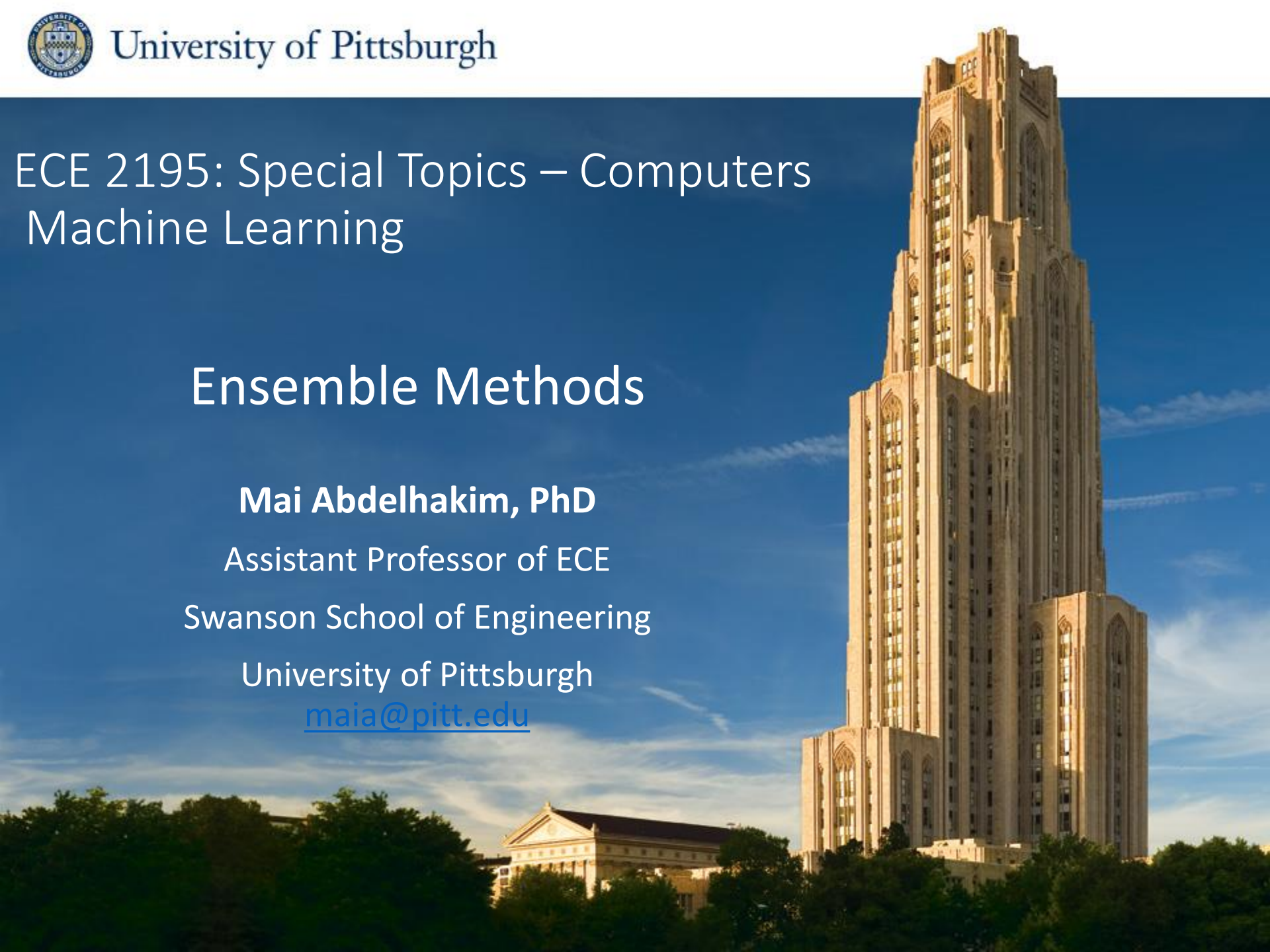


University of Pittsburgh

ECE 2195: Special Topics – Computers Machine Learning

Ensemble Methods

Mai Abdelhakim, PhD

Assistant Professor of ECE

Swanson School of Engineering

University of Pittsburgh

maia@pitt.edu

Ensemble Methods Combine Learners

- Strong classifier can be generated by multiple weak classifiers
  - Intiution: Each classifier may make different errors, combining them would result in less errors
- Divide and conquer
  - Divide space into smaller, easier-to-learn partitions
- Ensemble or Combining methods must have
  - Components algorthims (e.g. classifiers) - base algorithms
  - Method of combining output

How to achieve achieve diversity

- Classifiers are diverse if their decision boundary is adequately different from each other
- One way to achieve diversity is to train algorithms on different sets of data
  - Via resampling (e.g. boostrapping aggregation = bagging)
- Another way is to use different tuning parameters/features
- Use of different methods (NN, decision trees, logistic regression, etc)

Combining Methods

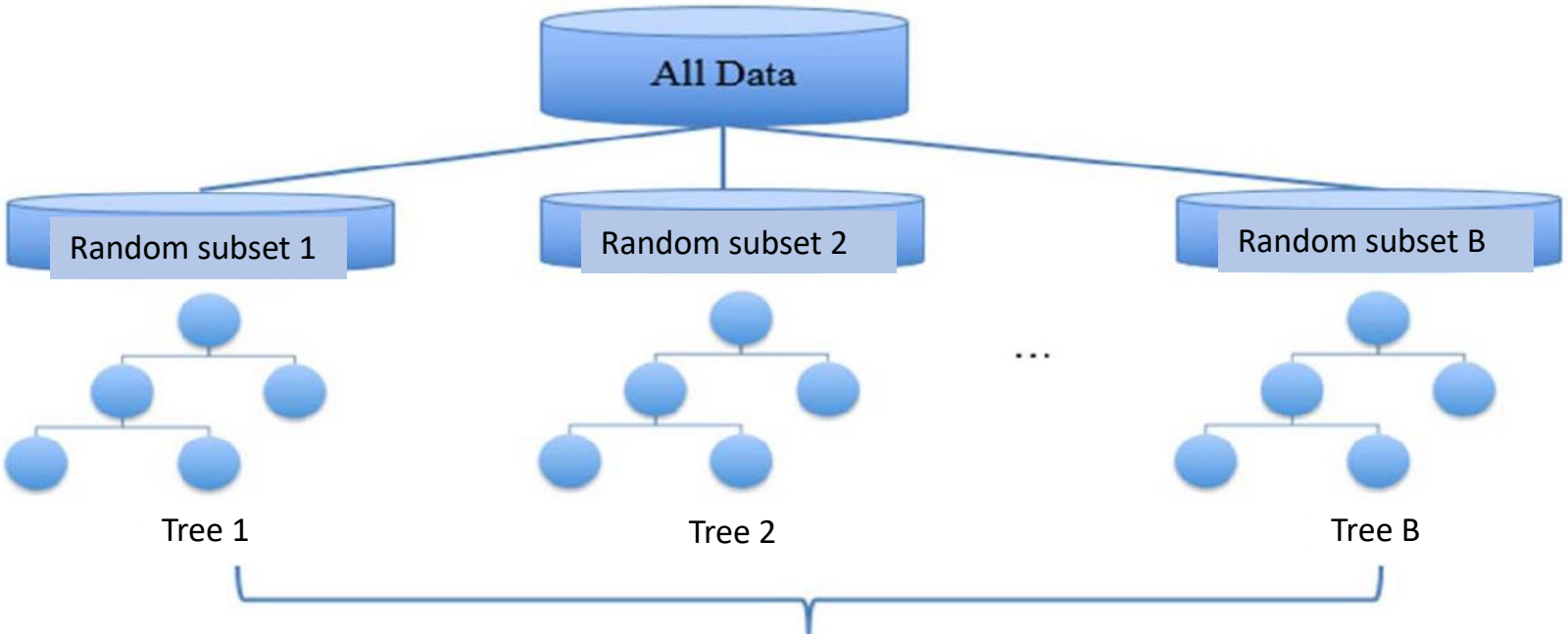
Combining methods are also called **ensemble methods**

- Bagging (not specific to decision trees), also called *bootstrap*
- Random Forests
- Boosting (not specific to decision trees)
- Other techniques: e.g. stacking

Bagging (Bootstrap)

- General procedure for reducing the variance of a statistical learning method (not specific to decision trees)
- Idea:
  1. **Create multiple training sets** from the the original training set – Say we now have **B training subsets**
     - **Repeated sampling with replacement** (meaning that an observation can be found in more than one of these subsets)
  2. **Train base algorithms** (B models) each is trained with a different training subset
  3. Get **overall response** using B models for prediction
     - Regression: predicted response is **average** of predicted responses by different models
     - Classification: predicted response is obtained by **majority vote** from different models

Bagging (Bootstrap)



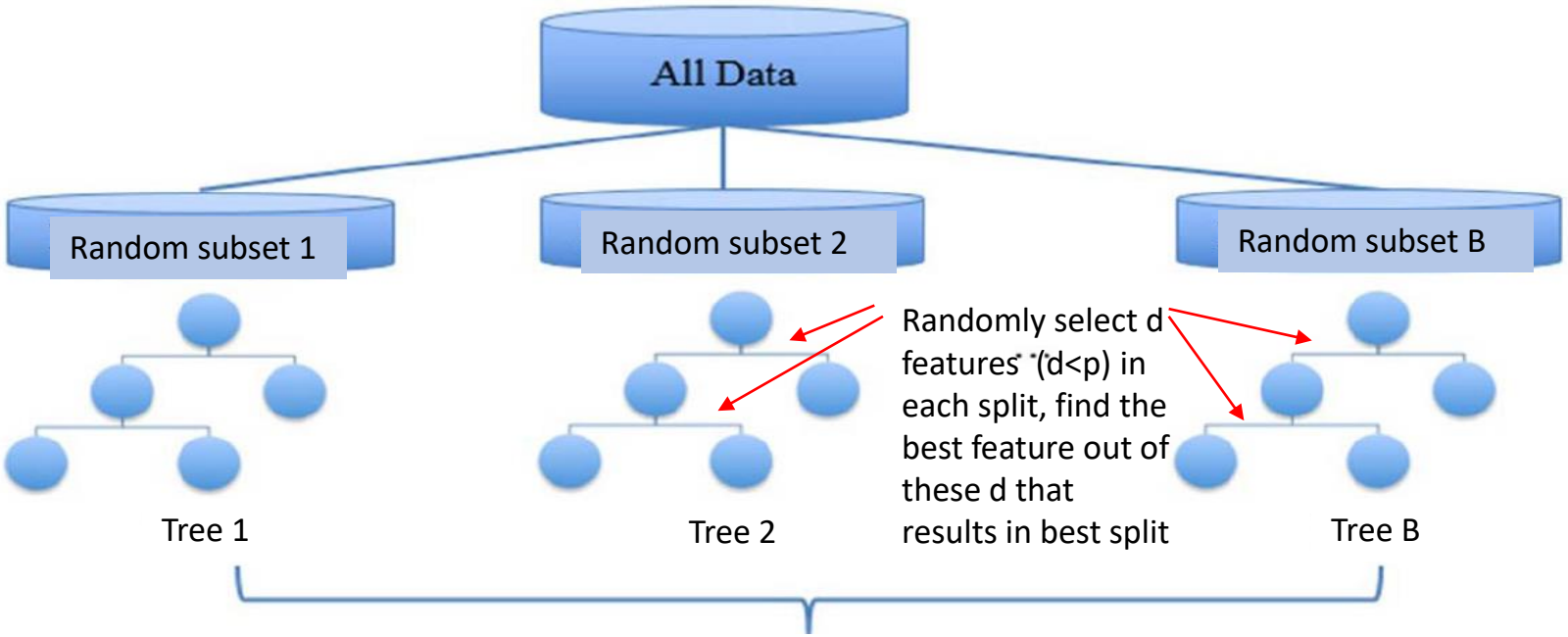
Use voting for classification, or averaging for regression to get final response

Ref.: Kun Xie et al, ASCE 2017

Random Forests: Try to de-correlate the trees

- Have better classification performance compared to bagging
- Random forests builds a number of decision trees on bootstrapped training samples
- Similar to Bagging, however, in each split in a tree, we **randomly select subset of features** (d features < p), then find the best one out of these d features for splitting
  - This is instead of considering all features and finding the best one
  - We can select $d = \sqrt{p}, p/2 \ldots$
    find best feature out of the d randomly selected features to find the split
- Aggregate the results through voting (classification) or averaging (regression)

Random Forests



Python function

- Random Forest

from sklearn.ensemble import RandomForestClassifier

forestModel= RandomForestClassifier (n_estimators=M, max_features=d, max_depth=m, random_state=0)

  - n_estimators: number of trees to grow in the forest
  - max_features: maximum number of features considered to find best split
  - max_depth is depth of the tree

Boosting

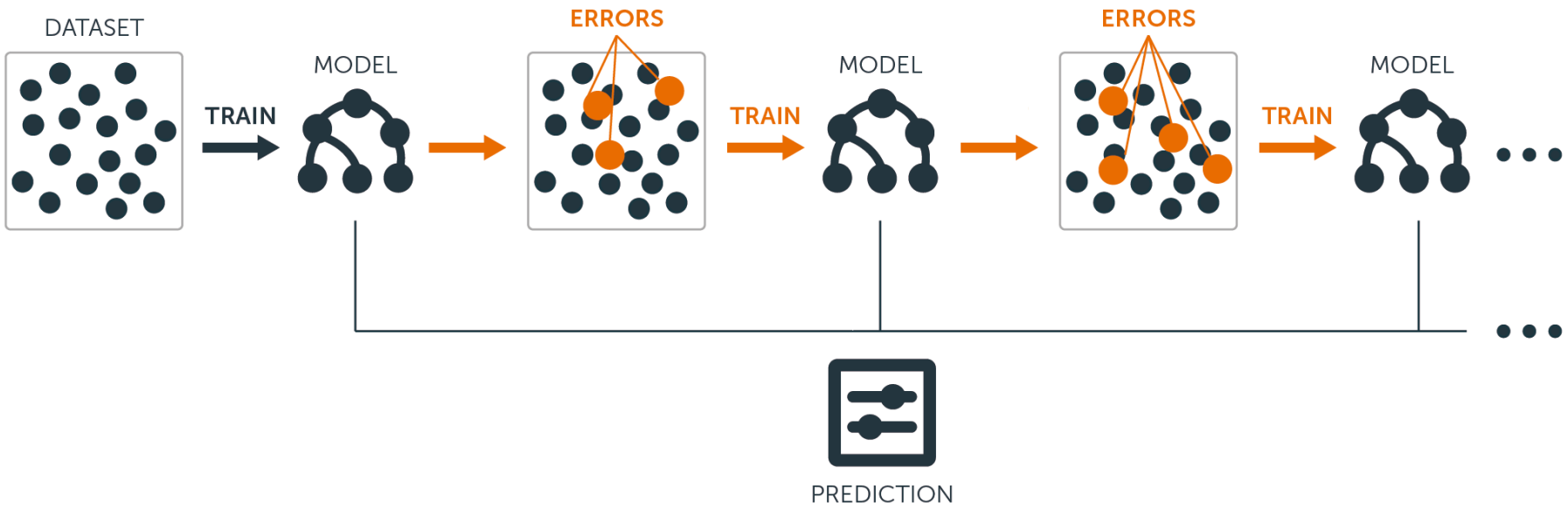
- General approach that can be applied to statistical learning methods (not specific to decision trees)
- **Purpose**: **Combine output of multiple weak learners to produce a powerful learner**
- In **bagging**, the models created are **independent** of each other
- In **boosting**, models (e.g. decision trees) are gown **sequentially**, and each model uses information from previous model

Adaptive Boosting (AdaBoost)

- AdaBoost (1997) is a popular boosting algorithm

- When applied with classification decision trees:
  - Build trees in sequence
  - Each tree can be very small (only few nodes)
  - Each tree **focuses on areas where previous trees do not perform well**
    - Made through assigning a weight for each data point, and the weight gets higher when the point is misclassified
  - Combine results using
    - **Weighted** majority vote used for classification
    - **Weighted** average can be used for regression

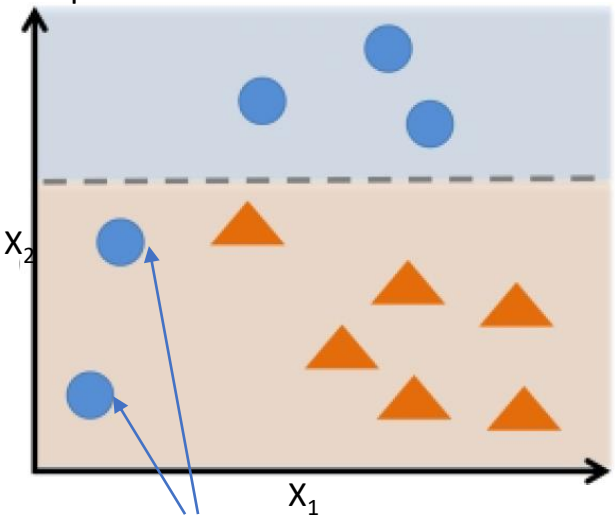
AdaBoost: Models Trained in Sequence



https://blog.bigml.com/2017/03/14/introduction-to-boosted-trees/

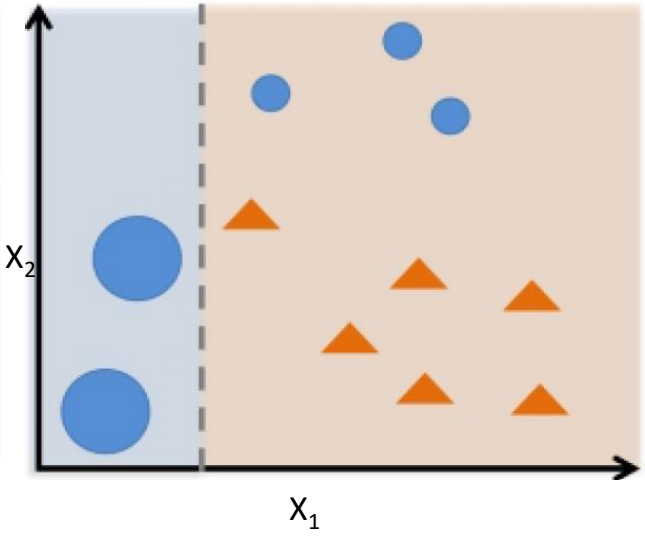
Example: AdaBoost with 3 Learners – stumps (single-split tree) & 2 features

First tree (weak learner) result in this splitting of feature space

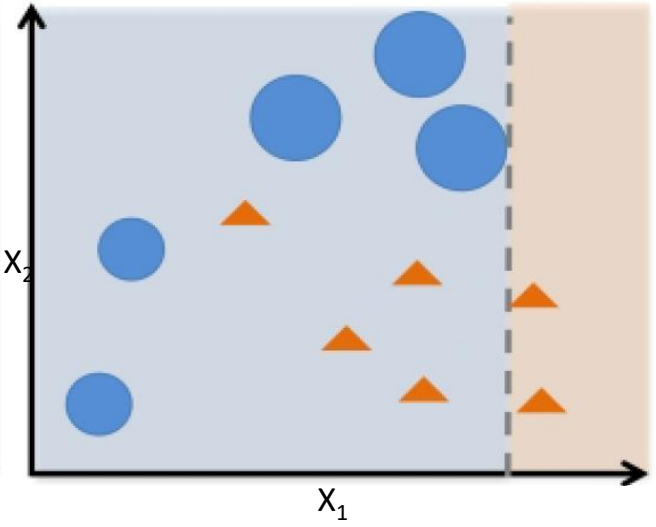


These samples were misclassified by first tree, will get higher weight

Second tree focuses on misclassified samples from previous tree (have higher weight) and results in this splitting of feature space



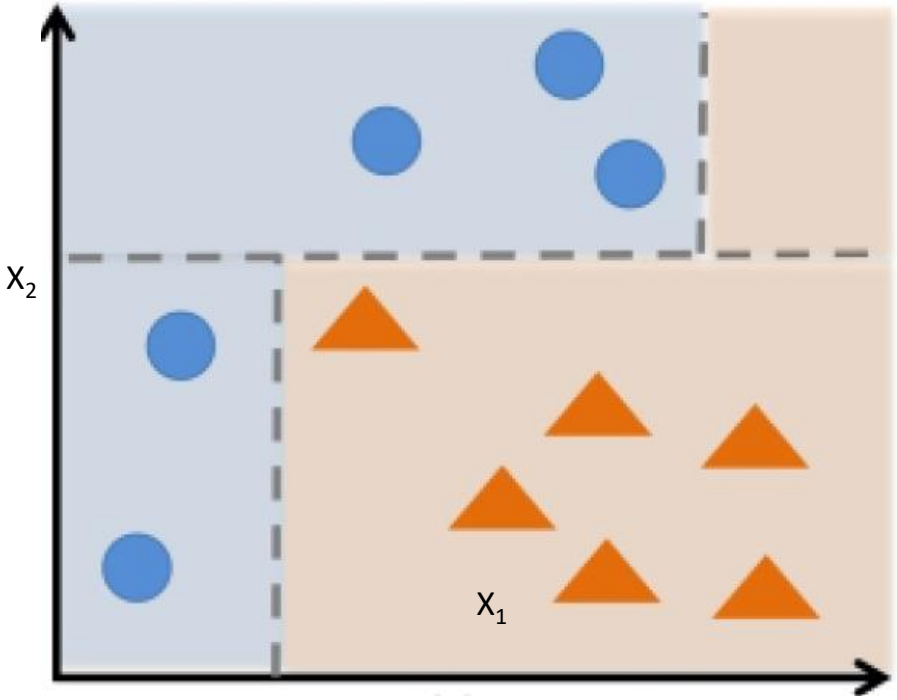
Third tree focuses on misclassified samples from previous and results in this splitting of feature space



Raschka, Python machine learning, Chapter 7

Example: AdaBoost with 3 Learners …Cont.

- When combining the results of the previous three learners using **weighted majority vote** we get:



Raschka, Python machine learning, Chapter 7

- Note: if predicted labels of 5 classifiers are {1,1,1,-1,-1}
  - Typical majority gets final prediction to be **1**
  - Weighted majority with weights {0.1,0.1,0.1, 0.6,0.1)
    - 0.1x1 +0.1x1 +0.1x1 +0.6x(-1) +0.1 x(-1) = - 0.4 < 0 ➔Final prediction is **-1**

AdaBoost: Combining via weights

- Consider two classes, with response coded as Y=-1 or Y=1, and feature x (positive and negative class)
- We have M classifiers:
  - The prediction of classifier m is $G_m(x)$, m=1,2,…M
    - $G_m(x)$ is either -1 or 1 (predicted class label)
- The **final prediction** from all of classifiers are combined through **weighted majority vote**

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

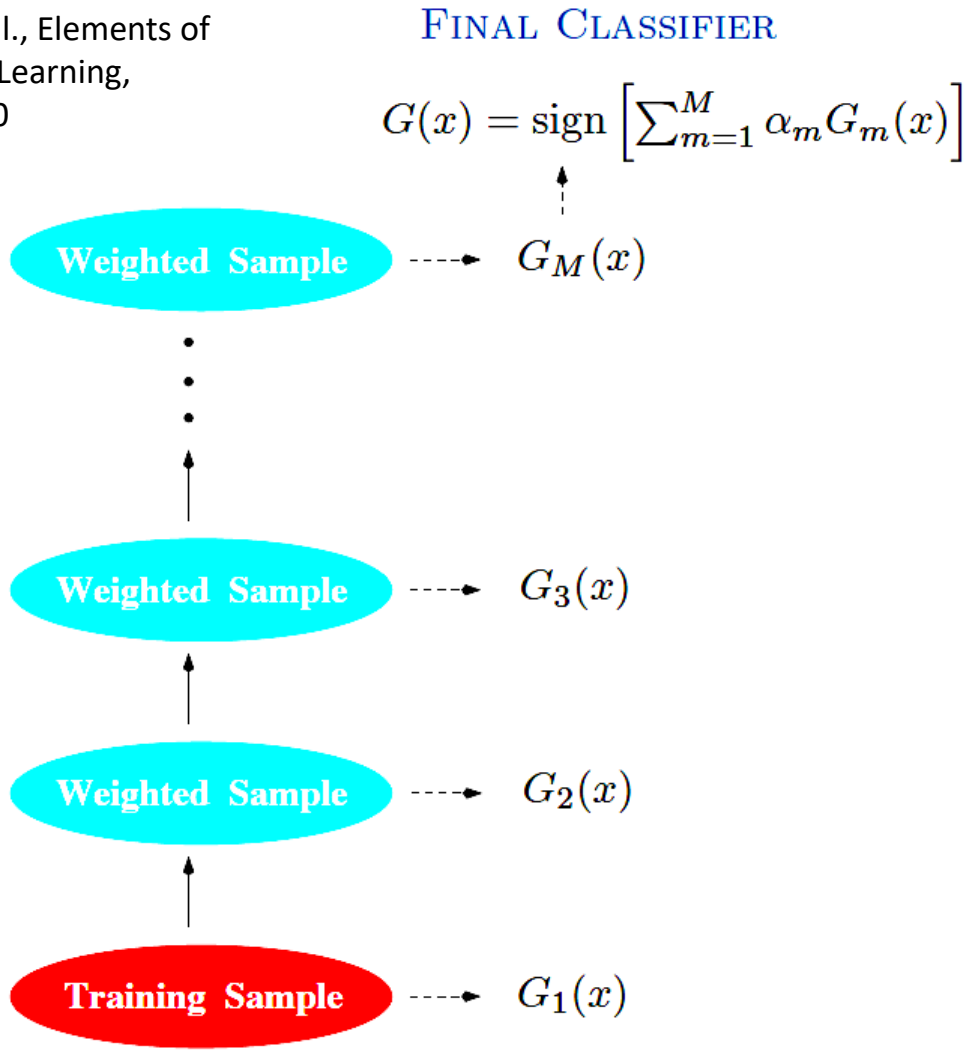
Note:
sign(F) = -1, if F is negative
sign(F) = 1, if F is positive

- α_m are the weights computed by the boosting algorithm

AdaBoost

Hastie et al., Elements of Statistical Learning, Chapter 10

- Every boosting step we apply **weights to each training observation**
  - Initially weights are equal (set to 1/n for all observations)
  - For successive iterations, the weights are modified
    - Observations that are **misclassified** have their **weights increased**
    - Observations that were **correctly classified** have their **weights decreased**
- Each successive classifier is hence forced to concentrate on training observations that are missed by previous ones



AdaBoost Algorithm: Minimize the weighted error

1. Initialize equal weights of all observation (w_i=1/n)
2. For m from 1 to M, (loop over classifiers in sequence)
   1. train classifier m: $G_m(x)$
   2. Compute **weighted error**: $err_m = \sum_{i=1}^{n} w_i \, I\left(y_i \neq G_m(x_i)\right)$
   3. Compute weights of classifier m: $\alpha_m = 0.5 \log_e \left(\frac{1-err_m}{err_m}\right)$
      - α_m is higher when errors are lower
   4. Update weights of each observations
      - $\boldsymbol{w_i \leftarrow w_i \exp(-\alpha_m y_i \hat{y}_i)}$
        - The exponent $(-\alpha_m y_i \hat{y}_i)$ is negative if $y_i = \hat{y}_i =$ ➔ ***weight*** **decrease if samples are classified correctly**
          - Otherwise, exponent is positive ➔ weight increase misclassified samples
      - Normalize the final weights by dividing by $\sum_{i=1}^{n} w_i$

$$Note: I(y_i \neq G_m(x_i)) = \begin{cases} 0 & If\ y_i = G_m(x_i) \\ 1 & If\ y_i \neq G_m(x_i) \end{cases}$$

Raschka, Python machine learning, Chapter 7

Example:

In this example we decide y=1 if $x \leq 3$

Sample indices	x	y	Weights	$\hat{y}$(x <= 3.0)?	Correct?	Updated weights
1	1.0	1	0.1	1	Yes	0.072
2	2.0	1	0.1	1	Yes	0.072
3	3.0	1	0.1	1	Yes	0.072
4	4.0	-1	0.1	-1	Yes	0.072
5	5.0	-1	0.1	-1	Yes	0.072
6	6.0	-1	0.1	-1	Yes	0.072
7	7.0	1	0.1	-1	No	0.167
8	8.0	1	0.1	-1	No	0.167
9	9.0	1	0.1	-1	No	0.167
10	10.0	-1	0.1	-1	Yes	0.072

err_1 =0.3, α_1=0.5*ln((1-0.3)/0.3)=0.424 ➔ get updated weights in last column, then normalize to get values in last column

Calculation details in Raschka, Python machine learning, Chapter 7

Python

- AdaBoost:

```
from sklearn.ensemble import AdaBoostClassifier

BoostModel= AdaBoostClassifier(n_estimators=M)
```

  - n_estimators is the number of models
  - Take base_estimator parameter which is DecisionTreeClassifier, max_depth=1 by default

Gadient boosting = Gradient Descent & Boosting

- Generalize Adaboost to work with other loss functions
- Consists of week learners
- Add learner to minimize error of previous weak learner
- Sequential models, each try to reduce error of previous learner (move in direction of gradient)
- E.g. 1000's of trees , Stumps (tree of 1 split) work well!

Functional gradient descent - Example

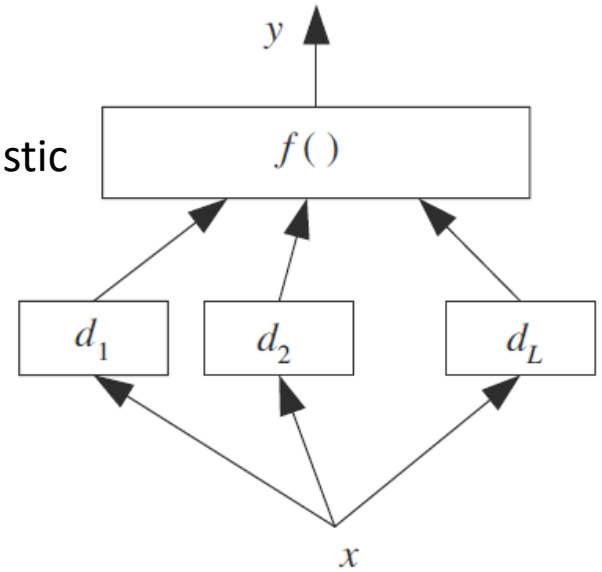
- Fit first learner with (x, y) samples
  - First learner gets predicted output $F(x_i)$
- Introduce **residual** to the second learner $H(x_i)$ (x, $y - F(x_i)$)
  - $(x_1, y_1 - F(x_1))$, $(x_2, y_2 - F(x_2))$…
- With these only we combine
  - $F(x_i) + H(x_i) = F(x_i) + y_i - F(x_i)$
- Note if a loss function is L= RSS/2= $\sum (y - F(x))^2/2$, this error is proportional to the gradient w.r.t output
  - $\frac{\partial L}{\partial F(x_i)} = -(y_i - F(x_i)) = F(x_i) - y_i$
  - $F(x_i) - \frac{\partial L}{\partial F(x_i)} = F(x_i) + y_i - F(x_i)$

Combining methods - stacking

- Stacking : Different base methods
  - Bagging & boosting same base methods
- All base algorithms are trained using the available data
- **Combiner algorithm** is trained to make a final prediction using all the predictions of the other algorithms and the true label

Combiner
e.g. Linear/Logistic regression

Base models
e.g. KNN, DT,..



Ethem Alpaydın, “Introduction to Machine Learning”