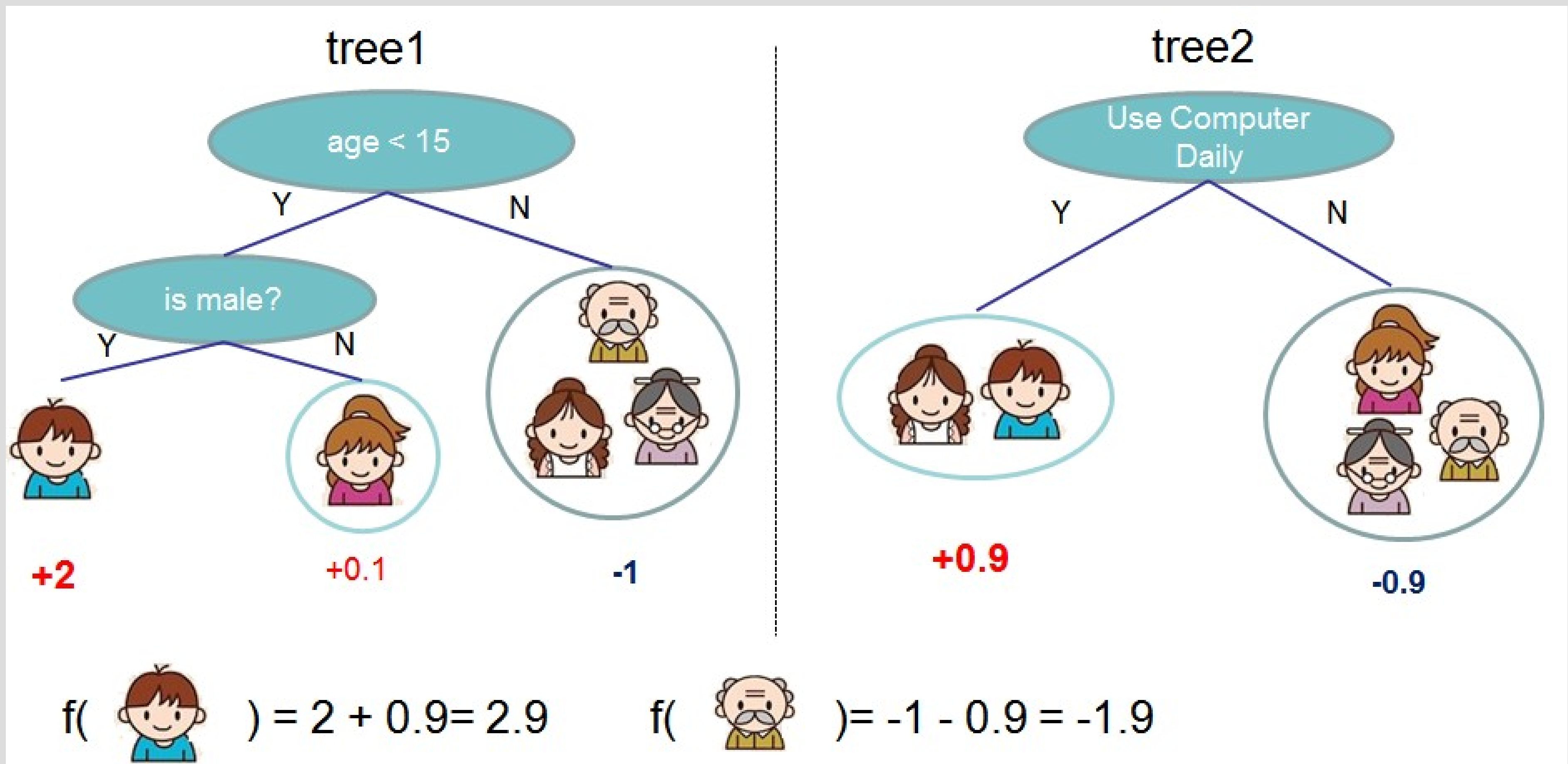
Ensemble Learning (Ch. 18.3-18.4, 18.10)



Making Trees

From last time:

Example	A	B	C	D	E	Ans
1	T	low	big	twit	5	T
2	T	low	small	FB	8	T
3	F	med	small	FB	2	F
4	T	high	big	snap	3	T
5	T	high	small	goog	5	F
6	F	med	big	snap	1	F
7	T	low	big	goog	9	T
8	F	high	big	goog	7	T
9	T	med	small	twit	2	F
10	F	high	small	goog	4	F

Entropy

Recap, entropy equation:

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

What is the best attribute to split on?

Example	A	B	C	D	E	Ans
1	T	low	big	twit	5	T
2	T	low	small	FB	8	T
3	F	med	small	FB	2	F
4	T	high	big	snap	3	T
5	T	high	small	goog	5	F
6	F	med	big	snap	1	F
7	T	low	big	goog	9	T
8	F	high	big	goog	7	T
9	T	med	small	twit	2	F
10	F	high	small	goog	4	F

Entropy

Technically, the “example” attribute gives us 100% accuracy...

This is obviously wrong as this “attribute” cannot be extrapolated, but it shows:

(1) Attributes with more values tend to be more prioritized

(2) We are making an implicit assumption that attributes are related to output

Statistics Rant

Next we will do some statistics

\rantOn
Statistics is great at helping you make correct/accurate results

Consider this runtime data, is alg. A better?

A	5.2	6.4	3.5	4.8	3.6
B	5.8	7.0	2.8	5.1	4.0

Statistics Rant

Not really... only a 20.31% chance A is better (too few samples, difference small, var large)

A	5.2	6.4	3.5	4.8	3.6
B	5.8	7.0	2.8	5.1	4.0

Yet, A is faster 80% of the time... so you might be mislead in how great you think your algorithm is
\rantOff

Decision Tree Pruning

We can frame the problem as: what is the probability that this attribute just randomly classifies the result

Before our “A” split, we had with 5T and 5F
A=T had 4T and 2F

So 6/10 of our examples went A=T...
if these 6/10 randomly picked from the 5T/5F we should get 5*6/10 T on average randomly

Decision Tree Pruning

Formally, let p=before T=5, n=before false=5

$p_{A=T}$=T when "A=T" = 4

$n_{A=F}$=F when "A=T" = 2

... and similarly for $p_{A=F}$ and $n_{A=F}$

Then we compute the expected "random" outcomes:

$$\hat{p}_k = p \cdot \frac{p_k + n_k}{p + k}$$

$$\hat{n}_k = n \cdot \frac{p_k + n_k}{p + k}$$

5 * 6/10 = 3 T on average by "luck"

Decision Tree Pruning

We then compute (a “test statistic”):

$$x = \sum_k \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

$$= \frac{(p_{A=T} - \hat{p}_{A=T})^2}{\hat{p}_{A=T}} + \frac{(n_{A=T} - \hat{n}_{A=T})^2}{\hat{n}_{A=T}}$$

$$+ \frac{(p_{A=F} - \hat{p}_{A=F})^2}{\hat{p}_{A=F}} + \frac{(n_{A=F} - \hat{n}_{A=F})^2}{\hat{n}_{A=F}}$$

$$= \frac{(4-3)^2}{3} + \frac{(2-3)^2}{3} + \frac{(1-2)^2}{2} + \frac{(3-2)^2}{2}$$

$$= 1.667$$

Decision Tree Pruning

Once we have “x” we can jam it into the χ^2 (chi-squared) distribution:

$$\chi^2(1)(x) = \chi^2(1)(1.667) = 0.19671$$

= [possible attribute values] -1 (degrees of freedom)

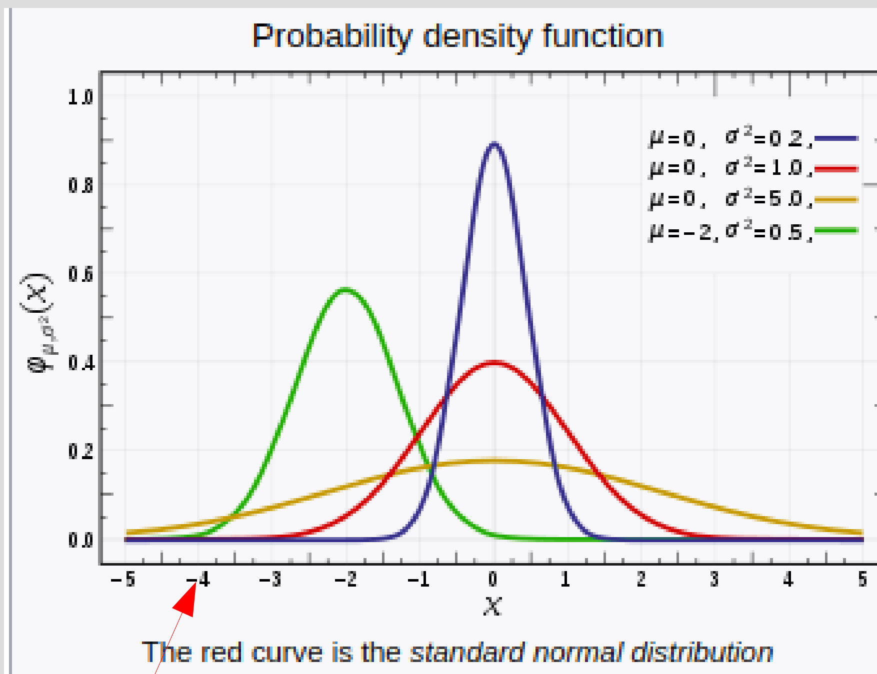
So there is a 19.67% chance this variable is just “randomly” assigning... so we might want to not use “A” here (other places maybe)

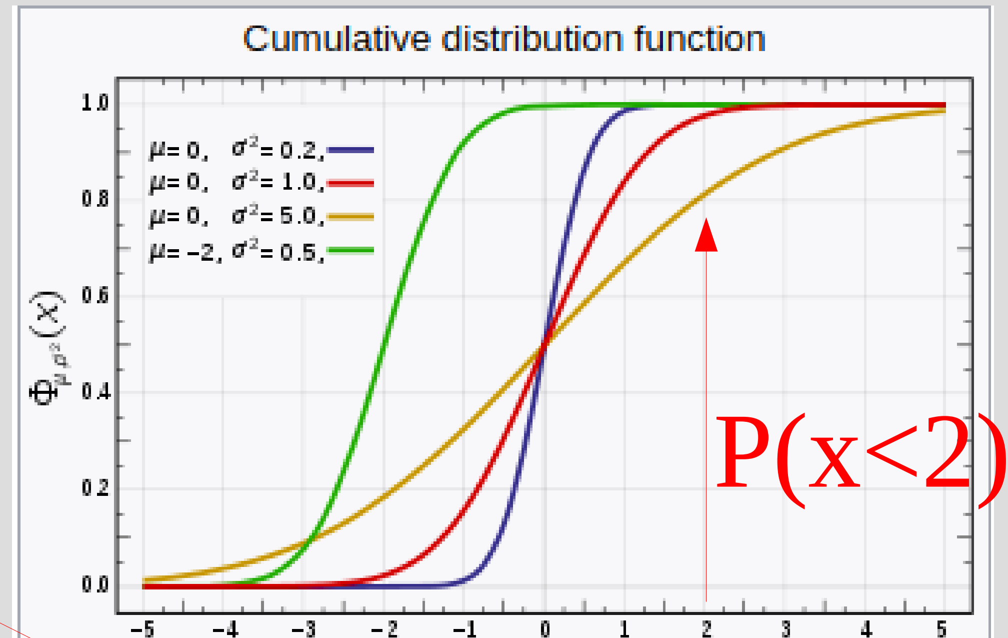
for T/F happens when x>3.841

The “typical” threshold we look for is 5% of being “random”... if so, could collapse node

What is this x^2 thing?

I think most people are familiar with the "bell"/normal/Gaussian distribution:





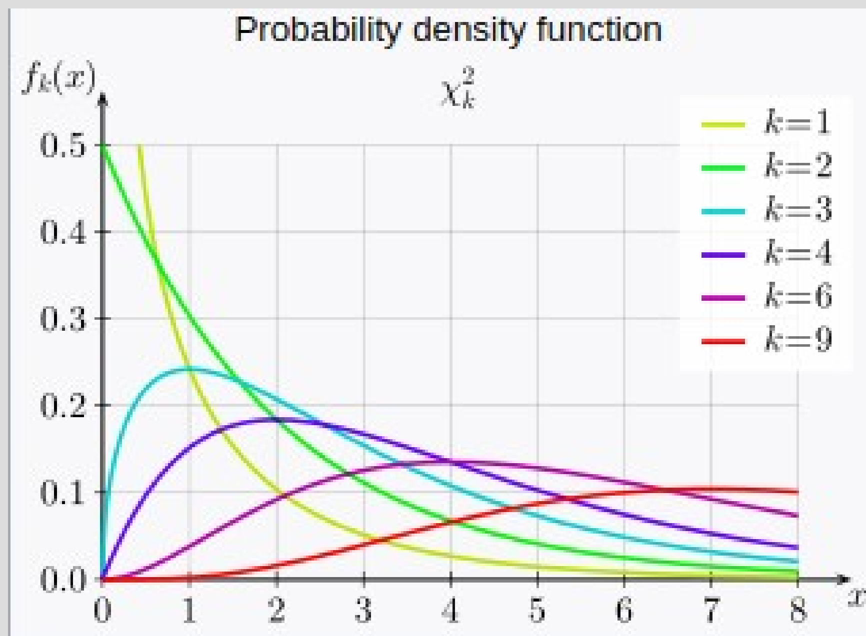
$N(\mu,\sigma^2)(x)$ needs 2 paramters: μ,σ

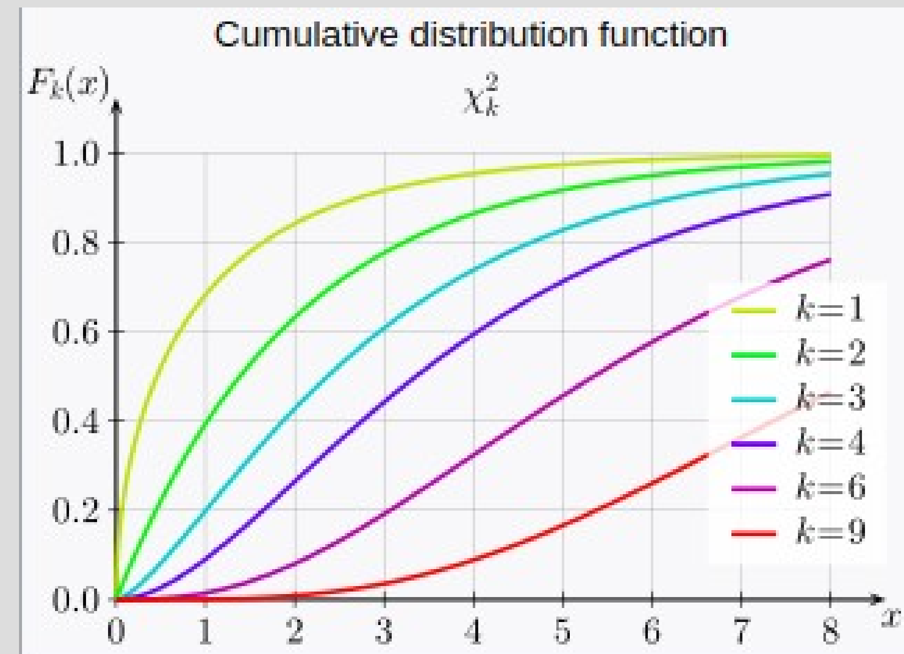
$$\int_{-\infty}^{x} P(z)dz = P(x < z)$$

What is this χ^2 thing?

a statistics thing... out of the scope of this course

χ^2 is just a different distribution that only requires 1 parameter (degrees of freedom)





Written both as $\chi^2(k,x)$ or $\chi^2(k)(x)$

Decision Tree Pruning

So, suppose you had a "bad" attribute (conflicting examples/inputs in this case):

more T than F so just "guess" T

Notice the attribute "X" is not really helping (at all...), so you could just remove it

Decision Tree Pruning

When should you apply this pruning?

Why?

Decision Tree Pruning

You should only do this type of pruning **after** building the whole tree

If you do it at the start, you can miss out on interactions that require multiple attributes

A simple example is “P xor Q”
- No matter which attribute you pick (P or Q) first, it will seem just “random”

Complications

There are a number of complications:

(1) Attributes with more possible “values” seem better than they are

(2) Integers/doubles you typically want to threshold to remove issue of (1)

(3) If you want a continuous output rather than a classification, your leaf needs to be a function rather than a single value

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

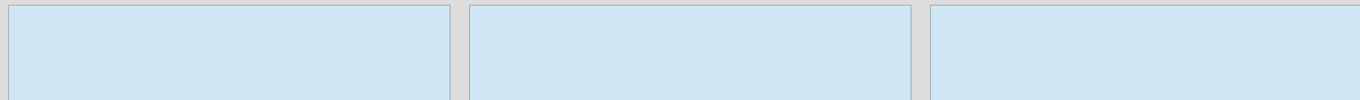
You then do k separate learning attempts, each time leaving one subset out of training as test

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

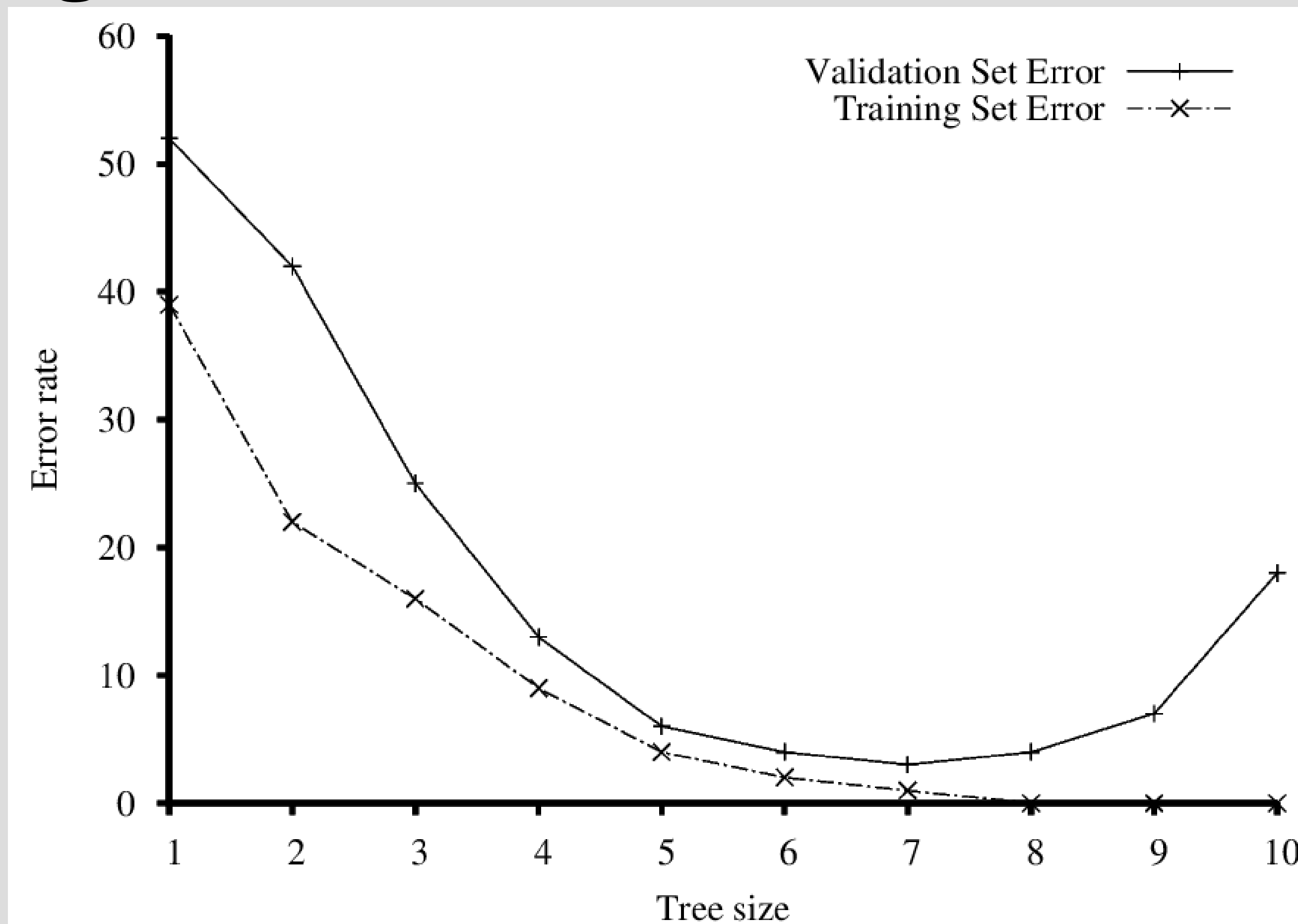
You can then use the “sub-training” set to train on and “validation” set to measure

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.)

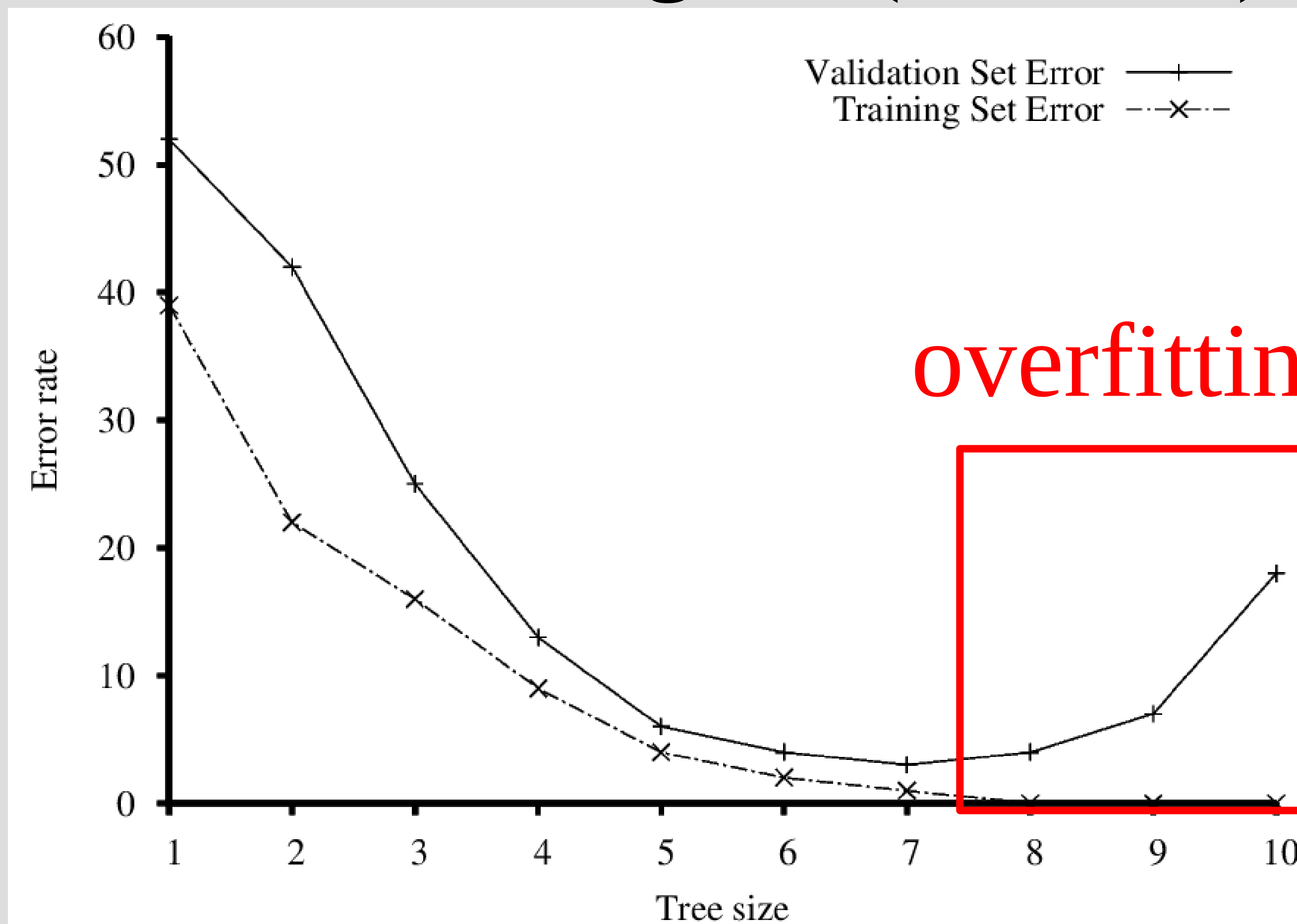
Overfitting

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):

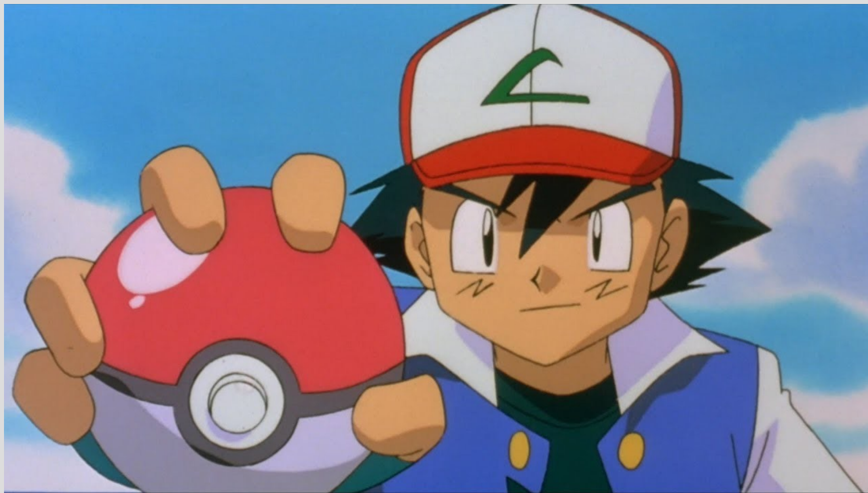
$$P(E_j | E_{j-1}, E_{j-2}, ...) = P(E_j)$$

Identically distributed (no example bias):

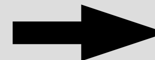
$$P(E_j) = P(E_{j-1}) = P(E_{j-2}) = ...$$

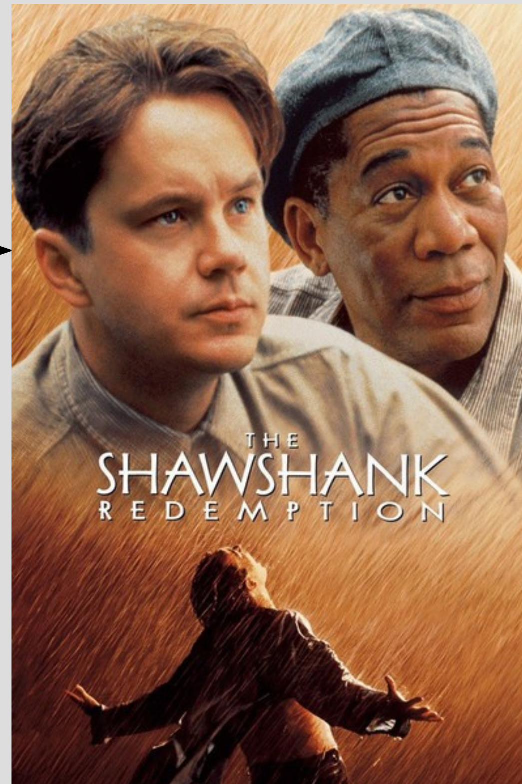
Online Learning

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 <u>online learning</u> 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 < β < 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:

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