

复习课



关于考试

■ 总成绩

- 平时成绩 40%+ 考试 60%

■ 考试题型

- 简答题
- 演算题
- 推导题
- 证明题
- 推导题、证明题：一定灵活性或综合性

■ 考试要求

- 时间：120分钟，6月18日下午
- 闭卷考试：不能携带任何资料
- 英文作答（实在忘记的单词：...）
- 不带计算器：
计算不复杂，根据题目需要给出分数形式结果即可



Lecture 1

Introduction



Example: “Sheep” vs. “Goat” (Cont.)

Generalization

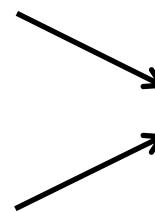
[泛化能力/推广能力]

The ultimate goal!

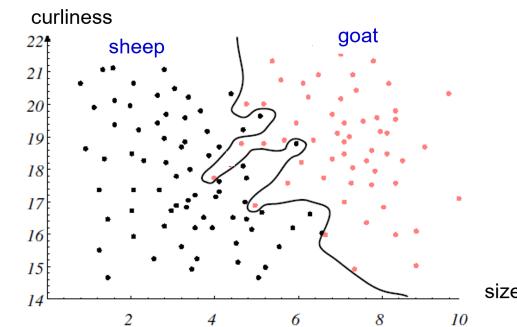
The central aim of designing a classifier is to make correct decisions when presented with *novel (unseen/test)* patterns, not on training patterns whose labels are already known

Performance on
the training set

Simplicity of
the classifier



Tradeoff



Generalization Error

Definitions of the **generalization error** and **empirical error** from
“*Foundations of Machine Learning*”

Definition 2.1 (Generalization error) *Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and an underlying distribution \mathcal{D} , the generalization error or risk of h is defined by*

$$R(h) = \mathbb{P}_{x \sim \mathcal{D}} [h(x) \neq c(x)] = \mathbb{E}_{x \sim \mathcal{D}} [1_{h(x) \neq c(x)}],$$

where 1_ω is the indicator function of the event ω .²

The generalization error of a hypothesis is not directly accessible

Definition 2.2 (Empirical error) *Given a hypothesis $h \in \mathcal{H}$, a target concept $c \in \mathcal{C}$, and a sample $S = (x_1, \dots, x_m)$, the empirical error or empirical risk of h is defined by*

$$\hat{R}_S(h) = \frac{1}{m} \sum_{i=1}^m 1_{h(x_i) \neq c(x_i)}.$$

Lecture 2

Linear Regression



Linear Regression – loss function

- Minimize mean-squared error (MSE):

Loss function: How much \hat{y} differs from the true y

$$E_{(w,b)} = \sum_{i=1}^m (y_i - wx_i - b)^2$$

- Calculate the derivatives of $E_{(w,b)}$ with respect to w and b :

$$\frac{\partial E_{(w,b)}}{\partial w} = 2 \left(w \sum_{i=1}^m x_i^2 - \sum_{i=1}^m (y_i - b)x_i \right)$$

$$\frac{\partial E_{(w,b)}}{\partial b} = 2 \left(mb - \sum_{i=1}^m (y_i - wx_i) \right)$$

Linear Regression - Least Square Method

- We have the closed-form solutions

$$w = \frac{\sum_{i=1}^m y_i(x_i - \bar{x})}{\sum_{i=1}^m x_i^2 - \frac{1}{m} (\sum_{i=1}^m x_i)^2}$$

$$b = \frac{1}{m} \sum_{i=1}^m (y_i - wx_i)$$

where $\bar{x} = \frac{1}{m} \sum_{i=1}^m x_i$



Multivariate Linear Regression

- Rewrite w and b as $\hat{w} = (w; b)$, the data set is represented as

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{md} & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T & 1 \\ \mathbf{x}_2^T & 1 \\ \vdots & \vdots \\ \mathbf{x}_m^T & 1 \end{pmatrix}$$

$$\mathbf{y} = (y_1; y_2; \dots; y_m)$$



Multivariate Linear Regression - Least Square Method

□ Least square method

$$\hat{\mathbf{w}}^* = \arg \min_{\hat{\mathbf{w}}} (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$$

Let $E_{\hat{\mathbf{w}}} = (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})$ and find the derivative with respect to $\hat{\mathbf{w}}$

$$\frac{\partial E_{\hat{\mathbf{w}}}}{\partial \hat{\mathbf{w}}} = 2\mathbf{X}^T(\mathbf{X}\hat{\mathbf{w}} - \mathbf{y})$$

The closed-form solution of $\hat{\mathbf{w}}$ can be obtained by making the equation equal to 0.

Multivariate Linear Regression - Least Square Method

- If $\mathbf{X}^T \mathbf{X}$ is a full-rank matrix or a positive definite matrix, then

$$\hat{\mathbf{w}}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where $(\mathbf{X}^T \mathbf{X})^{-1}$ is the inverse of $\mathbf{X}^T \mathbf{X}$, the learned multivariate linear regression model is

$$f(\hat{\mathbf{x}}_i) = \hat{\mathbf{x}}_i^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- $\mathbf{X}^T \mathbf{X}$ is often not full-rank
 - gradient descent (which is more broadly applicable)
 - pseudo-inverse

Lecture 3

Logistic Regression



Binary Classification

- The predictions and the output labels

$$z = \mathbf{w}^T \mathbf{x} + b \quad y \in \{0, 1\}$$

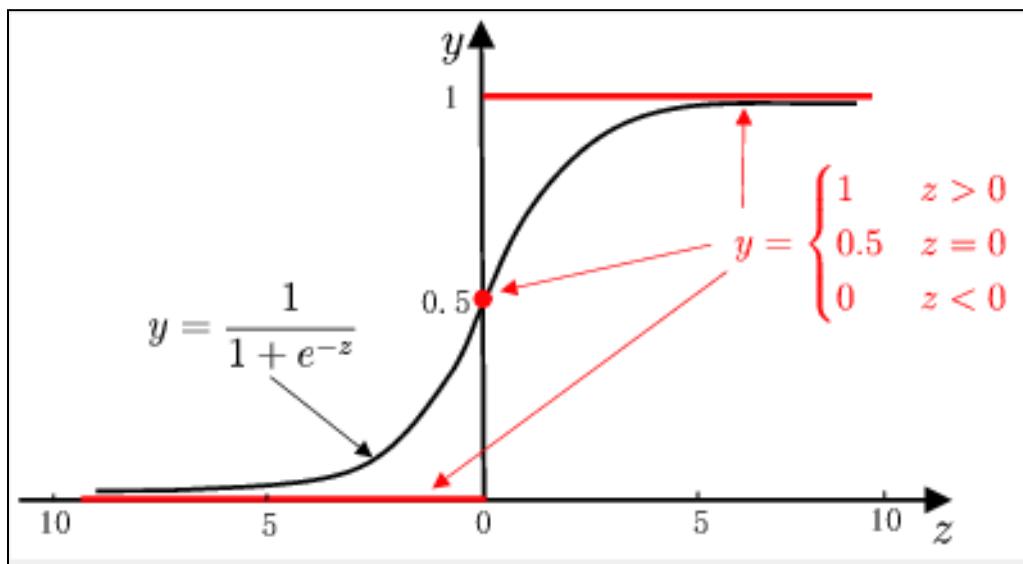
- The real-valued predictions of the linear regression model need to be converted into **0/1**.
- Ideally, the unit-step function is desired

$$y = \begin{cases} 0, & z < 0; \\ 0.5, & z = 0; \\ 1, & z > 0, \end{cases}$$

- which predicts positive for z greater than 0, negative for z smaller than 0, and an arbitrary output when z equals to 0.

Binary Classification

- Disadvantages of unit-step function
 - not continuous
- Logistic (sigmoid) function: a surrogate function to approximate the unit-step function
 - monotonic differentiable



Comparison between unit-step function and logistic function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

Logistic Regression

Data: Inputs are continuous vectors of length d . Outputs are discrete labels.

$$\mathcal{D} = \left\{ \mathbf{x}^{(i)}, y^{(i)} \right\}_{i=1}^m \text{ where } \mathbf{x} \in \mathbb{R}^d \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Prediction: Output is the most probable class.

$$\hat{y} = \operatorname{argmax}_{y \in \{0,1\}} p_{\boldsymbol{\theta}}(y \mid \mathbf{x})$$



Log odds

■ Apply logistic function

$$y = \frac{1}{1 + e^{-z}} \quad \text{transform into} \quad y = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}$$

■ Log odds

- the logarithm of the relative likelihood of a sample being a positive sample

$$\ln \frac{y}{1 - y} = \mathbf{w}^T \mathbf{x} + b$$

■ Logistic regression has several nice properties

- without requiring any prior assumptions on the data distribution
- it predicts labels together with associated probabilities
- it is solvable with numerical optimization methods.

Logistic regression - maximum likelihood

■ Maximum likelihood

- Given the training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
- Maximizing the probability of each sample being predicted as the ground-truth label
 - the log-likelihood to be maximized is:

$$\ell(\mathbf{w}, b) = \log \prod_{i=1}^m p(y_i \mid \mathbf{x}_i; \mathbf{w}, b)$$

- assumption that the training examples are independent:

$$\ell(\mathbf{w}, b) = \sum_{i=1}^m \log p(y_i \mid \mathbf{x}_i; \mathbf{w}, b)$$

Logistic regression - maximum likelihood

- Log odds can be rewritten as

$$\ln \frac{p(y = 1 \mid \mathbf{x})}{p(y = 0 \mid \mathbf{x})} = \mathbf{w}^T \mathbf{x} + b$$

and consequently,

$$p(y = 1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}^T \mathbf{x} + b}}{1 + e^{\mathbf{w}^T \mathbf{x} + b}} = \text{sigmoid}(\mathbf{w}^T \mathbf{x} + b)$$

$$\begin{aligned} p(y = 0 \mid \mathbf{x}) &= \frac{1}{1 + e^{\mathbf{w}^T \mathbf{x} + b}} = 1 - \text{sigmoid}(\mathbf{w}^T \mathbf{x} + b) \\ &= \text{sigmoid}(-(\mathbf{w}^T \mathbf{x} + b)) \end{aligned}$$

Logistic regression - maximum likelihood

- Transform into minimize negative log-likelihood
 - Let $\beta = (\mathbf{w}; b)$, $\hat{\mathbf{x}} = (\mathbf{x}; 1)$, $\mathbf{w}^T \mathbf{x} + b$ can be rewritten as $\beta^T \hat{\mathbf{x}}$
 - Let $p_1(\hat{\mathbf{x}}_i; \beta) = p(y = 1 | \hat{\mathbf{x}}; \beta)$

$$p_0(\hat{\mathbf{x}}_i; \beta) = p(y = 0 | \hat{\mathbf{x}}; \beta) = 1 - p_1(\hat{\mathbf{x}}_i; \beta)$$

the likelihood term in can be rewritten as

$$p(y_i | \mathbf{x}_i; \mathbf{w}_i, b) = y_i p_1(\hat{\mathbf{x}}_i; \beta) + (1 - y_i) p_0(\hat{\mathbf{x}}_i; \beta)$$

- maximizing log-likelihood is equivalent to minimizing

$$J(\beta) = \sum_{i=1}^m \left(-y_i \beta^T \hat{\mathbf{x}}_i + \log\left(1 + e^{\beta^T \hat{\mathbf{x}}_i}\right) \right)$$

Logistic regression - maximum likelihood

- Transform into minimize negative log-likelihood
 - Let $\beta = (\mathbf{w}; b)$, $\hat{\mathbf{x}} = (\mathbf{x}; 1)$, $\mathbf{w}^T \mathbf{x} + b$ can be rewritten as $\beta^T \hat{\mathbf{x}}$
 - Let $p_1(\hat{\mathbf{x}}_i; \beta) = p(y = 1 | \hat{\mathbf{x}}; \beta)$

$$p_0(\hat{\mathbf{x}}_i; \beta) = p(y = 0 | \hat{\mathbf{x}}; \beta) = 1 - p_1(\hat{\mathbf{x}}_i; \beta)$$

the likelihood term in can be rewritten as

$$p(y_i | \hat{\mathbf{x}}_i; \hat{\mathbf{w}}, b) = p_1(\hat{\mathbf{x}}_i; \beta)^{y_i} p_0(\hat{\mathbf{x}}_i; \beta)^{1-y_i}$$

- maximizing log-likelihood is equivalent to minimizing

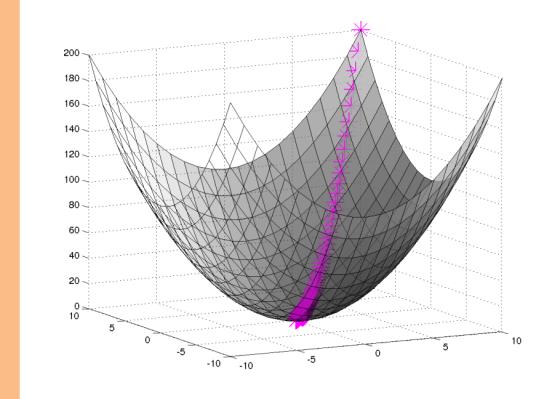
$$J(\beta) = \sum_{i=1}^m -[y_i \log p_1(\hat{\mathbf{x}}_i; \beta) + (1 - y_i) \log p_0(\hat{\mathbf{x}}_i; \beta)]$$

The Cross-Entropy loss!

Gradient Descent

Algorithm 1 Gradient Descent

```
1: procedure GD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:      $\theta \leftarrow \theta - \alpha \nabla_{\theta} J(\theta)$ 
5:   return  $\theta$ 
```



$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \frac{d}{d\theta_1} J(\theta) \\ \frac{d}{d\theta_2} J(\theta) \\ \vdots \\ \frac{d}{d\theta_N} J(\theta) \end{bmatrix}$$

$$\theta^{t+1} = \theta^t - \eta \nabla J_{\theta}(\theta))$$

Gradient for Logistic Regression

- The cross-entropy loss function

$$J(\boldsymbol{\beta}) = \sum_{i=1}^m -[y_i \log p_1(\hat{\mathbf{x}}_i; \boldsymbol{\beta}) + (1 - y_i) \log p_0(\hat{\mathbf{x}}_i; \boldsymbol{\beta})]$$

- The gradient

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = - \sum_{i=1}^m \hat{\mathbf{x}}_i (y_i - p_1(\hat{\mathbf{x}}_i; \boldsymbol{\beta}))$$

- Instead of using the sum notation, we can more efficiently compute the gradient in its matrix form

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathbf{X}(\sigma(\mathbf{X}^T \boldsymbol{\beta}) - \mathbf{y})$$

$\mathbf{X} \in \mathbb{R}^{d \times m}$
 σ : sigmoid



Lecture 4

Model Selection and Evaluation



Performance Measure

Error rate and **accuracy** are the most commonly used performance measures in classification problems :

- Error rate is the proportion of misclassified samples to all samples
- Accuracy is the proportion of correctly classified samples instead

Error rate

$$E(f; D) = \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(\mathbf{x}_i) \neq y_i)$$

Accuracy

$$\begin{aligned} \text{acc}(f; D) &= \frac{1}{m} \sum_{i=1}^m \mathbb{I}(f(\mathbf{x}_i) = y_i) \\ &= 1 - E(f; D) . \end{aligned}$$



Performance Measure

- We often want to know “*What percentage of the retrieved information is of interest to users?*” and “*How much of the information the user interested in is retrieved?*” in applications like information retrieval and web search. For such questions, **precision** and **recall** are better choices.
- In binary classification, there are four combinations of the ground-truth class and the predicted class, namely *true positive*, *false positive*, *true negative*, and *false negative*. The four combinations can be displayed in a confusion matrix.

The confusion matrix of binary classification

Ground-truth class	Predicted class	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

$$\text{Precision} \quad P = \frac{TP}{TP + FP}$$

$$\text{Recall} \quad R = \frac{TP}{TP + FN}$$



Lecture 6

Support Vector

Machines



The Lagrange Method

Consider a general optimization problem (called as primal problem)

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & g_i(x) \geq 0, i = 1, \dots, k \\ & h_j(x) = 0, j = 1, \dots, m. \end{aligned}$$

We define its Lagrangian as

$$L(x, u, v) = f(x) - \sum_{i=1}^k \lambda_i g_i(x) + \sum_{j=1}^m u_j h_j(x)$$

Lagrangian multipliers $\lambda \in \mathbb{R}^k, u \in \mathbb{R}^m$.



The Dual Problem

A re-written Primal Problem :

$$\min_x \max_{\lambda \geq 0, u} L(x, \lambda, u)$$

The Dual Problem:

$$\max_{\lambda \geq 0, u} \min_x L(x, \lambda, u)$$

Although the primal problem is not required to be convex, the dual problem is always convex.

Theorem (weak duality):

$$d^* = \max_{\lambda \geq 0, u} \min_x L(x, \lambda, u) \leq \min_x \max_{\lambda \geq 0, u} L(x, \lambda, u) = p^*$$

Theorem (strong duality, e.g., *Slater's condition*):

If the primal is a convex problem, and there exists at least one strictly feasible \tilde{x} , meaning that $\exists \tilde{x}, g_i(\tilde{x}) > 0, i = 1, \dots, k, h_j(\tilde{x}) = 0, j = 1, \dots, m$.

$$d^* = p^*$$



Karush–Kuhn–Tucker (KKT) conditions

Necessary conditions

If x^* and λ^*, u^* are the primal and dual solutions respectively with **zero duality gap**, we will show that x^*, λ^*, u^* satisfy the KKT conditions.

$f(x^*) = d(\lambda^*, u^*)$ by zero duality gap assumption

$$= \min_x f(x) - \sum_{i=1}^k \lambda_i^* g_i(x) + \sum_{j=1}^m u_j^* h_j(x), \text{ by definition}$$

stationarity

$$\leq f(x^*) - \sum_{i=1}^k \lambda_i^* g_i(x^*) + \sum_{j=1}^m u_j^* h_j(x^*) \Rightarrow \text{equality: } x^* \text{ minimizes } L(x, \lambda^*, u^*)$$

$$\leq f(x^*) \Rightarrow \text{equality: } \lambda_i^* g_i(x^*) = 0$$

complementary slackness

For **convex problems with strong duality** (e.g., when Slater's condition is satisfied), the KKT conditions are **necessary and sufficient optimality conditions**, i.e., x^* and (λ^*, u^*) are primal and dual optimal if and only if the KKT conditions hold.

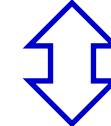
The Primal Form of SVM

Maximum margin: finding the parameters \mathbf{w} and b that maximize

$$\arg \max_{\mathbf{w}, b} \frac{2}{\|\mathbf{w}\|}$$

s.t. $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, i = 1, 2, \dots, m.$

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b &\geq 1, & \text{if } y_i = +1; \\ \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1, & \text{if } y_i = -1 \end{aligned}$$



$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

s.t. $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, i = 1, 2, \dots, m.$

This is an optimization problem with linear, inequality constraints.

Dual problem

■ Lagrange multipliers

- Step-1: introducing a Lagrange multiplier $\alpha_i \geq 0$, gives the Lagrange function

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\boldsymbol{w}\|^2 - \sum_{i=1}^m \alpha_i (y_i (\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1)$$

- Step-2: Setting the partial derivatives of $L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ with respect to \boldsymbol{w} and b to 0 gives

$$\boldsymbol{w} = \sum_{i=1}^m \alpha_i y_i \boldsymbol{x}_i, \quad \sum_{i=1}^m \alpha_i y_i = 0.$$

- Step-3: Substituting back

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j - \sum_{i=1}^m \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^m \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, 2, \dots, m. \end{aligned}$$

Sparsity of the solution

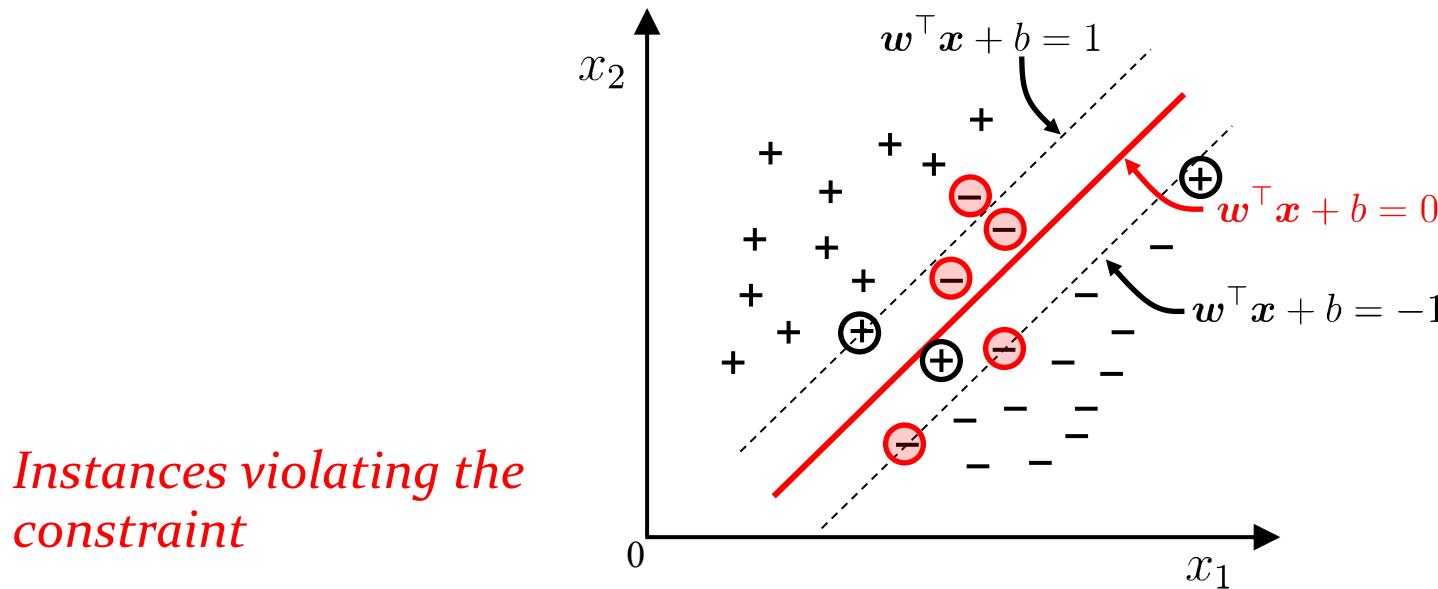
- desired model: $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b$
- KKT conditions:
 - $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$, $\sum_{i=1}^m \alpha_i y_i = 0$. stationarity
 - $\begin{cases} \alpha_i \geq 0, \\ y_i f(\mathbf{x}_i) \geq 1, \\ \alpha_i (y_i f(\mathbf{x}_i) - 1) = 0. \end{cases}$ dual constraints
 - primal constraints
 - complementary slackness

$$y_i f(\mathbf{x}_i) > 1 \rightarrow \alpha_i = 0$$

Sparsity of the solution of SVM: once the training completed, most training samples are no longer needed since the final model only depends on the support vectors.

Key idea #2: the slack variables

- Q: It is often difficult to find an appropriate kernel function to make the training samples linearly separable in the feature space. Even if we do find such a kernel function, it is hard to tell if it is a result of overfitting.
- A: Allow a support vector machine to make mistakes on a few samples: *soft margin*.



ℓ_1 relaxation of the penalty term

The discrete nature of the penalty term on previous slide, $\sum_i 1_{\xi_i > 0} = \|\vec{\xi}\|_0$, makes the problem intractable.

A common strategy is to replace the ℓ_0 penalty with a ℓ_1 penalty: $\sum_i \xi_i = \|\vec{\xi}\|_1$, resulting in the following full problem

$$\min_{\mathbf{w}, b, \vec{\xi}} \frac{1}{2} \|\mathbf{w}\|_2^2 + C \cdot \sum_i \xi_i$$

subject to $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i .

Remarks:

- (1) Also a quadratic program with linear ineq. constraints (just more variables): $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) + \xi_i \geq 1$.

The Lagrange dual problem

The associated Lagrange function is

$$L(\mathbf{w}, b, \vec{\xi}, \vec{\lambda}, \vec{\mu}) = \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \lambda_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^n \mu_i \xi_i$$

(stationary point) To find the dual problem we need to fix $\vec{\lambda}$, $\vec{\mu}$ and maximize over $\mathbf{w}, b, \vec{\xi}$:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum \lambda_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial L}{\partial b} = \sum \lambda_i y_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0, \quad \forall i$$

The Lagrange dual problem

This yields the Lagrange dual function

$$L^*(\vec{\lambda}, \vec{\mu}) = \sum \lambda_i - \frac{1}{2} \sum \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j, \quad \text{where}$$
$$\lambda_i \geq 0, \mu_i \geq 0, \lambda_i + \mu_i = C, \text{ and } \sum \lambda_i y_i = 0.$$

The dual problem would be to maximize L^* over $\vec{\lambda}, \vec{\mu}$ subject to the constraints.

Since L^* is constant with respect to the μ_i , we can eliminate them to obtain a reduced dual problem:

$$\max_{\lambda_1, \dots, \lambda_n} \sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

subject to $\underbrace{0 \leq \lambda_i \leq C}_{\text{box constraints}}$ and $\sum \lambda_i y_i = 0$.

What
changed?



What about the KKT conditions?

The KKT conditions are the following

$$\mathbf{w} = \sum \lambda_i y_i \mathbf{x}_i, \quad \sum \lambda_i y_i = 0, \quad \lambda_i + \mu_i = C$$

$$\lambda_i(y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i) = 0, \quad \mu_i \xi_i = 0$$

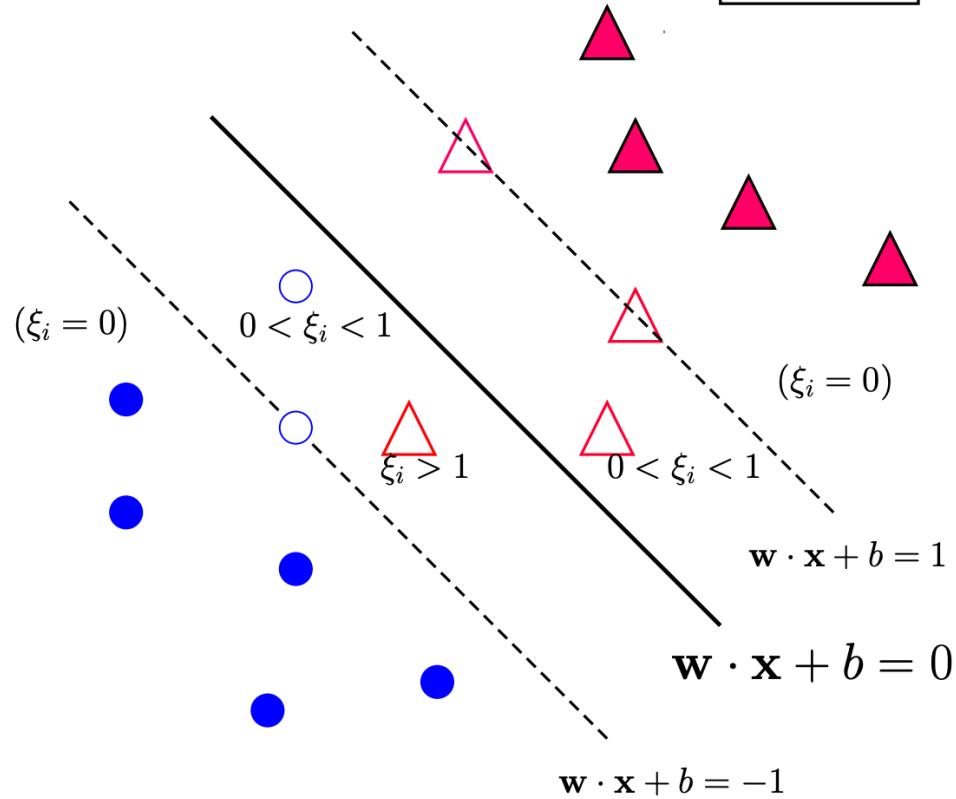
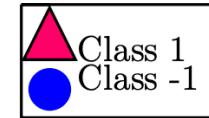
$$\lambda_i \geq 0, \quad \mu_i \geq 0$$

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

We see that

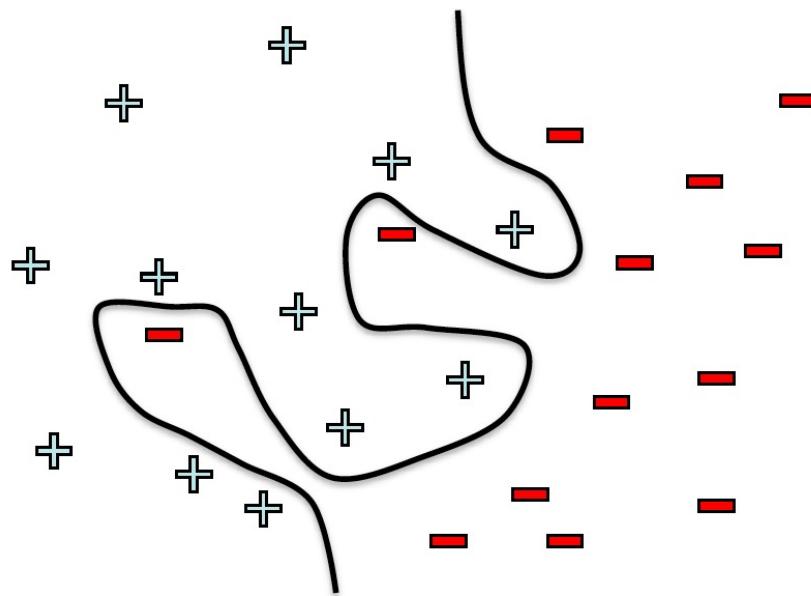
- The optimal \mathbf{w} has the same formula: $\mathbf{w} = \sum \lambda_i y_i \mathbf{x}_i$.
- Any point with $\lambda_i > 0$ and correspondingly $y_i(\mathbf{w} \cdot \mathbf{x} + b) = 1 - \xi_i$ is a support vector (not just those on the margin boundary $\mathbf{w} \cdot \mathbf{x} + b = \pm 1$).

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \forall i$$



What if the data is not linearly separable?

**Use features of features
of features of features....**



$$\phi(x) = \begin{pmatrix} x^{(1)} \\ \vdots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \vdots \\ e^{x^{(1)}} \\ \vdots \end{pmatrix}$$

Kernel SVM

Let $\phi(\mathbf{x})$ denote the mapped feature vector of \mathbf{x} , the separating hyperplane $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$ can be expressed as

Primal
Problem

$$\begin{aligned} & \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t. } & y_i (\mathbf{w}^\top \phi(\mathbf{x}_i) + b) \geq 1, \quad i = 1, 2, \dots, m. \end{aligned}$$

Dual
Problem

$$\begin{aligned} & \min_{\alpha} \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \boxed{\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)} - \sum_{i=1}^m \alpha_i \\ \text{s.t. } & \sum_{i=1}^m \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, 2, \dots, m. \end{aligned}$$

Prediction

$$f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b = \sum_{i=1}^m \alpha_i y_i \boxed{\phi(\mathbf{x}_i)^\top \phi(\mathbf{x})} + b$$



What are good kernel functions?

- Linear kernel

- $K(x_i, x_j) = \phi(x_i)\phi(x_j) = x_i \cdot x_j$

- Polynomial

- $K(x_i, x_j) = (x_i \cdot x_j + 1)^n$

- Gaussian (also called Radial Basis Function, or RBF)

- $K(x_i, x_j) = e^{\frac{\|x_i - x_j\|^2}{2\sigma^2}}$

- ...

Quadratic kernel

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z} + c)^2 = \left(\sum_{j=1}^n x^{(j)} z^{(j)} + c \right) \left(\sum_{\ell=1}^n x^{(\ell)} z^{(\ell)} + c \right) \\ &= \sum_{j=1}^n \sum_{\ell=1}^n x^{(j)} x^{(\ell)} z^{(j)} z^{(\ell)} + 2c \sum_{j=1}^n x^{(j)} z^{(j)} + c^2 \\ &= \sum_{j,\ell=1}^n (x^{(j)} x^{(\ell)}) (z^{(j)} z^{(\ell)}) + \sum_{j=1}^n (\sqrt{2c} x^{(j)}) (\sqrt{2c} z^{(j)}) + c^2, \end{aligned}$$

Feature mapping given by:

$$\Phi(\mathbf{x}) = [x^{(1)2}, x^{(1)}x^{(2)}, \dots, x^{(3)2}, \sqrt{2c}x^{(1)}, \sqrt{2c}x^{(2)}, \sqrt{2c}x^{(3)}, c]$$

Representer theorem

SVM $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b = \sum_{i=1}^m \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b$

SVR $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b = \sum_{i=1}^m (\hat{\alpha}_i - \alpha_i) y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b$

Conclusion: The learned models of SVM and SVR can be expressed as a linear combination of the kernel functions. A more generalized conclusion(representer theorem): for any increasing function Ω and any non-negative loss function l , the optimization problem

$$\min_{h \in \mathbb{H}} F(h) = \Omega(\|h\|_{\mathbb{H}}) + l(h(\mathbf{x}_1), \dots, h(\mathbf{x}_m))$$

Solution can be written in the form of
$$h^* = \sum_{i=1}^m \alpha_i \kappa(\cdot, \mathbf{x}_i)$$



Lecture 8

Backpropagation



Backpropagation Summary

- Forward pass:** for each training example, compute the outputs for all layers:

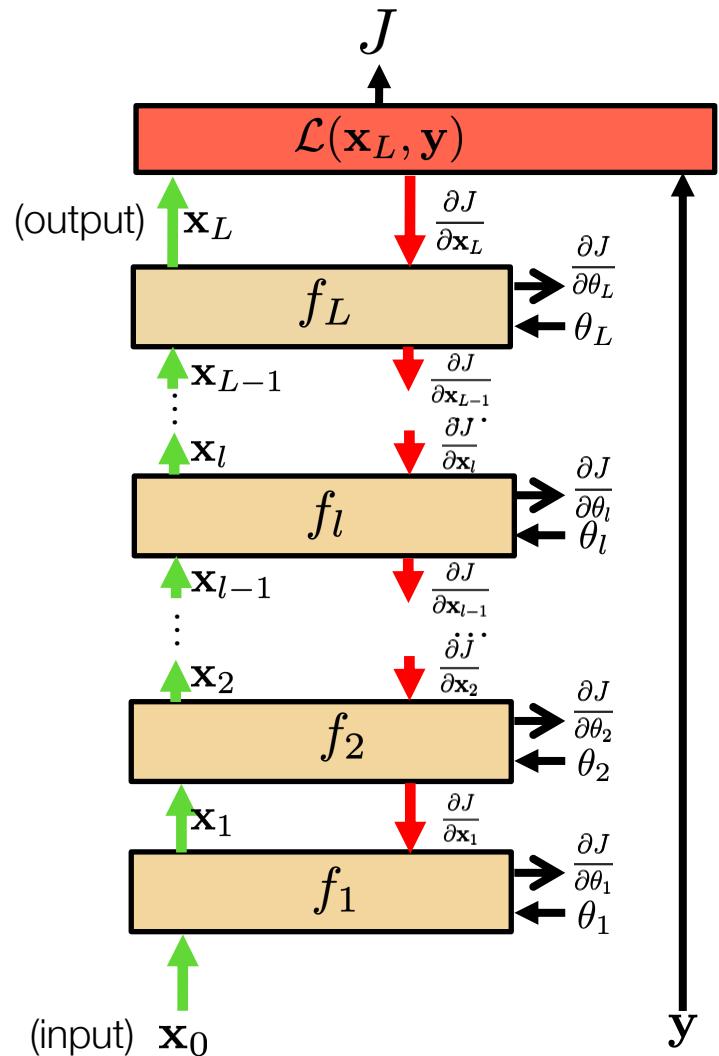
$$\mathbf{x}_l = f_l(\mathbf{x}_{l-1}, \theta_l)$$

- Backwards pass:** compute loss derivatives iteratively from top to bottom:

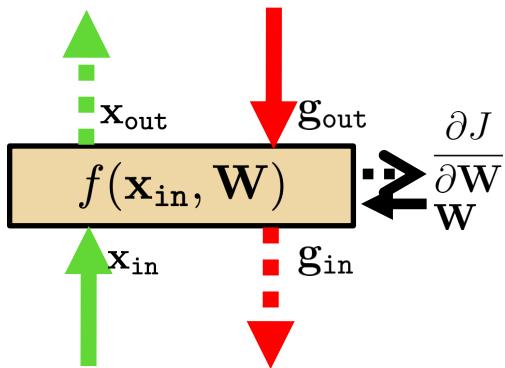
$$\frac{\partial J}{\partial \mathbf{x}_{l-1}} = \frac{\partial J}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l}{\partial \mathbf{x}_{l-1}}$$

- Parameter update:** Compute gradients w.r.t. weights, and update weights:

$$\frac{\partial J}{\partial \theta_l} = \frac{\partial J}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l}{\partial \theta_l}$$



Linear layer



- Forward propagation: $\mathbf{x}_{\text{out}} = f(\mathbf{x}_{\text{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\text{in}}$
- Backprop to input:

$$\mathbf{g}_{\text{in}} = \mathbf{g}_{\text{out}} \cdot \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{x}_{\text{in}}} = \mathbf{g}_{\text{out}} \cdot \frac{\partial \mathbf{x}_{\text{out}}}{\partial \mathbf{x}_{\text{in}}}$$

If we look at the i component of output \mathbf{x}_{out} , with respect to the j component of the input, \mathbf{x}_{in} :

$$\frac{\partial \mathbf{x}_{\text{out}_i}}{\partial \mathbf{x}_{\text{in}_j}} = \mathbf{W}_{ij} \rightarrow \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{x}_{\text{in}}} = \mathbf{W}$$

Therefore:

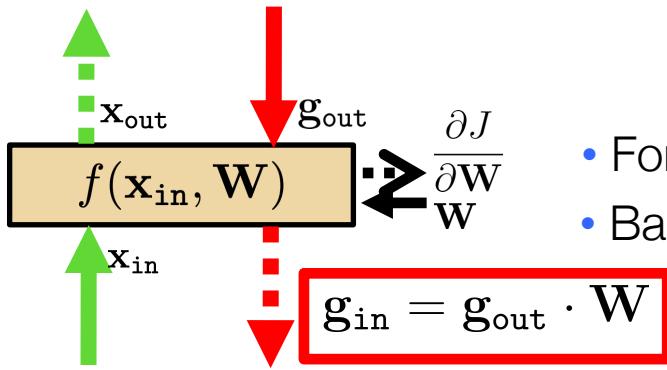
$$\mathbf{g}_{\text{in}} = \mathbf{g}_{\text{out}} \cdot \mathbf{W}$$

$$\mathbf{x}_{\text{out}} = \mathbf{W} \mathbf{x}_{\text{in}}$$

With \mathbf{W} being a matrix of size $|\mathbf{x}_{\text{out}}| \times |\mathbf{x}_{\text{in}}|$

$$\mathbf{g}_{\text{in}} = \mathbf{g}_{\text{out}} \cdot \mathbf{W}$$

Linear layer



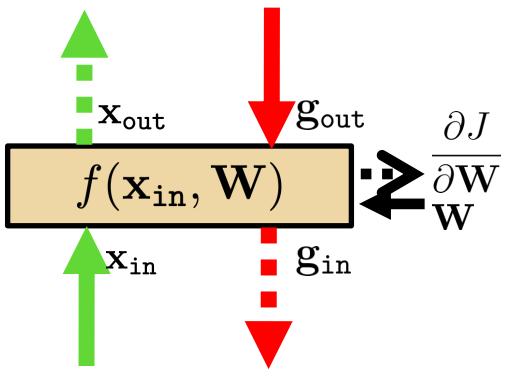
- Forward propagation: $\mathbf{x}_{\text{out}} = f(\mathbf{x}_{\text{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\text{in}}$
- Backprop to input:

$$\mathbf{g}_{\text{in}} = \mathbf{g}_{\text{out}} \cdot \mathbf{W}$$

A diagram showing matrix multiplication. On the left is a red vector labeled \mathbf{g}_{in} consisting of four red squares. In the middle is a red vector labeled \mathbf{g}_{out} consisting of four red squares. To its right is a blue matrix labeled \mathbf{W} consisting of a 2x2 grid of blue squares. A horizontal equals sign is placed between the \mathbf{g}_{out} and the \mathbf{W} terms.

Now let's see how we use the set of outputs to compute the weights update equation (backprop to the weights).

Linear layer



- Forward propagation: $\mathbf{x}_{out} = f(\mathbf{x}_{in}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{in}$
- Backprop to weights:

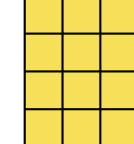
$$\frac{\partial J}{\partial \mathbf{W}} = \mathbf{g}_{out} \cdot \frac{\partial f(\mathbf{x}_{in}, \mathbf{W})}{\partial \mathbf{W}} = \mathbf{g}_{out} \cdot \frac{\partial \mathbf{x}_{out}}{\partial \mathbf{W}}$$

If we look at how the parameter W_{ij} changes the cost, only the i component of the output will change, therefore:

$$\frac{\partial J}{\partial \mathbf{W}_{ij}} = \frac{\partial J}{\partial \mathbf{x}_{out_i}} \cdot \frac{\partial \mathbf{x}_{out_i}}{\partial \mathbf{W}_{ij}} \underset{\substack{\uparrow \\ \frac{\partial \mathbf{x}_{out_i}}{\partial \mathbf{W}_{ij}} = \mathbf{x}_{in_j}}}{=} \frac{\partial J}{\partial \mathbf{x}_{out_i}} \cdot \mathbf{x}_{in_j}$$

$$\frac{\partial J}{\partial \mathbf{W}} = \mathbf{x}_{in} \cdot \frac{\partial J}{\partial \mathbf{x}_{out}} = \mathbf{x}_{in} \cdot \mathbf{g}_{out}$$

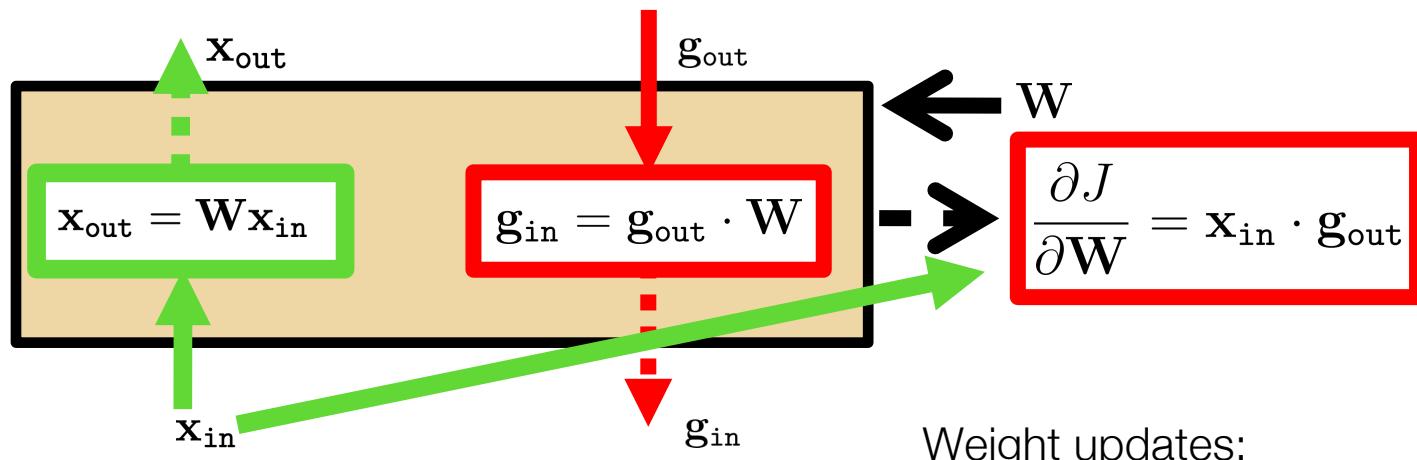
$$\frac{\partial J}{\partial \mathbf{W}} = \mathbf{x}_{in} \mathbf{g}_{out}$$



And now we can update the weights:

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^k + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^T$$

Linear layer



Weight updates:

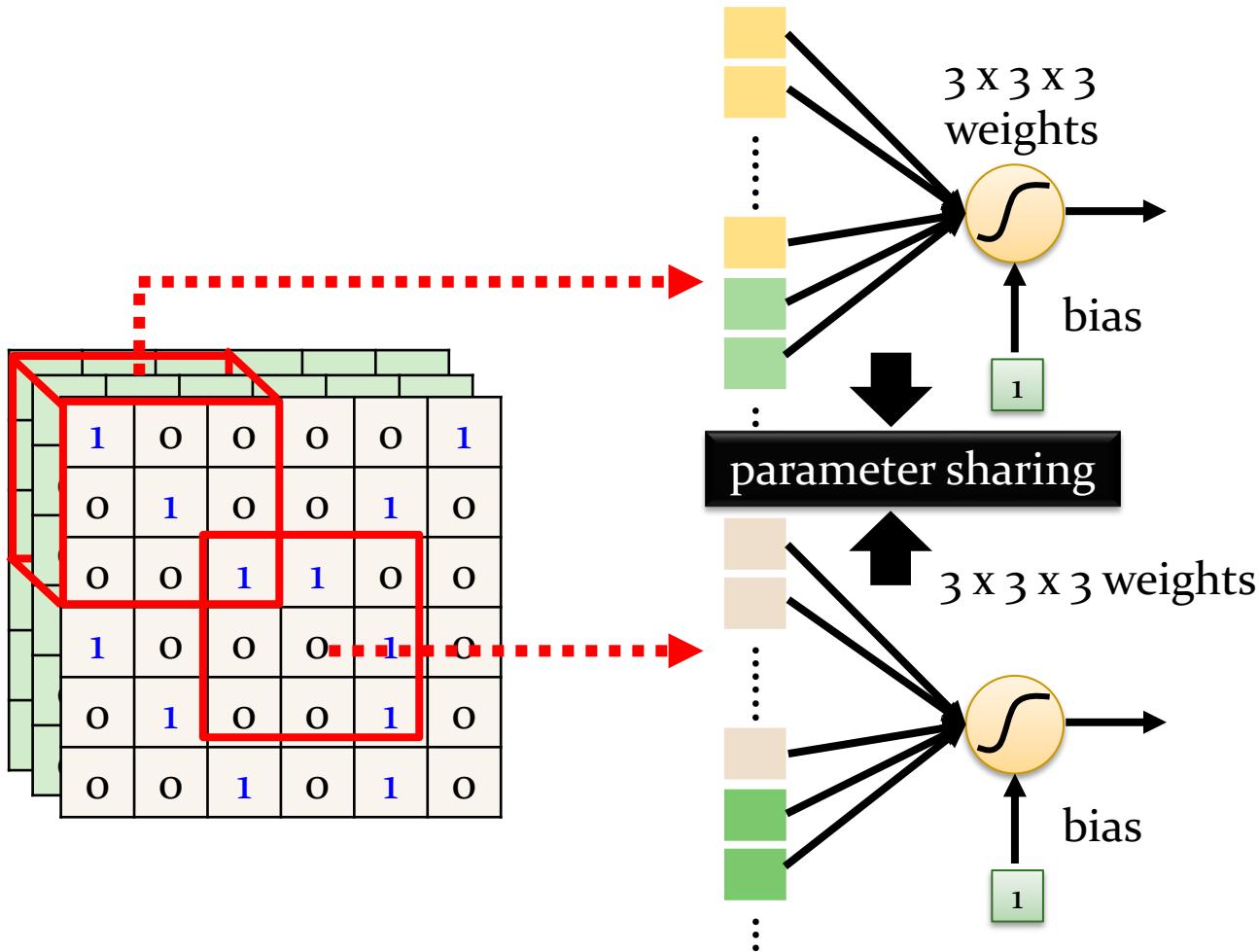
$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^k + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^T$$

Lecture 9

Convolution Neural Network



Simplification 2



Pooling – Max Pooling

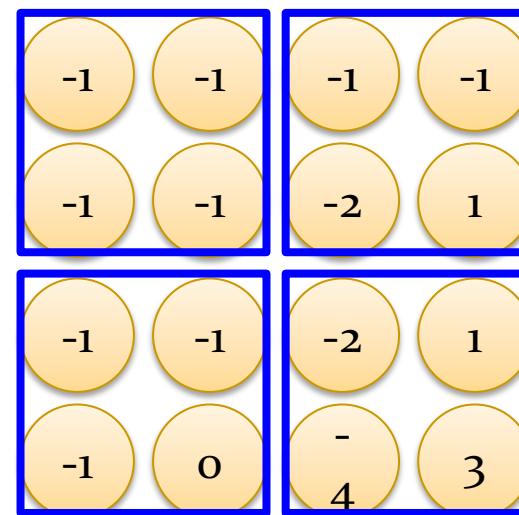
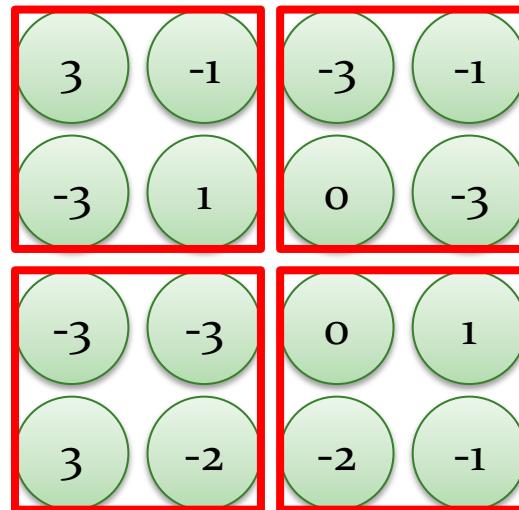
No learnable parameters!

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



Chapter 11

Decision Tree



Basic Process

Algorithm 4.1 Decision Tree Learning.

Input: Training set $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$;
Feature set $A = \{a_1, a_2, \dots, a_d\}$.

Process: Function TreeGenerate(D, A)

```
1: Generate node  $i$ ;  
2: if All samples in  $D$  belong to the same class  $C$  then  
3:   Mark node  $i$  as a class  $C$  leaf node; return  
4: end if  
5: if  $A = \emptyset$  OR all samples in  $D$  take the same value on  $A$  then  
6:   Mark node  $i$  as a leaf node, and its class label is the majority class in  $D$ ; return  
7: end if  
8: Select the optimal splitting feature  $a_*$  from  $A$ ;  
9: for each value  $a_*^v$  in  $a_*$  do  
10:   Generate a branch for node  $i$ ; Let  $D_v$  be the subset of samples taking value  $a_*^v$  on  $a_*$ ;  
11:   if  $D_v$  is empty then  
12:     Mark this child node as a leaf node, and label it with the majority class in  $D$ ; return  
13:   else  
14:     Use TreeGenerate( $D_v, A \setminus \{a_*\}$ ) as the child node.  
15:   end if  
16: end for
```

Output: A decision tree with root node i .

(1) All samples in the current node belong to the same class.

(2) The current feature set is empty, or all samples have the same feature values.

(3) There is no sample in the current node.



Split Selection: Information Gain

- Suppose that the discrete feature a has V possible values $\{a^1, a^2, \dots, a^V\}$. Then, splitting the data set D by feature a produces V child nodes, where the v th child node D^v includes all samples in D taking the value a^v for feature a . Then, the *information gain* of splitting the data set D with feature a is calculated as

$$\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} \text{Ent}(D^v)$$

is the importance of each node. The greater the number of samples, the greater the impact of the branch node.

- In general, **the higher the information gain, the more purity improvement** we can expect by splitting D with feature a .
- The decision tree algorithm ID3 [Quinlan, 1986] takes information gain as the guideline for selecting the splitting features.

Pruning

- Why pruning?
 - *Pruning* is the primary strategy of decision tree learning algorithms to **deal with overfitting**.
 - If there are too many branches, then the learner may be misled by the peculiarities of the training samples and incorrectly consider them as the underlying truth.
- General Pruning Strategies
 - *pre-pruning*
 - *post-pruning*
- How to evaluate generalization ability after pruning?
 - We can use the **hold-out method** to reserve part of the data as a validation set for performance evaluation.



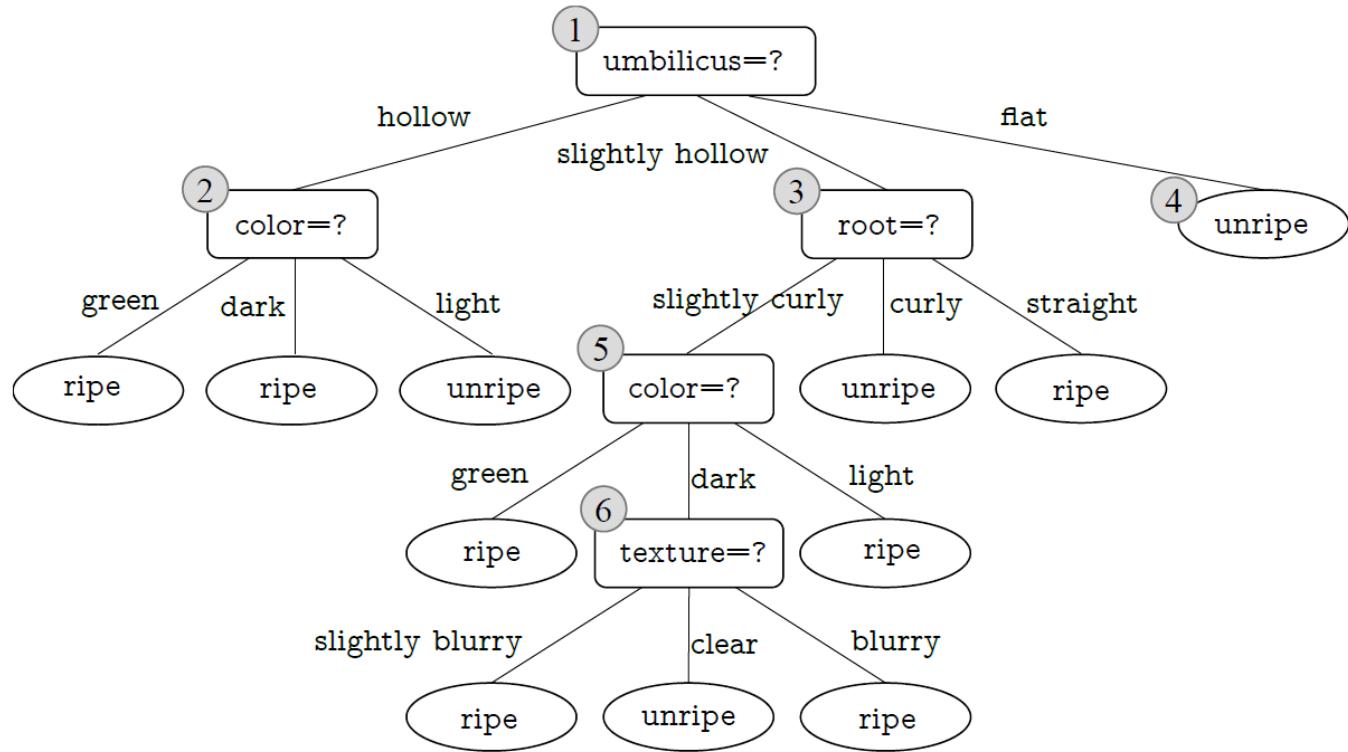
Pruning: Pre-pruning

- Pre-pruning decides by comparing **the generalization abilities before and after splitting.**
 - If the validation accuracy decreases after pruning, the splitting is accepted.
 - Otherwise, the splitting is rejected.
- When no splitting is performed, this node is marked as a leaf node and its label is set to the majority class.

Pruning: Post-pruning

- Post-pruning allows a decision tree to grow into a complete tree. Then it takes a bottom-up strategy to examine every non-leaf node in the completely grown decision tree.

The validation accuracy of this decision tree is 42.9%



Chapter 12

Bayesian Classifier



Bayes Decision Theory

- Bayesian decision theory is a fundamental decision-making approach under the probability framework.
 - When all relevant probabilities were known, Bayesian decision theory makes optimal classification decisions based on the probabilities and costs of misclassifications.
- Let us assume that there are N distinct class labels, that is, $y = \{c_1, c_2, \dots, c_N\}$. Let λ_{ij} denote the cost of misclassifying a sample of class c_j as class c_i . Then, with the posterior probability $P(c_i | x)$ we can calculate the expected loss of classifying a sample x as class c_i , that is, the conditional risk of the sample x :

$$R(c_i | x) = \sum_{j=1}^N \lambda_{ij} P(c_j | x) \quad (7.1)$$

- Our task is to find a decision rule $h : X \mapsto Y$ that minimizes the overall risk:

$$R(h) = \mathbf{E}_x [R(h(x) | x)] \quad (7.2)$$

Bayes Decision Theory

- The overall risk $R(h)$ is minimized when the conditional risk $R(h(\mathbf{x}) \mid \mathbf{x})$ of each sample \mathbf{x} is minimized.
- This leads to the Bayes decision rule: to minimize the overall risk, classify each sample as the class that minimizes the conditional risk $R(c \mid \mathbf{x})$

$$h^*(\mathbf{x}) = \operatorname{argmin}_{c \in \mathcal{Y}} R(c \mid \mathbf{x})$$

- where h^* is called the **Bayes optimal classifier**, and its associated overall risk $R(h^*)$ is called the Bayes risk.
- $1 - R(h^*)$ is the best performance that can be achieved by any classifiers, that is, the theoretically achievable upper bound of accuracy for any machine learning models.

Bayes Decision Theory

- For generative models, we must evaluate:

$$P(c | \mathbf{x}) = \frac{P(\mathbf{x}, c)}{P(\mathbf{x})}$$

- According to Bayes' theorem, $P(c | \mathbf{x})$ can be written as:

$$P(c | \mathbf{x}) = \frac{P(c)P(\mathbf{x} | c)}{P(\mathbf{x})}$$

the prior probability represents the proportion of each class in the sample, which can be estimated by the frequency of each class in the training set

the class-conditional probability, also known as the likelihood, of the sample \mathbf{x} with respect to class c

the evidence factor, which is independent of the class

Naïve Bayes Classifier

$$P(c \mid \mathbf{x}) = \frac{P(c)P(\mathbf{x} \mid c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^d P(x_i \mid c)$$

Since $P(x)$ is the same for all classes, from the Bayes decision rule, we have

$$h_{nb}(\mathbf{x}) = \operatorname{argmax}_{c \in \mathcal{Y}} P(c) \prod_{i=1}^d P(x_i \mid c)$$

which is the formulation of the Naïve Bayes classifier.

Naïve Bayes Classifier

- To train a Naïve Bayes classifier, we compute the **prior probability** $P(c)$ from the training set D and then compute the **conditional probability** $P(x_i | c)$ for each attribute.
- Let D_c denote a subset of D containing all samples of class c . Then, The prior probability can be estimated by

$$P(c) = \frac{|D_c|}{|D|}$$

- For discrete attributes, let D_{c,x_i} denote a subset of D_c containing all samples taking the value x_i on the i -th attribute. Then, the conditional probability $P(x_i | c)$ can be estimated by
- For continuous features, suppose $p(x_i | c) \sim N(\mu_{c,i}, \sigma_{c,i}^2)$, where $\mu_{c,i}$ and $\sigma_{c,i}^2$ are, respectively, the mean and variance of the i -th feature of class c . Then, we have

$$P(x_i | c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

Laplace (add-1) Smoothing

- To avoid “removing” the information carried by other features, a common choice is the Laplace smoothing.
- Let N denote the number of distinct classes in the training set D , N_i denote the number of distinct values the i -th feature can take. Then, we write smoothed version of prior probability and conditional probability as:

$$\hat{P}(c) = \frac{|D_c| + 1}{|D| + N} ,$$

Why?

$$\hat{P}(x_i | c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i}$$

Text Classification

The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

Multinomial Distribution

Suppose one does an experiment of extracting n balls of k different colors from a bag, replacing the extracted balls after each draw. Balls of the same color are equivalent. Denote the variable which is the number of extracted balls of color i ($i = 1, \dots, k$) as X_i , and denote as p_i the probability that a given extraction will be in color i .

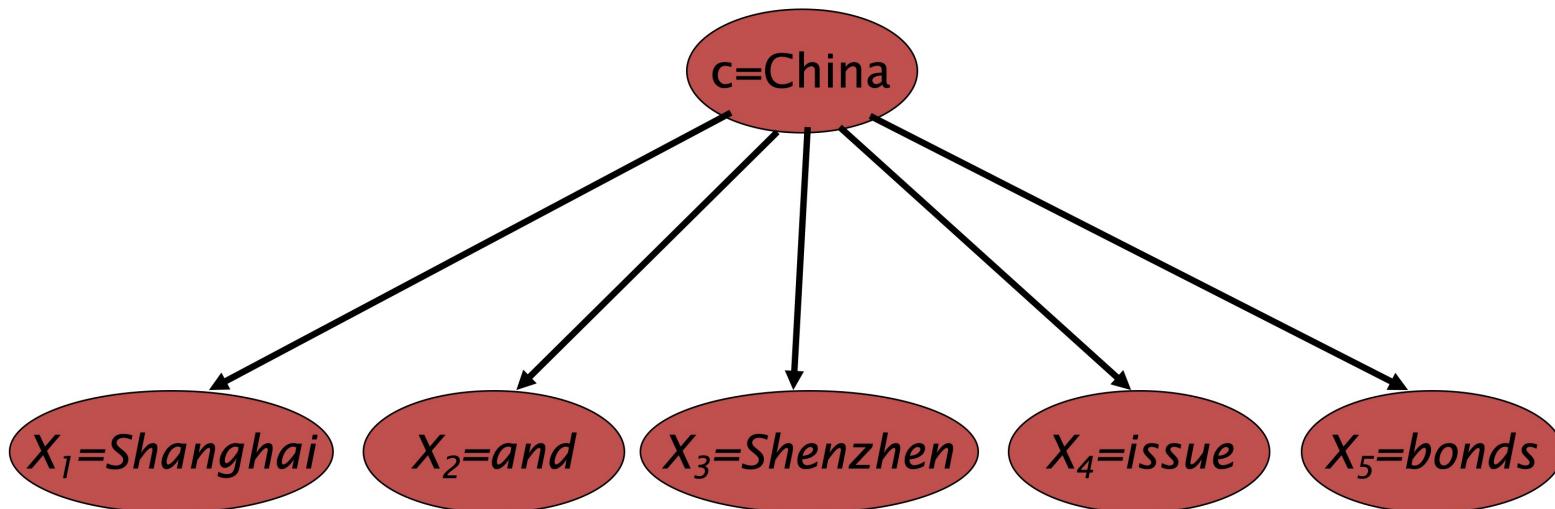
The **probability mass function** of this multinomial distribution is:

$$f(x_1, \dots, x_k; n, p_1, \dots, p_k) = \Pr(X_1 = x_1 \text{ and } \dots \text{ and } X_k = x_k)$$
$$= \begin{cases} \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \times \cdots \times p_k^{x_k}, & \text{when } \sum_{i=1}^k x_i = n \\ 0 & \text{otherwise,} \end{cases}$$

for non-negative integers x_1, \dots, x_k .

Generative Model for Naive Bayes

$$P(x_i | c)$$



Text Classification

Consider a naive Bayes model with the classes positive (+) and negative (-) and the following model parameters:

w	P(w +)	P(w -)
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.1	0.1
...

$$P(\text{"I love this fun film"}|+) = 0.1 \times 0.1 \times 0.01 \times 0.05 \times 0.1 = 0.0000005$$

$$P(\text{"I love this fun film"}|-) = 0.2 \times 0.001 \times 0.01 \times 0.005 \times 0.1 = .0000000010$$

Note that this is just the likelihood part of the naive Bayes model.

Text Classification

To apply the naive Bayes classifier to text, we need to consider word positions, by simply walking an index through every word position in the document:

positions \leftarrow all word positions in test document

$$c_{NB} = \operatorname{argmax}_{c \in C} P(c) \prod_{i \in \text{positions}} P(w_i | c)$$

Naive Bayes calculations are done in log space, to avoid underflow and increase speed

$$c_{NB} = \operatorname{argmax}_{c \in C} \log P(c) + \sum_{i \in \text{positions}} \log P(w_i | c)$$

Naive Bayes is a **linear classifiers**.

Training the Naive Bayes Classifier

Let N_c be the number of documents in our training data with class c and N_{doc} be the total number of documents. Then:

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)}$$

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V} (\text{count}(w, c) + 1)} = \frac{\text{count}(w_i, c) + 1}{(\sum_{w \in V} \text{count}(w, c)) + |V|}$$

Text Classification

	Doc	Words	Class
Training	1	Chinese Beijing Chinese	c
	2	Chinese Chinese Shanghai	c
	3	Chinese Macao	c
	4	Tokyo Japan Chinese	j
Test	5	Chinese Chinese Chinese Tokyo Japan	?

Priors:

$$P(c) = ?$$

$$P(j) = ?$$

Conditional Probabilities:

$$P(\text{Chinese}|c) = ?$$

$$P(\text{Tokyo}|c) = ?$$

$$P(\text{Japan}|c) = ?$$

$$P(\text{Chinese}|j) = ?$$

$$P(\text{Tokyo}|j) = ?$$

$$P(\text{Japan}|j) = ?$$

Choosing a class:

$$P(c|d5) = ?$$

$$P(j|d5) = ?$$

$$\hat{P}(w|c) = \frac{\text{count}(w,c)+1}{\text{count}(c)+|V|}$$

Text Classification

Priors:

$$P(c) = \frac{3 + 1}{4 + 2} = \frac{2}{3}$$

$$P(j) = \frac{1 + 1}{4 + 2} = \frac{1}{3}$$

Text Classification

Conditional Probabilities:

$$P(\text{Chinese}|c) = (5+1) / (8+6) = 6/14 = 3/7$$

$$P(\text{Tokyo}|c) = (0+1) / (8+6) = 1/14$$

$$P(\text{Japan}|c) = (0+1) / (8+6) = 1/14$$

$$P(\text{Chinese}|j) = (1+1) / (3+6) = 2/9$$

$$P(\text{Tokyo}|j) = (1+1) / (3+6) = 2/9$$

$$P(\text{Japan}|j) = (1+1) / (3+6) = 2/9$$

Text Classification

Choosing a class:

$$P(c|d5) \propto \frac{2}{3} * \left(\frac{3}{7}\right)^3 * \frac{1}{14} * \frac{1}{14} \approx 0.00027$$

$$P(j|d5) \propto \frac{1}{3} * \left(\frac{2}{9}\right)^3 * \frac{2}{9} * \frac{2}{9} \approx 0.00018$$

Chapter 13

Ensemble Learning

Bagging

- Bagging = Bootstrap AGGregatING
-

Algorithm 8.2 Bagging.

Input: Training set: $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\}$;
Base learning algorithm \mathfrak{L} ;
Number of training rounds T .

Process:

- 1: **for** $t = 1, 2, \dots, T$ **do**
- 2: $h_t = \mathfrak{L}(D, \mathcal{D}_{bs})$.
- 3: **end for**

Output: $H(\mathbf{x}) = \arg \max_{y \in \mathcal{Y}} \sum_{t=1}^T \mathbb{I}(h_t(\mathbf{x}) = y)$.

The bootstrap is one of the most important ideas in all of statistics!

Random Forests

- Random Forests = bagged decision trees, with one extra trick to decorrelate the predictions
 - When choosing each node of the decision tree, choose a random set of input features, and only consider splits on those features
- Random forests are probably the best black-box machine learning algorithm — they often work well with no tuning whatsoever.
 - one of the most widely used algorithms in Kaggle competitions

Chapter 14

Clustering

***k*-means Convergence**

Objective

$$\min_{\mu} \min_C \sum_{i=1}^k \sum_{x \in C_i} |x - \mu_i|^2$$

1. Fix μ , optimize C :

$$\min_C \sum_{i=1}^k \sum_{x \in C_i} |x - \mu_i|^2 = \min_c \sum_i^n |x_i - \mu_{x_i}|^2$$

Step 1 of kmeans

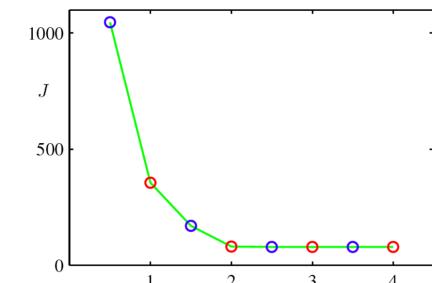
2. Fix C , optimize μ :

$$\min_{\mu} \sum_{i=1}^k \sum_{x \in C_i} |x - \mu_i|^2$$

- Take partial derivative of μ_i and set to zero, we have

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Step 2 of kmeans



Kmeans takes an alternating optimization approach, each step is guaranteed to decrease the objective – thus guaranteed to converge

[Slide from Alan Fern]

Hierarchical Clustering

- Hierarchical Clustering aims to create a tree-like clustering structure by dividing a data set at different layers. The hierarchy of clusters can be formed by taking either a *bottom-up* strategy (**Agglomerative** , 聚集) or a *top-down* strategy (**Divisive** , 分裂).

- **AGNES algorithm** (*bottom-up Hierarchical Clustering*)

starts by considering each sample in the data set as an initial cluster. Then, in each round, two nearest clusters are merged as a new cluster, and this process repeats until the number of clusters meets the pre-specified value.

We define the distances of given clusters C_i and C_j in different forms.

Hierarchical Clustering

Minimum distance (single-linkage , “单链接”) :

$$d_{\min}(C_i, C_j) = \min_{x \in C_i, z \in C_j} \text{dist}(x, z)$$

Maximum distance (complete-linkage , “全链接”) :

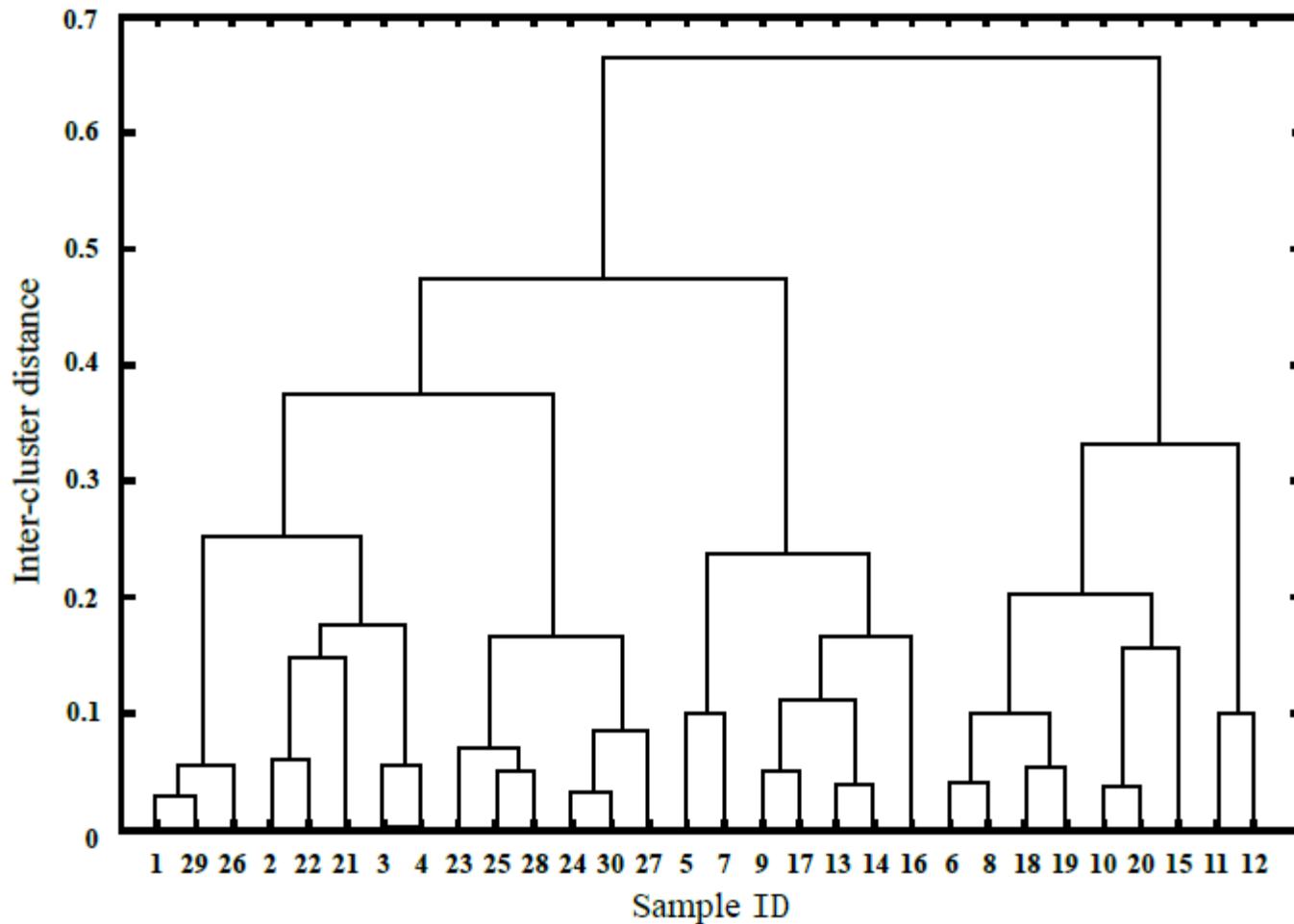
$$d_{\max}(C_i, C_j) = \max_{x \in C_i, z \in C_j} \text{dist}(x, z)$$

Average distance (average-linkage , “均链接”) :

$$d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{z \in C_j} \text{dist}(x, z)$$

Hierarchical Clustering – dendrogram

- The dendrogram (树状图) of AGNES :



Updating Distance Matrix

Let us assume that we have five samples (a, b, c, d, e) and the following matrix of pairwise distances between them:

	a	b	c	d	e
a	0	17	21	31	23
b	17	0	30	34	21
c	21	30	0	28	39
d	31	34	28	0	43
e	23	21	39	43	0

In this example, $D_1(a, b) = 17$ is the lowest value of D_1 so we cluster samples a and b.

Updating Distance Matrix

We then proceed to update the initial distance matrix D_1 into a new matrix D_2 , reduced in size by one row and one column. Let's consider the **single-linkage** clustering:

$$D_2((a,b),c) = \min(D_1(a,c), D_1(b,c)) = \min(21, 30) = 21$$

$$D_2((a,b),d) = \min(D_1(a,d), D_1(b,d)) = \min(31, 34) = 31$$

$$D_2((a,b),e) = \min(D_1(a,e), D_1(b,e)) = \min(23, 21) = 21$$

	(a,b)	c	d	e
(a,b)	0	21	31	21
c	21	0	28	39
d	31	28	0	43
e	21	39	43	0

What if we adopt the complete-linkage clustering?