lecture 24

- Boosting
- Nearest neighbor classifier

AdaBoost: ensemble method for classification (or regression) (adaptive boosting)
- trains multiple learners on weighted sample pts. (like bagging)
  - different weights for each learner
  - increased weights for misclassified pts.
  - higher weight to more accurate learners

Input: $n \times d$ design matrix X, label vector $\vec{y} \in \mathbb{R}^n$, $y_i = \pm 1$

Main ideas: ("trees")
- train T classifiers, $G_1 \dots G_T$
- Weight of sample pt. x_i in G_t grows according to how $G_1 \dots G_{t-1}$ misclassified it
  - ↳ if x_i misclassified by every learner, its weights grow even more and vice versa
- train G_t to train harder to correctly classify sample pts. w/ larger weights
- **Metalearner:** linear combo of all learners

$$M(\vec{z}) = \sum_{t=1}^{T} \beta_t G_t(z)$$

- $G_t(\vec{z}) = \pm 1$, M continuous
- return $\text{Sign}(M(\vec{z}))$

Although boosting was originally designed for decision trees, it works well for most learning algorithms

recall to weight pts. in trees, we used weighted entropy (proportion of weight in each class)

What β_T and G_T do we use?
- pick loss fn. and find G_T, β_T to minimize $\text{Risk} = \frac{1}{n} \sum_{i=1}^{n} L(M(x_i), y_i)$

exponential loss fn: $L(p, \ell) = e^{-p\ell} = \begin{cases} e^{-p} & \ell = +1 \\ e^{p} & \ell = -1 \end{cases}$

↳ AdaBoost meta learner uses this, $G_i(z)$ uses another loss fn, if any

Important: ℓ, G_t binary but $f = M(x_i)$ continuous

advantage of exponential loss fn: pushes hard against badly misclassified pts.
↳ better than squared error for metalearners
↳ same reason why in NNs we use cross-entropy instead of squared error

$$n \cdot \text{Risk} = \sum_{i=1}^{n} \mathcal{L}(M(x_i), y_i)$$
$$= \sum_{i=1}^{n} e^{-y_i M(x_i)}$$
$$= \sum_{i=1}^{n} e^{-y_i \sum_{t=1}^{T} \beta_T G_T(x_i)}$$
$$= \sum_{i=1}^{n} \prod_{t=1}^{T} e^{-\beta_t y_i G_t(x_i)}$$
$$= \sum_{i=1}^{n} (w_i)^T e^{-\beta_T y_i G_T(x_i)}, \quad w_i^T = \prod_{t=1}^{T-1} e^{-\beta_t y_i G_t(x_i)}$$

Now, split sum in 2 parts depending on correct classification: $y_i = G_T(x_i)$
or incorrect: $y_i \neq G_T(x_i)$

$$\rightarrow n \cdot \text{Risk} = e^{-\beta_T} \sum_{y_i = G_T(x_i)} w_i^T + e^{\beta_T} \sum_{y_i \neq G_T(x_i)} w_i^T$$
$$= e^{-\beta_T} \sum_{i=1}^{n} w_i^T + \left(e^{\beta_t} - e^{-\beta_t}\right) \sum_{y_i \neq G_T(x_i)} w_i^T$$

What G_T minimizes the risk?
- the learner who minimizes w_i^T across the misclassified pts

Now we know what weight to assign to each pt.
- minimize risk ≡ minimize total weight of misclassified pts. for weight fn. w_i^T

$$w_i^{T+1} = w_i^T e^{-\beta_T y_i G_T(x_i)} = \begin{cases} w_i^T e^{-\beta_T} & y_i = G_T(x_i) \\ w_i^T e^{\beta_T} & \text{o/w} \end{cases}$$

(recursive)
↳ benefit of choosing loss fn.

- Not always possible to pick learner that always classifies pts. correctly (like SVMs and non-lin. separable data)
- NP-hard to find optimal linear classifier
  - ↳ in practice G_T won't be true minimizer of training error

Optimal value of β_t:

Set gradient of risk to 0: $0 = -e^{-\beta_T} \sum_{i=1}^{n} w_i^{(T)} + \left(e^{\beta_T} + e^{-\beta_T}\right) \sum_{y_i \neq G_T(x_i)} w_i^T$

divide both sides by 1st term: $0 = -1 + \left(e^{2\beta_T} + 1\right) err_T$, $err_T = \frac{\sum_{y_i \neq G_T(x_i)} w_i^T}{\sum_{i=1}^{n} w_i^T}$

optimal metalearner $\Rightarrow \beta_T = \frac{1}{2} \ln\left(\frac{1 - err_T}{err_T}\right)$

$err_T = 0 \Rightarrow \beta_T = \infty$ (perfect learner gets infinite vote/weight)

$err_T = 1/2 \Rightarrow \beta_T = 0$ (learner w/ 50% acc. gets no vote at all)

- more accurate learners get more votes in metalearner
- acc < 50% $\Rightarrow$ negative vote (learner w/ 40% acc $\equiv$ learner w/ 60% acc w/ vote sign flipped)

AdaBoost Algorithm:

1) initialize weights $w_i = \frac{1}{n}$, $i = 1 \ldots n$
2) for $t = 1 \ldots T$:
   a) compute weighted error rate $err = \frac{\sum_{\text{misclassified}} w_i}{\sum_{\text{all } i} w_i}$
   b) compute weight $\beta_T = \ln\left(\frac{1-err}{err}\right)$
   c) re-weight points: $w_i = w_i \cdot \begin{cases} e^{\beta_t} & \text{misclassified} \\ e^{-\beta_t} & \text{o/w} \end{cases} = w_i \cdot \begin{cases} \sqrt{\frac{1-err}{err}} & \text{misclass.} \\ \sqrt{\frac{err}{1-err}} & \text{o/w} \end{cases}$
3) return metalearner $h(\vec{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t G_t(x_i)\right)$

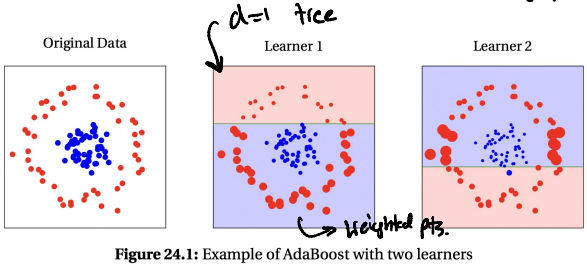

Figure 24.1: Example of AdaBoost with two learners

Why boost decision trees? Why short trees?

- it's fast
  - training many learners and running many learners at classification time
    - short decision trees fast at training and testing
- no h-param search needed
- easy to make tree beat 55% training acc. (or any other threshold) consistently
- easy B/V control
  - boosting can overfit ⟶ make AdaBoost trees short to avoid overfitting
    - as # learners ↑, B ↓
    - variance often decreases at first (since successive trees focus on different features)
      - increases later
- AdaBoost & short trees is a form of subset selection
  - features that don't improve metalearner aren't used at all
    - reduces overfitting & runtime especially if lots of useless features
- linear decision boundaries don't boost well
  - boosting linear classifiers gives another rough linear classifier
    - SVMs not good to use
  - nonlinear methods benefit more from boosting
    - allow boosting to reduce bias faster
  - Sometimes AdaBoost used w/ d=1 trees, but this is linear (not ideal)
    - d=2 much better

<u>Facts about AdaBoost:</u>

- posterior prob. can be approximated: $P(y=1|\vec{x}) \approx \frac{1}{1+e^{-2M(\vec{x})}}$
- if data vulnerable to outliers, use exponential loss
  - corrupted data ⟹ use different loss fn. (better loss fns usually have more complicated weights)
- if every learner beats acc. $\mu = 50\%$, then metalearner acc. will eventually be 100% (proven in HW)

cont. below

Nearest Neighbor Classification

given query pt. q, find k samples closest to q
- use distance metric of choice

- regression can give average label of k pts, classification returns majority of k labels
(neither are what we want)

thm Cover & Hart: as $n \to \infty$, 1-NN (1 nearest neighbor) error rate $< 2B - B^2$, B = Bayes risk
- if only 2 classes, 1-NN error rate is $\leq 2B - 2B^2$

→ lots of data → NN works quite well

requirement of thm: training & test pts. must all be drawn independently from same probability distr.

- applies for any dist metric, not just Euclidean

thm Fix & Hodges: $n \to \infty$, $k \to \infty$, $\frac{k}{n} \to 0$, then k-nn error converges to B

→ k-NN converges to be Bayes optimal