Approx K-NN: No backtrack

Classification and Regression Tree
CART

Goal: build a tree that's
(1) as small as possible, and
(2) only has 1 label per leaf node ("pure" leaves)

NP-Hard.

Idea: split recursively to minimize "impurity" at each node of tree.

Heuristic that measures how "diverse" the labels are in the dataset.
Gini Impurity (Classification)

Empirical distribution of label $y$

\[ P_k = \frac{\left| \{ (x,y) \mid y = k \} \right|}{|\mathcal{D}|} = \frac{\# \text{ of examples w/ label } k}{\# \text{ of examples}}. \]

\[ \text{Gini} \implies G(S) = \sum_k P_k (1 - P_k) \]

\# of ways to draw 2 examples with unequal labels

\[ \sum_k \sum_{\ell \neq k} \left| \left\{ (x,y) \mid y = k \right\} \cdot \left\{ (x,y) \mid y = \ell \right\} \right| \]

\[ = \sum_k \left( \# \text{ with label } k \right) \cdot \left( \frac{n - \left( \# \text{ with label } k \right)}{15} \right) \]

\[ = \sum_k \left( \# \text{ with label } k \right) \cdot \left( n - \left( n \cdot P_k \right) \right) = n^2 \sum_k P_k (1 - P_k) \]

\[ G^T(S) = \frac{|S_L|}{|S|} G^T(S_L) + \frac{|S_R|}{|S|} G^T(S_R) \]
Entropy.

$P_1 = \frac{1}{2} \quad P_2 = \frac{1}{4} \quad P_3 = P_4 = P_5 = P_6 = \frac{1}{16}$

\[ H(S) = \sum_k - P_k \log_2 P_k \]

$H(S)$ is a compact and monotonic function of $D_{KL}(p \parallel q) = \sum_k P_k \log \left( \frac{P_k}{q_k} \right)$

\[ H^T(S) = \frac{|S_L|}{|S_L|} \cdot H^T(S_L) + \frac{|S_R|}{|S_L|} \cdot H^T(S_R) \]

\[ D_{KL}(p \parallel q) = \sum_k P_k \log \left( \frac{P_k}{q_k} \right) \]

\[ D_{KL}(p \parallel \text{uniform}) = \sum_k P_k \log \left( \frac{V_c}{P_k} \right) = \sum_k P_k \log \frac{1}{P_k} - \sum_k P_k \log \frac{1}{V_c} \]
Ball Trees

\[ D_a \quad D_b \]

\[ X_{test} \quad X_i \quad C \]
Impurity

- Gini impurity ≈ how often will random labels disagree
- Entropy ≈ information
- Least squares

ID3

Input: Dataset D
Search all features:
  search all splits for feature:
  evaluate impurity (entropy)
pick feature/split that minimizes impurity
construct node
recursively call ID3 on subsets

How to split?
- categorical features → 1 child per category
- real-valued features → threshold

Base cases:
1. ∃ x s.t. ∀ (x,y) ∈ D, y = \hat{y} = \bar{x}
2. ∃ y s.t. ∀ (x,y) ∈ D, y = \hat{y} = \text{leaf } \hat{y}
3. |D| = 0, predict majority/average of "parent" dataset
Why not stop when impurity doesn't decrease.

\[ P_x = \frac{1}{2} \quad P_0 = \frac{1}{2} \]

Decision tree inference time is proportional to depth.

Inference very fast!

Overfitting:

\[ \text{bias}^2 + \text{variance} + \text{noise} \]
Ensembling: average the prediction of some models.

- draw m independent datasets \( D_1, \ldots, D_m \)
- for each dataset:
  - run \( \text{ID} \rightarrow \) hypothesis \( h_i \)
- output \( \hat{h}(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x) \)

How does this effect: bias² variance noise?

\[
\text{Var} \left( \frac{1}{m} \sum_{i=1}^{m} h_i(x) \right) = \frac{1}{m^2} \text{Var} \left( \sum_{i=1}^{m} h_i(x) \right)
= \frac{1}{m^2} \sum_{i=1}^{m} \text{Var} (h_i(x))
= \frac{1}{m^2} \cdot m \cdot \text{Var} (h_i(x))
= \frac{1}{m} \text{Var} (h_i(x))
\]
Bootstrap Aggregating

Instead of drawing from source dist, we draw with replacement from D.

given D, sample D anew to get dataset D'.

draw m datasets from D each of size n

for each, E train a decision tree (ID3)

average: $h(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x)$

still reduces variance!

Idea: individual examples are still i.i.d. distributed according to $P$

- even though they're not independent, they're "independent enough" to reduce variance.

Random Forest $\Rightarrow$ Full algorithm Bagging Trees
General Gradient Boosted Regression Trees

Input: loss $l$, step size $\alpha$, CART alg.

Init: $H_0 = 0$, $H_0(x) = 0$

Loop from $t=1:T$

$r_i = l'(H_{t-1}(x); y_i)$ for all $i \in \{1, \ldots, n\}$

$h_t = \text{CART}(\{ (x_i, r_i) | i \in \{1, \ldots, n\} \})$

if $\sum_{i=1}^{n} r_i h(x_i) < 0$:

update ensemble by adding $h$

$H_T(x) = H_{T-1}(x) + \alpha h(x)$

else:

halt and return $H_{T-1}$.

end loop

return $H_T$.

could replace with $l(H_{T-1} + \alpha h) < l(H_{T-1})$
Random Forest

Bagging + CART + Subsampling features

//
// each node searches
// all features & splits
// only search k features at random

- increases "diversity" of trees

Hyper params: k = # of features we search
m = # of datasets/trees = set as large as possible

1. Sample m "bagged" datasets from D
   call D_1, D_2, ..., D_m

2. For each bagged dataset D_i
   run a variant of CART/ID3
   at each split choose k features at random and only consider splitting those
   outputting hypothesis h_i

3. Output random forest classifier/regressor

\[ \hat{h}(x) = \frac{1}{m} \sum_{i=1}^{k} h_i(x). \]
Two Variants of RF:

- build tree using one dataset
  and label the leaves with another
  independent dataset
  (prove consistency)

- build smaller trees (not to full depth)
  (pruning)
Boosting vs. Bagging

- sequentially call our tree-building algo, at each step build a tree that improves the example

Goal: create an ensemble classifier/regressor

\[
H_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x) = H_{T-1}(x) + \alpha_T h_T(x)
\]

ensemble after \( T \) steps

\( \alpha \) = scalar weights

\( h_t(x) \) = hypothesis formed at step \( t \)

Goal: minimize loss function

\[
l(H) = \frac{1}{n} \sum_{i=1}^{n} l(H(x_i), y_i)
\]

the hypothesis I pick at step \( T \) is

\[
h_T = \arg \min_{h \in H} \frac{1}{n} \sum_{i=1}^{n} l(H(x_i) + \alpha h(x_i), y_i)
\]

for simplicity, assume $\alpha$ fixed

\[
= \arg \min_{h \in H} \frac{1}{n} \sum_{i=1}^{n} \left[ l(H(x_i), y_i) + \alpha \langle h(x_i), \nabla_l (H(x_i), y_i) \rangle \right]
\]

\[
= \arg \min_{h \in H} \frac{1}{n} \sum_{i=1}^{n} \langle h(x_i), \nabla_l (H(x_i), y_i) \rangle
\]
\[
\arg \min_{\text{h} \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} h(x_i) \ell'(h(x_i), y_i) \\
= \arg \min_{\text{h} \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} h(x_i) \ell(h(x_i), y_i) \\
= \arg \min_{\text{h} \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i) \\
\text{let } r_i = \frac{\ell(h(x_i), y_i)}{\ell(h(x_i), y_i)} = \frac{\ell'(h(x_i), y_i)}{\ell(h(x_i), y_i)} \\
= \arg \min_{\text{h} \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} h(x_i) \cdot r_i' \\
\text{if this is negative, then } \ell(h_{t+1}) \geq \ell(h_t)
\]

General boosting (a.k.a. "AnyBoost")

1. Assume that hypothesis h has constant 
\[
\sum_{i} (h(x_i))^2. \text{ e.g. if } h(x_i) \in \{+1, -1\}, \\
= \arg \min_{\text{h} \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (h(x_i) + r_i)^2 - \frac{1}{2} h(x_i)^2 - \frac{1}{2} r_i^2
\]

\[
a b = \frac{1}{2} (a + b)^2 - \frac{1}{2} a^2 - \frac{1}{2} b^2 \\
t_i = -r_i = -\ell'(h(x_i), y_i)
\]

\[
= \arg \min_{\text{h} \in \mathcal{H}} \frac{1}{2n} \sum_{i=1}^{n} (h(x_i) - (-r_i))^2 \\
\approx \text{CART}(h_{(x_1, t_1)}, (x_2, t_2), ..., (x_n, t_n))
\]
AdaBoost

Assumption: binary classification \((y_i \in \{+1, -1\})\)

\[\Rightarrow\] weak learner also binary \((h_i(x) \in \{+1, -1\})\)

\[\Rightarrow\] "step size" \(\alpha\) set by line search

\[\Rightarrow\] set optimally to minimize loss

loss Exponential loss

\[l(H) = \sum_{i=1}^{n} \exp(-y_i H(x_i))\]

Single ex loss:

\[l(H(x); y) = \exp(-y H(x))\]

\[l'(H(x); y) = -y \exp(-y H(x))\]

Boosting goal:

\[\arg\min_{H(x)} \sum_{i=1}^{n} l'(H(x_i); y_i) - h(x_i)\]

\[= \arg\min_{h} \sum_{i=1}^{n} y_i \exp(-y H(x_i)) - h(x_i)\]

let

\[w_i = \frac{1}{Z} \exp(-y_i H(x_i)),\]

\[Z = \sum_{i=1}^{n} \exp(-y H(x_i))\]

\[= \arg\min_{H(x)} \sum_{i=1}^{n} w_i y_i H(x_i)\]

\[= \arg\min_{h} \sum_{i=1}^{n} w_i y_i H(x_i)\]

\[= \log \sum_{i=1}^{n} w_i y_i H(x_i)\]

\[= \log \exp(-y_i H(x_i))\]

\[= 1 - \sum_{i=1}^{n} w_i\]

\[\text{if } y_i \neq H(x_i)\]

\[\text{if } y_i = H(x_i)\]
arg\ \min_k\ \arg\ \min_h\ \sum_{i \in y \neq h(x_i)} w_i = \arg\ \min_h\ \sum_{i \in y \neq h(x_i)} w_i \exp(-y_i h(x_i)) = \arg\ \min_h\ \sum_{i \in y \neq h(x_i)} w_i \exp(-y_i h(x_i))

Choosing $\alpha$:

$\alpha = \arg\ \min \sum_{i = 1}^n \exp(-y_i (H(x_i) + \alpha h(x_i)))$

$= \arg\ \min \sum_{i = 1}^n \exp(-y_i H(x_i)) \exp(-y_i h(x_i) \alpha)$

$= \arg\ \min \sum_{i = 1}^n w_i \exp(-y_i h(x_i) \alpha)$

$= \arg\ \min \sum_{i \in y \neq h(x_i)} w_i \exp(-y_i h(x_i) \alpha) + \sum_{i \in y = h(x_i)} w_i \exp(-y_i h(x_i) \alpha)$

$= \arg\ \min \sum_{i \in y \neq h(x_i)} w_i \exp(-y_i) + \sum_{i \in y = h(x_i)} w_i \exp(y_i)$

Let $\epsilon = \sum_{i \in y \neq h(x_i)} w_i$ = weighted classification error of $h$

$= \arg\ \min (1-\epsilon) \exp(-\alpha) + \epsilon \exp(\alpha)$

$= \alpha = \frac{1}{2} \log\frac{1-\epsilon}{\epsilon}$
Finishing up AdaBoost:

- Incrementally update $U_i$

$$w_i' = \frac{1}{Z'} \exp \{ -y_i (H_b(x_i) + \alpha h(x_i)) \}$$

$$Z' = \sum_{i=1}^{n} \exp \{ -y_i (H(x_i) + \alpha h(x_i)) \}$$

$$= \sum_{i=1}^{n} Z \cdot w_i \cdot \exp \{ -\alpha y_i h(x_i) \}$$

$$= Z \cdot \left( \sum_{i: y_i = h_b(x_i)} w_i \exp(\alpha) + \sum_{i: y_i \neq h_b(x_i)} w_i \exp(-\alpha) \right)$$

$$= Z \cdot \left( \varepsilon \exp(\alpha) + (1-\varepsilon) \exp(-\alpha) \right)$$

Recall: $\alpha = \frac{1}{2} \log \left( \frac{1 - \frac{1}{\sqrt{\varepsilon}}} {\frac{1}{\sqrt{\varepsilon}}} \right)$

$$Z' = Z \cdot \left( \varepsilon \sqrt{\frac{1 - \varepsilon}{\varepsilon}} + (1 - \varepsilon) \cdot \sqrt{\frac{\varepsilon}{1 - \varepsilon}} \right)$$

$$L(H + \alpha H_b) = L(H) \cdot \left( 2 \sqrt{\varepsilon (1-\varepsilon)} \right)$$
if we halt whenever the classifier $h$
has weighted test error $E$ too
close to $\frac{1}{2}$, i.e. suppose require
that $E \leq \frac{1}{2} - \delta$

then: $2n \sqrt{E(1 - E)} \leq 2n \sqrt{\left(\frac{1}{2} - \delta\right)(\frac{1}{2} + \delta)}$

$\leq 2 \sqrt{\frac{1}{4} - \delta^2}$

$= \sqrt{1 - 4\delta^2}$

$L(H + ah) \leq L(H) \cdot \sqrt{1 - 4\delta^2}$

after we add $k$ trees to the ensemble.

$L(H) \leq L(O) \cdot (1 - 4\delta^2)^{k/2} \leq n \cdot (1 - 4\delta^2)^{k/2}.$

if $\delta$ constant, need $O(\log n)$ trees
to achieve small loss.
\[ L(w) = f_1(w) + f_2(w) = (w-1)^2 + (w+1)^2 = 2w^2 + 2 \]

\[ L'(w) = 4w \quad f_1'(w) = 2(w-1) \]

\[ L'(w)f_1'(w) = 8w(w-1) \neq 0 \]

Why SGD:

\[ x_i, y_i \text{ example selected at time } t \]

\[ L(w_{t+1}) = L(w_t) - \alpha \nabla L(w_t)^T \nabla L(h(x_i; w_t); y_i) + \delta(w) \]

\[ E[L(w_{t+1})] = L(w_t) - E[\alpha \nabla L(w_t)^T \nabla L(h(x_i; w_t); y_i)] + \delta(w) \]

\[ = L(w_t) - \alpha \nabla L(w_t)^T E[\nabla L(h(x; w_t); y_i)] + \delta(w) \]

\[ E[\nabla L(h(x; w_t); y_i)] = \sum_{i=1}^{n} \frac{1}{n} \nabla L(h(x; w_t); y_i) \]

\[ = \nabla_w \left( \frac{1}{n} \sum_{i=1}^{n} L(h(x_i; w_t); y_i) \right) \]

\[ = \nabla_w \left( \frac{1}{n} \sum_{i=1}^{n} L(w_t) \right) \]

\[ = L(w_t) - \alpha \nabla L(w_t)^T \nabla L(w_t) + C \alpha^2 \]

\[ \frac{1}{n} \left\| \nabla L(w_t) \right\|^2 \geq 0 \]

SGD vs GD:

* SGD is faster per iteration
* SGD generalizes better!
* SGD helps avoid local minima
Neural Networks a.k.a. "Multilayer Perceptron"

\[ a_i = \sigma (w_{i1} x_1 + w_{i2} x_2 + w_{i3} x_3 + \cdots + w_{in} x_n + b_i) \]

\[ h = \sigma (w_{1a} a_1 + w_{2a} a_2 + \cdots + w_{na} a_n + b_h) \]

\[ a \in \mathbb{R}^3, \quad a = \sigma (Wx + b) \]

\[ a_1 = \sigma (w_{11} x_1 + w_{12} x_2 + w_{13} x_3 + \cdots + w_{1n} x_n + b_1) \]
\[ a_2 = \sigma (\cdots) \]
\[ a_3 = \sigma (\cdots) \]
\[ h = \sigma (w_{1a} a_1 + w_{2a} a_2 + \cdots + w_{na} a_n + b_h) \]

\[ h(x) = \frac{1}{\beta} (u^T \sigma (Wx + b) + c) \]

\[ h(x) = u^T \phi(x) \kappa, \text{ where } \phi(x) = \sigma (Wx + b) \]
What is $\sigma$?

- ReLU: Rectified linear unit.

$\text{ReLU}(x) = \sigma(x) = \max(0, x)$

ReLU networks (nets where all nonlinearities are ReLUs) are piecewise linear.

All continuous piecewise linear functions can be expressed as a ReLU network.

Why does this hold? $\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$

$f''(x) = c_0 + a_1 \text{ReLU}'(x-b_1) + a_2 \text{ReLU}'(x-b_2) \ldots$

$f(x) = c_0 x + c_1 + a_1 \text{ReLU}(x-b_1) + a_2 \text{ReLU}(x-b_2) \ldots$
Let \( f(x) = \text{sign}(x) = \begin{cases} \frac{x}{|x|} & \text{if } x \leq 0 \\ -\frac{x}{|x|} & \text{if } x > 0 \end{cases} \)

\[ f(x) = -x + 2 \text{ReLU}(x) \]

Let \( f(x,y) = \max(x,y) = \begin{cases} x & \text{if } x \geq y \\ y & \text{if } x < y \end{cases} \)

\[ f(x,y) = \text{ReLU}(y-x) + x \]

Any function \( \Phi \) can be approximated as an ANN:

- Linear function + ReLU
- "Universality" to approx any function!

Choose the "architecture" + "weights/parameters" \( \omega_0, \omega_1, \ldots \)
Common Nonlinearities

- ReLU
- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$
- "Smoothed" ReLU
- "Saturating" ReLU

$\sigma(x) = \max(0, \min(x, L))$
**How to learn a DNN:**

\[ \min_w \sum_{i=1}^{n} l(h(x_i; w); y_i) \]

- **STOCHASTIC Gradient descent!**)
- Using a subsample of \( D \) to compute gradients at each step
- **+ automatic differentiation**
- \( \neq \) not numerical differentiation
  \[ f'(x) \neq \frac{f(x+0.001) - f(x)}{0.01} \]
- \( \neq \) not symbolic diff
- About the same time as computing \( f \) to compute \( \Delta f \)
- \( \rightarrow \) **BACKPROPAGATION**

**GD:** \( w \leftarrow w - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla l(h(x_i; w); y_i) \)

**SGD:** pick \( i \) from \( 1 \ldots n \) at random
\( w \leftarrow w - \alpha \nabla l(h(x_i; w); y_i) \)

\[ L(w - \alpha \nabla L(w)) = L(w) - \alpha L(w)^T \nabla L(w) + o(\alpha^2) \]

next \( w \)
\[ \| \nabla L(w) \|^2 \geq 0 \]

\[ L(w - \alpha \nabla l(h(x; w), y_i)) = L(w) - \alpha \nabla l(h(x; w), y_i) + o(\alpha) \]
Convolutional Neural Network

Image is what? Multidim array \((3 \times W \times H)\)

e.g. a \(16 \times 16\) coloring: \((3 \times 16 \times 16)\) array

\[
h(x) = U \cdot \text{RELU} \left( \frac{W x + b}{W \cdot \text{VEC}(x) + b} \right)
\]

Want: something shift-invariant

\[
\text{out} = \text{conv}(\text{in}, \text{filter})
\]

\((d-kh) \times (d-kh)\) \(d \times d\) \(k \times k\)
Conv layers:

\[
\begin{bmatrix}
  \frac{\text{[}} \frac{\text{[}} \frac{\text{[}} \frac{\text{[}} \\
  \text{[}} \frac{\text{[}} \frac{\text{[}} \frac{\text{[}} \\
  \text{[}} \\
  \text{[}}
\end{bmatrix} \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}
\]

\[W\]

\[\text{(out)}_j = \sum_i \text{conv}(\text{(in)}_i, W_{j,i})\]

CNN

Input \[x\] → [Conv] → [Fully] → [Conv] → [Fully] → ...

\[\text{vec} \rightarrow \text{MLP} \rightarrow \text{output} \]

\text{flatten}
DNN and Overfitting

A model is overparameterized when
\[ d \gg n \]
where \( d \) is the number of parameters and \( n \) is the number of training examples.

"DNNs don't overfit" as much as other models when trained using standard methods.

Counter-overfitting methods:

- \( \|w\|_2 \) regularization + \( \lambda \|w\|_2^2 \) "weight decay"

- dropout: While training, for each step of SGD, "remove" each hidden node from the network w.p. \( p > 0 \) (e.g. \( p = 1/2 \))

- batch normalization