More Ensemble Methodology: On Forests and Boosted Ensembles

Alexander G. Ororbia II

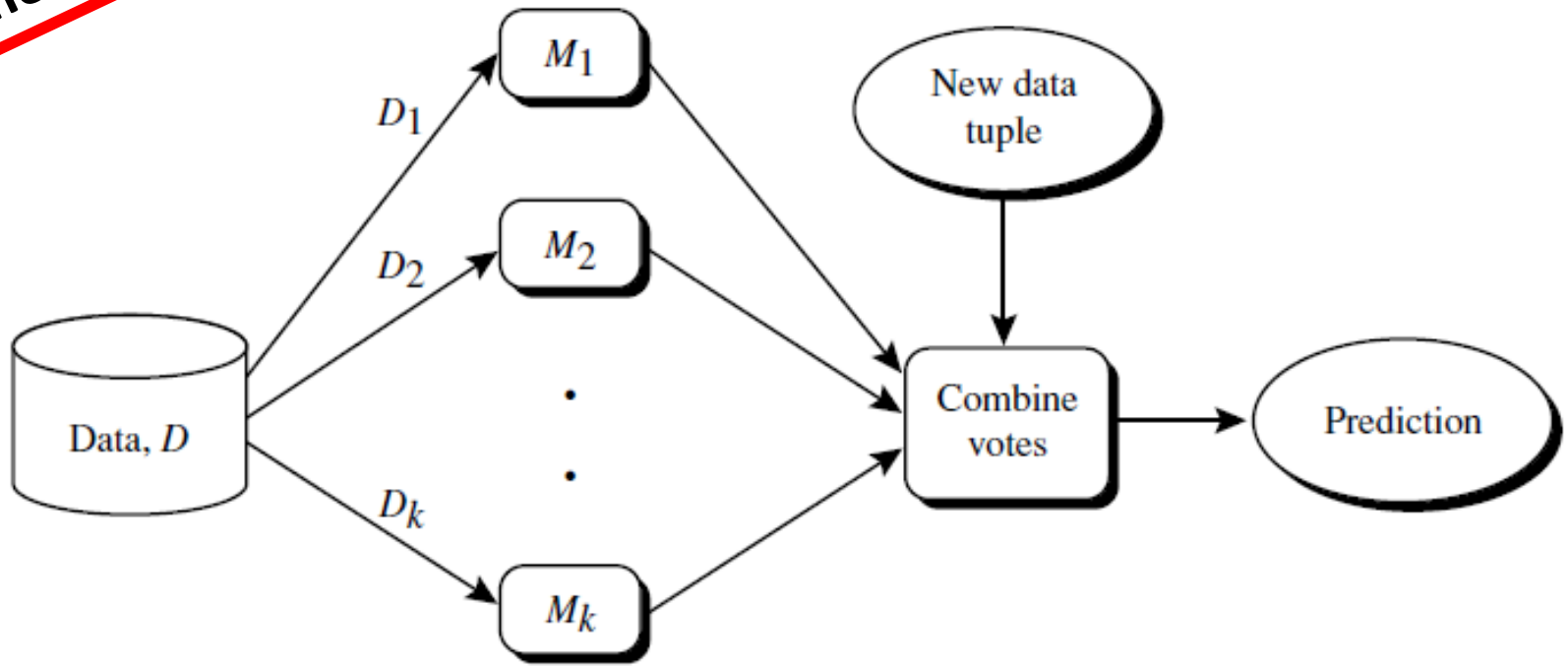
Introduction to Machine Learning

CSCI-335

4/13/2026

A *Bagging* Ensemble System

Last time!



Increasing classifier accuracy: Ensemble methods generate a set of classification models, $M_1, M_2, \ldots, M_k$. Given a new data tuple to classify, each classifier "votes" for the class label of that tuple. The ensemble combines the votes to return a class prediction.

Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

Output: The ensemble—a composite model, $M*$.

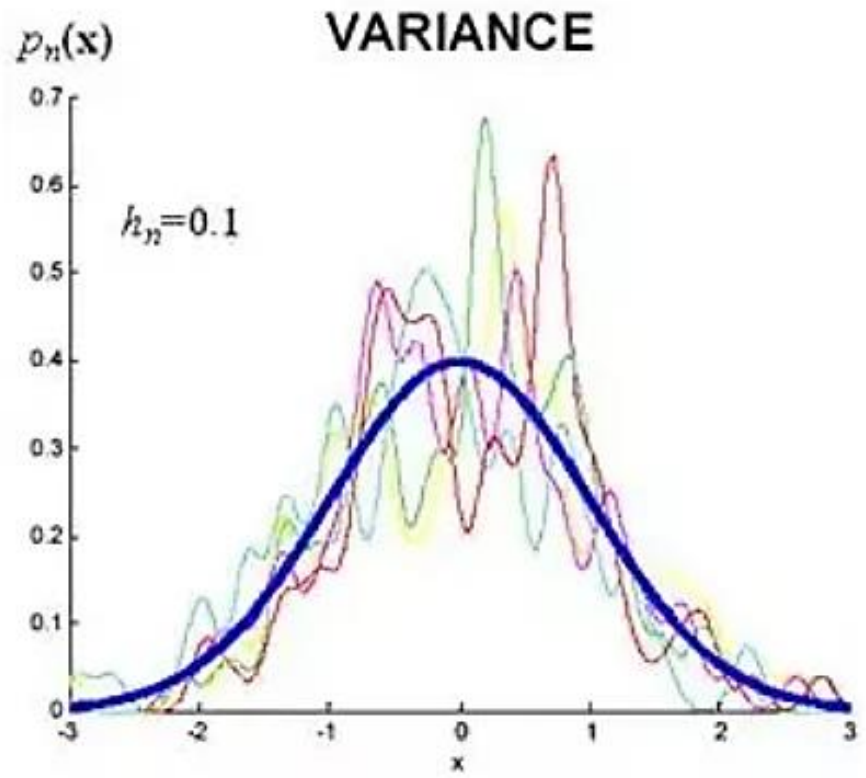
Method:

(1) **for** $i = 1$ to k **do** // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i and the learning scheme to derive a model, M_i;
(4) **endfor**

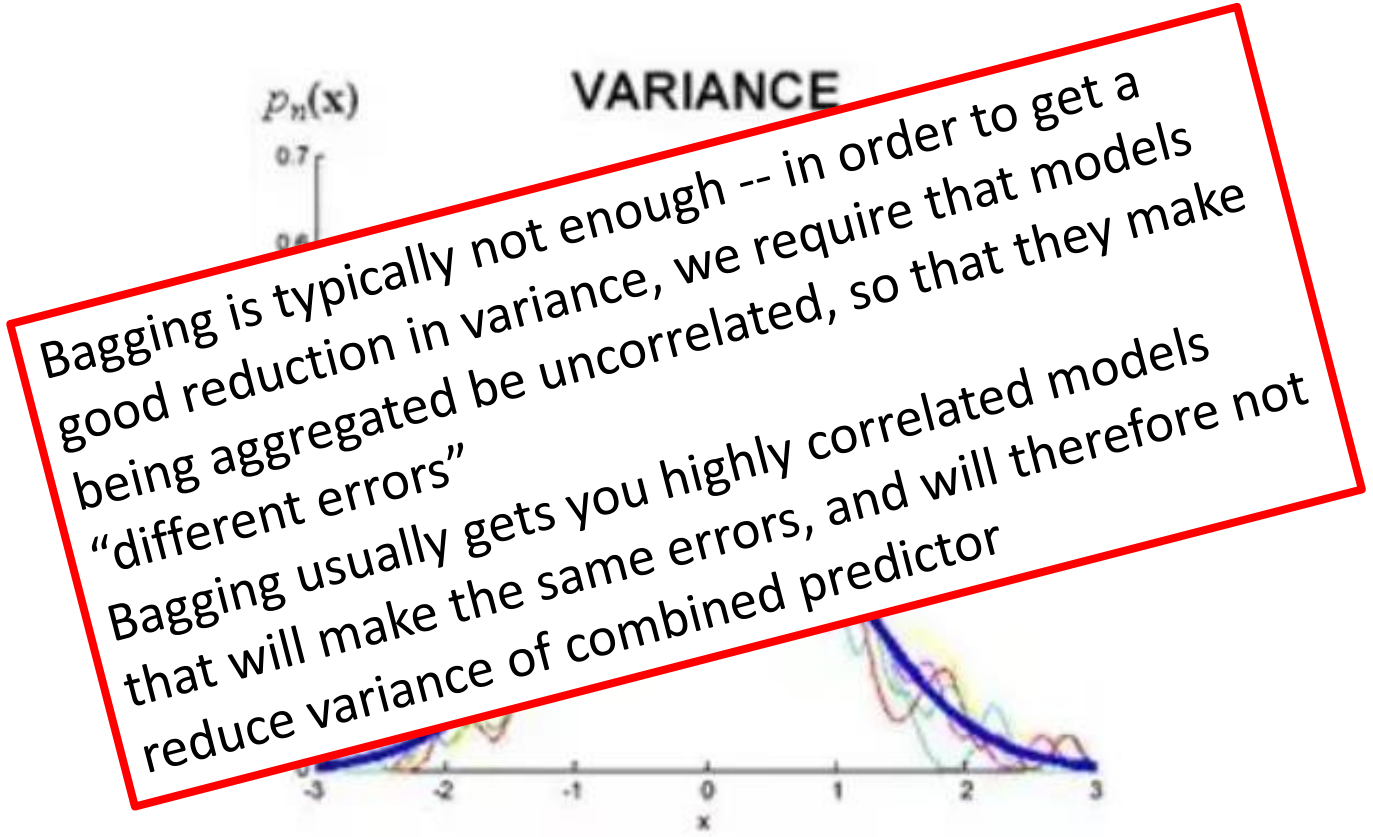
To use the ensemble to classify a tuple, X:

let each of the k models classify X and return the majority vote;

Insight: By ensembling predictors by averaging (or generally aggregating) many low bias, high variance predictors, we can reduce the variance while retaining low bias!



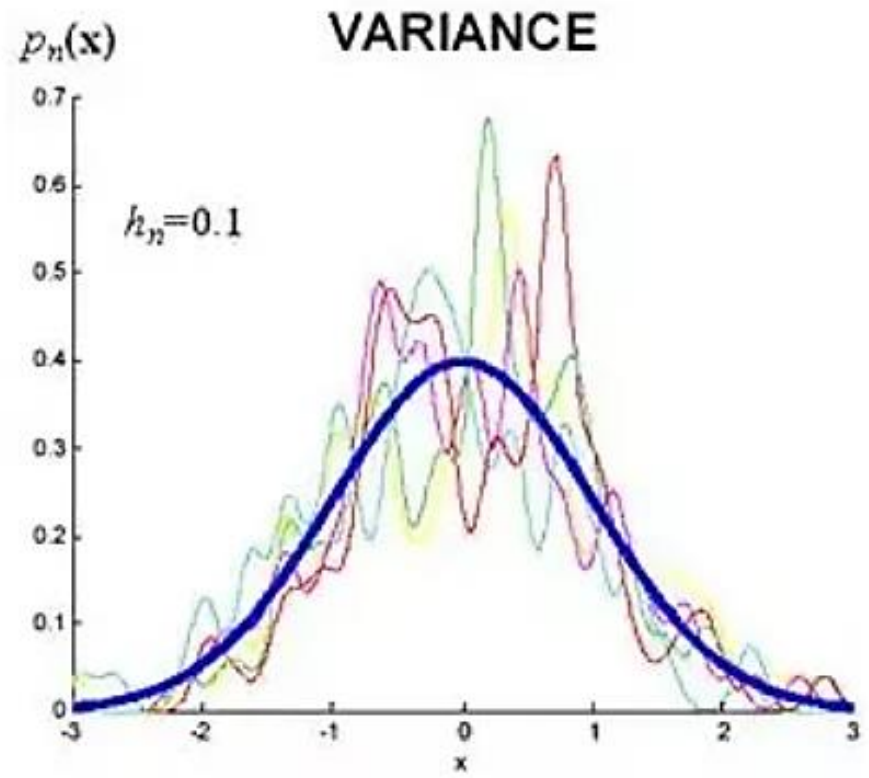
Insight: By ensembling predictors by averaging (or generally aggregating) many low bias, high variance predictors, we can reduce the variance while retaining low bias!



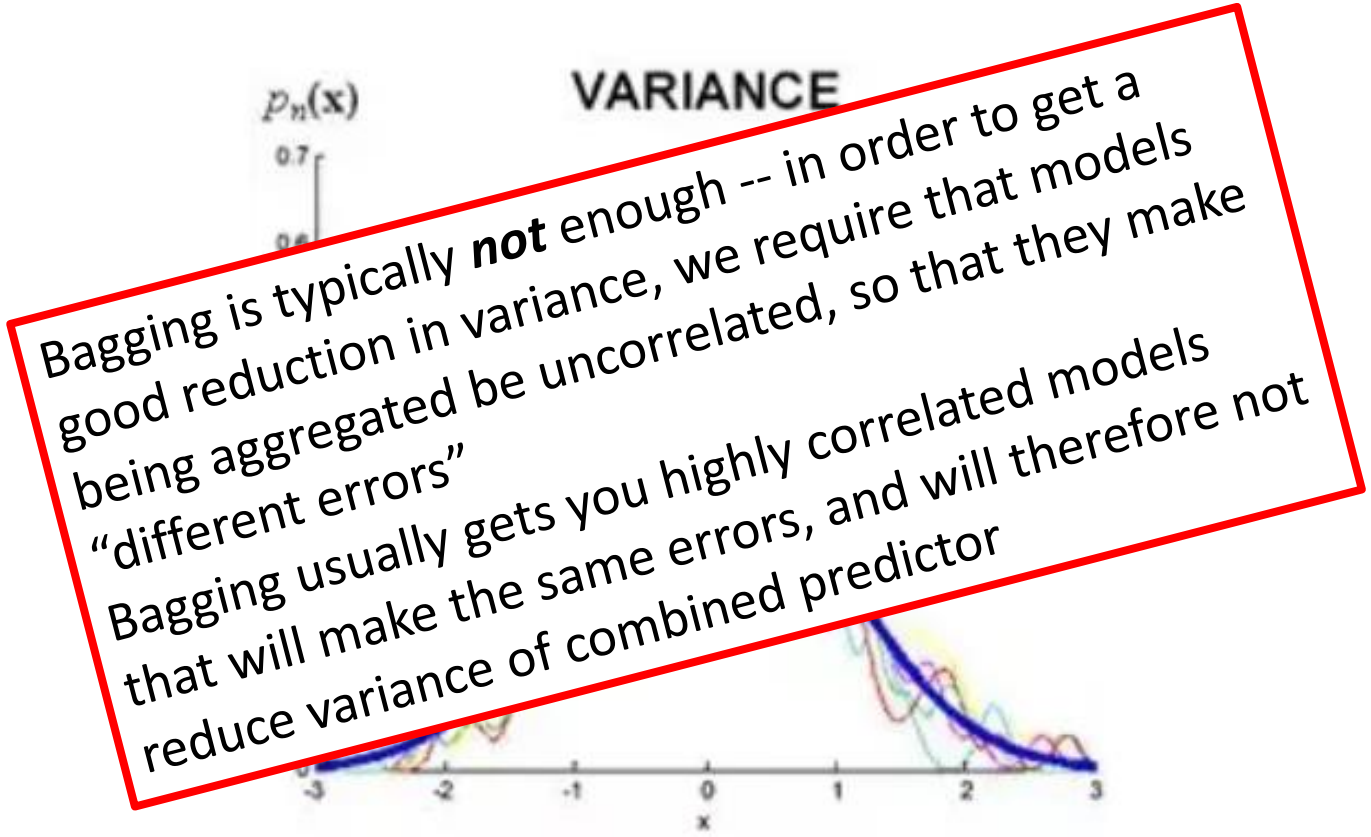
Bagging is typically not enough -- in order to get a good reduction in variance, we require that models being aggregated be uncorrelated, so that they make "different errors"

Bagging usually gets you highly correlated models that will make the same errors, and will therefore not reduce variance of combined predictor

Insight: By ensembling/bagging predictors by averaging (or aggregation of) many low bias, high variance predictors, we can reduce the variance while retaining low bias!

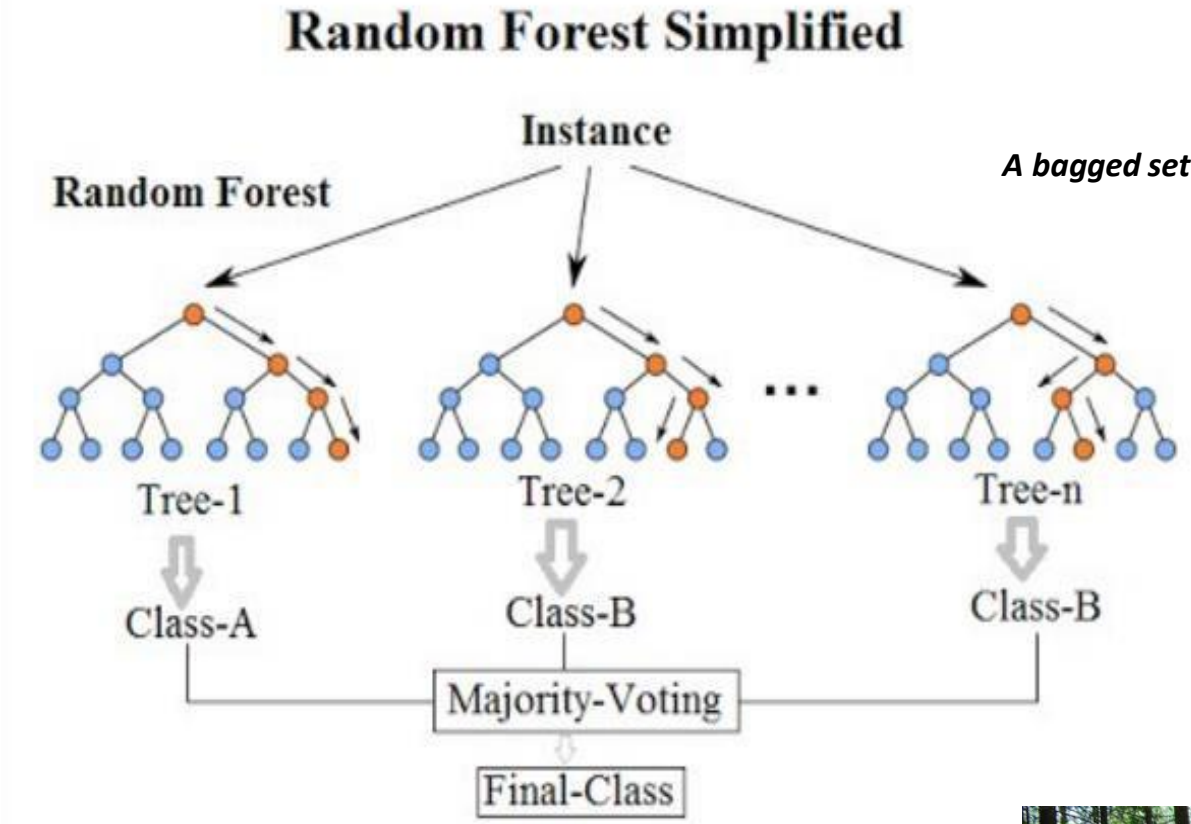


Insight: By ensembling/bagging predictors by averaging (or aggregation of) many low bias, high variance predictors, we can reduce the variance while retaining low bias!



Bagging is typically ***not*** enough -- in order to get a good reduction in variance, we require that models being aggregated be uncorrelated, so that they make "different errors"

Bagging usually gets you highly correlated models that will make the same errors, and will therefore not reduce variance of combined predictor



A bagged set of "randomly split" DTs = Random Forest

A Random Forest!

A Forest!

The Random Forest Algorithm

Algorithm 1 Random Forest

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

```
1  function RandomForest(S, F)
2      H ← ∅
3      for i ∈ 1, ..., B do
4          S^(i) ← A bootstrap sample from S
5          h_i ← RandomizedTreeLearn(S^(i), F)
6          H ← H ∪ {h_i}
7      end for
8      return H
9  end function
10 function RandomizedTreeLearn(S, F)
11     At each node:
12         f ← very small subset of F
13         Split on best feature in f
14     return The learned tree
15 end function
```

AdaBoost
(Freund and Schapire, 1996)

Ensemble Learning Algorithm

By far the most popular; often "tried first"

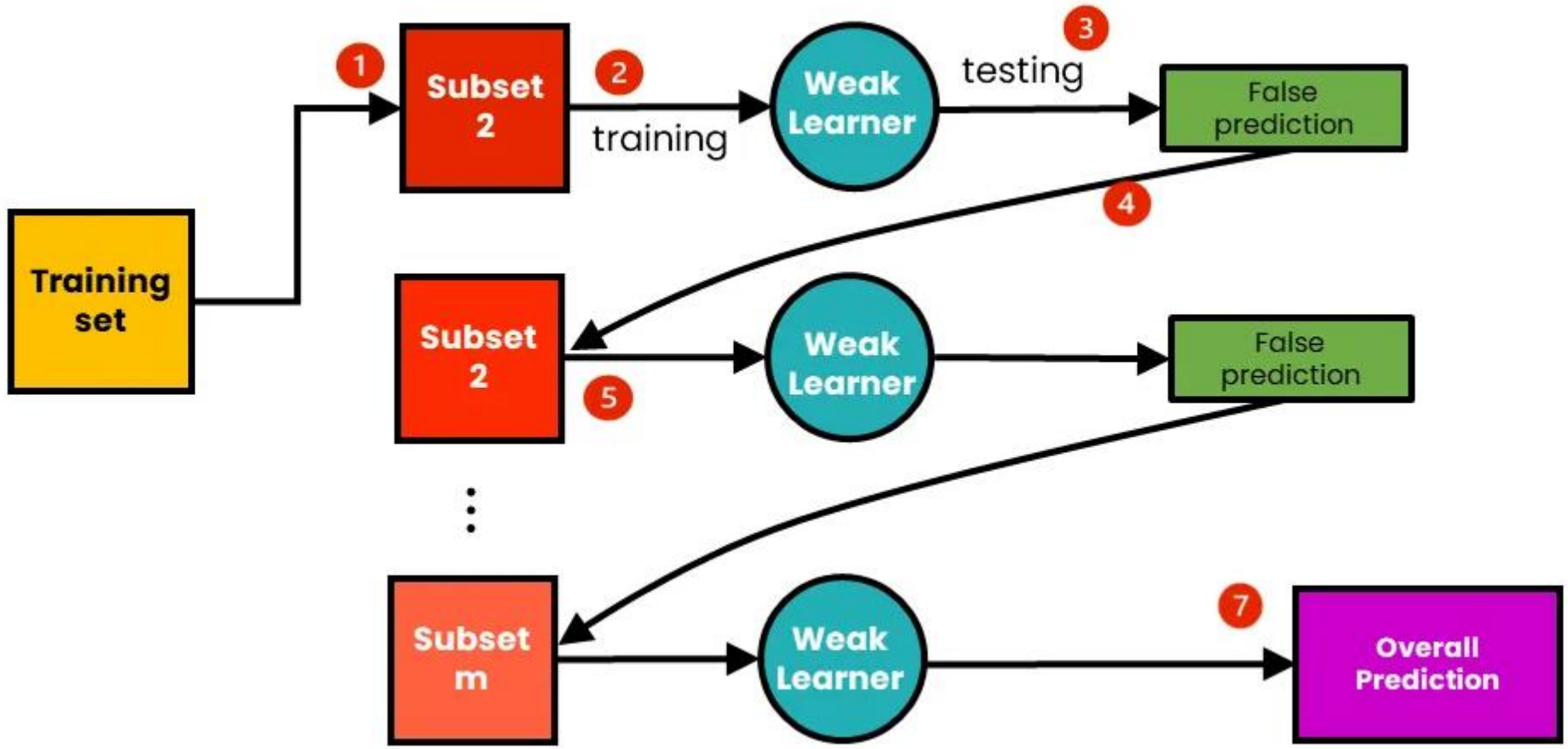
Strategy:

Create an ensemble of functions (e.g. classifiers) making decisions by weighted majority vote

- Higher accuracy -> higher vote weight

Bias training by adapting function to the training examples for which the system frequently fails

- Higher error example -> higher likelihood of example selection


Training set
1
Subset 2
2
training
Weak Learner
3
testing
False prediction
4
Subset 2
5
Weak Learner
False prediction
Subset m
Weak Learner
7
Overall Prediction

Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) **for** $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Eq. 8.34)
(6) **if** $error(M_i) > 0.5$ **then**
(7) go back to step 3 and try again;
(8) **endif**
(9) **for** each tuple in D_i that was correctly classified **do**
(10) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(11) normalize the weight of each tuple;
(12) **endfor**

Notice we keep sampling a classifier until we get a decent weak learner

To use the ensemble to classify tuple, X:

(1) initialize weight of each class to 0;
(2) **for** $i = 1$ to k **do** // for each classifier:
(3) $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(X)$; // get class prediction for X from M_i
(5) add w_i to weight for class c
(6) **endfor**
(7) return the class with the largest weight;

AdaBoost: Properties

"Perfect" Learning of the Training Data

For large enough K, If learning algorithm *L* is a *weak learner* (i.e. always produces a result better than chance), AdaBoost returns hypothesis *h* with no errors on the training data

Avoids Overfitting

As K increases, test set performance may increase even after training error reaches 0

This is *not* normally the case for learning algorithms which often have the risk of *overfitting* the data

Reason for this not completely understood

- Approximates Bayesian learning, which is optimal?

Example Experiment

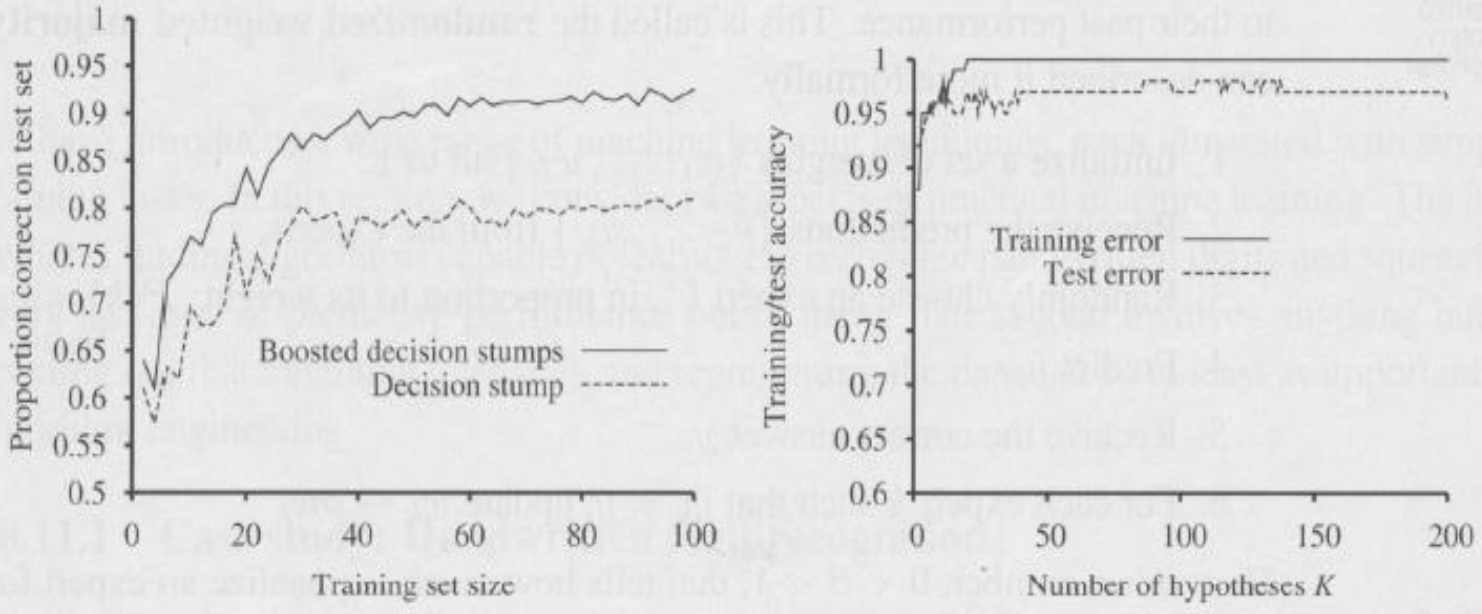


Figure 18.35 (a) Graph showing the performance of boosted decision stumps with $K = 5$ versus unboosted decision stumps on the restaurant data. (b) The proportion correct on the training set and the test set as a function of K, the number of hypotheses in the ensemble. Notice that the test set accuracy improves slightly even after the training accuracy reaches 1, i.e., after the ensemble fits the data exactly.

QUESTIONS?