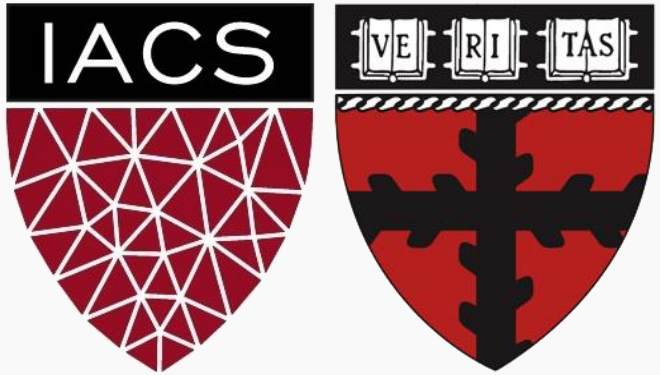
Advanced Section #7:
Decision trees and Ensemble methods

Camilo Fosco

CS109A Introduction to Data Science

Pavlos Protopapas and Kevin Rader



Outline

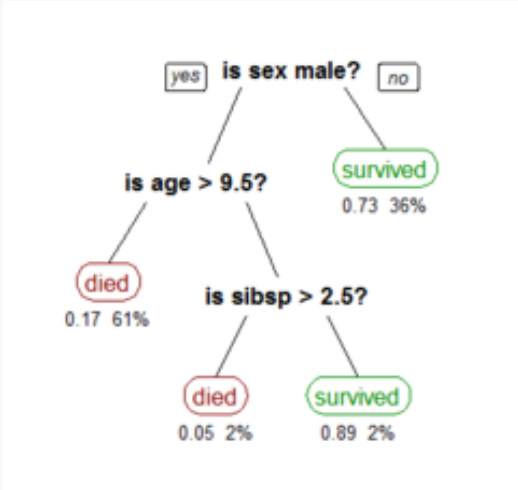
- Decision trees
  - Metrics
  - Tree-building algorithms
- Ensemble methods
  - Bagging
  - Boosting
  - Visualizations
- Most common bagging techniques
- Most common boosting techniques

DECISION TREES

The backbone of most techniques

What is a decision tree?

- Classification through sequential decisions.
- Similar to human decision making.
- Algorithm decides what path to follow at each step.
- The tree is built out by choosing features and thresholds that minimize the error of the prediction product, based on different metrics that we'll explore next.



Metrics for decision tree learning

Gini impurity Index: measures how often a randomly chosen element from a subset S would be incorrectly labeled if randomly labeled following the label distribution of the current subset.

$$Gini(S) = 1 - \sum_{i=1}^{J} p_i^2$$

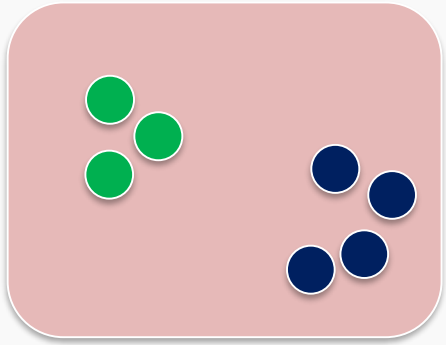
Number of classes

Proportion of elements of class i in subset S

- Measures purity.
- When all elements in S belong to one class (max purity), the sum equals one and the gini index is thus zero.

Metrics for decision tree learning

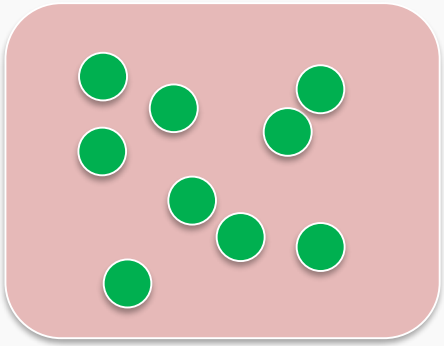
Gini examples:



Gini = P(picking green)P(picking label black) + P(picking black)P(picking label green)

= 1 – [P(picking green)P(picking label green) + P(picking black)P(picking label black)]

$$= 1 - \left[\frac{3}{7}\cdot\frac{3}{7} + \frac{4}{7}\cdot\frac{4}{7}\right] = 0.4898$$



Gini = P(picking green)P(picking label black) + P(picking black)P(picking label green)

= 1 – [P(picking green)P(picking label green) + P(picking black)P(picking label black)]

$$= 1 - [1\cdot 1 + 0\cdot 0] = 0$$

Metrics for decision tree learning

Information Gain (IG): Measures difference in entropy between parent node and children given a particular split point.

Subset S (parent)

Split point

$$\mathrm{IG}(\mathrm{S},\mathrm{a}) = \mathrm{H}_{\mathrm{parent}}(\mathrm{S}) - \mathrm{H}_{\mathrm{children}}(\mathrm{S}|\mathrm{a})$$

Entropy (parent)

Weighted sum of entropy (children)

Where H is entropy, defined as:

$$H(T) = -\sum_i p_i \log_2 p_i$$

And the p_i correspond to the fractions of each class present in a child node resulting from a split in the tree.

Metrics for decision tree learning

Misclassification Error (ME): we split the parent node's subset by searching for the lowest possible average misclassification error on the child nodes.

$$I_G(p) = 1 - \max(p_i)$$

- In practice, generally avoided as in some cases, the best possible split might not yield error reduction at a given step.
- In those cases, the algorithm finishes and tree is cut short.

Tree-building algorithms

ID3: Iterative Dichotomiser 3. Developed in the 80s by Ross Quinlan.

- Uses the top-down induction approach described previously.
- Works with the IG metric.
- At each step, algorithm chooses feature to split on and calculates IG for each possible split along that feature.
- Greedy algorithm.

Tree-building algorithms

C4.5: Successor of ID3, also developed by Quinlan ('93). Main improvements over I3D:

- Works with both continuous and discrete features, while ID3 only works with discrete values.
- Handles missing values by using **fractional cases** (penalizes splits that have multiple missing values during training, fractionally assigns the datapoint to all possible outcomes).
- Reduces overfitting by pruning, a bottom-up tree reduction technique.
- Accepts weighting of input data.
- Works with multiclass response variables.

Tree-building algorithms

CART: Most popular tree-builder. Introduced by Breiman et al. in 1984. Usually used with Gini purity metric.

- Main characteristic: **builds binary trees.**
- Can work with discrete, continuous and categorical values.
- Handles missing values by using **surrogate splits**.
- Uses **cost-complexity pruning**.
- Sklearn uses CART for its trees.

Many more algorithms…

Feature	C4.5	CART	CHAID	CRUISE	GUIDE	QUEST
Unbiased Splits				√	√	√
Split Type	*u*	*u,l*	*u*	*u,l*	*u,l*	*u,l*
Branches/Split	≥2	2	≥2	≥2	2	2
Interaction Tests				√	√	
Pruning	√	√		√	√	√
User-specified Costs		√	√	√	√	√
User-specified Priors		√		√	√	√
Variable Ranking		√			√	
Node Models	*c*	*c*	*c*	*c,d*	*c,k,n*	*c*
Bagging & Ensembles					√	
Missing Values	*w*	*s*	*b*	*i,s*	*m*	*i*

b, missing value branch; *c*, constant model; *d*, discriminant model; *i*, missing value imputation; *k*, kernel density model; *l*, linear splits; *m*, missing value category; *n*, nearest neighbor model; *u*, univariate splits; *s*, surrogate splits; *w*, probability weights

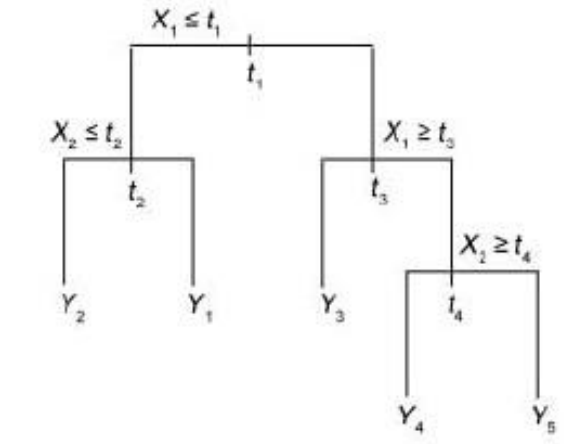
Regression trees

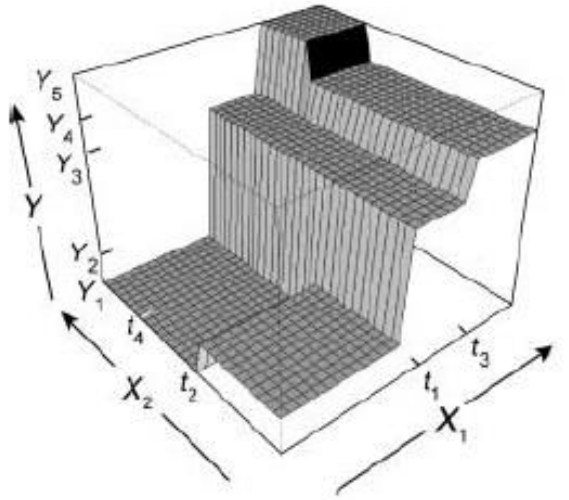
Can be considered a *piecewise constant regression model*.

Prediction is made by averaging values at given leaf node.

Two advantages: interpretability and modeling of interactions.

- The model's decisions are easy to track, analyze and to convey to other people.
- Can model complex interactions in a tractable way, as it subdivides the support and calculates averages of responses in that support.





Regression trees

Question: how do we build a regression tree?

Least Squares Criterion (implemented by CART):

1. For each predictor, split subset at each observation (quantitative) or category (categorical) and calculate the variance of each split.
2. Average variances, weighted by the number of observations in each split. This corresponds to calculating an impurity measure:

$$Q(split) = \sum_{m=1}^{M} \frac{|R_m|}{N} \sum_{y_i \in R_m} (y_i - \bar{c}_m)^2$$

   Where N is the number of elements in the node before splitting, M is the number of regions after the split, $|R_m|$ is the number of elements in splitted region m, and $\bar{c}_m$ is the average response in region R_m.
3. Choose split with smaller impurity.

Regression trees - Cons

Two major disadvantages: difficulty to capture simple relationships and instability.

- Trees tend to have high variance. Small change in the data can produce a very different series of splits.
- Any change at an upper level of the tree is propagated down the tree and affects all other splits.
- Large number of splits necessary to accurately capture simple models such as linear and additive relationships.
- Lack of smoothness.

Surrogate splits

- When an observation is missing a value for predictor X, it cannot get past a node that splits based on this predictor.
- We need surrogate splits: Mimic of original split in a node, but using another predictor. It is used in replacement of the original split in case a datapoint has missing data.
- To build them, we search for a feature-threshold pair that most closely matches the original split.
- "Association": measure used to select surrogate splits. Depends on the probabilities of sending cases to a particular node + how the new split is separating observations of each class.

Surrogate splits

- Two main functions:
  - They split when the primary splitter is missing, which could never happen in the training data, but being ready for future test data increases robustness.
  - They reveal common patterns among predictors in dataset.
- No guarantee that useful surrogates can be found.
- CART attempts to find at least 5 surrogates per node.
- Number of surrogates usually varies from node to node.

Surrogate splits - example

- Imagine situation with multiple features, two of them being phone_bill (continuous) and marital_status (categorical)
- Node 1 splits based on phone_bill. Surrogate search might find that marital_status = 1 generates a similar distribution of observations in left and right node.
- This condition is then chosen as top surrogate split.

	Left child	Right child
Phone_bill > 100	550R, 99G	50R, 301G
Marital_status = 1	510R, 128G	51R, 311G

Surrogate splits - example

- In our example, primary splitter = phone_bill
- We might find that surrogate splits include marital status, commute time, age, city of residence.
  - Commute time associated with more time on the phone
  - Older individuals might be more likely to call vs text
  - City variable hard to interpret because we don't know identity of cities

- Surrogates can help us understand primary splitter.

Pruning

Reduces the size of decision trees by removing branches that have little predictive power. This helps reduce overfitting. Two main types:

- **Reduced Error Prunning:** Starting at leaves, replace each node with its most common class. If accuracy reduction is inferior than a given threshold, change is kept.
- **Cost Complexity Pruning:** remove subtree that minimizes the difference of the error of pruning that tree and leaving it as is, normalized by the difference in leaves:

$$\frac{err(T,S) - err(T_0,S)}{|leaves(T)| - |leaves(T_0)|}$$

Cost Complexity Pruning

- Denote the large tree T_0, and define a subtree T $\subset T_0$ as a tree that can be obtained by collapsing any number of its internal nodes.
- We then define the cost-complexity criterion:

$$C_\alpha(T) = L(T) + \alpha|T|$$

where L(T) is the loss associated with tree T, |T| is the number of terminal nodes in tree T, and α is a tuning parameter that controls the tradeoff between the two.

The pruning algorithm, as seen in the lecture:

1. Start with a full tree T_0 (each leaf node is pure)
2. Replace a subtree in T_0 with a leaf node to obtain a pruned tree T_1. This subtree should be selected to minimize

$$\frac{Error(T_0) - Error(T_1)}{|T_0| - |T_1|}$$

3. Iterate this pruning process to obtain $T_0, T_1, \ldots, T_L$ where T_L is the tree containing just the root of T_0
4. Select the optimal tree T_i by cross validation.

This corresponds to minimizing $C_\alpha(T)$.

ENSEMBLE METHODS

Assemblers 2: Age of weak learners

What are ensemble methods?

- Combination of weak learners to increase accuracy and reduce overfitting.
- Train multiple models with a common objective and fuse their outputs. Multiple ways of fusing them, can you think of some?
- Main causes of error in learning: noise, bias, variance. Ensembles help reduce those factors.
- Improves stability of machine learning models. Combination of multiple learners reduces variance, especially in the case of unstable classifiers.

What are ensemble methods?

- Typically, decision trees are used as base learners.
- Ensembles usually retrain learners on subsets of the data.
- Multiple ways to get those subsets:
  - Resample original data with replacement: Bagging.
  - Resample original data by choosing troublesome points more often: Boosting.
- The learners can also be retrained on modified versions of the original data (gradient boosting).

Bagging

- Boostrap aggregating (Bagging): ensemble meta-algorithm designed to improve stability of ML models.
- Main idea:
  - resample data to generate a subset S.
  - Train a weak learner $\hat{g}^*$, e.g. tree stumps, on the sampled data.
  - Repeat the process K times. When done, combine the K classifiers into one classifier by averaging or maj-voting the outputs:

Regression: $$\hat{g}_{bag}(\cdot) = \frac{1}{K}\sum_{i=1}^{K} \hat{g}_i^*(\cdot)$$ (Average)

Classification: $$\hat{g}_{bag}(\cdot) = \underset{j}{\text{argmax}} \sum_{i=1}^{K} \mathbb{I}_{j=\hat{g}_i^*(\cdot)}$$ (Majority Vote)

Bagging

- Bagging is generally not recommended when the simple classifier shows high bias, as the technique does no bias reduction.
- Variance is strongly diminished.

Question: should we subsample with or without replacement?

Answer: both work. Typically, with replacement is used. See "Observations on Bagging", Buja et al., 2006* - proves that identical results are obtained if:

$$\frac{(N-1)}{M_{with}} = \frac{N}{M_{wo}} - 1$$

Number of observations

Sample size with replacement

Sample size without replacement

*Buja and Stuetzel, 2006, section 7.

Boosting

- Sequential algorithm where at each step, a weak learner is trained based on the results of the previous learner.
- Two main types:
  - **Adaptive Boosting**: Reweight datapoints based on performance of last weak learner. Focuses on points where previous learner had trouble. Example: AdaBoost.
  - **Gradient Boosting**: Train new learner on residuals of overall model. Constitutes gradient boosting because approximating the residual and adding to the previous result is essentially a form of gradient descent. Example: XGBoost.

Gradient Boosting

Gradient Boosting

- Task is to estimate target continuous function F(x). We measure goodness of estimation with loss function $L(y, F(x))$.
- Gradient boosting assumes that:

$$F(x) = \alpha_0 + \alpha_1 h_1(x) + \cdots + \alpha_M h_M(x)$$

- Basic Gradient boosting workflow:
  1. Initialize $F_0(x) = \alpha_0$
  2. Estimate α_m and $h_{m(x)}$ such that:

$$L(y, F_{m-1}(x) + \alpha_m h_m(x)) < L(y, F_{(m-1)}(x))$$

  3. Update $F_m(x) = F_{m-1}(x) + \alpha_m h_m(x)$
  4. Repeat from 2, M times.

Gradient Boosting

$$L(y, F_{m-1}(x) + \alpha_m h_m(x)) < L(y, F_{(m-1)}(x))$$

If we can find a vector r_m that we can plug in here to make this equation true, we can train a basic learner $h_m(x)$ to predict r_m from x!

We are basically searching for a vector that points to the direction that reduces our loss… does that sound familiar?

Gradient descent!

Gradient Boosting

By solving a simple 1D optimization problem, we could also find the optimal α_m for each step, by computing:

$$\alpha_m = argmin_\gamma L(y, F_{m-1}(x) + \gamma h_m(x))$$

This gives us an updated Gradient Boosting algorithm:

1. Initialize $F_0(x) = \alpha_0$
2. Compute negative gradient per observation: $r_{m_i} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$
3. Train base learner $h_m(x)$ on the gradients
4. Compute α_m with line search strategy
5. Update $F_m(x) = F_{m-1}(x) + \alpha_m h_m(x)$
6. Repeat from 2, M times.

Gradient Boosting

Where do the residuals come in?

If we consider Mean Squared Error as our loss function, the per-observation gradient is:

- $\frac{\partial L(y_i, F_m(x_i))}{\partial F_m(x_i)} = \frac{\partial\left(\frac{1}{2n}\sum_i (y_i - F_m(x_i))^2\right)}{\partial F_m(x_i)} = \frac{\partial\left(\frac{1}{2}(y_i - F_m(x_i))^2\right)}{\partial F_m(x_i)} = y_i - F_m(x_i)$

The derivation we found before works with any loss function.

Gradient Tree Boosting

When dealing with decision trees, we can take the concept further by selecting a specific α_m for each of the tree's regions. The output of a tree is:

$$h_m(x) = \sum_{J_m} b_{jm} 1_{R_{jm}}(x)$$

Number of leaves

Disjoint regions partitioned by the tree

The model update rule becomes:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \alpha_{jm} \mathbf{1}_{R_{jm}(x)}$$

$$\alpha_{jm} = argmin_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$$

Let’s look at graphs!

GRAPH TIME

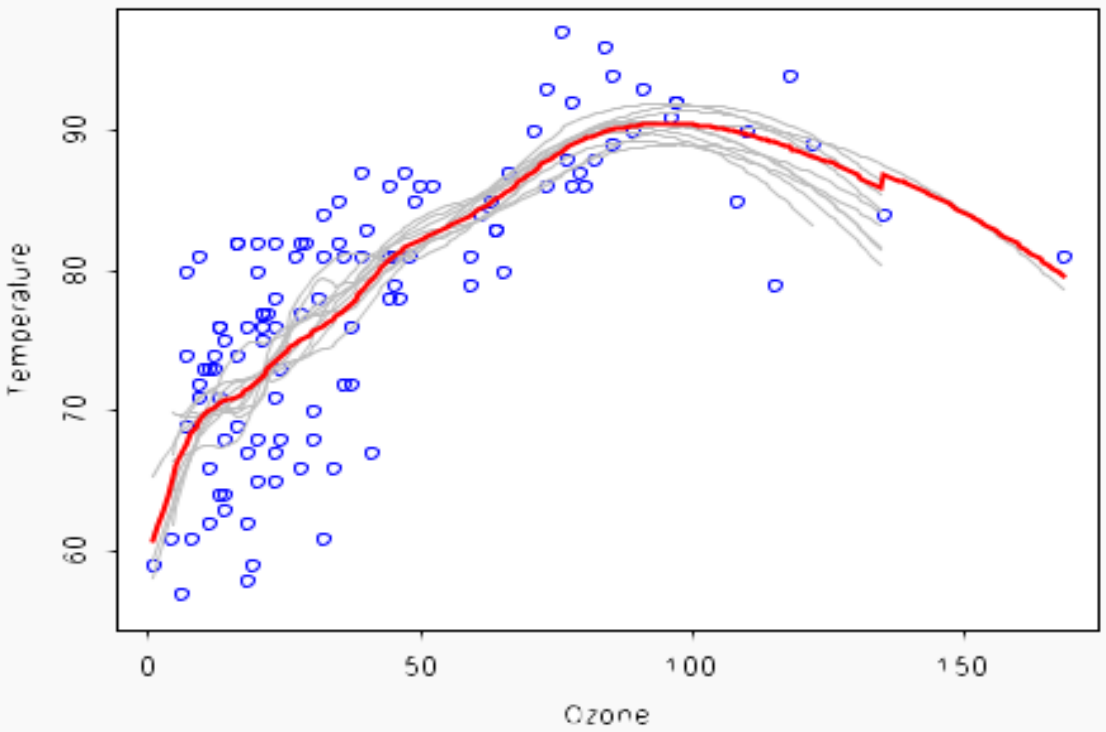
http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

COMMON BAGGING TECHNIQUES

Random Forests, of course.

Bagged Trees

- Basics of Bagging applied to the letter: resample dataset, train trees, combine predictions.
- Can be used in Sklearn with the BaggingClassifier() class.
- Pure bagged trees have generally worse performance than boosting methods, because of high tree correlation (lots of similar trees).



Question: why are these trees often correlated?

Random Forests

- Similar to bagged trees but with a twist: we now choose a **random subset of predictors** when defining our trees.
  - Question: Do we choose a random subset for each tree, or for each node?
- Random Forests essentially perform bagging over the predictor space and build a collection of **de-correlated trees**.
- This increases the stability of the algorithm and tackles correlation problems that arise by a greedy search of the best split at each node.
- Adds diversity, reduces variance of total estimator at the cost of an equal or higher bias.

Random Forests

Random Forest steps:

1. Construct subset $(x_1^*, y_1^*), \ldots, (x_n^*, y_n^*)$ by sampling original training set with replacement.
2. Build N tree-structured learners $h(x, \Theta_k)$, where at each node, M predictors at random are selected before finding the best split.
   - Gini Criterion.
   - No pruning.
3. Combine the predictions (average or majority vote) to get the final result.

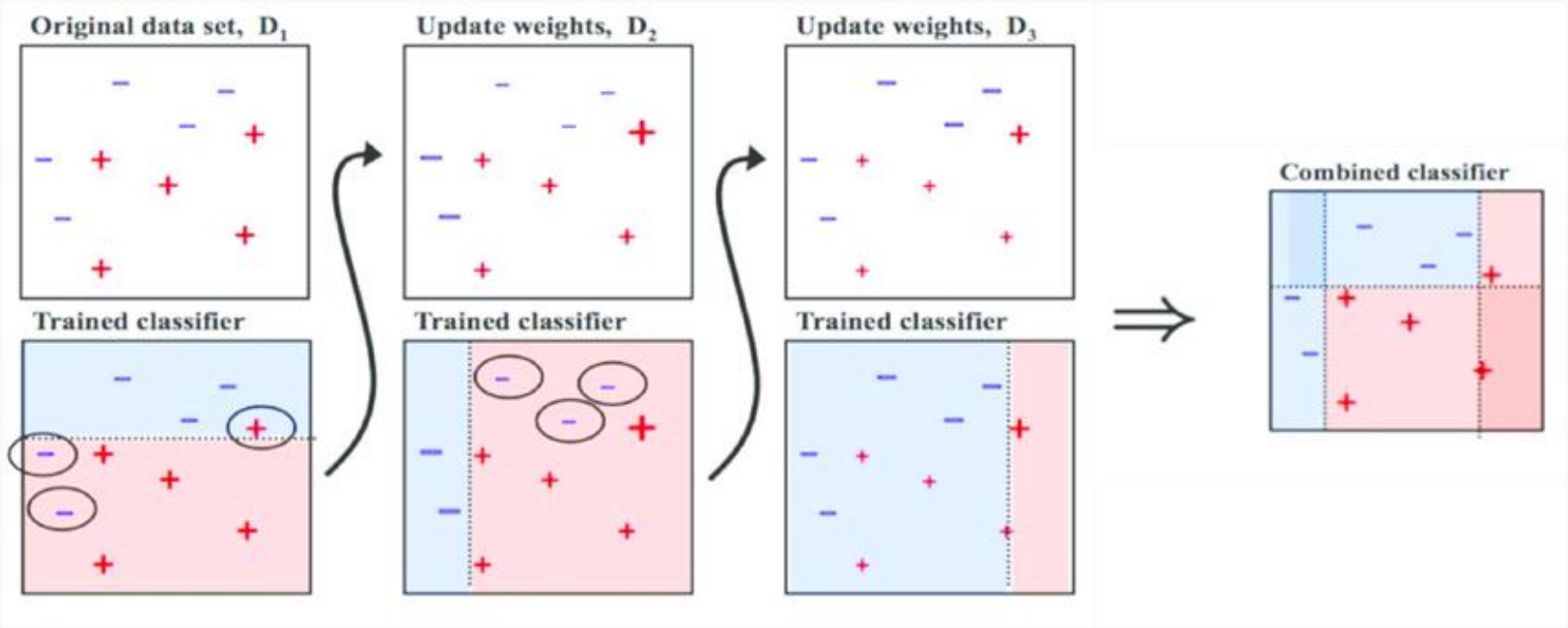
Question: why don't we need to prune?

COMMON BOOSTING TECHNIQUES

Kaggle killers.

AdaBoost

- AdaBoost is the essential boosting algorithm. It reweights the dataset before each new subsampling based on the performance of the last classifier.
- Main difference with bagging: SEQUENTIAL.



Instead of resampling, uses training set re-weighting. At each iteration, the re-weighting factor is given by:

$$\alpha_m = \frac{1}{2}\ln\frac{(1-\epsilon_m)}{\epsilon_m}$$

Where ϵ_m is the weighted error of weak classifier h_m:

$$\epsilon_m = \frac{\sum_{y_i \neq h_m(x_i)} w_i^{(m)}}{\sum_{i=1}^{N} w_i^{(m)}}$$

Letting $w_i^{(1)} = 1$ and $w_i^{(m)} = e^{-\left(y_i F_{m-1}(x_i)\right)}$ for m>1

It can be shown that AdaBoost can also be described in the gradient boosting framework, where the loss being minimized is exponential loss:

$$L = \sum_i e^{-y_i F(x_i)}$$

Splitting the loss into correctly and incorrectly classified datapoints and differentiating, we can get to the results above.

In general AdaBoost has been known to perform better than SVMs with less parameters to tune. Main parameters to set are:

- Weak classifier to use
- Number of boosting rounds

Disadvantages:

- Can be sensitive to noisy data and outliers.
- Must adjust for cost-sensitive or imbalanced problems
- Must be modified for multiclass problems

XGBoost

XGBoost is essentially a very efficient Gradient Boosting Decision Tree implementation with some interesting features:

- **Regularization:** Can use L1 or L2 regularization.
- **Handling sparse data:** Incorporates a sparsity-aware split finding algorithm to handle different types of sparsity patterns in the data.
- **Weighted quantile sketch:** Uses distributed weighted quantile sketch algorithm to effectively handle weighted data.
- **Block structure for parallel learning:** Makes use of multiple cores on the CPU, possible because of a block structure in its system design. Block structure enables the data layout to be reused.
- **Cache awareness:** Allocates internal buffers in each thread, where the gradient statistics can be stored.
- **Out-of-core computing:** Optimizes the available disk space and maximizes its usage when handling huge datasets that do not fit into memory.

XGBoost

Three main forms of gradient boosting are supported:

Gradient Boosting algorithm, as we defined above.

Stochastic Gradient Boosting with sub-sampling at the row, column and column per split levels.

- Random procedure where we subsample observations and features

Regularized Gradient Boosting with both L1 and L2 regularization.

- We add a regularization term to the loss function that we are optimizing:

$$L_R\big(y, F(x)\big) = L\big(y, F(x)\big) + \Omega(F)$$

Where $\Omega(F) = \gamma T + \frac{1}{2}\lambda\|w\|^2$

Leaf weights: prediction of each leaf

Number of leaves

XGBoost

- XGBoost uses second-order approximation to the loss function to quickly optimize the following objective:

$$L^{(m)} = \sum_i l\big(y_i, F_{m-1}(x_i) + h_m(x_i)\big) + \Omega(h_m)$$

The second order approximation is:

$$L^{(m)} \approx \sum_{\mathrm{i}=1}^{\mathrm{n}} l\big(y_i, F_{m-1}(x_i)\big) + g_i h_m(x_i) + \frac{1}{2} k_i h_m^2(x_i) + \Omega(h_m)$$

Second order gradient of loss w.r.t F(x)

First order gradient of loss w.r.t F(x)

Removing constant terms:

$$L^{(m)} \approx \sum_{\mathrm{i}=1}^{\mathrm{n}} g_i h_m(x_i) + \frac{1}{2} k_i h_m^2(x_i) + \Omega(h_m)$$

This expression is used in XGBoost to define a structure score for each tree. Expanding the regularization term, and defining $I_j = \{i|q(x_i) = j\}$ as the instance set of leaf j, we can compute the optimal weight of leaf j with:

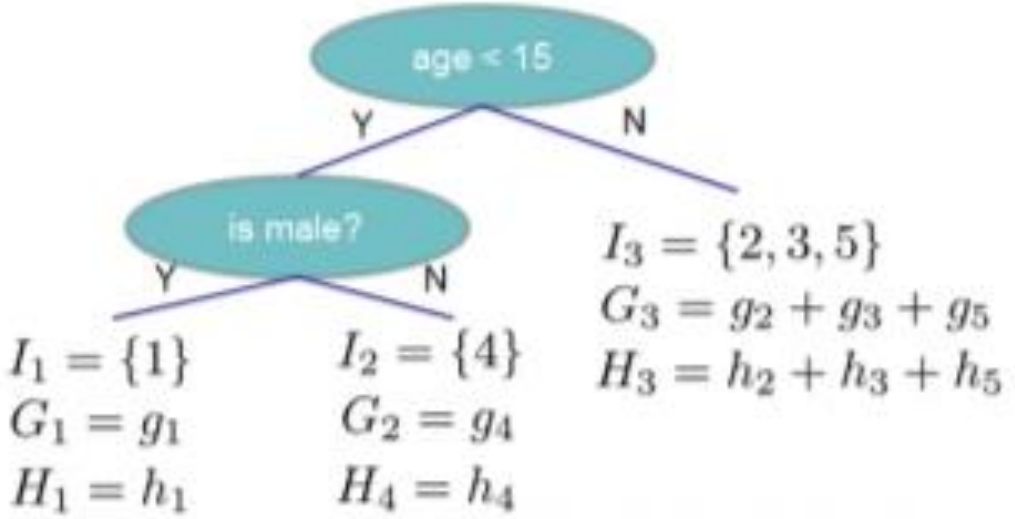
$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

With this, we can calculate the optimal loss value for a given tree structure:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T.$$

How would we calculate this in practice?

Instance index		gradient statistics
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5



$$Obj = -\sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

- Remember, we still want to find the tree structure that minimizes our loss, which means best score structure. Doing this for all possible tree structures is unfeasible.
- A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used instead.
- Assume that I_L and I_R are the instance sets of left and right nodes after the split. Letting $I = I_L \cup I_R$, then the loss reduction after the split is given by:

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

XGBoost adds multiple other important advancements that make it state of the art in several industrial applications.

In practice:

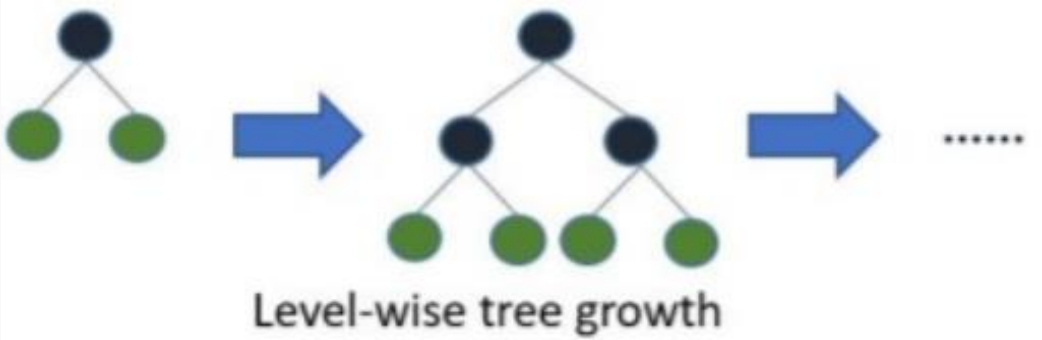
- Can take a while to run if you don't set the n_jobs parameter correctly
- Defining the eta parameter (analogous to learning rate) and max_depth is crucial to obtain good performance.
- Alpha parameter controls L1 regularization, can be increased on high dimensionality problems to increase run time.

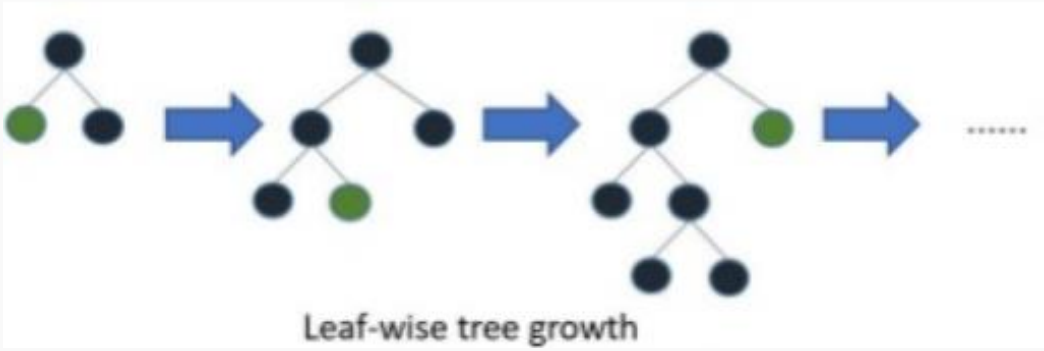
General approach to parameter tuning:

- Cross-validate **learning rate**.
- Determine the **optimum number of trees for this learning rate**. XGBoost can perform cross-validation at each boosting iteration for this, with the “cv” function.
- **Tune tree-specific parameters** (max_depth, min_child_weight, gamma, subsample, colsample_bytree) for chosen learning rate and number of trees.
- Tune **regularization parameters** (lambda, alpha).

LGBM

- Stands for Light Gradient Boosted Machines. It is a library for training GBMs developed by Microsoft, and it competes with XGBoost.
- Extremely efficient implementation.
- Usually much faster than XGBoost with low hit on accuracy.
- Main contributions are two novel techniques to speed up split analysis: Gradient based one-side sampling and Exclusive Feature Building.
- Leaf-wise tree growth vs level-wise tree growth of XGBoost.





Gradient-based one-side sampling (GOSS)

- Normally, no native weight for datapoints, but in can be seen that instances with larger gradients (i.e., under-trained instances) will contribute more to the information gain metric.
- LGBM keeps instances with large gradients and only randomly drops instances with small gradients when subsampling.
- They prove that this can lead to a more accurate gain estimation than uniformly random sampling, with the same target sampling rate, especially when the value of information gain has a large range.

Exclusive Feature Bundling (EFB)

- Usually, feature space is quite sparse.
- Specifically, in a sparse feature space, many features are (almost) exclusive, i.e., they rarely take nonzero values simultaneously. Examples include one-hot encoded-features.
- LGBM bundles those features by reducing the optimal bundling problem to a graph coloring problem (by taking features as vertices and adding edges for every two features if they are not mutually exclusive), and solving it by a greedy algorithm with a constant approximation ratio.

CatBoost

- A new library for Gradient Boosting Decision Trees, offering appropriate handling of categorical features.
- Presented as a workshop at NIPS 2017.
- Fast, scalable and high-performance. Outperforms LGBM and XGBoost on inference times, and in some datasets, in accuracy as well.
- Main idea: deal with categorical variables by using random permutations of the dataset and calculating the average label value for a given example using the label values of previous examples with the same category.

THANK YOU!

Random Forests – Generalization error

In the original RF paper, Breiman shows that an upper bound for RF's generalization error is given by:

$$PE^* \leq \frac{\bar{\rho}}{s^2}(1 - s^2)$$

Where s is the strength of the set of classifiers $h(x, \Theta)$:

$$s = E_{X,Y} mr(X, Y)$$

And mr(X,Y) is the margin function for random forests.

Two main components involved in RF generalization error:

- Strength of individual classifiers
- Correlation between them