NUS National University of Singapore

School of Computing

Ensemble Learning

Associate Professor
HUANG, Ke-Wei

Agenda

"Machine Learning With R" Chapter 11.

1. Variance-Bias Decomposition
2. Bagging
3. Random Forest
4. Stacking
5. Boosting (most important)
   - AdaBoost
   - GBM, XGBoost, and LightGBM (next lecture)

Variance-Bias Decomposition

- This decomposition is an important theoretical concept that you should bear in mind. As a novice, you may not feel the importance. But when you become an experienced data scientists, improving prediction performance depends a lot on this theory.

- Prediction error of any model can be decomposed into 3 parts:
  Variance + Bias + Noise

- If your model performance is not good, you can also think along this 3 aspects.

Variance-Bias Decomposition

1. Noise: even if your model is perfect, there is still error, which is defined as noise here.

Several comments:

1. If you already include all useful features and the model is perfect in capturing how all those x may affect y, then there is no way you can further improve accuracy by reducing noise.
2. In practice, it could be you do not have all useful features: (1) collecting/buying more data? (2) features engineering?

Variance-Bias Decomposition

2. Bias: your model is not perfect. Even if you have all useful features, the predicted value is different from the actual value because your model’s functional problem. In other words, Bias is the error from your model.
   - We try all possible algorithms and parameters tuning. The best combination of model + parameters is the best model you have based on your validation procedure.
   - Roughly, the state of art in practice is: CNN/RNN-LSTM or GBM (XGBoost or LightGBM)
   - This may not be true when your dataset is small in rows or columns, like those in assignments.

Variance-Bias Decomposition

3. Variance: for many algorithms, if you re-run the same code, the predicted output and performance could be different **because the training dataset is different**.

- This is Bagging’s focus to tackle. Since your model is built on one training set, your model may not be good because of some special examples in that training set, especially if the algorithm is sensitive to small changes in datasets ⇔ specifically, Decision Trees and KNN
- There is a fundamental tradeoff between variance and bias.

Variance-Bias Decomposition

As in Chapter 2, if we assume that $Y = f(X) + \varepsilon$ where $\mathrm{E}(\varepsilon) = 0$ and $\mathrm{Var}(\varepsilon) = \sigma_\varepsilon^2$, we can derive an expression for the expected prediction error of a regression fit $\hat{f}(X)$ at an input point $X = x_0$, using squared-error loss:

$$\begin{aligned}
\mathrm{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] \\
&= \sigma_\varepsilon^2 + [\mathrm{E}\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - \mathrm{E}\hat{f}(x_0)]^2 \\
&= \sigma_\varepsilon^2 + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\hat{f}(x_0)) \\
&= \text{Irreducible Error} + \mathrm{Bias}^2 + \text{Variance}. \qquad (7.9)
\end{aligned}$$

The first term is the variance of the target around its true mean $f(x_0)$, and cannot be avoided no matter how well we estimate $f(x_0)$, unless $\sigma_\varepsilon^2 = 0$. The second term is the squared bias, the amount by which the average of our estimate differs from the true mean; the last term is the variance; the expected squared deviation of $\hat{f}(x_0)$ around its mean. Typically the more complex we make the model $\hat{f}$, the lower the (squared) bias but the higher the variance.

Overview

- Ensemble learning means those methods that can improve prediction performance without introducing a new learning algorithm, but by combining results from one or few existing algorithms.
- The key point is we have learned quite a number of methods and you observe unstable results even from only one method. Can we make better predictions?
- Are two heads better than one? (can 3 normal people outperform a wise man?) If so, under what conditions?
- Is democracy (majority voting by everyone) really better than dictatorship (decision by one)? If so, under what conditions?

1. Bagging Overview

- Full name is bootstrap aggregating = Bagging
- Invented in 1994 by Leo Breiman.
- This algorithm can be applied to all basic algorithms we have covered so far.
- **The idea is to use bootrapping (see foonotes) on your training dataset to create *N* training datasets. On each of the *N* training dataset, we build a classifier by the standard algorithm (especially decision tree).**
- The idea is to reduce the "variance" of your predictive model without hurting/increasing "bias". If your predictive model is very sensitive to the training dataset, then this method works well.
- The most famous case is decision tree.

Bagging Overview

- For example, you use bootstrapping to create *N* training datasets.
- You apply C50 to build *N* trees.
- Now you have *N* trees… what is next?
  - Categorical DV=> Use *N* trees with majority voting to classify a new test record during the prediction stage. For example if Tree 1 and 2 both vote YES and Tree 3 votes NO, then 2/3 probability "YES" is the prediction. For classification, "YES" is the final prediction.
  - Numeric DV => we just use average of predicted numbers.
- **Both theory and empirical experiments show that bagging helps decision tree a lot!**
- **But, "the number of trees" is one new parameter and your performance improvement will saturate.**

Bagging Algorithm

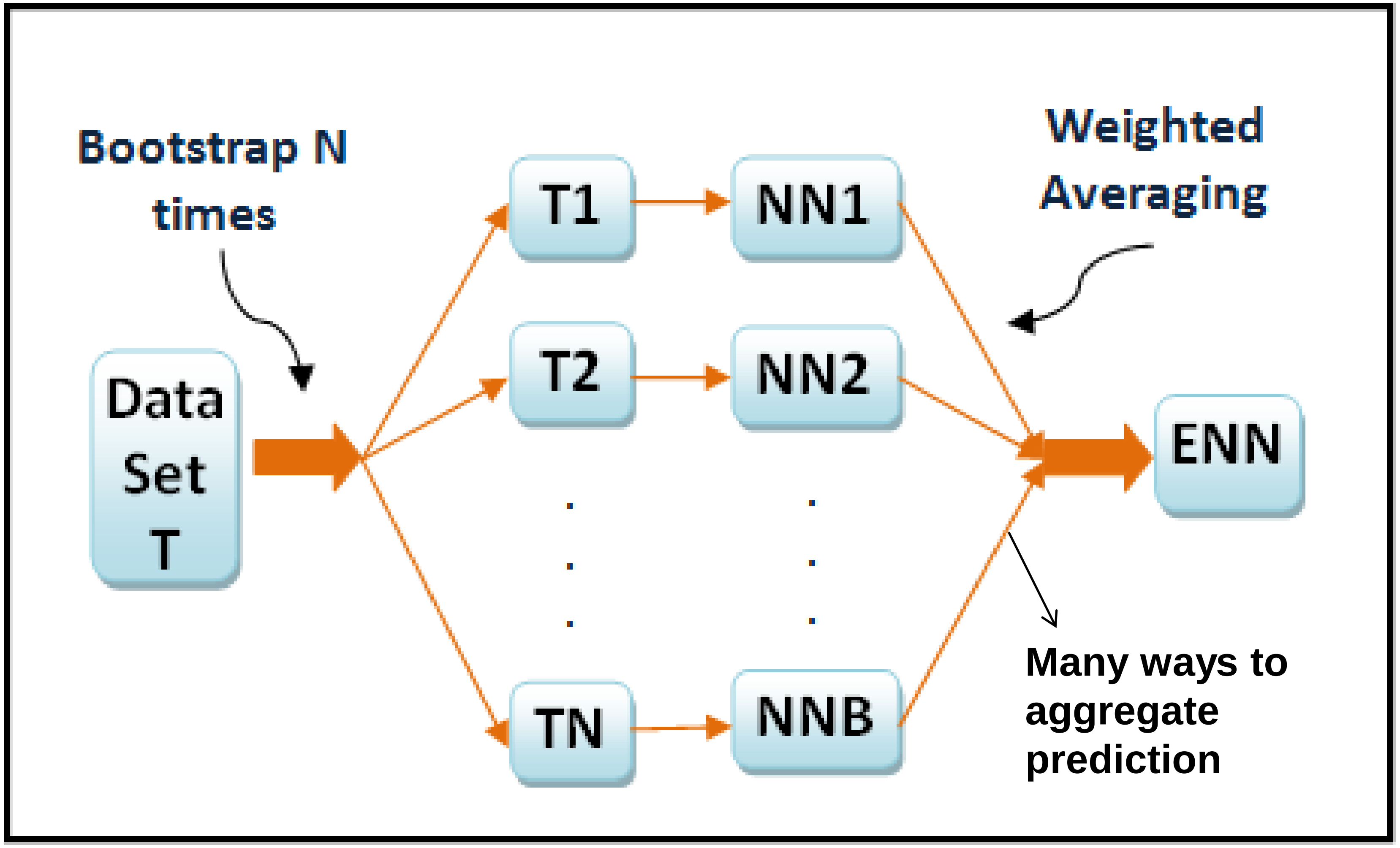
Model Generation

```
Let n be the number of instances in the training data.
For each of t iterations:
  Sample n instances with replacement from training data.
  Apply the learning algorithm to the sample.
  Store the resulting model.
```

Classification

```
For each of the t models:
  Predict class of instance using model.
Return class that has been predicted most often.
```

Bagging


Bootstrap N times
Data Set T
T1
T2
TN
NN1
NN2
NNB
Weighted Averaging
ENN
Many ways to aggregate prediction

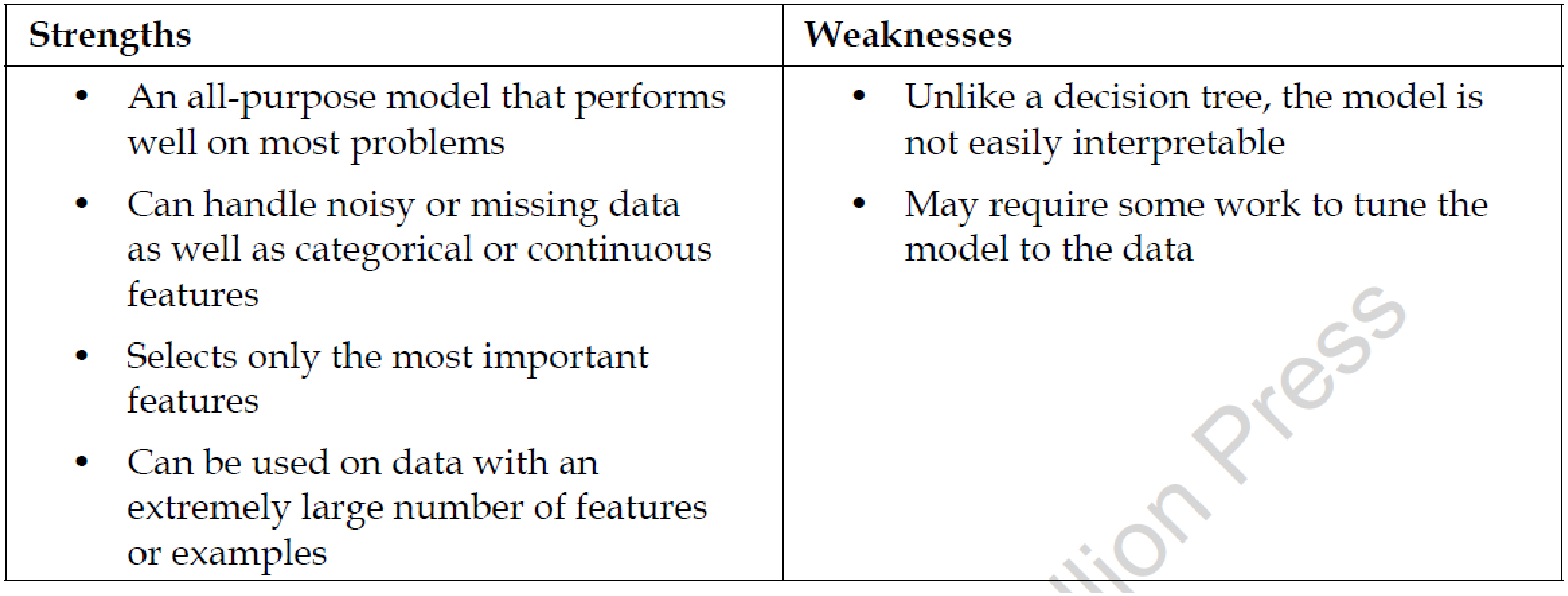
Bagging: Remarks

- Back to our analogy about a group of normal people making decision together, bagging shows one generic benefit of "democracy" => majority voting = averaging quality decision, which is more stable and won't be too bad, relative to decision making by one "normal person" as a dictator.
- Research shows that bagging of decision is better often when the pruning is switched off.
- Averaging predicted probabilities (if available) is even better than majority voting of predicted label classes.

2. Random Forest

- Another famous method is called **random forests (or decision tree forests)**.
- It combines the basic principles of bagging with **random feature selection** to add additional diversity to the decision tree models.
- The key idea is in addition to creating several training sets by bootstrapping, each tree uses **only a small, random portion of the full feature set** to build different trees on the same training set.
  - This alleviates the over-fitting and pruning trouble of decision tree.
- Random forests can handle dataset with extremely large number of attributes, where the so-called "curse of dimensionality" might cause other models to fail.
- Its error rates for most learning tasks are on par with nearly any other method. Performance is quite good.

Random Forest



Strengths	Weaknesses
• An all-purpose model that performs well on most problems • Can handle noisy or missing data as well as categorical or continuous features • Selects only the most important features • Can be used on data with an extremely large number of features or examples	• Unlike a decision tree, the model is not easily interpretable • May require some work to tune the model to the data

In short, Random Forest is one very important algorithm that you should bear in mind as a candidate for future analytics exercise. Its performance is quite good and also SVM and NN are very slow when you have a lot of attributes.

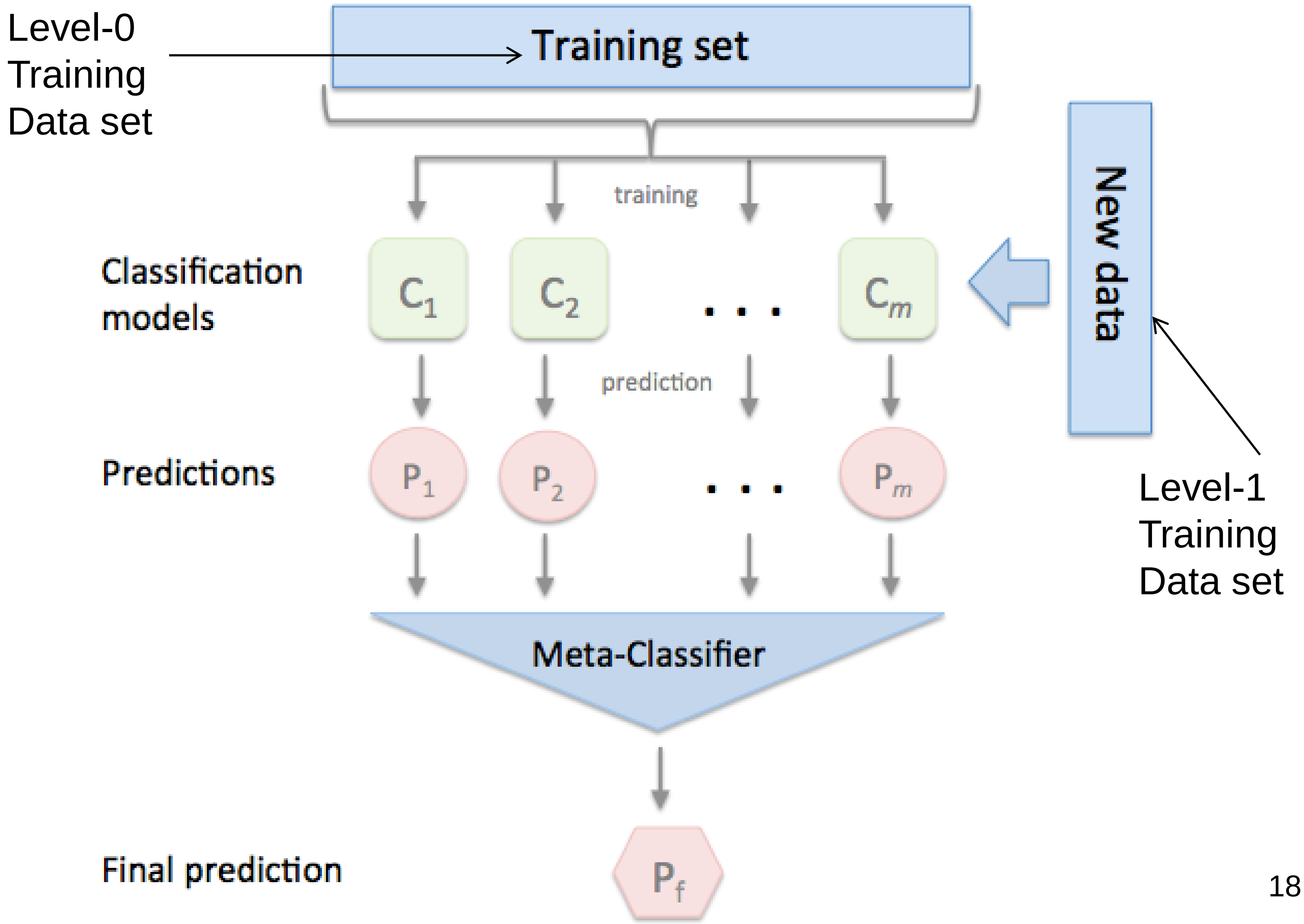
3. Stacking

- Bagging is used for the same learning scheme (same algorithm and same parameters).
- We can apply different learning schemes (different algorithms) on the same training dataset and then use majority voting to make a prediction.
  - This won’t work well in general. What is the problem of this approach? What is the difference from Bagging?
- Stacking approach applies a “meta-learning” approach to find out which are the better algorithms and also the best aggregating function from several learning schemes.

Stacking

- First, we need to split the whole training dataset into two sets, one for level-0 (baseline learning by all kinds of algorithms) and one for level-1 (meta learning).
- Second, *K* different algorithms are applied to level-0 to build *K* different models. This step is the same as you've learned.
- We can use the *K* built predictive models on **level-1** data to make *K* predictions for each record.
- The *K* predicted values of labels (either class or numbers) are the input attributes to the level-1 meta learning while the true values at level 1 is the class or values for prediction and validating performance.

Stacking


Level-0 Training Data set
Training set
training
Classification models
C_1
C_2
. . .
C_m
New data
prediction
Predictions
P_1
P_2
. . .
P_m
Level-1 Training Data set
Meta-Classifier
Final prediction
P_f

Stacking

- Research suggests this approach instead of applying both level-0 and level-1 on the same dataset.
- Also, you can try 10 fold-cross validation to split the dataset into level 0 and level 1 and apply several times.
- One question remains: what is the best method for level 1? The **rule of thumb (means no scientific answer)** is to use a simple and transparent method, such as linear regression or simple decision tree as the meta-learner at level 1.
- The original suggestion is “relatively global and smooth” algorithm.

Stacking

- Stacking is very important in practice because most the data competition winners and experienced data scientists all claim this method helps.
- Stacking can stabilize your prediction performance. It can improve your average prediction performance typically only by a small numbers.
- NO science about the best meta-1 method.
- More references provided in the footnote to read if you are interested in. Roughly speaking, several algorithms proposed in the literature, but no one algorithm consistently works in all real-world cases.
- So my suggestion is you can just use RF or GBM as level-1 meta learner.

4. Boosting

- Boosting is an important invention in the data mining literature because it indeed boosts the performance of almost all classification algorithms.
- It was invented in 1996-1997
  - Freund, Yoav, and Robert E. Schapire. "Experiments with a new boosting algorithm." *ICML*. Vol. 96. 1996.
  - Freund, Yoav, and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of Computer and System Sciences* 55.1 (1997): 119-139.
- Boosting typically can improve the prediction performance quite significantly and that is why it is quite important in practice.

Two Differences: Boosting vs Bagging

1 For classification, one way to improve performance is to ignore the predictions from low-performing classifiers. We don’t do majority voting and better classifiers will have a higher weight.

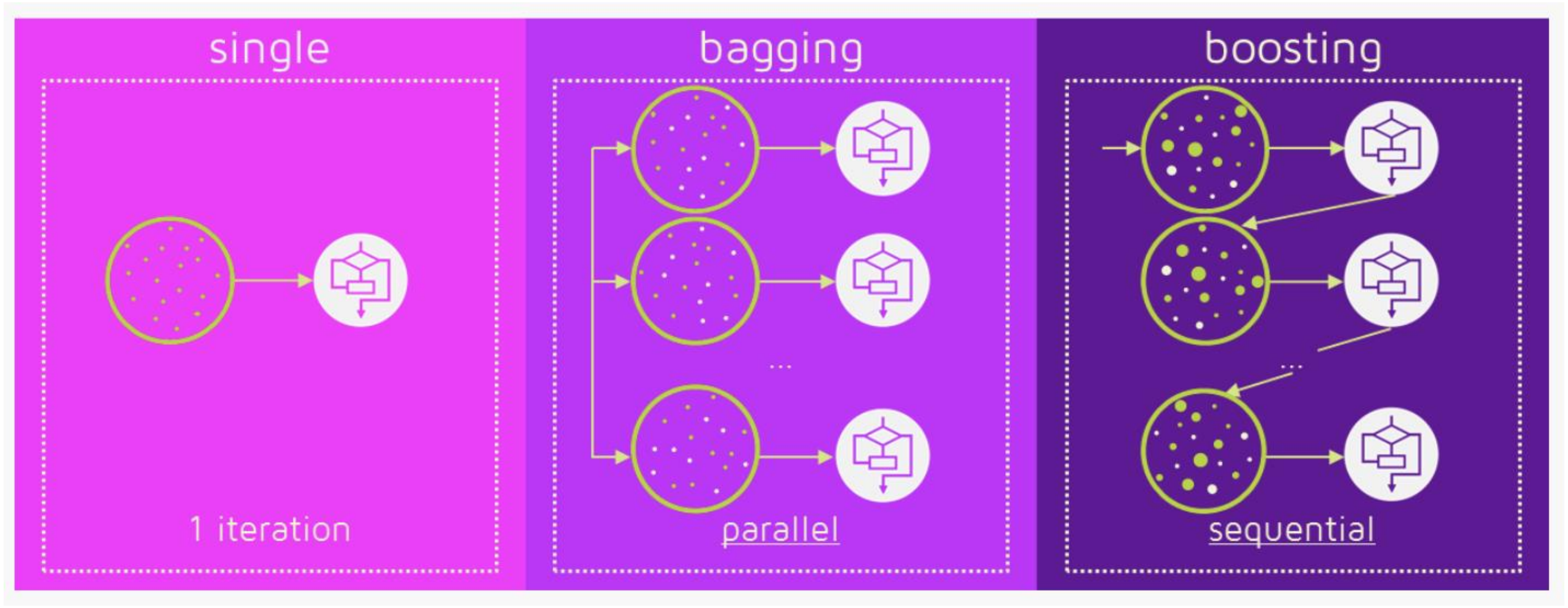
2 The other idea is we hope different algorithms “complement each” in the sense that they perform better on different subsets of training dataset.

- Back to the democracy example, group decision making can achieve better performance when (1) dumb ideas are ignored, (2) different people are experts in different areas => no one in the world knows everything.

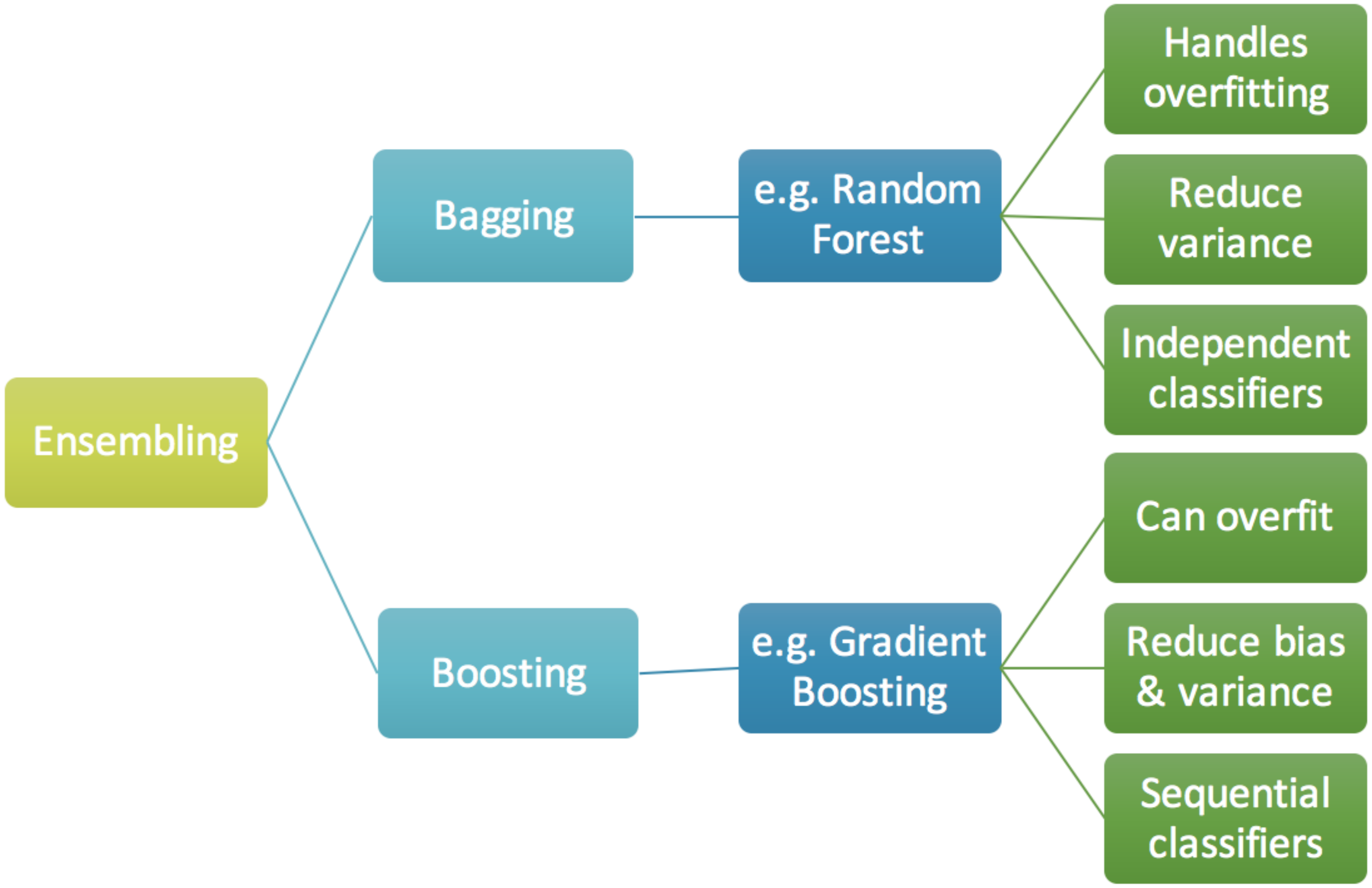
More differences: Boosting

- Boosting is sequential whereas in bagging or RF, individual models are built in parallel.
- In boosting, each new model is influenced by the performance of those built previously.
- **Boosting encourages new models to become experts for instances handled incorrectly by earlier ones by assigning greater weight to those instances.**
- **A final difference is that boosting weights a model's contribution by its confidence rather than giving equal weight to all models.**

Boosting



Boosting


Ensembling
Bagging
e.g. Random Forest
Handles overfitting
Reduce variance
Independent classifiers
Boosting
e.g. Gradient Boosting
Can overfit
Reduce bias & variance
Sequential classifiers

The first successful boosting: AdaBoost

- There are many (new and important) variants on the idea of boosting.
- Textbook describes a widely used method called *AdaBoost.M1* which is also the algorithm invented in 1997.
- Like bagging, it can be applied to any classification learning algorithm.
  - For the basic explanations here, we assume the classification algorithm works with weighted instances/records.
- The first step of the algorithm is to build a classifier as usual (on all training data with equal weights).

AdaBoost

Step 2: We will iteratively classify the weighted dataset several times from now on.

- In each iteration, the idea is: the weight of correctly classified instances in the previous round is relatively **decreased**, and that of misclassified ones is relatively **increased**.
- When we increase the weight in this way, the new classifier will be trained with focus on classifying those “harder” instances/records.
- How much is the weight? (see next slides)

Step 3: Output weights for classifiers to predict in the future.

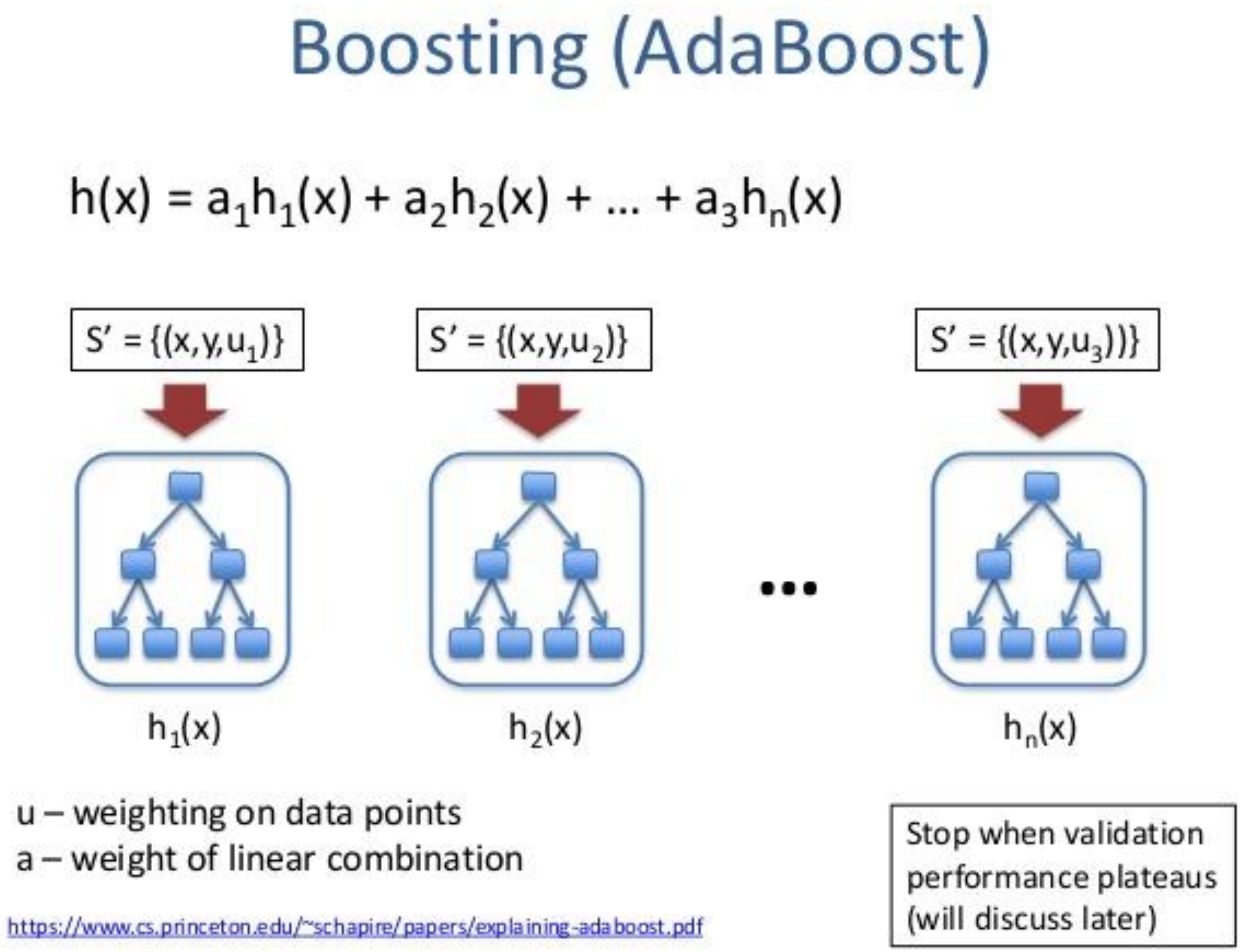
(Optional reading) AdaBoost.M1

Algorithm 10.1 *AdaBoost.M1.*

```
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
        $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
    (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 $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
```

Should be – α_m*I

Boosting



Boosting (AdaBoost)

$$h(x) = a_1h_1(x) + a_2h_2(x) + \dots + a_3h_n(x)$$

u – weighting on data points
a – weight of linear combination

https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

Stop when validation performance plateaus (will discuss later)

Overview of Gradient Boosting

- Gradient boosted model (GBM) is a generalization of tree boosting that attempts to mitigate problems of AdaBoost, so as to produce an accurate and effective off-the-shelf procedure for data mining.
- Invented in 2001,
  - Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189-1232.
- One of the best performing method today.
- The high-level theoretical idea is similar to steepest descent method we've discussed. If our goal is to minimize sum of squared error by choosing a function f(x) to approximate y, then

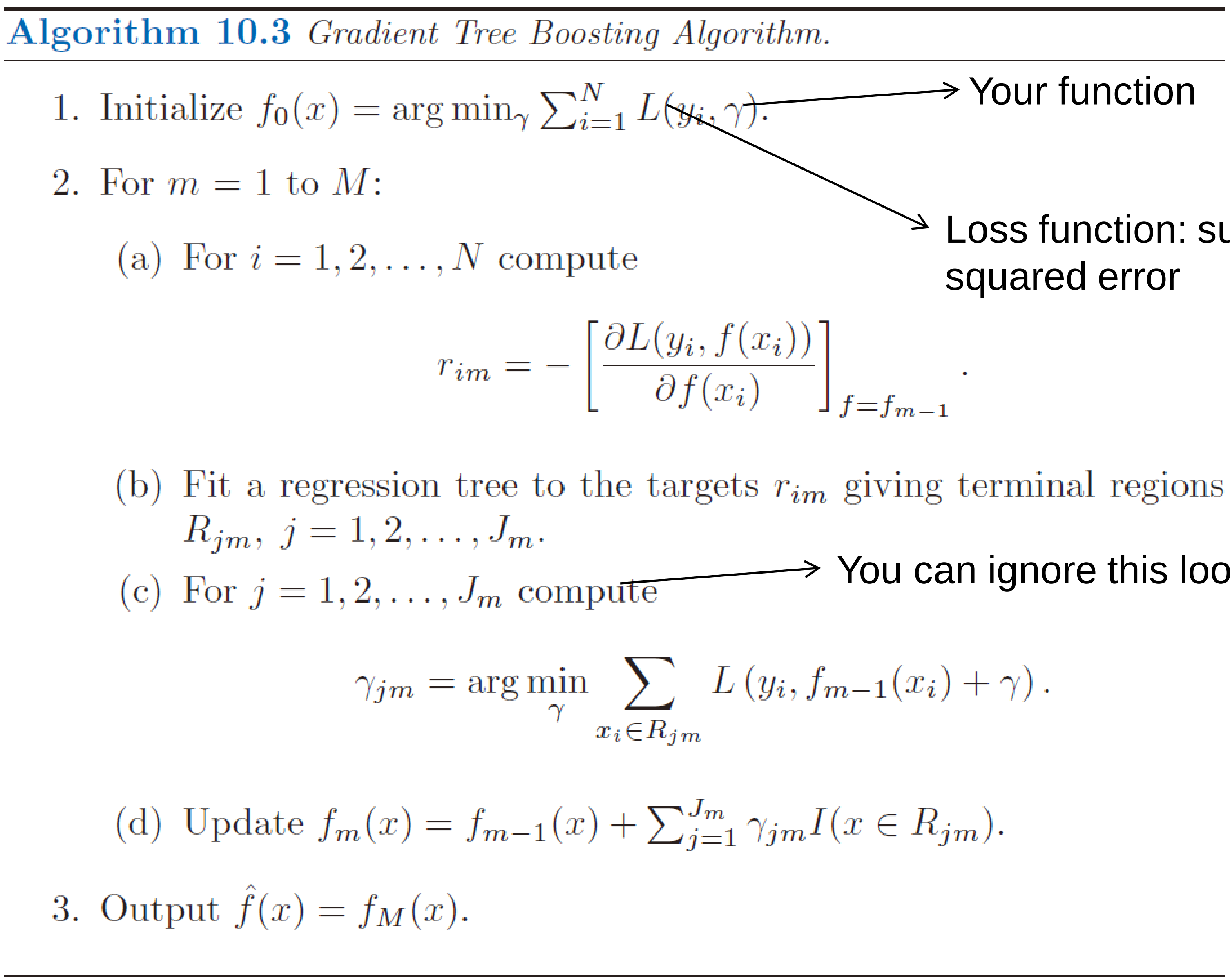
Gradient Boosting: Simplest Implementation

- Sum of Squared Error = sum of $(y-f(X))^2$
- If you differentiate this expression with respect to f(X), then it is just
  $2 \times (y-f(X))=2 \times$ residual.

The simplest implementation is not that complicated. (GBM can have a complicated version.) SO,

- We first use a regression tree f1 to predict y.
- In the next iteration, we build a 2nd regression tree f2 to predict residual = y-f1.
- Iteratively we use f3 to predict residual = y-f1-f2.
- We stop by carefully evaluating the prediction performance on the test set (by 10-fold cross validation) to avoid overfitting.

(Optional reading) Gradient Boosting



Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

```
1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^N L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm},\ j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$.
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

Your function

Loss function: sum of squared error

You can ignore this loop

Boosting: Final Remarks

- Boosting is an amazing method that can turn a weak classifier into an accurate, strong classifier.
- With more classifiers, Boosting algorithm can achieve very small error rates on the training set (although it may suffer from over fitting).
- Boosting often produces classifiers that are significantly more accurate on fresh data than ones generated by bagging.
- More about XGBoost and LightGBM next week.

Final Remarks of Ensemble Method

- **Boosting (especially XGBoost and LightGBM) is an important technique to drastically increase the accuracy on the training set and test set.**
- **Stacking is also very important in practice. It can further improve your prediction performance a bit.**

- Random Forest is the second most important method learned today. You should try this in the future too, especially if the number of attributes are large. Although simple, its performance is quite good.

- Bagging is a must-use for algorithms sensitive to small changes in the training set => GBM and Random Forest already take care of this issue in the algorithm.