

CS109/Stat121/AC209/E-109

Data Science

Ensemble Learning and Random Forests

Hanspeter Pfister, Joe Blitzstein, and Verena Kaynig

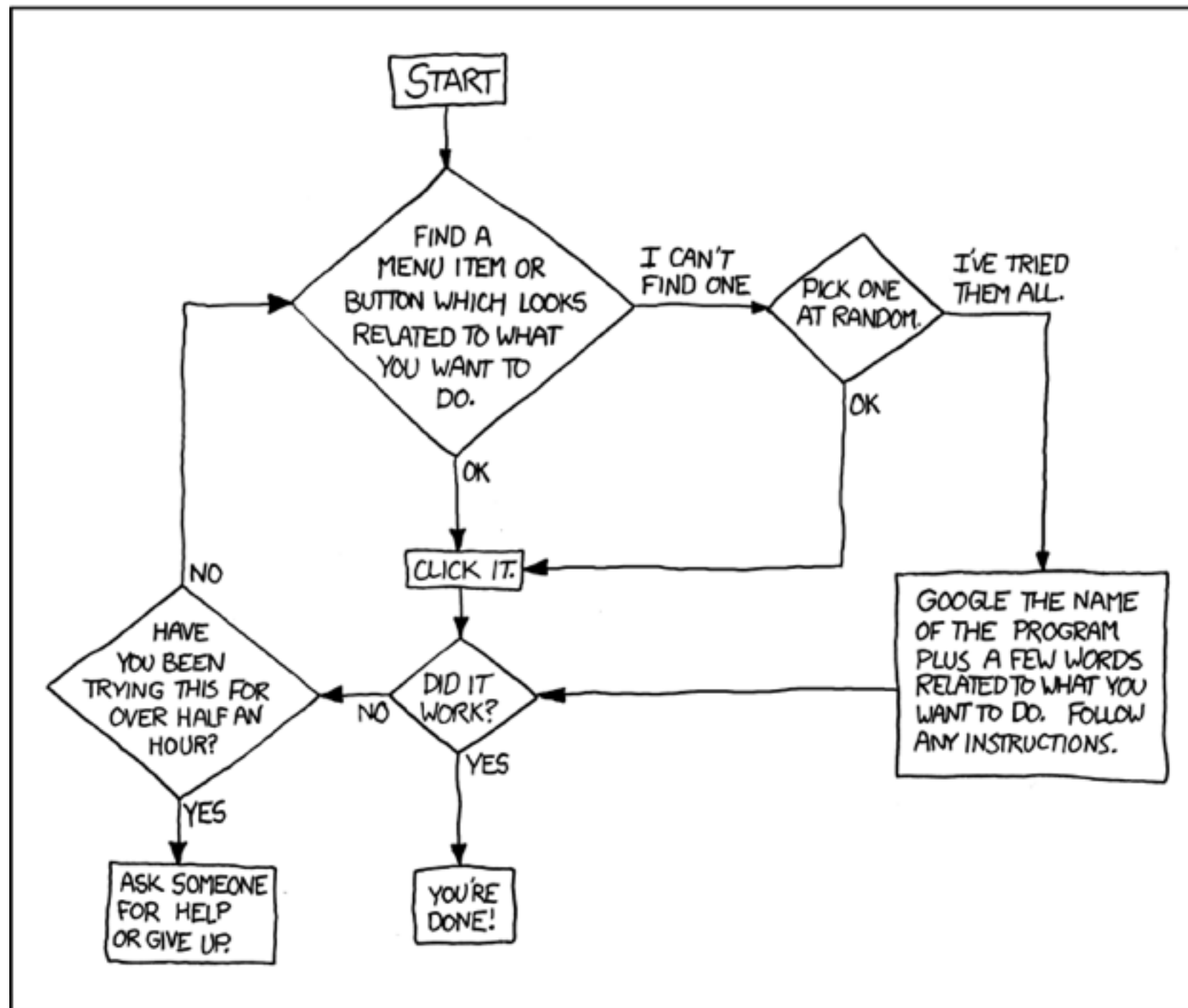
Many learners, which individually may be very weak but together can do well, a wisdom of crowds scenario.

This Week

- HW3 due next Thursday (10/22) at 11:59 pm Eastern. Start early!

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS,
AND OTHER "NOT COMPUTER PEOPLE."

WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY
PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:



PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN.
CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

Decision Trees

- Easy to understand
- Easy to visualize
- Can be used both for classification and for regression
- Can handle many types of predictors (continuous, categorical, etc.)
- Poor predictive accuracy

You can easily explain it to someone that doesn't understand statistics.

The kinds of trees you can come up with is extremely vast in terms of structure. It's naive to think that your model will come up with the single best tree. And, it's not clear that the one best tree is any good. Nature doesn't always work in a very 'flowchart' sort of way.

Ensemble Learning

- Combine many learners, each of which may on its own be weak
- Weighted average, weighted majority voting
- Wisdom of crowds?
- Several methods such as bagging, random forests, and boosting

If you're working with non-categorical/continuous then you do this ^

If you're handling categorical data then you do this ^

Suppose you're very indecisive, so whenever you want to watch a movie, you ask your friend Willow if she thinks you'll like it. In order to answer, Willow first needs to figure out what movies you like, so you give her a bunch of movies and tell her whether you liked each one or not (i.e., you give her a labeled training set). Then, when you ask her if she thinks you'll like movie X or not, she plays a 20 questions-like game with IMDB, asking questions like "Is X a romantic movie?", "Does Johnny Depp star in X?", and so on. She asks more informative questions first (i.e., she maximizes the information gain of each question), and gives you a yes/no answer at the end. Thus, Willow is a **decision tree for your movie preferences**.

But Willow is only human, so she doesn't always generalize your preferences very well (i.e., she overfits). In order to get more accurate recommendations, you'd like to ask a bunch of your friends, and watch movie X if most of them say they think you'll like it. That is, instead of asking only Willow, you want to ask Woody, Apple, and Cartman as well, and they vote on whether you'll like a movie (i.e., **you build an ensemble classifier**, aka a forest in this case).

Now you don't want each of your friends to do the same thing and give you the same answer, so you first give each of them slightly different data.... You don't change your love/hate decisions, you just say you love/hate some movies a little more or less (formally, **you give each of your friends a bootstrapped version of your original training data**). For example, whereas you told Willow that you liked Black Swan and Harry Potter and disliked Avatar, you tell Woody that you liked Black Swan so much you watched it twice, you disliked Avatar, and don't mention Harry Potter at all....

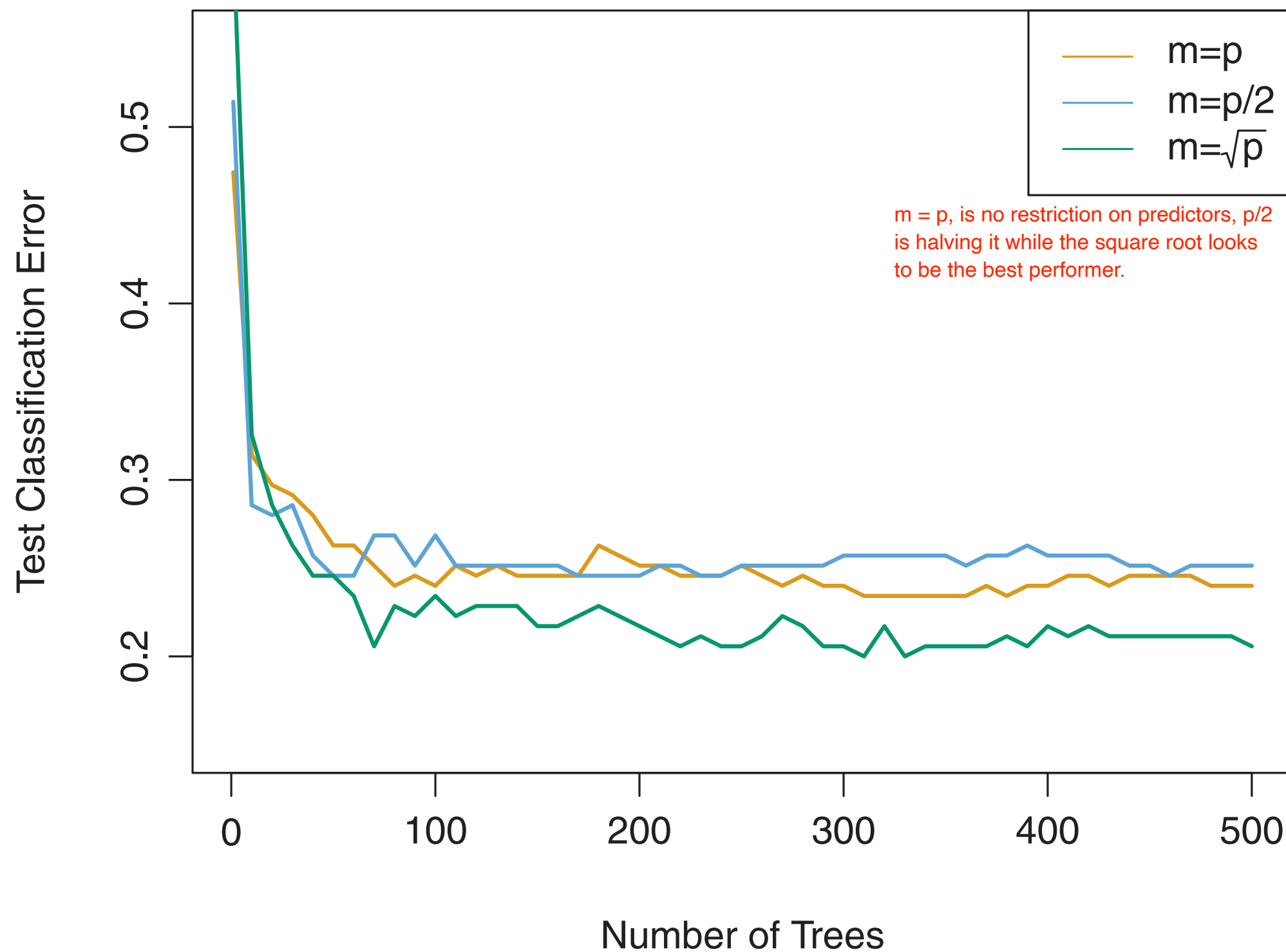
There's still one problem with your data, however. While you loved both Titanic and Inception, it wasn't because you like movies that star Leonardo DiCaprio. Maybe you liked both movies for other reasons. Thus, you don't want your friends to all base their recommendations on whether Leo is in a movie or not. So when each friend asks IMDB a question, only a random subset of the possible questions is allowed (i.e., **when you're building a decision tree, at each node you use some randomness in selecting the attribute to split on**, say by randomly selecting an attribute or by selecting an attribute from a random subset). This means your friends aren't allowed to ask whether Leonardo DiCaprio is in the movie whenever they want.... And so **your friends now form a random forest**.

Random Forests

- Extension of bagging of decision trees: generate bootstrap training samples, and a tree based on each, but also randomize set of predictors allowed each time a split is considered.
- Typically, # of random predictors chosen to be of order the square root of the total number of predictors.
- All trees are fully grown
- No pruning
- Two parameters
 - Number of trees
 - Number of features

So if you have 100 predictors, you randomly choose 10 of them. This would be bad if you're doing it once, but here you're doing it many times.

M = the number of features you're allowed to have/make decisions on.



There's no reason to think that the positive biases will outweigh the negative ones.

This question of trying to figure out whether a book is good or bad by looking at it carefully or by taking the reports of a lot of people who looked at it carelessly is like this famous old problem: Nobody was permitted to see the Emperor of China, and the question was, What is the length of the Emperor of China's nose?

To find out, you go all over the country asking people what they think the length of the Emperor of China's nose is, and you average it. And that would be very "accurate" because you averaged so many people. But it's no way to find anything out; when you have a very wide range of people who contribute without looking carefully at it, you don't improve your knowledge of the situation by averaging.

If you're performing random forests, it's not like the emperor's nose because you're capturing something, based on the data, versus no data in this example.

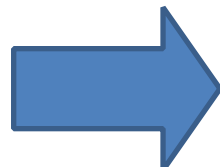
- Richard Feynman

In ensemble methods, while you may be performing simple calculations, that may not be very good at the singular level, still captures something useful in the data but by aggregating them you're creating something positive. The models picking out negative association, they'll get cancelled out by the positive associations.

Boosting

- Also ensemble method like Bagging
- But:
 - weak learners evolve over time
 - votes are weighted
- Better than Bagging for many applications

Bagging doesn't evolve over time like boosting.

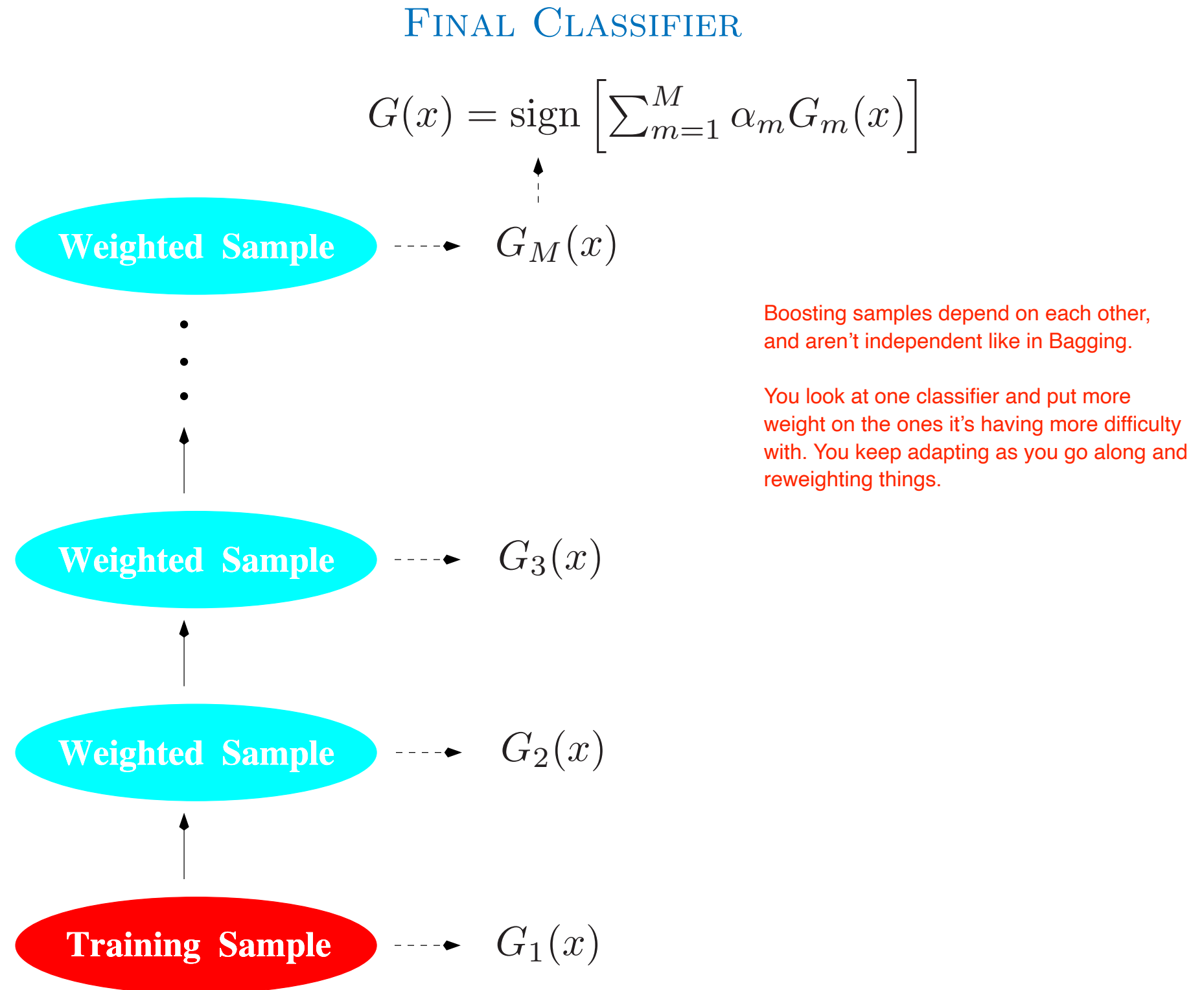
 Very popular method

Boosting

“Boosting is one of the most powerful learning ideas introduced in the last twenty years.”

Hastie-Tibshirani-Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer (2009)

Boosting Schematic



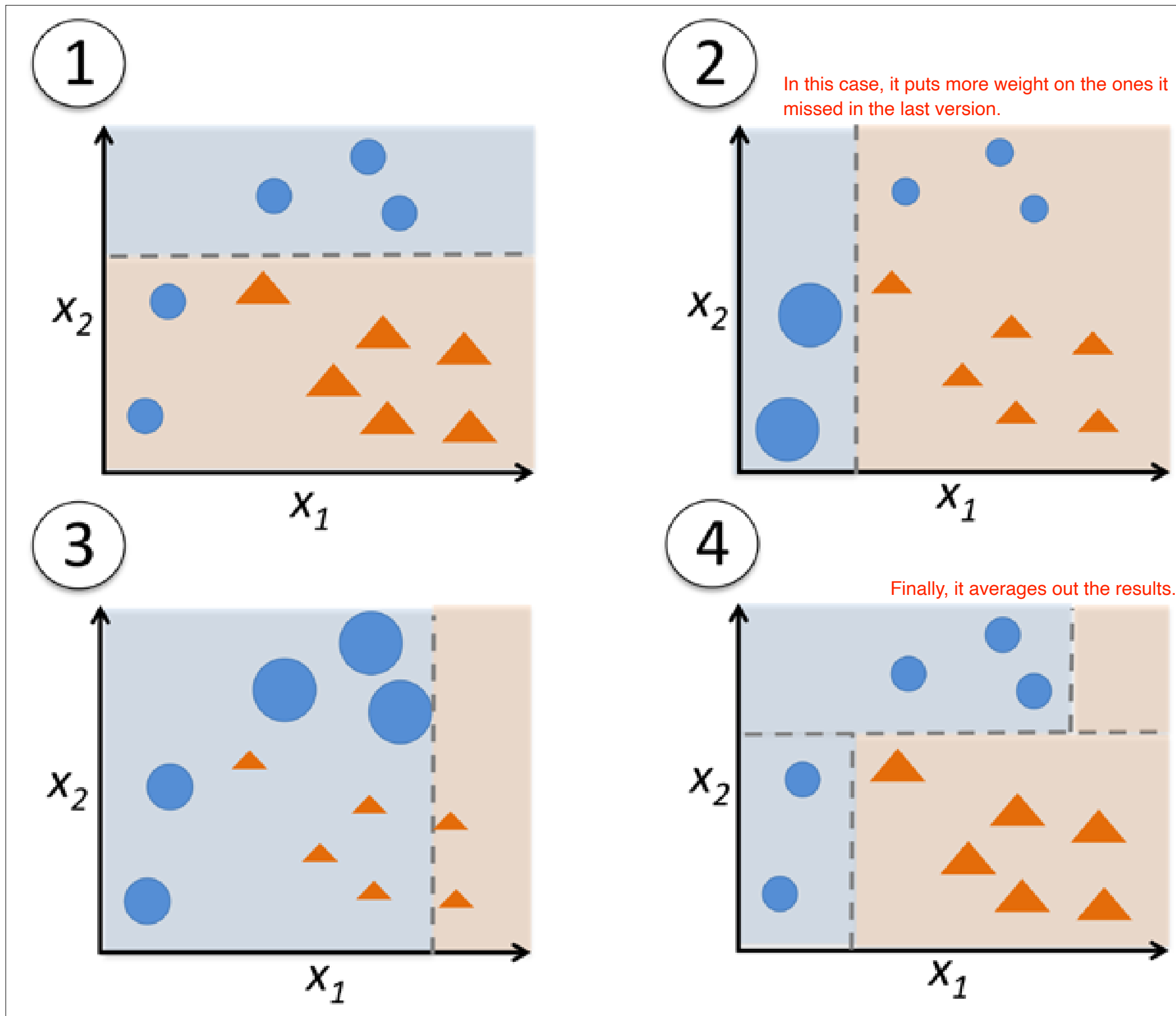
Tuning Parameters for Boosting

- number of trees
- number of splits in each tree (often *stumps* work well)
- parameters controlling how weights evolve

Stumps are one split, one of the simplests trees actually.

A stump will just look at circles vs triangles... and that's it.

Size is meant to imply 'weight'.



In this example, it then weights more the ones it missed from the previous model.

Finally, it averages out the results.

AdaBoost

Algorithm 10.1 *AdaBoost.M1*.

You start with equal weights.

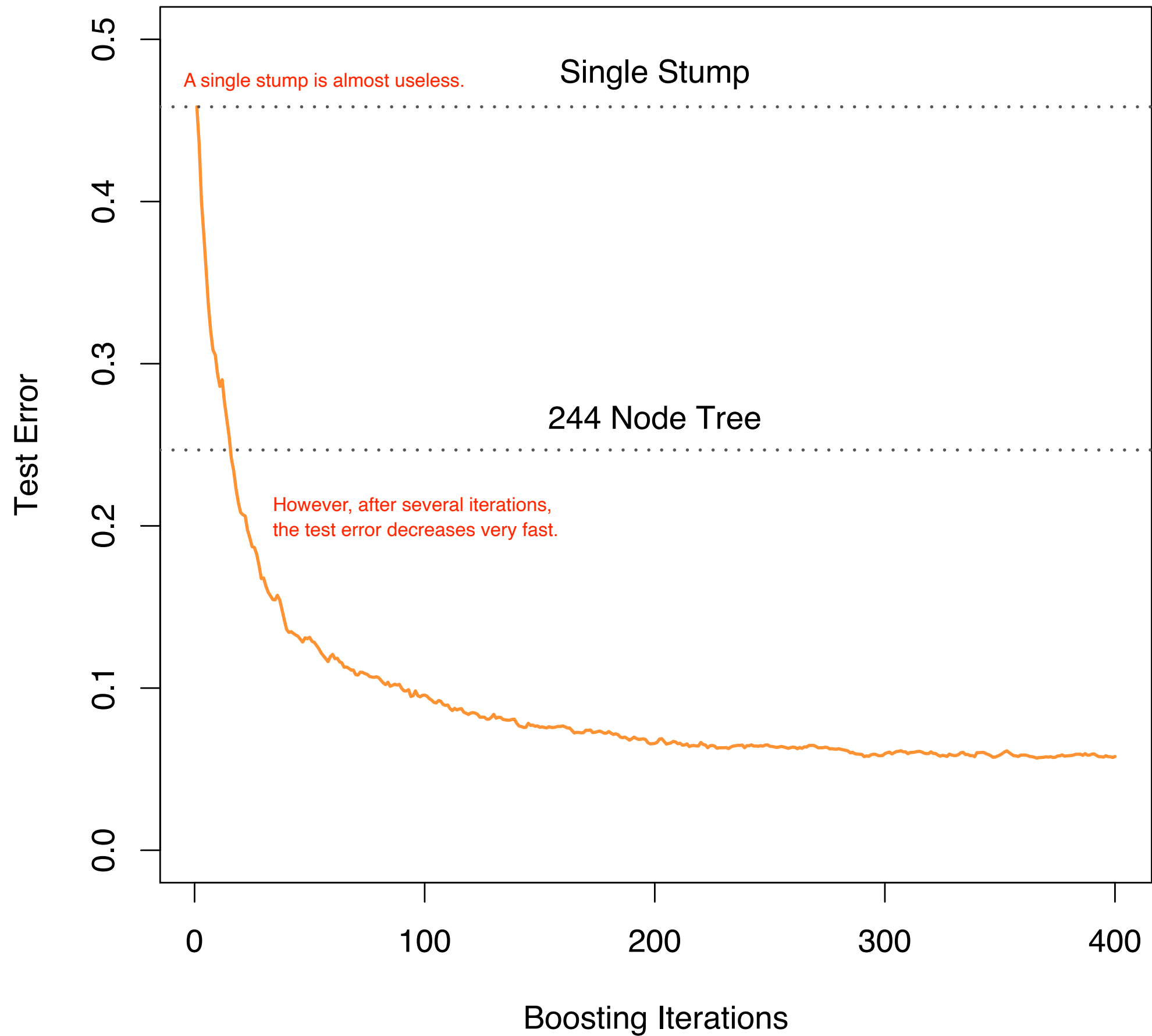
1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \dots, N$.
2. For $m = 1$ to M :
 - (a) Fit a classifier $G_m(x)$ to the training data using weights w_i .
 - (b) Compute

$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$

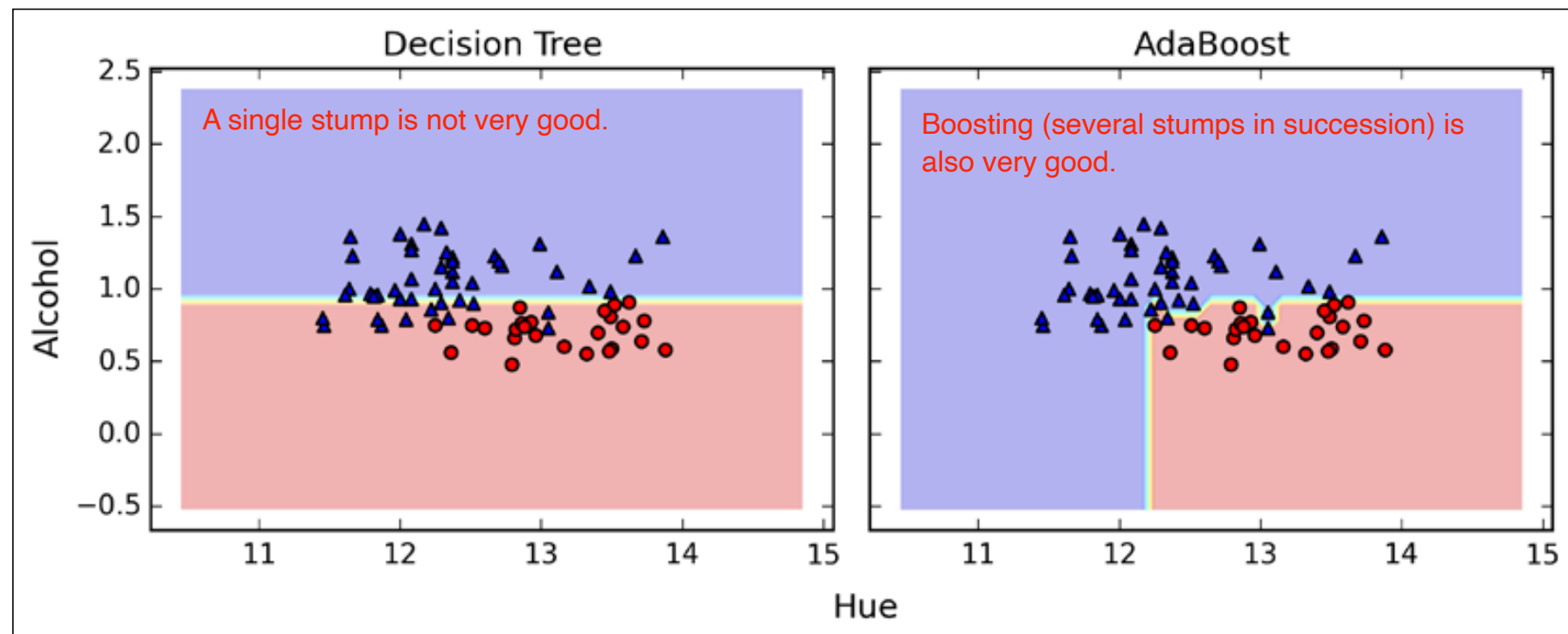
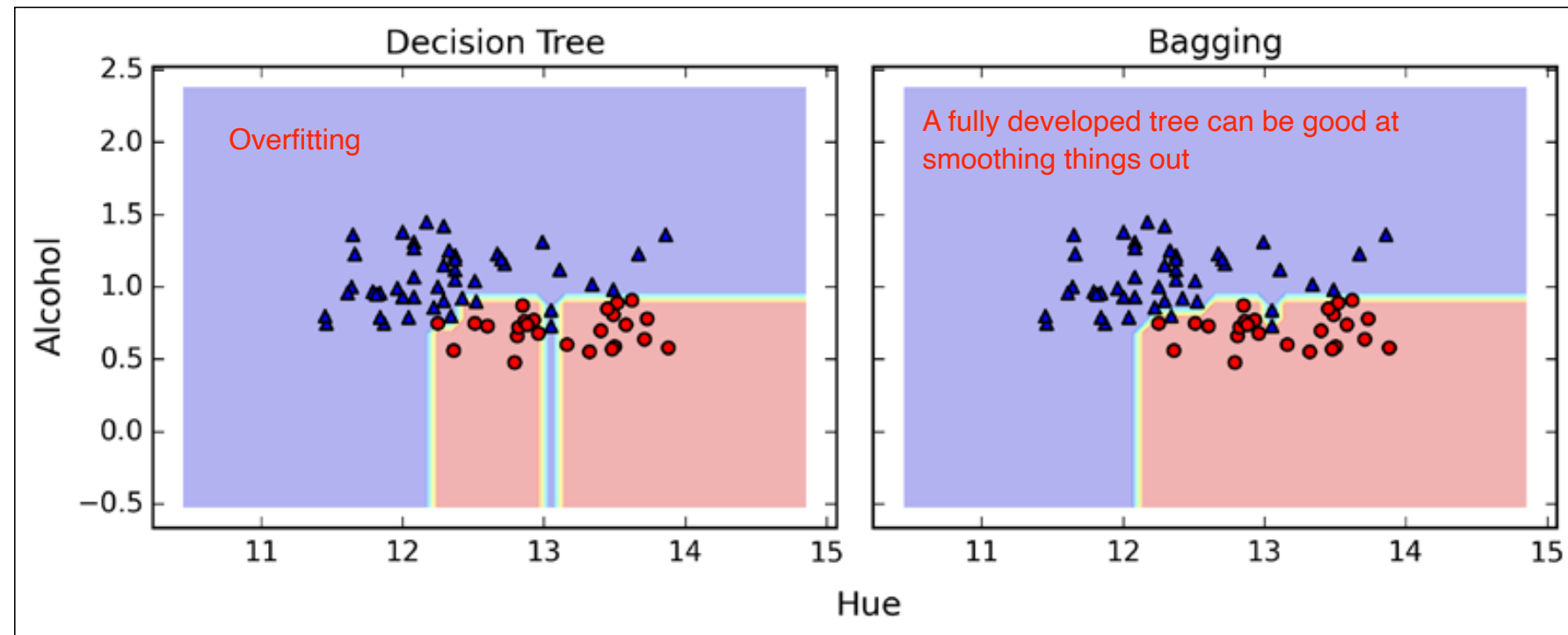
It resets the weights moving forward. If the result is 0, then the weight is unadjusted. If the value results in 1, that means it's misclassified and will be reweighted in the next iteration.

- (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
 - (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \dots, N$.
3. Output $G(x) = \text{sign} \left[\sum_{m=1}^M \alpha_m G_m(x) \right]$.

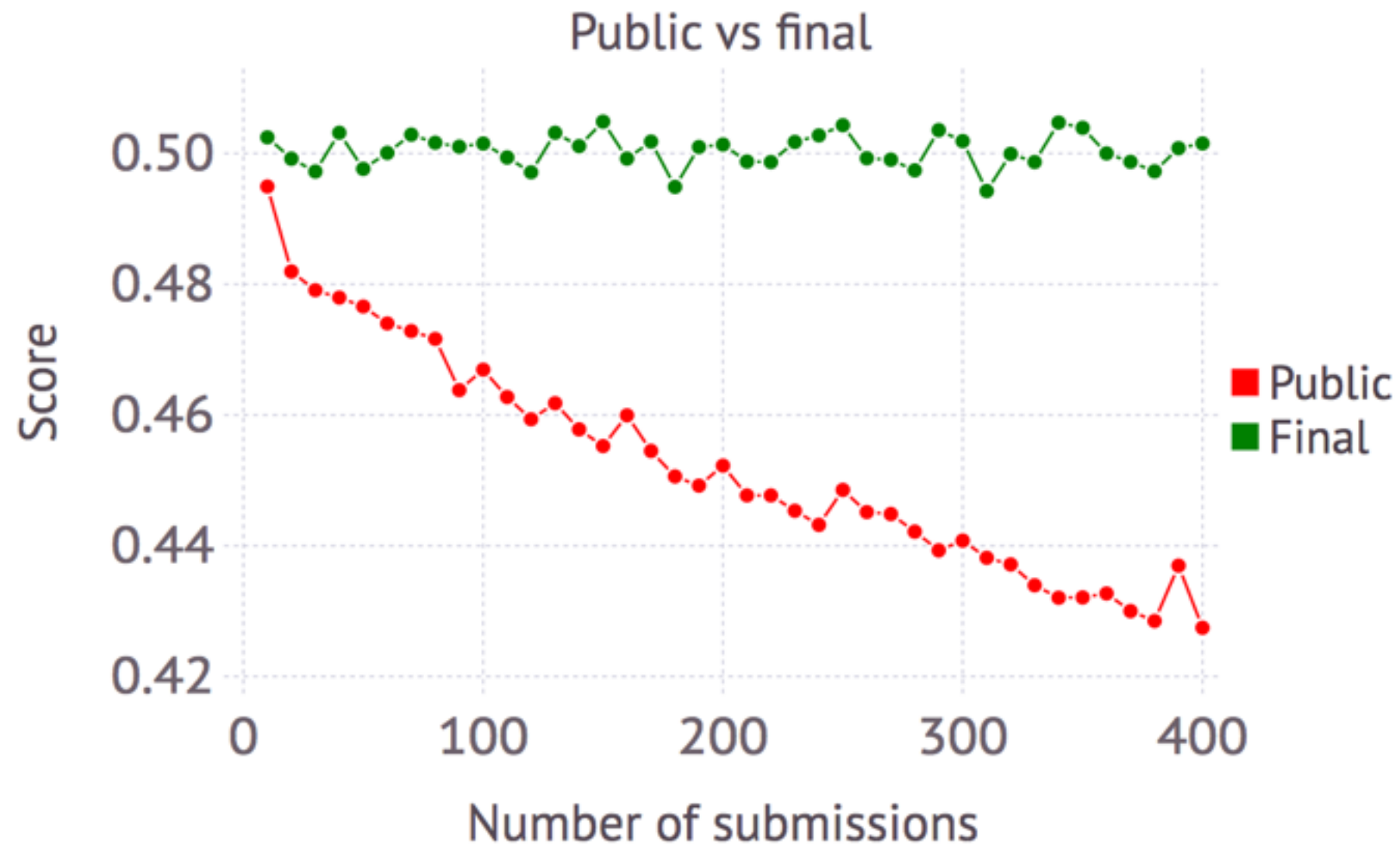
Your overall classifier is the weighted majority voting.



Wine Example



“Competing in a data science contest without reading the data”
<http://blog.mrtz.org/2015/03/09/competition.html>



“Competing in a data science contest without reading the data”
<http://blog.mrtz.org/2015/03/09/competition.html>

Algorithm (Wacky Boosting):

1. Choose $y_1, \dots, y_k \in \{0, 1\}^N$ uniformly at random.
2. Let $I = \{i \in [k]: s_H(y_i) < 0.5\}$. How well did that particular classification did.
3. Output $\hat{y} = \text{majority}\{y_i: i \in I\}$, where the majority is component-wise.