Local Search Algorithms

- A general class of algorithms for solving optimization problems
- Some terms:
  - A **state** is defined as an assignment of values to all variables of interest
  - Define a **neighborhood function**, which defines a set of states that you can move/hop to from your current state. To manage the computational complexity, a neighborhood function is selected in such a way that each state has linear or in the worst-case polynomial number of states as its neighbors
    - Example neighborhood function: Change the value of exactly one variable from the current state

Example of State and Neighborhood

- A Markov network over 5 binary variables {A,B,C,D,E}
- Example of a state:
  - State s: A=0, B=1, C=0, D=0, E=1
- Let neighborhood function of each state be defined as follows:
  - All states which differ from the current state in value of just one variable
- The following are neighbors of state “s” according to our definition of neighborhood:
  - s1: A=1, B=1, C=0, D=0, E=1 …… A flipped from 0 to 1
  - s2: A=0, B=0, C=0, D=0, E=1 …… B flipped from 1 to 0
  - s3: A=0, B=1, C=1, D=0, E=1 …… C flipped from 0 to 1
  - s4: A=0, B=1, C=0, D=1, E=1 …… D flipped from 0 to 1
  - s5: A=0, B=1, C=0, D=0, E=0 …… E flipped from 1 to 0

Local Search: Basic Algorithm for a Maximization Problem

- Assumption: given a state “s”, we can easily calculate the value of the objective function denoted by score(s)
- Algorithm:
  - s = a random state
  - Best = s
  - Until time runs out do
    - s’ = a neighbor of s having the highest score
    - If score(Best) < score(s’) then
      - Best=s’
    - If score(s’)>=score(s) then
      - s=s’
  - Return Best
- Issues:
  - Algorithm will get stuck in a **local maxima** if there does not exist a neighbor s’ of s such that score(s’)>=score(s)
  - Algorithm will get stuck in a **plateau** if there does not exist a neighbor s’ of s such that score(s’)=score(s)

Advanced Local Search Algorithm

- Escape local maximas and plateaus using random walks
- Algorithm (Input: a random walk probability p):
  - s = a random state
  - Best = s
  - Until time runs out do
    - With probability p   // random walk step
      - s'= a random neighbor of s
    - Else (with probability 1-p):
      - s' = a neighbor of s having the highest score // locally optimal move
    - If score(Best) < score(s') then
      - Best=s'
    - s=s'
  - Return Best

[Random walks for SAT solving: https://www.cs.rochester.edu/u/kautz/walksat/](https://www.cs.rochester.edu/u/kautz/walksat/)

MPE by Local Search

- Straight-forward
  - State = assignment of values to all non-evidence variables
  - Scoring function: score(s) is P(x,e) where e is evidence and x is the assignment of values to all the non-evidence variables in state s
    - In Bayesian networks, this is easy. Project the assignment (x,e) on each CPT to yield a probability value and take the product of all these probability values
    - In Markov networks, this is also easy. We need to know P(x,e) up to a normalization constant and therefore we don't have to compute the partition function Z.
      - Project the assignment (x,e) on each potential to yield a potential value and take the product of all these potential values.

MAP by Local Search

- State = Assignment of values to the MAP variables “Y”
- Score of a state having assignment Y=y is P(y,e)
  - P(y,e) can be computed by summing out all the non-MAP variables from the graphical model via Bucket elimination
- Thus, given a network with “n” non-MAP variables and an elimination order of width “w” over the non-MAP variables
  - Complexity of computing the score is O(nexp(w+1))
- MAP is a hard problem. Even local search requires that inference over the non-MAP variables is tractable (has low polynomial complexity).

Recap

- Exact MPE and MAP
  - Bucket elimination
  - Branch and Bound Search
- Approximations
  - Mini bucket elimination
  - Branch and Bound Search
  - Local Search

STATISTICAL METHODS IN AI/ML

Vibhav Gogate
University of Texas, Dallas

LEARNING: Lecture 1

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Learning Graphical models: Basic Framework

- Generating a graphical model by hand, expert, etc. is not possible
  - Thousands and sometimes millions of variables (e.g., the Web domain)
- **Input:** A data set $\mathcal{X} = \{\mathbf{x}[1], \ldots, \mathbf{x}[M]\}$ having M examples or samples. (Assumption: the M examples are independent and identically distributed (IID); generated from P^*)
- **Output:** A graphical model $\widetilde{M}$ representing a distribution $\widetilde{P}$ such that
  - it is as close as possible to P^*
  - it solves the task/problem that you are interested in as accurately as possible

Evaluating Learning Performance

- Given candidate models, how do I evaluate which is better?
  - Non-trivial task. How do I define the notion of best?
- Various performance metrics depending upon your learning goal.

Performance metric 1: Relative Entropy or KL distance

$$\mathbf{D}(P^*||\widetilde{P}) = \sum_{\mathbf{x}} P^*(\mathbf{x}) \log(P^*(\mathbf{x})) - \sum_{\mathbf{x}} P^*(\mathbf{x}) \log(\widetilde{P}(\mathbf{x}))$$

- We can not evaluate this directly (exponentially large). However, we can consider our data as samples and use the following Monte Carlo estimate:

$$\widehat{\mathbf{D}}(P^*||\widetilde{P}) = \frac{1}{M}\sum_{i=1}^{M} \log(P^*(\mathbf{x}^{(i)})) - \frac{1}{M}\left[\sum_{i=1}^{M} \log(\widetilde{P}(\mathbf{x}^{(i)}))\right]$$

- The first term is a constant (no need to evaluate). The term in [...] is called the log-likelihood of the data.

Performance metric 1: Maximum likelihood learning (MLE)

$$\widehat{\mathbf{D}}(P^*||\widetilde{P}) = \frac{1}{M}\sum_{i=1}^{M}\log(P^*(\mathbf{x}^{(i)})) - \frac{1}{M}\left[\sum_{i=1}^{M}\log(\widetilde{P}(\mathbf{x}^{(i)}))\right]$$

- We should prefer models that have the maximum value for $\sum_{i=1}^{M}\log(\widetilde{P}(\mathbf{x}^{(i)}))$ (log-likelihood)
- Will likely minimize the error (i.e., improve accuracy)
- Since logarithm is monotonic, maximizing the log-likelihood is same as maximizing the likelihood:

$$L(\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(M)}) = \exp\left[\sum_{i=1}^{M}\log(\widetilde{P}(\mathbf{x}^{(i)}))\right] = \prod_{i=1}^{M}\widetilde{P}(\mathbf{x}^{(i)})$$

Performance metric 2: Task directed learning

- You may be interested in a specific task
  - Classification task: Given a set of documents, find the topic of each document
    - Classification error: # of mis-classified instances.
    - Hamming loss: When we are interested in multi-class labeling, we count the number of variables that are mis-classified
  - Query Variables $\mathbf{Y}$: You may be interested in querying only a subset of the variables. Let the other variables $\mathbf{X} \setminus \mathbf{Y}$ be denoted by $\mathbf{Z}$.
    - Maximize conditional log likelihood of data:

$$\sum_{i=1}^{M} \log\left(\widetilde{P}\left(\mathbf{y}^{(i)}|\mathbf{z}^{(i)}\right)\right)$$

Basic Machine learning Concepts: Review

- Overfitting: the learned model to the training set. Extreme example: The data is the model.
- Generalization: the data is a sample, there is vast amount of samples that you have never seen. Your model should generalize well to these "never-seen" samples.
- Bias-Variance tradeoff: Richer vs constrained models. Example: high treewidth vs low treewidth models
  - Can learn low treewidth models (Example: learning trees is easy). However, a tree may not represent all independencies of P^* (not a minimal I-map).
  - Cannot learn high treewidth models (limited data). However, they may be closer to P^*.

Basic Machine learning Concepts: Review

- Regularization: Encode a soft constraint for simpler models in our objective function.
  - Note: Restricting our model class reduces over-fitting. This imposes a hard constraint. Regularization is a soft constraint.
- Training versus Test-set: Hold out some data as test data.
  - k-fold cross validation: A special way of holding out data. Divide the data into k bins. Run your algorithm k times. Each time use the i-th bin as test data.

Data Observability

- Fully observed: Complete data so that each of our training instances is an assignment of values to all variables
- Partially observed: There exists training instances t such that one or more variables in t are not observed (missing values)
- Hidden variables: The data contains hidden variables whose value is never observed.