Lecture 9: Decision Trees

wbg231

December 2022

introduction

- i am going to try to write my in class notes on latex today and see how it goes.
- these are out first inherently non linear classifier
- we will also discuss ensemble tree methods.

Decision Trees

regression trees

- previously we discussed classifiers that are typically linear, as well as non-linear extensions using kernels
- but they are still fundamentally linear.
- suppose we want to predict basketball player salaries. if there years are bellow a certain value predict a certain number, if it is greater than or equal to a certain number go to another branch and then consider there shot percentage and branch the pay based off of that
- so there are three values depending on 2 features, that can be thought of as a vertical and horizontal Decision boundary that is in 2 dimensions.
- in higher dimensions we just kind of scale up this idea

classification

- we can extend the idea naturally to classification where we just assign regions in deferent sections of our Decision thresholds as a different classes
- the leaf nodes gives us the value we predict
- the thinking is we partition the data until we are comfortable with the outcome

tree set up

- binary tree 2 children
- each branch is serrated based off of an if else statement
- each node has a subset of data points, and the parent node has the union of all the children data points
- for a continuous variable we may say $y \leq c$ for some constant
- for discrete we would say something like $y[i] = c$
- then the leaves are terminal nodes we use for prediction

constructing a tree

- want to find the optimal Decision tree that minimizes mean square error
- often finding the exact minimum is imposable because there is a combinatorial explosion of trees
- we can try to approximate using a greedy approximation - that is make the best choice at each point
- this may not results in the global optimal trees
- can think of this as a 3d cube.

how to find the best split point

- we could enumerate all possible split points, and try all of them this does not work for real numbers
- we are only working with a finite number of data points. that we can use to train
- we want to think of all possible intervals not all possible real numbers
- we only need to think about split point between two possible real values.
- assume that all possible values are sorted. we think of any input within the same interval as equal
- it is common to split halfway between each adjacent values
- there are n examples, so we will get n-1 possible intervals

overfilling

- as we keep splitting data each data point gets its own region so that is pretty much a nearest neighbor model of n=1
- this will over fit the training data
- we want to have some generalization
- so how do we control the complexity of our hypothesis space? could look at depth of the tree, limit the number of total nodes, limit the number of terminal nodes (ie total number of subregions), or we could require that each terminal node holds a set number of data points
- there is also backward pruning we can start with a really over fit tree, than slowly prune out the nodes that are similar to one another.
- as we keep pruning the tree our validation error will go down to a point

what makes a good split?

- we have n-1 possible split points
- in classification we want to predict the majority label for each region
- we want splits that have a higher majority ie the region has a better consensus.
- sometimes it is hard to tell which split is better, in some case prefer lower average error, in some case prefer lower variance
- we want in general pure nodes that have the highest consensus

misclassification error in a node

- $\hat{p}_{nk}$ is the error guessing a certain class
- we predict that class that gets the least on average wrong for each region
- there are some impurity measures for node
- some a misclassification error
- some are gini index which want to distribution to be most balanced
- we can also use entropy a sa measure.
- gini index and entropy both work better in practice than misclassification error
- misclassification error is aline
- gini index or entropy are kind of bowed down

quantifying the impurity of a split

- we can now use these ideas to quantifying the impurity of each node
- and use that to quantify the impurity of e each split
- we can use $Q(n)$ do denote the impurity measure for each node
- then we can take weighted average of all impurity measures over all nodes in a split as the impurity measure of the split

tree interminability

- small trees can be written out easily and understood
- larger or deeper trees are a lot less interminable
- linear models are really good with linear relationships, trees are not as good with linear models,
- but when the thing is not linear a linear model will fail overall

review

- trees are non-metric they do not rely on the geometry of the space. we do not need inner products or anything like taht
- they are non-linear
- they are also non-parametric. a parametric model means we are not making assumptions on our data distribution but we are still learning model parameters
- they are also interminable when small
- cons for them are they are not good at capturing linear Decision boundaries
- really large trees have very high variance
- they tend to over fit

ensemble methods

recap of stats

- we have a data set that is iid.

- a list of n data points
- sampled form a parametric distribution $P(*|\theta)$
- a statistic is a function of the dataset $s = s(\mathcal{D})$ like the sample variance, mean or covariance of our data
- a point estimation

bias and variance of estimators

- statistic are random variables because the data set is random
- standard deviation of estimators called standard error
- distribution of estimator is called hte sampling distribution
- bias is $bias = E[\hat{\theta}] - \theta$
- variance is defined as usual
- why does variance mater if estimator is unbiased?
- basically because variance is risk we want to minimizes that
- ideally we want low bias and low variance

variance of the mean estimator

- let $\hat{\theta}$ be unbiased estimator with variance σ^2
- in order to reduce variance we can take a lot of realizations of $\hat{\theta}$ call them $\hat{\theta}_1 ... \hat{\theta}_n$ where each were trained on a different data set then we take the average of all of them
- we know that the average of an average is still unbiased
- $var(\frac{1}{n}\Sigma_i^n \hat{\theta}_i) = \frac{\sigma^2}{n}$
- but this requires that we have n different data sets to use for training
- so we can have n independent prediction functions and call the average prediction function the average of our prediction functions
- again averaging reduces variance of our prediction, while still being unbiased
- this shows how we can get a smaller variance from many different estimators based on many different coppices of the training set.

bootstrap

- we can take a sample from our original data set (assuming it is large and iid) and treat that as a stand in for our overall population
- we sample with replacement because it allows all samples to be independent
- we can quantify how likely it is one sample does not show up in b samples of size n.

bootstrap methods

- simulate b samples from the distribution by taking b bootstrap samples of size n form the original data
- for each bootstrap sample we can compute our estimator of interest
- we are going to use these values, as if they were drawn from the original distribution
- we can empirically show that this often works well.

ensample methods

- we combine multiple weak models into a single stronger model
- a weak model should have a higher bias, but will have a lower variance over all
- we can use bootstrap idea to average and try to overcome this issue.
- two ideas parallel ensample (bagging) build models in parallel and average at end
- or could bag where we build one model and try to build the next model sequentially to mitigate mistakes of each other

bagging

- short hand for bootstrap aggregating
- we take b bootstraps and fit one weak learners for each of these models
- the bagged prediction function is a combination function (usuals the weighted average) of all prediction functions
- for classification we can just take the majority vote

- what is nice about this is that we can keep increasing the number of trees and it does not lead to overfilling became we are not increasing the depth of each tree its self
- the downside of this is that the predictor is less interminable so that is not ideal, sine we would have to explain many different trees that we are using as a majority vote

out of bag error estimation

- we know that around 63% of our data will appear in our training sample
- the remaining 37% can be used as validation which we will call out of bag training data.
- we can form another set that never appeared in any of our training models and use that as a validation set.
- this will be a good estimator of our test error, and will help us measure current performance

bagging classification example

- building a single tree vs boots strapping our data
- when we sample our new data set it is likely we will end up in different tree
- the model has high variance, but if we average them then we overall achieve less variance
- it helps the most when base learners have low bias, but have high variance

motivating random forests

- the motivating principle of bagging is that we can have multiple iid estimators that we can average over to reduce variance
- what if the estimators are correlated? for a large n the covariance will dominate our limiting the befit of averaging
- bootstrap are independent samples from the training set but they are not independent from the distribution they are all depending on the same distribution
- so how do we reduce the

random forests

- we want to reduce dependance between trees.
- when constructing each tree we only look at a subset of features to use as splitting variables this means that the estimators end up being less correlated
- let m be the number of features each individual tree can chose
- when we start it out from one tree we have a high error, because each tree can only look at a few features
- but as we average more and more trees the error goes down
- the one with square root of p has the lowest test error because it is reducing the correlation between our trees

review

- single tree has low bias high variance
- ensample reduce variance at the cost of some bias
- can use bootstrap to simulate many samples from one dataset
- bootstrap samples are however correlated
- so we can use random forest to try to reduce the correlation between each of our predictors so we can average them together to reduce variance

boosting

- this is a sequential method
- bagging each estimator trained in parallel
- boosting we want to reduce the error rate of a high bias estimators by ensample estimators that are trained in sequentially (without boot strapping)
- like bagging boosting is a general method that is popular with Decision trees the idea is to fit the data very closely as a combination of simple trees.

overview boosting

- start with a weak learner (that only does a bit better than chance)
- each weak learner focused on different trailing examples (re weighted data) this works because we want later models to fix the errors of the earlier ones
- we will focus on ada boosting first

ada boots

- the setting is binary classification $y = \{-1, 1\}$
- we have a base hypothesis space
- we want a weak learner with a single split sometimes, or a tree with very few splits, or a linear Decision function

weighted training data

- have a dataset that is a tuple of x,y
- have a weight vector w that is associated with each sample
- we have weighted empirical risk
- we still have to assign the weight some where. but if we have a uniform weight it is the same as the original empirical risk

diagram

- first train from training sample with uniform weights
- train classifier g_1
- up wight samples that g_1 got wrong to train g_2
- up wight samples that g_1, g_2 got wrong for g_3
- then in the end we take a final classifier which is a weighted sum of all of our classifiers weighted by there accuracy

algorithm sketch

- start with equal weights
- repeated m times
- train classifier on weighted training data
- increase weights of points that are misclassified by current classifier
- in the end predict as a weighted sum of our predictors weighted by accuracy
- want $\alpha_i \geq 0$ and $\alpha_i = ln(\frac{1-err_i}{err_i})$ large if estimator i does a good job
- higher weighted error means we assign a lower weight
- let w_i be current weight, want w_{i+1} to be weights at next round
- if g_m classifies x_i correctly keep w_i same
- otherwise we update it as $w_{i+1} \leftarrow w_i e^{\alpha_m}$

algorithm

- initially all weights uniform
- i think i get the broad idea of the algorithm but it is hard to write down as he is talking
- just look on slide 48 it is there.

ada boost with Decision stump

- Decision stump means only one Decision criteria at the root node
- after 120 rounds (still only having our tree depth as 1) we can learn a pretty sophisticated Decision boundary that does pretty well

does ada boost over fit

- all of the individual trees are quite simple, but it is possible that ada boost can over fit especially if we have a really large number of trees
- in practice ada boots is in general pretty resistant to over fitting.
- the test error can continue to decrease even as a training eros may go up
- often this algorithm does pretty well

ada boost for face detection

- this is from 2001
- we are trying to detect faces in images
- we want to understand how to locate peoples face in real time
- this was done using a tweaked ada boost
- it takes a pre defined weak classifier
- we already have a fixed set of weak classifiers one just need to be picked form there
- it has a good way to do inference in real time
- it has a rectangular filter, that has weights to be one or minus one
- we multiply this matrix of weights with the reginal image pixels and return the sum of the product within the tree.
- for each box we are going to look for this little rectangular region that we can defined as either a vertical or rectangular split
- we want to keep it simple when possible
- over 180,000 Decision trees to select from even with this
- how do we do this efficient?

integral image

- this is kinda of a dynamic programming approach
- store the sum in the 2d array for all pixels
- and if we want the sum of in some pixels we can just subtract the areas over any rectangular region
- so any filter can be decomposed as a sum of rectangular region
- and at most we are doing 16 times retrieval as opposed to doing multiplication

learning procedure

- we can think of our learning procedure as similar to what we learned in the ada boost algorithm
- here we want to show that this works in different settings
- so initially we set weights as certain things depending on there class (so in this case if they are a face) this lets us up weight certain samples to over come class imbalance
- for each feature j we train a classifier h_j which returns a number between zero and one then we take the one with lowest error
- we can update our example weights and don't update if we get it correct
- the final classifier is one if we pash a certain threshold and zero otherwise
- the other trick is if we have 1000 classifiers in our ensemble we can cut out certain classifiers depending on the space
- we can adjust the threshold of each classifier to make sure that there is no false negative since in this case we want to make sure we get all the faces.
- a false negative means there is a face, but we do not think there is a face .
- so then if one model says that it is zero we can say that it is not a face because we have made sure that false negatives very rarely happen, then we can not run the rest of the models when there is a 0 predicted by one of them.

summary

- boosting is used to reduce bias from shallowing Decision trees
- each classifier is trained to reduce errors of its previous ensemble
- ada boost is very powerful
- next week we are going to make boosting algorithms more general (why are we weighting what is the objective function)
- what is gradient boosting?
- ok that is the end of lecture.
- we can expand this to multi class models as well they are pretty easy to modify and expand as need be in this case
-