Summary Topics

- Supervised Learning
  - Model Selection (i.e. Learning Parameters)
    - Frequentist and Bayesian
  - Learning algorithm evaluation
  - Assumptions on Data
  - Generative and Discriminative Classifiers
  - PAC Theory
- Supervise Learning Algorithms
- Unsupervised Learning (K-Means)
- Reinforcement Learning
Supervised Learning

• Given: Training examples
  \[ \{(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_P, f(x_P))\} \]
of some unknown function (system) \( y = f(x) \)

• Find \( \hat{f}(x) \) (i.e. an approximation)
  – Predict \( y' = \hat{f}(x') \), where \( x' \) is not in the training set

Two Types of Supervised Learning

• Classification \( y \in \{c_1, c_2, \ldots, c_N\} \)
  – Model output is a prediction that the input belongs to some class
  – If the input is an image, the output might be chair, face, dog, boat,... etc.

• Regression \( y \in \mathbb{R} \)
  – The output has infinitely many values
  – If the input is stock features, the output could be a prediction of tomorrow’s stock price
Goal of Supervised Learning

• Build a model that does best on Future Data!

Assumptions on Regression Data

• Data \((x_1, y_1), \ldots, (x_N, y_N)\) where \(x_i \in \mathbb{R}^d\) are independently identically distributed (iid) from \(D(x)\) and \(y \in \mathbb{R}\) are generated from \(y_i = f(x_i) + \rho\)

• where \(f(x) \in \mathbb{R}\) is a real valued function defined on \(x \in \mathbb{R}^d\) and \(\rho\) is a random variable

\[E[\rho] = 0, \quad \text{Var}[\rho] = c, \quad c \in \mathbb{R}, \quad 0 \leq c < \infty\]
Assumptions on Classification Data

- Assume data $(x_1, y_1), \ldots, (x_N, y_N)$ where $x_i \in \mathbb{R}^d$ and $y \in \{c_1, \ldots, c_K\}$ (i.e. K classes). The prior probability of each class is $p_k$ and each class is iid from a pdf $h_k(x)$. Then the posterior probability of class $c_k$ given $x$ is

$$\Pr(y = c_k | x) = \frac{p_k h_k(x)}{\sum_{i=1}^{K} p_i h_i(x)}$$

Building Supervised Learning Models: Frequentist Model Selection

The data $(x_1, y_1), \ldots, (x_N, y_N)$ is used with a Learning Algorithm to produce a Model $M(x)$ that makes predictions $\hat{y} = M(x)$. The Learning Parameters determine the choices of the Learning Algorithm and the Model.
Learning Parameters

• These dictate how the learning algorithm will build a model
• Changing the learning parameters changes how good the model is
• **Goal:** Choose the learning parameters that produce the best model

Measuring Model Accuracy: Regression

• Assume a set of data \((x_1, y_1), ..., (x_K, y_K)\)
• Regression accuracy
  – Two commonly used metrics
    • Mean Square Error
      \[
      \text{error}_{M(x)} = \frac{1}{K} \sum_{i=1}^{K} (y_i - M(x_i))^2 = \frac{1}{K} \sum_{i=1}^{K} (y_i - \hat{y}_i)^2
      \]
    • Relative Error
      \[
      \text{error}_{M(x)} = \frac{\sum_{i=1}^{K} (y_i - M(x_i))^2}{\sum_{i=1}^{K} (y_i - \bar{y})^2}
      \]
Measuring Model Accuracy: Classification

- Assume a set of data \((x_1, y_1), \ldots, (x_K, y_K)\)
- Classification accuracy

\[
error_{M(x)} = \frac{1}{K} \sum_{i=1}^{K} c(x_i, y_i, M(x_i))
\]

Where

\[
c(x_i, y_i, M(x_i)) = \begin{cases} 0 & \text{if } y_i = M(x_i) \\ 1 & \text{otherwise} \end{cases}
\]

Picking the Best Learning Parameters

- Partition learning data into disjoint sets
  - Training Set \((x_1, y_1), \ldots, (x_T, y_T)\)
    - Used to build the model
  - Validation Set \((x_1, y_1), \ldots, (x_V, y_V)\)
    - Used to evaluate model
- Pick the Learning Parameters that give the lowest error on the Validation Set

\[
error_{M(x)} = \frac{1}{V} \sum_{i=1}^{V} c(x_i, y_i, M(x_i))
\]
How Big Should the Training and Validation Sets Be?

- It Depends…
- If you have Lots of data for learning
  - Randomly putting half the data into each set is often sufficient
- If you only have a Small data set for learning
  - Usually do N-Fold Cross Validation

N-Fold Cross-Validation

- Partition the data $D_0 = \{(x_1, y_1), \ldots, (x_M, y_M)\}$ into N disjoint sets $T_1, \ldots, T_N$
- For i from 1 to N, do
  - Use $T_i$ for validation and the remaining $S_i$ for training
    - Training Set: $S_i = \{D_0 - T_i\}$
    - Error on validation $T_i$: $error_{T_i}$
- Return the average error on validation sets
  $$ error_{M(x)} = \frac{1}{N} \sum_{i=1}^{N} error_{T_i} $$
  
  Pick the learning parameters that minimize this error!
Does My Cross Validation Error Reflect the True Error of My Model?

• **No!!!!!!!!!!!!!!!!!!!!!!**
• Need to do randomized experiments
  – e.g. 100 experiments
    • 90% data for learning (use cross validation on this set to pick learning parameters)
    • 10% for testing
    • Report average test error over the 100 experiments

Bayesian Model Selection

• Pick the hypothesis that has maximum probability given the data – **Bayes Theorem:**

\[
P(h|D) = \frac{P(D|h)P(h)}{P(D)}
\]

  • \(P(h)\) = prior probability of hypothesis \(h\)
  • \(P(D)\) = prior probability of training data \(D\)
  • \(P(h|D)\) = probability of \(h\) given \(D\)
  • \(P(D|h)\) = probability of \(D\) given \(h\)

• **Learning parameters are chosen to maximize the probability of the hypothesis given the data**
Generative and Discriminative Classifiers

- **Generative Classifier Models**: model the distributions that generate the data (e.g. Bayesian density models.)
  \[
  \hat{y} = \arg \max_k \{ \hat{p}_k \hat{h}_k (x) \}
  \]

- **Discriminative classifier Models**: model only the boundaries (e.g. trees, SVMs, Nearest Neighbor, Neural Networks, etc.)

Supervised Learning Algorithms I

- Linear Regression
- Ridge Regression
  - Linear and Kernel
- Lasso Regression
  - Linear and Kernel
- Naïve Bayes Classifier
- Perception Classification
- Support Vector Machines
  - Classification and Regression
Supervised Learning Algorithms II

• K Nearest Neighbors
  – Classification and Regression
• Decision Trees
  – 1-R stump
• Neural Networks
  – Classification and Regression
• Bagging
  – Classification and Regression
• Random Forests
  – Classification and Regression
• Boosting Classifiers

Linear Regression

– Main Assumptions:
  • Linear weighted sum of attribute values.
  • Attributes and target values are real valued.
– Hypothesis Space
  • Fixed size (parametric) : Limited modeling potential
    \[ y = \sum_{i=1}^{d} \beta_i x_i + \beta_0 \]
  • Can be made non-linear using basis functions (now linear in basis function space)
    \[ y = \sum_{i=1}^{K} \beta_i \phi_i(x) + \beta_0 \]
Linear Regression Learning Algorithms

- Minimum Least Square Error
  \[ \hat{\beta}_{\text{MSE}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{d} \beta_j x_{ij} \right)^2 \right\} \]

- Ridge Regression
  \[ \hat{\beta}_{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{d} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{d} \beta_j^2 \right\} \]

- Lasso
  \[ \hat{\beta}_{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{d} \beta_j x_{ij} \right)^2 \right\} \]
  subject to: \[ \sum_{j=1}^{d} |\beta_j| \leq s, \quad s > 0 \]

Linear Regression Summary

- Good points
  - Does feature selection - LASO

- Bad points
  - Slow learning on very large datasets (>20,000)

- Software
  - LARS: http://www.stat.stanford.edu/~hastie/Papers/LARS/
Perceptron Algorithm: Finds a Linear Separating Hyper-Plane

\[ y = \text{sgn} \left[ \beta_0 + \sum_{i=1}^{d} \beta_i x_i \right] \]

Linear Hyperplanes

Linearly Separable

Not Linearly Separable
Nonlinear Perceptron Algorithm

- Use a nonlinear basis function space

\[ y = \text{sgn} \left[ \beta_0 + \sum_{i=1}^{K} \beta_i \varphi_i(x) \right] \]

- Basis functions can be kernels

Perceptron Algorithm

- Works by gradient descent

\[ \beta = \beta - \rho \left[ \frac{\partial L}{\partial \beta} \right] \]

\[ L(\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d) = -\sum_{i \in M} y_i \left( \hat{\beta}_0 + \left( \hat{\beta}_1, \ldots, \hat{\beta}_d \right)^T x_i \right) \]

where \( M \) is the set of misclassified training examples
Perceptron Summary

• Good points
  – Convergence guaranteed if problem is separable
    • In basis function space or linear space
  – Works on large data sets
    • Algorithm works by gradient descent

• Bad points
  – Won’t converge if data isn’t separable

• Learning Parameters
  – Learning rate, choice of nonlinear basis functions,…

Support Vector Machines

– Main Assumption:
  • Build a model using minimal number of training instances (Support Vectors).

– Hypothesis Space
  • Variable size (nonparametric): Can model any function given the right kernels
    – e.g. Gaussian
Linear Support Vector Machines

We’d like the hyperplane with maximum margin
- size of margin is $\frac{2}{||\mathbf{w}||}$

So view our problem as a constrained optimization problem:

Minimize $||\mathbf{w}||^2$, subject to

$$y_i(x_i \cdot \mathbf{w} + b) - 1 \geq 0, \forall i$$

What Happens When Data is Not Separable: Soft Margin SVM

Add a Slack Variable

$$\xi_i = \begin{cases} 0 & \text{if } x_i \text{ correctly classified} \\ \text{distance to margin} & \text{otherwise} \end{cases}$$
Soft Margin SVM: Constraint Optimization Problem

- Given data:
  \[(x_1, y_1), ..., (x_N, y_N)\]

- Minimize \[\frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \xi_i\] subject to:
  \[y_i (w^T x_i + b) \geq 1 - \xi_i, \quad \forall i = (1, ..., N)\]

Dual Problem (Non-separable data)

- Maximize
  \[W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle\]

- Subject to
  \[0 \leq \alpha_i \leq C, \quad i = 1, ..., N\]
  \[\sum_{i=1}^{N} \alpha_i y_i = 0\]
Mapping into Nonlinear Space

\[ \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]
\[ (x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2} x_1 x_2, x_2^2) \]

Kernel Trick

Replace \[ \langle \mathbf{x}_i, \mathbf{x}_j \rangle \]

with

\[ K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \]

Can use the same algorithms in nonlinear kernel space!
Nonlinear SVMs

Maximize:

\[ W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \]

Boundary:

\[ f(x) = \text{sgn} \left( \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b \right) \]

Need Mercer Kernels

\[ K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle \]

\[ = \langle \Phi(x_j), \Phi(x_i) \rangle \]

\[ = K(x_j, x_i) \]
Gram (Kernel) Matrix

Training Data: \( \langle x_1, y_1 \rangle, \ldots, \langle x_N, y_N \rangle \)

\[
K = \begin{pmatrix}
K(x_1, x_1) & \cdots & K(x_1, x_N) \\
\vdots & \ddots & \vdots \\
K(x_N, x_1) & \cdots & K(x_N, x_N)
\end{pmatrix}
\]

Properties:
- Positive Definite Matrix
- Symmetric
- Positive on diaginal
- \( N \) by \( N \)

Commonly Used Mercer Kernels

- Polynomial
  \[
  K(x_i, x_j) = \left( \langle x_i, x_j \rangle + c \right)^d
  \]
- Sigmoid
  \[
  K(x_i, x_j) = \tanh(\kappa \langle x_i, x_j \rangle + \theta)
  \]
- Gaussian
  \[
  K(x_i, x_j) = \exp\left( -\frac{1}{2\sigma^2} \| x_i - x_j \|^2 \right)
  \]
SV Regression: $\varepsilon$-Insensitive Loss

Goal: generalize SV pattern recognition to regression, preserving the following properties:

- formulate the algorithm for the linear case, and then use kernel trick
- sparse representation of the solution in terms of SVs

$\varepsilon$-Insensitive Loss:

$$|y - f(x)|_\varepsilon := \max\{0, |y - f(x)| - \varepsilon\}$$

Estimate a linear regression $f(x) = \langle w, x \rangle + b$ by minimizing

$$\frac{1}{2}\|w\|^2 + \frac{C}{m} \sum_{i=1}^{m} |y_i - f(x_i)|_\varepsilon.$$
Formulation as an Optimization Problem

Estimate a linear regression
\[ f(x) = \langle w, x \rangle + b \]
with precision \( \varepsilon \) by minimizing
\[
\begin{align*}
\text{minimize} & \quad \tau(w, \xi, \xi^*) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*) \\
\text{subject to} & \quad (\langle w, x_i \rangle + b) - y_i \leq \varepsilon + \xi_i \\
& \quad y_i - (\langle w, x_i \rangle + b) \leq \varepsilon + \xi_i^* \\
& \quad \xi_i, \xi_i^* \geq 0
\end{align*}
\]
for all \( i = 1, \ldots, m \).

Dual Problem, In Terms of Kernels

For \( C > 0, \varepsilon \geq 0 \) chosen a priori,
\[
\begin{align*}
\text{maximize} & \quad W(\alpha, \alpha^*) = -\varepsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} (\alpha_i^* - \alpha_i)y_i \\
& \quad - \frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(x_i, x_j)
\end{align*}
\]
subject to \( 0 \leq \alpha_i, \alpha_i^* \leq C, \ i = 1, \ldots, m \), and \( \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0 \).

The regression estimate takes the form
\[ f(x) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i)k(x_i, x) + b, \]
SVM Summary

• Good points
  – Picks a subset of the data that explains it all
  – Gives good models
• Bad points
  – Slow on large datasets (<20,000)
  – Difficult to pick good kernels
• Software
  – LIBSVM, SVMlight, etc

K Nearest Neighbor

– Main Assumption:
  • An effective distance metric exists.
– Hypothesis Space
  • Variable size (nonparametric): Can model any function

Classify according to Nearest Neighbor
Separates the input space
Nearest Neighbor Algorithm

- Given training data \((x_1, y_1), \ldots, (x_N, y_N)\)
- Define a distance metric between points in inputs space. Common measures are:
  - Euclidean (squared) \(D(x, x_i) = \sum_{j=1}^{d} (x_j - x_{i,j})^2\)
  - Weighted Euclidean \(w_j \geq 0\)
    \[D(x, x_i) = \sum_{j=1}^{d} w_j (x_j - x_{i,j})^2\]

K-Nearest Neighbor Model

- Given test point \(x\)
- Find the \(K\) nearest training inputs \(x_1, \ldots, x_N\)
  to \(x\) given the distance metric \(D(x, x_i)\)

- Denote these points as
  \((x_1, y_1), \ldots, (x_K, y_K)\)
K-Nearest Neighbor Model

• Regression:

\[ \hat{y} = \frac{1}{K} \sum_{k=1}^{K} y_k \]

• Classification:

\[ \hat{y} = \text{most common class in set} \ \{y_1, \ldots, y_K\} \]

K-Nearest Neighbor Model: Weighted by Distance

• Regression:

\[ \hat{y} = \frac{\sum_{k=1}^{K} D(x, x_k) y_k}{\sum_{k=1}^{K} D(x, x_k)} \]

• Classification:

\[ \hat{y} = \text{most common class in weighted set} \ \left\{ \frac{1}{D(x, x_1)} y_1, \ldots, \frac{1}{D(x, x_K)} y_K \right\} \]
K Nearest Neighbor Summary

• Good points
  – Fast learning
  – Can learning anything
  – Trivial to implement
• Bad points
  – Slow predictions for large datasets
  – Distance metric is critical
• Software

Decision Trees

– Main Assumption:
  • Data effectively modeled via decision splits on attributes.
– Hypothesis Space
  • Variable size (nonparametric): Can model any function
1-R (A Decision Tree Stump)

- Main Assumptions
  - Only one attribute is necessary.
  - Finite number of splits on the attribute.
- Hypothesis Space
  - Fixed size (parametric): Limited modeling potential

Decision Tree Summary

- Good points
  - Does feature selection
  - Models can be interpreted
  - Works on large data sets
- Bad points
  - Model performance often not best
- Learning Parameters
  - Depth of tree, choice of splitting function, pruning, etc...
- Software
Naïve Bayes

– Main Assumptions:
  • All attributes are equally important.
  • All attributes are statistically independent (given the class value)

– Hypothesis Space
  • Fixed size (parametric): Limited modeling potential

\[
\Pr[y|x] = \frac{\Pr[y]\Pr[x_1|y]\Pr[x_2|y] \cdots \Pr[x_n|y]}{\Pr[x]}
\]

Naïve Bayes Summary

• Good points
  – Easy to use
  – Works on large data sets
  – Gives probabilities

• Bad points
  – Model performance often not best
  – Probabilities often don’t reflect reality

• Software
  – WEKA:
    http://www.cs.waikato.ac.nz/~ml/weka/index.html
Neural Networks

- Main Assumption:
  - Many simple functional units, combined in parallel, produce effective models.

- Hypothesis Space
  - Variable size (nonparametric): Can model any function
Neural Networks Summary

- **Good points**
  - Can work on very large data sets
    - Gradient descent

- **Bad points**
  - Not easy to interpret
  - Can be difficult to choose a topology

- **Software**
  - **WEKA:**
  - **LUSH:** [http://lush.sourceforge.net/](http://lush.sourceforge.net/)

\[
s(x) = \frac{1}{1+e^{-x}}
\]

\[
\text{Nice property: } \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))
\]

We can derive gradient decent rules to train

- One sigmoid unit
- **Multilayer networks of sigmoids** → Backpropagation
Bagging

– Main Assumption:
  • Combining many unstable predictors to produce a ensemble (stable) predictor.
  • Unstable Predictor: small changes in training data produce large changes in the model.
    – e.g. Neural Nets, trees
    – Stable: SVM (sometimes), nearest Neighbor.

– Hypothesis Space
  • Variable size (nonparametric): Can model any function

Bagging (continued)

• Each predictor in ensemble is created by taking a bootstrap sample of the data.
• Bootstrap sample of N instances is obtained by drawing N example at random, with replacement.
• On average each bootstrap sample has 63% of instances
  – Encourages predictors to have uncorrelated errors.
Bagging Reborn: Random Forests

- Injecting the right kind of randomness makes accurate models
  - As good as SVMs and sometimes better
  - As good as boosting
- **Very little playing with learning parameters is needed to get very good models**
- Traditional tree algorithms spend a lot of time choosing how to split at a node
  - Random forest trees put very little effort into this

R.I. (Random Input) Forests

- For K trees:
  - Build each tree by:
    - Selecting, at random, at each node a small set of features (F) to split on. Common values of F are
      \[ F = 1 \]
      \[ F = \log(M) + 1 \]
    - For each node split on the best of this subset
    - Grow tree to full length

Regression: average over trees
Classification: vote
R.C. (Random Combination) Forests

• For K trees:
  – Build each tree by:
    • Create F random linear sums of L variables
      \[ A_f = \sum_{i=1}^{L} b_{ij} x_{ind(i)}, \ldots, A_F = \sum_{i=1}^{L} b_{ij} x_{ind(i)} \]
      \( b_{ij} = \text{uniform random} \ [-1,1] \)
    • At each node split on the best of these linear boundaries
    • Grow tree to full length

Regression: average over trees
Classification: vote

Random Forests Summary

• Good points
  – Fast learning
  – Can learning anything
  – Feature weighting
  – Very little parameter tuning
  – Gives a good estimate of model accuracy
    • Uses out of bag examples

• Bad points
  – Models are huge

• Software
  – http://www.stat.berkeley.edu/users/breiman/RandomForests/
Boosting

– Main Assumption:
  • Combining many weak predictors (e.g. tree stumps or 1-R predictors) to produce an ensemble predictor.

– Hypothesis Space
  • Variable size (nonparametric): Can model any function
    – However, some functions may not learnable

Boosting (Continued)

• Each predictor is created by using a biased sample of the training data
  – Instances (training examples) with high error are weighted higher than those with lower error

• Difficult instances get more attention
Boosting Summary

• Good points
  – Fast learning
  – Can learning anything
  – Feature weighting
  – Very little parameter tuning
• Bad points
  – Can overfit data
• Software

PAC (Probably Approximately Correct) Learning Theory

• Want to ensure that $1-\delta$ of the time, the hypothesis error is less than $\epsilon$
  \[
P_D^n \left[ \text{error} \left( f, h \right) > \epsilon \right] < \delta
\]

• Ex: want to obtain a 90% ($\epsilon = 1 - .9$) correct hypothesis 95% ($\delta = 0.05$) of the time.
VC Dimension

**Definition:** Let $S = \{x_1, ..., x_m\}$ be a set of $m$ examples. A hypothesis space $H$ can **trivially fit** $S$, if for every possible labeling of the examples in $S$, there exists an $h \in H$ that gives this labeling. If so, than $H$ is said to **shatter** $S$.

**Definition:** The Vapnik-Chervonenkis dimension (VC-dimension) of a hypothesis space $H$ is the size of the largest set of examples that can be trivially fit (shattered) by $H$.
Unsupervised Learning: K-Means Via Expectation Maximization

Given:
- Instances from $X$ generated by mixture of $k$ Gaussian distributions
- Unknown means $(\mu_1, \ldots, \mu_k)$ of the $k$ Gaussians
- Don’t know which instance $x_i$ was generated by which Gaussian

Determine:
- Maximum likelihood estimates of $(\mu_1, \ldots, \mu_k)$

Generating Data from Mixture of $k$ Gaussians

Each instance $x$ generated by
1. Choosing one of the $k$ Gaussians with uniform probability
2. Generating an instance at random according to that Gaussian
General EM Algorithm

Given:
- Observed data $X = \{x_1, \ldots, x_m\}$
- Unobserved data $Z = \{z_1, \ldots, z_m\}$
- Parameterized probability distribution $P(Y|h)$, where
  $- Y = \{y_1, \ldots, y_m\}$ is the full data $y_i = x_i \cup z_i$
  $- h$ are the parameters

Determine:
- $h$ that (locally) maximizes $E[\ln P(Y|h)]$

Converges to Local Maximum.

Bayesian Belief Network

Network represents a set of conditional independence assertions:
- Nodes are asserted to be conditionally independent of non-descendants, given their immediate predecessors.
- Directed acyclic graph

EM Algorithm can be used to infer unobserved variables.