BEYOND DECISION TREES:

BAGGING AND BOOSTING METHODS

Viviana Acquaviva, CUNY

DECISION TREES RECAP

GOOD

SIMPLE TO UNDERSTAND

FAST

ACCURATE (TEND TO HAVE LOW BIAS)

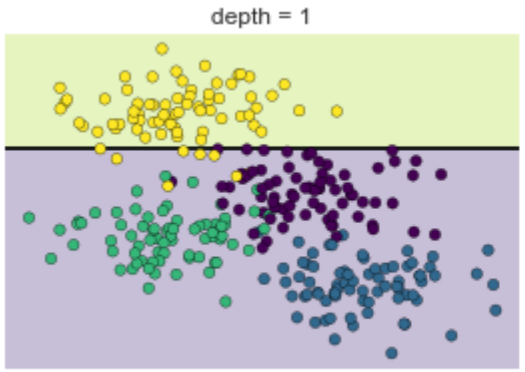
BAD

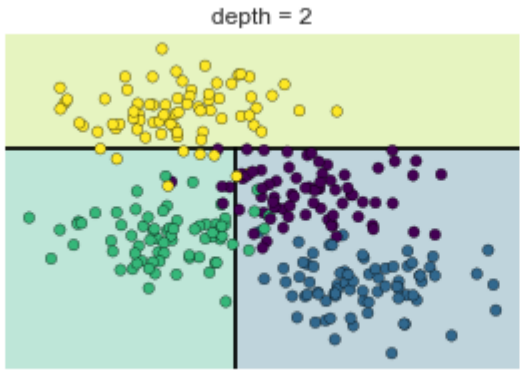
TEND TO HAVE HIGH VARIANCE!

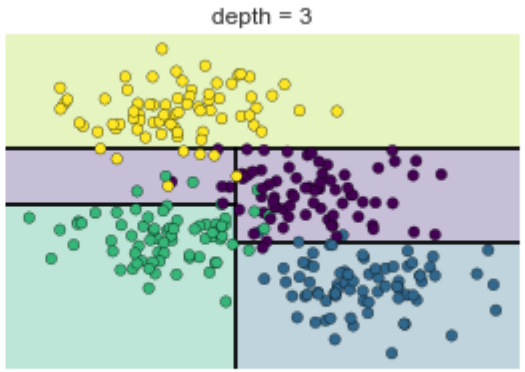
WHY?

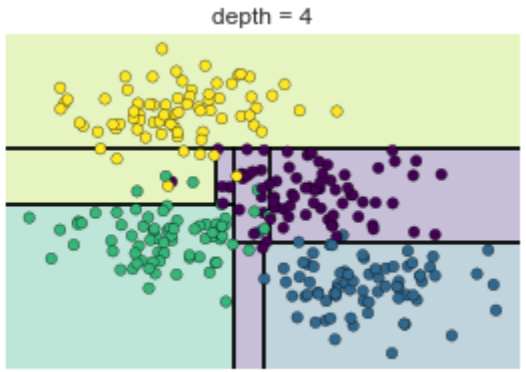
IT is VERY EASY to overfit (keep splitting more than necessary), which makes for poor generalization.

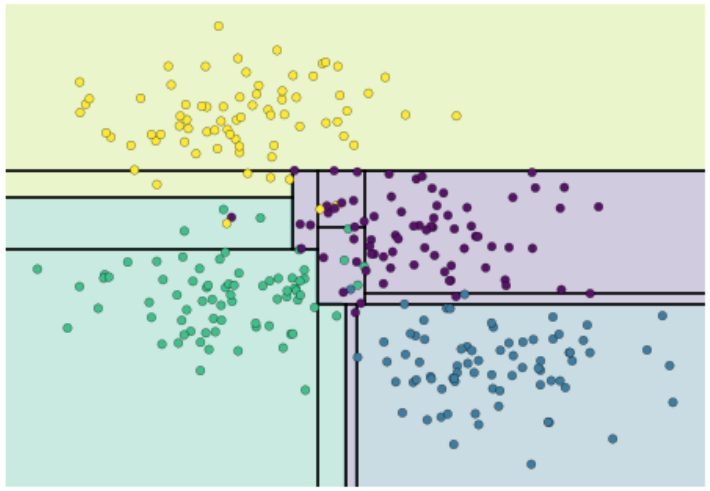
Example (from VanDerPlas)











depth = 5

Traditional approach:
"pruning" trees
(get rid of small leaves)

How can we improve?
Consider bias-variance decomposition

Theorem. For the squared error loss, the bias-variance decomposition of the expected generalization error at $X = \mathbf{x}$ is

$$E_L\{Err(\varphi_L(\mathbf{x}))\} = \text{noise}(\mathbf{x}) + \text{bias}^2(\mathbf{x}) + \text{var}(\mathbf{x})$$

If we are willing to take a hit in bias,
we can reduce variance by
combining RANDOMIZED trees,
and still have an improved model
(= lower test scores).
This is the core of ensemble/bagging methods.

Slide adapted from Gilles Louppes

Furthermore….

$$E_L\{Err\,(\psi_{L,\theta_1,\ldots,\theta_M}(\mathbf{x}))\} = \text{noise}(\mathbf{x}) + \text{bias}^2(\mathbf{x}) + \text{var}(\mathbf{x}),$$

where

$$\text{noise}(\mathbf{x}) = Err\,(\varphi_B(\mathbf{x})),$$

$$\text{bias}^2(\mathbf{x}) = (\varphi_B(\mathbf{x}) - E_{L,\theta}\{\varphi_{L,\theta}(\mathbf{x})\})^2,$$

$$\text{var}(\mathbf{x}) = \rho(\mathbf{x})\,\sigma^2_{L,\theta}(\mathbf{x}) + \frac{1-\rho(\mathbf{x})}{M}\,\sigma^2_{L,\theta}(\mathbf{x}).$$

M is the number of trees, $\rho(\mathbf{x})$ is the Pearson correlation coefficient between the predictions of two randomized trees built on the same learning set.

Slide adapted from Gilles Louppes

Insights on the generalization error of random forests

- Bias :Identical to the bias of a single randomized tree.
- Variance : var **(x) =** ρ**(x)** $\sigma^2_{L,\theta}$**(x) +** $\frac{1-\rho(x)}{M}\sigma^2_{L,\theta}$**(x)**
  As $M \to \infty$, var **(x)** $\to$ ρ**(x)** $\sigma^2_{L,\theta}$**(x)**
  - The stronger the randomization, ρ**(x)** $\to 0$, var**(x)** $\to 0$
  - The weaker the randomization, ρ**(x)** $\to 1$, var**(x)** $\to \sigma^2_{L,\theta}$ **(x)**

Bias-variance trade-off take-home point. Randomization **increases bias** but makes it possible to **reduce the variance** of the corresponding ensemble model. The crux of the problem is to **find the right trade-off** (reducing variance does not guarantee higher test scores!)

Slide adapted from Gilles Louppes

Randomization techniques: how to build different trees

- Bootstrap samples (with replacement) — selects slightly different bootstrap samples
- Random selection of $K \leq p$ split variables — choose best split not among all features, but among some random ones

} Random Forests

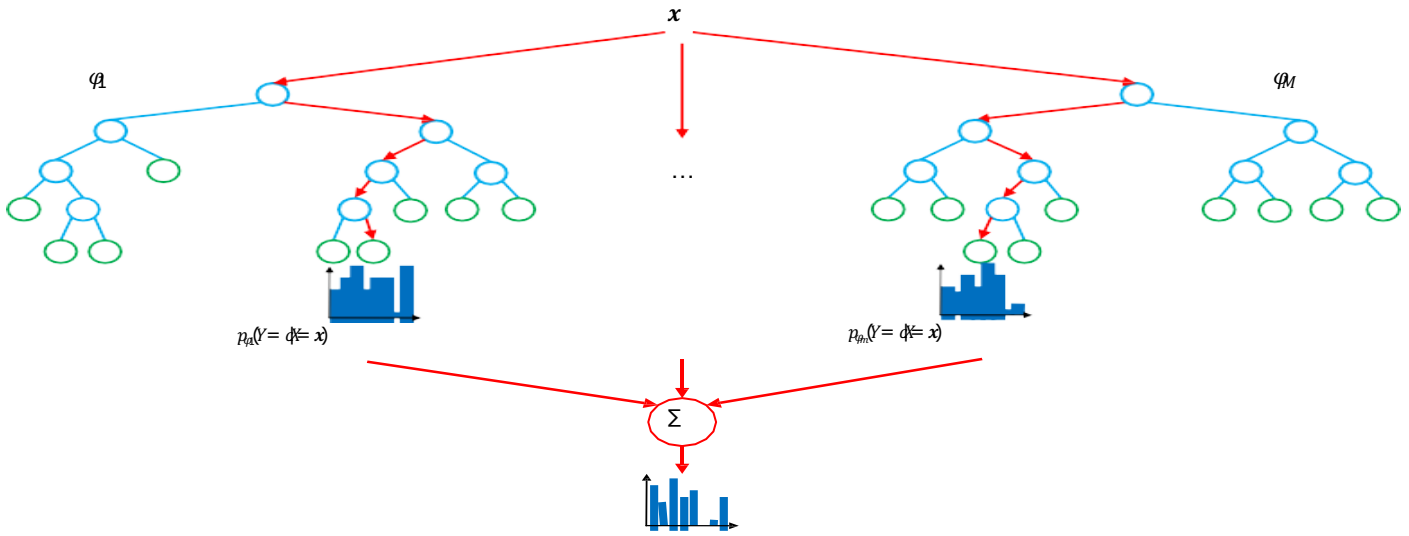
- Random selection of the threshold — threshold value for split is chosen at random

} Extra Random Trees

These become parameters of the model, as we will see in the coding part.

Figure from
Gilles Louppes

How to combine trees?



In scikit: performance of random forest (or extra random trees)
is the **average of performance over all the trees**
and the final prediction (class, or number)
is the **average of all predictions**

Boosted ensemble methods

(Gradient Boosted Trees, AdaBoost, XGB, LightGBM…)

Bagging methods like RF, ERT work by building full trees in parallel and then averaging the predictions.

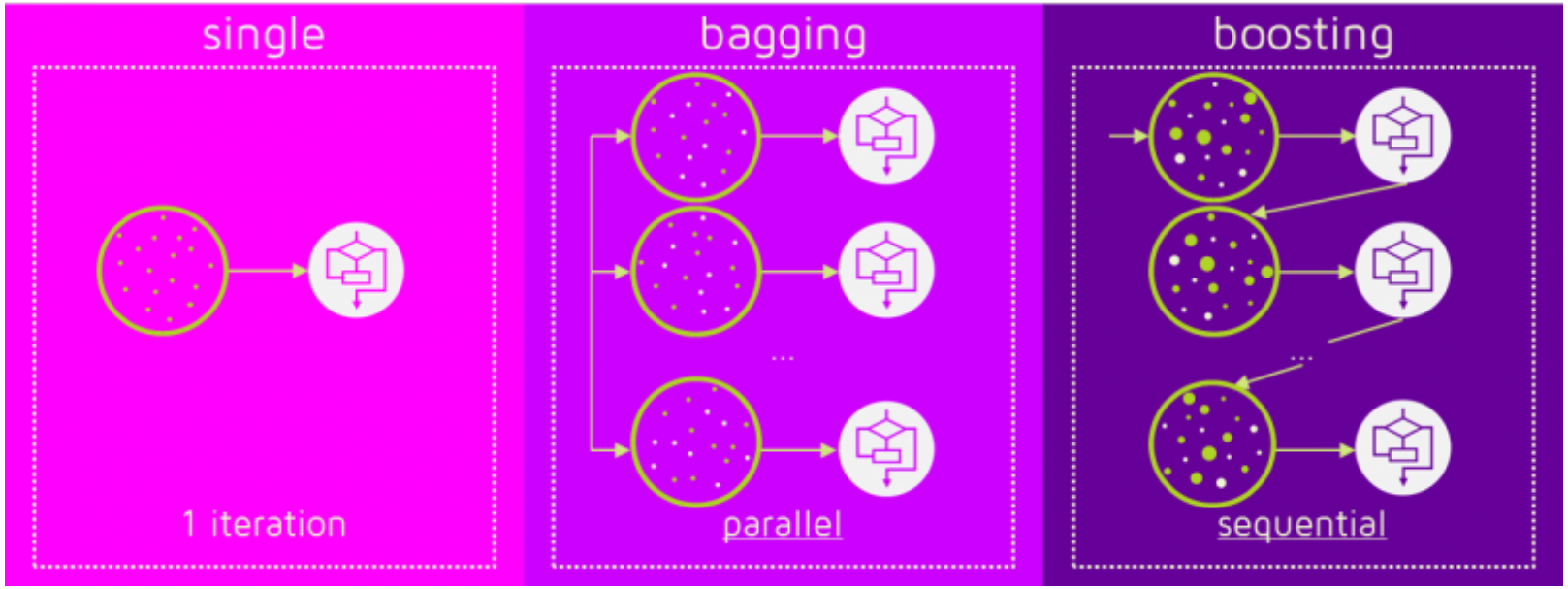
Boosting methods work sequentially: a small tree (sometimes called a stump) is created and used to make predictions, then the next step focuses on getting the problematic examples right and this procedure is applied iteratively.

Amazing resource on GBTs:

https://www.youtube.com/watch?v=sRktKszFmSk

Tree vs Random Forest vs Boosted Trees

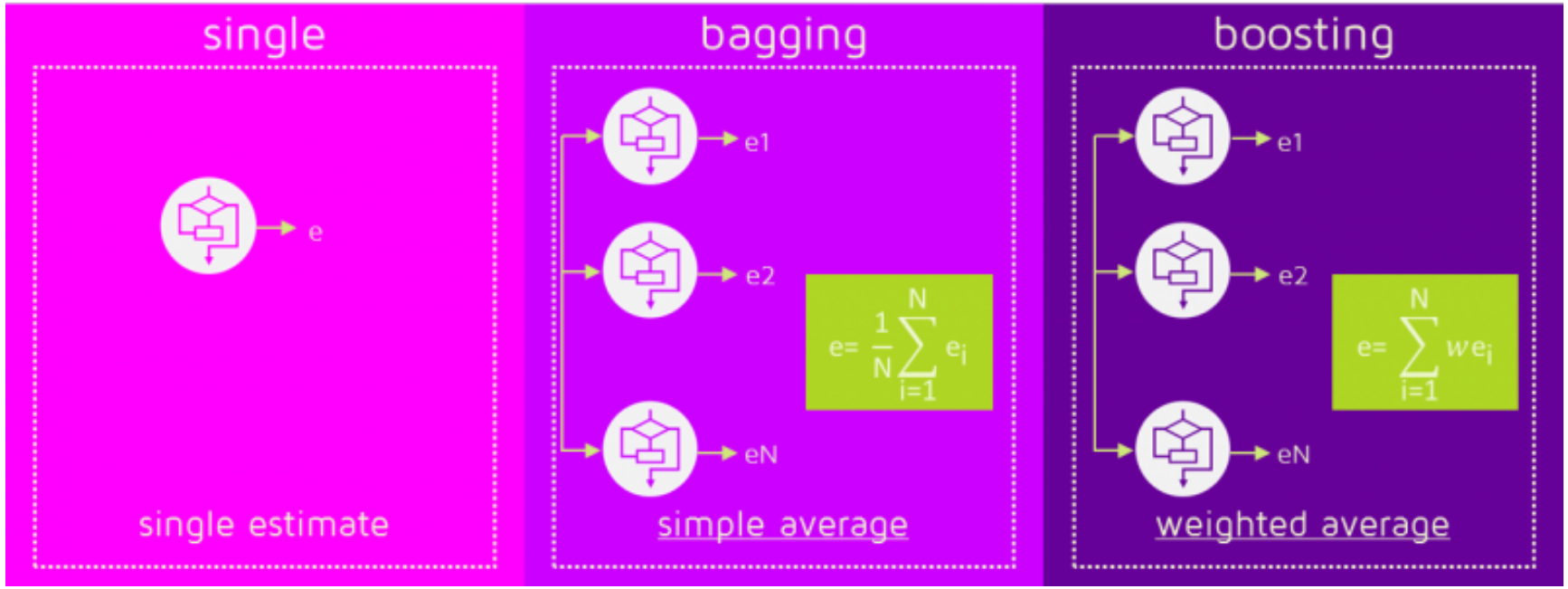
Picture: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/



In Boosting algorithms, after each training step,
the weights of the objects in the learning set are redistributed.
Misclassified data are assigned higher weights to emphasize the most difficult cases.

Ensembling

Picture: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/



In Boosting algorithms**, the weight of different learners (stumps) in the final classification depends on their performance.**
(unlike bagging algorithms in which the final prediction is a simple average)

One example: Gradient Boosted Trees

Simple idea:

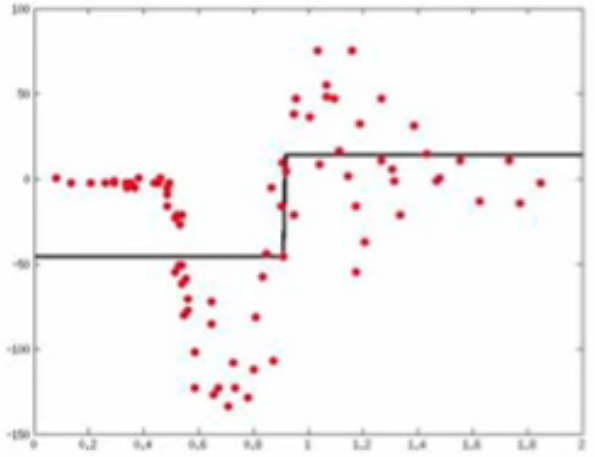
1. Start with a weak learner (1-split tree) and fit a model F to data y: $F_1(x) = y$

2. Calculate residuals (i.e., where model is failing) $F_1(x) – y$ and make model h(x)

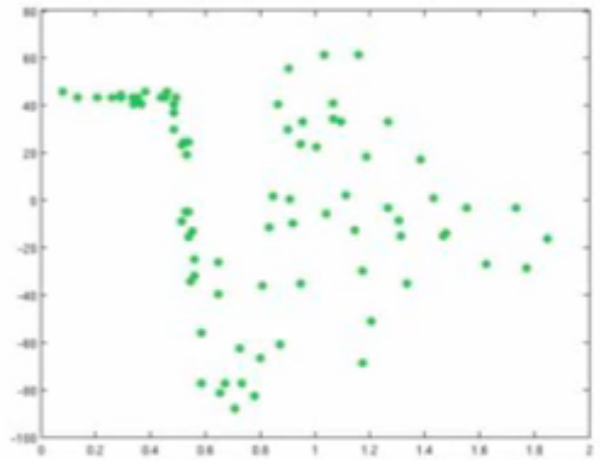
3. Make new model: $F_2(x) = F_1(x) + h(x)$

4. Repeat from 1, until convergence

Learn a simple predictor…



Then try to correct its errors



One example: Gradient Boosted Trees

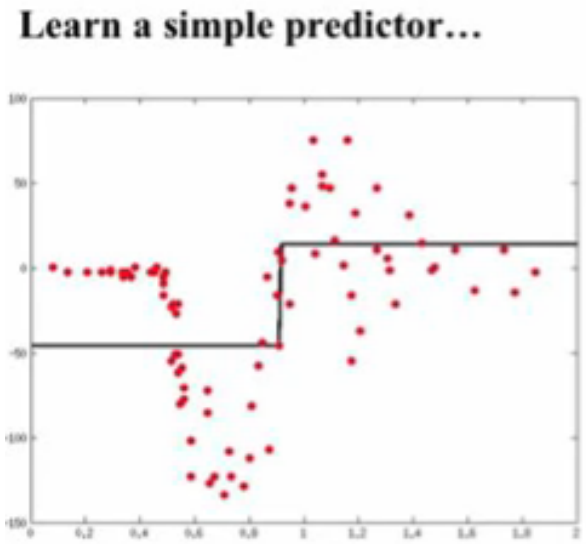
Simple idea:

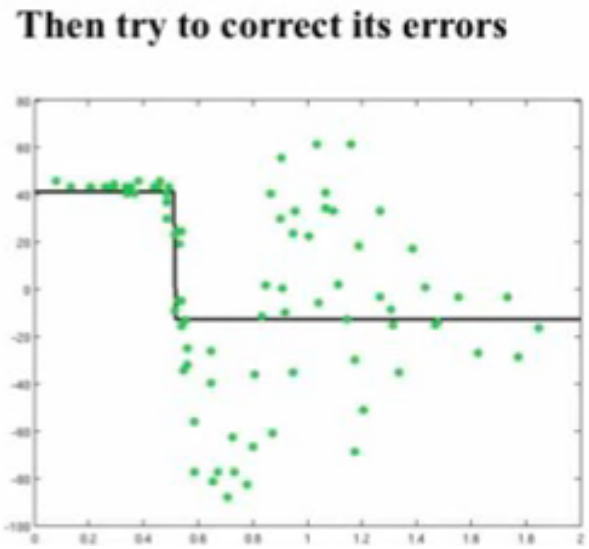
1. Start with a weak learner (1-split tree) and fit a model F to data y: $F_1(x) = y$

2. Calculate residuals (i.e., where model is failing) $F_1(x) – y$ and make model h(x)

3. Make new model: $F_2(x) = F_1(x) + h(x)$

4. Repeat from 1, until convergence





One example: Gradient Boosted Trees

Simple idea:

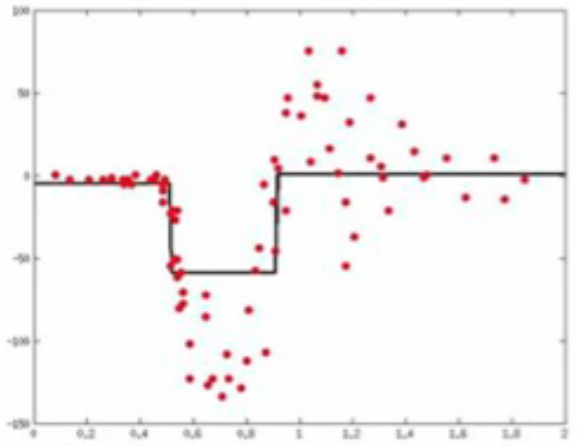
1. Start with a weak learner (1-split tree) and fit a model F to data y: $F_1(x) = y$

2. Calculate residuals (i.e., where model is failing) $F_1(x) – y$ and make model h(x)

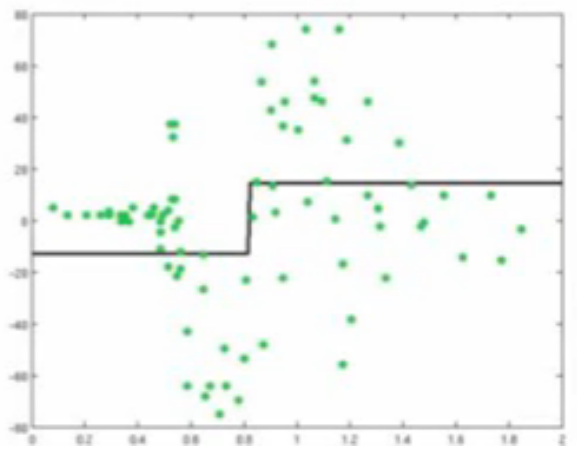
3. Make new model: $F_2(x) = F_1(x) + h(x)$

4. Repeat from 1, until convergence

Combining gives a better predictor…



Can try to correct its errors also, & repea



One example: Gradient Boosted Trees

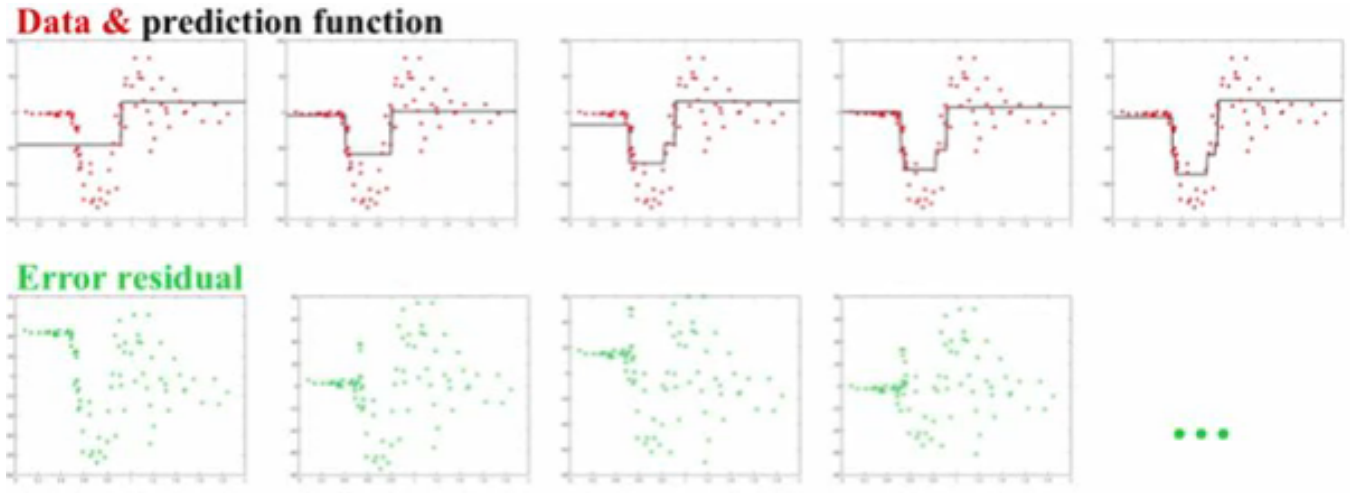
Simple idea:

1. Start with a weak learner (1-split tree) and fit a model F to data y: $F_1(x) = y$

2. Calculate residuals (i.e., where model is failing) $F_1(x) – y$ and make model h(x)

3. Make new model: $F_2(x) = F_1(x) + h(x)$

4. Repeat from 1, until convergence



Let's put the "Gradient" in GBM

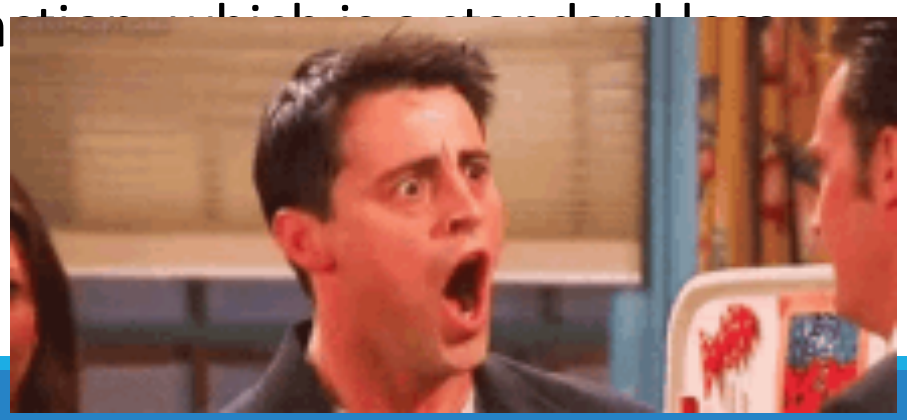
The general idea of "Gradient Descent" is that the most efficient way to minimize a function is to move towards its (negative) gradient.

Used in many minimization problems.

If we want to minimize the loss function, we can fit a model to its negative gradient components.

In fact, fitting the residuals works because the residuals of a function are the derivative of the mean square error fun
function.

ML Limericks @MLimericks · Mar 14
Those were the days, my good friend
We wrote gradients down by hand
They weren't too complex
Our losses were convex
You're too young, you won't understand



Let's put the "Gradient" in GBM

Modified idea to work with any differentiable loss function L:

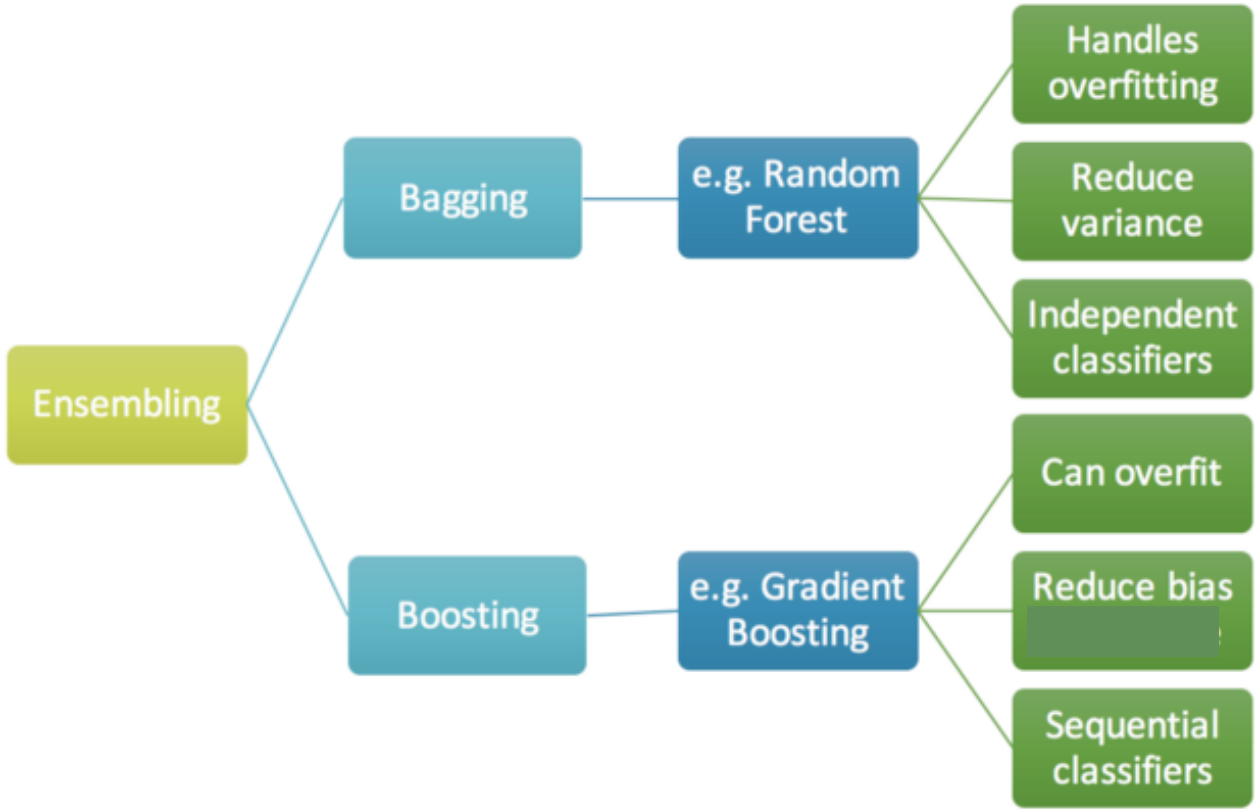
1. Initialize F to median of y
2. Calculate pseudo-residuals = - dL/dF
3. Fit model to pseudo - residuals, h(x)
4. Make new model: $F_2(x) = F_1(x) + \alpha^* h(x)$ (α = learning rate)
5. Repeat from 2, until convergence

The learning rate is a parameter of boosted models (shrinkage).

Small α values require larger # of iterations but often gives better predictions

α can be chosen on a step-by-step basis and becomes the weight of each predictor in the final ensemble

Pros and Cons Summary



https://nbviewer.jupyter.org/github/groverpr/Machine-Learning/blob/master/notebooks/01_Gradient_Boosting_Scratch.ipynb

Variable importance

A really cool feature of ensemble methods is that because of the way randomization works, it is easy to build an understanding of

which features carry the majority of the relevant information.

Typically, variables are ranked according to two criteria:

The mean decrease of impurity (MDI) : summing total impurity reductions at all tree nodes where the variable appears (Breiman et al., 1984) ;

The mean decrease of accuracy (MDA) : measuring accuracy reduction on out-of-bag samples when the values of the variable are randomly permuted (Breiman, 2001).

Variable importance

A variable is most important if it leads to a large Mean Decrease of Impurity or Mean Decrease of Accuracy (in other words, it has a large impact on the performance).

Feature ranking can be used to gain understanding of the data and to reduce the size of a data set (for example, one can feature-rank the original data set and picked the first x out of y features, using information loss as a guidance).

Cool example:

Here!

What can you use Bagging/Boosting methods for?

Both classification and regression problems!

You can also always turn a classification problem into a regression problem by predicting a probability that on object belongs to a certain class

They are an all-purpose, generally fast and accurate sets of algorithms

Without going neural, usually the best method out there

(see this: https://lavanya.ai/2019/06/08/kaggle-leaderboard/)

References / Further Reading

On Random Forests:

Gilles Louppes Ph. D. thesis (Random Forests: from Theory to Practice)

arXiv preprint arXiv:1407.7502

scikit-learn handbook (various things, but see this for a graphical example of bias/variance decomposition: https://scikit-learn.org/stable/auto_examples/ensemble/plot_bias_variance.html#sphx-glr-auto-examples-ensemble-plot-bias-variance-py)

Jake VanDerPlas book, “In depth: Random Forests” (see course page)

On Boosted Trees:

https://www.youtube.com/watch?v=sRktKszFmSk

http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/

https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

(https://nbviewer.jupyter.org/github/groverpr/Machine-Learning/blob/master/notebooks/01_Gradient_Boosting_Scratch.ipynb) not sure of original source…