Discretization

- Divide range of continous attribute into intervals (e.g., by binning)
  - Why?
    - Some algo only accept categorical attributes
    - Reduce data size. Prepare for further analysis.
- Concept hierarchy: replace low-level concept w/ higher level concepts (e.g., "age = 25" → "young")

Data Warehouse

	OLTP	OLAP
users	clerk, IT professional	knowledge worker
function	day to day operations	decision support
DB design	application-oriented	subject-oriented
data	current, up-to-date detailed, flat relational isolated	historical, summarized, multidimensional integrated, consolidated
usage	repetitive	ad-hoc
access	read/write index/hash on prim. key	lots of scans
unit of work	short, simple transaction	complex query
# records accessed	tens	millions
#users	thousands	hundreds
DB size	100MB-GB	100GB-TB
metric	transaction throughput	query throughput, response

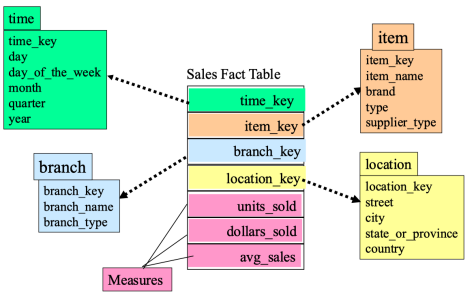
Core Concepts

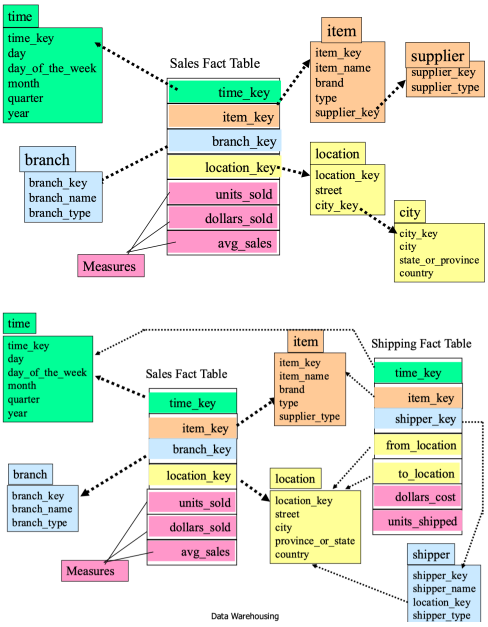
Definition: A *subject-oriented, integrated, time-variant, nonvolatile* collection of data to support management decision making.

OLTP vs OLAP:

- **OLTP (Online Transaction Processing)**:
  - *Access*: Read/Write, atomic transactions.
  - *Focus*: Concurrency (ACID).
  - *Data*: Current, detailed, relational (3NF). *Users*: Clerks, DBAs.
- **OLAP (Online Analytical Processing)**:
  - *Access*: Read-only (mostly), complex aggregate queries.
  - *Focus*: Throughput.
  - *Data*: Historical, consolidated, multi-dim.
  - *Users*: Managers, Analysts.

Multidimensional Data Model





Data Cube (N-dim Tensor):

- **Dimensions**: Perspectives (Time, Loc).
- **Measures**: Numerical facts (Sales).
- **Fact Table**: Contains keys to dimensions + measures.

Schemas (Modeling): **Star Schema**:

- *Structure*: One central Fact Table + set of Dimension Tables.
- *Logic*: Dim tables are **un-normalized**.
  - (e.g. `City`, `Country` in same table).
- *Pros*: Simple, Min Joins (Fast).
- *Cons*: Redundant data (who cares? storage cheap).

Snowflake Schema:

- *Structure*: Dim tables are **normalized** (split into hierarchies).
- *Pros*: No redundancy, easy maintenance.
- *Cons*: More Joins (Slow). Avoid in DW.

Fact Constellation:

- *Structure*: Many fact Tables share dimension tables (Galaxy schema).
- *Case*: Complex systems (e.g. sales & shipping share time & location).

Aggregations Functions

Essentially: how compress-able are the states?

Distributive:

- Examples: `sum`, `count`, `max`
- Characteristic: $F(D) = F(\{F(D_1), F(D_2), ..., F(D_n)\})$
- Example: $\max(D) = \max(\{\max(D_1), \max(D_2), ..., \max(D_n)\})$
- Local solutions can be perfectly aggregated to global solution!
- Communication Complexity: $O(1)$

Algebraic:

- Examples: `avg`, `var`
- Characteristics: $F(D) \neq F(\{F(D_1), F(D_2), ..., F(D_n)\})$
- Example: $\text{avg}(D) \neq \text{avg}(\{\text{avg}(D_1), \text{avg}(D_2), ..., \text{avg}(D_n)\})$
- Solution:
  - pass fixed size (k) feature vector or sufficient statistic.
  - don't pass avg. Pass $\vec{v} =< \sum x_i, N >$
  - don't pass stdev. Pass $\vec{v} =< \sum x_i, \sum x_i^2, N >$
- Not as friendly as *Distributive*, but still quite efficient!
- Communication Complexity: $O(k)$

Holistic:

- Examples: `median`, `rank`, `mode`
- Characteristics: size of return state grows with data size.
- Storage requirement for sub-aggregate has no constant upperbound

OLAP Operations (Geometric Transforms)

Roll-up (Drill-up):

- *Action*: Climbing up hierarchy or Dimension reduction.
- *SQL*: `GROUP BY (Day)` → `GROUP BY (Month)`.

Drill-down:

- *Action*: Stepping down hierarchy or Introducing new dimension.
- *SQL*: Reverse of Roll-up. Detailed view.

Slice & Dice:

- *Slice*: Selection on **one** dimension. `WHERE time=Q1`
- *Dice*: Selection on **two+** dimensions. `WHERE time=Q1 AND loc=US`

Pivot (Rotate):

- *Action*: Rotate axes for visualization. (kind of like transpose)

Implementation & Optimization

1. Partial Materialization (Cube Computation):

- *Problem*: 2^N cuboids. Full materialization too big. No mat. too slow.
- *Solution*: Greedy Algo. Select views with max benefit/space ratio.
- *Iceberg Cube*: Only store cells w/ value > threshold (sparsity).

2. Indexing Techniques:

- *Bitmap Index*: For low-cardinality columns (Gender). fast.
- *Join Index*: Map `Join(Fact.key, Dim.key)` to avoid runtime joins.

3. Architecture:

- *ROLAP*: Relational (Star schema). Scalable, slower.
- *MOLAP*: Multi-dim Array (Direct cube). Fast, sparse, limited scale.
- *HOLAP*: Hybrid (Detail in ROLAP, Aggregates in MOLAP).

Web Mining

Input: Unstructured/Semi-structured Data. **Goal**: Discover patterns.

1. Information Retrieval (Text DB)

Data Abstraction (Feature Eng):

- **Bag of Words**: Document d as set of terms. Ignore order.
- **Vector Space Model**: $d \in \mathbb{R}^{|V|}$ (High-dim, sparse).
  - Value = Term Frequency (TF) or TF-IDF.
  - *Preprocessing*:
    - Stop list (remove 'the'),
    - Stemming ('mining' → 'mine').

Similarity (Metric):

- Euclidean fails in high-dim (curse of dimensionality) & length bias.
- **Cosine Similarity**: Measure angle (direction), not magnitude.

$$\text{Sim}(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\| \times \|d_2\|}$$

Evaluation (Set Theory):

- **Precision**: $\frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{|\{\text{Retrieved}\}|}$ (Quality of result).
- **Recall**: $\frac{|\{\text{Relevant}\} \cap \{\text{Retrieved}\}|}{|\{\text{Relevant}\}|}$ (Coverage of truth).

Challenges (Why Keyword Match Fails):

- **Synonymy**: Different words, same meaning (e.g., "ML" vs "Learning"). → Low Recall.
- **Polysemy**: Same word, different meanings (e.g., "Apple"). → Low Precision.

2. Taxonomy (The 3 Pillars)

Content Mining (NLP):

- *Input*: HTML Text. *Task*: Classification/Clustering.
- *Logic*: Text Mining + HTML Tags weight.

Structure Mining (Graph):

- *Input*: Links (Edges). *Task*: Find Authority/Hubs.
- *Logic*: "Link as Vote". Graph Theory.

Usage Mining (Logs/BI):

- *Input*: Server Logs (IP, Time, URL). *Task*: User Profiling.
- *Logic*: ETL → Sessionize → Pattern Mining.

3. Core Algorithms

PageRank (Google):

- *Intuition*: Link=endorsement. Importance via recursive voting.
- *Logic*: Random Surfer Model / Markov Chain Stationary Dist.

$$\text{PR}(A) = (1-d) + d \sum_{T_i \to A} \frac{\text{PR}(T_i)}{C(T_i)}$$

HITS (Hubs & Authorities):

- **Hub**: Guide page pointing to many authorities.
- **Authority**: Content page pointed to by many hubs.
- Mutually reinforcing: Good Hub → points to good Auth.

Web Usage Process:

1. **Clean**: Remove crawlers, graphics requests.
2. **Identify**: Split IP streams into User Sessions.
3. **Pattern**: Apriori / Sequential Mining on sessions.

Exam Questions

Network → Matrix → DBSCAN

1. Analysis and Parameter Proposal

The dissimilarity matrix (Fig. 4) represents **structural equivalence** (Hamming distance of adjacency vectors) rather than path length. Notably, hubs D and E have high dissimilarity to others ($dge3$) and to each other ($d = 6$).

To distinguish tight structural groups from loose connections:

- **Radius (ε) = 2**: A threshold of $\varepsilon \geq 3$ would merge the entire graph into a single cluster (since $d(A, D) = 3, d(A, E) = 3$). $\varepsilon = 2$ exploits the natural gap in the data to separate strong ties.
- **MinPts = 2**: Chosen to allow the detection of the smallest possible social units (dyads) visible in the graph (e.g., A-F).

2. Execution (Neighborhood Analysis)

We calculate the ε-neighborhood $N_{\{\varepsilon\}}(x) = \{ym\,\text{id}\, d(x, y) \leq 2\}$ for each node.

Node	Neighborhood ($dle2$)	Status (MinPts=2)
A	$\{A, F\}$ (Size: 2)	Core Point
F	$\{F, A\}$ (Size: 2)	Core Point
B	$\{B, C\}$ (Size: 2)	Core Point
C	$\{C, B\}$ (Size: 2)	Core Point
D	$\{D\}$ (Size: 1, min dist is 3)	**Noise**
E	$\{E\}$ (Size: 1, min dist is 3)	**Noise**

3. Final Clustering Result

DBSCAN expands clusters from the core points by merging overlapping neighborhoods.

- **Cluster 1**: $\{A, F\}$ (High structural similarity)
- **Cluster 2**: $\{B, C\}$ (High structural similarity)
- **Outliers**: D and E. Despite being central graph nodes, they are **structurally unique** (dissimilar neighbors) and fail to meet the density criteria.

Datawarehouse

1. **Missing Value Imputation**
   The missing values are filled based on the class defined by attributes "Fever" and "Disease".

   1. Imputing Age (Patient 9910115)
   - **Class Definition:** Fever = "No", Disease = "Yes".
   - **Matching Records:**
     - Patient 9303034: Age 55
     - Patient 9910111: Age 46
   - **Calculation (Mean):** $\frac{55+46}{2} = 50.5$

   2. Imputing Sex (Patient 9576737)
   - **Class Definition:** Fever = "Yes", Disease = "No".
   - **Matching Records:**
     - Patient 9100123: Male
     - Patient 9576732: Male
   - **Calculation (Mode):** Male

   Answer:
   - Missing Age: **50.5**
   - Missing Sex: **Male**

2. **Data Smoothing by Bin Means**
   Raw Data (Age): $\{65, 55, 12, 35, 46, 16, 105, 28\}$

   Step 1: Sort the Data $\{12, 16, 28, 35, 46, 55, 65, 105\}$

   Step 2: Partition into 4 Equi-depth Bins Total items = 8. Items per bin = $\frac{8}{4} = 2$.
   - Bin 1: $\{12, 16\}$
   - Bin 2: $\{28, 35\}$
   - Bin 3: $\{46, 55\}$
   - Bin 4: $\{65, 105\}$

   Step 3: Calculate Bin Means and Smooth
   - Bin 1 Mean: $\frac{12+16}{2} = 14$
   - Bin 2 Mean: $\frac{28+35}{2} = 31.5$
   - Bin 3 Mean: $\frac{46+55}{2} = 50.5$
   - Bin 4 Mean: $\frac{65+105}{2} = 85$

   Smoothed Values List: Bin 1: 14, 14 **Bin 2:** 31.5, 31.5 **Bin 3:** 50.5, 50.5 **Bin 4:** 85, 85

3. **3D Data Cube Design**
   Context: Hospital Revenue Analysis. **Star Schema Structure:**

   1. Dimension Tables:
   - **Time Dimension:** Attributes include `Time_Key`, `Day`, `Month`, `Quarter`, `Year`. Provides temporal granularity.
   - **Location Dimension:** Attributes include `Location_Key`, `Department_Name`, `Hospital_Branch`, `City`, `Region`. Provides geographical/organizational hierarchy.
   - **Treatment Dimension:** Attributes include `Treatment_Key`, `Procedure_Code` (e.g., CPT), `Category` (e.g., Surgery, Pathology), `Risk_Level`. Describes the service provided.

   2. Fact Table:
   - **Revenue_Fact:** Contains foreign keys linking to the three dimensions (`Time_Key`, `Location_Key`, `Treatment_Key`) and numerical measures such as `Total_Revenue` ($) and `Patient_Count`.

4. **OLAP Operations**
   Based on the context in (c), assume the current query view is: **"Total Revenue by Month and Hospital Branch."**

   1. Roll Up (Aggregation/Generalization)
   - **Operation:** Climb up the concept hierarchy on the **Time** dimension.
   - **Action:** Change grouping from `Month` to `Quarter` or `Year`.
   - **Result:** The data becomes less detailed, showing total revenue per Quarter/Year, summarizing the monthly variations.

   2. Drill Down (Detailed View/Specialization)
   - **Operation:** Step down the concept hierarchy on the **Location** dimension.
   - **Action:** Expand `Hospital_Branch` to `Department_Name`.
   - **Result:** The data becomes more detailed, breaking down the branch's revenue to show specifically which departments (e.g., ER, Oncology, Pediatrics) generated the funds.

Sequence Mining

Section B [80%]: Long Questions

B1. [20 marks]

Given the following tourist data records... (Refer to original image for full dataset) Mappings: Hamburg (H), Toronto (T), Osaka (O), Beijing (B), London (L), Vancouver (V), Sydney (S).

Tourist ID	Sequence
30001	V → T → O → B → L
30002	T → B → O → T → H
30003	O → T → B → O
30004	B → H
30005	B → S → O → H → O → H

a) List ALL possible sub-sequences (with different lengths) arising from Tourist 30003. (6 marks)
Answer:

The sequence for Tourist 30003 is $\langle O, T, B, O \rangle$. A sub-sequence is formed by deleting zero or more items from the original sequence while maintaining the relative order of the remaining items.

Length 1:
- $\langle O \rangle$
- $\langle T \rangle$
- $\langle B \rangle$

Length 2:
- $\langle O, T \rangle$
- $\langle O, B \rangle$
- $\langle O, O \rangle$
- $\langle T, B \rangle$
- $\langle T, O \rangle$
- $\langle B, O \rangle$

Length 3:
- $\langle O, T, B \rangle$
- $\langle O, T, O \rangle$
- $\langle O, B, O \rangle$
- $\langle T, B, O \rangle$

Length 4:
- $\langle O, T, B, O \rangle$

Note: While the element O appears twice, strictly unique patterns are listed above. If treating indices as distinct, one might list $\langle O, T \rangle$ twice (index 1 & 2 vs index 4 & 2), but in sequential mining pattern generation, we focus on the unique sequence signatures.

b) What is the largest itemset size (i.e., the maximum number of items in an itemset) found by the frequent itemset phase of sequential association rule mining? What is the longest sequence length (i.e., the maximum number of itemsets in a sequence) found by the sequence phase of sequential association rule mining? Note that the minimum support is unknown and you may assume it $> 0\%$. State your other assumption(s) when necessary. (4 marks)
Answer:

1. **Largest Itemset Size: 1**
   - *Reasoning:* The provided data records show events occurring strictly sequentially (e.g., Vancouver → Toronto). There are no concurrent events indicated (e.g., no entries like "(Vancouver, Toronto)" meaning same-time visit). Therefore, every itemset (element) in the database contains exactly 1 item. The mining process cannot find an itemset larger than the input data supports.
2. **Longest Sequence Length: 6**
   - *Reasoning:* The longest sequence refers to the maximum number of sequential events (itemsets). Tourist 30005 has the sequence $\langle B, S, O, H, O, H \rangle$, which has a length of 6.
   - *Assumption:* We assume the unknown minimum support is low enough (e.g., $1/N$) to make the longest transaction in the database frequent. If support were effectively 0%, the algorithm discovers the longest sequence existing in the dataset.

c) Show the transformation step (step 3 of the sequential pattern mining process) for Tourist 30005 using $\text{min\_sup} = 35\%$**. (4 marks)**
Answer:

Step 1: Determine Support Threshold

Total Tourists (N) = 5.

$\text{min\_sup} = 35\%$.

Required Count $= 0.35 \times 5 = 1.75$.

Items must appear in at least **2** sequences to be considered frequent.

Step 2: Identify Frequent Items (1-sequences)
- **V:** Count 1 (30001) → Fail.
- **T:** Count 3 (30001, 30002, 30003) → Keep.
- **O:** Count 4 (30001, 30002, 30003, 30005) → Keep.
- **B:** Count 5 (All) → Keep.
- **L:** Count 1 (30001) → Fail.
- **S:** Count 1 (30005) → Fail.
- **H:** Count 3 (30002, 30004, 30005) → Keep.

Frequent Items set: $\{T, O, B, H\}$.

Step 3: Transformation

Map the original sequence of Tourist 30005 by removing infrequent items (S) and retaining frequent items (B, O, H) while preserving order.

Original Sequence: $\langle B \to S \to O \to H \to O \to H \rangle$
- B: Keep
- S: Remove (Infrequent)
- O: Keep
- H: Keep
- O: Keep
- H: Keep

Transformed Sequence:

$$\langle B, O, H, O, H \rangle$$

d) Compute the support, confidence and interest of the following sequential association rule.

Beijing Osaka ⇒ Hamburg

(6 marks)

Answer:

We interpret the rule as: Given a sequence contains Beijing (B) followed eventually by Osaka (O), does it also contain Hamburg (H) eventually after O?
- **Rule:** $\langle B, O \rangle \Rightarrow \langle H \rangle$
- **Composite Sequence:** $\langle B, O, H \rangle$

1. Calculate Counts (N=5):

- **Antecedent Support Count ($\langle B...O \rangle$):**
  - 30001: $\langle V, T, O, B, L \rangle$ (Order is $O \to B$, not $B \to O$) → No.
  - 30002: $\langle T, \boldsymbol{B}, \boldsymbol{O}, T, H \rangle$ → Yes (1).
  - 30003: $\langle O, T, \boldsymbol{B}, \boldsymbol{O} \rangle$ → Yes (2).
  - 30004: $\langle B, H \rangle$ (No O) → No.
  - 30005: $\langle \boldsymbol{B}, S, \boldsymbol{O}, H, O, H \rangle$ → Yes (3).
  - $\text{Count}(\text{Antecedent}) = 3$.
- **Rule Support Count ($\langle B...O...H \rangle$):**
  - 30002: $\langle ...B...O...H \rangle$ → Yes (1).
  - 30003: Contains B, O, but no H. → No.
  - 30005: $\langle ...B...O...H... \rangle$ → Yes (2).
  - $\text{Count}(\text{Rule}) = 2$.
- **Consequent Support Count (H occurs in sequence):**
  - Appears in 30002, 30004, 30005.
  - $\text{Count}(H) = 3$.

2. Compute Metrics:

- **Support (s):**

$$s = \frac{\text{Count}(\langle B, O, H \rangle)}{N} = \frac{2}{5} = \mathbf{0.4} \text{ or } \mathbf{40\%}$$

- **Confidence (c):**

$$c = \frac{\text{Count}(\langle B, O, H \rangle)}{\text{Count}(\langle B, O \rangle)} = \frac{2}{3} \approx \mathbf{0.67} \text{ or } \mathbf{66.7\%}$$

- **Interest (Lift):**

$$\text{Interest} = \frac{\text{Confidence}}{P(\text{Consequent})} = \frac{2/3}{3/5} = \frac{2}{3} \times \frac{5}{3} = \frac{10}{9} \approx \mathbf{1.11}$$

Naive Bayes

Outlook	Temp	Humid	Wind	Play?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Query: $X = (\text{Sunny}, \text{Cool}, \text{High}, \text{Strong}) - \text{Play?}$

Step 1: Prior Probabilities

$$P(\text{Yes}) = \frac{9}{14}, \quad P(\text{No}) = \frac{5}{14}$$

Step 2: Likelihoods (from training counts)

Feature	P(feat|Yes)	P(feat|No)
Sunny	2/9	3/5
Cool	3/9	1/5
High	3/9	4/5
Strong	3/9	3/5

Step 3: Apply Naive Bayes

$$P(X|\text{Yes}) = \frac{2}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} = \frac{54}{6561} \approx 0.0082$$

$$P(X|\text{No}) = \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} = \frac{36}{625} \approx 0.0576$$

Unnormalized posteriors:

$$P(\text{Yes}) \cdot P(X|\text{Yes}) = \frac{9}{14} \cdot \frac{54}{6561} \approx 0.0053$$

$$P(\text{No}) \cdot P(X|\text{No}) = \frac{5}{14} \cdot \frac{36}{625} \approx 0.0206$$

Step 4: Normalize & Classify

$$P(\text{Yes}|X) = \frac{0.0053}{0.0053 + 0.0206} \approx \mathbf{20.4\%}$$

$$P(\text{No}|X) = \frac{0.0206}{0.0053 + 0.0206} \approx \mathbf{79.6\%}$$

Prediction: No (Don't Play Tennis)

Laplace Smoothing (for zero counts)

$$P(x_i \mid C) = \frac{N_{x_i,C} + \alpha}{N_C + \alpha \cdot k}$$

where $\alpha = 1$ (typically), k = number of feature values.