Statistical Learning VI

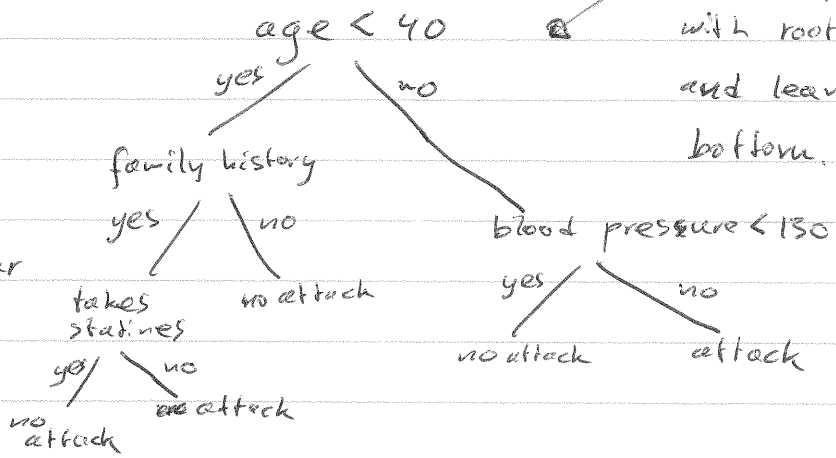
1. Decision Trees
   a) General Setup
   b) Growing
   c) Pruning
2. Bagging + Random Forests
3. Boosting

2. Decision Trees

a) General Set-up

mathematicians see trees upside down with root at the top and leaves at the bottom.

decision tree for classification of heart attack next year



Advantages: * easy to interpret
* nice algorithms

Disadvantages: * not competitive with logistic regression/kernel methods in terms of prediction accuracy
* instability/learning algorithms have high variance
⇒ small changes in data lead to very different tree
⇒ undermines interpretation

(will be fixed later in this lecture)

Fig. 9.2

Estimates class or regression value independently per region R_m with ERM:

$$\hat{c}_m = \underset{c}{\operatorname{argmin}} \sum_{x_i \in R_m} L(y_i, c)$$

regression: $L(y,c) = (y-c)^2$ "squared error" $\Rightarrow$ c_m is avg y_i in R_m

classification: 0/1-loss $\Rightarrow$ c_m is most common class in R_m

or multinomial model with log loss:

(for two classes for simplicity)

* c_m is now probability $P(Y=1 \mid x \in R_m)$

$$\hat{c}_m = \underset{c}{\operatorname{argmin}} \sum_{x_i \in R_m} -\log P(y=y_i \mid x_i \in R_m)$$

$$= \underset{c}{\operatorname{argmin}} -n_1 \log c - n_0 \log(1-c) \qquad (n_k = \#k \text{ in } R_m)$$

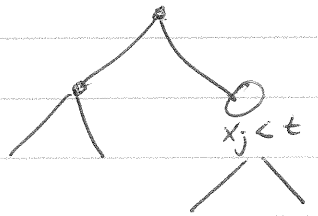
$$= \underset{c}{\operatorname{argmin}} -\hat{p}_{m,1} \log c - \hat{p}_{m,0} \log(1-c)$$

where $\hat{p}_{m,k} = \dfrac{n_k}{n_0 + n_1}$

$\hat{c}_m = \hat{p}_{m,1}$ is ML estimate for Bernoulli model

$-\hat{p}_{m,1} \log \hat{p}_{m,1} - \hat{p}_{m,0} \log \hat{p}_{m,0}$ is "entropy"

b) Growing



How to choose feature x_j + threshold t?

Greedy approach: split region R_m into R_{m1} and R_{m2} that minimize

$$\text{Loss}(R_{m1}) + \text{Loss}(R_{m2})$$

$$\min_c \sum_{x_i \in R_{m1}} L(y_i, c)$$

Regression: always use squared error loss

Classification:

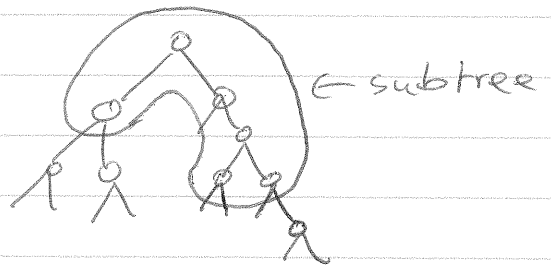
- use entropy or "Gini index" while growing
- 0/1-loss to estimate class in each region

Fig. 9.3: entropy and Gini more sensitive to changes around 0 and 1.

Pruning:

* Bigger tree → more regions → better fit training data
* Too large → overfitting because too little data per region
  too small → underfitting.

* Difficult to know when to stop growing tree
* Solution: grow very big tree, then prune to smaller tree.



choose subtree $T_\alpha \subseteq T$ that minimizes:

$$\sum_{i=1}^{N} L(y_i, T_\alpha(x_i)) + \alpha \cdot |T_\alpha|$$

$T_\alpha(x_i) = \hat{c}_m$ for m s.t. $x_i \in R_m$.

$|T_\alpha|$: nr of leaf nodes in tree

1. Bagging $T = \begin{pmatrix} Y_1 \\ X_1 \end{pmatrix}, \ldots, \begin{pmatrix} Y_N \\ X_N \end{pmatrix}$

Motivation: suppose we have high variance method (e.g. decision trees without pruning). How do we reduce variance?

Bootstrap + Aggregation

high variance = very different estimates when run on new data set. (different e.g. trees)

- simulate many data sets by bootstrap and "average out" the variance.

Bootstrap:

- Z_b is sample of size N, sampled with replacement from T.

$b = 1, \ldots, B$

- Train estimators $\hat{f}_b$ for each Z_b

Aggregation:

for regression: $\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x)$ is average

for classification:

* majority vote among $f_1, \ldots, f_B$
* if classifiers provide ~~weights~~ estimates $\hat{p}_1(Y=1|x), \ldots, \hat{p}_B(Y=1|x)$ then average those.

Bagging reduces variance, but does not help (or very little) if model cannot express ~~even~~ what we need to learn, i.e. has large bias. Figures 8.9, 8.10, 8.12

Bootstrap samples: want diversity of estimators $\hat{f}_b$ (e.g. ~~many~~ diverse set of trees)

Random forests: (optionally: see Ch. 15)

decision trees (without pruning) + bagging + some small extra tricks (e.g. to increase diversity among generated trees)

~~One of the best methods "out of the box", without engineering features or tuning parameters.~~

2. Boosting

"weak learners": high bias, low variance (assume that only slightly better than random guessing...)

(e.g. decision stumps = decision trees with only one binary decision)

How can be turn these into very good learners? (Fig 8.12, right)

$\begin{pmatrix} y_1 \\ x_1 \end{pmatrix}, \begin{pmatrix} y_2 \\ x_2 \end{pmatrix}, \ldots, \begin{pmatrix} y_N \\ x_N \end{pmatrix}$ $\quad y \in \{-1, +1\}$

$w_1 \qquad w_2 \qquad\qquad w_N$

Fig. 10.1, 10.2: to train G_m, increase weights for examples misclassified by a combination of previous classifiers.

Algorithm 10.1

- force it to concentrate on difficult examples)
- But why like AdaBoost?

Forward Stagewise Additive Modeling

Additive models: $f(x) = \sum_{m=1}^{M} \beta_m b_{\gamma_m}(x)$ (β_m: weights; b_{γ_m}: basis functions with parameters γ_m)

Fit by ERM:

$$\min_{(\beta_m, \gamma_m)_{m=1,\ldots,M}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b_{\gamma_m}(x)\right)$$

often computationally difficult.

FSAM:

For $m = 1, \ldots, M$:

$$f_m(x) = f_{m-1}(x) + \beta_m b_{\gamma_m}(x)$$

Find parameters β_m, b_{γ_m} by ERM:

$$\min_{\beta_m, \gamma_m} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x) + \beta_m b_{\gamma_m}(x)\right)$$

Boosting as FSAM:

basis functions: G_m

exponential loss: $L(y, f(x)) = e^{-y \cdot f(x)}$

$$\beta_m, G_m = \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} e^{-y_i (f_{m-1}(x_i) + \beta \cdot G(x_i))}$$

$$= \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^{(m)} \cdot e^{-\beta y_i G(x_i)}$$

equivalent to AdaBoost weights with alpha_m = 2 * beta_m

$$w_i^{(m)} = \frac{e^{-y_i f_{m-1}(x_i)}}{\text{normalisation}}$$

make weights sum to 1

Solve first for G given any fixed β:

$$\min_G \sum_{i: y_i = G(x_i)} w_i^{(m)} e^{-\beta} + \sum_{i: y_i \neq G(x_i)} w_i^{(m)} e^{\beta}$$

$$= \min_G (e^{\beta} - e^{-\beta}) \sum_{i=1}^{N} w_i^{(m)} L_{01}(y_i, G(x_i)) + e^{-\beta} \underbrace{\sum_{i=1}^{N} w_i^{(m)}}_{1}$$

weighted ERM:

$$\min_G \sum_{i=1}^{N} w_i^{(m)} L_{01}(y_i, G(x_i)) \leftarrow \text{err}_m$$

(view 2d in AdaBoost as approximately solving this)

Solve for β:

$$0 = \frac{d}{d\beta} (e^{\beta} - e^{-\beta}) \text{err}_m + e^{-\beta} = (e^{\beta} + e^{-\beta}) \text{err}_m - e^{-\beta}$$

$$(e^{2\beta} + 1) \text{err}_m = 1$$

$$e^{2\beta} \text{err}_m = 1 - \text{err}_m$$

$$e^{2\beta} = \frac{1 - \text{err}_m}{\text{err}_m}$$

$$\beta = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}$$

is what boosting does!

Compare Fig. 10.3 and 10.2

training error to 0

but exponential loss and test set error keep decreasing!

Why?

Why exponential loss?

Estimate in population:

$$f_B(x) = \underset{f(x)}{\operatorname{argmin}} \; \mathbb{E}_{y|x}\left[e^{-y \cdot f(x)}\right]$$

$$= \frac{1}{2} \log \frac{\Pr(y=1|x)}{\Pr(y=-1|x)}$$

so exponential loss tries to estimate log odds of classes.

Slide: surrogate losses

* squared error bad because punishes large positive margin
* exponential loss: really dislikes large negative margin

⇓

AdaBoost gives large weights to observations that are misclassified with large margin

⇓

not robust in settings where this is unavoidable: large noise or mislabeled data

Then might be better to use alternative loss like binomial deviance, although might be computationally more challenging.