Introduction to boosted decision trees

Katherine Woodruff

Machine Learning Group Meeting

September 2017

Outline

1. Intro to BDTs
   - Decision trees
   - Boosting
   - Gradient boosting
2. When and how to use them
   - Common hyperparameters
   - Pros and cons
3. Hands-on tutorial
   - Uses xgboost library (python API)
   - See next slide

Before we start...

The hands-on tutorial is in Jupyter notebook form and uses the XGBoost python API.

There are three options for following along:

1. Download the notebook from github and run it
   - You need Jupyter notebook, numpy, matplotlib, pandas installed
   - `git clone https://github.com/k-woodruff/bdt-tutorial`
   - The data used in the tutorial is included in the repository (only ~2MB)
   - Then just install xgboost (instructions are also in the notebook)
2. Copy the code from the notebook
   - If you don't have Jupyter, but do have numpy, matplotlib, and pandas
   - Can install xgboost and copy the code directly from the notebook and execute it in an ipython session
   - https://github.com/k-woodruff/bdt-tutorial/blob/master/bdt_tutorial.ipynb
   - Can download the data here: https://github.com/k-woodruff/bdt-tutorial/tree/master/data
3. Just observe
   - If you don't have and don't want to install the python packages
   - You can follow along by eye from the link in option 2

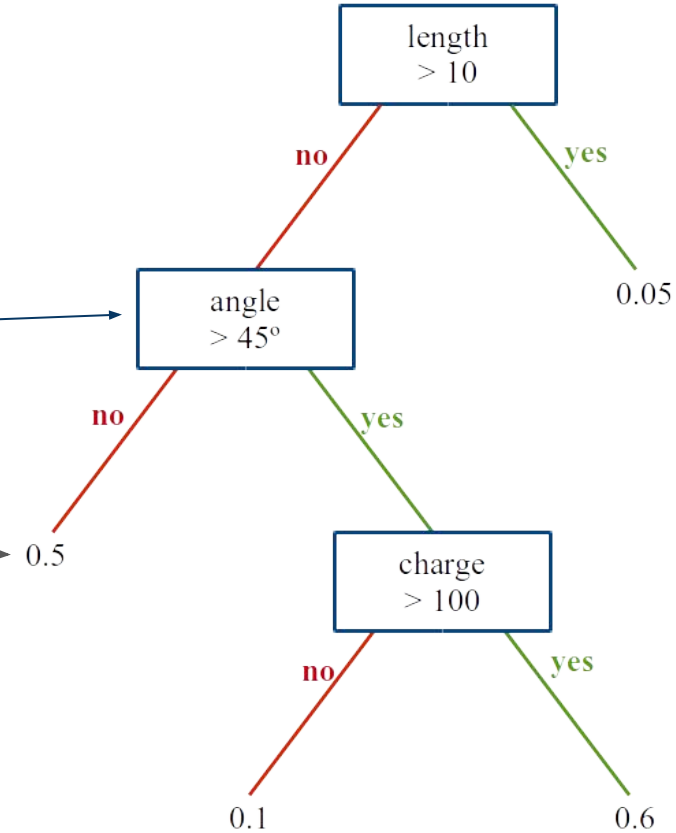
If you want to do 1 or 2 you should start the xgboost installation now.

Decision/regression trees

A decision tree takes a set of input features and splits input data recursively based on those features.

Structure:

- Nodes
  - The data is split based on a value of one of the input features at each node
  - Sometime called "interior nodes"
- Leaves
  - Terminal nodes
  - Represent a class label or probability
  - If the outcome is a continuous variable it's considered a "regression tree"



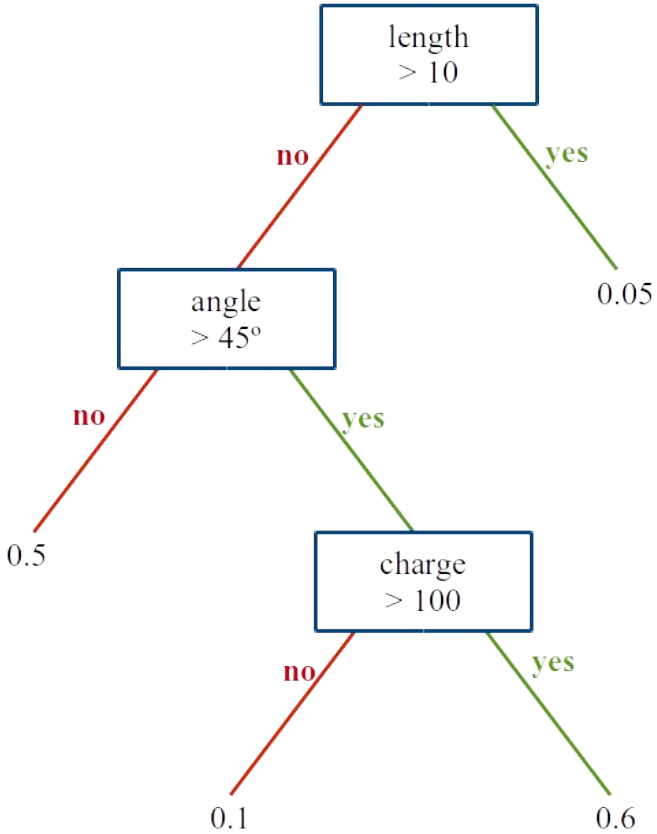
[1] https://en.wikipedia.org/wiki/Decision_tree_learning

Decision/regression trees

A decision tree takes a set of input features and splits input data recursively based on those features.

Learning:

- Each split at a node is chosen to maximize information gain or minimize entropy
  - Information gain is the difference in entropy before and after the potential split
  - Entropy is max for a 50/50 split and min for a 1/0 split
- The splits are created recursively
  - The process is repeated until some stop condition is met
  - Ex: depth of tree, no more information gain, etc...



Tree boosting

Boosting is a method of combining many weak learners (trees) into a strong classifier.

Usually:

- Each tree is created iteratively
- The tree's output (*h(x)*) is given a weight (*w*) relative to its accuracy
- The ensemble output is the weighted sum:

$$\hat{y}(x) = \sum_t w_t h_t(x)$$

- After each iteration each data sample is given a weight based on its misclassification
  - The more often a data sample is misclassified, the more important it becomes
- The goal is to minimize an objective function

$$O(x) = \sum_i l(\hat{y}_i, y_i) + \sum_t \Omega(f_t)$$

  - $l(\hat{y}_i, y_i)$ is the loss function --- the distance between the truth and the prediction of the *i*th sample
  - $\Omega(f_t)$ is the regularization function --- it penalizes the complexity of the *t*th tree

[1] https://en.wikipedia.org/wiki/Boosting_(machine_learning)

Types of boosting

There are many different ways of iteratively adding learners to minimize a loss function.

Some of the most common:

- AdaBoost
  - "Adaptive Boosting"
  - One of the originals
  - Freund and Schapire: http://www.sciencedirect.com/science/article/pii/S002200009791504X
- Gradient Boosting
  - Uses gradient descent to create new learners
  - The loss function is differentiable
  - Friedman: https://statweb.stanford.edu/~jhf/ftp/trebst.pdf
- XGBoost
  - "eXtreme Gradient Boosting"
  - Type of gradient boosting
  - Has become very popular in data science competitions
  - Chen and Guestrin: https://arxiv.org/abs/1603.02754

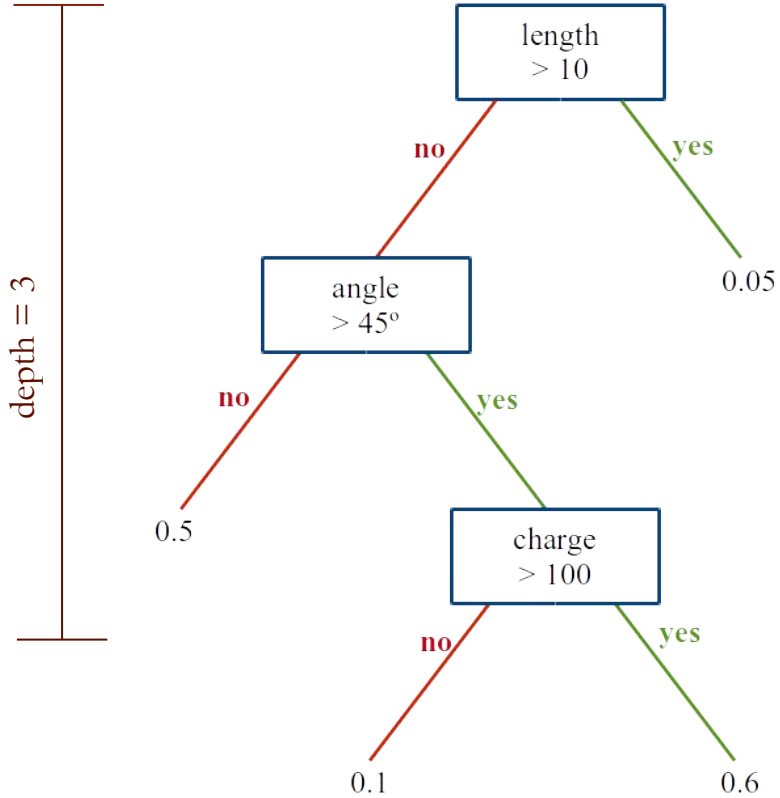
[1] https://en.wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting
[2] https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

Tunable parameters

Common **tree parameters**:

These parameters define the end condition for building a new tree. They are usually tuned to increase accuracy and prevent overfitting.

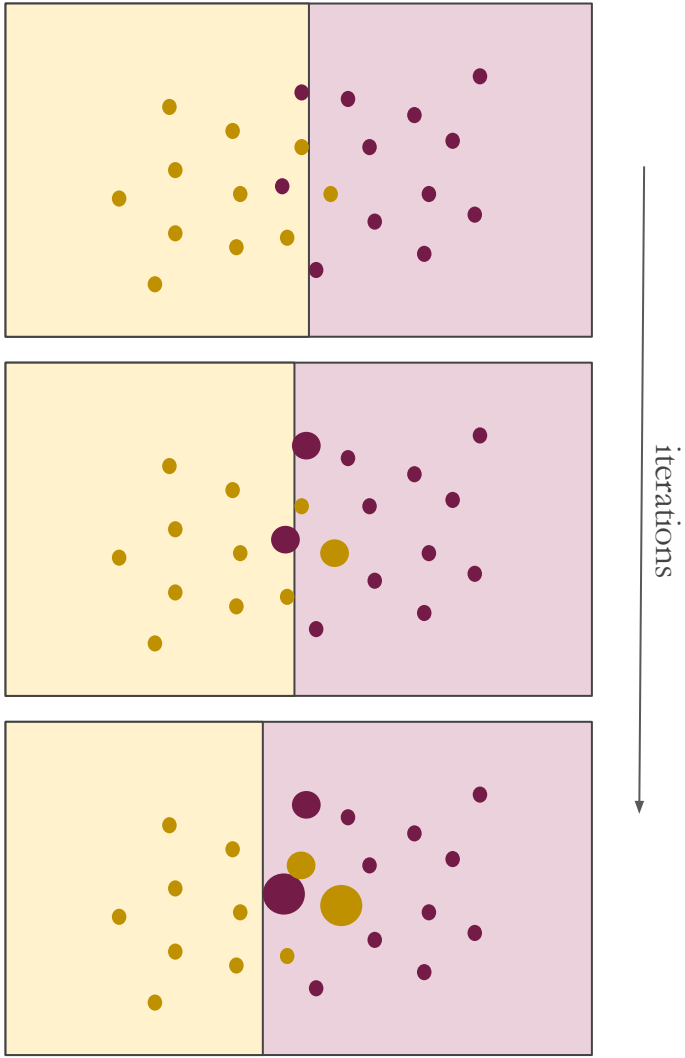
- Max. depth: how tall a tree can grow
  - Usually want < 10
  - Sometimes defined by number of leaves
- Max. features: how many features can be used to build a given tree
  - Features are randomly selected from total set
  - The tree doesn't have to use all of the available features
- Min. samples per leaf: how many samples are required to make a new leaf
  - Usually want < 1% of data
  - Sometimes defined by samples per split



Tunable parameters

Common **boosting parameters**:

- Loss function: How to define the distance between the truth and the prediction
  - Use binary logistic when you have two classes
- Learning rate: how much to adjust data weights after each iteration
  - Smaller is better but slower
  - Somewhere around 0.1
- Subsample size: How many samples to train each new tree
  - Data samples are randomly selected each iteration
- Number of trees: How many total trees to create
  - This is the same as the number of iterations
  - Usually more is better, but could lead to overfitting



[1] https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

Pros and cons of using boosted trees

Benefits:

- Fast
  - Both training and prediction is fast
- Easy to tune
- Not sensitive to scale
  - The features can be a mix of categorical and continuous data
- Good performance
  - Training on the residuals gives very good accuracy
- Lots of available software
  - Boosted tree algorithms are very commonly used
  - There is a lot of well supported, well tested software available.

Problems:

- Sensitive to overfitting and noise
  - Should always crossvalidate!
  - Modern software libraries have tools to avoid overfitting

Thanks!