How to generalize histograms to high dimensions?

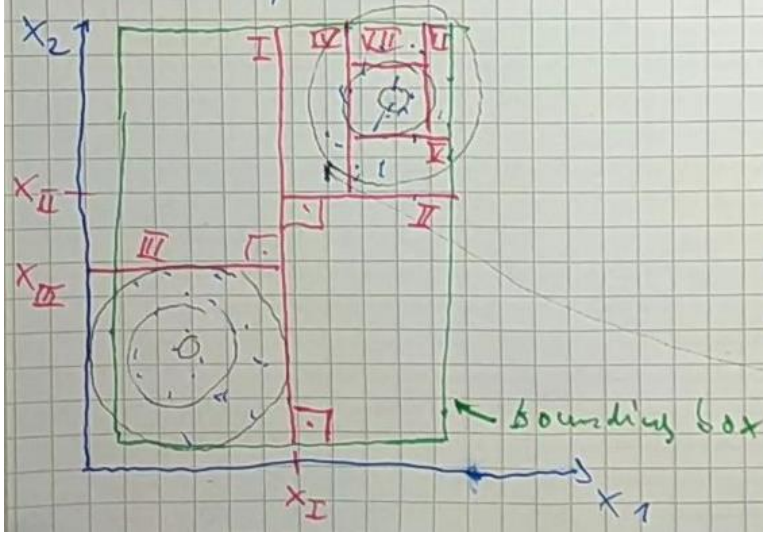
(1) Naive Bayes Classifier: $\hat{y} = \underset{k}{\operatorname{argmax}}\ p(Y=k \mid X) = \underset{k}{\operatorname{argmax}}\ p(X \mid Y=k) \cdot p(Y=k)$

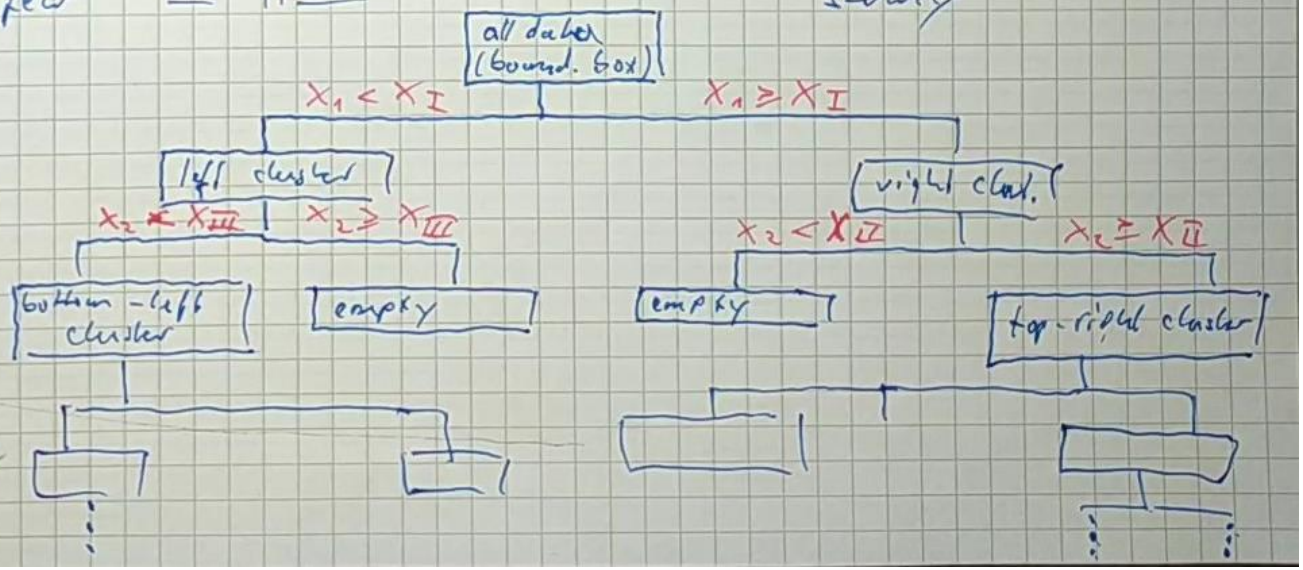
$$\hat{y} = \underset{k}{\operatorname{argmax}} \left(\prod_{j=1}^{D} \hat{p}_{jk}(X) \right) p(Y=k)$$

$\hat{p}_{jk}(X)$: one 1-D histogram per feature per class ($D \cdot C$ histograms in total)
or one 1-D Gaussian — " —

- often, the product decomposition $p(X \mid Y=k) = \prod_j \hat{p}_{jk}(X)$ is a bad approximation
  → transform features into new features z where the assumption is approx. true
  - linear: $z_i = x_i \cdot W$ (e.g. principal component analysis, independent comp. analysis } later)
  - non-linear: $z_i = f_\theta(x_i)$ (f_θ a neural network ⇒ next semester)

(2) adaptive histogram: many bins where $p(x)$ changes quickly
⇒ density tree few — " — slowly





Algorithm (generic form: works for density estimation, classification, regression, ...)

(0) create root node of the tree and put all data in root + store bounding box
⇒ put root on a stack

(1) while stack is not empty:

(a) take top node from stack

(b) if termination criterion is ~~not~~ reached:

(I) create a leaf node which stores the desired response ($\hat{p}_m$ for density tree ← leaf node index)

(c) otherwise

(I) find best possible split of the present node, create two children, put the data in the respective child + store new bounding boxes

(II) put children on the stack

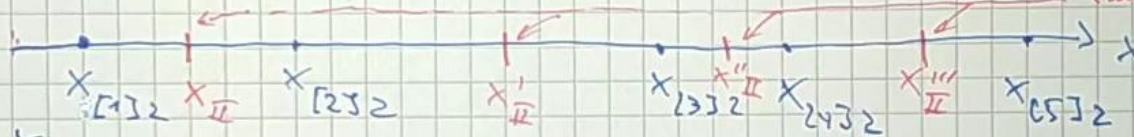
---

termination criteria:

- generic: - stop when a predefined maximum tree depth is reached (hyperparameter)
  - stop —"— minimum # of instances in node (—"—)
  - stop when splitting would give an undesirable shape of the bounding box (very narrow along one dimension)
- application-specific: - density estimation: statistical test that the distribution in the node is close to uniform
  - classification ("decision tree"): all instances in the node have same label ("pure node")

- **split criteria**: - usually, one restricts splitting to axis-aligned split planes
  e.g. at 90° to a single feature axis ⇒ decide for left or right child by a 1-dimensional threshold, e.g. $\rightarrow x_2 < x_{II}$, $x_2 \geq x_{II}$
  (single feature; left child; right child)

  - oblique splits: comp. new feature as a linear combination $\tilde{x}_i = x_i \cdot w^T$ and define threshold on $\tilde{x}_i$ ⇒ sometimes more accurate, more expensive
  - restrict possible thresholds: mid point between neighboring instances



$$x_{II} = \frac{x_{[1]2} + x_{[2]2}}{2} \quad \text{etc.}$$

  when node to be split contains N_m instance, D features: $(N_m - 1) \cdot D$ candidate splits
  ⇒ exhaustive search over all candidate splits to find best one
  or: restrict the search to random subset of size $\sqrt{D}$ of the features
  [in every node, the random subset must be different]

- **score function for best split**: ≙ maximally reduces error ⇒ compare error estimate of current node err_m with children $err_e + err_r$ for each candidate
  ⇒ choose split of maximum improvement.
  (bounding-box volume)

e.g. density tree: leave-one-out error: $err = \int (p^*(x) - \hat{p}^{(x)})^2 dx = \text{const.} - \frac{2}{N} \sum_i \hat{p}_{-i}(x_i) + \sum_m \hat{p}_m^2 \cdot V_m$

⇒ contribution of node m: $\boxed{err_m = -\frac{2}{N} \sum_{i: x_i \in \text{bin } m} \hat{p}_{-i}(x_i) + \hat{p}_m^2 V_m = -2 \frac{N_m (N_m - 1)}{N (N-1) V_m} + \frac{N_m^2}{N^2} \cdot \frac{1}{V_m}}$

split algorithm: for each candidate split:

compute N_l, N_r : number of instances going to left and right child

V_l, V_r : volume of bounding boxes of left and right child

$err_l + err_r = err_{new}$

execute split where err_{new} is smallest

- update bounding box: we know bb of parent x_m^- (lower left corner)

x_m^+ (upper right corner)

if we split along feature X_j at threshold $\tilde{x}$

new left bounding box $x_l^- = x_m^-$, $x_{l,j'}^+ = \begin{cases} x_{m,j'}^+ & \text{if } j' \neq j \\ \tilde{x} & \text{if } j' = j \end{cases}$

right bounding box $x_{r,j'}^- = \begin{cases} x_{m,j'}^- & \text{if } j' \neq j \\ \tilde{x} & \text{if } j' = j \end{cases}$ $\quad x_r^+ = x_m^+$

- volume of bounding box: $V_m = \prod_{j=1}^{D} (x_{m,j}^+ - x_{m,j}^-)$, likewise for left/right

---

prediction: • for new test instance x, find the leaf node containing x by traversing the tree, e.g. go left if $x_j < \tilde{x}$, right otherwise, with

j : index of split feature in present node, $\tilde{x}$: threshold in present node

• return the response stored in the leaf

=> density estimation: $\hat{p}(x) = \sum_{m \in \text{all leaves}} \hat{p}_m \underbrace{\mathbb{I}[x \in \text{leaf}_m]}_{\text{evaluate by tree traversal}}$ $\qquad \hat{p}_m = \frac{N_m}{N \cdot V_m}$

Decision trees: apply recursive subdivision principle to learn posterior $p(Y=k|x)$

[⇒ discriminative, non-parametric model]

advantage over nearest-neighbor: – tree search is more efficient than nearest neighbor search ($O(\text{depth}) < O(N)$)

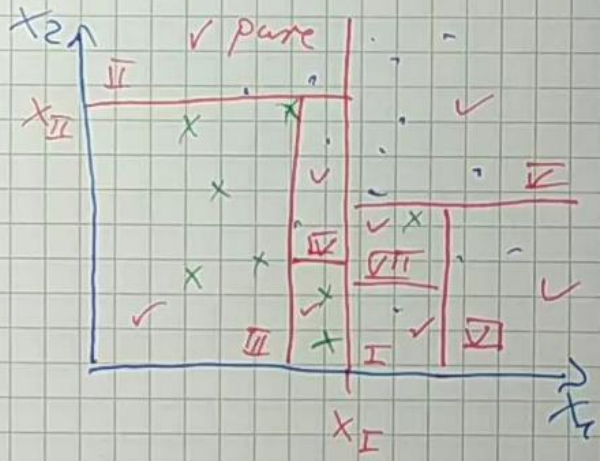
two modifications w.r.t. density tree:

- response of leaf nodes ⇒ label of majority class in the leaf ⇒ decision rule is

$$\hat{Y} = \sum_m \hat{Y}_m \, \mathbb{1}[x \in \text{leaf}_m]$$

- criterion for optimal splits: prefer splits that separate classes well

How to measure if the node is (close to) pure, or contains a mix of all labels?



C4.5 algorithm: entropy

$$H_m = -\sum_{k=1}^{C} \hat{p}_{m,k} \log \hat{p}_{m,k} \qquad \hat{p}_{m,k} = \frac{N_{m,k}}{N_m}$$

[if node is pure ⇒ contains only label k'

$N_{m,k'} = N_m \Rightarrow \hat{p}_{m,k'} = 1$, $\forall k \neq k': \hat{p}_{m,k} = 0$

$\Rightarrow H_m = 0$

if node contains all classes equally: $\hat{p}_{m,k} = \frac{1}{C}$ for all $k \Rightarrow H_m = \log C$ (maximum entropy)]

CART alg: Gini impurity: $G_m = 1 - \sum_{k=1}^{C} \hat{p}_{m,k}^2$ $\quad [0 \leq G_m \leq 1 - \frac{1}{C}]$

best split minimizes $N_l \cdot H_l + N_r \cdot H_r$ or $N_l \cdot G_l + N_r \cdot G_r$ for resulting children l and r

problem: density trees and decision trees tend to overfit (use greedy best split criterion, which depends very much on particular TS)

two common solutions:

- pruning: for each subtree, compute a "overfitting score", replace subtree with single node when score above some threshold (e.g. use validation dataset to compute scores)
- forest: train several trees (which are all different) and return the average (density forest) or majority label (decision forest)
  - two tricks to make the trees different:
    - in each node consider a different random subset (of size $\sqrt{D}$) of features when searching for best split
    - train each tree on a different random subset of the TS, e.g. "bootstrapping": create new TS' by uniform sampling with replacement

$TS' = \{\}$ initially empty

for $i = 1, \dots, N$:

$\quad i = \text{uniform}(1, N)$ random index

$\quad TS' = TS' + \{x_i\}$

⇒ chance that a instance x_i is never chosen: $\frac{1}{e} \approx \frac{1}{3}$

ensemble does not overfit much (why is not yet clear theoretically)

⇒ empirical evaluation: homework