12-1 Useful "non-parametric" learning methods

Non-parametric really means that we don't pick a fixed-size parameterization of the hypothesis class in advance. We'll look at two really useful, easy-to-implement, fast-to-run methods for supervised learning.

Decision trees

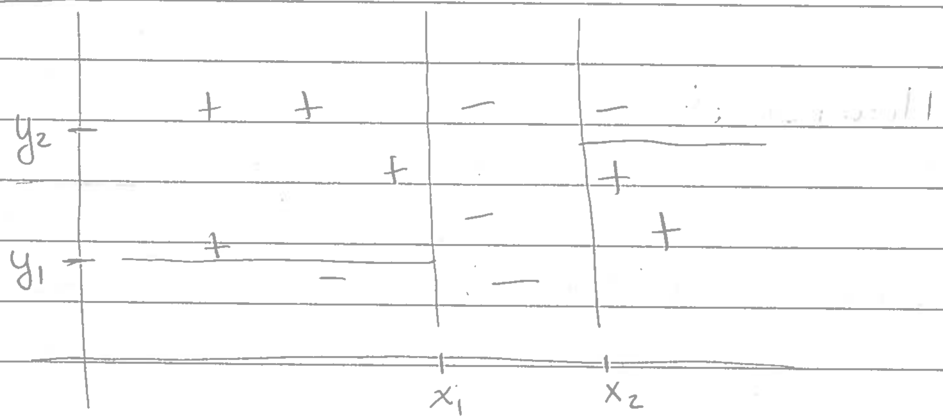
Idea: recursively partition the input space using axis-aligned splits, and fit a simple (usually constant) model in each partition.

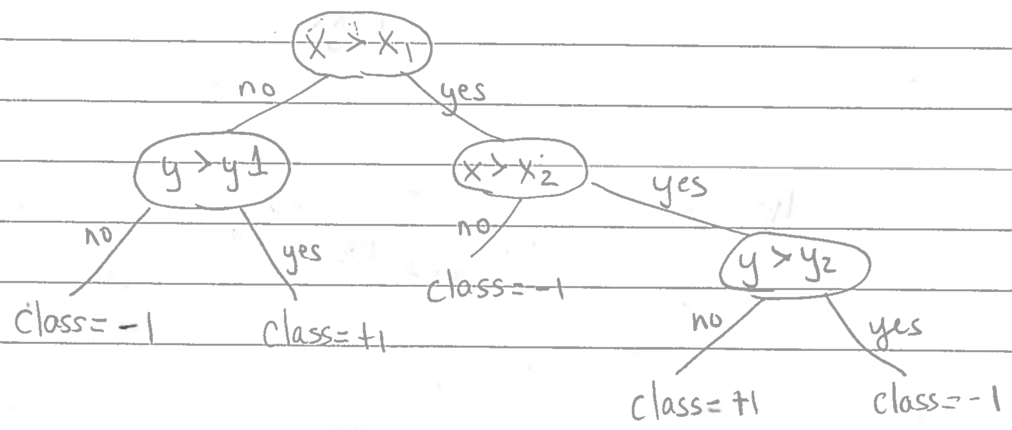
Pros: easy to train, understandable by humans

Cons: lower accuracy, not good for very large input spaces (e.g. images)

CART/ID3: invented in parallel in AI & stats communities in the 80's.

Classification





12-2

We'd like to find a small tree (our regularizer) that predicts training data well. Can always be perfect! but may not want to be.

Tree can be described by:

- set of regions $R_1 \dots R_M$
- output values $O_1, \dots O_M$

Hypothesis: $h(x) = O_m$ if $x \in R_m$

Error in region: $E_m = \left|\{i \mid x^{(i)} \in R_m \text{ and } y^{(i)} \neq O_m\}\right|$

NP hard! We'll do greedy approx.

Objective: find R, O to minimize

$$cM + \sum_{m=1}^{M} E_m$$

Multiclass is easy

Define empirical probability of an item from region R_m being in class K as

$$\hat{p}_{mk} = \frac{\left|\{i \mid x^{(i)} \in R_m \text{ and } y^{(i)} = k\}\right|}{N_m}$$

$$N_m = \left|\{i \mid x^{(i)} \in R_m\}\right|$$

BuildTree (D, K): D = data, K = min leaf size

if $|D| < K$:

return Leaf$\left(\arg\max_k \left|\{i \mid x^{(i)} \in D, y^{(i)} = k\}\right|\right)$

else:

find dimension v and split-point s that yields D_L and D_R:

$$D_L = \{(x^{(i)}, y^{(i)}) \in D \mid x_v^{(i)} < s\}$$

$$D_R = \{(x^{(i)}, y^{(i)}) \in D \mid x_v^{(i)} \geq s\}$$

that minimizes

$$|D_L| \cdot \text{impurity}(D_L) + |D_R| \cdot \text{impurity}(D_R)$$

return Node(v, s, BuildTree(D_L, k), BuildTree(D_R, k))

(12-3) Measures of impurity

- Entropy $\quad Q(D) = \overset{\text{entropy}}{H}\left(\{y^{(i)} \mid (x^{(i)}, y^{(i)}) \in D\}\right)$

$$= -\sum_K \hat{P}_{DK} \log \hat{P}_{DK}$$

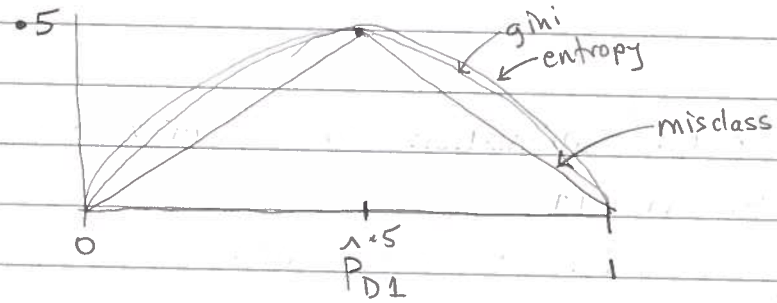
- Gini index

$$\sum_K \hat{P}_{DK}(1-\hat{P}_{DK})$$

- Misclassification error $\quad E_m / N_m = 1 - \hat{P}_{D\,\mathrm{maj}_D K}$

People usually use entropy. Not a big difference.

two class

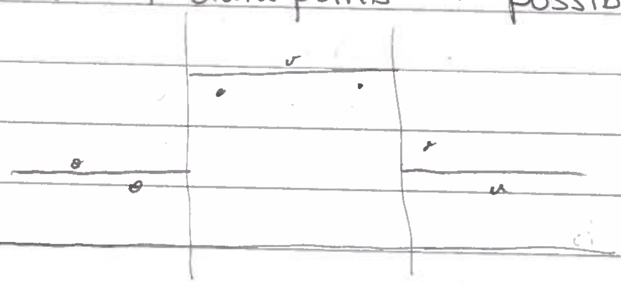


Generally

- For best results, grow a very big tree (small k) and then prune it.
- Easy to incorporate different loss functions (greedily minimize expected loss at each stage)
- Discrete features easy
- Each iteration takes time $O(d \cdot \underset{\text{consider split}}{n} \cdot \underset{\text{compute } Q}{n})$

  n = number of data points = # possible splits

Regression



h = Piecewise constant function.

12-5

Random forest last step

Combine votes of all trees to get resulting hypothesis.

Works surprisingly well. Always try as baseline!

Nearest neighbor

Lazy! Don't do any work at training time — just remember your data

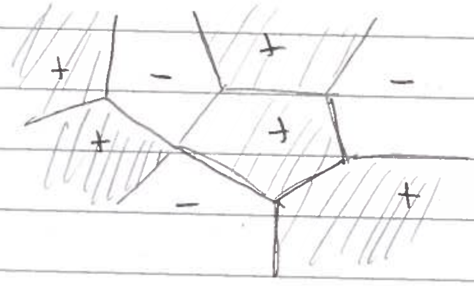
Need distance metric $d(x, x')$

- $d(x,x) = 0$
- $d(x,x') \geq 0$
- $d(x,x') = d(x',x)$
- $d(x,x'') \leq d(x,x') + d(x',x'')$

Hypothesis:

$$h(x) = y^{(i)} \quad \text{where } i = \arg\min_i d(x, x^{(i)})$$

that is, the output of the closest training point.



Voronoi partition

Variations

- Find K nearest neighbors; output average (regression) or majority (classification)
- Fit locally linear regression models

Use good data structures (e.g. ball trees) to find neighbors efficiently.

Works poorly in high dimensions, often.

12-4

Prediction at leaf

$$O_m = \underset{\{i \mid x^{(i)} \in R_m\}}{\text{average}} \; y^{(i)}$$

Error at leaf

$$E_m = \sum_{\{i \mid x^{(i)} \in R_m\}} (y^{(i)} - O_m)^2$$

Pick j, s split that minimizes

$$E_{D_{left}} + E_{D_{right}}$$

Random forests

Why do trees often have high prediction error? Because they have high estimation error (variance).

→ easy to overfit

→ highly sensitive to data points

Idea: reduce variance (of any learning algorithm) by bagging (bootstrap aggregation):

- randomly sample B subsets of size n, with replacement, from D
- train a new hypoth on each one ⇒ h_i
- return

$$h(x) = \frac{1}{B} \sum_i h_i(x) \qquad \text{for regression}$$

$$h(x) = \underset{i}{\text{majority}} \left(h_i(x) \right) \qquad \text{for classification}$$

Random forest

for $b = 1 .. B$:

D_b = bootstrap sample of size n from D

T_b = decision tree but at each iteration

- select m dimensions at random
- pick best split among them