Ch1. ML Basics

1. Key concept & definitions in ML.

ML Definition:
make computer learn from data. (automation, extract knowledge, predict future)

ML Strategy:
use "data + model" strategy.

ML Pipeline:
model construction ~ training ~ evaluation.

ML ingredient:
Data ~ Experience.
Model ~ a function with some params need to be optimised.
Loss function ~ judge how well the params are set.
Training Algorithm ~ optimise model params. use error function to judge model's performance.

2. Typical ML learning types:

1. ML learning tasks:
Supervised, Unsupervised, Reinforcement Learning.
Classification
Regression
Clustering

2. Supervised learning:
exist a "teacher" provide target output for each data pattern.

training example: < input data pattern, target output>.

typical: classification, regression.

3. Unsupervised Learning:
no explicit "teacher" providing expected output, learn from untagged data.

learn "hidden structure" from unlabelled data.

typical: clustering, generative modeling (estimate distribution)
unsupervised representation learning.

4. Reinforcement Learning.
exist a "teacher" provide reward & punishment.

5. Classification & Regression.
classification: 分类. input data patterns ~ category memberships.
⇒ Binary classification
Multi-class classification
Multi-label classification
Structured classification

regression: 迴歸(R語言). "estimate relationship among input and output vars".
input var ~ continuous output vars.

6. Clustering.
arrange objects into groups based on their similarity.

Ch 2. K-NN Classification

1. K-NN rule for classification.

* core idea: assign datapoint to most common class in its k-nearest neighbor.

* pseudo code:

```
Testing point x_te.
for each training point x_tr:
    measure distance [x_te, x_tr].
sort distances.
pick k-nearest points.
assign x_te to the most common class.
```

* usage: binary classification.

2. K-NN rule for regression:

* core idea: predicted value of x_{pr} = average (sum of its k-nearest neighbors' value)

* pseudo code:

```
Testing point x_te.
for each training point x_tr:
    measure distance [x_te, x_tr].
sort distances.
pick k-nearest points: x_NN1, ..., x_NNk.
output y_te = (1/k) · Σ_{i=1}^{k} y_NNi.
```

* usage: value prediction for discreate points, face prediction.
  (compare Euclidean distance using image pixels)

3. Distance Calculation.

1. Euclidean distance & Minkowski distance:

$$d(p,q) = \left[\sum_{i=1}^{d} |p_i - q_i|^k\right]^{\frac{1}{k}}.$$

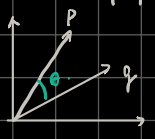
$\Rightarrow$ $k=1$: Manhattan distance
$k=2$: Euclidean distance
$k\geq 3$: Minkowski distance.

2. Similarity Measuring:

cosine 实际上是 similarity.

越大越相似. { i. use inner product.
ii use cosine (p,q) as the measurement of distance: $d(p,q) = 1 - S_{cos}(p,q)$

$$S_{cos}(p,q) = \frac{S_{inner}(p,q)}{\|p\|_2 \cdot \|q\|_2} = \frac{\vec{p}\cdot\vec{q}}{|\vec{p}|\cdot|\vec{q}|}.$$



4. K-NN behavior

training sample size:

- small → insufficient
- large → more information, cost time & space
- noisy → inaccurate

neighbor number K: hyper parameter.

- small: may model noise, inaccurate. (sensitive to noise)
- large: include too many irrelevant info, may negatively affect prediction. (inaccurate prediction)

Ch3. Machine Learning Experiments.

1. Classification Performance Measures

* accuracy & error

classification { accuracy rate: $\frac{\#\text{ correctly classified ones}}{\text{total}}$; error rate: $\frac{\#\text{ wrong classified ones}}{\text{total}}$

can't be used solely as evaluation standard: unreliable for assessing unbalanced data.

* Confusion Matrix

confusion matrix: (error matrix):

TP	FP
TN	FN

行: samples in predicted class
列: sample in actual class.

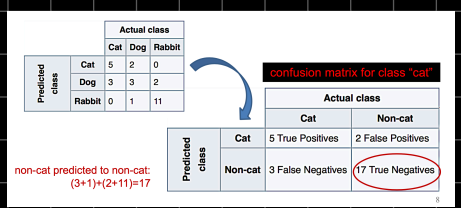
* Important: convert from large confusion matrix to 2×2 confusion matrix for specific class.

* 计算 binary model 的 F_1, precision, sensitivity, ... 时: 务必标对 0 和 1 类.

* TP, FP, TN, FN

TP ~ true positive　　FP ~ false positive

FN ~ false negative　　TN ~ true negative.



* Precision & Recall (sensitivity, true positive rate, TPR)

assess ability to classify positive samples.

$\text{Precision} = \frac{TP}{TP + FP}$　　accuracy on Positive

$\text{recall} = \frac{TP}{TP + FN}$　　how many Ps retrieved from all positive samples

* Specificity & FPR

assess ability to classify negative samples.

$\text{specificity} = \frac{TN}{TN + FP}$　　how many Fs retrieved

FPR (false positive rate): $1 - \text{specificity} = \frac{FP}{TN + FP}$.

* F_1 Score.

F_1 score: $\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

2. Regression Performance Measures

1. RMSE 根均方误差. $\sqrt{\frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{ij} - \hat{y}_{ij})^2}$ (multi)　　$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ (single)

2. MAE 均绝对误差. $\frac{1}{mn}\cdot\sum_{i=1}^{n}\cdot\sum_{j=1}^{m}\cdot|y_{ij} - \hat{y}_{ij}|$ (multi)　　$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$ (single)

y_{ij}: 预测值
$\hat{y}_{ij}$: 理想值
$\bar{y}$: y_{ij} 平均值

3. MAPE 均绝对值比率差. $\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{|y_{ij}-\hat{y}_{ij}|}{|y_{ij}|}$ (multi) $\frac{1}{n}\sum_{i=1}^{n}\frac{|y_i-\hat{y}_i|}{|y_i|}$ (single)

4. R^2: $1-\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$. FOR SINGLE OUTPUT ONLY!

3. Concepts:

1. sample error, true error, bias error, variance error

error s * 样本误差: error computed by a performance metric using a set of data samples.
sample error: always can be calculated:
- infinite samples: 样本误差 converge to 真实误差.
- insufficient samples: 样本误差无法作为真实误差的有效估计.

error D * 真实误差: classifier ~ 误分类概率.
regression ~ 预测值和真实值差的期望.

* Bias: the amount that prediction differs from target value 系统性偏差.
provide optimistically biased estimate of hypothesis over future samples
⇒ Hold out: use new set of test examples independent from training samples.

Bias Error: $(y-E[f])^2$: how much averaged prediction is close to true value.

* Variance: accuracy of new set of test samples vary from true accuracy. 内部差异
⇒ Increase set of test samples: smaller set of test samples may cause high variance.

Variance Error: $E[(f-E[f])^2]$: how much prediction varies among different realisations of model.

2. Under-fitting & Over-fitting.

* Bias-Variance Decomposition:

$$E[(y-f(x))^2] = \underbrace{E[(f(x)-\hat{y})^2]}_{\text{方差 error}} + \underbrace{(y-\hat{y})^2}_{\text{bias error}}, \quad \hat{y}=E[f(x)].$$

variance error: 预测结果内部的离散程度.

bias error: 预测结果整体和真实值的偏差程度.

* Over-fitting: low bias error, high variance error
⇒ over-complex model, sensitive to fluctuations in training set.

* Under-fitting: high bias error, low variance error.
⇒ excessively simple model.

4. Confidence interval & hypothesis comparison (Z-test)

1. 置信区间 confidence interval

Confidence interval tells you 置信区间.

With p probability, the true error lies in the interval of

Confidence level: p	50%	68%	80%	90%	95%	98%	99%
Constant:							

$error_D \in [error_S - a, error_S + a]$. interval around the sample error.

Confidence level: p	50%	68%	80%	90%	95%	98%	99%
Constant: z_p	0.67	1.00	1.28	1.64	1.96	2.33	2.58

given p, compute a:

$a = z_p \cdot \sqrt{\frac{error(1-error)}{n}}$ $\quad error_D \in [error_S - a,\ error_S + a]$.

* confidence interval is an approximate
* work well for over 30 samples ($n > 30$) and sample error not too close to 0 or 1.

2. 假设检验.

Z-test : 适用 ≥ 30 samples.

given: classifier A ~ $error_{S_1}(A)$ find: $error_{S_1}(A) - error_{S_2}(B) > 0$.

classifier B ~ $error_{S_2}(B)$ wonder: $P(error_D(A) > error_D(B))$.

注: 两个 classifier 的 errors 可基于不同的 sample set!

Perform Z-test:

$$z_p = \frac{d}{\sigma} : \begin{cases} d = |error_{S_1}(A) - error_{S_2}(B)| \\ \sigma = \sqrt{\frac{error_{S_1}(A)(1-error_{S_1}(A))}{n_1} + \frac{error_{S_2}(A)(1-error_{S_2}(A))}{n_2}} \end{cases}$$

↓

find the closest P in table

Confidence level: p	50%	68%	80%	90%	95%	98%	99%
Constant: z_p	0.67	1.00	1.28	1.64	1.96	2.33	2.58

$\Rightarrow C = 1 - \frac{1-P}{2}$, finally.

5. Data splitting strategies & usage

Solution:

1. Holdout: dataset ⇒ test set + training set (not suitable for small dataset, error rate estimate may be misleading)
2. Random Subsampling (randomly select x samples for testing, rest for training, repeat k times)
3. K-fold Cross Validation + Leave-one-out. (divide entire set into k portions; use k-1 portion for training, remaining one test)
3. Bootstrap (有放回取样) (randomly pick M (有放回) for training, rest for testing)

ML-Experiment:

1. Model Training
2. Model Evaluation

注: 1. training-testing set 分别独立. (否则 high bias error)

2. 使用合理的 data splitting strategy.

3. Model Selection.

Ch 4. ML Models

1. Basic ML ingredients

i> model : a function that provide answer / solution to given task T.

ii> objective function: O(θ) compute performance P of given task T. (loss function) 目标函数 = 损失函数.

iii> optimisation : min(O(θ)). (training process)

2. Probabilistic & Deterministic approaches for classification & regression

* classification
  - deterministic — thresholding: $\hat{y} = \begin{cases} \text{class 1} & f(x) \geq t \\ \text{class 2} & f(x) < t \end{cases}$ t: treat as hyper parameter. commonly: t=0.
  - probabilistic — Probability Inference: $\hat{y} = \underset{k}{\text{argmax}}\ p(\text{class } k \mid x)$.

model posterior class probability $p(\text{class } k \mid x)$:

1. construct posterior function directly. — logistic regression classifier
2. construct likelihood $p(x \mid \text{class } k)$ and prior $p(\text{class } k)$ functions, then calculate posterior function: — naive Bayes

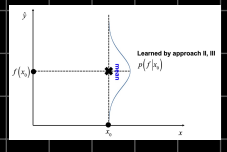
$$p(\text{class } k \mid x) = \frac{p(\text{class } k, x)}{p(x)} = \frac{p(x \mid \text{class } k) \cdot p(\text{class } k)}{p(x)}.$$

* regression
  - deterministic — find mapping function
  - probabilistic — model posterior output probability $p(f \mid x)$.

model posterior output probability $p(f \mid x)$: f: 输出函数.

⇒ Estimate output using condition mean: $\hat{y} = \mathbb{E}_f[f \mid x]$.

在每一个 x 上, 输出函数 (值) 都有一个概率分布, 取 mean 作为模型在该点 x 上的预测值.



1. model $p(f \mid x)$ directly.
2. construct likelihood $p(x \mid f)$ and prior $p(f)$. then calculate posterior:

$$p(f \mid x) = \frac{p(x, f)}{p(x)} = \frac{p(x \mid f) \cdot p(f)}{p(x)}.$$

3. Principle & Usage of linear model (single output + multiple output):

linear model - single output: $\hat{y} = f(x_1, \cdots, x_d)$.

single datapoint $\vec{x}$ (列向量): $\hat{y} = w^T \cdot \vec{x}$.

multiple datapoints $\vec{X}$ (一行行): $\vec{Y} = \vec{X} \cdot w$. Y: 一行行!

linear model - multi output: $[\hat{y}_1, \cdots, \hat{y}_c] = f(x_1, \cdots, x_d)$.

single datapoint $\vec{x}$ (列向量): $\hat{y} = w^T \cdot \vec{x}$.

multiple datapoints $\vec{X}$ (一行行): $\vec{Y} = \vec{X} w$. Y: 一行行!

1. linear model for classification:

- non-probabilistic : separating hyperplane
- probabilistic : logistic regression.

non-probabilistic: decision boundry separation
超平面. 2D~ 线. 3D~ 面. nD~ 超平面.

Binary Classification: Construct a single-output linear model, then construct the discriminant function by thresholding ($t=0$).

$$f = w_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d,$$
$$\hat{y} = \begin{cases} +1, & f \geq 0, \\ -1, & f < 0. \end{cases}$$

- +1 indicates the input is from class 1.
- -1 indicates the input is from class 2.

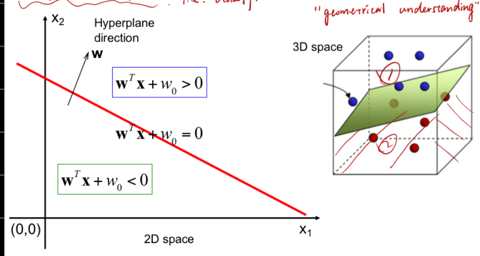
Multiclass Classification: Construct a multi-output linear model, then construct the discriminant function for each class.

$$f_i = w_{0i} + w_{1i}x_1 + w_{2i}x_2 + \ldots + w_{di}x_d,\ i = 1,2,\ldots,c$$
$$\hat{y}_i = \begin{cases} +1, & f_i \geq 0, \\ -1, & f_i < 0. \end{cases}$$

- +1 indicates the input is from class i.
- -1 indicates the input is not from class i.

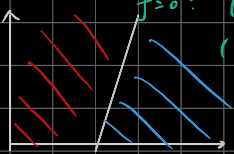
Setting a linear function to zero results in a **hyperplane** that partitions the space to two parts. i.e. classify.

"geometrical understanding"



e.g.

$$f = 3 + 5x_1 - x_2$$
$$\hat{y} = \begin{cases} 1 & f \geq 0 \\ -1 & f < 0 \end{cases}$$



f=0: classification boundary (decision boundary).

create line to separate datapoints into 2 groups.

probabilistic inference for classification: logistic regression

classification using probabilistic inference

compute class posterior using sigmoid (binary) / softmax (multi-class) from linear model

Logistic Regression

⇒ Binary: sigmoid function

Binary Classification:

- Use a single-output linear model.

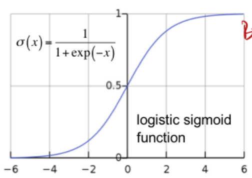
$$f = w_0 + w_1x_1 + w_2x_2 + \cdots + w_dx_d = \mathbf{w}^T\tilde{\mathbf{x}}$$

bias (w_0)

One way: MLE — Need to find the optimal $\boldsymbol{w}$ using the training data.

- Use the logistic sigmoid function to convert a real-valued number to probability.

$\sigma(\mathbf{w}^T\tilde{\mathbf{x}})$



Binary:

$$p(c_1|\mathbf{x}) = \sigma(f) = \frac{1}{1+\exp(-\mathbf{w}^T\tilde{\mathbf{x}})}$$
$$p(c_2|\mathbf{x}) = 1 - \sigma(f) = \frac{1}{1+\exp(\mathbf{w}^T\tilde{\mathbf{x}})}$$

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

⇒ multinomial : softmax function

Multi-class Classification:

- Use a multi-output linear model.

$$f_1 = w_{01} + w_{11}x_1 + w_{21}x_2 + \ldots + w_{d1}x_d = \mathbf{w}_1^T\tilde{\mathbf{x}}$$
$$f_2 = w_{02} + w_{12}x_1 + w_{22}x_2 + \ldots + w_{d2}x_d = \mathbf{w}_2^T\tilde{\mathbf{x}}$$
$$\vdots$$
$$f_c = w_{0c} + w_{1c}x_1 + w_{2c}x_2 + \ldots + w_{dc}x_d = \mathbf{w}_c^T\tilde{\mathbf{x}}$$

Need to find the optimal $\boldsymbol{w}_i$ using the training data.

$$P(C_k|x) = \frac{e^{w_k^T x}}{e^{w_1^T x} + e^{w_2^T x} + \cdots + e^{w_c^T x}}$$

$$= \frac{e^{w_k^T x}}{\sum_{j=1}^{c} e^{w_j^T x}}$$

← 将线性组合转为正数
← 归一化到[0,1].

– Use the **softmax function** to convert outputs to probabilities.

$$p(c_k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \tilde{\mathbf{x}})}{\sum_{j=1}^{c} \exp(\mathbf{w}_j^T \tilde{\mathbf{x}})}, k = 1, 2, \ldots c$$

$$\text{Softmax}(a_1, a_2, \cdots, a_K) = \left[\frac{e^{a_1}}{\sum_{i=1}^{k} e^{a_i}}, \frac{e^{a_2}}{\sum_{i=1}^{k} e^{a_i}}, \cdots, \frac{e^{a_k}}{\sum_{i=1}^{k} e^{a_i}} \right].$$

$k = 2$ 时: softmax 就是 logistic sigmoid function.

3. linear model for regression: (only care about single output)

non-probabilistic:

$f = w_0 + w_1 x_1 + \cdots + w_d x_d = w^T \tilde{x}$. □·□ = □

⇒ directly use output as predicted value.

probabilistic:

need to estimate $p(f|x)$.

⇒ assume gaussian distribution: $p(f|x) = N(w^T x, \delta^2)$

then $E_{f|x}[f|x]$.

⇒ 等价于: $y = w^T x + \varepsilon$. 加 x gaussian noise: 均值 0. 方差 δ^2.

4. Principle & Use of linear basis function model

linear basis function model : 非线性可分的低维空间 $\xrightarrow{\varphi}$ 线性可分的高维空间

directly formulate many non-linear functions (basis function) : $\{\varphi_i(x)\}_{i=1}^{D}$

they work as "feature extractor".

$$\hat{y} = w_0 + w_1 \varphi_1(x) + w_2 \varphi_2(x) + \cdots + w_D \varphi_D(x) = w^T \varphi(x).$$

5. Core idea of kernel method + inner product in kernel space.

kernel method : 用核函数直接计算用 φ 映射所形成的高维空间中向量内积.

Kernel Name	Expression K(x,y)	Comments
Linear	$\mathbf{x}^T\mathbf{y}$	No parameter
Polynomial	$(\mathbf{x}^T\mathbf{y}+1)^p$	p is a user defined parameter
Gaussian, also called radial basis function (RBF)	$\exp\left(-\|\mathbf{x}-\mathbf{y}\|_2^2/(2\sigma^2)\right)$	σ is the user defined width parameter
Hyperbolic tangent	$\tanh(\alpha\mathbf{x}^T\mathbf{y}+\beta)$	α & β are user defined parameters

$$\varphi(x_i)^T \cdot \varphi(x_j) = k(x_i, x_j).$$

原来: $f(x) = w_0 + w_1 x_1 + \cdots + w_K x_K$.

↓ 核函数所在的特征空间

$f(x) = w_0 + w_1 \varphi(x_1) + \cdots + w_K \cdot \varphi(x_K)$.

(w_0 是 bias).

不知道 $\varphi(x)$. 但知前向 weight: $w_1, \cdots, w_K$ 可表为新空间 φ 内一组是 $\varphi(x_1), \cdots, \varphi(x_K)$ 的线性组合.

⇒ 从而预测函数就可用核函数表示.

$$f(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + w_0$$

$$= \left(\sum_{i=1}^{n} a_i \boldsymbol{\phi}(\mathbf{x}_i) \right)^T \boldsymbol{\phi}(\mathbf{x}) + w_0$$

$$= \sum_{i=1}^{n} a_i \boldsymbol{\phi}^T(\mathbf{x}_i) \boldsymbol{\phi}(\mathbf{x}) + w_0$$

$$= \sum_{i=1}^{n} a_i k(\mathbf{x}_i, \mathbf{x}) + w_0$$

- Training becomes to find the best combination coefficients ***a***.

$$\mathbf{a} = [a_1, a_2, \ldots a_n]^T$$

- This looks similar to a linear basis function model, where *f* is a linear combination of many nonlinear basis functions instead.

Ch 5. Loss function.

1. calculation of SSE, MSE, Hinge loss 和 cross entropy loss.

Recall RMSE root mean square error

$$RMSE = \sqrt{\frac{1}{mn} \cdot \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} - \hat{y}_{ij})^2}$$

simplified ↓

Training Data: $D_{tr} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$

Feature vector: $\mathbf{x}_i \in R^d$

Target output: $\mathbf{y}_i \in R^c$ where $\mathbf{y}_i = [y_{i1}, y_{i2}, \dots y_{ic}]$

SSE: sum-of-squares error loss

$$SSE = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{c} (\hat{y}_{ij} - y_{ij})^2.$$

y_i / y_{ij}: true value

$\hat{y}_{ij}$ / $\hat{y}_i$: predicted value.

MSE: mean-square-error.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2.$$

Hinge loss:

$$O(\theta) = \sum_{i=1}^{N} \max(0, 1 - y_i \cdot f(\theta, x_i))$$

Hinge loss assesses classification error.

Given **+1/-1 label coding**, hinge loss over N training samples is

$$O(\theta) = \sum_{i=1}^{N} \max\left(0, 1 - y_i f(\theta, x_i)\right)$$

class label $y_i \in \{-1, +1\}$

your prediction model: $f(\theta, x_i)$

Hinge loss classification (用 Hinge loss 作 objective fun)

Let *f* be a linear model and add a regularisation term to the loss. This results in **support vector machine**.

$$\min_{(\mathbf{w}, w_0) \in \Re^{d+1}} C \sum_{i=1}^{N} \max\left(0, 1 - y_i\left(\mathbf{w}^T \mathbf{x}_i + w_0\right)\right) + \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

- C: regularisation parameter (hyper-parameter)
- $\sum_{i=1}^{N} \max(\dots)$: hinge loss
- $\frac{1}{2}\mathbf{w}^T\mathbf{w}$: regularisation term

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Cross Entropy loss:

Cross entropy loss computed over N training samples is

$$O = -\sum_{i=1}^{N} \left[y_i \log_b\left(p(c_1 | \mathbf{x}_i)\right) + (1 - y_i) \log_b\left(p(c_2 | \mathbf{x}_i)\right) \right]$$

does b really counts?

You can use natural log (*ln*, *b*=*e*) or log base 2 (*b*=2).

Cross entropy loss computed over N training samples is

$$O = -\sum_{i=1}^{N} \sum_{k=1}^{c} y_{ik} \log_b\left(p(c_k | \mathbf{x}_i)\right)$$

(each class ↓ over $\sum_{k=1}^{c}$; ↑ for each sample over $\sum_{i=1}^{N}$.)

Cross entropy measures distance between probability distributions.

Its discrete version can be used to examine the **distance** between the predicted class probabilities (**posterior**) and the **true probabilities**.

$$H(p, q) = -\left[p(1)\log(q(1)) + p(0)\log(q(0))\right] \text{ (two classes)}$$

$$H(p, q) = -\left[p(1)\log(q(1)) + p(2)\log(q(2)) + \dots + p(c)\log(q(c))\right] \text{ (multiple classes)}$$

2. Basic idea of (log) likelihood, MLE (maximum likelihood estimator):

1. 似然: likelihood, log likelihood.

- **Likelihood:** Given the observed data, it is the conditional probability assumed for the observed data given some parameter values.

  假设：参数选择θ应使"在当前的参数选择下，给定数据集的分类结果和训练集中结果一致的可能性最大.

  $$\text{Likelihood}(\theta | \text{data}) = p(\text{data} | \theta)$$

- **Log likelihood**: Take the natural logarithm of the likelihood.

  自然对数

likelihood: $p(x | f)$, $p(x | \text{class}_k)$.

回忆: "计算 posterior probability using prior, likelihood + Bayes Thm.

2. MLE (maximum likelihood estimator):

A model can be trained by **maximising** the likelihood (or log likelihood) function of the training samples.

Assume independence between samples.

- **Maximum likelihood estimator** (MLE):

$$\max_{\theta} = \prod_{i=1}^{N} p(\mathbf{x}_i, y_i \mid \theta)$$

ideal prediction results; disappoint; one; possible selection

- **Log likelihood maximisation**:

$$\max_{\theta} = \sum_{i=1}^{N} \log p(\mathbf{x}_i, y_i \mid \theta)$$

Likelihood of N training samples:

$$L = \prod_{i=1}^{N} N\left(y_i \middle| \mathbf{w}^T \tilde{\mathbf{x}}_i, \sigma^2\right)$$

Assume a Gaussian distribution for likelihood estimation.

$$p(\text{data}_i \mid \theta) = p(y_i \mid \theta) = N\left(y_i \middle| \mathbf{w}^T \tilde{\mathbf{x}}_i, \sigma^2\right)$$

Log-likelihood:

$$O = \ln(L) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta\frac{1}{2}\sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\tilde{\mathbf{x}}_i\right)^2$$

(essentially SSE ⇔ MLE)

Sum-of-squares error function!

where $\beta^{-1} = \sigma^2$

In this case, an MLE is equivalent to a sum-of-squares error minimizer.

3. Concept of Regularisation & Formation of regularison term:

- A regularisation term can be added to the error function. For instance, in the single-output case, we have

$$\min O_\lambda(\mathbf{w}) = \text{sum of suqares error} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\tilde{\mathbf{x}}_i\right)^2 + \boxed{\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}}$$

$\mathbf{w}^T\mathbf{w} = \sum_{j=1}^{d+1} w_j^2$

λ is a positive real-valued number set by the user.

- Other type of regulariser:

$$\min O_\lambda(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\tilde{\mathbf{x}}_i\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d+1}|w_j|^{\boxed{q}}$$

Here q is a positive integer set by the user.
- ❖ The case of $q=1$ (l_1-regularisation) for regression is known as lasso.
- ❖ The case of $q=2$ (l_2-regularisation) for regression is known as ridge regression.

- Regularisation prevents the model from over-fitting to training data.
- When λ is too large, it will lead to under-fitting though.

lasso, l_1-regularisation: $\frac{\lambda}{2}\sum_{j=1}^{d+1}|w_j|^1 = \frac{\lambda}{2}|w|$

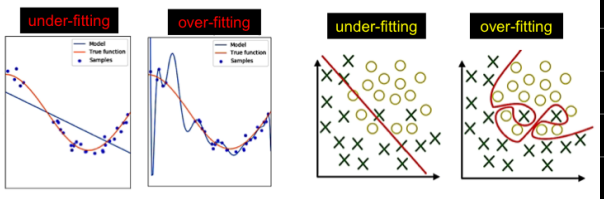
ridge, l_2-regularisation: $\frac{\lambda}{2}\sum_{j=1}^{d+1}|w_j|^2 = \frac{\lambda}{2}w_j^2 = \frac{\lambda}{2}w^Tw$.

general: l_K-regularisation: $\frac{\lambda}{2}\sum_{j=1}^{d+1}|w_j|^K$.

作用：防止模型过拟合。λ太大时会欠拟合。

过拟合
- Over-fitting: Fit too closely to a particular set of data (e.g., training data), and may therefore fail to fit new data.

欠拟合
- Under-fitting: Cannot capture the underlying trend of the training data.
- Prevent over-fitting, e.g., $O_\lambda(\mathbf{w})$ gives less emphasis to the sum of squares error of the training data.



4. Principle & Usage of (regularised) least square model for classification / regression.

linear least squares (LLS): train a model by minimising sum-of-squares error

- single: $\min O(w) = \frac{1}{2}\sum_{i=1}^{N}(y_i - w^T\tilde{x}_i)^2$.
- multiple: $\min O(w) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{C}(y_{ij} - w_j^T\tilde{x}_i)^2$.

↑ regularisation term (l_2-regularisation)

⇒

- A regularisation term can be added to the error function. For instance, in the single-output case, we have

$$\min O_\lambda(\mathbf{w}) = \text{sum of suqares error} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\tilde{\mathbf{x}}_i\right)^2 + \boxed{\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}}$$

$\mathbf{w}^T\mathbf{w} = \sum_{j=1}^{d+1} w_j^2$

λ is a positive real-valued number set by the user.

- Other type of regulariser:

$$\min O_\lambda(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\tilde{\mathbf{x}}_i\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d+1}|w_j|^{\boxed{q}}$$

Here q is a positive integer set by the user.
- ❖ The case of $q=1$ (l_1-regularisation) for regression is known as lasso.
- ❖ The case of $q=2$ (l_2-regularisation) for regression is known as ridge regression.

Likelihood of An Individual Sample

The University of Manchester

Consider the *i*-th training sample $(\mathbf{x}_i, y_i)$

- **Binary classification**:

$$p(\mathbf{x}_i, y_i | \theta) = \theta(\mathbf{x}_i)^{y_i}\left(1 - \theta(\mathbf{x}_i)\right)^{1-y_i}$$

- **Multi-class classification**: Assume the class label follows categorical (multinomial) distribution (generalise Bernoulli distribution to more than two options):

$$p\left(\mathbf{x}_i, \mathbf{y}_i | \theta\right) = \prod_{k=1}^{c} \theta_k(\mathbf{x}_i)^{y_{ik}}$$

Given N training samples and assume sample independence.

- **Binary classification**:

$$L = \prod_{i=1}^{N} p(\mathbf{x}_i, y_i | \theta) = \prod_{i=1}^{N} \theta(\mathbf{x}_i)^{y_i}\left(1 - \theta(\mathbf{x}_i)\right)^{1-y_i}$$

- **Multi-class classification**:

$$L = \prod_{i=1}^{N} p\left(\mathbf{x}_i, \mathbf{y}_i | \theta\right) = \prod_{i=1}^{N}\prod_{k=1}^{c} \theta_k(\mathbf{x}_i)^{y_{ik}}$$

Assumption:

1. Binary ⇒ class label follow Bernoulli distribution
2. Multi-class ⇒ class labels follow multinomial distribution

为一个基于 likelihood 函数的损失函数.

Negative Log likelihood Loss

The University of Manche

$$L = \prod_{i=1}^{N}\prod_{k=1}^{c} \theta_k(X_i)^{y_{ik}}$$

- Classification loss: negative log likelihood.
- Negative log likelihood loss is equal to the cross-entropy loss, if

θ 建模: 可用 sigmoid / softmax 等.

$$\theta\left(\mathbf{x}\right) = p\left(c_1 | \mathbf{x}\right)$$

$$\theta_k\left(\mathbf{x}\right) = p\left(c_k | \mathbf{x}\right)$$

Ch 6. Training & Optimisation.

1. Principle & Usage of normal equation

损失函数: $O(\theta_1, \cdots, \theta_m)$.

损失函数优化: $\min O(\theta_1, \cdots, \theta_m) \Rightarrow$ 目标: $\frac{\partial O}{\partial \theta_1} = \frac{\partial O}{\partial \theta_2} = \cdots = \frac{\partial O}{\partial \theta_m} = 0$.

Example. 使用 SSE (sum-of-square error) 作为损失函数:

single input:
1. linear model $\hat{y}_i = w^T \cdot \tilde{x}_i$
2. loss function $O(w) = \frac{1}{2} \| \tilde{X} \cdot w - y \|_2^2$. (sum of square error loss / objective function)

multiple input:
1. $\hat{y}_i = w^T \tilde{x}_i$
2. $O(w) = \frac{1}{2} \| \tilde{X} \cdot w - Y \|_2^2$.

normal equation 由 linear least square 引入:

linear least square 损失函数核心形如 "$\tilde{X} \cdot w - Y$" 或 "$\tilde{X} \cdot w - y$". ($\tilde{X}$: expanded feature matrix). 要考虑 bias 计算, 加一列 1.

$\Rightarrow$ 损失函数最小: 求使得 $\tilde{X} \cdot w - Y$ 为 0 的解 w.

$X: N \times d$

$\Rightarrow$ 利用 normal equation (datapoint 数多于 feature 的情形, 即 $N > d$. 常见.)

normal equation's principle:

$O(w) = \frac{1}{2} \| \tilde{X} \cdot w - Y \|_2^2$.

目标: 让 $\nabla O(w) = 0$: $w = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T Y$.

$\nabla O(w) = \tilde{X}^T \tilde{X} w - \tilde{X}^T Y$.

normal equation for least square problem.

normal equation 的形式:

1. for linear least square model:

$N > d$: $w = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$ (single output)

over-determined

$w = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T Y$ (multiple output)

> The quantity $(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T$ and $\tilde{\mathbf{X}}^T (\tilde{\mathbf{X}} \tilde{\mathbf{X}}^T)^{-1}$ are called Moore-Penrose inverse (or called pseudoinverse) of $\tilde{\mathbf{X}}$.

称 $(\tilde{X}^T \tilde{X})^{-1} \tilde{X}$ 和 $\tilde{X}^T (\tilde{X} \tilde{X}^T)^{-1}$ 为 pseudo matrix.

适用于任何 X. 即使 X 为 singular 也行.

$N < d$: $w = \tilde{X}^T (\tilde{X} \tilde{X}^T)^{-1} y$ (single output)

under-determined

$w = \tilde{X}^T (\tilde{X} \tilde{X}^T)^{-1} Y$ (multiple output)

2. for l_2-regularisation linear least square model: (只看 $N > d$ 情形)

$w = (\tilde{X}^T \tilde{X} + \lambda I)^{-1} \tilde{X}^T Y$.

> Hyperparameter: $\lambda > 0$
> - It controls the balance between the data dependent error function and the regularisation term.
> - To distinguish it from those model parameters to be optimised during the learning process (e.g., $\boldsymbol{W}$), λ is referred to as a hyperparameter.
>
> Training, validation and testing:
> - Training set: A set of samples used for learning, that is to optimise the model parameters.
> - Validation set: A set of samples used to select (tune) the hyperparameters.
> - Test set: A set of samples used only to assess the performance of a fully-specified classifier (trained with selected hyperparameters).

λ 为超参数: control balance between data-dependent error function and regularisation term.

λ 的选择: 用 cross-validation 来获得.

用法: 依 normal equation 构造 optimal 权重的计算式并求解. 注意 train set. validation set. test set.

2. Derivation of least square model's solution

计算方法：1. 直接从所得的 objective function 中 取每个权重值的对应偏导.
让 所有偏导数 $\nabla O(w_0), \nabla O(w_1), \cdots, \nabla O(w_n)$ 全为0，计算出相应的权重值.

2. 利用 normal equation 计算 $W = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T Y$. (或 $W = \tilde{X}^T(\tilde{X}\tilde{X}^T)^{-1}Y$).

3. Principle of Gradient Descendent

梯度下降法：目标仍为最小化 loss function (objective function).

原理：损失函数视为一个关于权重 W 的函数.

随机猜一个 initial weight. 由 $O(w)$ 关于权重各分量的偏导(梯度)决定 W 各个分量的优化方向. 最终目标仍为：让 $\nabla O(w) = 0$，最小化 $O(w)$.

分量的优化方式：$w^{(n+1)} = w^{(n)} - \eta \cdot \nabla O(w^{(n)})$.

学习率 learning rate：超参数. η 小 $\Rightarrow$ "学的慢". η 大 $\Rightarrow$ "学过头".

梯度下降法变体：

Recall the error function computed using all the training samples: $O(\mathbf{w}) = \sum_{i=1}^{N} O_i(\mathbf{w})$

- Gradient descent (GD) computes the gradient of the error function using all the training samples.

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla O(\mathbf{w}^{(t)}) = \mathbf{w}^{(t)} - \eta \sum_{i=1}^{N} \nabla O_i(\mathbf{w}^{(t)})$$

Update using all the N training samples.

- Stochastic gradient descent (SGD) estimates the gradient of the error function using one training sample.

pick i

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla O_i(\mathbf{w}^{(t)})$$

Update using only one training sample.

- Mini-batch gradient descent (MBGD) estimates the gradient of the error function using a small set of training samples.

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \sum_{n \in I} \nabla O_i(\mathbf{w}^{(t)})$$

I denotes a small set of training samples.

Update using a subset of training samples.

GD：所有全算.

SGD：随机算一个.

Mini-Batch：算一小批.

4. Derive weight updating equation for least squares model:

构造损失函数 $O(w)$ ~ 对 $O(w)$ 关于 w 每个分量求偏导.

Ch. 7. Artificial Neural Network

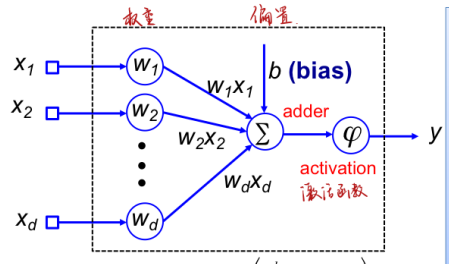
1. Single neuron model (adder & activation) → single layer perceptron.

人工神经网络：对生物神经元的模拟.

1. Single neuron model:

adder:

- Multiple inputs $[x_1, x_2, \ldots, x_d]$ and one output y.



Basic elements of a typical neuron include:

- A set of **synapses or connections**. Each of these is characterised by a weight (strength).
- An **adder** for summing the input signals, weighted by the respective synapses.
- An **activation function**, which squashes the permissible amplitude range of the output signal.

neuron $y = \varphi\left(\sum_{i=1}^{d} w_i x_i + b\right)$ (Δ_{bias})

Given d input, a neuron is modeled by $d+1$ parameters.

上层连接

adder (和权重线性组合).

激活函数.

# of params: $n+1$. # of weights: n.

input: $[x_1, \cdots, x_d]$. output: $y = \varphi(w_1x_1 + \cdots + w_dx_d + b)$. 类似于多input, 单output的linear model.

常用激活函数:

activation:

- Identify function: $\varphi(v) = v$
- Threshold function:

$$\varphi(v) = \begin{cases} 1 & if \quad v \geq 0 \\ -1 & if \quad v < 0 \end{cases}$$

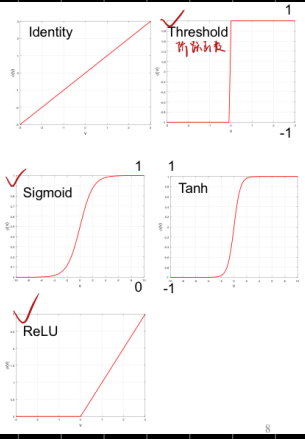
- Sigmoid function ("S"-shaped curve):

$$\varphi(v) = \frac{1}{1+\exp(-v)} \in (0,1) \quad or$$

$$\varphi(v) = \tanh(v) = \frac{\exp(2v)-1}{\exp(2v)+1} \in (-1,+1)$$

- Rectified linear unit (ReLU):

$$\varphi(v) = \begin{cases} v & if \quad v \geq 0 \\ 0 & if \quad v < 0 \end{cases}$$



注: 对 regression problem: 连续输出, 用 identity (linear) activation function.

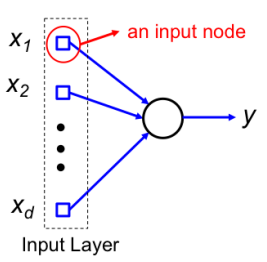
对 binary classification: 概率模型, 用 sigmoid activation function 表示类别概率.

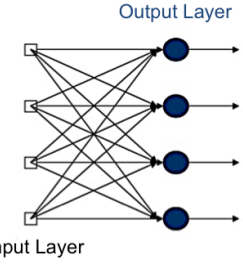
对 computer vision: ReLU 好用.

2. Single Layer Perceptron:

1个输入数任意的 input layer 和 1个输出数任意的 output layer.

A single layer perceptron has one input layer and one output layer.





An SLP is similar to a multi-input multi-output linear model.

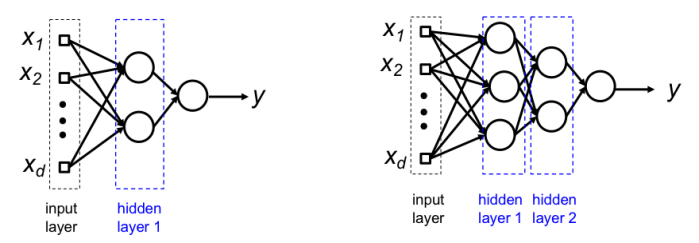
注: 输出数任意!

类似于 multi-input, multi-output 的线性模型.

3. multi-layer perceptron → feed-forward neural network

network calculations given weights & input; calculate # of parameters given layer # and layer size

Hidden Layer: 介于 input layer 和 output layer 间的隐藏层.



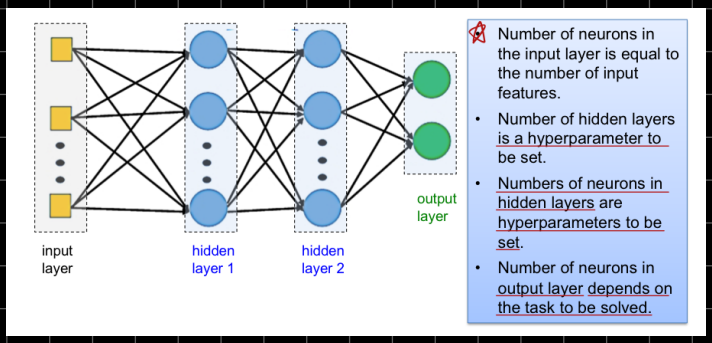
formulate complex, non-linear functions.

each hidden layer find a partial solution to the problem, combine in next layer.

Multi-layer Perceptron (MLP).

多层感知机，又称前馈神经网络 (feed-forward artificial neural network).

含至少三层节点：输入～隐藏层～输出.

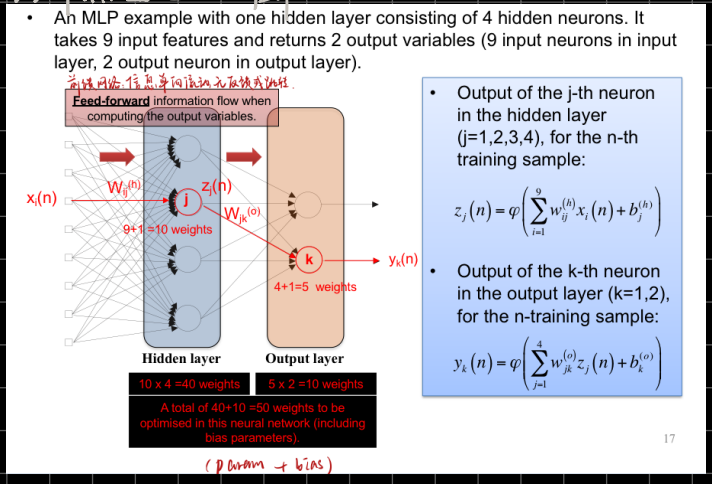


input 层节点数 = 输入 feature 数

层数和每层结构 (节点数): hyper parameter.

output 层节点数：看具体问题而定.

网络节点上值的计算：



(param + bias)

计算值～将线性组合 $w_1x_1+\cdots+w_nx_n+b$ 代入激活函数 $\varphi(x)$.

参数计算～每层上每个节点参数量为 n+1.
(即: # of weight + # of bias)

3. Principle & Usage of Perceptron Algorithm

Perceptron Algorithm:

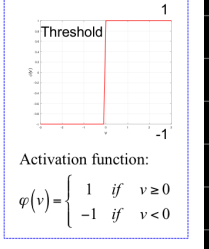
- When the activation function is set as the threshold function, the model is still linear, and it is known as the perceptron of Rosenblatt.

a psychologist

single neuron model:

$$\text{output} = \begin{cases} 1 & \text{if } \mathbf{w}^T\tilde{\mathbf{x}} \geq 0 \\ -1 & \text{if } \mathbf{w}^T\tilde{\mathbf{x}} < 0 \end{cases}$$

for binary classification



- The perceptron algorithm was invented in 1958 by Frank Rosenblatt (1928-1971, Psychologist).

使用 threshold function 作为激活函数的人工神经元.

Application: Binary classification.

updating weight:

1. 只用误分类的 sample!

2. 更新规则:

$$w^{(t+1)} = w^{(t)} + \eta \cdot y_i \tilde{x}_i$$

目标：最小化 perceptron criterion:

不仅直接使用"错误分类的样本总数"为目标函数，因它对关于w的分段常函数，无法使用梯度下降优化。

- **Perceptron criterion** assesses classification error. （另一个目标函数）

$$O(\mathbf{w}) = -\sum_{i \in \text{Misclassified Set}} y_i \left(\mathbf{w}^T \tilde{\mathbf{x}}_i\right)$$

If a sample is correctly classified, applies an error penalty of zero; if incorrectly classified, applies an error penalty of the following quantity:

$$\left| y_i \left(\mathbf{w}^T \tilde{\mathbf{x}}_i\right) \right| = -y_i \left(\mathbf{w}^T \tilde{\mathbf{x}}_i\right)$$

正确分类=0误差
错误分类=$-y_i \cdot w^T x_i$

亦可用 SGD 优化。（$O(w) = -y_i w^T x_i$, $\nabla O(w) = -y_i x_i$）

- Stochastic gradient descent (SGD) is used to minimise the perceptron criterion.
- Calculate the error using one **misclassified** sample:

$$O_i(\mathbf{w}) = -y_i \mathbf{w}^T \tilde{\mathbf{x}}_i$$

⇒ SGD 实际上 cover 了 perceptron algorithm.

- Calculate its gradient: $\frac{\partial O_i(\mathbf{w})}{\partial \mathbf{w}} = -y_i \tilde{\mathbf{x}}_i$
- SGD update: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta y_i \tilde{\mathbf{x}}_i$ y_i: true label

$w^{T+1} = w^T - \eta \cdot \nabla O(w) = w^T + \eta \cdot y_i \tilde{x}_i$

4. Training Method for neural network:

- Hebbian Learning Rule "用进废退"
- Gradient Based Training

Hebbian Learning Rule: "ancient traing rule"

1. start with random value: w_{ij} = random (...)
2. each iteration update a bit: $w_{ij} += \Delta w_{ij}$.

To find suitable value of w_{ij}
w_{ij} connects i and j

- The weight change is formulated as

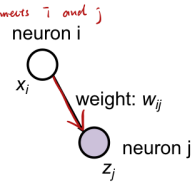
$$\Delta w_{ij} = F(x_i, z_j)$$

simplest form

- The simplest form of the function F

$$\Delta w_{ij} = \eta x_i z_j$$

think: "activated same ⇒ $\Delta w_{ij} > 0$"
"activated asynchronously ⇒ $\Delta w_{ij} \leq 0$".
problem: what is η?



For this neuron:
pre-synaptic signal: x_i
post synaptic signal: z_j

- Can be unstable.

⇒ usually used in biological-inspired neural network

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} + \Delta w_{ij}^{(n+1)}$$
$$\Delta w_{ij} = \eta \cdot x_i \cdot z_j$$

神经元 i 当前值　　神经元 j 当前值.

can be unstable.

Gradient-based Training: (most commonly used)

A commonly used approach:

- Treat $\boldsymbol{z}=\boldsymbol{\varphi}(\boldsymbol{x}, \boldsymbol{W}_{NN})$ as the new features, to be used as the input of a linear predictor.
- A loss function is minimised to find the optimal neural network weights $\boldsymbol{W}_{NN}$ together with the weights of the linear predictor $\boldsymbol{W}_P$.
- Training (optimisation) methods: stochastic gradient descent, mini-batch gradient descent.

Proved in Real World

- More accurate and stable than Hebbian learning rules in practice.

5. Basic idea of Back Propogation.

误差反向传播法：实际上就是链式求导法则，从后往前利用各层节点值的计算函数，算出每一个节点的偏导（更新方向）。

- It uses the chain rule to iteratively compute the gradient for each layer.

Given $z(y(x))$, chain rule: $\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$ 链式法则：计算复合函数偏导数

计算例子：见 Exercise 例题。

- It can be viewed as a process of calculating the error contribution of each neuron after processing a batch of training data.

prevent over-fitting: when training data isn't sufficient

- After choosing a loss function ***Loss*(W_{NN})**, a regularisation term, e.g., the sum of the all the squared weights, is added to the final optimisation objective function for training:

$$O(\mathbf{W}_{NN}) = loss(\mathbf{W}_{NN}) + \underbrace{\lambda \frac{1}{2} \|\mathbf{W}_{NN}\|_2^2}_{\text{regularisation term}}$$

← distance from optimizing loss function process is from fitting too much

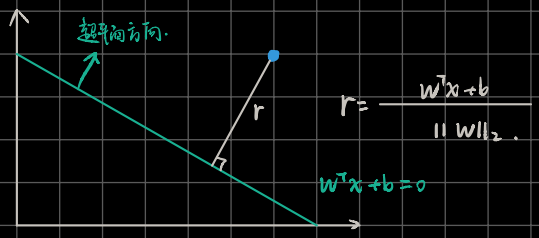
- This is called l_2-regularisation. The regularisation term plays the same role as in the regularised least squares model.

- The regularisation parameter $\lambda \geq 0$ is a hyper-parameter.

Ch 8. Support Vector Machine

1. Key concepts & basic idea of SVM: (separation margin, support vector, slack variables).

1. 背景知识：超平面和点间距离。

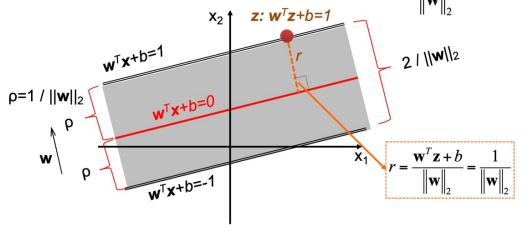


2. SVM：找两个超平面：$w^T x + b = \pm 1$.

- We focus on two parallel hyperplanes:

$$\begin{cases} \mathbf{w}^T\mathbf{x} + b = 1 \\ \mathbf{w}^T\mathbf{x} + b = -1 \end{cases}$$

- Geometrically, distance between these two planes is $\frac{2}{\|\mathbf{w}\|_2}$



图标：对线性可分的 datapoints，通过最小化 weights，最大化 hard separation margin.

- Given two parallel hyperplanes below, we separate two classes of data points by preventing the data points from falling into the margin:

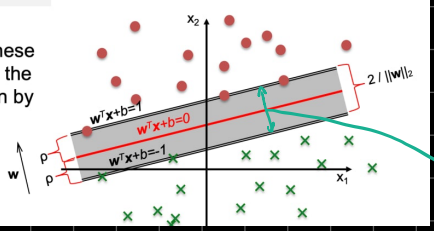
$$\begin{cases} \mathbf{w}^T\mathbf{x} + b \geq 1, & \text{if } y = 1, \\ \mathbf{w}^T\mathbf{x} + b \leq -1, & \text{if } y = -1. \end{cases} \quad \Longleftrightarrow \quad y\left(\mathbf{w}^T\mathbf{x} + b\right) \geq 1$$

equivalent expression

"find an optimal hyperplane to separate the two classes of datapoints with widest margin".

- The region bounded by these two hyperplanes is called the separation "margin", given by

$$\rho = \frac{2}{\|\mathbf{w}\|_2} = \frac{2}{\sqrt{\mathbf{w}^T\mathbf{w}}}$$

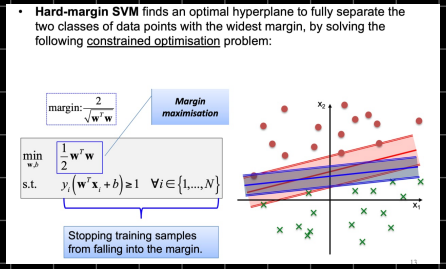


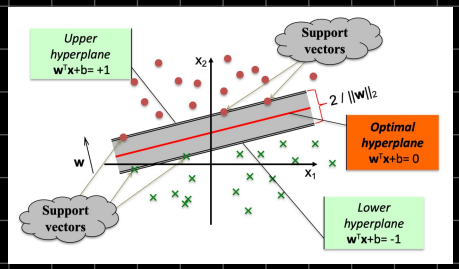
Separation Margin

Hard - Margin SVM:

find optimal hyperplane to fully separate two class of datapoints with widest margin.

- **Hard-margin SVM** finds an optimal hyperplane to fully separate the two classes of data points with the widest margin, by solving the following constrained optimisation problem:





Support Vector:
$y_i(w^T x_i + b) = 1$
(for hard margin SVM)

Capable of solving linear separable data patterns.

Soft - margin SVM

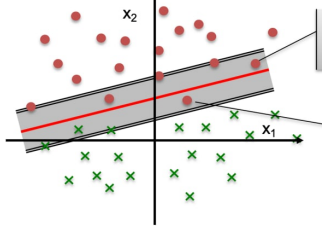
Introduced slack variable $\xi_i \geq 0$, $i = 1, 2, \ldots, N$, in order to relax hard-margin SVM's constraint.

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i, & \text{if } y_i = 1, \\ \mathbf{w}^T\mathbf{x}_i + b \leq -\left(1 - \xi_i\right), & \text{if } y_i = -1. \end{cases} \quad \Longleftrightarrow \quad y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i$$

equivalent expression

ξ_i：第 i 个数据点可以偏离 separation margin 的程度.

We don't push all the points to stay outside the margin any more.



Point within region of separation, but still in the right side: $0<\xi_i\leq 1$

Point in the wrong side of the decision boundary: $\xi_i>1$

Kernel SVM: 使用核函数将 SVM dual problem 映射到另一空间上，在那里问题线性可分.

The SVM dual problem in the original space:

Dual problem

$$\max_{\lambda\in\mathbb{R}^N} \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$
$$\text{s.t.} \sum_{i=1}^{N}\lambda_i y_i = 0,\quad 0\leq\lambda_i\leq C$$

Move from the original space to a kernel induced space.

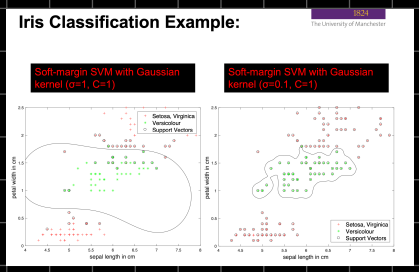
Kernel SVM with the modified dual problem:

Dual problem

$$\max_{\lambda\in\mathbb{R}^N} \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \boldsymbol{\phi}(\mathbf{x}_i)^T\boldsymbol{\phi}(\mathbf{x}_j)$$
$$\text{s.t.} \sum_{i=1}^{N}\lambda_i y_i = 0,\quad 0\leq\lambda_i\leq C$$

$$\max_{\lambda\in\mathbb{R}^N} \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$$
$$\text{s.t.} \sum_{i=1}^{N}\lambda_i y_i = 0,\quad 0\leq\lambda_i\leq C$$

Iris Classification Example:



适用数据非线性的情况.

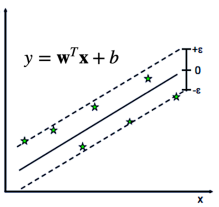
3. Support Vector Regression:

One simple way to formulate a linear SVM regressor:

$$\min_{(\mathbf{w},b)\in R^{d+1}} \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{subject to } \left|\mathbf{w}^T\mathbf{x}_i + b - y_i\right| \leq \epsilon, i = 1,2,\ldots,N$$

The parameter ϵ is a hyperparameter.



It can be extended to fit nonlinear data patterns by using kernel trick.

目标：让超平面 $w^Tx+b=0$ 尽可能多地接近现有 datapoints.

4. SVM Training:

Hard-Margin SVM

Hard-margin SVM training: the process of solving the following constrained optimisation problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{s.t.} \quad y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\geq 1 \quad \forall i\in\{1,\ldots,N\}$$

How to derive the dual form can be found in the SVM notes as optional reading materials.

The above problem is solved by solving a dual problem as shown below.

Dual problem

$$\max_{\lambda\in\mathbb{R}^N} \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$
$$\text{s.t.} \sum_{i=1}^{N}\lambda_i y_i = 0,\quad \lambda_i\geq 0$$

Soft-margin SVM

In addition to maximising the margin as before, we need to keep all slacks ξ_i as small as possible to minimise the classification errors. The modified SVM optimisation problem becomes:

$$\min_{(\mathbf{w},b)\in\mathbb{R}^{N+1},\{\xi_i\}_{i=1}^N} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i$$
$$\text{s.t.} \left.\begin{array}{l} y_i\left(\mathbf{w}^T\mathbf{x}+b\right)\geq 1-\xi_i \\ \xi_i\geq 0 \end{array}\right\} \forall i\in\{1,\ldots,N\}$$

$C\geq 0$ is a user defined parameter, which controls the **regularisation**. This is the trade-off between complexity and nonseparable patterns.

同时求优化指标越小越好.

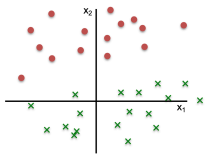
The above constrained optimisation problem can be converted to a QP problem.

Dual problem

$$\max_{\lambda\in\mathbb{R}^N} \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$
$$\text{s.t.} \sum_{i=1}^{N}\lambda_i y_i = 0,\quad 0\leq\lambda_i\leq C$$

Soft-margin SVM

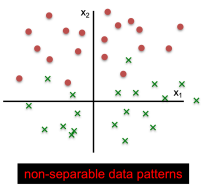
2. linear / non linear, separable / non-separable data patterns



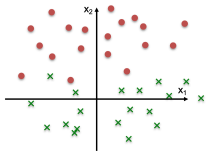
separable data patterns

In practice, no datasets are ideally linearly separable. This means that there always will be some data points misclassified by a linear hyperplane.

What if the data points look like this?

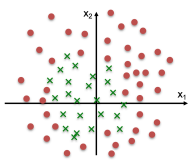


non-separable data patterns



linear data patterns

What if the data points look like this?



non-linear data patterns

3. Usage of different types of SVM.

数据：
- 线性
  - 可分 ~ hard margin, linear SVM
  - 不可分 ~ soft margin, linear SVM
- 非线性
  - 可分 ~ hard margin, kernel SVM
  - 不可分 ~ soft margin, kernel SVM

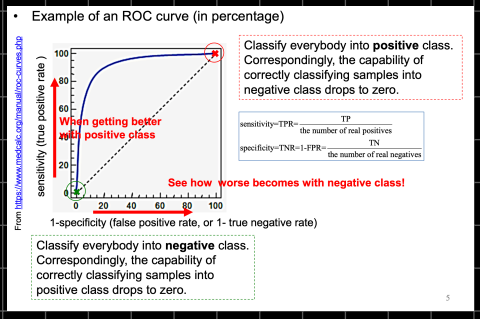
4. Pros & Cons of different machine learning models.

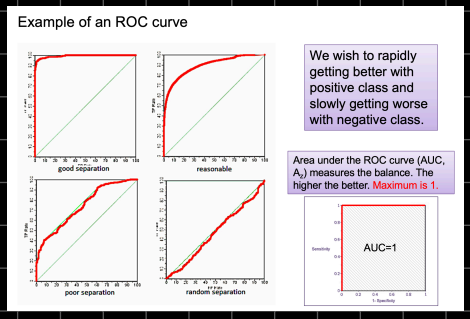
1. ROC (Reciver Operating Characteristic) curve

x 轴：FPR (1- specifity, $\frac{FP}{TF+FP}$)

y 轴：TPR (precision, sensitivity, $\frac{TP}{TP+FN}$)

作用：observe performance change over a varying value. (改变是 binary classifier 的 threshold).





2. different ML models

k-NN

Pros (+):
- A simple intuitive method with no training.
- Can be used for both classification and regression.
- Can handle both linear and nonlinear data patterns.
- Easy to implement for multi-class classification.
- There are only two things to decide, which can be relatively easy.
  - What hyper-parameter value *k* to use?
  - What distance measure to use?

Cons (-):
- Slow with large-scale data.
  - The computational complexity of distance calculation suffers from curse of dimensionality.
  - The computational complexity of neighbour search suffers from training data size.
- High memory cost (needs to store all the training data)
- Not good at dealing with imbalanced data.
- Sensitive to outliers.

(Regularised) Linear Least Squares

Pros (+):
- The most widely used modelling method.
- It is what most people mean when they say they have used "regression", "linear regression" or "least squares" to fit a model to their data.
- Can be used for both regression and classification.
- Makes efficient use of the data. Good results can be obtained with relatively small data.
- Easy to explain and understand.
- Low computational cost: fast training and prediction.
- Low memory.
- In the least squares case, there is no hyper-parameter to set. In the regularised case, there are two things to decide: (1) the form of regularisation term (e.g., l_2, l_1), and regularisation parameter.

Cons (-): not very expressive
- Limited model expressive power, cannot deal with nonlinear data patterns.
- Sensitive to outliers.
- When being used in classification, the performance is sometimes sensitive to how the target output is set, and it does not offer probabilistic interpretation.

Logistic Regression

Pros (+):

- A simple and effective method for classification.
- Low computational cost: fast training and prediction.
- Low memory.
- Offer nice probabilistic interpretation.
- It does not assume specific distribution, e.g., Gaussian, in the model.
- No need to set hyper-parameter.

Cons (-):

- It is a linear classifier and cannot handle nonlinear data patterns.
- Not very good with multi-class classification.
- It is for classification only.

Artificial Neural Network

Pros (+):

- Capable of learning and modelling non-linear and complex relationships .
- It can generalise, supported by advanced training strategies.
- Does not impose restrictions and assumptions to the input.
- Learned information is all stored in network weights.
- Good choice for black-box modelling.

Cons (-):

- Need to determine a "good" network architecture, which is sometimes not easy.
- Long training time for large-scale data.
- Difficult to interpret the learned model.
- The solution is a local optimal.

Support Vector Machines

Pros (+):

- Usually provide competitive performance. It is a good choice when we have no idea on the data.
- Unlike ANN, it is not solved for local optima. globalness
- Scale relatively well to high dimensional data.
- It can generalise.
- Can handle both linear and nonlinear data patterns. (kernel SVM & linear SVM)

Cons (-):

- Need to choose a "good" kernel and select hyper-parameters, which are not easy.
- Long training time for large-scale data.
- Difficult to interpret the learned model.

3. No free lunch Thm:

There is no context-independent or problem-independent reason to favour one method over another. You **always** need to

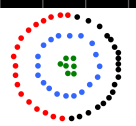
- Understand the data/task/problem to be processed/solved first!
- Experiment with different machine learning models!

Ch 9. Cluster Analysis

1. Definition, key concepts & tasks in clustering

Definition: 将相似的物体聚集在同一组内，不相似的聚在其他组. (unsupervised learning)

Key Tasks:
1. define an appropriate distance (similarity) measure.
2. identify a cluster number.
3. specify clustering algorithm to group objects into sensible clusters.
4. evaluate whether the clustering result is good or not.



Different clusters are distinguished by colors.

Is this a reasonable clustering output?

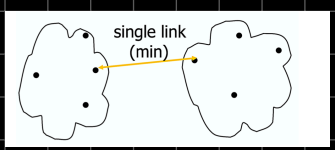
For data points with complex patterns, different algorithms are likely to produce different clustering results.

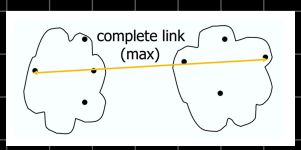
- Often which distance (similarity) measure to use and what cluster number to adopt is treated as a model selection process, where cluster number is a hyper-parameter.
- There are some clustering algorithms that attempt to decide a cluster number for you, but they requires to set other type of hyper-parameters.

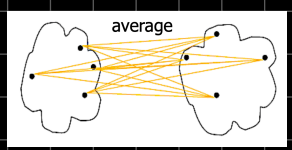
2. Distance calculation between clusters: (single-link, complete-link, average-link)

measure the distance (similarity) between 2 data clusters:

* single link: 两个 cluster 间点距离最小值. $\min_{p \in cluster1, q \in cluster2} d(x_p, x_q)$
* complete link: 两个 cluster 间点距离最大值. $\max_{p \in cluster1, q \in cluster2} d(x_p, x_q)$
* average link: 两个 cluster 间点距离平均值. $\frac{1}{n_i n_j} \sum_{p \in cluster_i, q \in cluster_j} d(x_p, x_q)$







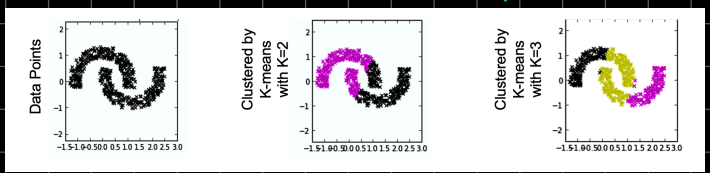
3. Principle & usage of K-means algorithm

usage: group a set of datapoints to k clusters.

Steps:

- Pre-determine the **cluster number** *k*.
- Choose your **distance (or similarity) measure**.
- Initialise by **random** selecting k cluster center points.
- Iterate:
  - Step 1: **Calculate distances (or similarities)** between the data points and the cluster center points.
  - Step 2: Find the **nearest** cluster center to each data point, and assign the data point to that cluster.
  - Step 3: Calculate the new cluster center for each cluster, by **averaging** its member points. update cluster center.
  - Step 4: Go back to Step 1), stop when there is no change in the membership of each cluster.

Issues:
1. sensitive to initial cluster center points.
2. k need to be specified in advance.
3. can't handle noisy data & outliers.
4. not suitable for complex data patterns.



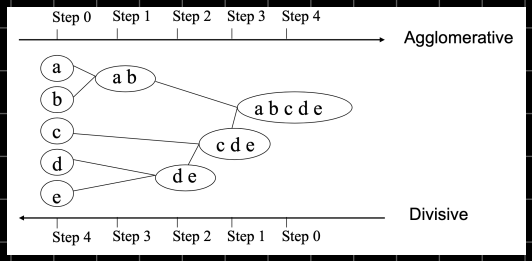
4. Basic idea of hierarchical clustering.

Idea: 1. group objects into tree of clusters ⇒ nested partitions layer by layer.

2. partition dataset sequentially, no need to know cluster number.

typical strategies:

- Agglomerative (bottom-up strategy).
  - Start from treating each data point as a cluster (atomic cluster).
  - Then **merge** the atomic clusters into larger and larger clusters.
- Divisive (top-down strategy).
  - Start from treating all data points as one single cluster.
  - Then **divide** this cluster into smaller and smaller clusters.



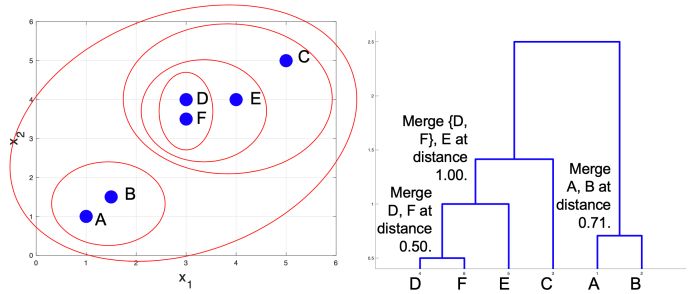
5. Principle & usage of agglomerative algorithm:

principle: 自底向上，每次 iteration 依 distance matrix 中距离最小的一对 cluster 合并。初始时每个 datapoint 视为一个 cluster。最后 cluster 数收敛到 1。 (非常类似 Kruskal Algorithm)

- Implemented in three steps:
  - Step 1: Calculate the distance matrix **D** between all the data points. ← 一开始算的就是点间 distance，可用 Euclid.
  - Step 2: Set each data point as a cluster. The between-cluster matrix **M** is equal to **D**.
  - Step 3: Repeat the following until the cluster cluster becomes one,
    - Merge the two closest clusters.
    - Update the between-cluster distance matrix **M**. ← 可用 single link, complete link, average link.
- Major drawbacks:
  - Sensitive to cluster distance measures and noise/outliers.
  - Less efficient: $O(N^2 \log N)$, where N is the number of total data points.

注：某个中间 cluster 的"生命周期"：
the distance difference between it's created & merged.

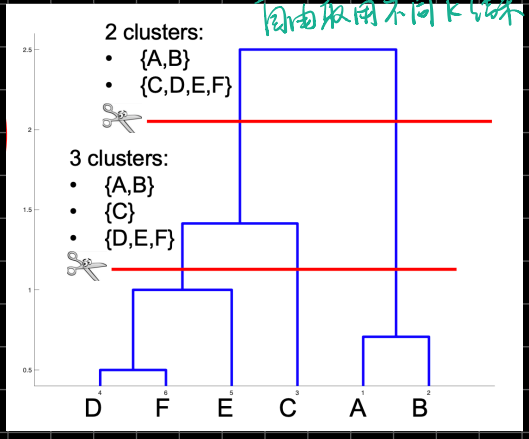
Example: Apply the agglomerate algorithm to the 6 data points below.



A: (1, 1) B: (1.5, 1.5)
C: (5, 5) D: (3, 4)
E: (4, 4) F: (3, 3.5)

Lifetime of a cluster is the distance difference between that it is created and merged. E.g., Lifetime of {D, F} = 1.00-0.50=0.50

同样取用不同 k 的结果



6. Typical internal & external indexes for cluster validation

目标：validate if the clustering make sense.

* internal criteria (indexes)：不用外部信息（如现有的分类结果），只基于"纯内"评价。
* external criteria (indexes)：利用外部信息，validate against ground truth.

Typical internal indexes : F- ratio index

- Normally, a good clustering result should have small within-cluster variance while large between-cluster variance (K is cluster number).
  - Within-cluster variance (SSW): $SSW = \sum_{i=1}^{K} \sum_{p \in \text{cluster}_i} d^2(\mathbf{x}_p, \mathbf{c}_i)$
  - Between-cluster variance (SSB): $SSB = \sum_{i=1}^{K} n_i d^2(\mathbf{c}_i, \mathbf{c})$
  - F-ratio index: $F = K \frac{SSW}{SSB}$

n_i: number of data points in cluster i

$\mathbf{c}_i = \frac{1}{n_i} \sum_{p \in \text{cluster}_i} \mathbf{x}_p$: cluster center

$\mathbf{c} = \frac{1}{N} \sum_{p=1}^{N} \mathbf{x}_p$: whole data center

- The above measures can be used to select cluster number.

聚类内点 variance 尽可能小

不同聚类中心距全体的中心尽可能大.

$F = K \cdot \frac{SSW}{SSB}$. 可用于寻找合适的 K.

F- ratio

Typical External Validation : Rand Index

validate against a set of ground truth class labels

- It is computed by considering all pairs of data points by looking into agreement and disagreement against the "ground truth".
- It is defined based on the following agreement/disagreement table:

Obviously, a better match has larger a and d, while smaller b and c.

	Pairs of data points from the same class	Pairs of data points from different classes
Pairs of data points from the same cluster	a= # agreement	b= # disagreement
Pairs of data points from different clusters	c= # disagreement	d= # agreement

- Rand index is computed by $Rand = \frac{a+d}{a+b+c+d}$

- Assume there are *K* clusters and *C* classes, rand index can also be computed from the following contingency table:

C classes

K clusters

$$\begin{bmatrix} n_{11} & n_{12} & \cdots & n_{1C} \\ n_{21} & n_{22} & \cdots & n_{2C} \\ \vdots & \vdots & \ddots & \vdots \\ n_{K1} & n_{K2} & \cdots & n_{KC} \end{bmatrix}$$

n_{ij} is the number of data points in both cluster i and class j.
N is the total number of data points.

$$a = \frac{1}{2} \sum_{i=1}^{K} \sum_{j=1}^{C} n_{ij}(n_{ij} - 1)$$

$$b = \frac{1}{2} \left(\sum_{j=1}^{C} \left(\sum_{i=1}^{K} n_{ij} \right)^2 - \sum_{i=1}^{K} \sum_{j=1}^{C} n_{ij}^2 \right)$$

$$c = \frac{1}{2} \left(\sum_{i=1}^{K} \left(\sum_{j=1}^{C} n_{ij} \right)^2 - \sum_{i=1}^{K} \sum_{j=1}^{C} n_{ij}^2 \right)$$

$$d = \frac{1}{2} \left(N^2 + \sum_{i=1}^{K} \sum_{j=1}^{C} n_{ij}^2 - \sum_{j=1}^{C} \left(\sum_{i=1}^{K} n_{ij} \right)^2 - \sum_{i=1}^{K} \left(\sum_{j=1}^{C} n_{ij} \right)^2 \right)$$

知道即可. 不考.

Ch 10. Deep Learning

1. Basic Deep Learning techniques

Definition: deep learning ~ techniques for learning using neural networks.
also considered a kind of representation (feature) learning techniques

e.g.: AlexNet

Advanced neural network architecture:

- CNN: 卷积神经网络.
  used to automatically learn good feature vector for image from its pixels.
- RNN: 循环神经网络.
  useful for learning from sequential data.

Attention Mechanism:
注意力机制: 基于权重和 (weight sum) 的 mathematical diagram, 用 weight functions 定位显著信息.

用处: help identify contributing parts in neural network to decision making.

缓解 Neural network 的 Black box 效应. make it easier to be interpreted.

- **Black box decision**: A decision supported by model parameters and mathematical operations that are "hard" to understand.


Why are these two images similar?

- **Attention mechanism**: A type of effective mathematical diagrams based on weighted sum, which uses weight functions to locate salient information.
- Embed an attention mechanism to a neural network would help identify the contributing parts to final decision making.

Training tips for Deep neural network:

1. Pre-train: (ResNet, VGGNet trained on ImageNet data)
   use a pre-trained neural network if data share similar structure to existing training data that has been used to train some well-known neural networks.

2. Batch Normalisation:
   remove mean & scale by standard deviation for hidden layers' output.

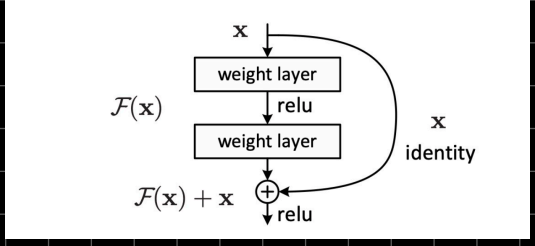
3. Regularisation: 防止过拟合.

Typical regularisation techniques:

- Parameter norm penalties, remember $O(\mathbf{W}_{NN}) = loss(\mathbf{W}_{NN}) + \lambda \frac{1}{2} \|\mathbf{W}_{NN}\|_2^2$? (L_2-regularisation (ridge))
- Dataset augmentation by creating fake data and adding to the training set.
- Add small perturbation (noise) to network weights.
- Early stopping by treating the number of training iterations as a hyper-parameter and finding the best one by using validation data.
- Sparse representation by forcing hidden representation to have more zeros.
- Ensemble methods by training multiple models separately at the same time, and letting them vote the final output.
- Dropout by randomly removing a percentage of neurons during training.

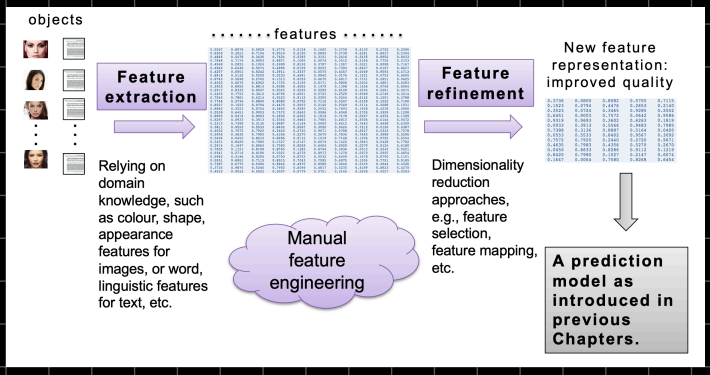
加 regularisation 项
加假数据
扰动 权重
提前中止学习
稀疏化表示
Ensemble Method
Drop out

4. Skip connection. (in ResNet)



2. Differences between deep learning & traditional machine learning strategies

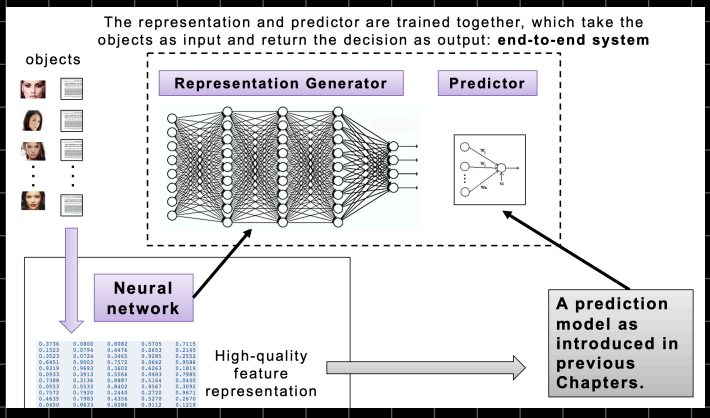
traditional ML: use manual feature engineering:
feature extraction ~ feature refinement ~ prediction model.



deep learning: use neural network to directly provide high-quality feature representation

用 neural network 取代了 feature extraction 和 feature refinement.

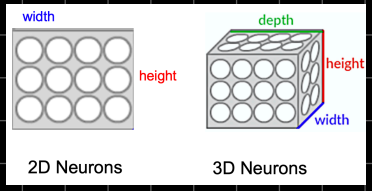
representation and predictor are trained together, take obj as input, return decision as output ⇒ end-to-end system.



3. Convolutional Network (architecture, convolutional, pooling, fully-connected layers)

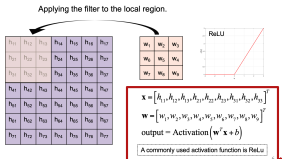
main recipe:

1. 2D / 3D neurons.



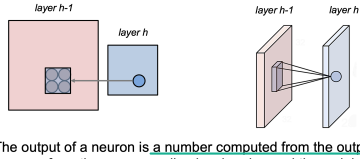
2. Convolution Layers 卷积层
receptive field

- The operation for applying the filter to a local region of neurons:



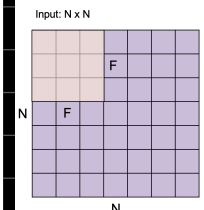
卷积核中有权重
没有 weight sharing

- **Local connectivity**: Each neuron inside a layer is connected to only a small region of the previous layer, called a receptive field.



- The output of a neuron is a number computed from the output of those neurons from the corresponding local region and the weights of the convolutional filter: y =Activation($\mathbf{w}^T\mathbf{x}+b$)
- Weight sharing: One same filter of the size of the local region slides over all spatial locations (in different words, slides over all the neurons in the layer).

Slide the filter over all spatial locations.



"滑动"的步长.

stride S

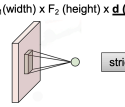
- The sliding process results in an activation map of size:

$$\left(\frac{N-F}{S}+1\right)\times\left(\frac{N-F}{S}+1\right)$$

- You will need to set appropriate filter size and stride size so that (N-F)/S is an integer.

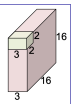
- Let's look at the case of 3D neurons. 同时包含深度.

Green cube is a convolutional filter:
F_1(width) x F_2 (height) x **d (depth)**



Red cube is one layer of 3D neurons:
N_1 (width) x N_2 (height) x **d (depth)**

Example:
Given a layer of 16 x 16 x 3 neurons, a 2 x 2 x 3 convolutional filter, The output of each neuron is denoted by h_{ijk} and the weight by w_i. For instance, the output of the first neuron in the next layer is computed as follows:



$$\mathbf{x}=[h_{111},h_{121},h_{211},h_{221},h_{112},h_{122},h_{212},h_{222},h_{113},h_{123},h_{213},h_{223}]^T$$
$$\mathbf{w}=[w_1,w_2,w_3,w_4,w_5,w_6,w_7,w_8,w_9,w_{10},w_{11},w_{12}]^T$$
$$\text{output}=\text{Activation}(\mathbf{w}^T\mathbf{x}+b)$$

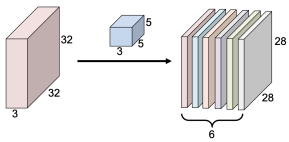
- The resulting activation map based on one convolutional filter is 2D of size:

$$\left(\frac{N_1-F_1}{S}+1\right)\times\left(\frac{N_2-F_2}{S}+1\right)$$

- Usually, a set of T (T>1) convolutional filters are applied to obtain T separate activation maps.

For instance, when applying six 5 x 5 x 3 filters to a 32 x 32 x 3 input with stride 1, it results in 6 activation maps of size 28 x 28.

$$\left(\frac{32-5}{1}+1\right)\times\left(\frac{32-5}{1}+1\right)=28\times28$$



注：一般会用不止一个卷积核！

一个 map 的大小：

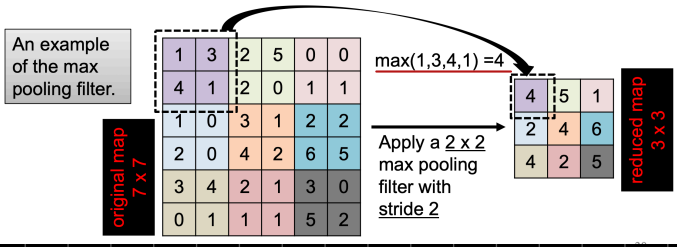
$$\left(\frac{N_1-F_1}{S}+1\right)\left(\frac{N_2-F_2}{S}+1\right).$$

3. Pooling Layer 池化层

take convolution layer's output: activation map as input, then reduce the size of each activation map. ("down sampling")

池化核（pooling filter）用法和卷积核（convolutional layer）用法基本一样.

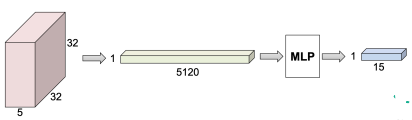
- The pooling layer takes the activation maps returned by a convolutional layer as the input, and reduces the size of each activation map separately.
- It can be viewed as an operation of down-sampling. 降采样
- A pooling filter slides over all the spatial locations in an activation map in the same way as a 2D convolutional filter.



4. Fully connected Layer 全连接层.

- The output of a convolutional (or pooling) layer is a set of activation maps, sored in a 3D array.
- These values in the 3D array can be moved to a 1D array. For instance:

  32 x 32 x 5 array ➔ 5120 x1 array
- The fully connected layers are equivalent to a multilayer perceptron (MLP) taking the generated 1D array as the input.

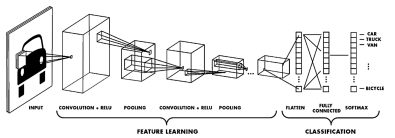


接收卷积层 or 池化层的输出：
a set of activation map. 存在 3D matrix 里.

拍扁铺平. 以 1D array 为输出：

5. CNN architecture:

- A CNN is a sequence of convolutional (and pooling) layers, followed by fully connected layers in the end:



Used image is from MathWorks: https://ww2.mathworks.cn/solutions/deep-learning/convolutional-neural-network.html

- Its unique architecture (2,3D neurons, local connectivity, etc.) makes it suitable for processing 2,3D data with typical local patterns, particularly images (n x n x 3 input where 3 corresponds to the R, G and B channels).

a sequence of convolutional / pooling layers,
后接一个全连接层.
输入是 image.