Model Ensembles and Genetic Algorithms

Instructor: Jesse Davis

Slides from: Martine De Cock, Pedro Domingos, Russ Greiner, David Page, Jude Shavlik

Announcements

- Homework 3 is due next week
- Homework 2 will be returned next week and we’ll go over it at the start of class
- Lecture notes are available online

Outline

- Homework 3 Issues
- Model Ensembles
- Genetic Algorithms

Homework 3

- Remember to include code descriptions
- Ballpark accuracy is 97-99%
- Do extra credit if you haven’t already done it

Outline

- Homework 3 Issues
- Model Ensembles
- Genetic Algorithms

Motivation

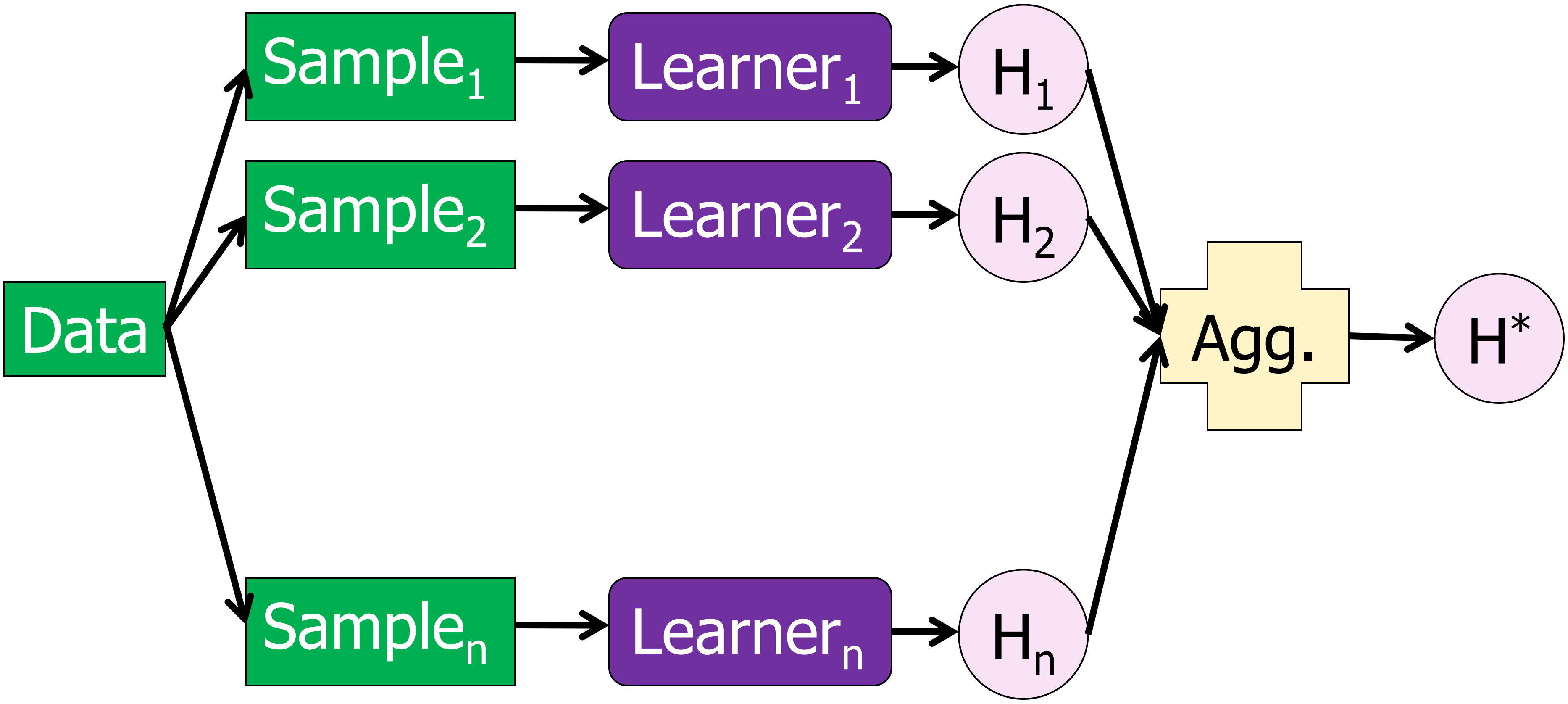
- One good learner produces one effective classifier
  - Could learning many classifiers help?
  - Why not learn { h1, h2, h3 }, then h*(x) = majority{ h1(x), h2(x), h3(x) }
- If classifiers make INDEPENDENT mistakes, then h* is more accurate!

Ensemble

- Assume: Independent errors (30%) and majority vote
- Probability that majority is wrong…

- Area under curve for ≥11 wrong is 0.026
- Order of magnitude improvement!

Overview



Data
Sample1
Sample2
Samplen
Learner1
Learner2
Learnern
H1
H2
Hn
Agg.
H*

Challenges

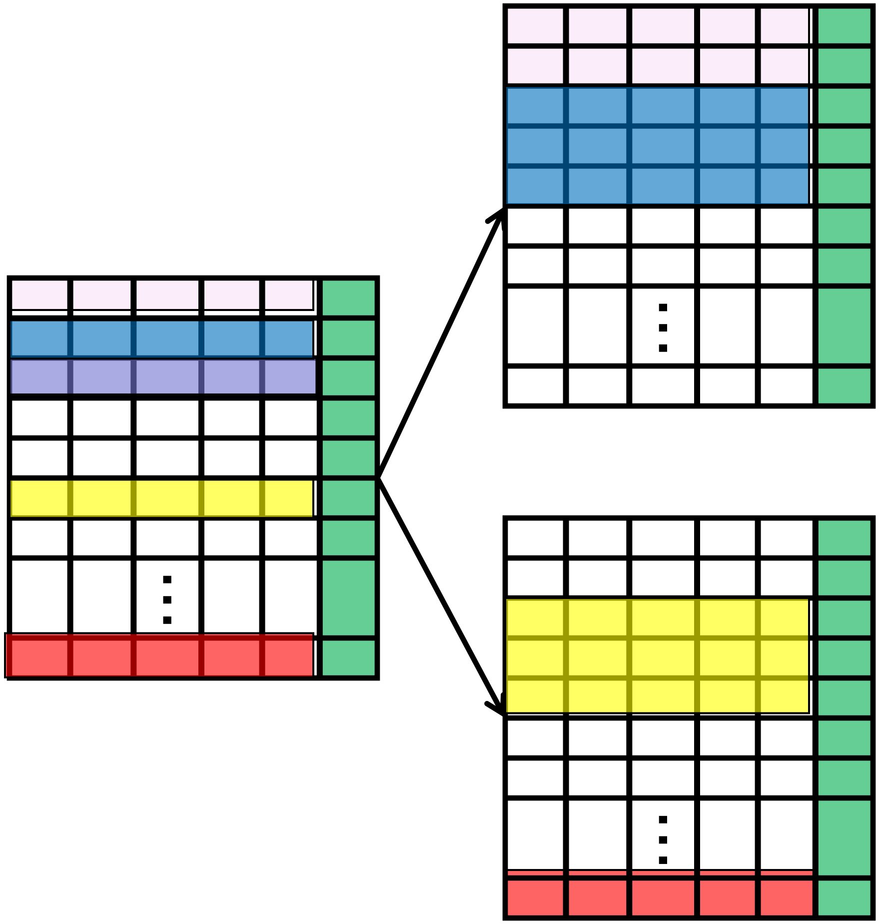
- How to generate the base classifiers?
  - Different learners?
  - Bootstrap samples?
  - Etc.
- How to integrate/combine them?
  - Average
  - Weighted Average
  - Instance-specific decisions
  - Etc.

Ensemble Approaches

- Sample data set
  - Bagging
  - Boosting
- Manipulate features
  - Input feature
  - Target features
- Add randomness
  - Data
  - Algorithm
- Stacking

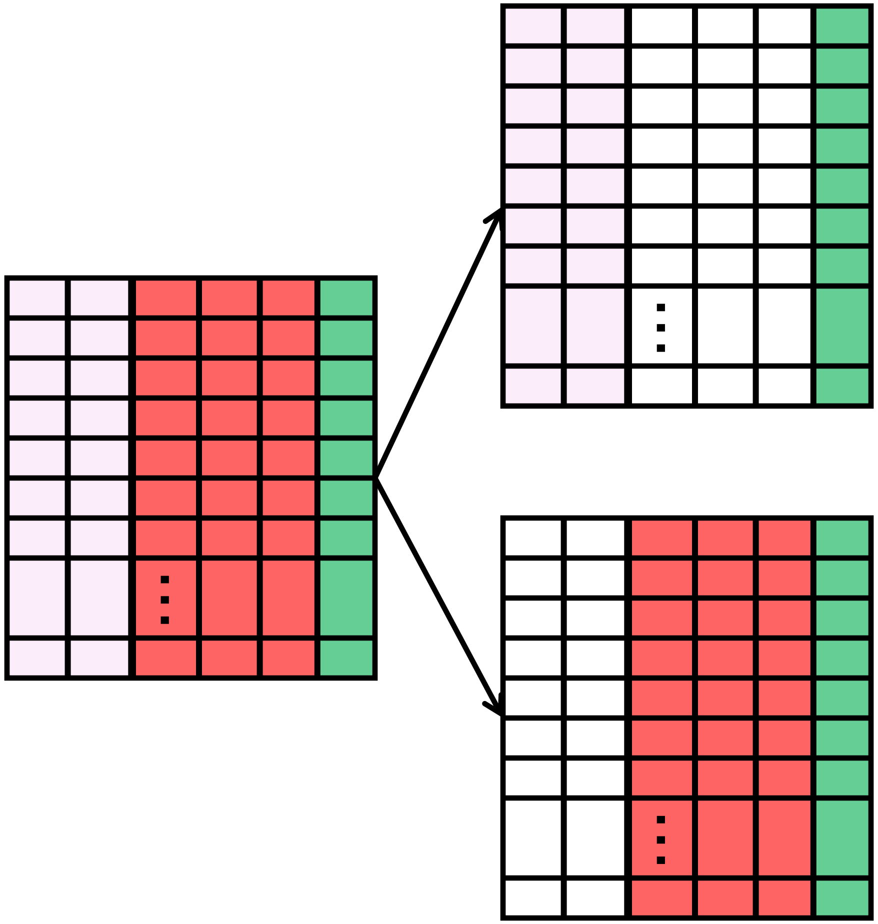
Ensemble Approaches

- Sample data set
  - Bagging
  - Boosting
- Manipulate features
  - Input feature
  - Target features
- Add randomness
  - Data
  - Algorithm
- Stacking



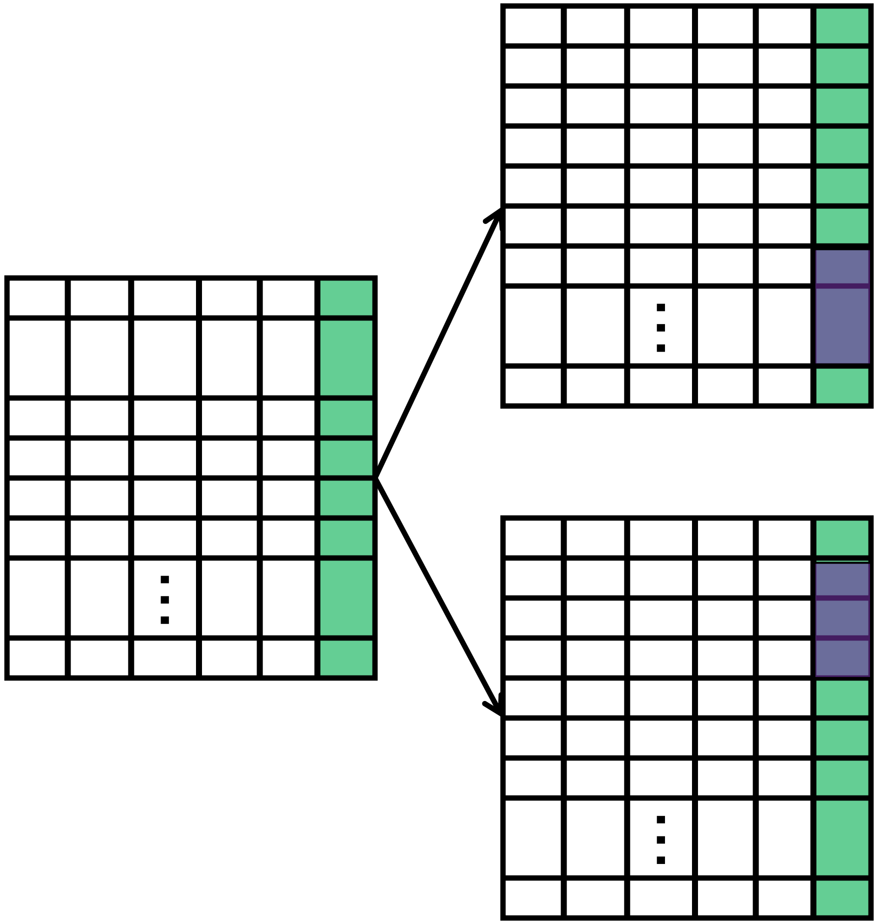
Ensemble Approaches

- Sample data set
  - Bagging
  - Boosting
- Manipulate features
  - Input feature
  - Target features
- Add randomness
  - Data
  - Algorithm
- Stacking



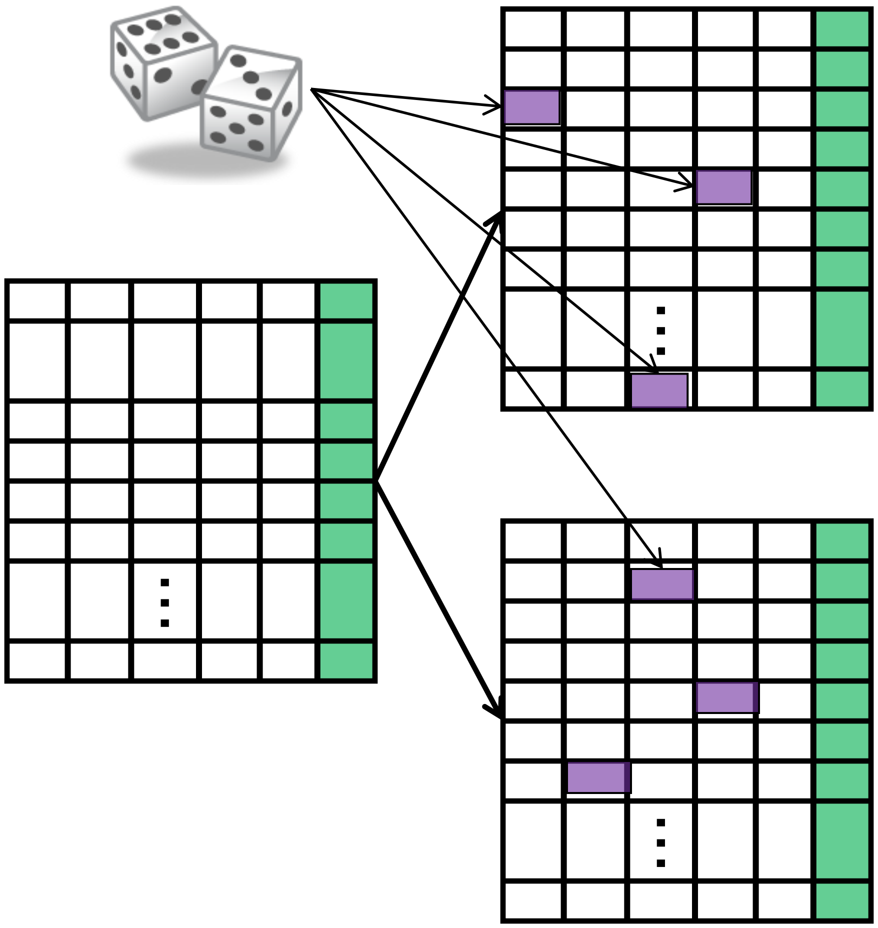
Ensemble Approaches

- Sample data set
  - Bagging
  - Boosting
- Manipulate features
  - Input feature
  - Target features
- Add randomness
  - Data
  - Algorithm
- Stacking



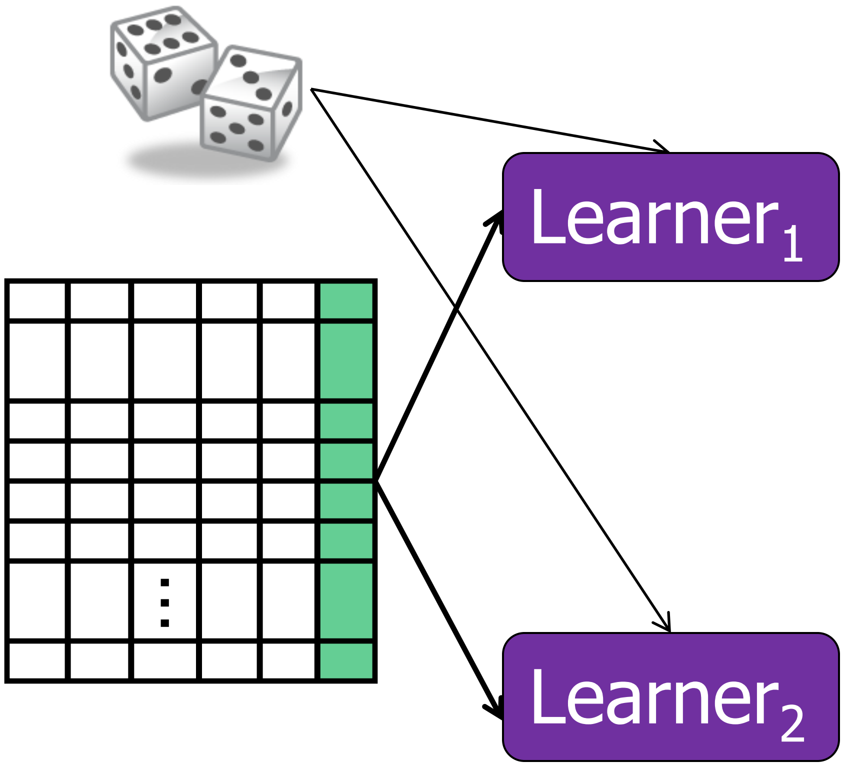
Ensemble Approaches

- Sample data set
  - Bagging
  - Boosting
- Manipulate features
  - Input feature
  - Target features
- Add randomness
  - Data
  - Algorithm
- Stacking



Ensemble Approaches

- Sample data set
  - Bagging
  - Boosting
- Manipulate features
  - Input feature
  - Target features
- Add randomness
  - Data
  - Algorithm
- Stacking



Sampling Based Ensembles

- Learner is **UNSTABLE** if: minor variations in training data results in major changes in classifier output
  - Unstable: Decision-tree, neural network, rule learning algorithms
  - Stable: Linear regression, nearest neighbor, linear threshold algorithms, etc.
- Subsampling is best for unstable learners:
  - Bagging
  - Cross-Validated Committees
  - Boosting

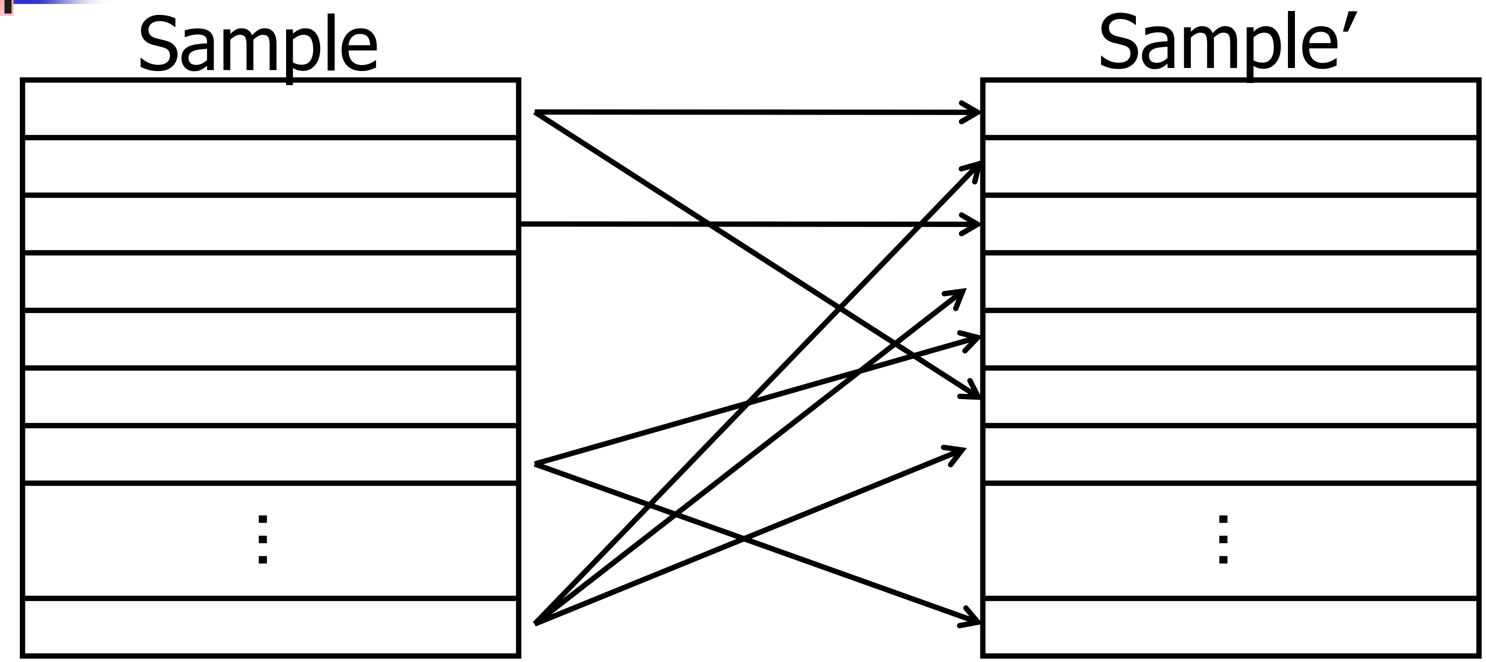
Bagging: **B**ootstrap **Agg**regating

Given: Data set S, integer T

- For i = 1, …, T
  - S_i = bootstrap replicate of S (i.e., sample with replacement)
  - h_i = Apply learning algorithm to S_i

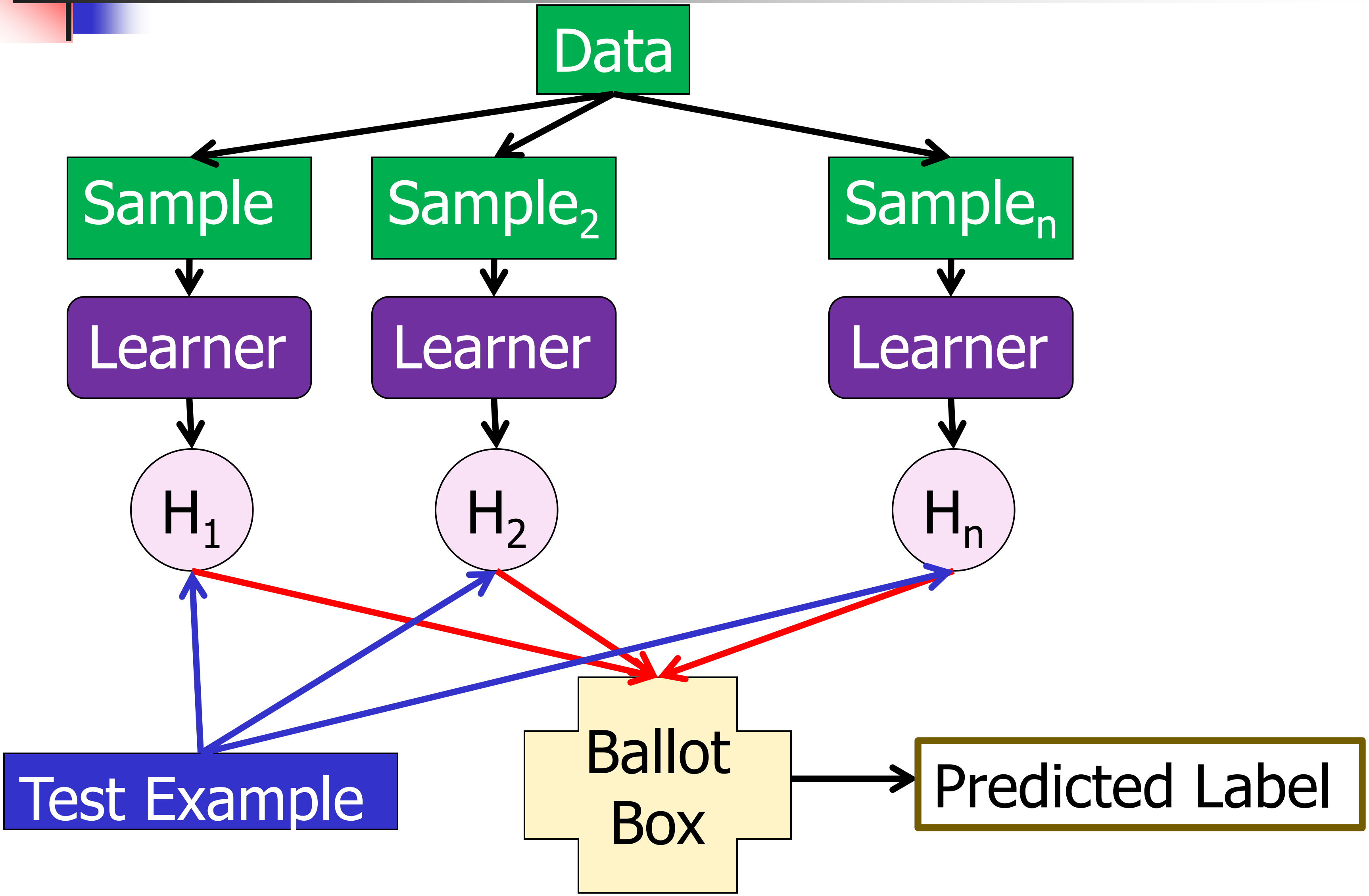
- Classify test instance using unweighted vote

Bagging: **B**ootstrap **Ag**gregating



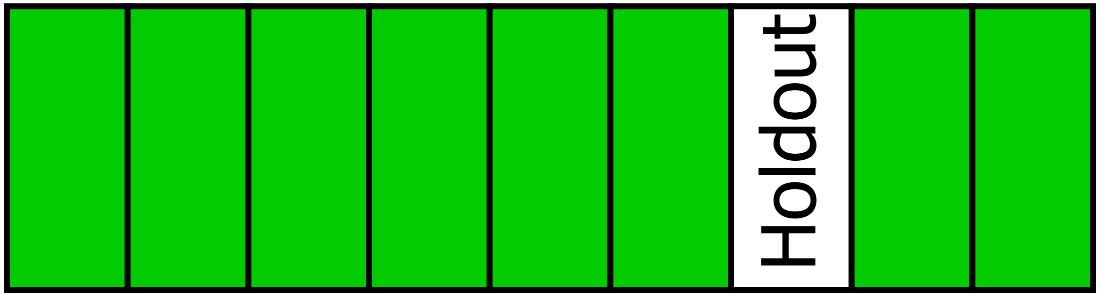
- Draw |Sample| examples with replacement
- Each sample’ contains 63.2% of original examples (+ duplicates)

Voting


Data
Sample
Sample2
Samplen
Learner
Learner
Learner
H1
H2
Hn
Test Example
Ballot Box
Predicted Label

Cross-validated Committees

- Partition training set into k disjoint subsets
- Create k training sets
  - Hold out one subset in turn
  - Learn model on each train set



- Classify test example with unweighted voting

Boosting

- Idea: General method for combining weak learners
  - Need ability to guess better than chance
  - Combine them to produce highly accurate predictor

- Needs sufficient data [and number of models]

AdaBoost

- Given: Data $S = \{(x_1, y_1),\ldots,(x_n, y_n)\}$, integer T
  - x_i = features
  - y_i = correct label
- $w_1(i) = 1/n$
- for $t = 1, \ldots, T$:
  - Find classifier h_t, with small error ϵ_t with $\epsilon_t = P_i[h_t(x_i) \neq y_i] = \sum_{h_t(x_i) \neq y_i} w_t(i)$
  - If $\epsilon_t > ½$ then break
  - Update distribution $w_t(i)$

AdaBoost

- Updating D_t
  - $\epsilon_t = P_{i \text{ in } w_t}[h_t(x_i) \neq y_i] = \Sigma_{h_t(x_i) \neq y_i} w_t(i)$
  - $\alpha_t = \epsilon_t / 1 - \epsilon_t$
  - $w_{t+1}(i) = w_t(i) \alpha_t^{1 - [h_t(x_i) \neq y_i]}$
  - Normalize: $w_{t+1}(i) / \Sigma\, w_{t+1}(j)$

- Output: $\text{argmax}_y = \Sigma_t \log(1/ \alpha_t)\, [h_t(x) = y]$

$\textsc{AdaBoost}(S,\ \mathit{Learn},\ k)$

S: Training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, $\ y_i \in Y$

Learn: Learner(S, weights)

k: # Rounds

For all i in S: $w_1(i) = 1/m$

For $r = 1$ to k do

For all i: $p_r(i) = w_r(i) / \sum_i w_r(i)$

$h_r = \mathit{Learn}(S, p_r)$

$\epsilon_r = \sum_i p_r(i)\, \mathbf{1}[h_r(i) \neq y_i]$

If $\epsilon_r > 1/2$ then

$k = r - 1$

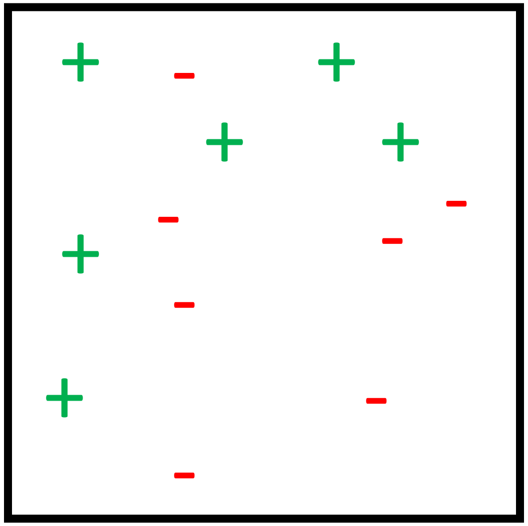
Exit

$\beta_r = \epsilon_r / (1 - \epsilon_r)$

For all i: $w_{r+1}(i) = w_r(i) \beta_r^{1 - \mathbf{1}[h_r(x_i) \neq y_i]}$

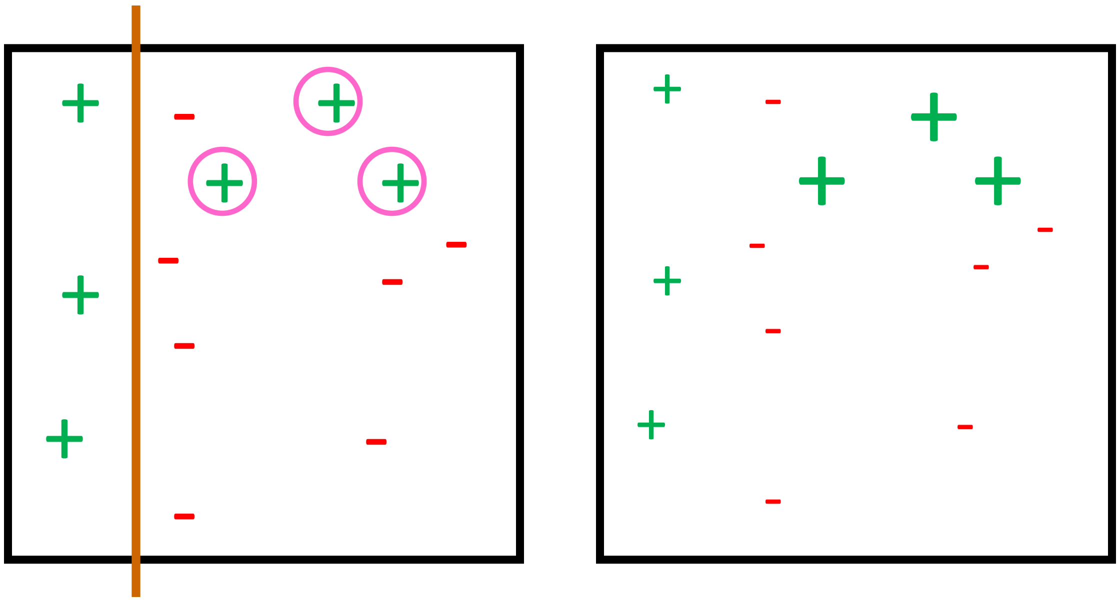
Output: $h(x) = \mathrm{argmax}_{y \in Y} \sum_{r=1}^{k} (\log \frac{1}{\beta_r})\, \mathbf{1}[h_r(x) = y]$

Boosting Example



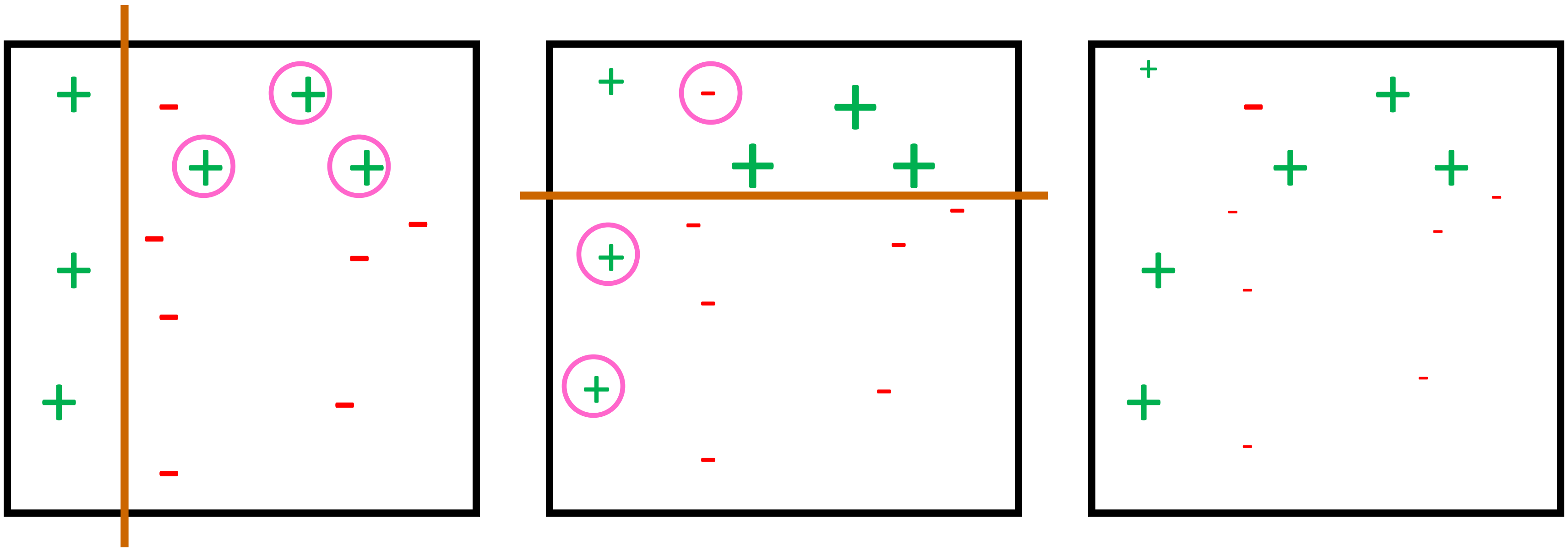
- Assume that we are going to make one axis parallel cut through feature space

Boosting Example

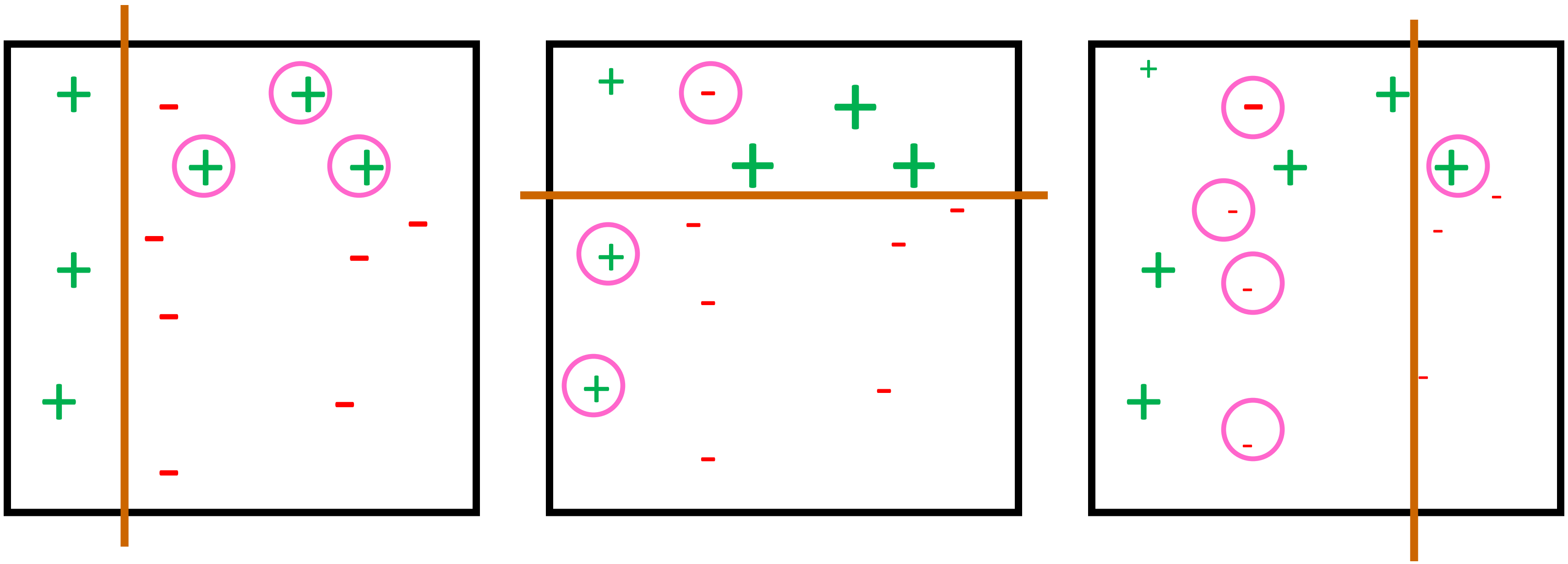


- Errors: 3
- Upweight the mistakes, downweight everything else

Boosting Example



Boosting Example



Learning from Weighted Examples

Q: How can a learning algorithm use distribution over examples?

- Reweighting: Can modify many learning algorithms to deal with weighted instances:
  - DT + Rule learners:
    - Entropy, information-gain equations count occurrences in data
    - Modify to use each instance's weight
  - Naïve Bayes: Use weight when building CPT
  - kNN: Multiple vote from an instance by its weight

Learning from Weighted Examples

Q: How can a learning algorithm use distribution over examples?

- Resampling: Given initial data set and distribution, produce new sample s'
  - Typically, of same size
  - Sample proportion to weights
  - Reweighting is better as resampling is just an approximation

Resampling Algorithm

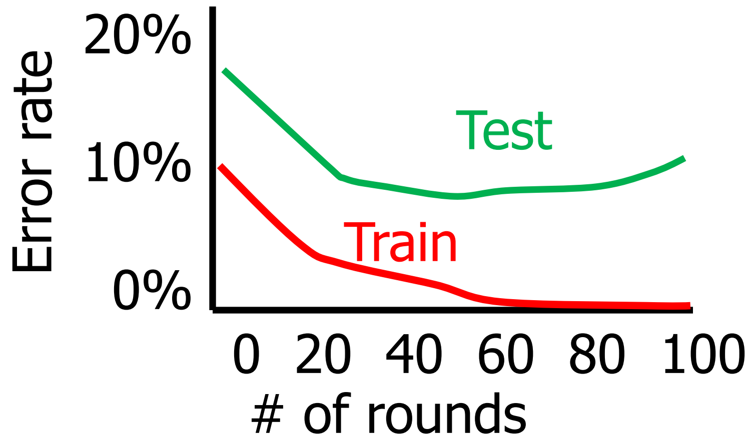
- Goal: Build S’
- Given: weights $(w_1, \ldots, w_n)$ for each example and $\Sigma\, w_i = 1$
- For i = 1 to n do:
  - Draw r from uniform (0,1)
  - Pick x_k such that $\Sigma^{k-1} w_i < r < \Sigma^{n} w_i$
- Return S’

Training Error for AdaBoost

Theorem: If $\gamma_t = ½ - \epsilon_t$ **then**
training_error(h*) $\leq \exp(-2\ \Sigma\gamma_t^2)$

- **If** $\gamma_t \geq \gamma > 0$ **then**
  training_error(h*) $\leq \exp(-2\gamma^2)$
- AdaBoost is adaptive:
  - Does not need to know γ or T a priori
  - Exploit $\gamma_t >> \gamma$

How Will # of Rounds Effect Generalization?

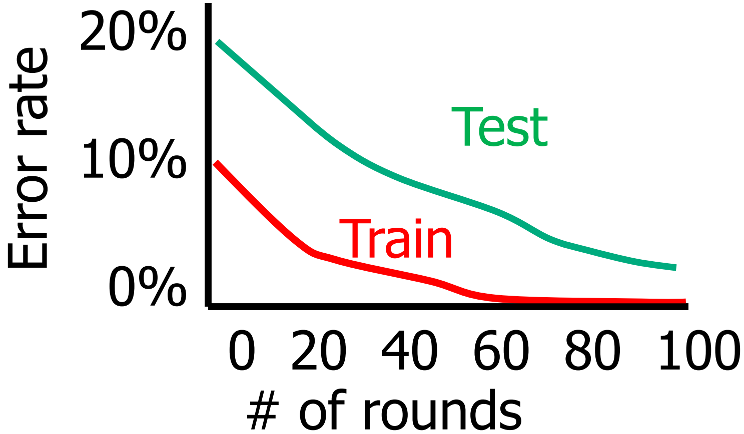


Expect

- Training error to drop or reach 0
- Test error to increase when h* becomes too complex: "Occam's razor" (i.e., overfitting)

- Hard to know when to stop training

Empirical Results



- Often, test error does not increase, even after 1000 rounds!
- Test error continues to drop, even after training error is 0!
- Occam's razor: "simpler is better" appears to not apply!

Explanation: Margins

- Key idea:
  - Training error only measures whether classifications are right or wrong
  - Should also consider confidence of classifications
- h* is weighted majority vote of weak classifiers
- Measure confidence by margin: Strength of vote
  - (weighted vote +) – (weighted vote -)

High conf. -	Low conf.	High conf. +
-1	0	1

AdaBoost Advantages

- Fast, simple and easy to program
- No parameters to tune (except T, sometimes)
- Flexible: works with any learning algorithm
- No prior knowledge needed about weak learner
- Provably effective, given weak classifier
- Versatile: can use with data that is textual, numeric, discrete, etc.
- Has been extended to learning problems well beyond binary classification

Notes on AdaBoost

- AdaBoost's performance depends on both the data and the weak learner

- AdaBoost can fail if:
  - weak classifiers too complex ->overfitting
  - weak classifiers too weak (error goes to 0 too quickly) -> underfitting

- Empirically, AdaBoost seems especially susceptible to uniform noise

Boosting Conclusions

- Boosting is a practical tool for classification and other learning problems
  - Grounded in rich theory
  - Performs well experimentally
  - Often (not always!) resistant to overfitting
  - Many applications and extensions
- Many ways to think about why boosting works
  - None is entirely satisfactory
  - Considerable room for further theoretical and experimental work

Manipulate Input Features

- Different learners see different subsets of features (of each training instances)
- Empirically: Mixed results
- Technique works best when input features highly redundant

Manipulating Target

- Sparse outputs Y = { y1, … yK }
  - Could learn 1 classifier, into Y (|Y| values)
  - Or could learn K binary classifiers:
    - y1 vs Y – y1
    - y2 vs Y – y2
    - then vote
  - Encoding by partition output labels into 2 subsets, create log k models
    - y1-y4 is pos, y5-y8 is neg
    - y1,y3,y5,y7 is pos, y2,y4,y6,y8 is neg

New Idea: Error-Correcting Codes

- Create more than log K models
  - "Error-Correcting Codes" (some redundancy)

- Given: Integer L
- For i = 1 to L
  - Partition labels into two disjoint sets
  - Build classifier to distinguish between these sets of examples

New Idea: Error-Correcting Codes

- L bit code word for each output label $y_{k,}$ ith bit
  - 1 if y_k is in new 'pos' class for h_i
  - 0 if y_k is in new 'neg' class for h_i
- Label unseen test example
  - Apply each h_i to example
  - Create bit vector, ith bit is
    - 1 if h_i predicts positive
    - 0 if h_i predicts negative
  - Using hamming distance to find closest class

Add Randomness to Learner

- Neural networks:
  - Different initial values
  - Not really independent
- Decision trees:
  - Consider top 20 attributes choose one at random?
  - Produce 200 classifiers
  - To classify new instance: Vote
- FOIL
  - Choose any test w/foil gain within 80% of top
  - Good empirical performance

Random Forrests

A variant of BAGGING

<u>Algorithm</u>

Repeat *k* times

1. Draw with replacement *N* examples, put in train set
2. Build d-tree, <u>but</u> in each recursive call
   A. Choose (w/o replacement) *i* features
   B. Choose best of these *i* as the root of this (sub)tree
3. Do NOT prune

More on Random Forrests

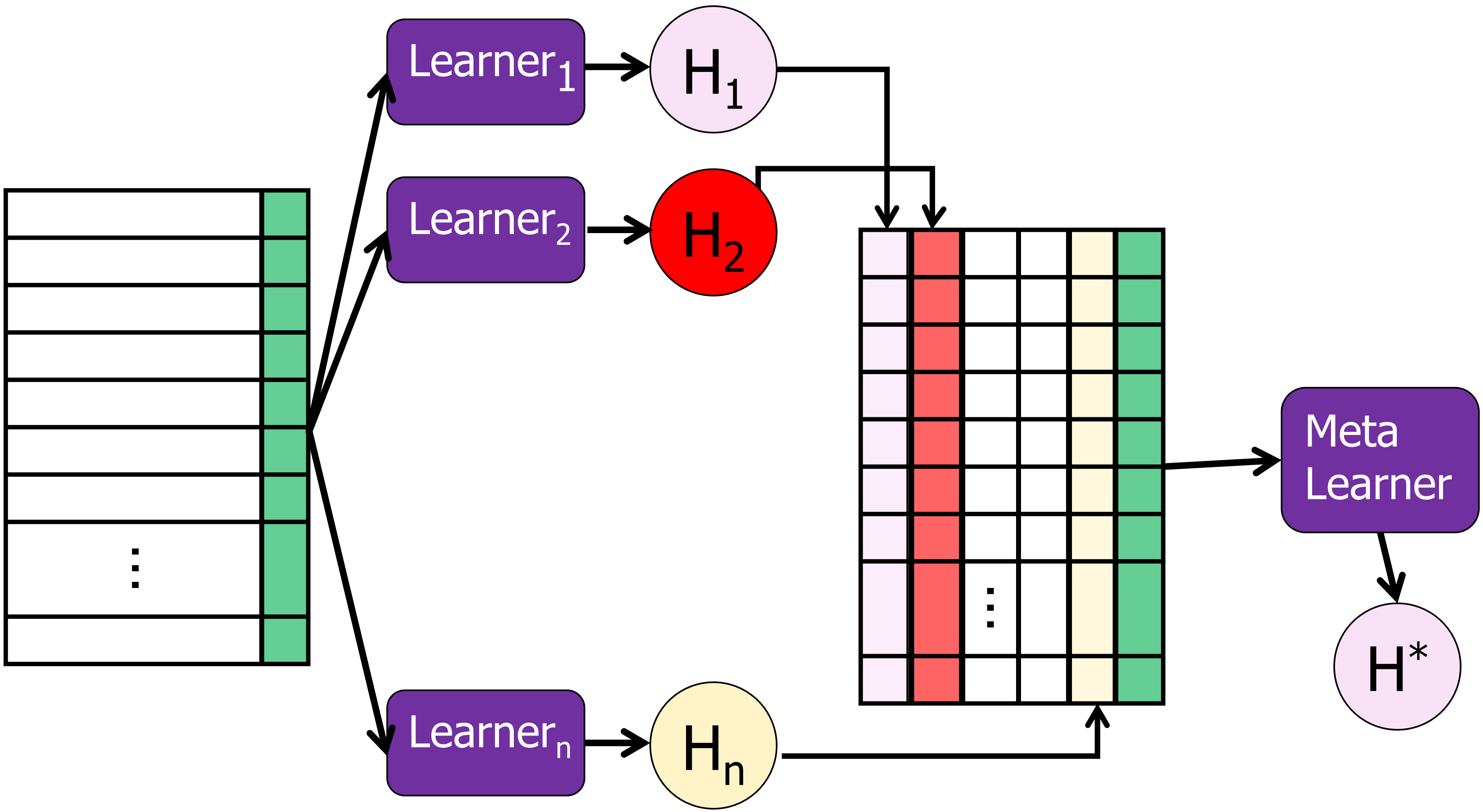
- Increasing *i*
  - Increases correlation among individual trees (BAD)
  - Also increases accuracy of individual trees (GOOD)
- Can use tuning set to choose good setting for *i*
- Overall, random forests
  - Are very fast (e.g., 50K examples, 10 features, 10 trees/min on 1 GHz CPU in 2004)
  - Deal with large # of features
  - Reduce overfitting substantially
  - Work very well in practice

Stacking

- Given: Learners L1,…,Ln
- Idea: Learn when each learner is good

- Let h_t (-i) = $L_t(S – x_i)$ be classifier learned using L_t, on all but instance x_i
- Let $y'_i(t) = h_t(x_i)$
- New train set: { [[$y'_i(1)$, $y'_i(2)$,…, $y'_i(n)$], y_i] }i

Stacking



Ensemble Recommendations

- Use Bagging with low bias and high variance classifiers
  - Decision trees
- Always try AdaBoost
  - Typically produces excellent results
  - Works especially well with very simple learners such as decision stumps

Why Do Ensembles Work?

- Bias/Variance explanation
- Statistical explanation
- Representational explanation
- Computational explanation

Bias/Variance Explanation

- Error has three components:
  - Inherent error: Inability to distinguish between two objects with different labels
  - Bias: Inability to represent the true target concept
  - Variance: Fluctuations due to variations in data sample

- Ensembles can address both bias and variance!

Statistical Explanation

- How can the learning algorithm select among set of equally good hypothesis?
- Bayes optimal classifier: Weighted majority vote of all hypotheses
  - Weighted by their posterior probability
  - Provably the best possible classifier
- Ensemble learning approximates Bayes optimal

Representational Explanation

- Optimal target function may not be ANY individual classifier, but may be (approximated by) ensemble averaging
  - E.g.: Decision trees boundaries are axis-parallel hyperplanes
  - Averaging a large number of such “staircases”, can approximate diagonal decision boundary with arbitrarily good accuracy

Computational Explanation

- Most learning algorithms search through hypotheses space find one “good” model
- Most interesting hypothesis spaces are:
  - Huge/infinite
  - Heuristic search is essential
- Learner might get stuck in a local minimum
- One strategy for avoiding local minima:
  - Repeat the search many times with random restarts -> bagging!

Effects of Bagging

- If bootstrap replicate approx'n is correct, then bagging would reduce variance without changing bias
- In practice, bagging can reduce both bias and variance
- For high-bias classifiers, it can reduce bias
- For high-variance classifiers, it can reduce variance

Effects of Boosting

- In the early iterations, boosting primarily reduces bias
- In later iterations, boosting primarily reduces variance (apparently)

Ensembles Summary

- Motivation: Committee of experts is typically more effective than a single supergenius
- Key issues:
  - Generating base models
  - Integrating responses from base models
- Popular ensemble techniques
  - manipulate training data: bagging and boosting
  - manipulate output values: error-correcting output coding
- Why does ensemble learning work?

Outline

- Homework 3 Issues
- Model Ensembles
- Genetic Algorithms

Biological Evolution

Lamarck:

- Species "transmute" over time

Darwin:

- Consistent, heritable variation among individuals in population
- Natural selection of the fittest

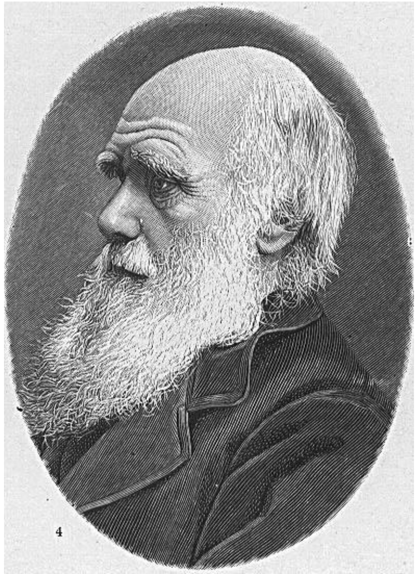
Mendel/Genetics:

- A mechanism for inheriting traits
- Mapping: Genotype $\rightarrow$ Phenotype

Evolutionary algorithms

Search algorithms based on the evolutionary principle of natural selection and survival of the fittest

"Although the belief that an organ so perfect as the eye could have been formed by natural selection is enough to stagger anyone; yet in the case of any organ, if we know a long series of gradations in complexity, each good for its possessor, then, under changing conditions of life, there is no logical impossibility in the acquirement of any conceivable degree of perfection through natural selection."



Charles R. Darwin
1809-1882

Evolutionary computation (EC)

- Genetic algorithms (GA)
  - Most popular technique
  - Pioneered by Holland and students in 1960/70s
- Evolution strategies (ES)
  - Aimed at solving real-valued optimization problems
  - Developed by Rechenberg and Schwefel in 1960/70s
- Genetic programming (GP)
  - Solutions are computer programs
  - Developed by Koza in 1990s

Genetic Algorithms (GAs)

- Search algorithms (optimization algorithms)
- Based on the natural principle of **survival of the fittest**
- Work on a set of solutions **(population)**
- Best individuals of the population survive **(selection)** and produce offspring **(crossover)**
- Variations occur through random changes **(mutation)** yielding a constant source of diversity

Checklist for applying a GA

1. define a **coding scheme** for individuals as bitstrings
2. define a **fitness function**
3. run the GA (involves setting parameters)

Example: find the global maximum of the function

$$f(x) = x^2$$

over {0,..., 31}

1. represent each number as a bitstring of length 5
2. use f as the fitness function

e.g. number: 13	string: **01101**	fitness: 169
e.g. number: 24	string: **11000**	fitness: 576

Representing Hypotheses

Represent

$$(Outlook = Overcast \vee Rain) \wedge (Wind = Strong)$$

by

$$\begin{array}{cc} Outlook & Wind \\ 011 & 10 \end{array}$$

Represent

$$\text{IF} \quad Wind = Strong \quad \text{THEN} \quad PlayTennis = yes$$

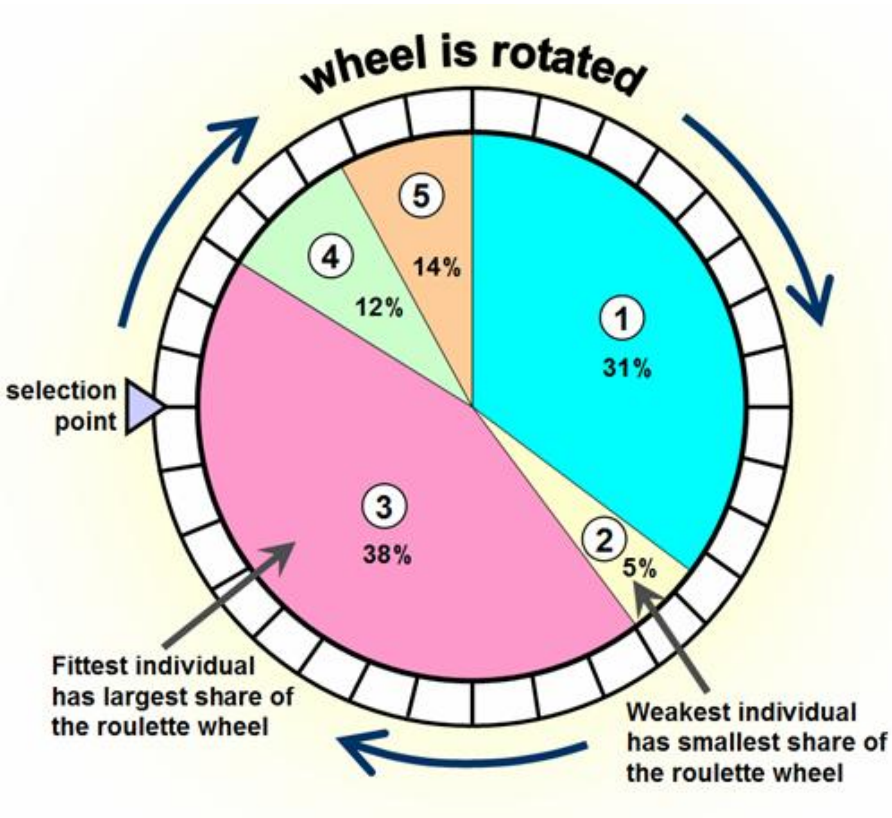
by

$$\begin{array}{ccc} Outlook & Wind & PlayTennis \\ 111 & 10 & 10 \end{array}$$

The Canonical GA

1. Randomly generate an initial population of size m
2. Do until termination condition is met:
   // build a new generation
   1) **select** m individuals for reproduction
      // some might be chosen more than once
   2) create offspring by **crossing** individuals
   3) occassionaly **mutate** some individuals
3. Return best solution found

Roulette Wheel Selection



- Probabilistic nature helps to escape from local optima
- Fit inviduals are more likely to survive and become parents
- Even least fit individual in current population has *some* probability of becoming a parent

Selection: example

Individual No.	String (genotype)	x value (phenotype)	$f(x)$ x^2	pselect_i $\frac{f_i}{\sum f_j}$
1	0 1 1 0 1	13	169	0.14
2	1 1 0 0 0	24	576	0.49
3	0 1 0 0 0	8	64	0.06
4	1 0 0 1 1	19	361	0.31

e.g. for the new generation (random experiment):

– no. 1 and no. 4 are selected

– no. 2 is selected twice

– no. 3 dies

Selecting Fittest Hypotheses

Fitness-proportionate selection:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

... can lead to *crowding*

Tournament selection:

- Pick h_1, h_2 at random with uniform probability
- With probability p, select the more fit

Rank selection:

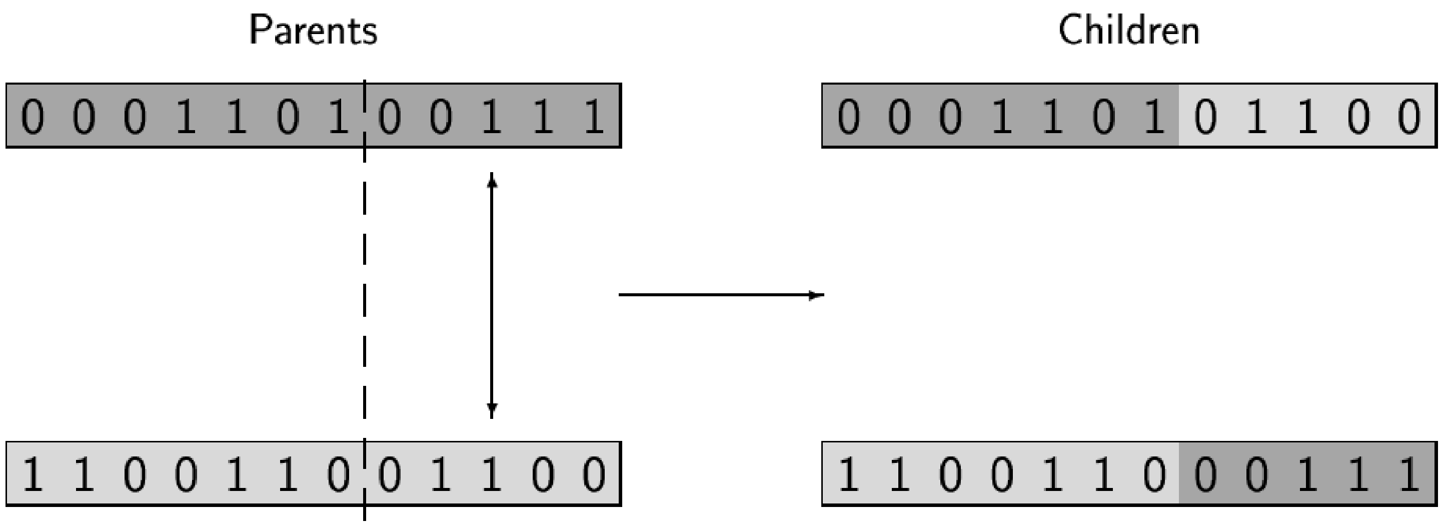
- Sort all hypotheses by fitness
- Prob. of selection is proportional to rank

The Canonical GA

1. Randomly generate an initial population of size m
2. Do until termination condition is met:
   // build a new generation
   1) **select** m individuals for reproduction
      // some might be chosen more than once
   2) create offspring by **crossing** individuals
   3) occassionaly **mutate** some individuals
3. Return best solution found

Crossover

- merges information from parents into offspring



- offspring may be worse or the same as parents
- hope is that some are better by combining elements of parents with good traits

Crossover: Example

selection

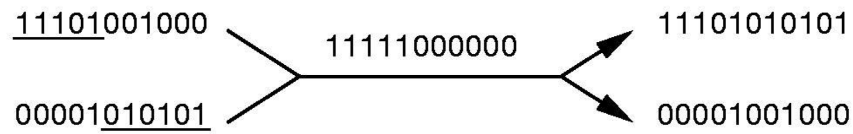
Individual No.	String (genotype)	x value (phenotype)	$f(x)$ x^2	pselect$_i$ $\frac{f_i}{\sum f_j}$
1	0 1 1 0 1	13	169	0.14
2	1 1 0 0 0	24	576	0.49
3	0 1 0 0 0	8	64	0.06
4	1 0 0 1 1	19	361	0.31

Set of selected individuals	Crossover site (random)	New population	x value	$f(x)$ x^2
0 1 1 0 | 1	4	0 1 1 0 0	12	144
1 1 0 0 | 0	4	1 1 0 0 1	25	625
1 1 | 0 0 0	2	1 1 0 1 1	27	729
1 0 | 0 1 1	2	1 0 0 0 0	16	256

Operators for Genetic Algorithms

Initial strings *Crossover Mask* *Offspring*

Single-point crossover:



The Canonical GA

1. Randomly generate an initial population of size m
2. Do until termination condition is met:
   // build a new generation
   1) **select** m individuals for reproduction
      // some might be chosen more than once
   2) create offspring by **crossing** individuals
   3) occassionaly **mutate** some individuals
3. Return best solution found

Mutation

- random deformation of genetic information
- responsible for preserving and introducing diversity
- inversion of a single bit

  0**1**101 ⟶ 0**0**101

  bitwise inversion of the whole bitstring

  01101 ⟶ 10010

  replace bitstring by randomly chosen one

  01101 ⟶ 11001
- keep probability low to avoid chaotic behaviour

The Canonical GA

1. Randomly generate an initial population of size m
2. Do until termination condition is met: ?
   // build a new generation
   1) **select** m individuals for reproduction
      // some might be chosen more than once
   2) create offspring by **crossing** individuals
   3) occassionaly **mutate** some individuals
3. Return best solution found

Termination conditions

- reaching some (known/hoped for) fitness
- reaching maximum allowed number of generations
- reaching minimum level of diversity
- reaching a specified number of generations without fitness improvement

More on Encoding

- Genotype is mostly a bitstring (binary encoding)
- Natural for Boolean decision variables
- Often used to encode non-binary information
  - Anything can be represented in binary
  - ... (to some arbitrary precision)
- Other encodings for numeric results
  - Integer, floating point

Sparseness problem

- consider the simple optimization problem of finding the largest integer in [0, 1, . . . , 8]
- encoded as standard binary, the fitness function is

Chromosome	Fitness	Chromosome	Fitness
0000	0	1000	8
0001	1	1001	undefined
0010	2	1010	undefined
0011	3	1011	undefined
0100	4	1100	undefined
0101	5	1101	undefined
0110	6	1110	undefined
0111	7	1111	undefined

Discontinuity problem

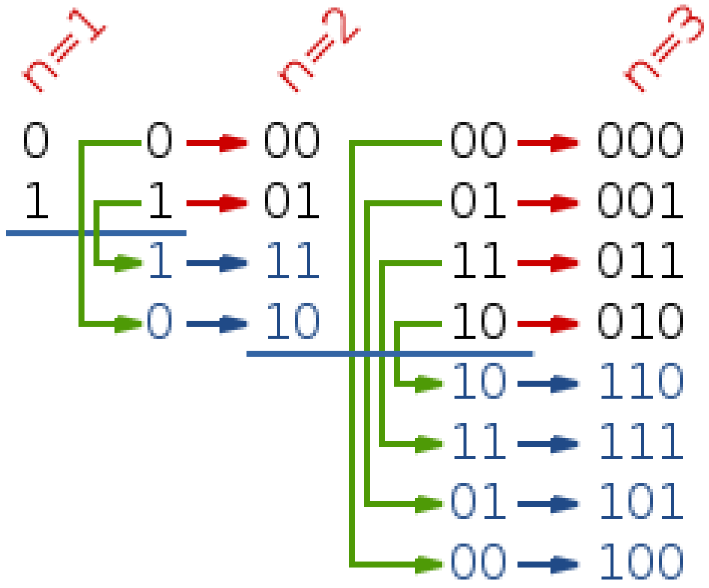
- consider the simple optimization problem of finding the largest integer in [0, 1, . . . , 15]
- standard binary encoding has some problems:
  - Hamming distance between chromosomes encoding adjacent integers is not constant
  - chromosomes that differ in only one or two bits may encode for totally different solutions (e.g. 0000 -> 0, 1000 -> 9)
  - chromosomes that differ in all bits may encode very similar solutions (e.g. 1000 -> 9, 0111 -> 8)

Solution: Gray Codes

Decimal	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

- Invented by Gray in 1940s
- Adjacent integers are encoded by chromosomes that differ in one gene

Solution: Gray code



- reflected binary code

Traveling salesman problem (TSP)

Starting in Seattle, find the shortest route to visit all other cities exactly once and then return to Seattle.

Solution with genetic algorithm

1. coding scheme: string of integer numbers

2. fitness function: based on the route length

3. - selection (as usual)

$$p_1 = (1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9)$$
$$p_2 = (4\ \ 5\ \ 2\ \ 1\ \ 8\ \ 7\ \ 6\ \ 9\ \ 3)$$

   - crossover (e.g. partially mapped or PMX)

$$o_1 = (*\ \ 2\ \ 3\,|\,1\ \ 8\ \ 7\ \ 6\,|\,*\ \ 9) \quad\Rightarrow\quad o_1 = (4\ \ 2\ \ 3\ \ 1\ \ 8\ \ 7\ \ 6\ \ 5\ \ 9)$$
$$o_2 = (*\ \ *\ \ 2\,|\,4\ \ 5\ \ 6\ \ 7\,|\,9\ \ 3) \quad\Rightarrow\quad o_2 = (1\ \ 8\ \ 2\ \ 4\ \ 5\ \ 6\ \ 7\ \ 9\ \ 3)$$

   - mutation (e.g. swap two cities)

Next Class

- Learning theory
- Support vector machines
- Active learning

Summary

- Ensembles:
  - Old paradigm: Learn one model
  - New paradigm: Learn many models!
  - Good empirical results
- Genetic algorithms:
  - Based on biological principles, which is appealing
  - Significant hand-crafting to get good results

Questions?