Lecture 9: Decision Trees

wbg231

December 2022

1 Decision trees

Decision tree set up

- we are going to focus on binary trees ie each node has max of 2 children
- each node is a subset of data point
- the data splits of each node only use a single feature
- predictions are made at terminal or leaf nodes
- our goal is to find boxed $R_1 \cdots R_j$ that minimize

$$\sum_{j=1}^{j} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

  subject to complexity constraints
- so that is the best disjoint subsets on which to split our input data
- we cant find the best overall possible subsets b/c computationally intractable but can greedly chose starting from the root until we hit a stopping condition
- they we predict a values in a terminal node as

$$\hat{y}_{R_j} = E[y_i | x_i \in R_j]$$

optimal splits

- to keep the number of splits tractable we sort the points based on there values of what ever feature we are splitting on

$$x_{j(1)} \cdots x_{j(n)}$$

  points sorted with regard to the jth feature
- any point between two adjacent features is equivalent so normally just take halfway point

overfitting

- Decision trees tend to over fit, they will on there own put every point in it's own region if just let keep splitting
- there are a lot of ways to control for this, like limiting the total number of nodes, limiting number of terminal nodes, limiting tree depth , require minimum number of data points in a terminal node
- could also prune a tree
    1. so we first fit a really big tree
    2. then we prune the tree greedily cutting until validation accuracy stops improving

good splits

- let $y - \{1 \cdots k\}$ and let m represent region R_m with n_m observations in that node
- the proportions of observations in that node with class k is given by

$$\hat{p}_{m,k} = \frac{1}{n_m} \sum_{i:x_i \in R_m} \mathbb{I}(y_i = k)$$

  and we predict the majority class ie class $K(m) = argmax_k \hat{p}_{mk}$
- three measure of node impurity
    1. misclassification error

$$1 - \hat{p}_{mk}(m)$$

       that is just 1 minus the majority label, we want this to be low
    2. gini index

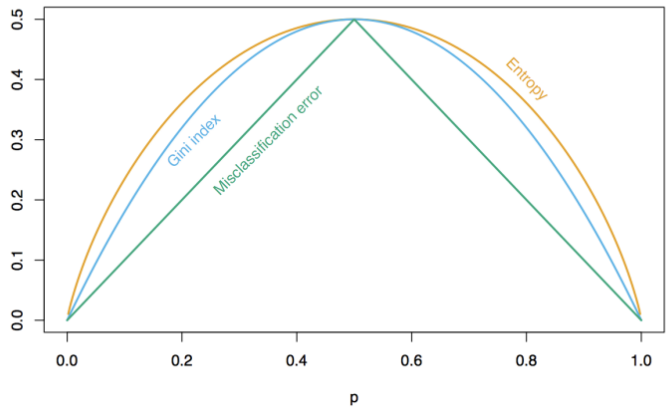
$$\sum_{k=1}^{k} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

       so this is saying given all classes we could predict, we take the product of those classes and there compleat and sum them, this encourages solutions that are close to zero or 1
    3. entropy/information gain

$$-\sum_{k=1}^{k} \hat{p}_{mk} log(\hat{p}_{mk})$$

- which one works best is a case by case question

(p is the relative frequency of class 1)



- overall we want to find the splits that minimize the average weighted node impurity

$$\frac{n_l Q(R_l) + n_r Q(R_r)}{n_l + n_r}$$

  where Q is a node impurity measure, and index l, r correspond to left and right nodes of some split

- small trees at least are really interpretable (this falls off with size though)

trees vs linear models

- trees are really not good at capturing linear Decision boundary's but can capture non linear ones quite well
- Decision trees only model one feature at a time, and split on that so they do not know how to model the relationship between two features really. so something that has a constant relationship $y = x$ they will have to learn meany splits to try to split that space based on some rule corresponding to the value of just one feature
- trees are non-linear, the Decision boundary's they produce will not be linear
- trees are non metric, they do not rely on the geometry of the space they are in
- they are non parametric, as in make no assumptions about how the data is disputed

- additionally they are quite interpretable
- they are bad at capturing linear relationships in data though
- they also have high variance and tend to overfit, that is they are sensitive to small changes in the training data

2 bagging and random forests

- point statistics are when we estimate some parameter about the data with the data
- statistics are random variables so they have probability distributions called a sampling distribution
- the standard deviation of the sampling distribution is called the standard error

variance of mean

- let $\theta(\hat{D})$ be an unbiased estimator with variance σ^2
- standard error of that estimator is $\sqrt{var(\hat{\theta})}$
- consider taking a new estimator that takes the average of iif $\hat{\theta}_1 \cdots \hat{\theta}_n$ where $\hat{\theta}_i = \hat{\theta}(D^i)$ so that is estimator calculated on n idd data sets
- this estimator will be unbiased and constant so we see that

$$var(\frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_i) = \frac{\sigma^2}{n}$$

- this holds for all point estimates on the data. averaging iid predictors will result in lower variance overall
- we do not have independent training sets though

Bootstrap sample

- we can simulate independent draws by bootstrapping ie drawing many sub-samples form our data set with replacement and treating those as datasets to train many models
- this often works well in practice

Ensemble methods

- the key ideas are
  - combine multiple weak models into a single more powerful one
  - average iid estimates reduce the variance with out changing bias
  - we can use a bootstrap to simulate independent samples and average them
  - parallel ensemble methods (like bagging) build models independently
  - sequential ensemble (like boosting) models are built sequentially (at each step we try to improve on what the last model did poorly on)

bagging

- draw B bootstrap samples $D^1 \cdots D^N$ from our original data $\mathcal{D}$
- train b independent predictors on each dataset $\hat{f}_i(D^i)$
- then our bagged prediction function is some combination of our estimators that is
$$\hat{f}_{avg}(x) = combine(\hat{f}_1(x) \cdots \hat{f}_b(x))$$
- general method that is often used with trees
- reduces overfitting, and variance
- but makes the final estimator less interpretable

out of bag error

- each estimator is only trained on roughly $\frac{2}{3}$ of the data , so the remaining $\frac{1}{3}$ of the data can be used as a validation set
- bagging does well **when the base learners are unbiased but have high variance**

correlated predictions

- for $\hat{\theta}_1 \cdots \hat{\theta}_n$ idd where $E[\hat{\theta}] = \theta, var(\hat{\theta}) = \sigma^2$
- what happens if the estimator are not independent that is correlated?
- however $var(\frac{1}{n}\Sigma_{i=1}^n \hat{\theta}_i = cov(\frac{1}{n}\Sigma_{i=1}^n \theta_i, \frac{1}{n}\Sigma_{j=1}^n \theta_j) = \frac{1}{n^2} cov(\Sigma_{i=1}^n \theta_i, \Sigma_{j=1}^n \theta_j) = \Sigma_{i=1}^n \Sigma_{j=1}^n \frac{1}{n^2} cov(\theta_i, \theta_j) = \frac{1}{n}^2 (\Sigma_{i=1}^n var(\theta_i) + \Sigma_{j \geq i} cov(\theta_i, \theta_j) = \frac{\sigma^2}{n} + \frac{1}{n^2}\Sigma_{i=1}^n \Sigma_{j \geq i} cov(\theta_i, \theta_j)$

- so when they are correlated that covariance term will dominate since we are not taking independent samples from the join $P_{x \times y}$
- we can reduce this dependance between estimates using random forests
- bootstrap samples are independent samples from the training set, but not independent samples from the joint distribution $P_{x \times y}$ (that is the distribution of clasesses with our dataset may not match that of the true data generating process)

random forests

- teh idea is to build a collection of trees except when constructing each tree node restrict the choice of splitting variable to a random subset of features
- when this is the case, a small subset off features can not deaminate all trees
- note that this does not eliminate all corelation between trees it just reduces it
- the choice of m (ie the size of subset of features we chose is important)

boosting

- bagging : reduce the variance of low bias, high variance estimators by enfeebling many estimators trained in parallel (on different bootstrapped datasets)
- boosting: reduce the error rate of high bias estimators by enameling many estimators in a trained sequence
- the main idea is instead of fitting the data very closely using one large Decision tree, train gradually using a sequence of simpler trees

boosting overview

- a weak learner (base learner) is a classifier that just does a bit better than chance
- weak learners learn simple rules about the data one at a time
- the key ideas is that each weak learner focus on a different part of the training examples (re-weighted data) , and by doing this each can make a different contribution to the final prediction model
- a set of smaller simpler trees may be more interpretable

ada boost

- binary classification, with bay hypothesis space $\mathcal{H} = \{h : X \to \{-1, 1\}\}$ that is we learning functions that map our input space to the output space.
- each base learner is trained on the weighted data.
- training set $\mathcal{D} = \{(x_1, y_1) \cdots (x_n, y_n)\}$
- weights are $w \in \mathbb{R}^n$ are associated with the examples
- the weighted empirical risk is

$$\hat{R}_n^w(f) := \frac{1}{W} \sum_{i=1}^{n} w_i \ell(f(x_i), y_i)$$

  where $W = \sum_i w_i$

sketch of ada boost

- start with equal weights $w_i = 1 \forall i \in [1, n]$
- repeat for $m = 1 \cdots M$ (where M is the number of classifiers we want ot train)
- train base classifier $G_m(x)$ on the weighted training data, this classifier may not fit well
- increase the value of weights in elements that $G_m(x)$ got wrong
- our final predictor is $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$ α is another weighting parameter but for our model
- we want α_m to be non negative and larger for G_m that do well
- the weighted zero one loss of g_m is

$$err_m = \frac{1}{W} \sum_{i=1}^{n} w_i \mathbb{I}(y_i \neq G_m(x_i))$$

  so that is a weighted average of the points the classifier missed with added weight on the points it was assigned to focus on
- then we set $\alpha_m = ln(\frac{1-err_m}{err_m})$ so that means having higher weighted error for na estimator means that estimator will have lower weight in our final prediction

- here is the full algorithm

AdaBoost: Algorithm

Given training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.

1. Initialize observation weights $w_i = 1$, $i = 1, 2, \ldots, n$.
2. For $m = 1$ to M:
   1. Base learner fits weighted training data and returns $G_m(x)$
   2. Compute *weighted empirical 0-1 risk*:

$$\text{err}_m = \frac{1}{W} \sum_{i=1}^{n} w_i 1(y_i \neq G_m(x_i)) \quad \text{where } W = \sum_{i=1}^{n} w_i.$$

   3. Compute *classifier weight*: $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$.
   4. Update *example weight*: $w_i \leftarrow w_i \cdot \exp[\alpha_m 1(y_i \neq G_m(x_i))]$
3. Return *voted classifier*: $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

- so notice here that we update our weights at each step to take into account the classifier weights that is $w_i \leftarrow w_i e^{\alpha_m} = w_i \frac{1-err_m}{err_m}$
- so that is if one of our estimators did badly on points it was supposed to do well on we increase the weights on those points further for the next estimator
- ada boost does not tend to overfit
- then there is this weird filtering for faces example