Optimization and Complexity

Decision Systems Group
Brigham and Women’s Hospital,
Harvard Medical School
Aim

• Give you an intuition of what is meant by
  - Optimization
  - P and NP problems
  - NP-completeness
  - NP-hardness

• Enable you to recognize formals of complexity theory, and its usefulness
Overview

• Motivating example
• Formal definition of a problem
• Algorithm and problem complexity
• Problem reductions
  – NP-completeness
  – NP-hardness
• Glimpse of approximation algorithms and their design
What is optimization?

• Requires a *measure* of optimality
  – Usually modeled using a mathematical function

• Finding the solution that yields the globally best value of our measure
What is the problem?

• Nike: Just do it
• Not so simple:
  – Even problems that are simple to formally describe can be intractable
  – Approximation is necessary
  – Complexity theory is a tool we use to describe and recognize (intractable) problems
Example: Variable Selection

- Data tables T and V have n predictor columns and one outcome column. We use machine learning method L to produce predictive model L(T) from data table T. We can evaluate L(T) on V, producing a measure E(L(T), V).

- We want to find a maximal number of predictor columns in T to delete, producing T’, such that
  \[ E(L(T’), V) = E(L(T), V) \]

- There is no known method of solving this problem optimally (e.g., NP-hardness of determining a minimal set of variables that maintains discernibility in training data, aka the rough set reduct finding problem).
Search for Optimal Variable Selection

- The space of all possible selections is huge
- 43 variables, $2^{43} - 1$ possibilities of selecting a non-empty subset, each being a potential solution
- one potential solution pr. post-it gives a stack of post-its reaching more than half way to the moon
Search for Optimal Variable Selection

- Search space
  - discrete
  - structure that lends itself to *stepwise search* (add a new or take away one old)
  - optimal point is not known, nor is optimal evaluation value
Popular Stepwise Search Strategies

- Hill climbing:
  - select starting point and always step in the direction of most positive change in value
Popular Stepwise Search Strategies

- Simulated annealing:
  - select starting point and select next stepping direction stochastically with increasing bias towards more positive change
Problems

• Exhaustive search: generally intractable because of the size of the search space (exponential in the size of variables)

• Stepwise: no consideration of synergy effects
  – Variables $a$ and $b$ considered in isolation from each other are excluded, but their combination would not be
Genetic Algorithm Search

- population of solutions
- Stochastic selection of parents with bias towards “fitter” individuals
- “mating” and “mutation” operations on parents
- Merging of old population with offspring
- Repeat above until no improvement in population
GA Optimization
Animation

HST 951 Spring 2003
Addressing the Synergy Problem of Stepwise Search

- Genetic algorithm search
  - Non-stepwise, non-exhaustive
  - Pros:
    - Potentially finds synergy effects
    - Does not a priori exclude any parts of the search space
  - Cons:
    - Computationally expensive
    - Difficult to analyze, no comprehensive theory for parameter specification
Variable Selection for Logistic Regression using GA

• Data:
  – 43 predictor variables
  – Outcome: MI or not MI (1 or 0)
  – Training ($T$, 335 cases) and Holdout ($H$, 165 cases) from Sheffield, England
  – External validation ($V$, 1253 cases) from Edinburgh, Scotland

HST 951 Spring 2003
GA Variable Selection for LR: Generational Progress

Fitness value evolution

- Max
- Mean
- Min

Generation

Fitness

HST 951 Spring 2003
GA Variable Selection for LR: Results

- Table presenting results on validation set E, including SAS built-in variable selection methods (removal/entry level 0.05)

<table>
<thead>
<tr>
<th>Selection</th>
<th>Size</th>
<th>ROC AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genetic</td>
<td>6</td>
<td>0.95</td>
</tr>
<tr>
<td>none</td>
<td>43</td>
<td>0.92</td>
</tr>
<tr>
<td>Backward</td>
<td>11</td>
<td>0.92</td>
</tr>
<tr>
<td>Forward</td>
<td>13</td>
<td>0.91</td>
</tr>
<tr>
<td>Stepwise</td>
<td>12</td>
<td>0.91</td>
</tr>
</tbody>
</table>
Problem Example

• Boolean formula $f$ (with variables)
  - Is there a truth assignment such that $f$ is true?
  - Does this given truth assignment make $f$ true?
  - Find a satisfying truth assignment for $f$
  - Find a satisfying truth assignment for $f$ with the minimum number of variables set to true
Problem Formally Defined

• A problem P is a relation from a set I of instances to a set S of solutions: $P \subseteq I \times S$
  
  – Recognition: is $(x,y) \in P$ ?
  
  – Construction: for x find y such that $(x,y) \in P$
  
  – Optimization: for x find the best y such that $(x,y) \in P$
Solving Problems

• Problems are solved by an algorithm, a finite description of steps, that compute a result given an instance of the problem.
Algorithm Cost

• Algorithm cost is measured by
  – How many operations (steps) it takes to solve the problem (time complexity)
  – How much storage space the algorithm requires (space complexity)

on a particular machine type as a function of input length (e.g., the number of bits needed to store the problem instance).
O-Notation

• O-notation describes an upper bound on a function

• let $g, f: \mathbb{N} \rightarrow \mathbb{N}$

  $f(n)$ is $O(g(n))$

  if there exists constants $a, b, m$

  such that for all $n = m$

  $f(n) = a \times g(n) + b$
O-Notation Examples

\[ f(n) = 9999999999999999 \]

is O(1)

\[ f(n) = 1000000n + 100000000 \]

is O(n)

\[ f(n) = 3n^2 + 2n - 3 \]

is O(n^2)

(Exercise: convince yourselves of this please)
Worst Case Analysis

• Let \( t(x) \) be the running time of algorithm A on input \( x \)

• Let \( T(n) = \max\{t(x) \mid |x| = n\} \)
  - I.e., \( T(n) \) is the worst running time on inputs not longer than \( n \).

• A is of time complexity \( O(g(n)) \) if \( T(n) \) is \( O(g(n)) \)
Problem Complexity

• A problem P has a time complexity $O(g(n))$ if there exists an algorithm that has time complexity $O(g(n))$

• Space complexity is defined analogously
Decision Problems

- A *decision problem* is a problem $P$ where the set of Instances can be partitioned into $Y_P$ of positive instances and $N_P$ of non-positive instances, and the problem is to determine whether a particular instance is a positive instance.

- Example: satisfiability of Boolean CNF formulae, does a satisfying truth assignment exist for a given instance?
Two Complexity Classes for Decision Problems

- **P** – all decision problems of time complexity $O(n^k)$, $0 = k = \infty$
- **NP** – all decision problems for which there exists a non-deterministic algorithm with time complexity $O(n^k)$, $0 = k = \infty$
What is a non-deterministic algorithm?

- **Algorithm**: finite description (program) of steps.
- **Non-deterministic algorithm**: an algorithm with “guess” steps allowed.
Computation Tree

- Each guess step results in a “branching point” in a computation tree.
- Example: satisfying a Boolean formula with 3 variables.

$$(((\neg a \land b) \lor \neg c)$$
Non-deterministic algorithm time complexity

- A ND algorithm A solves the decision problem P in time complexity t(n) if, for any instance x with |x| = n, A halts for any possible guess sequence and $x \in Y_P$ if and only if there exists at least one sequence which results in YES in time at most t(n)
P and NP

- We have that $P \subseteq NP$

- If there are problems in NP that are not in $P$ is still an open problem, but it is strongly believed that this is the case.
Problem Reduction

• A reduction from problem $P_1$ to problem $P_2$ presents a method for solving $P_1$ using an algorithm for $P_2$.
  – $P_2$ is then intuitively at least as difficult as $P_1$
Problem Reduction

• Problem $P_1$ is *reducible* to $P_2$ if there exists an algorithm $R$ which solves $P_1$ by querying an *oracle* for $P_2$. In this case, $R$ is said to be a *reduction* from $P_1$ to $P_2$, and we write $P_1 = P_2$.

• If $R$ is of polynomial time complexity we write $P_1 =^p P_2$.
NP-completeness

• A decision problem $P$ is NP-complete if
  – It is in NP, and
  – For any other problem $P'$ in NP we have that $P' =^p P$,
• This means that any NP problem can be solved in polynomial time if one finds a polynomial time algorithm for NP-complete $P$
• There are problems in NP for which the best known algorithms are exponential in time usage, meaning that NP-completeness is a sign of problem intractability
Optimization Problems

- Problem P is a quadruple \((I_P, S_P, m_P, g_P)\)
  - \(I_P\) is the set of instances
  - \(S_P\) is a function that for an instance \(x\) returns the set of feasible solutions \(S_P(x)\)
  - \(m_P(x,y)\) is the positive integer measure of solution quality of a feasible solution \(y\) of a given instance \(x\)
  - \(g_P\) is either min or max, specifying whether \(P\) is a maximization or minimization problem

- The optimal value for \(m_P\) for \(x\) over all solutions is denoted \(m_P(x)\). A solution \(y\) for which \(m_P(x,y) = m_P(x)\) is called optimal and is denoted \(y^*(x)\).
Optimization Problem
Example

• Minimum hitting set problem
  – \( I = \{ C \mid C \subseteq 2^U \} \)
  – \( S = \{ H \mid H \cap c \neq \emptyset, \ c \in C \} \)
  – \( m(C,H) = |H| \)
  – \( g = \min \)
Complexity Class NPO

An optimization problem \((I, S, m, g)\) is in NPO if

1. An element of \(I\) is recognizable as such in polynomial time
2. Solutions of \(x\) are bounded in size by a polynomial \(q(|x|)\), and are recognizable as such in \(q(|x|)\) time
3. \(m\) is computable in polynomial time

Theorem: For an NPO problem, the decision problem whether \(m(x) = K\) \((g=\text{min})\) or \(m(x) = K\) \((g=\text{max})\) is in NP
Complexity Class PO

• An optimization problem $P$ is said to be in $PO$ if it is in $NPO$ and there exists an algorithm that for each $x$ in $I$ computes an element $y^*(x)$ and its value $m(x)$ in polynomial time.
NP-hardness

- NP-completeness is defined for decision problems
- An optimization problem $P$ is NP-hard if for every NP decision problem $P'$ we have that $P' \equiv^p P$
- Again, NP-hardness is a sign of intractability
Approximation Algorithms

• An algorithm that for an NPO problem P always returns a feasible solution is called an approximation algorithm for P

• Even if an NPO problem is intractable it might not be difficult to design a polynomial time approximation algorithm
Approximate Solution Quality

- Any feasible solution is an approximate solution, and is characterized by the distance from its value to the optimal one.
- An approximation algorithm is characterized by its complexity, and by the ratio of the distance above to the optimum, and the growth of this performance ratio with input size.
- An algorithm is a p-approximate algorithm if the performance ratio is bounded by the function p in input size.
Some Design Techniques for Approximation Algorithms

- **Local search**
  - Given solution, search for better “neighbor” solution
- **Linear programming**
  - Formulate problem as a linear program
- **Dynamic Programming**
  - Construct solution from optimal solutions to sub-problems
- **Randomized algorithms**
  - Algorithms that include random choices
- **Heuristics**
  - Exploratory, possibly learning strategies that offer no guarantees
Thank you

That’s all folks