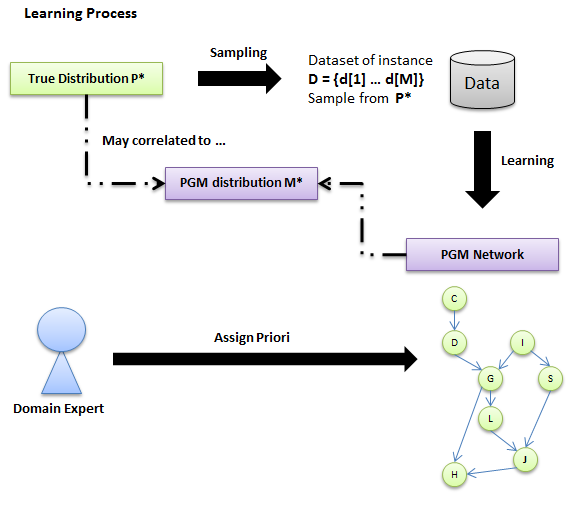
**PGM Learning Tasks & Metrics –**

**Why you want to use PGM learning rather than other machine learning algorithms?**

Useful when it’s **not** simple yes/no binary prediction but for prediction over a structured objects like sequence, graphs, trees for NLP or speech recognition. Allows explore correlations between several predicted variables gives increase on performance. It can also incorporate prior knowledge into model which is not easy for other models. It also multi-functional in which you can learn single model for multiple tasks (I, II, III).

****

**Situations before learning:**

1. **Known network structure + Complete data => Network + CPD**
2. **Unknown network structure + Complete data => Network + CPD**
3. **Known network structure + Incomplete data => Network + CPD**
4. **Unknown network structure + Incomplete data => Network + CPD**
5. **Latent variables + Incomplete data => Network + CPD**

**PGM Learning Task I:**

**Goal:** Answer general probabilistic queries (conditional probability or MAP) about new instances

**Simple metric**: Trainset likelihood (The probability of model M\* generates the sampled data D – assume independently and identical distributed **IID**)

**Eventually** we want to test M\* on **new data** for test result

**PGM Learning Task II**:

**Goal:** Specific predication task on new instances, predict target variables y from observed variables x; ex. image segmentation, speech recognition

**Often care about specialized objectives**: ex. pixel-level segmentation accuracy

**Often convenient to select model to optimize**: likelihood P(D:M\*) or conditional likelihood P(Y|X :M\*)

**Important to evaluate the true business objective on the test data**

**PGM Learning Task III**:

**Goal:** Knowledge discovery of M\* structure

* Distinguish direct vs indirect dependencies
* Possibly directionality of edges
* Presence and location of hidden variables

**Often train using likelihood**: poor surrogate for structural accuracy

**Evaluate by comparing to prior knowledge about model M\***

**Avoiding over fitting:**

**Select M to optimize train set likelihood tends to over fit model to statistical noise.**

**Parameter over fitting:**

* Parameters fit random noise in training data
* Use regularization / parameter priors

**Structure over fitting**:

* Training likelihood always increases for more complex structure
* Bound or penalize model complexity

**Selecting Hyperparameters**:

* Regularization for overfitting involves hyperparameters: **Parameter priors** + **complexity penalty**
* **Choice of hyperparameters makes a big difference to performance**
* **Create validation set to perform cross-validation to select best parameters**

**PGM Learning Tasks & Metrics – Maximum Likelihood Estimation**

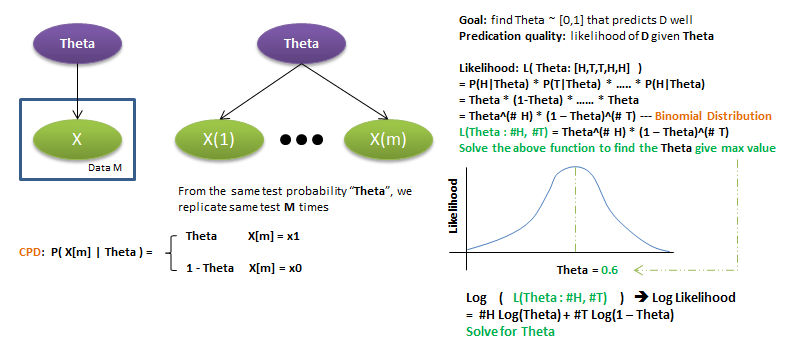
**MLE Principle:** Choose **theta** to maximize **L(Theta: D )**

**Bernoulli distribution:**

**P is a Bernoulli distribution:**

P(X=1) = **Theta**, P(X=0) = 1 - **Theta**

**D** = [X(1) … X(m)] sampled (case are Independent, identical distributed – from the same distribution (**IID**)) from **P**

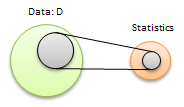


**Sufficient Statistics:**

**For example – D = [X(1) …. X(m)] => #H and #T , those are the sufficient statistics**

**A function S(D) is a Sufficient Statistics from instances if:**

**S(D) = S(D’) -> L(Theta: D) = L(Theta: D’)**

****

**Sufficient Statistics – Multinomial Distribution:**

**Sufficient Statistics: For a dataset D over variable X with K values, the sufficient statistics are counts <m1,…,mk> where Mi is the # of times that X[m] = x’ in D**

**S(xi) = (0,0,…,1,0,0) K dimension, the true value = 1, other = 0, Sum up all I X by k dimensions, get (#, #, ….., #)**

**Multinomial Likelihood function:** L( Theta: D ) = Product(Theta(i) where i in 1:k) ^ Mi

**Sufficient Statistics – Gaussian Distribution:**

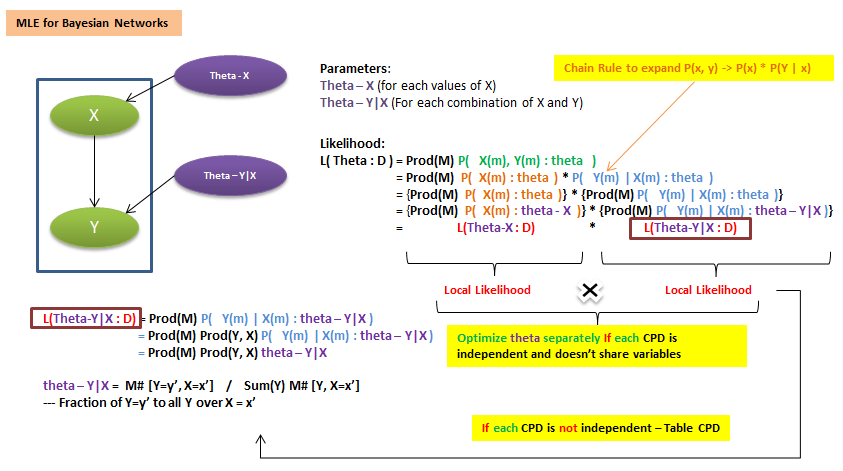
**P(X) ~ N(u, sd^2)**

**= rewrite equation (u, sd, x)**

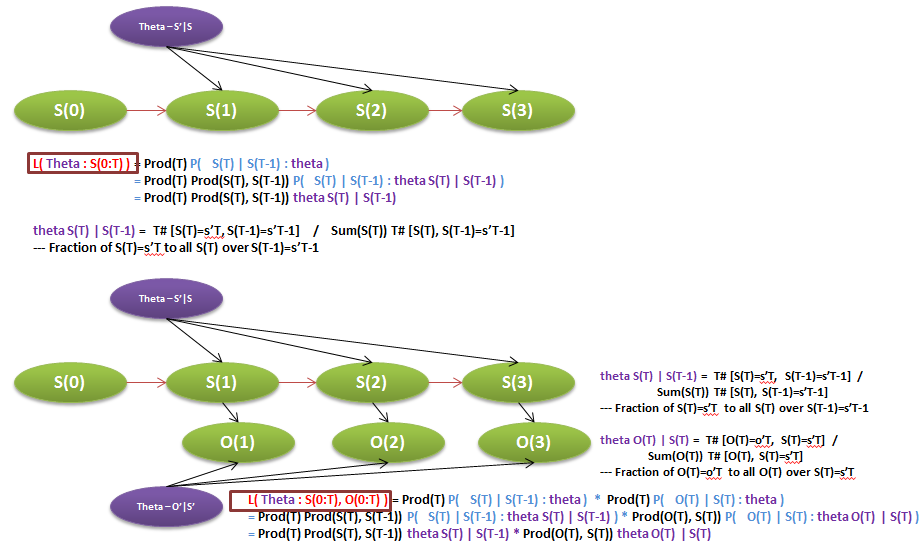
**Sufficient Statistics: S(x) = <1,x,X^2> --- S(D) = (sum(m) x^2(m), sum(m) x(m), m)**

**Gaussian Likelihood function**: L( mean, sd: D ) => **u** = equation (x), **sd** = equation (x, u)

**PGM Learning Tasks & Metrics – Max Likelihood for BNs**

****

**Example : Shared Parameters**

****

**Summary:**

* **For BN with disjoint sets of parameters in CPDs, likelihood decomposes as product of local likelihood functions, one per variable - P( X(m) : theta )**
* **For table CPDs, local likelihood further decomposes as product of likelihood for multinomial, one for each parent combination - P( Y(m) | X(m) : theta )**
* **For network has shared CPDs, sufficient statistics accumulate over all uses of CPD –** like sequence models to counts, with T, cumulated sufficient statistics for calculation

**Fragmentation & Overfitting:**

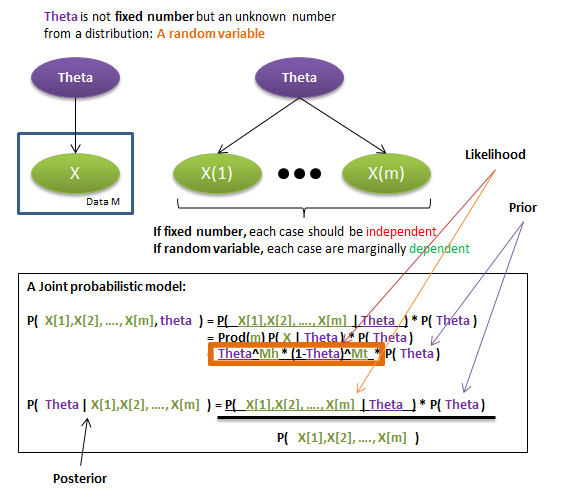
**# of “buckets” increases exponentially with number of parent variables in table CPDs, so instances in a large number U for each bucket is very little, not good estimation. Therefore, with limited data, we should use simple structure for the model.**

**PGM Learning Tasks & Metrics – Bayesian Estimation**

**Limitation of MLE: ex. 7 out of times = 0.7 which is the same as 7,000 out of 10,000 time. However, the later one should have a high certainty but MLE can’t tell the difference.**

**Bayesian Estimation:**

**Treats parameters as random variables (Learning is then a special case of inference)**

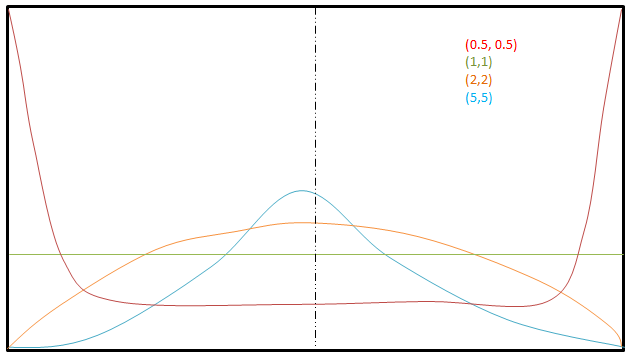
****

**Random variable for theta: Dirichlet Distribution**

**A multinomial distribution over K values – Dirichlet Distribution – Theta ~ Dirichlet (a1, …, ak)**

**Hyperparameters correspond to the number of samples we have seen**

**\*More higher the number, more centralized in distribution (Weighted by sample number – ex.7,000)**

****

**Dirichlet Priors & Posteriors:**

**P( Theta | D ) proportional to P( D | Theta ) \* P( Theta )**

**When likelihood P( D | Theta ) is [multinomial distribution] --- Ex.** Data counts (M1,…,Mk)

**AND prior P( Theta ) is [Dirichlet Distribution] --- Ex.** Dir(a1,…,ak)

**Then the Posterior P( Theta | D ) is also [Dirichlet Distribution] \*Conjugate Pair Update --- Ex.** Dir(a1+M1,…,ak+Mk)

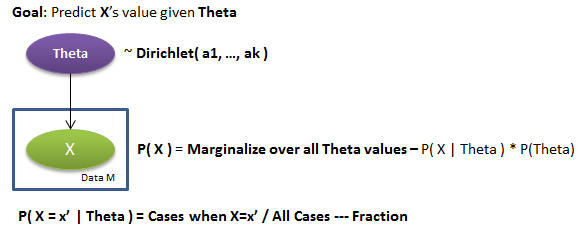
**Conjugate Pair Update: Posterior has the same form as prior, can be updated in closed form using sufficient statistics from data**

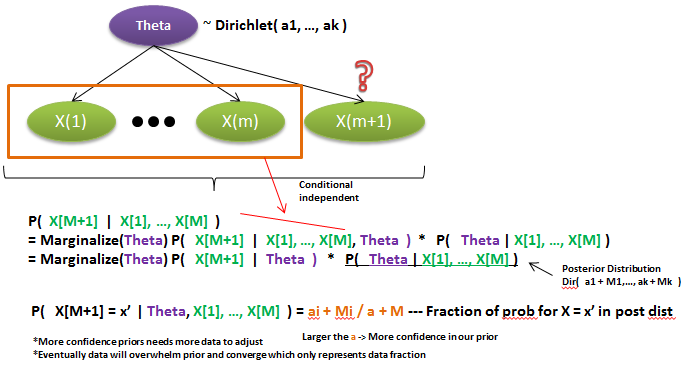
**PGM Learning Tasks & Metrics – Bayesian Prediction**

**Bayesian Prediction combines sufficient statistics from two groups – real sample data + imaginary Dirichlet sample (Prior distribution) => a + M**

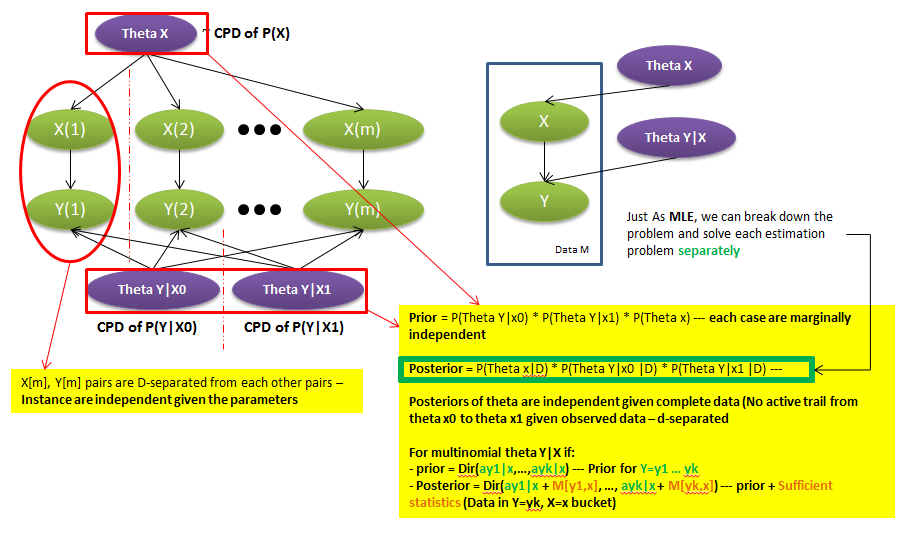
**As data increased, eventually data will dominate the prediction and the result will be the same as MLE**

**Dirichlet Parameters shows the prior belief and strength**

****



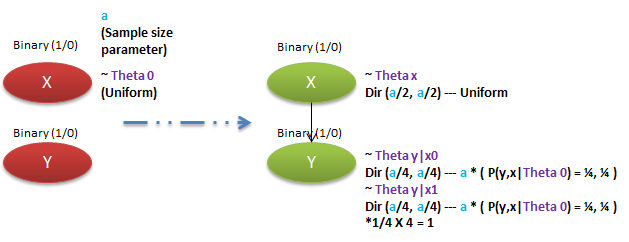
**PGM Learning Tasks & Metrics – Bayesian Estimation for Bayesian Network**

****

**Assessing Priors for BNs:**

We need hyperparameters **a(x|u)** for each node **X**, value **x’’’**, and parent assignment **u**

* **Prior** network with parameters **Theta 0 –** like **Uniform Distribution**
* Equivalent **sample size** parameter **a (If large a – strong prior – takes more data to adjust)**
* **Prior a(x|u) := a \* P(x,u|Theta 0)**

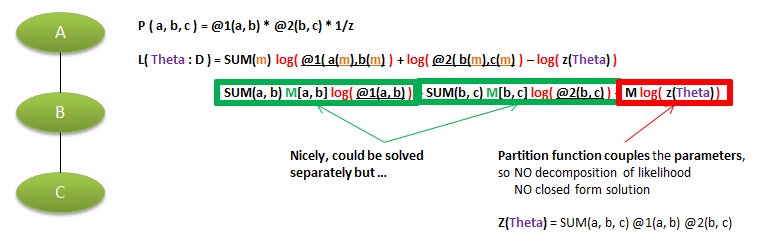
****

**PGM Learning Tasks & Metrics – Learning undirected model**

**We study the parameter estimation problem for Markov Networks (MN).**

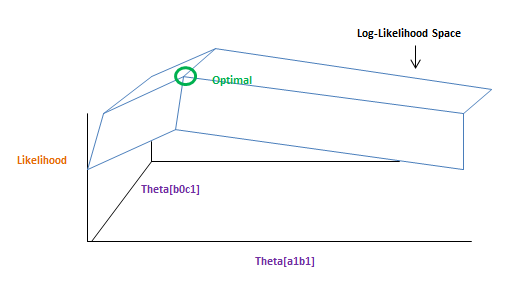
**PGM Learning Tasks & Metrics – Maximum Likelihood for Log-Linear Models**

**Log likelihood in Markov Network (Markov Random Field – MRF)**

****

**Example – Log-Likelihood Function**

**P (A, B, C) = 1/z exp ( Theta[a1b1] I (a1,b1) + Theta[b0c1] I (b0, c1) )**

****

**Log-Likelihood for general log-linear model**

**P( X1,…,Xn : Theta ) = 1/z( Theta ) exp SUM(k) { Theta(k) \* feature-k( D(k) ) }**

**L( Theta : D ) = SUM(k) Theta(k) { SUM(m) feature-k( x(m) ) } – M log( Z(theta) )**

**\*Sum over each theta k, for each k, sum over each data instance m**

**Log( Z (theta) ) = log SUM(exponential space-x) exp {SUM(k) theta(k) \* feature-k(x)**

**The log-partition function --- Log( Z (theta) )**

**Theorem:**

**Derivative\_theta(k) ( Log(Z(theta)) ) = E(Feature-k) :** Expectation value of feature-k distribution over theta k parameterition **(A vector of derivative for theta 1 to K)**

**Derivative\_pairs[theta-k, theta-j] ( Log(Z(theta)) ) = Cov(feature-k; feature-i) :** Covariance between random variable feature k and feature i **(A vector of derivative for theta 1 to K)**

**Therefore, for Log( Z (theta) ):**

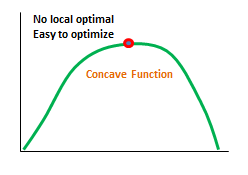
**Log-partition function is convex -> subtracted by “-“ -> negation to concave**

**Therefore, for Log-likelihood function L( Theta : D ):**

**SUM(k) Theta(k) { SUM(m) feature-k( x(m) ) } ------- Linear function**

**– M log( Z(theta) ) ------------------------------------------- Concave function**

**So, Linear function X Concave function = L( Theta : D ) which is still concave function**

****

**Maximum Likelihood Estimation (MLE)**

**1/m L( Theta : D ) = SUM(k) Theta(k) { 1/m SUM(m) feature-k(x[m]) } – log( Z(theta) )**

**1/m:** avoid scaling issue when D increases

**Derivative\_theta(k) of “1/m L( Theta : D )”**

**= Derivative: SUM(k) Theta(k) { 1/m SUM(m) feature-k(x[m]) } ----- Expectation value from D**

**= Derivative: log( Z(theta) ) -------------------------------------------------------- Expectation value from model**

* **Derivative\_theta(k) of “1/m L( Theta : D )”**
* **= Expectation value from D - Expectation value from model**

**Theorem**: **Theta** is the **MLE** if and only if -> **Expectation value from D = Expectation value from model**

**\*It holds for every feature k**

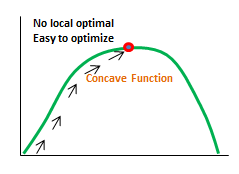
**Computation: Gradient Ascent**

**Derivative\_theta(k) of “1/m L( Theta : D )” = Expectation value from D - Expectation value from model**

**Above a concave function**

**Use gradient ascent – typically L-BFGS (a quasi-Newton method)**

**For gradient, need expected feature counts M: (in data) and (relative to current model) – require probabilistic inference at each gradient step (Cost of computation compares to BN)**

****

**Features are always within clusters in cluster graph or clique tree due to family preservation, so one calibration (Tune one time) is sufficient for all feature expectations**

**PGM Learning Tasks & Metrics – Maximum Likelihood for Conditional Random Field CRFs**

**Estimation for CRFs: (Conditional Random Field – CRF)**

**P( Y | X ) = 1/Zx(theta) un-normalized-P( X, Y ), Zx(theta) = SUM(Y) un-normalized-P( X, Y )**

**\*know Y probability given X / only normalize on Y**

**D = {( x[m], y[m] )}M,m=1 \*pair of data x,y**

**Conditional Log-likelihood function: L(y|x) ( theta : D ) = SUM(m) log( P( y[m] | x[m], theta ) )**

**L(y|x) ( theta : D ) = L(y|x) ( theta : ( x[m], y[m] ) )**

**= SUM(k) theta(k) \* feature-k(x[m], y[m]) – log( Zx[m]( theta ) )**

**Derivative\_theta(k) of “1/m L(y|x) ( theta : D )”**

**= 1/m SUM(m) feature-k( x[m], y[m] ) – E-model(theta) ( feature-k(x[m], Y) )**

**\*Last term, X is fixed and only get expectation from Y**

**= Expectation value from D - Expectation value from model**

**Example:**

**Feature1(Ys, Xs) = 1(Ys=g) \* Gs**

**Feature2(Ys, Yt) = 1(Ys=Yt)**

**Derivative L(y|x) ( theta : ( x[m], y[m] ) ) = feature-k(x[m], y[m]) – Emodel[ feature-k( x[m], Y) ]**

**Derivative1 = SUM(s) 1{ Ys[m] = g } Gs[m] – SUM(s) Pmodel( Ys = g | X[m] ) Gs[m]**

**Derivative2 = SUM(s,t) 1{ Ys[m] = Yt[m] } – SUM(s,t) Pmodel( Ys = Yt | X[m] )**

**Computation:**

**MRF – Derivative – Requires inference at each gradient step**

**CRF – Derivative – Requires inference for each X[m] at each gradient step**

**However, for inference of P( Y | X ), we need to compute distribution only over Y – usually much simpler. If we learn an MRF, need to compute P( Y, X ), which may be much more complex**

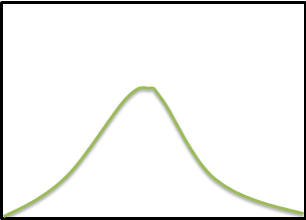
**CRF learning very similar to MRF learning –**

* **Likelihood function is concave**
* **Optimized using gradient ascent (Usually L-BFGS)**

**PGM Learning Tasks & Metrics – MAP Estimation for MRFs, CRFs (Maximize A Posterior)**

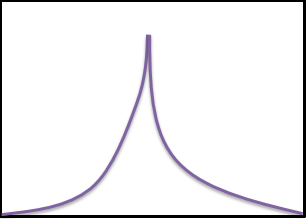
**Gaussian Parameter Prior:**

**P ( Theta : Alpha^2 ) – Mean, SD / Hyper-parameter – Alpha**

****

**Laplacian Parameter Prior:**

**P ( Theta : Beta ) – Mean, SD / Hyper-parameter - Beta**

****

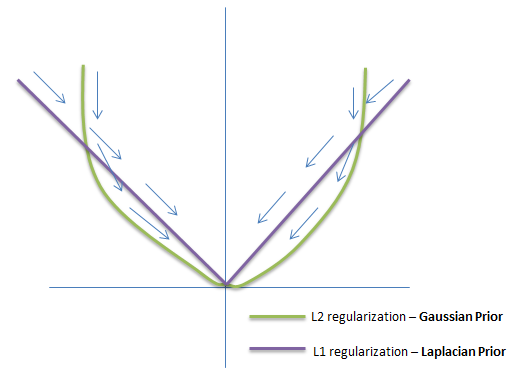
**MAP Estimation & Regularization:**

**With prior – P ( Theta : Alpha^2 ) ; P ( Theta : Beta )**

**Likelihood - Argmax(theta) P( D, theta ) = argmax(theta) P( D | theta ) P(Theta)**

**Logarithm (Likelihood) = argmax(theta) { L( that : D ) + log( P(theta) ) }**

**log( P(theta) ) ----- Regularization term**

****

**Summary:**

**In undirected models, parameter coupling prevents efficient Bayesian estimation. But we can still use parameter priors to avoid overfitting of MLE (MAP) – typical use L1, L2 priors.**

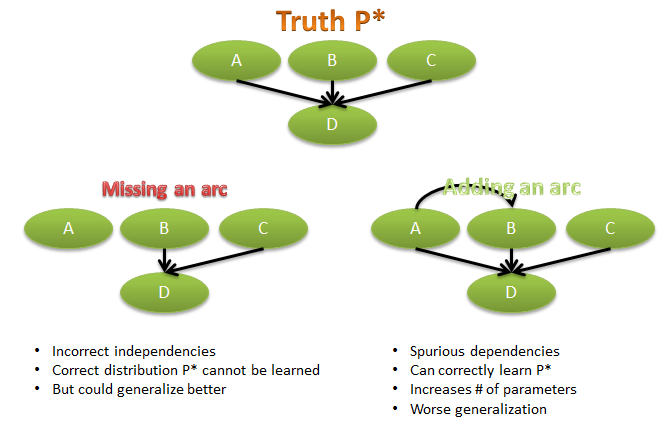
**L1 can induce sparse solution (Less variables) due to its ability to feature selection.**

**PGM Learning Tasks & Metrics – Structure Learning**

**Why we need structure learning?**

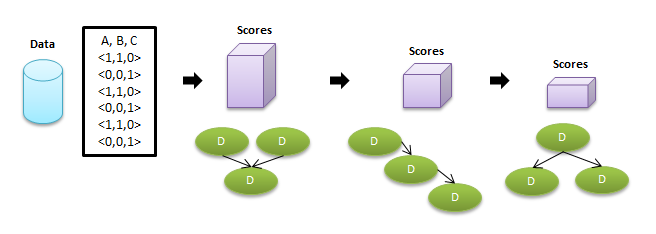
* **To learn model for new queries, when domain expertise is not perfect**
* **For structure discovery, when inferring network structure is good in itself**

**Important of Accurate Structure:**

****

**Score-Based Learning:**

**Define scoring function that evaluates how well a structure matches the data**

****

**It is an algorithm that optimizes “Score” by searching through the space of all structure combinations**

**PGM Learning Tasks & Metrics – Likelihood Structure Score**

**BIC score explicitly penalizes model – complexity (# of independent parameters)**

**Likelihood Score:**

**Find (Graph, theta) that maximize the likelihood**

**Likelihood score - Score L( Graph : D ) = L( [theta, Graph], D )**

**For Graph, we figure out the best fitting theta, then we calculate the log-likelihood given this graph and theta which is the likelihood score for this Graph**

**Score L( Graph0 : D ) - Score L( Graph1 : D ) = ? (the positive or negative)**

**Limitation:**

**More edge always helps, full connected graph maximized the score.**

**How to avoid the limitation which causes over fitting?**

* **Restricting the hypothesis space (More easy to implement)**
  + **Restrict # of parents or # of parameters**
* **Scores that penalize complexity (Better than hard constraint above which can’t consider real relationships)**
  + **Bayesian score averages over all possible parameters values**
  + **Explicitly**

**PGM Learning Tasks & Metrics – BIC Score \ Asymptotic Consistency**

**BIC Score function - Which adds penalty to likelihood score function. / Asymptotically equivalent to BIC / Asymptotically consistent / But for small M (# instances), BIC tends to underfit**

**Score BIC ( Graph : D ) = L([theta, Graph], D ) – (log M / 2) Dim[Graph]**

**\*M=#raining instance / Dim[Graph]=#independent parameters**

**= Likelihood Score – Penalty**

**(Trade-off between fit to data and model complexity) – Asymptotic Behavior**

**-Mutual information grows linearly with M while complexity grows logarithmically with M, so as M grows, more emphasis is given to “fit to data”. As M-> infinite, the true structure Graph \* maximizes the score (Consistence). Asymptotically, spurious edges will not contribute to likelihood and will be “penalized”. Required edges will be added due to linear growth of likelihood term compared to logarithmic growth of model complexity.**

**BIC is asymptotically consistent:**

**If data generated by Graph \*, networks I-equivalent to G\* will have highest score as M grows to infinite.**

**PGM Learning Tasks & Metrics – Bayesian Score**

**Bayesian score averages over parameters to avoid overfitting.**

**Find a graph that maximize:**

**P ( Graph | D ) = P ( D | Graph ) \* P(Graph) / P (D)**

**= Marginal likelihood \* Prior.over.structure / Marginal.probability.of.data**

**Bayesian score - Score B ( Graph : D ) = log P( D | Graph ) + log P( Graph )**

**Marginal likelihood – P ( D | Graph ) = P ( D | theta(prior) ) – over all possible prior parameter theta**

**= P ( x[1],…,x[M] | Graph ) --- Leverage gamma function**

**Prior.over.structure – P ( Graph ) = structure prior P(G)**

**= Uniform prior: ~ constant**

**= Prior penalizing # of edges ~ C|G| (0<c<1)**

**= Prior penalizing # of parameters**

**Prior.over.Parameter – P ( Theta | Graph ) = BDe Prior ( As before prior network initiation)**

**= a: equivalent sample size**

**= B0: network representing prior probability of events**

**\*A single network provides priors for all candidate networks**

**\*BDe requires assessing prior network / Can naturally incorporate prior knowledge / I-equivalent networks have same score**

**PGM Learning Tasks & Metrics – Search over structure – Tree Structured Networks**

**Structure learning is an optimization over the combinatorial space of graph structures. Decomposability -> allows us to break that up to different families -> network score is a sum of terms for different families -> optimal tree can be find via standard MST (Max-weight spanning tree) algorithms**

**Optimization Problem:**

**Input –**

* **Training data**
* **Scoring function (Including priors, if needed)**
* **Set of possible structures**

**Output –**

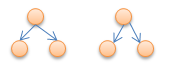
* **A network that maximizes the score**

**Key Property –**

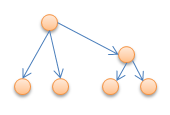
* **Decomposability**

**Learning Trees / Forests:**

**Forest – At most one parent per variable**

****

**Tree - Why we use tree structure? – Elegant math / Efficient optimization / Sparse parameterization**

****

**Learning Forest -**

**P(i) = parent of Xi, or 0 if Xi has no parent**

**Score(Graph : D) = SUM(i) Score(Xi | P(i) : D)**

**Score(Xi : D) - nodes has no parent**

**Score(Xi | P(i) : D) – nodes has parent**

**= SUM(i) { Score(Xi | P(i) : D) – Score(Xi : D) } + SUM(i) Score(Xi : D)**

**= edge scores + parent scores = score**

**Learning Forest II –**

**Set W(I ->j) = Score( Xj | Xi ) – Score( Xj ), weight from I to j**

**For likelihood score, W(i->j), all edge weights are nonnegative - > Optimal structure is always a tree (connected)**

**For BIC or BDe, weights can be negative - > Optimal structure might be a forest**

**Learning Forest III –**

**A score satisfies score equivalence if I-equivalent structures have the same score – Such scores include likelihood, BIC and BDe. For such score, we can show W(i->j) = W(j->i), and use an undirected graph.**

****

**Learning Forest IV –**

**Define a undirected graph with nodes (1,…,n)**

**Set W(I,j) = max[ Score( Xj | Xi ) – Score( Xj ), 0 ] \*** Optimize things without to worry about negative edges

**Optimize to find a tree or forest with maximal weight!**

* **Standard algorithms for max-weight spanning trees in O(n^2) time**
* **Remove all edges of weight 0 to produce a forest**

**PGM Learning Tasks & Metrics – Search over structure – Learning general Graphs: Heuristic search**

**To learn a non-tree structured network.**

**Useful for building better predictive models:**

* **When domain experts don’t know the structure**
* **For knowledge discovery**

**Finding highest scoring structure is NP-hard. Typically solved with simple heuristic search.**

**Optimization Problem:**

**Input:**

* **Training data**
* **Scoring function**
* **Set of possible structures**

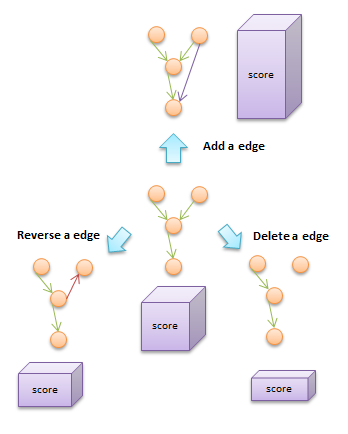
**Output:**

* **A network that maximizes the score**

**Non-tree structure problem:**

**Problem is not obvious for general networks: Ex. Allowing two parents, greedy algorithm is no longer guaranteed to find the optimal network. -> Theorem:** Finding maximal scoring network structure with at most K parents for each variable is NP-hard for K > 1

**So, it is in a standard Heuristic Search:**

****

**Design choices:**

* **Search operators:**
  + **Local steps: edge addition, deletion, reversal (small step)**
  + **Global steps: take entire node out and stick into somewhere else (large step)**
* **Search techniques:**
  + **Greedy hill climbing**
  + **Best first search**
  + **Simulated annealing**
  + **…**

**🡺Greedy hill climbing:**

1. **Start with a given network**
   1. **Empty network**
   2. **Best tree**
   3. **A random network**
   4. **Prior knowledge**
2. **At each iteration:**
   1. **Consider score for all possible changes**
   2. **Apply change that most improves the score**
3. **Stop when no modification improves score**

**🡪Greedy hill climbing Pitfalls:**

**Can get stuck in – local maxima, plateaux (all flat around – very similar neighbors)**

**🡪Why edge reversal? – If delete and then add costs much more than directly reverse the edge.**

**Solution for pitfalls:**

* **Random restart: When gets stuck, take some number of random steps and then start climbing again**
* **Tabu list:** 
  + **Keep a list of K steps most recently taken**
  + **Search cannot reverse any of these steps**

**PGM Learning Tasks & Metrics – Search over structure – Learning general Graphs: Search/decomposability**

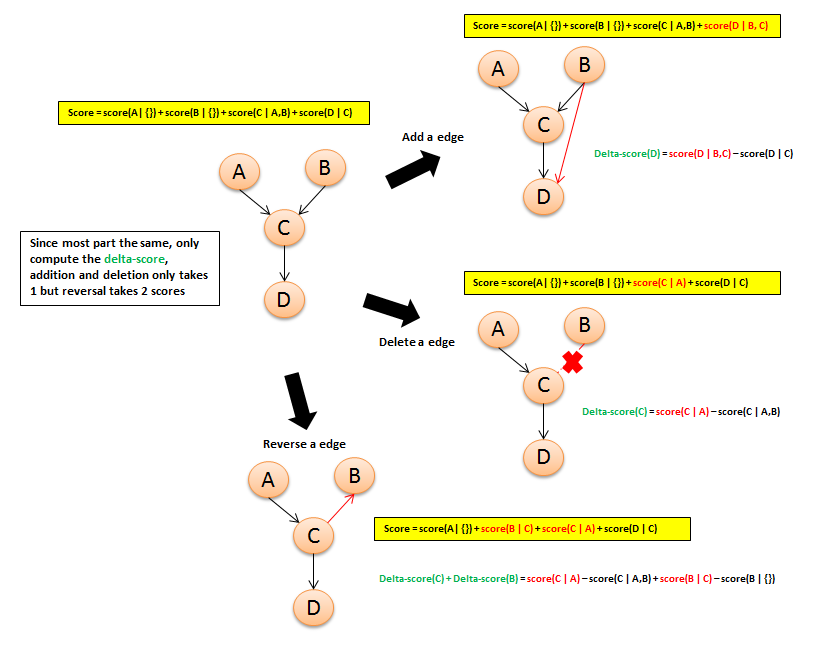
**Search process – Heuristic Search 🡪 Naïve Computational Analysis:**

* **Operators per search step: n(n-1) possible edges, each edge 3 ways to treat it**
* **Cost per network evaluation:** 
  + **Components in score**
  + **Compute sufficient statistics**
  + **Acyclicity check**

**Total: O(n^2 (Mn + m)) per search step 🡪 can get extremely large when n is large**

**How to solve?**

**Exploiting Decomposability:**

****

**PGM Learning Tasks & Metrics – Learning with incomplete Data Overview**

**The incomplete data situation arises very often. There are also some challenges when we try to deal this situation –**

* **We need to understand the mechanism for missingness**
* **Since multi-modal likelihood function, the global optimal solution can be multiple**
* **Parameters can be correlated to each other**
* **Multimodal likelihood function is complex to deal with**

**Incomplete Data –**

* **Hidden Variables**
* **Missing Values**

**Challenges –**

* **Foundational –** is the learning task well defined**?**
* **Computational –** how can we learn with incomplete data**?**

**Why we need to identify Hidden Variables?**

* **Much lesser parameters in the computation**
* **Discovering clusters in the data (summarize many nodes into few nodes)**

**Treating missing Data:**

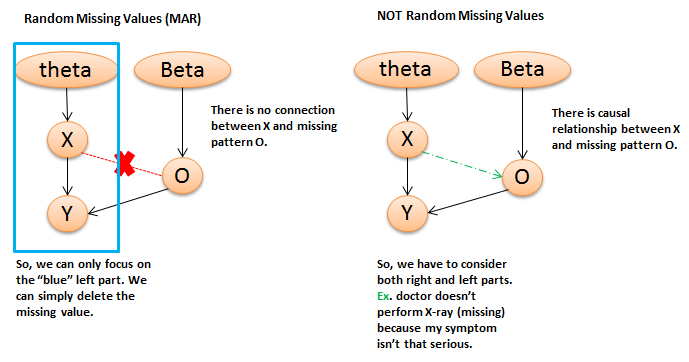
* **Missing at random (MAR) – can delete missing value**
* **Missing NOT at random (MNAR) – imputation (Modeling missing data mechanism)**

**Modeling Missing Data Mechanism:**

**X = {X1,…,Xn} are random variables**

**O = {O1,…,On} are “observability variables 1/0 ” --- always observed!**

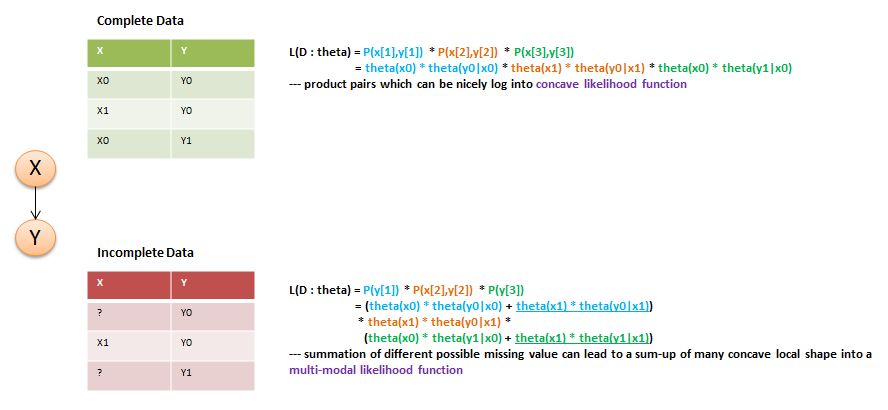
**Y = {Y1,…,Yn} are new random variables –If O=1,Y=X, otherwise, Y=? --- always observed!**

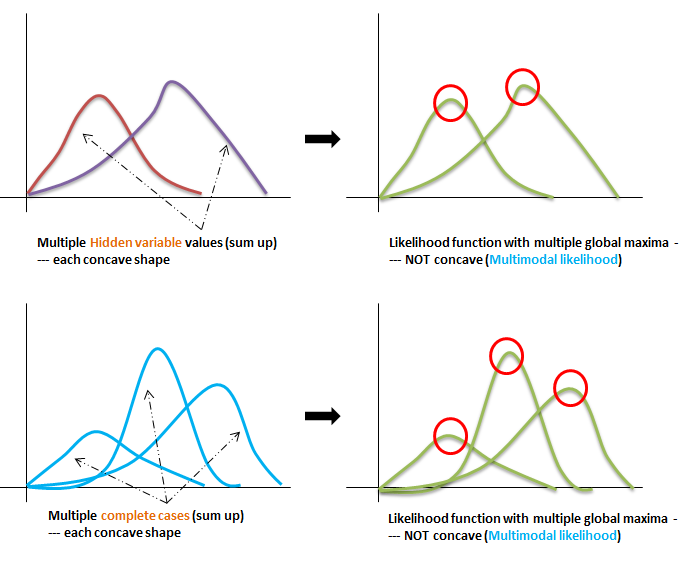
****

**Multi-modality:**

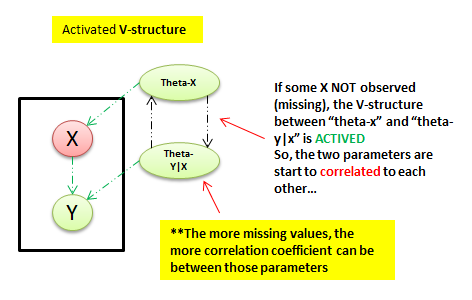
**Likelihood can have multiple global maxima when hidden variable has multiple values. If the number of hidden variable increased, the # of global maxima increases exponentially.**

**Multiple local and global maxima can also occur with missing data (Not only hidden variables)**

****

****

**Parameter Correlations:**

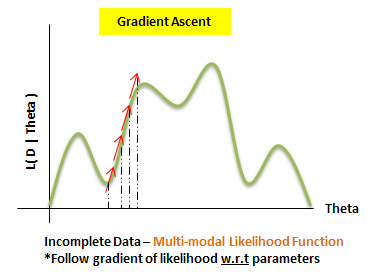
****

**PGM Learning Tasks & Metrics – Likelihood optimization (missing data) - Expectation Maximization (EM)**

**Usually two approaches to optimize the Multi-modal likelihood function:**

* **General Optimization: ex. <Gradient Ascent>**
* **Specialized Optimization: ex. <Expectation Maximization (EM)>**

**Gradient Ascent:**

****

**Luckily, we have a close form solution for solving & computing the Gradient:**

**Theorem:**

**Gradient of Log-likelihood function log P( D | prior-theta ) relative to particular theta-x(i)|U(i)**

**= 1/** **theta-x(i)|U(i) SUM(m) P( x(i), u(i) | d[m], prior-theta )**

**\*Sum of probabilities of all m evidence for P for X=xi, U=ui, given prior-theta**

**Gradient Ascent 🡪 Need to run inference over each data instance at every iteration (Expensive)**

* **Pros:** Flexible, can be extended to non-table CPDs
* **Cons:** Need define legal constrained optimization, non-zero, sum up to 1, etc / Fpr reasonable convergence, need to combine with advanced methods (Conjugate gradient, line search), however, it increases computational cost.

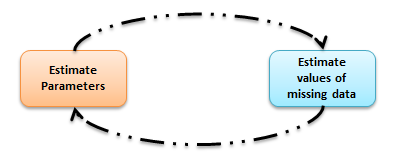
**Expectation Maximization (EM):**

**Special-purpose algorithm designed for optimizing likelihood functions. Commonly used in practice.**

**Intuition:**

**Chicken-egg issue ->**

* **Parameter estimation is easy given complete data**
* **Computing probability of “missing data” which gives complete data is easy given parameters**

****

**Process:**

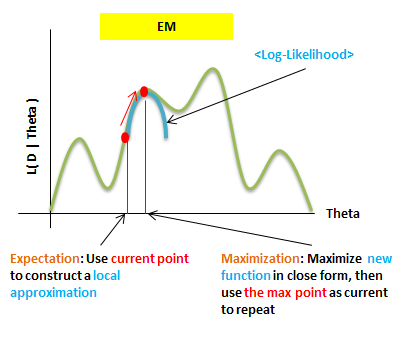
1. **Pick a starting point for the parameters**
2. **Iterate:**
   1. **E-Step (Expectation): “Complete” the data using current parameters**
   2. **M-Step (Maximization): Estimate parameters relative to data completion (MLE)**
   3. **Repeat the previous two steps till convergence**

**\*Guaranteed to improve Likelihood – L ( Theta : D ) at each iteration**

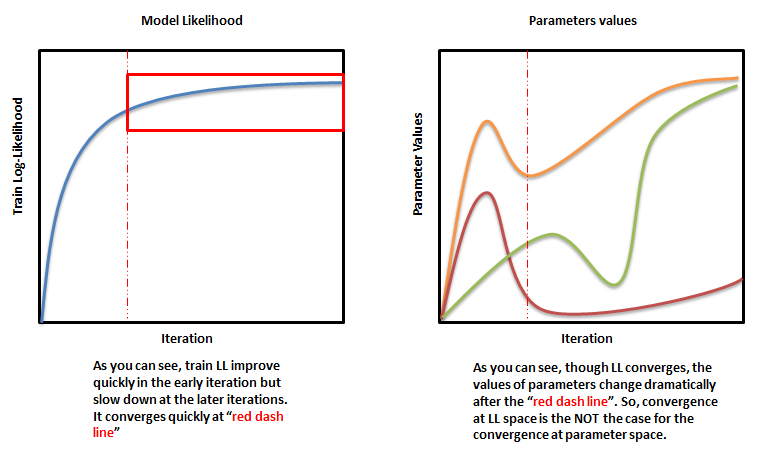
**Expectation Maximization 🡪 Need to run inference over each data instance at every iteration (Expensive)**

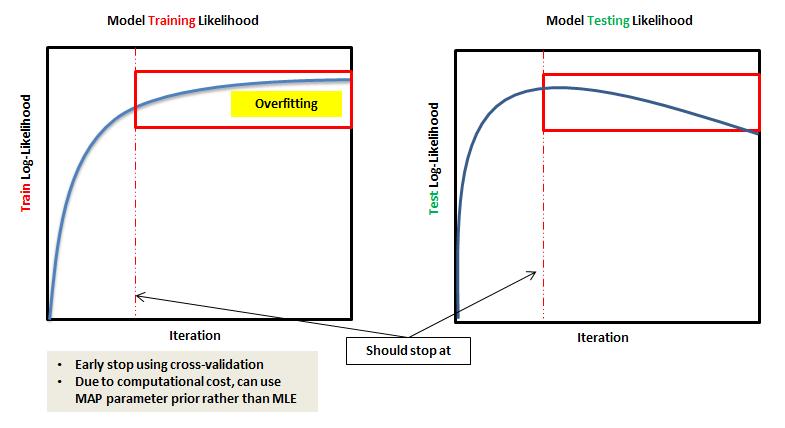
* **Pros:** Easy to implement on top of **MLE** for complete data / Make rapid progress, especially in the early iterations
* **Cons:** Convergence slows down at later iterations, so need to switch to “conjugate gradient, etc” to finish

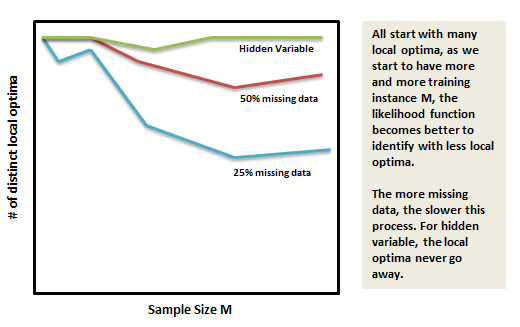
**PGM Learning Tasks & Metrics – Learning with incomplete Data – Analysis of EM Algorithm**

****

**PGM Learning Tasks & Metrics – Learning with incomplete Data – EM in Practice**

****

****

****

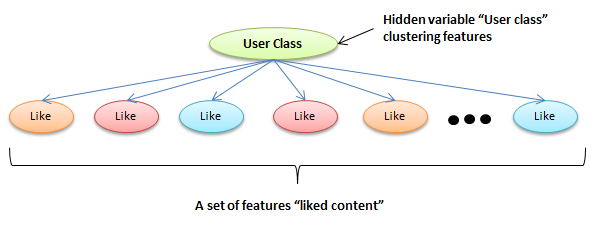
**\*Each local optimum’s likelihood value can be dramatically different. So we need to be careful to NOT to stuck on local optima. Initialization is critical:**

* **Use multiple random starts (Randomly select initial parameter values)**
* **Use prior knowledge to choose prior value**
* **From the output of a simpler clustering algorithm (K-mean, Hierarchy) rather than EM**

**PGM Learning Tasks & Metrics – Learning with incomplete Data – Hidden Variable**

**Most common use of EM is to deal with hidden variable. Often, a critical component in constructing models for richly structured domains.**

**Latent variables satisfy Missing at random (MAR), so can use EM.**

****

**\*Important to pick the values for hidden variable – impact likelihood function**

* **If we use likelihood for evaluation, more values is always better**
* **Can use scores that penalizes complexity**
  + **BIC – tends to under fit**
  + **Extensions of BDe to incomplete data (Approximation of Bayesian score)**
* **Can use metrics of cluster coherence to decide whether to add/remove cluster**
* **Bayesian methods (Dirichlet process) can leverage over different cardinalities**

**PGM Learning Tasks & Metrics – Learning Summary**

**Typical Learning Loop:**

1. **Design model “template” – Graph’**
2. **Select hyper-parameters via CV an training set**
3. **Training on training set with chosen hyper-parameters**
4. **Evaluate performance on held-out set**
5. **Error analysis & Model redesign (Go back to previous to repeat)**
6. **Done. Then, Report result on separate test set**

**Learning Problem:**

* **Hypothesis (Model-Graph) space**
* **Objective function**
* **Optimization algorithm**

**------------------------------------------------------------------------------------------------------------------------------------------**

**#1 - Hypothesis (Model-Graph) space**

**We are searching for:**

* **Parameters**
* **Structure**

**Imposing constraints:**

* **For computational efficiency**
* **To reduce model capacity**
* **To incorporate prior knowledge**

**#2 - Objective function**

* **Penalized likelihood**
  + **L((G, theta-prior) : D) + R(G, theta-prior)**
  + **Parameter prior + Structured complexity penalty**
* **Bayesian Score**
  + **Log P( G | D ) + Log P(G) + Constant**

**#3 – Optimization algorithm**

* **Continuous**
  + **Closed form – multi-normal**
  + **Gradient Ascent – no close form**
  + **EM – missing data**
* **Discrete**
  + **Max Spanning Tree (MST)**
  + **Hill Climbing – add, delete, reverse**
* **Discrete & Continuous – computationally expensive**

**Hyperparameters:**

**Estimate them on validation set – cross validation**

* **Equivalent sample size for parameter prior**
* **Regularization strength for L1 and L2**
* **Stopping criterion for EM**
* **Strength of structure penalty**
* **Set of features**
* **# of values in hidden variable (# clusters)**

**Model Evaluation Criterion:**

* **Log-likelihood on test set**
* **Task-specific objective – error rate**
* **“Match” prior knowledge – if you know how many cluster. Ex**

**Under-fitting:**

* **Train / Test set performance both low**
* **Solution:**
  + **Decrease regularization**
  + **Reduce structure penalty**
  + **Add features via error analysis**

**Over-fitting:**

* **Training performance high, test performance low**
* **Solution:**
  + **Increase regularization**
  + **Impose capacity constraints**
  + **Reduce feature set**

**Optimization:**

* **Optimization may not to converge to good or not converge (can happen even problem is convex)**
* **Solution:** 
  + **Compare different learning rates**
  + **Different random initializations**

**Objective function mis0match to true performance:**

* **Objective function not design for performance optimization**
* **Solution:** 
  + **Redesign objective function to fit the performance optimization problem**