# Final Exam Review

## CM146 Winter 2018

March 17, 2018

# Information

About the Final Exam

- Time: March 22, 2018, Thursday, 11:30am-2:30pm
- Location: Franz Hall: 1178
- Closed-book, closed-notes, no cheat-sheet, no calculator
- The final exam will cover both materials **before** and **after** midterm.

Other Misc

- My OH: 12:00 pm - 2:00 pm, March 19, Monday
- ENG 6, public area in front of 386

# Outline

# Computational Learning Theory

- Set up:
  - Instance Space: $X$, the set of examples
  - Concept Space: $C$, the set of possible target functions: $f \in \mathcal{C}$ is the hidden target function
  - Hypothesis Space: $H$, the set of possible hypotheses. This is the set that the learning algorithm explores
  - What we want: A hypothesis $h \in H$ such that $h(x) = f(x)$
- Given a distribution $D$ over examples, the **error of a hypothesis** $h$ with respect to a target concept $f$ is

$$err_D(h) = Pr_{x \sim D}[h(x) \neq h(f)]$$

- For a target concept $f$, the **empirical error** of a hypothesis $h$ is defined for a training set $S$ as the fraction of examples $x$ in $S$ for which the functions f and h disagree, denoted by $err_S(h)$.
- **Overfitting**: When the empirical error on the training set $err_S(h)$ is substantially lower than $err_D(h)$.

# Probably Approximately Correct (PAC) Learning

- Theorem: Suppose we are learning a conjunctive concept with n dimensional Boolean features using m training examples. If

$$m > \frac{n}{\epsilon}\left(\log(n) + \log\left(\frac{1}{\delta}\right)\right)$$

- The concept class $C$ is **PAC learnable** by $L$ using $H$ if for $f \in C$, for all distribution $D$ over $X$, and fixed $\epsilon > 0, \delta < 1$, given m examples sampled i.i.d. according to $D$, the algorithm $L$ produces, with probability at least $(1 - \delta)$, a hypothesis $h \in H$ that has error at most $\epsilon$, where $m$ is polynomial in $1/\epsilon$, $1/\delta$, $n$ and size $H$.
- The concept class $C$ is **efficiently learnable** if $L$ can produce the hypothesis in time that is polynomial in $1/\epsilon$,$1/\delta$ and size($H$).
- A bound on how much the true error will deviate from the training error. If we have more than m examples, then with high probability $1 - \delta$,

$$err_D(h) - err_S(h) \leq \sqrt{\frac{\ln|H| + \ln(1/\delta)}{2m}}$$

# Shattering and VC dimension

- Definition: A set $S$ of examples is **shattered** by a set of functions $H$ if for every partition of the examples in $S$ into positive and negative examples there is a function in $H$ that gives exactly these labels to the examples.
- The **VC dimension** of hypothesis space H over instance space $X$ is the size of the largest finite subset of $X$ that is shattered by $H$.
- When there are infinite number of hypotheses in $H$, VC dimension is used to represent the expressiveness of a hypothesis space.

# Kernel Perceptron

- Perceptron:

$$\begin{aligned} &\text{if } y_i(\mathbf{w}^T\mathbf{x}_i) \leq 0 \\ &\quad \mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i \\ \Rightarrow &\mathbf{w}^* = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \end{aligned}$$

Kernel Perceptron:

$$\begin{aligned} &\text{if } y_i(\mathbf{w}^T\Phi(\mathbf{x}_i)) \leq 0 \\ &\quad \mathbf{w} \leftarrow \mathbf{w} + y_i\Phi(\mathbf{x}_i) \\ \Rightarrow &\mathbf{w}^* = \sum_{i=1}^{m} \alpha_i y_i \Phi(\mathbf{x}_i) \end{aligned} \tag{1}$$

- In prediction:

$$\text{perceptron prediction } sgn(\mathbf{w}^T\mathbf{x}^{test}) = sgn(\sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x}^{test})$$

$$\text{kernel perceptron prediction } sgn(\mathbf{w}^T\Phi(\mathbf{x}^{test})) = sgn(\sum_i \alpha_i y_i \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}^{test})) \tag{2}$$

where $\alpha_i$ is the number of mistakes made on $x_i$.

# Kernel

- **Kernel Trick**: Save time/space by computing the value of $K(\mathbf{x}, \mathbf{z})$ by performing operations in the original space (without a feature transformation!)

$$K(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z})$$

  So prediction with this high dimensional lifting map is

$$sgn(\mathbf{w}^T \Phi(\mathbf{x}^{test})) = sgn(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}^{test}))$$

- Positive semi-definite
- Symmetric
- Find the corresponding feature map of a kernel
- Prove the kernel is valid (How to?)

# Support Vector Machine

- Intuition: We want $\max_w \gamma$, where margin

$$\gamma = \min_{\mathbf{x}_i, y_i} \frac{y_i(\mathbf{w}^T \mathbf{x}_i + b)}{||\mathbf{w}||}$$

- Hard SVM:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{s.t.} \quad & \forall i, y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{aligned} \tag{3}$$

- Soft SVM:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \\ \text{s.t.} \quad & \forall i, y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{4}$$

or equivalently,

$$\min_{\mathbf{w}, b} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) \tag{5}$$

# Support Vector Machine

- Hinge Loss: $L_{hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$
  $L_{perceptron}(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\mathbf{w}^T\mathbf{x})$
- Large $C$: the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
  Small $C$: will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.
  $C \to 0$: you should get misclassified examples, often even if your training data is linearly separable.
  $C \to \infty$: turns to be Hard SVM.
- Solving the SVM optimization problem: Gradient descent (still very slow) $\to$ Stochastic gradient descent

$$J^t(w) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$
$$\nabla J^t = \begin{cases} w & \text{if } \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)) = 0 \\ w - Cy_i\mathbf{x}_i & \text{otherwise} \end{cases} \tag{6}$$

# Kernel Support Vector Machine

- Dual (Kernel) SVM

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & \forall i, 0 \leq \alpha_i \leq C \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0 \end{aligned} \tag{7}$$

  where $Q_{ij} = y_i y_j \phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$ and $\mathbf{e} = [1, \cdots, 1]^T$

- Dual Representation: In the optimum, the solutions of the primal and dual problem have the following relation

$$w^* = \sum_i \alpha_i^* y_i \Phi(\mathbf{x}_i)$$

- Decision function:

$$sgn(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}^{test}) + b)$$

- $\alpha_i = 0 \Rightarrow$the training sample doesnt affect the prediction
  $\alpha_i > 0 \Rightarrow$ support vectors: only SVs determine the weight!

# Boosting Algorithm - Adaboost

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution $D_t$.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t} \left[ h_t(x_i) \neq y_i \right].$$

- Choose $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$.
- Update:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).

Output the final hypothesis:

$$H(x) = \operatorname{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right).$$

# Boosting

- Here is an example
- Intuition - finding many rough rules of thumb can be a lot easier than finding a single, highly accurate classifier.
- Adaboost can fail if -
  - The weak classifiers are too complex and overfit.
  - The weak classifiers are too weak, essentially underfitting.

# Clustering

## Goal

Find an underlying distribution or organization in the data.

Setup: Given a dataset $D = \{x_n\}_{n=1}^{N}$ and $k$ we want to output

- $\{\mu_k\}_{k=1}^{K}$ prototypes or centroids
- $A(x_n) \in \{1, 2, ..., K\}$ : the cluster membership, i.e. cluster assigned to $x_n$

Distance measures (recap):

L2 Norm: $||x_1 - x_2||_2 = \sqrt{\sum_{i=1}^{n}(x_{1,i} - x_{2,i})^2}$

L1 Norm $||x_1 - x_2||_1 = \sqrt{\sum_{i=1}^{n}|x_{1,i} - x_{2,i}|}$

## K-means algorithm a.k.a Llyod's algorithm

- Step 0: randomly assign the cluster centers $\{\mu_k\}$
- Step 1: Minimize $J$ over $\{r_{nk}\}$ -- Assign every point to the closest cluster center

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|_2^2 \\ 0 & \text{otherwise} \end{cases}$$

- Step 2: Minimize $J$ over $\{\mu_k\}$ -- update the cluster centers

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\boldsymbol{x}_n}{\sum_n r_{nk}}$$

- Loop until it converges

# GMM

- Let $\theta$ represent all parameters $\{w_k, \mu_k, \Sigma_k\}$
- Step 0: Initialize $\theta$ with some values (random or otherwise)
- Step 1: Compute $\gamma_{nk} = p(z_n = k|\mathbf{x}_n)$ using the current $\theta$

$$p(z_n = k|\boldsymbol{x}_n) = \frac{p(\boldsymbol{x}_n|z_n = k)p(z_n = k)}{p(\boldsymbol{x}_n)} = \frac{p(\boldsymbol{x}_n|z_n = k)p(z_n = k)}{\sum_{k'=1}^{K} p(\boldsymbol{x}_n|z_n = k')p(z_n = k')}$$

$N(x|\mu_k, \Sigma_k)$ $\omega_k$

- Step 2: Update $\theta$ using the just computed $\gamma_{nk}$

$$\omega_k = \frac{\sum_n \gamma_{nk}}{\sum_k \sum_n \gamma_{nk}}, \quad \boldsymbol{\mu}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} \boldsymbol{x}_n$$

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}$$

- Loop until converge

# Bayesian Learning

## Goal

To find the best hypothesis from some space $H$ of hypotheses, using the observed data $D$.

What does best mean?

- Bayesian learning uses $P(h|D)$, the conditional probability of a hypothesis given the data, to define best.
- Calculate posterior $P(h|D)$ using Bayes theorem.

Maximum a Posteriori hypothesis, $h_{MAP}$ (recap):

- $h_{MAP} = argmax_{h \in H} P(h|D) = argmax_{h \in H} P(D|h)P(h)$.
- If we assume that the prior is uniform, we get Maximum Likelihood hypothesis, $h_{ML} = argmax_{h \in H} P(D|h)$.

# Expectation Maximization

## Goal

To compute MLE in unsupervised settings.

Setup: We only have observation of $\widetilde{P}(X)$

- In generative model, we have $P(X, Z|\theta)$
- We know $P(X|\theta) = \sum_Z P(X, Z|\theta)$
- Therefore, MLE is $argmax_\theta P(X|\theta) = argmax_\theta \sum_Z P(X, Z|\theta)$

EM algorithm (recap):

- Solves $argmax_\theta \sum_Z P(X, Z|\theta)$ by iteratively updating $\theta$
- In general, known to converge to a local maximum of the maximum likelihood function

# EM algorithm

## Given

A statistical model which generates a set $X$ of observed data, a set of unobserved latent data or missing values $Z$, and a vector of unknown parameters $\theta$, along with a likelihood function $L(\theta; X, Z) = P(X, Z|\theta)$.

- First, initialize the parameters $\theta$ to some random values.
- Compute the probability of each possible value of $Z$, given $\theta$.
- Then, use the just-computed values of $Z$ to compute a better estimate for the parameters $\theta$.
- Iterate steps 2 and 3 until convergence.

## The General EM Algorithm

Given a joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ over observed variables $\mathbf{X}$ and latent variables $\mathbf{Z}$, governed by parameters $\boldsymbol{\theta}$, the goal is to maximize the likelihood function $p(\mathbf{X}|\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$.

1. Choose an initial setting for the parameters $\boldsymbol{\theta}^{\text{old}}$.
2. **E step** Evaluate $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}})$.
3. **M step** Evaluate $\boldsymbol{\theta}^{\text{new}}$ given by

$$\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) \tag{9.32}$$

where

$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}). \tag{9.33}$$

4. Check for convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied, then let

$$\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}} \tag{9.34}$$

and return to step 2.

** Not required in the exam, just helping you understand the intuition

# Naive Bayes

- Logic: Bayes Learning
  $\rightarrow$ MAP estimation (of $h$): find the best hypothesis $h$
  $\rightarrow$ MAP prediction: predict probabilities of outcomes
  $\rightarrow$ Naive Bayes: assume independency
- MAP prediction: predict $y$ for the input $x$ using

$$\begin{aligned} \arg\max_y P(Y=y|X=x) &= \arg\max_y \frac{P(X=x|Y=y)P(Y=y)}{P(X=x)} \\ &= \arg\max_y P(X=x|Y=y)P(Y=y) \end{aligned} \tag{8}$$

- Naive Bayes: assume features are independent. Predict by maximizing joint distribution.

$$\begin{aligned} h_{NB}(\boldsymbol{x}) &= \operatorname*{argmax}_y P(y)P(x_1, x_2, \cdots, x_d|y) \\ &= \operatorname*{argmax}_y P(y)\prod_j P(x_j|y) \end{aligned}$$

# Naive Bayes

- Bayes classification
  $P(y|x) \propto P(x|y)P(y) = P(x_1, \cdots, x_n|y)P(y)$
  Difficulty: learning the joint probability $P(x_1, \cdots, x_n|y)P(y)$ is infeasible!
- Naive Bayes classification
  Assume all input features are class conditionally independent!
  $P(x_1, x_2, \cdots, x_n|y) = P(x_1|x_2, \cdots, x_n, y)P(x_2, \cdots, x_n|y)$
  $= P(x_1|y)P(x_2, \cdots, x_n|y)$
  $= P(x_1|y)P(x_2|y)(x_n|y)$
- Apply the MAP classification rule

# Naive Bayes

- Learning phase
  - Given a training set $S$ with each sample of the form $(\mathbf{X}, Y)$[1], each feature point will have $M$ dimension denoted by $X_j, j \in \{1, ..., M\}$.
  - The possible value of $Y$ is the set $\{y_1, \cdots, y_L\}$
  - For each feature dimension $x_j$, the possible value is the set $\{x_{j1}, \cdots, x_{jK}\}$.[2]
  - Learn the prior: $p(Y = y_i), i = 1, \cdots, L$
    Learn $1 - L$ parameters.
  - Learn the likelihood:
    $p(X_j = x_{jk}|Y = y_i), k = 1, \cdots, K; i = 1, \cdots, L; j = 1, \cdots, M$
    Learn $(1 - K) \times M \times L$ parameters.
- Testing phase
  Given a test point $\mathbf{X} = [X_1, X_2, \cdots, X_m]$
  $y^* = \arg\max_y p(Y = y) \prod_j p(X_j|Y = y)$

[1] Note that we use a upper case letter to distinguish with the notation of its value

[2] Note $K$ might be different for each feature $j$.

# Example: MAP Prediction

## Example: Tennis again

Prior

| Play tennis | P(Play tennis) |
|---|---|
| Yes | 0.3 |
| No | 0.7 |

Likelihood

| Temperature | Wind | P(T, W \| Tennis = Yes) |
|---|---|---|
| Hot | Strong | 0.15 |
| Hot | Weak | 0.4 |
| Cold | Strong | 0.1 |
| Cold | Weak | 0.35 |

| Temperature | Wind | P(T, W \| Tennis = No) |
|---|---|---|
| Hot | Strong | 0.4 |
| Hot | Weak | 0.1 |
| Cold | Strong | 0.3 |
| Cold | Weak | 0.2 |

**Input:**
Temperature = Hot (H)
Wind = Weak (W)

Should I play tennis?

$\text{argmax}_y$ P(H, W | play?) P (play?)

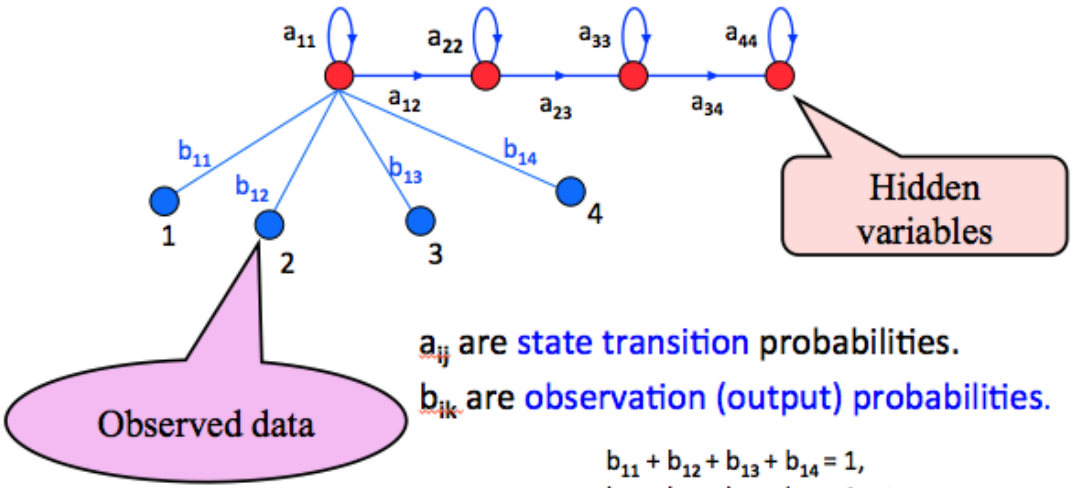
P(H, W | Yes) P(Yes) = 0.4 x 0.3
= 0.12

P(H, W | No) P(No) = 0.1 x 0.7
= 0.07

MAP prediction = Yes

# Hidden Markov Models

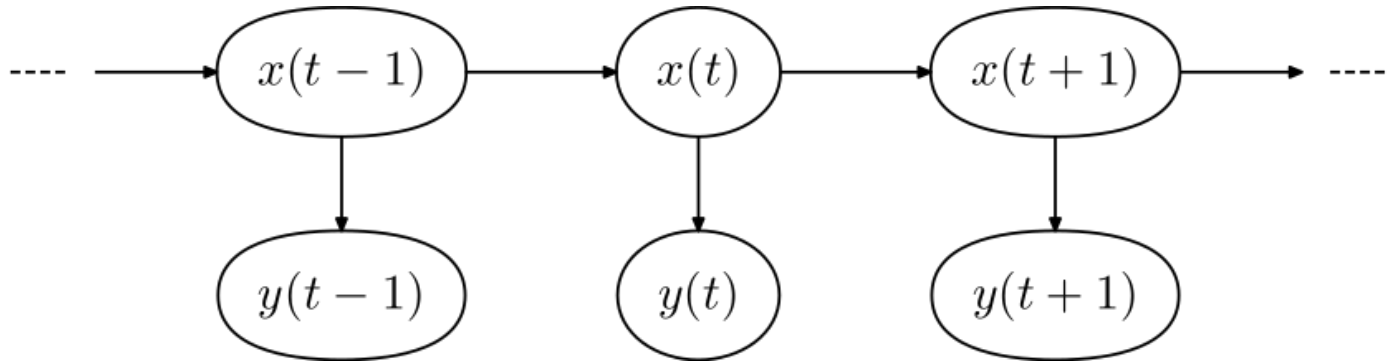
- Markov Models
  - N distinct states
  - Begins in some initial state(s)
  - At each time step, the system moves from current to next state according to transition probabilities associated with current state
  - This model is a discrete (finite) system
- **Hidden** Markov Models
  - We can not observe state



# Hidden Markov Models (Discrete Case)

- Hidden state at step(time) $t$: $x_t \in \{s_1, \cdots, s_N\}$
- Observations state at step(times) $t$: $y_t \in \{v_1, \cdots, v_M\}$
- Vector $\boldsymbol{\pi} \in \mathbb{R}^N$ represents the intitial probability
- Markov Chain property (First Order): probability of each subsequent state depends only on what was the previous state: $P(x_t|x_1, x_2, \cdots, x_{t-1}) = P(x_t|x_{t-1})$
- States are not visible, but each state randomly generates one of M observations (or visible states), which follows the observation probability: $p(y_t|x_t)$.

# Hidden Markov Models (Discrete Case)




If we discretize the transition and observation probabilities, we get two matrixs:
**Transition Matrix** $A \in \mathbb{R}^{M \times M}$, each element $A_{ij} = P(x_t = s_i | x_{t-1} = s_j)$, and $\sum_i A_{ij} = 1$
**Emission Matrix** $B \in \mathbb{R}^{M \times N}$, each element $B_{ij} = P(y_t = v_i | x_t = s_j)$, and $\sum_i B_{ij} = 1$
Then, along with the initial vector $\boldsymbol{\pi}$, the hidden markov model is represented by $Model = (A, B, \boldsymbol{\pi})$

# Concepts

- Supervised v.s. Unsupervised
- (One v.s. One) v.s. (One v.s. All)
- MAP v.s. MLE
- Deterministic v.s. Probabilistic
- Kmeans v.s. Kmedoids
- Hard SVM v.s. Soft SVM
- loss function : Hinge loss, "perceptron loss", etc.
- Weak hypothesis v.s. Strong hypothesis
- Kernel perceptron v.s. Kernel SVM
- Discriminative model v.s. Generative model
- Neural Network: how it works, back propagation,

# Good Luck!