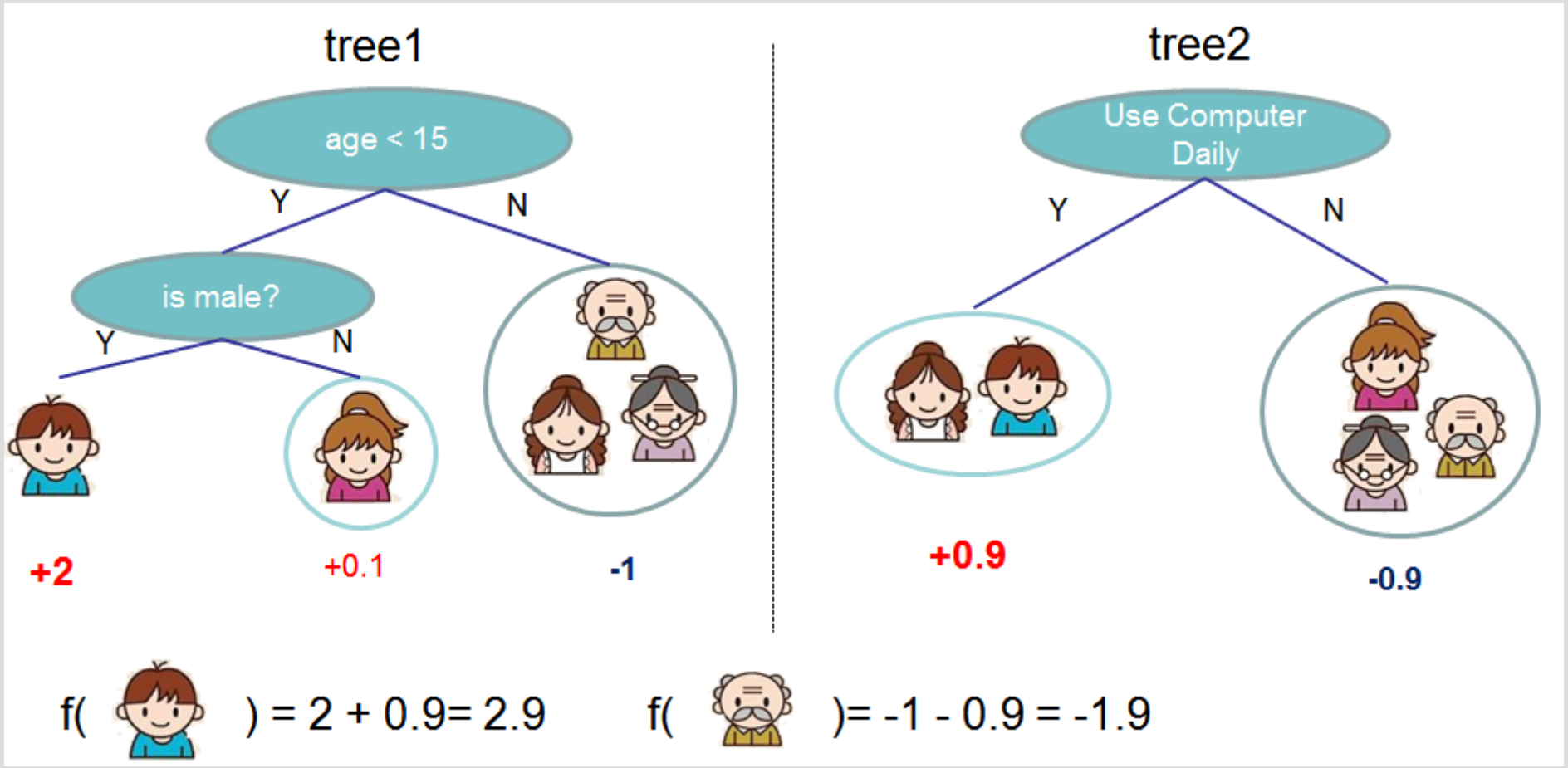
Ensemble Learning (Ch. 18.3-18.4, 18.10)



Overfitting

To avoid overfitting, we typically split the examples into a training and test sets

We only use the training sets to generate learning, then use the test set to estimate (called cross-validation)

This has the downside that if you have few examples, you are not using the test set as part of the learning algorithm

Overfitting

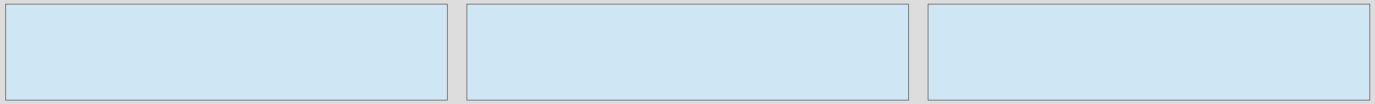
One way around this is k-fold cross-validation, where examples are split into k subsets

You then do k separate learning attempts, each time leaving one subset out of training as test

You then average (mean) the accuracy over these k learning attempts as a better estimate of the overall algorithm accuracy

Overfitting

For example, 3-fold cross-validation...
Data: (break into sets of three)



d_1 d_2 d_3

h are different learned decision trees or w/e learning alg

h_1 = learn on d_1&d_2... test on d_3=77% accuracy

h_2 = learn on d_1&d_3... test on d_2=75% accuracy

h_3 = learn on d_2&d_3... test on d_1=90% accuracy

Overall accuracy estimate = (77+75+90)/3

Overfitting

In general, if you have an algorithm that has parameters for the learning...

You **should not** use the test set to adjust the parameters (this is purpose defeating)

However, it could still be useful to measure parameters... so we could split the training data into subsets: sub-training and validation

Overfitting

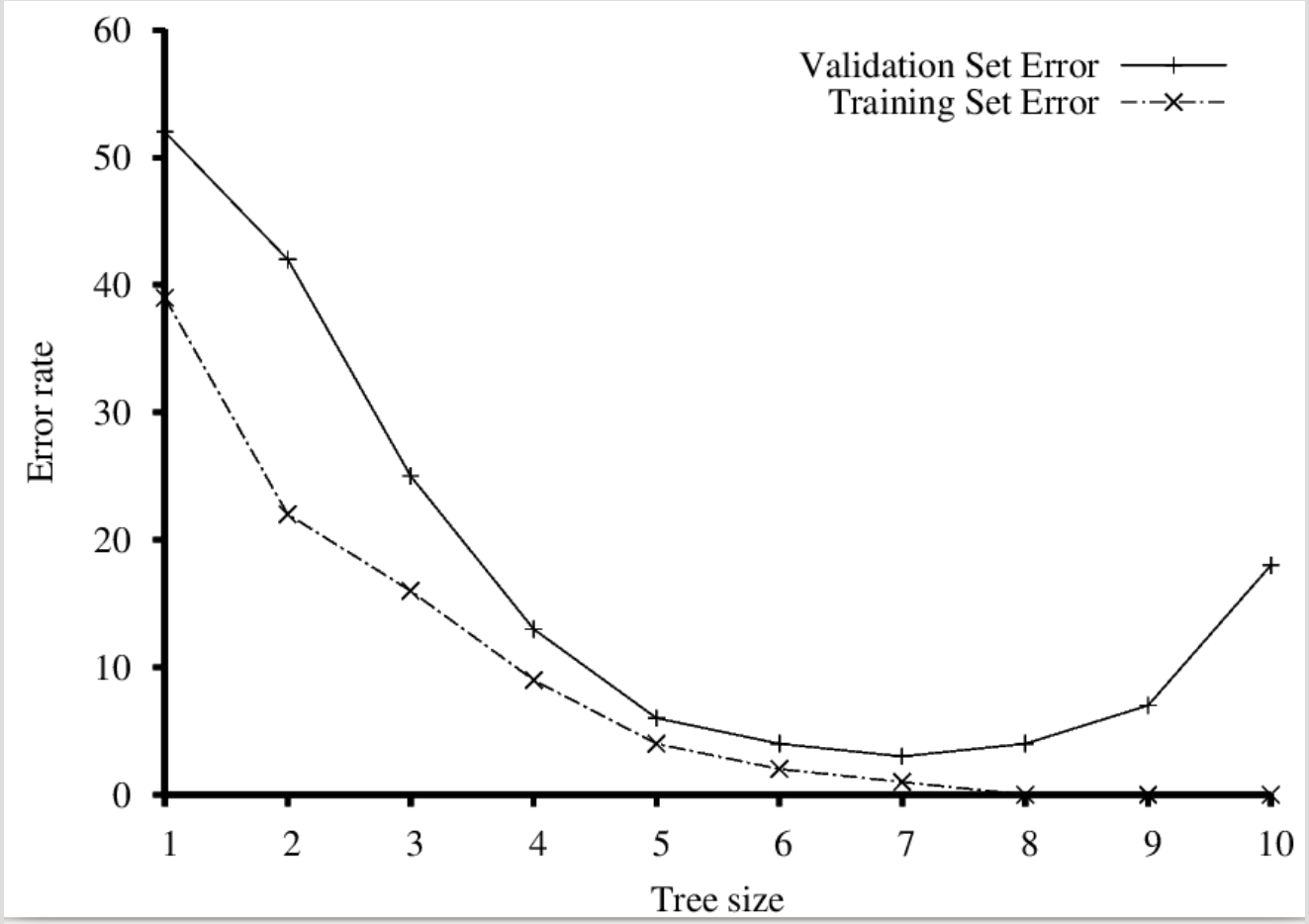
You can then use the “sub-training” set to train on and “validation” set to measure

For example, we could say the “size” of a decision tree is how many attributes are used (tree is generated in approximately same way, except in a BFS rather than DFS manner)

Then we use validation to estimate when “size” starts to overfit (i.e. find best param.)

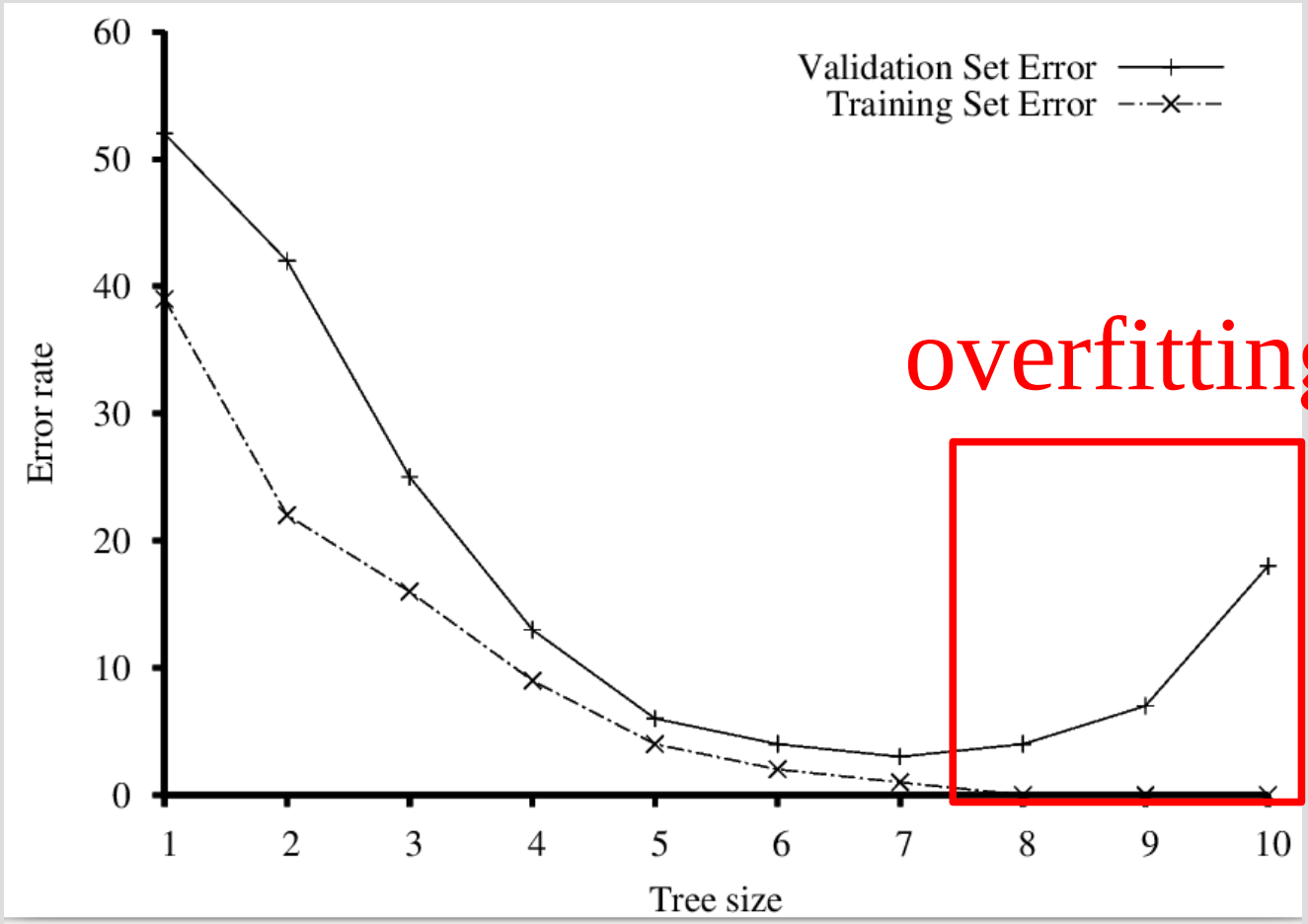
Overfitting

A “typical” learning algorithm would look something like this:



Overfitting

So we would guess that size=7 is optimal, then use the full training set (sub+val) to learn



Ensemble Learning

If we have multiple algorithms for predicting, we can use them together to get a better result than any individual prediction

This method is called ensemble learning, and there are a number of ways to do this

Take a simple example: You have three algs, all with 80% accuracy... If you use majority vote, what is the overall accuracy?

Ensemble Learning

A common ensemble technique is called boosting, where you weight training examples

This allows you to put more weight on data that is often misclassified, so when making multiple learning algs. can focus learning

This helps ensure there is not a “gap” in your learning, one such algorithm is “AdaBoost” (Adaptive Boosting)

Ensemble Learning

AdaBoost: (Set w[data] array = 1/size(data))
Loop k times: (k = number of classifiers)
    error = 0
    h[k] = learn from weighted data
    Loop over data:
        if h[k] misclassifies: error += w[data]
    Loop over data:
        if h[k] correct: w[data] *= error/(1-error)
    z[k] = log[(1-error)/error]
return weighted-vote(h,z) // z is weight

Ensemble Learning

AdaBoost has a nice property that if all of your classifiers are “weak learners” (accuracy > 50%)

Then enough k (number of classifiers), AdaBoost has 100% accuracy on training set

Though, obviously, this does not extend to 100% accuracy in practice (or on test set)

Online Learning

So far we have assumed that learning data has been i.i.d. (independent and identically distributed), which is often fine...

Independent(examples don't effect each other):

$$P(E_j | E_{j-1}, E_{j-2}, ...) = P(E_j)$$

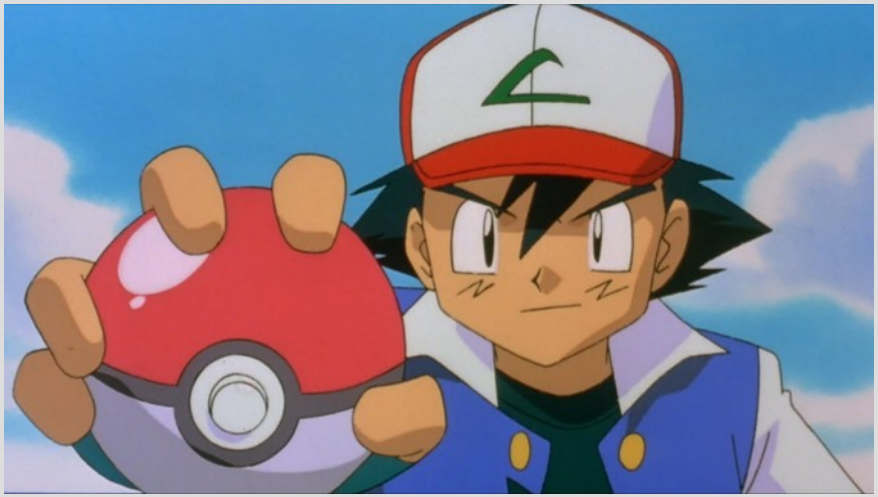
j^{th} example

Identically distributed (no example bias):

$$P(E_j) = P(E_{j-1}) = P(E_{j-2}) = ...$$

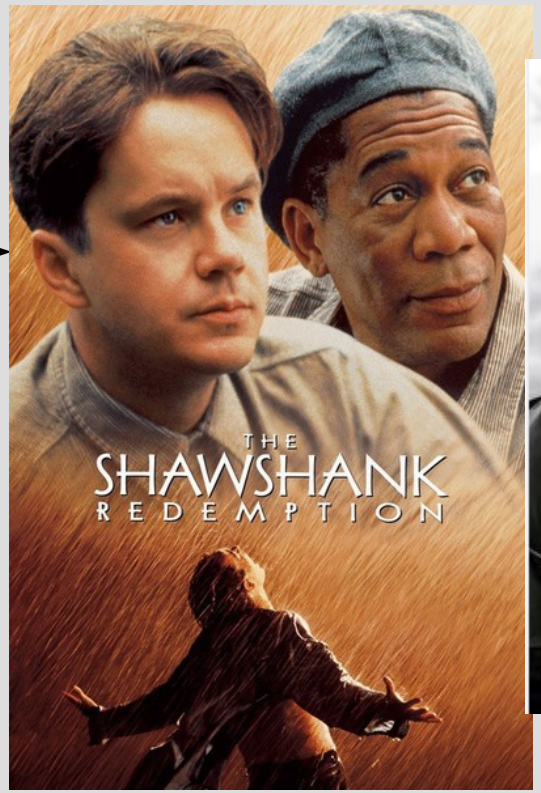
Online Learning

This is not always the case... for example your movie preferences has probably changed since you were a kid (not independent)



Kid

Now



Online Learning

We can still do “learning” even if examples change over time (as long as not too fast)

This is called online learning as we cannot just wait until all examples are given

Instead, the algorithm needs to be more iterative as it needs to:
(1) change over time (disregard old data)
(2) cannot recompute from scratch

Online Learning

One such algorithm is randomized weighted majority algorithm:

Assume you have “K” classifiers
Initialize weight of each classifier as 1
(1) Choose random classifier by:

$$P(classifier_x) = \frac{w(classifier_x)}{\sum_i w(classifier_i)}$$

(2) Predict using chosen classifier
(3) Get real result and adjust any incorrect classifiers by: $w(classifier_x) = \beta \cdot w(classifier_x)$

Online Learning

We can actually get a bound on how many incorrect classifications, M, we get:

$$M < \frac{(M^*)\cdot\ln(1/\beta)+\ln(K)}{1-\beta}$$

... where M* is the best classifier,
K is the number of classifiers,
β is a parameter (up to us), but $0 < \beta < 1$

Here, β determines how fast we adapt to changes (β near 0 is for faster changes)

Online Learning

If we set β close to 1, then we can get asymptotically close to the best classifier, M*

However, there is a trade-off, as β close to 1 also means we will "try" poor classifiers more before we give up on them

more "long term" as closer to M*, but have to pay a larger "constant" mistake penalty

For example, assume K=10...

β=0.25:

$$M < 1.848 \cdot M^* + 3.070$$

β=0.8:

$$M < 1.116 \cdot M^* + 11.51$$