Search Problems Definition
state space S, initial state $s_0 \in S$, actions $A(s)$ of state s, transition model/function $\text{Result}(s,a)$, goal test $G(s)$, action cost $c(s,a,s')$
State space graph: Node are states, Arcs represent transitions, goal test is a set of goal nodes
Search Tree: root node is start state, children correspond to successors, nodes show states, correspond to plans that achieve that states.
Node in Search Tree = Path in the state space

Local search: faster, memory efficient, but incomplete and suboptimal. General Idea: improves a single option until you can't make it better.
Hill Climbing: start wherever, REPEAT: move to the best neighbor. Problems: local maxima(stuck), plateaus(lost) Variants: Stochastic(randomly choose move) Random-restart(randomly restart)
Simulated Annealing: Gradually reduce temperature; higher temperature, more bad moves allowed, shake the system out of its local best(bad move accept rate:exp $\Delta E/T$)
Local Beam Search: greedily keep k (best) states. No optimal property. Running time $O(nkb \log kb)$ Variant: Stochastic (choosing successors randomly, avoid lack of diversity.)

MCTS Steps:
1. Selection, used for nodes we have seen before, pick according to UCB
2. Expansion, used when we reach the frontier, add one node per playout
3. Simulation, used beyond the search frontier; no UCB, just play randomly
4. Backpropagation, after reaching a terminal node, update value and visits for states expanded in selection and expansion

• Pros:	• Cons:
- Grows tree asymmetrically, balancing expansion and exploration	- Can't handle extreme tree depth
- Unaffected by branching factor	- Requires ease of simulation, massive computation resources
- Easy to adapt to new games	- Relies on random play being "weakly correlated"
- Heuristics not required, but can also be integrated	- Many variants, need expertise to tune
- Anytime algorithm: can finish on demand	- Theoretical properties not yet understood
- Trivially parallelizable	

SVC
• Apply Representer theorem, we can kernelize Soft-SVM as follows:
$$\min_{\alpha} \frac{1}{2}\alpha^T K \alpha + \frac{C}{n}\sum_{i=1}^{n} \max\left\{0, 1 - y_i \sum_{j=1}^{n} \alpha_j k(x_i, x_j)\right\}$$
- α_j is the weight of each reference point x_j to the prediction of x_i
- It is actually a Primal Form with kernel functions.

Hard-margin Support Vector Machine (SVM)
$$\min_{w,b} \frac{1}{2}\|w\|_2^2$$
$$\text{s.t. } y_i(w \cdot x_i + b) \geq 1, 1 \leq i \leq n$$

Soft-margin Support Vector Machine (SVM)
$$\min_{w,b,\xi} \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$$
$$\text{s.t. } y_i(w \cdot x_i + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0, 1 \leq i \leq n$$

• Define Hinge loss $\ell(f(x), y) = \max\{0, 1 - yf(x)\}$
- For the linear hypothesis: $\ell(f(x), y) = \max\{0, 1 - y(w \cdot x + b)\}$
• Theorem: Soft-SVM is equivalent to a Regularized Risk Minimization:
$$\min_{w,b} \frac{1}{2}\|w\|_2^2 + \frac{C}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(w \cdot x_i + b)\}$$

Stochastic Gradient Descent
• In each iteration t $(\leq T)$:
- Randomly sample a minibatch of $m \ll n$ points $\{(x_i, y_i)\}_{i=1}^{m}$
- Set $J^t(\theta^t) = \frac{1}{m}\sum_{i=1}^{m} \ell(\theta^t; x_i, y_i)$
- Compute gradient on minibatch: $\Delta^t = \nabla_\theta J^t(\theta^t)$
- Update parameters with learning rate η: $\theta^{t+1} = \theta^t - \eta\Delta^t$

Uninformed Search:

Criterion	Strategy	Frontier Implementation	Action Costs	Complete	Optimal	Time	Space	Comment
Depth-First	expand deepest node first	LIFO stack	$= c$	No	No	$O(b^m)$	$O(bm)$	m is tier num
Breadth-First	expand shallowest node first	FIFO queue	$= c$	Yes	Yes	$O(b^d)$	$O(b^d)$	
Iterative Deepening	DFS with increasing depth limit		$= c$	Yes	Yes	$O(b^d)$	$O(bd)$	
Uniform-Cost	expand lowest $g(n)$ [$g(n)$:root → n cost]	priority queue sorted by $g(n)$	$\geq \epsilon$	Yes	Yes	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$	Solution cost C^*, minimal arc cost ϵ

Heuristic: A function that estimates how close a state is to a goal, $h(n)$
Greedy Search: Strategy: expand a node that you think is closest to a goal state. Not Complete. Not optimal. Worst case: badly guided DFS.
A* search: Strategy: UCS + Greedy Search, sort by $f(n) = g(n) + h(n)$.
Tree search optimal & Admissible Heuristics: $0 \leq h(n) \leq h^*(n)$ estimate cost < real cost → Provide optimal property for tree search
Graph search optimal & Consistency of Heuristics: $h(n) - h(n') \leq c(n, a, n')$ estimate cost < real cost for each arc [Consistency → admissible] → Provide optimal property for graph search.
Creating Heuristics: Using the answer of a relaxed (constraint) problem; mixing heuristics

Constraint Satisfaction Problem(CSP) $P = (X, D, C)$, Variables: $X = \{X_1, \dots, X_n\}$, Domains: $D = \{D_1, \dots, D_n\}$, each domain $D_i = \{v_1, \dots, v_k\}$ for variable X_i; Constraints C, allowable combinations of values(binary CSP: only binary constraint); Assignment: $\{X_i = v_i, X_j = v_j\}$ (complete: every variable is assigned); Solution: a consistent and complete assignment
Solution: 1. Backtracking Search = DFS + variable-ordering + fail-on-violation(With Dynamic Ordering: (a)MCV Choose variable that has the fewest consistent values. (b)LCV Order values of selected X_i by decreasing number of consistent values of neighboring variables.)
Structured CSP problems: No loop constraint graph can be solved in linear time: $O(nd^2)$

Arc Consistency: Consistent arc $X_i \to X_j$: for every $x_i \in \text{Domains}_i$, exists $x_j \in \text{Domain}_j$ that is consistent; Enforce arc consistency: Reduce Domain_i
AC-3 Algorithm: if X_i lose a value, the arc points to X_i need to be rechecked. **Implementation:** Use a queue to store arcs to be checked. Time complexity: $O(c \cdot d \cdot d^2)$, each arc could be insert into queue for most d times, every time it takes d^2 time [Note: it isn't always effective]

Called when the domain of X_t is reduced.

```
function AC-3($X_t$, Domains) returns false if an inconsistency is found and true otherwise
    initialize queue with all arcs ($X_s$, $X_t$) for $X_s$ in Neighbours($X_t$)
    while queue is not empty do
        ($X_i$, $X_j$) ← RemoveFirst(queue)
        if EnforceArcConsistency(Domains, $X_i$, $X_j$) then      The domain of $X_i$ is reduced.
            if size of $Domains_i$ = 0 then return false
            for each $X_k$ in Neighbours($X_i$) do
                add ($X_k$, $X_i$) to queue
    return true
```

Constraint propagation:
$$\begin{tikzcd} X_k \arrow[r] & X_i \arrow[r] & X_j \end{tikzcd}$$

Standard Game Formulation
Initial state s_0. Players Player(s) who is playing s. Actions: Actions(s). Transition model: Result(s, a). Terminal test: Termnal_Test(s). Utility function: Utility(s, p).

Minimax Algorithm: a state-space tree search(DFS), choose the action leading to state with best achievable utility against an optimal adversary.
MAX nodes: Under our control
$$V(s) = \max_{a \in \text{Actions}(s)} V(\text{Result}(s, a))$$
MIN nodes: Under opponent's control
$$V(s) = \min_{a \in \text{Actions}(s)} V(\text{Result}(s, a))$$
Generalized Minimax: Each player maximizes its own component

Alpha-Beta Pruning

```
function MAX-VALUE($s$, $\alpha$, $\beta$) returns value
    if Terminal-Test($s$) then return Utility($s$)
    initialize $v = -\infty$
    for each successor $s'$ of state $s$:
        $v$ = max($v$, MIN-VALUE($s'$, $\alpha$, $\beta$))
        if $v \geq \beta$
            return $v$
        $\alpha$ = max($\alpha$, $v$)
    return $v$
```

MCTS(Monte Carlo Tree Search): use rollout
Naïve/Pure: do N complete random rollouts, choose the one with best percentage win
Better Rollout Policy: Exploitation(focus on more promising moves) & Exploration (focus on more uncertain nodes)
UCB I Formula(Upper confidence bound):
$$v_i + C \times \sqrt{\frac{\ln(N)}{n_i}}$$
tunable parameter
value estimate
total number of trials
num trials for arm i
Exploit: prefers higher payoff arm
Explore: prefers less played arm

Components of Learning: Input: $x \in \mathbb{R}^d$, Output: $y = \{0,1\}$, Target Function: $f: \mathcal{X} \to \mathcal{Y}$, Data: $(x_1, y_1), \dots, (x_n, y_n)$ [Supervised Learning], Hypothesis: $h: \mathcal{X} \to \mathcal{Y}$, Space: $\mathcal{H} = \{h\}$, Learning algorithm $\mathcal{A}$ to find the best $h \approx f$
Loss function: $l: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$, measure the difference

	Supervised Learning	Unsupervised Learning
Discrete Output	Classification 分类	Clustering 聚类
Continuous Output	Regression 回归	Embedding 嵌入

		Ground Truth	
		Actual Positive	Actual Negative
Predicted	Predicted Positive	TP True Positive	FP False Positive
	Predicted Negative	FN False Negative	TN True Negative

Precision$= \frac{\text{TP}}{\text{TP+FP}}$: How many selected items are relevant
Recall $= \frac{\text{TP}}{\text{TP+FN}}$: How many relevant items are selected

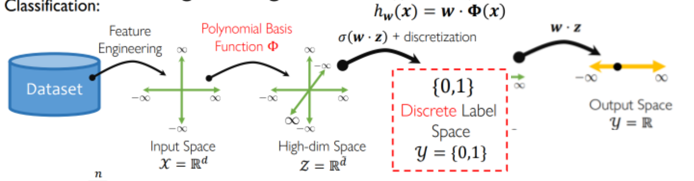
$$\text{TPR} = \frac{\text{TP}}{\text{TP + FN}}, \text{FPR} = \frac{\text{FP}}{\text{FP + TN}}$$
Sensitivity = Recall = TPR = $P(\hat{Y} = 1 | Y = 1)$
Specificity = 1 – FPR = $P(\hat{Y} = 0 | Y = 0)$

KNN: **When to use:** 1. Few attributes per instance (expensive computation); 2. Lots of training data (curse of dimensionality)
Advantages: 1. Agnostically learn complex target functions; 2. Do not lose information (store original data);3. Data number can be very large (big pro!); 4: Class number can be very large (biggest pro!)[All other ML algorithms may fail here!]
Disadvantages: 1. Slow at inference time (acceleration a must) 2. Fooled easily by irrelevant attributes (feature engineering crucial)

Z-score normalization:
For each feature dimension j, compute based on its samples: 1.
Mean $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$, **Variance** $\sigma_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2}$, 2:
normalize the feature into a new one $\hat{x}_{ij} \leftarrow \frac{x_{ij} - \mu_j}{\sigma_j}$

2-class Classification:

Logistic Regression



Hypothesis Space
$h_w(x) = w \cdot \Phi(x)$
Dataset
Feature Engineering
Polynomial Basis Function Φ
$\sigma(w \cdot z)$ + discretization
$w \cdot z$
Input Space
$\mathcal{X} = \mathbb{R}^d$
High-dim Space
$\mathcal{Z} = \mathbb{R}^d$
$\{0,1\}$
Discrete Label Space
$\mathcal{Y} = \{0,1\}$
Output Space
$\mathcal{Y} = \mathbb{R}$

$$\hat{\epsilon}(w) = -\sum_{i=1}^{n}\{y_i \log \sigma(h_w(x)) + (1 - y_i)\log[1 - \sigma(h_w(x))]\} + \lambda\Omega(w)$$

Linear Regression
Hypothesis Space
$h_w(x) = w \cdot \Phi(x)$
Gradient Descent
$\nabla_w \hat{\epsilon}(w) = 2Z^T(Zw - y) + 2\lambda w$
Loss Function
$$\boxed{\min_{w}} \sum_{i=1}^{n}(h_w(x_i) - y_i)^2 + \lambda\Omega(w)$$
Analytic Solution
$w = (Z^T Z + \lambda I)^{-1} Z^T y$
Well Done! A simple but complete model!

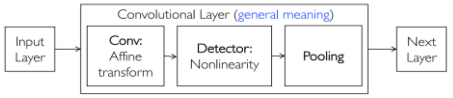
• Softmax function normalizes multiple outputs in a probability vector:
Softmax
$$p(y = i | x) = \frac{\exp(w_i^T x)}{\sum_{r=1}^{C} \exp(w_r^T x)}$$
Sum over all classes.

Hypothesis of decision tree: 1. Decision tree divide the feature space into axis-parallel rectangles 2. each rectangular region is labeled with a specific label
Entropy(Use in ID3 and C4.5): $H(\mathcal{D}) = -\sum_{k=1}^{k} \frac{|\mathcal{C}_k|}{|\mathcal{D}|} \log \frac{|\mathcal{C}_k|}{|\mathcal{D}|}$ k类对应数量 总数
Maximizing **Information Gain:** $H(\mathcal{D}_1 \cup \mathcal{D}_2) - \frac{|\mathcal{D}_1|}{|\mathcal{D}|} H(\mathcal{D}_1) - \frac{|\mathcal{D}_2|}{|\mathcal{D}|} H(\mathcal{D}_2)$

ID3-Algorithm
数据集 Class label 特征集

```
ID3(Examples, Target_attribute, Attributes)
• create a Root node for the tree; assign all Examples to Root;              Stop Criteria
• if all Examples are positive, return the single-node tree Root, with label=+;
• if all Examples are negative, return the single-node tree Root, with label=−;
• if Attributes is empty, return the single-node tree Root,
  with label = the most common value of Target_attribute in Examples;
• otherwise // Main loop:
      A ← the attribute from Attributes that best* classifies Examples;       O(dn) in each layer
      the decision attribute for Root ← A;
      for each possible value $v_i$ of A                     选出IG最大的
          add a new tree branch below Root, corresponding to the test $A = v_i$;
          let $Examples_{v_i}$ be the subset of Examples that have the value $v_i$ for A;
          if $Examples_{v_i}$ is empty                       样本没有, 测试可能有的
              below this new branch add a leaf node with label = the most common value
              of Target_attribute in Examples;
          else                                                depth min(d, log n)
              below this new branch add the subtree
              ID3($Examples_{v_i}$, Target_attribute, Attributes\{A});
• return Root;
```
* The best attribute is the one with the highest information gain.

CNN: 1. Local Assumption: local information is enough for recognition ➔ only connect local neurons **2.** Shift Invariance Assumption: if a feature is useful at spatial position ➔ share the weight of sliding windows, **Size** $\text{height}_{\text{new}} = \lfloor(\text{height} - \text{filter} + 2\cdot\text{pad})/\text{stride}\rfloor + 1$ $\text{weight}_{\text{new}} = \lfloor(\text{width} - \text{filter} + 2\cdot\text{pad})/\text{stride}\rfloor + 1$

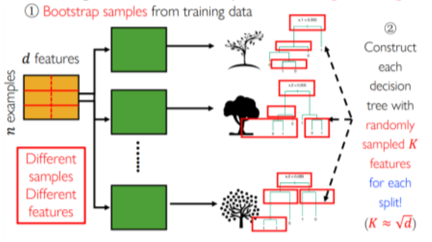


RNN: Local Dependency Assumption: The sequential information of all previous timestamps can be encoded into one hidden representation. **Temporal Stationarity Assumption:** If a feature is useful at time t_1, then it should also be useful for all time stamps t_2

Random Forest

- Bootstrap (自助法): randomly draw datasets with replacement from the training data, with the same sample size as the original training set

① Bootstrap samples from training data


d features
n examples
Different samples Different features
②
Construct each decision tree with randomly sampled K features for each split!
(K ≈ √d)

- **Probabilistic Reasoning = Modeling + Inference**
  - $X = \{x_1, \dots, x_D\}$ is a set of D random variables.
  - Query set R and condition set C are subsets of X.
- **Modeling**: How to specify a joint distribution $p(x_1, \dots, x_D)$ **compactly**?
- **Inference**: How to compute $p(R \mid C)$ **efficiently**?

- A **Bayesian network** is a **directed acyclic graph** (DAG) that specifies a joint distribution as a product of local conditional distributions, one for each node: $p(x_1, \dots, x_K) = \prod_{s=1}^{K} p(x_s|x_{\Gamma(s)})$ where $\Gamma(s)$ denotes the set of parents of x_s.

EM for GMM

- **E-step.** Evaluate all responsibilities using current parameters:

$$\gamma_i^j = \frac{\pi_j \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{c=1}^{k} \pi_c \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}$$

- **M-step.** Re-estimate the parameters using the responsibilities:

$$\pi_c^{\text{new}} = \frac{n_c}{n}$$

$$\boldsymbol{\mu}_c^{\text{new}} = \frac{1}{n_c}\sum_{i=1}^{n} \gamma_i^c \boldsymbol{x}_i, \quad \boldsymbol{\Sigma}_c^{\text{new}} = \frac{1}{n_c}\sum_{i=1}^{n} \gamma_i^c (\boldsymbol{x}_i - \boldsymbol{\mu}_c^{\text{new}})(\boldsymbol{x}_i - \boldsymbol{\mu}_c^{\text{new}})^T$$

Batch & Layer

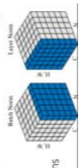
Recap: Batch Normalization (BN)

$$\bar{a}_i^l = \frac{g_i^l}{\sigma_i^l}(a_i^l - \mu_i^l) \qquad \mu_i^l = E_{x\sim P(x)}[a_i^l] \qquad \sigma_i^l = \sqrt{E_{x\sim P(x)}\left[(a_i^l - \mu_i^l)^2\right]}$$

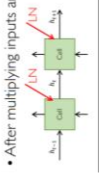
Layer normalization (LN) in RNNs

$$h^t = f\left[\frac{g}{\sigma^t} \odot (a^t - \mu^t) + b\right] \quad \mu^t = \frac{1}{C}\sum_{i=1}^{C} a_i^t \quad \sigma^t = \sqrt{\frac{1}{C}\sum_{i=1}^{C}(a_i^t - \mu^t)^2}$$

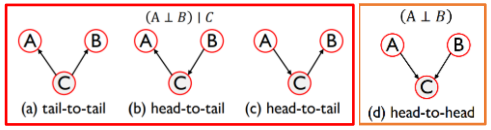
- $\boldsymbol{g}$ and $\boldsymbol{b}$ are shared over all time-steps, the statistics are computed across each feature (channel) and are independent of the batch size
- After multiplying inputs and states by weights but before nonlinearity



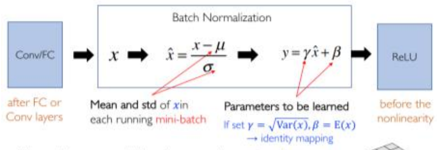
C: channel dim
N: example dim
(H, W): spatial dims



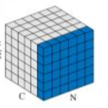
Conditional Independence **Marginal Independence**



判断条件独立：**1.** 找到 C 及 C 的所有祖先，断开**不是 h-to-h 的边。2.** 对其他的点，断开**是 h-to-h 的边**。
结果：如果不联通，则条件独立。


Batch Normalization
Conv/FC
x
$\hat{x} = \frac{x-\mu}{\sigma}$
$y = \gamma\hat{x} + \beta$
ReLU
after FC or Conv layers
Mean and std of x in each running mini-batch
Parameters to be learned
If set $\gamma = \sqrt{\text{Var}(x)}, \beta = \text{E}(x)$ → identity mapping
before the nonlinearity

- Normalize over mini-batch examples and nodes
- For a mini-batch of size n and feature maps of size w*h, the n*w*h elements are normalized together, and **each feature map** has a pair of **γ** and **β**
- At inference, use exponential moving average (EMA)



- How to infer a node with variable x_i from the remaining variables $x_{i\neq i}$ in directed models ? $p(x_i \mid \boldsymbol{x}_{\{j\neq i\}})$
- Find $S_1 \in \boldsymbol{x}_{\{j\neq i\}}$ which contents **Markov blanket**
- It means that S_1 contains all information one needs to infer x_i.

$$p(x_i \mid \boldsymbol{x}_{\{j\neq i\}}) = p(x_i \mid S_1 \sqcup (\boldsymbol{x}_{\{j\neq i\}} \setminus S_1)) = p(x_i \mid S_1)$$

- S_1 is called the Markov blanket of x_i. $x_i \perp (\boldsymbol{x}_{\{j\neq i\}} \setminus S_1) \mid S_1$

- A Markov boundary is the minimal Markov blanket. In a Bayesian network, it includes parents, children and the other parents of all of its children (co-parents). $p(x_i \mid \boldsymbol{x}_{\{j\neq i\}}) = p(x_i \mid x_{pa} \cup x_{ch} \cup x_{co\cdot pa})$

- **Marginal inference** · What is the probability of a given variable in our model after we sum everything else out?

$$p(y=1) = \sum_{x_1}\sum_{x_2}\cdots\sum_{x_n} p(y=1, x_1, x_2, \dots, x_n)$$

- Algorithm (Variable Elimination, VE)
- For each variable X_i (ordered according to O):
  1. Multiply all factors ϕ_i containing X_i ; 乘法
  2. Marginalize out X_i to obtain a new factor τ ; 加法
  3. Replace the factors ϕ_i with τ. 换掉
- Consider a general distribution $P(X, Y, E)$ over sets of:
  - Query variables Y; - Observed evidence variables E;
  - Unobserved variables X.

$$P(Y \mid E = e) = \frac{P(Y, E = e)}{P(E = e)}$$ Apply VE Algorithm!

- We can select the elimination ordering $X \to Y \to E$ to obtain $P(Y, E = e)$ and $P(E = e)$ in a single turn of VE algorithm.

Learn: Give data $\mathcal{D}$, find "best" parameter θ
Inference: Give data x and parameter θ, find latent variables z's distribution.

Parametric model $\{p(z;\theta) \mid \theta \in \Theta\}$
A sample dataset $\mathcal{D} = (z_1, \dots, z_N)$
Likelihood of $\hat{\theta} \in \Theta$ for sample $\mathcal{D}$ is:

$$p(\mathcal{D}; \hat{\theta}) = \prod_{n=1}^{N} p(z_n; \hat{\theta})$$

Log likelihood due to numerical reason:

$$\log p(\mathcal{D}; \hat{\theta}) = \sum_{n=1}^{N} \log p(z_n; \hat{\theta})$$

MLE(Maximum Likelihood estimator):

$$\hat{\theta} \in \underset{\theta\in\Theta}{\operatorname{argmax}} \log p(\mathcal{D}; \theta)$$

$$= \underset{\theta\in\Theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log p(z_n; \theta)$$

Bayes Decision Rule: to minimize error

$$h(\boldsymbol{x}) = \underset{y\in\mathcal{Y}}{\operatorname{argmax}}[p(Y = y|X = \boldsymbol{x})]$$

Discriminative models: Use conditional distribution(?)
MAP(Maximum A Posteriori Estimation)

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \log p(\theta|D) = \max_{\theta}\{\log p(D|\theta) + \log p(\theta)\}$$

Generative Models:
1. $p(X = \boldsymbol{x}|Y = y)$: 给定类，输入的分布
2. $p(Y = y)$: 类自身的分布
Then $p(Y = y|X = \boldsymbol{x}) \propto p(X = \boldsymbol{x}|Y = y)p(Y = y)$
假设 1 是伯努利分布➔Naïve Bayes
假设 1 是高斯分布➔Gaussian Discriminant Analysis
均假设 2 是伯努利分布

EM Algorithm:
z : latent variables, x :observed variables, θ: parametric model, $p(x, z|\theta)$
Key Idea: What if we know z's distribution $q(z)$?
ELBO: $L(q, \theta) = \sum_z q(z) \log\left(\frac{p(x,z|\theta)}{q(z)}\right)$
KL-Div: $\text{KL}[q(z)||p(z)] = \sum_z q(z) \log\left(\frac{q(z)}{p(z)}\right)$
Properties: $\text{KL}(p||q) \geq 0, \text{KL}(p||p) = 0$
推断隐变量 $\log p(\boldsymbol{x}|\boldsymbol{\theta}) = L(\boldsymbol{q}, \boldsymbol{\theta}) + KL[\boldsymbol{q}(\boldsymbol{z})||\boldsymbol{p}(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta})]$
E-Step: Maximizing $L(q, \theta)$ Over q for Fixed $\theta = \theta^{old}$, best q is $q^*(z) = p(z|x, \theta^{old}) \propto p(x|z, \theta^{old})q(z)$
学习参数 **M-Step:** Maximizing $L(q, \theta)$ Over θ for Fixed q, equivalent to **maximize the expected complete data likelihood**: $\max_{\theta} \sum_z q(z) \log p(x, z|\theta)$ [Using SGD]
MAP: M-Step $\theta^{new} = \underset{\theta}{\operatorname{argmax}}[L(q^*, \theta) + \log p(\theta)]$

GMM: k clusters, 每个都是高斯分布,数据没有标签
Dataset generation: 1. random choose a cluster z, 2. random choose a point x from cluster z
EM approach: E-step evaluate the responsibilities using current parameters for each point γ_i^j M-step, use MLE to maximize total "Expectation" Match.

Monte Carlo Estimator : draw i.i.d. sample $\mathcal{D} = \{x_1, \dots, x_n\}$ from distribution p, then use $\frac{1}{n}\sum_{i=1}^{n} f(x_i)$ to estimate $\mathbb{E}_p(f(x))$ [Unbiased, Variance $\frac{1}{n}\text{Var}_p(f(x))$]
EM with Monte Carlo:
M-step: $\theta^{new} = \cdots = \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{z\sim q} \log p(x, z|\theta) \approx \underset{\theta}{\operatorname{argmax}} \frac{1}{T}\sum_{i=1}^{T} \log p(x, z_i|\theta)$, z is sampling from $q^* = p(z|x, \theta^{old})$
Concrete sampling Method: 1. Use the reverse of CDF **2.** Rejection Sampling: for $kq(x) > p(x)$, draw $x \sim q, u \sim U[0, kq(x)]$; reject x if $u > p(x)$
[Problem: high rejection rate with high dimension]
Markov Chain Monte Carlo: Sampling from $p(x = j) = \pi_j \propto b_j$,where b_j is computable, while $\sum b_j = B$ is unknown.
Core idea: Construct a Markov Chain with stationary distribution π_j.
Method(Metropolis-Hastings): Random transition matrix(proposal distribution) $Q: Q_{ij} = \mathcal{N}(\boldsymbol{x}_j|\boldsymbol{x}_i, \boldsymbol{\epsilon}^2)$
1. start state $\boldsymbol{x}_0$; 2. sample $\boldsymbol{x}_*$ from Q; 3. $u \sim U[0,1]$, if $u > a_{ij} = \min\left(1, \frac{\pi_j Q_{ji}}{\pi_i Q_{ij}}\right)$, reject $\boldsymbol{x}_*$; else, accept $\boldsymbol{x}_*$.
Detailed Balance cond.: $\boldsymbol{\pi}$ is stationary distribution $\Leftrightarrow \pi_i P_{ij} = \pi_j P_{ji}$.

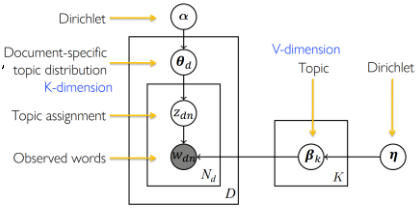
- Monte Carlo with Markov Chain: $\boxed{\frac{1}{T-T_0}\sum_{t=T_0}^{T-1} f(\boldsymbol{x}_t) \to \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{\pi}} f(\boldsymbol{x})}$

Gibbs Sampling: No rejection, with conditional distribution, a special case of Metro-Hast Algo with accept rate 1 (Proof TBD)
1. $x_1^{(t+1)} \sim p\left(x_1|x_2^{(t)}, x_3^{(t)}, \dots, x_n^{(t)}\right)$
2. $x_2^{(t+1)} \sim p\left(x_2|x_1^{(t+1)}, x_3^{(t)}, \dots, x_n^{(t)}\right)$
3. …
4. $x_j^{(t+1)} \sim p\left(x_j|x_1^{(t+1)}, \dots, x_{j-1}^{(t+1)}, x_{j+1}^{(t)}, \dots, x_n^{(t)}\right)$
5. …
6. $x_n^{(t+1)} \sim p\left(x_n|x_1^{(t+1)}, \dots, x_{n-1}^{(t+1)}\right)$

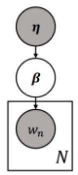
Gaussian Discriminant Analysis的参数[数据有标签]

$$\phi = \frac{\sum_{i=1}^{n} \mathbf{1}\{y_i=+1\}}{n}, \boldsymbol{\mu}_+ = \frac{\sum_{i=1}^{n} \mathbf{1}\{y_i=+1\}\boldsymbol{x}_i}{\sum_{i=1}^{n} \mathbf{1}\{y_i=+1\}}, \boldsymbol{\mu}_- = \frac{\sum_{i=1}^{n} \mathbf{1}\{y_i=-1\}\boldsymbol{x}_i}{\sum_{i=1}^{n} \mathbf{1}\{y_i=-1\}}$$

$$\boldsymbol{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu}_{y_i})(\boldsymbol{x}_i - \boldsymbol{\mu}_{y_i})^T$$



Dirichlet Distribution: $p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})}\prod_{k=1}^{K} \theta_k^{\alpha_k - 1}$,
$\boldsymbol{\theta}$ must satisfy $\sum_{i=1}^{K} \theta_i = 1$
Explanation: α_k is the importance of class k



Bayesian Estimation: The expectation of parameter using posterior as distribution

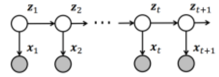
$$\hat{\theta} = \mathbb{E}_{\theta\sim p(\theta|D)}[\theta] = \int \theta \cdot p(\theta|D)\mathrm{d}\theta$$

Dirichlet-Multinomial Conjugate
$\boldsymbol{\beta} \sim \text{Dir}(\boldsymbol{\beta}|\boldsymbol{\eta}), w \sim \text{Mult}(1, \boldsymbol{\beta}), \boldsymbol{n}$ is count vector
The posterior Distribution is also Dirichlet! ➔ Simplify Below!

$$p(\boldsymbol{\beta}|\boldsymbol{\eta}, \boldsymbol{\mathcal{W}}) = \text{Dir}(\boldsymbol{\beta}|\boldsymbol{\eta} + \boldsymbol{n})$$

MAP: $\widehat{\boldsymbol{\beta}}_{\text{MAP}} = \max_{\boldsymbol{\beta}} \text{Dir}(\boldsymbol{\beta}, \boldsymbol{\eta} + \boldsymbol{n}), \left(\widehat{\boldsymbol{\beta}}_{\text{MAP}}\right)_i = \frac{\eta_i + n_i - 1}{\sum_{j=1}^{V} \eta_j + N - V}$

Bayes Est.: $\widehat{\boldsymbol{\beta}}_{\text{Bayes}} = \int \boldsymbol{\beta}\text{Dir}(\boldsymbol{\beta}|\boldsymbol{\eta} + \boldsymbol{n})\mathrm{d}\boldsymbol{\beta}, \left(\widehat{\boldsymbol{\beta}}_{\text{Bayes}}\right)_i = \frac{\eta_i + n_i}{\sum_{j=1}^{V} \eta_j + N}$



Hidden Markov Model
Hidden Variable z_t: Markov Chain, Transition $P(z_t|z_{t-1})$
Observation x_t: only depend on z_t, Emission $P(x_t|z_t)$
Inference for HMM: know x, guess z
Sequence error: $L(\boldsymbol{Z}, \widehat{\boldsymbol{Z}}) = \mathbf{1}(\widehat{\boldsymbol{Z}} \neq \boldsymbol{Z}), \hat{Z} = \underset{Z}{\operatorname{argmax}} p(Z|X) \propto \underset{Z}{\operatorname{argmax}} p(Z, X)$ [use joint distribution]
Viterbi Algorithm: negative log-probability, change to a shortest path model~~(TBD)~~
State error: $L(\boldsymbol{Z}, \widehat{\boldsymbol{Z}}) = \sum_t \mathbf{1}(\hat{z}_t \neq z_t), \hat{z}_t = \max_{z_t}(p(z_t|X))$[use marginal distribution]
Forward-Backward Algorithm, $p(z_t|X) \propto P(z_t|x_{1:t})P(x_{t+1:T}|z_t) = \alpha_t(z_t)\beta_t(z_t)$
Forward part: $\alpha_t(\boldsymbol{z_t}) = P(\boldsymbol{z_t}|\boldsymbol{x_{1:t}})$
1. Push by transition $P(z_t|x_{1:t-1}) = \sum_{z_{t-1}} P(z_t|z_{t-1})\alpha(z_{t-1})$
2. Reweighted by emission $\alpha_t(z_t) = P(z_t|x_{1:t}) \propto P(z_t, x_t|x_{1:t-1}) = P(x_t|z_t)P(z_t|x_{1:t-1})$
Backward part: $\beta_t(\boldsymbol{z_t}) = P(\boldsymbol{x_{t+1:T}}|\boldsymbol{z_t})$

$$\beta_t(z_t) = \sum_{z_{t+1}} \beta_{t+1}(z_{t+1})P(x_{t+1}|z_{t+1})P(z_{t+1}|z_t)$$

传播 转移

viterbi:

$$\widehat{\boldsymbol{Z}} = \underset{\boldsymbol{Z}}{\operatorname{arg\,min}}\left[\phi_1(\boldsymbol{z}_1) + \sum_{t=2}^{T}\left(\phi_t(\boldsymbol{z}_t) + \psi_t(\boldsymbol{z}_t, \boldsymbol{z}_{t-1})\right)\right]$$

Node
$$\phi_t(\boldsymbol{z}_t) = \begin{cases} -\log P(\boldsymbol{x}_1|\boldsymbol{z}_1) - \log P(\boldsymbol{z}_1) & t = 1 \\ -\log P(\boldsymbol{x}_t|\boldsymbol{z}_t) & t \geq 2 \end{cases}$$

edge
$$\psi_t(\boldsymbol{z}_t, \boldsymbol{z}_{t-1}) = -\log P(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$$