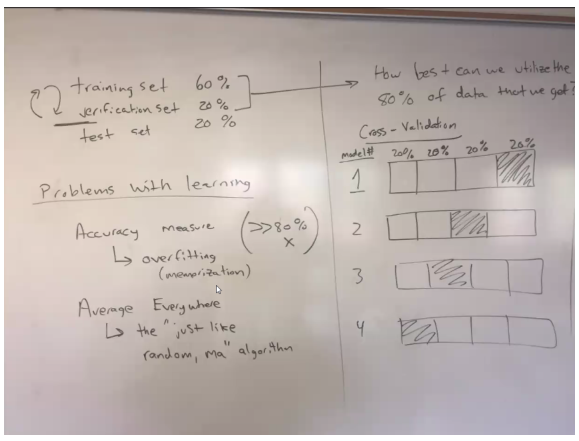
Machine Learning Notes

- fundamentally a taste of classification, where you identify the category of a new piece of data based upon some training data that was previously publised
    1. given data from a training set (60%), we develop a model which we use to predict the results of new data we don't have yet
    2. apply the model from 1 to a verification set (20%) (the test set), where we text the accuracy of our prediction. We know the answer for the verification set, but we do not train on them [1]teps 1 and 2 occur in a cycle
    3. apply the model to the test set (20%), which is hidden/secret/safe from developers
- success: if you were to be given an answer, your model would match the answer
- models that are bad:
    - memorizing the verification set



training set 60%
verification set 20%
test set 20%

How best can we utilize the 80% of data that we got?

Problems with learning

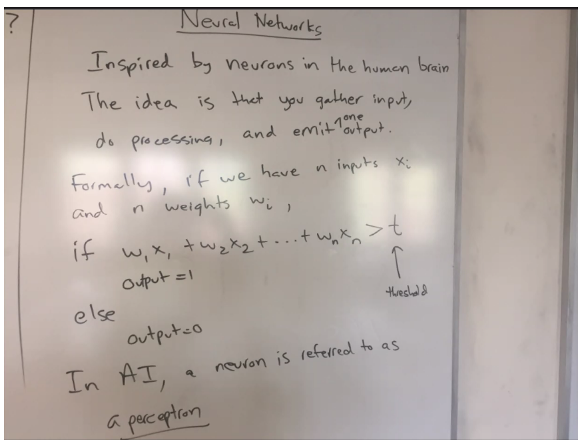
Accuracy measure ($>>80\%$ ×)
↳ overfitting (memorization)

Average Everywhere
↳ the "just like random, ma" algorithm

Cross - Validation

model#	20%	20%	20%	20%
1				▨
2			▨	
3		▨		
4	▨			

        * this may happen by accident if we have access to all of the data
        * this is why some of the data should be hidden to the developer (the actual test set, not the DS110 set)
    - how best can we utalize the 80% of data that we got?
        * you can run these results through a second classifier
        * dave really likes neural networks



Neural Networks

Inspired by neurons in the human brain

The idea is that you gather input, do processing, and emit one output.

Formally, if we have n inputs x_i and n weights w_i,

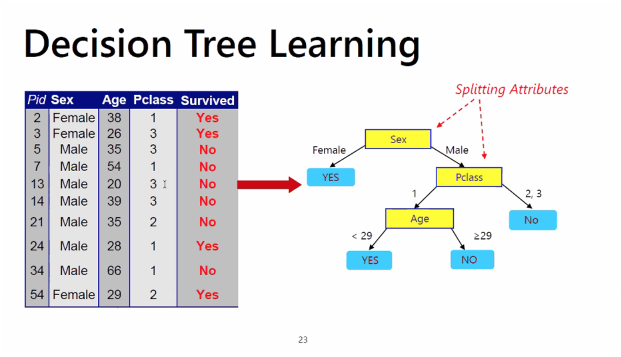
if $w_1x_1 + w_2x_2 + \ldots + w_nx_n > t$ ← threshold
    output = 1
else
    output = 0

In AI, a neuron is referred to as a perceptron

        * decisions trees are also a thing
            · basically think of 20 questions, we did an inclass example with the titanic dataset

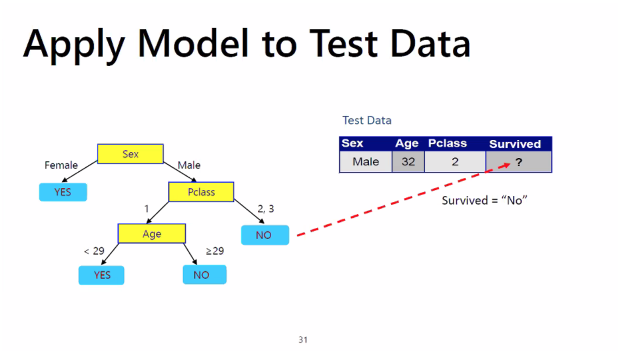
Decision Tree Learning

- you have some sort of splitting attribute, in this example: sex, age, pclass, survived
- you then follow the tree that based on the criteria you are givens



Decision Tree Learning

Pid	Sex	Age	Pclass	Survived
2	Female	38	1	Yes
3	Female	26	3	Yes
5	Male	35	3	No
7	Male	54	1	No
13	Male	20	3	No
14	Male	39	3	No
21	Male	35	2	No
24	Male	28	1	Yes
34	Male	66	1	No
54	Female	29	2	Yes



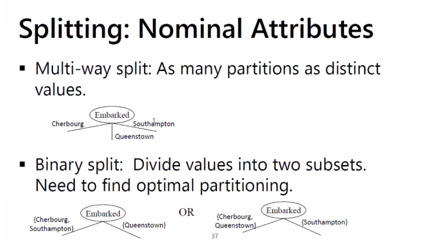
Apply Model to Test Data

Test Data

Sex	Age	Pclass	Survived
Male	32	2	?

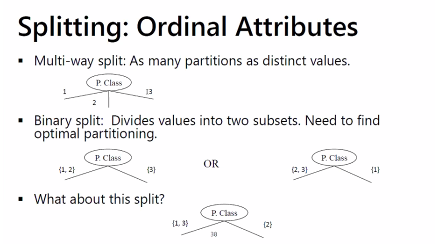
Survived = "No"

- it is important to note that the order of splitting factors matters, but sometimes they may produce the same results
- How do we get a tree?
  - exponentially many decision trees are possible
  - finding the optimal tree is infeasible
  - greedy methods find that near-optimal solutions do exist
    - ∗ greedy methods provide common sense solutions (i.e. travel south to go from butler to downtown)
    - ∗ could also not be optimal (i.e. cross a river going east to west by going south instead of traveling west to the bridge and then crossing)
- Tree Induction
  - Greedy strategy
    - ∗ split based on attribute test that optimizes a criterion
  - issues
    - ∗ how to split the records
      - · what attribute test condition?
      - · how to determine the best split?
      - · when do we stop?
    - ∗ attribute types
      - · nominal
      - · ordinal
      - · continuous
    - ∗ order of split
      - · 2-way split
      - · multi-way split
- example



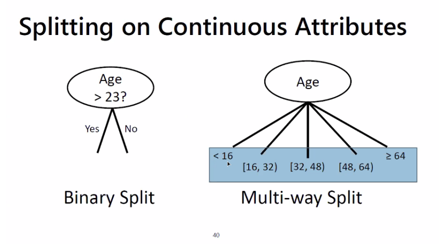
Splitting: Nominal Attributes

- Multi-way split: As many partitions as distinct values.
- Binary split: Divide values into two subsets. Need to find optimal partitioning.

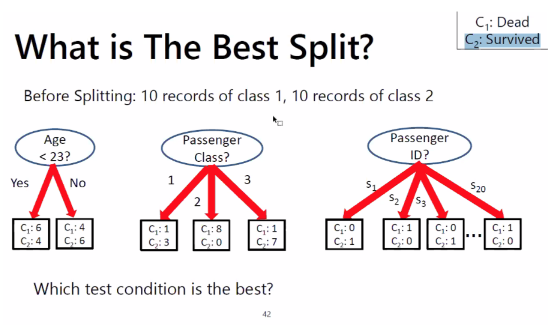


Splitting: Ordinal Attributes

- Multi-way split: As many partitions as distinct values.
- Binary split: Divides values into two subsets. Need to find optimal partitioning.
- What about this split?



Splitting on Continuous Attributes



What is The Best Split?

C_1: Dead
C_2: Survived

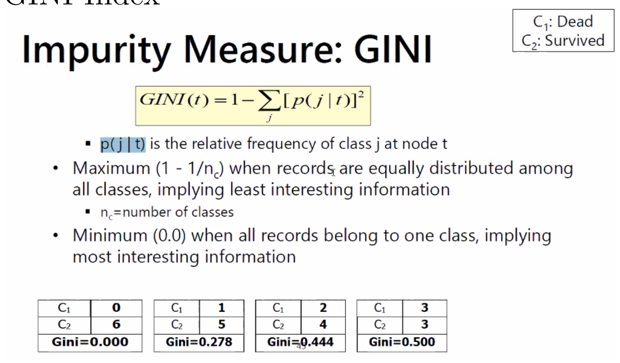
Before Splitting: 10 records of class 1, 10 records of class 2

Which test condition is the best?

- you want the split to be heavily skewed, so that you don't necessarily have to ask another question to have a good guess
- greedy approach
  - homogeneous class distribution preferred
- need a measure of node impurity - aka a way to define what an impure node is
- use node impurity to determine what nodes to start with
- start with the most pure nodes possible
- things you may want to specify:
  - the height and the width. If you make it too wide then you did not skew nodes well enough... if you make it too tall you have made too many decisions before you predict
  - you also may want to limit the branching factor
  - the smallest number of items in a category (node density)
  - how to do splits
  - GINI/entropy
- decision trees are just one gigantic if statement
- PROS
  - intuitive: easy interpretation for small trees
  - non parametric: incorportates both numeric and categorical attributes
  - fast: once rules are developed, prediction is rapid
  - robust to outliers: won't affect the majority classification
- CONS
  - overfitting: must be trained with great care, must put limits so that it does not overgrow — must prune the tree
  - rectangular classification: recursive partitioning of data may not capture complex relationships
  - hard to reconsider kinds of relationships – sometimes your classification may need to stretch across your partition lines
    - Example: age may not always do the same thing, so you either have to ask again about it later, or it doesn't do a good job of explaining

Measures of Node Impurity

- GINI Index



C$_1$: Dead
C$_2$: Survived

Impurity Measure: GINI

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

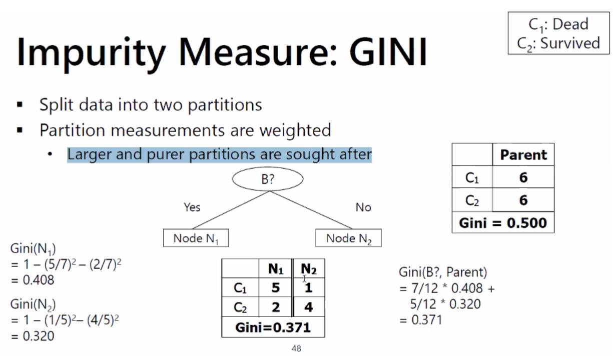
- p(j | t) is the relative frequency of class j at node t
- Maximum (1 - 1/n$_c$) when records are equally distributed among all classes, implying least interesting information
  - n$_c$=number of classes
- Minimum (0.0) when all records belong to one class, implying most interesting information

C$_1$	0
C$_2$	6
Gini=0.000	

C$_1$	1
C$_2$	5
Gini=0.278	

C$_1$	2
C$_2$	4
Gini=0.444	

C$_1$	3
C$_2$	3
Gini=0.500	



C$_1$: Dead
C$_2$: Survived

Impurity Measure: GINI

- Split data into two partitions
- Partition measurements are weighted
  - Larger and purer partitions are sought after

B?
Yes
No
Node N$_1$
Node N$_2$

	Parent
C$_1$	6
C$_2$	6
Gini = 0.500	

Gini(N$_1$)
= 1 – (5/7)^2 – (2/7)^2
= 0.408

Gini(N$_2$)
= 1 – (1/5)^2– (4/5)^2
= 0.320

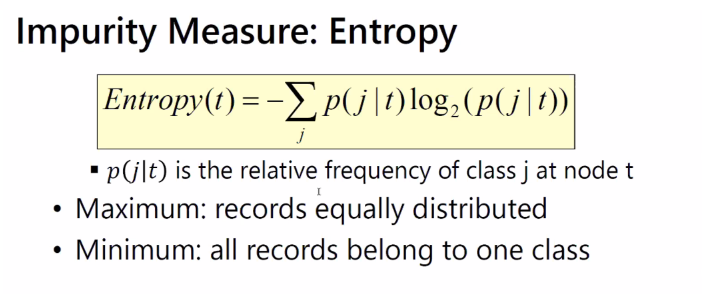
	N$_1$	N$_2$
C$_1$	5	1
C$_2$	2	4
Gini=0.371		

Gini(B?, Parent)
= 7/12 * 0.408 +
5/12 * 0.320
= 0.371

48

  - perform this calculation for every candidate B until you determine the best B to use for the first node in the tree

- Entropy
  - measures the amount of chaos in the decision and also what the information that it encloses
  - highly uncategorizable data works well with this
  - less sensitive to overfitting
  - slightly more obtuse, i.e. it isnt as clear why it classifies the way it does

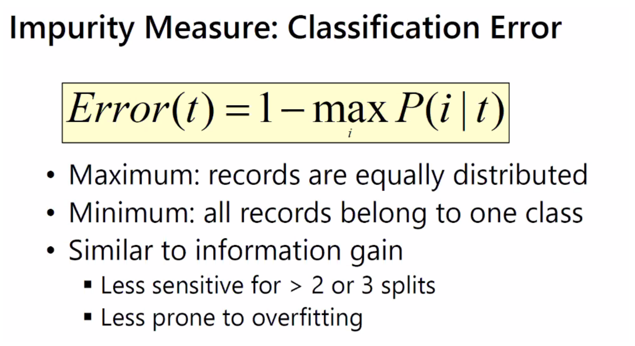


Impurity Measure: Entropy

$$Entropy(t) = -\sum_j p(j \mid t) \log_2 (p(j \mid t))$$

- $p(j|t)$ is the relative frequency of class j at node t
- Maximum: records equally distributed
- Minimum: all records belong to one class

- Misclassification error



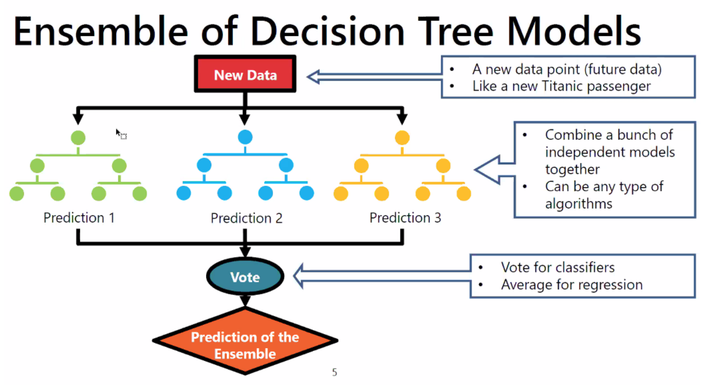
Impurity Measure: Classification Error

$$Error(t) = 1 - \max_i P(i \mid t)$$

- Maximum: records are equally distributed
- Minimum: all records belong to one class
- Similar to information gain
  - Less sensitive for > 2 or 3 splits
  - Less prone to overfitting

Ensemble Methods, Random Forests, and Boosting

- **Ensemble Methods**
  - improve model performance by combining *independent* multiple models
  - basic idea: monopolized on the monty carlo effect, if you're rolling the die of classification, the chance that all dies fail at the same time is really low
  - can be used for both classification and regression

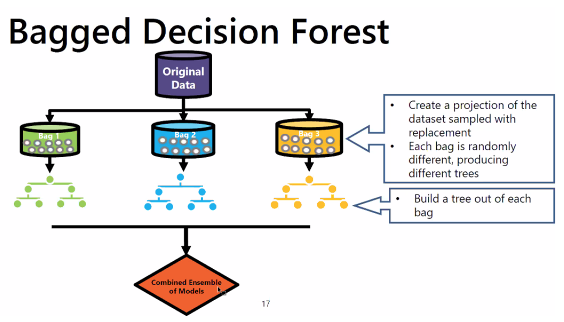

Ensemble of Decision Tree Models
New Data
A new data point (future data)
Like a new Titanic passenger
Combine a bunch of independent models together
Can be any type of algorithms
Prediction 1
Prediction 2
Prediction 3
Vote
Vote for classifiers
Average for regression
Prediction of the Ensemble
5

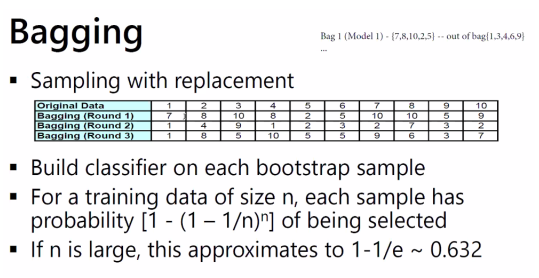
  - why does it work?
    - suppose there are 25 base classifiers
      - each classifier has error rate = 0.35
      - assume classifiers are independent
      - probability that the ensemble classifer makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i}(1-\varepsilon)^{25-i} = 0.06$$

11

  - two major methods:
    - bagging (all classifiers are created equal) – you bag until you get the same number as the original sample
      - reduces variance in estimate because you're constantly changing the sample that you're working with
      - prevents overfitting, because you cant have any guarentee that a specific piece of data may be in a given dataset
      - robust to outliers because they will, most likely, in general, be less likely to be selected than other stuff


Bagged Decision Forest
Original Data
Bag 1
Bag 2
Bag 3
Create a projection of the dataset sampled with replacement
Each bag is randomly different, producing different trees
Build a tree out of each bag
Combined Ensemble of Models
17



Bagging

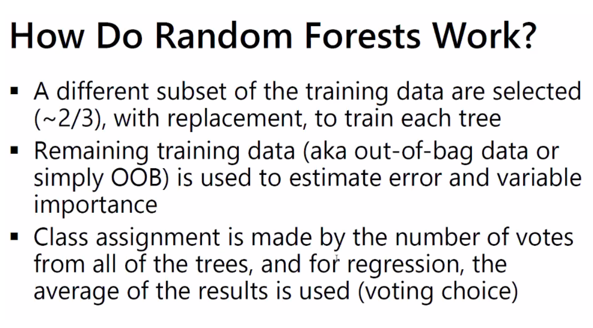
Bag 1 (Model 1) - {7,8,10,2,5} -- out of bag{1,3,4,6,9}
...

- Sampling with replacement

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

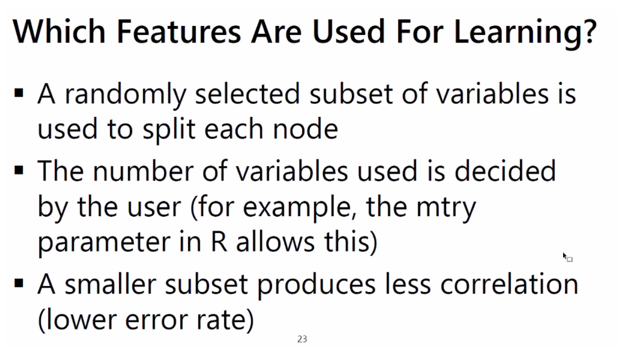
- Build classifier on each bootstrap sample
- For a training data of size n, each sample has probability $[1 - (1 – 1/n)^n]$ of being selected
- If n is large, this approximates to 1-1/e ~ 0.632

- **Random Forests**
  - an ensemble classifier using many decision tree models
  - can be used for classification or regression
  - two key ingredients needed:
    1. bagging (or sampling the data)
    2. random subset of features (or columns) when splitting nodes within trees – we say (randomly) what you're allowed to split on
  - it is important to note that a less correlation also produces a weaker model



How Do Random Forests Work?

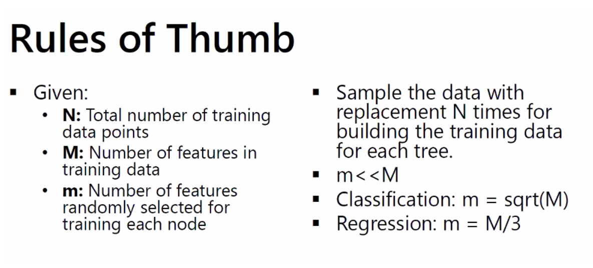
- A different subset of the training data are selected (~2/3), with replacement, to train each tree
- Remaining training data (aka out-of-bag data or simply OOB) is used to estimate error and variable importance
- Class assignment is made by the number of votes from all of the trees, and for regression, the average of the results is used (voting choice)



Which Features Are Used For Learning?

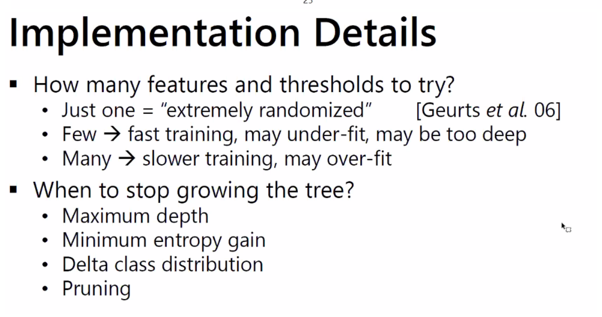
- A randomly selected subset of variables is used to split each node
- The number of variables used is decided by the user (for example, the mtry parameter in R allows this)
- A smaller subset produces less correlation (lower error rate)

23



Rules of Thumb

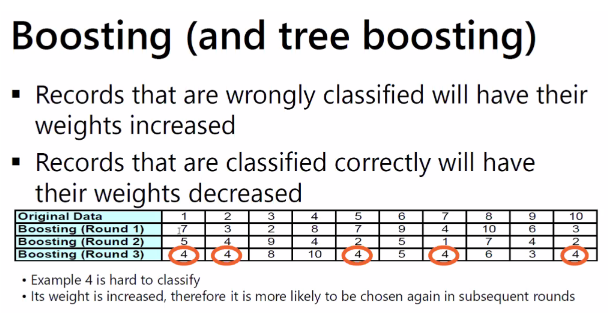
- Given:
  - **N:** Total number of training data points
  - **M:** Number of features in training data
  - **m:** Number of features randomly selected for training each node
- Sample the data with replacement N times for building the training data for each tree.
- m<<M
- Classification: m = sqrt(M)
- Regression: m = M/3



Implementation Details

- How many features and thresholds to try?
  - Just one = "extremely randomized" [Geurts *et al.* 06]
  - Few → fast training, may under-fit, may be too deep
  - Many → slower training, may over-fit
- When to stop growing the tree?
  - Maximum depth
  - Minimum entropy gain
  - Delta class distribution
  - Pruning

- **Boosting**
  - an iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
    - initially, all N records are assigned equal weights
    - then, when you get stuff wrong, you increase the weights of that stuff
    - do these things recursively M



Boosting (and tree boosting)

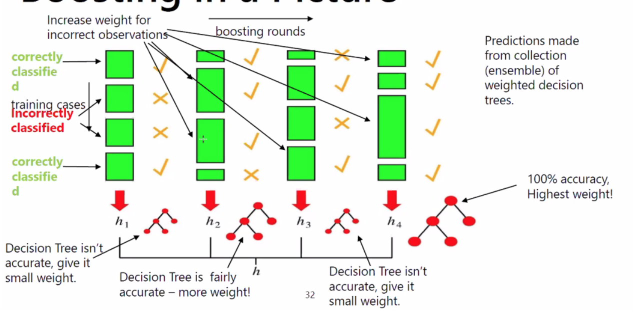
- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

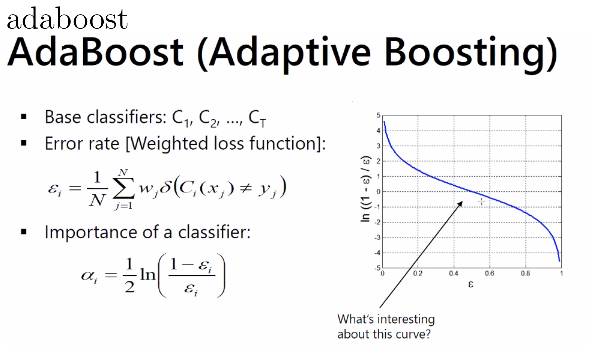
- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

  - boosting intuition
    - we adaptively weight each data case
    - data cases which are wrongly classified get high weight (the algorithm will focus on them)
    - each boosting round learns a new (simple) classifier on the weighed dataset
    - these classifiers are weighed to combine them into a single powerful classifier
    - classifiers that obtain low training error rate have high weight
    - we stop by monitroing a hold out set

Boosting in a Picture


Increase weight for incorrect observations
boosting rounds
correctly classified
training cases
incorrectly classified
correctly classified
Predictions made from collection (ensemble) of weighted decision trees.
100% accuracy, Highest weight!
h_1
h_2
h_3
h_4
h
Decision Tree isn't accurate, give it small weight.
Decision Tree is fairly accurate – more weight!
32
Decision Tree isn't accurate, give it small weight.

  - adaboost



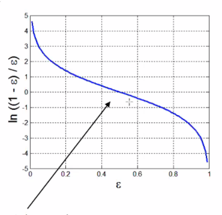
AdaBoost (Adaptive Boosting)

- Base classifiers: $C_1, C_2, \ldots, C_T$
- Error rate [Weighted loss function]:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



  - common misconception: a random forest and a boosted decision tree are *not* the same

Neural Networks

- Perceptrons
  - inspired by neurons in the human brain
  - basic idea: gather input, do some processing, and emit *one* output
  - formally, if we have n inputs x_i and n weights w_i, if $w_1x_1 + w_2x_2 + ... + w_nx_n > t$ where t is a given threshold, then output 1 else output 0
  - t, the threshold will be some value between 0 and 1
  - this whole thing is called a neuron/perception. In AI, a neuron is called a perceptron
  - each item has its own node, with certain weights on their path to the perceptron to determine one output
  - starting weights are given randomnly and then redetermined each time the model is run and something goes wrong
  - incremental gradient: the step size, in which we shall make modifications to the beginning weights
    - correction factor for wrong answers from the perceptron
    - we only adjust the things that contribute to us being wrong, in the student example, its the things with a value of 1
    - if we predicted too high, we decriment, if too low, we increment
  - what the fuck do you do if you can't converge?
  - CRITICAL FLAW: very simple functions that they cannot predict
    - cannot predict exclusive or correctly (LOL FAIL)
  - the lower t, the faster it trains, and the more likely it is to be wrong
  - this leads to "iterative deepening" in terms of training
  - in reality, the step size is discounted over time, i.e. it is not fixed
- A Generalized Neural Network
  - typically, you'll have a bipartide connected graph as your network
  - there is a 'hidden' layer of perceptrons in the middle
  - typically, you'll get multiple outputs at a time
  - to solve general problems on this neural network is...
    - perceptions now emit a real number [0,1]
    - these real numbers *propogate an error* and cause a problem
    - a technique called back propogation allows you to recompute based on future results to limit the effect of this error

Genetic Algorithms

- the basic idea is that you have some number of possible solutions called genomes. Then, we will "breed" (and possibly mutate) these genomes to create *offspring*. All offspring are scores using some scoring function, and the most promising ones become part of an updated genome set
- Examples:
  - Y#N# –¿ if I was top 10 last year, and I did not work hard, I will be top 10 this year
  - N#YN –¿ I did not get top ten last year, and I did work hard, and I did not drink –¿ top 10 this year
- Scoring function: test the rule for correctness on the table and represent it with a probablity of predictive success (for example, accuracy)
- Breeding: Y#N# and N#YN – arbitarily divide in half, then flop to create Y#YN and N#N#
- Mutation: Y#N# and N#YN and Y#YN and N#N# –¿ N#NY
  - lots of math goes into this, including concepts called simulated armealing and hill climbing (gradient descent)
- Example:
  - S = #Y## (2/6 probability), N#YN (3/6), YY## (4/6), #YY# (4/6), ###N (4/6)
  - you basically just pick two of them, and glue them together (breeding)
  - YY## and #YY# become YYY# and #Y##
  - do the following over and over again until stable
    1. breed
    2. mutate
    3. purge bad genomes from the genome set