中山大学计算机学院研究生人工智能应用基础系列课程

集成学习

马 锦 华

中山大学

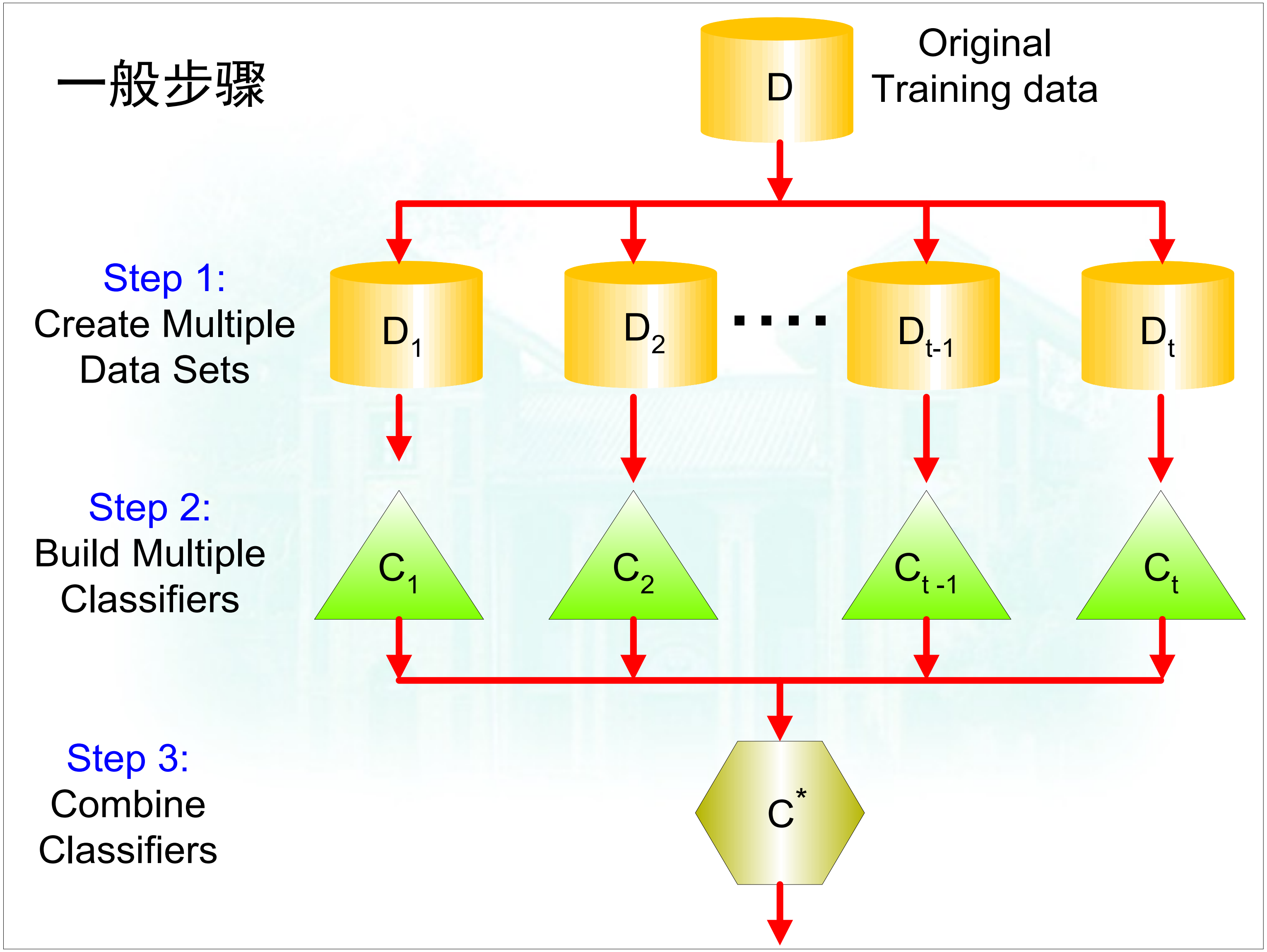
机器智能与先进计算
教育部重点实验室

集成学习

- 集成学习理论
- 传统集成学习方法
- 基于多核学习的集成
- 独立性假设下的集成与依赖性建模
- 深度学习中的集成

什么是集成学习

一般步骤



从臭皮匠到诸葛亮？

- 假设有T个分类器
  - 每个分类器都是弱分类器（错分率ε<50%）
  - 分类器的错分率相互独立
  - 通过简单多数投票集成T个分类器
  - 集成的错误率为

$$P\left(X \geq \left\lceil \frac{T}{2} \right\rceil\right) = \sum_{i=\left\lceil \frac{T}{2} \right\rceil}^{T} \binom{T}{i} \varepsilon^{i} (1-\varepsilon)^{T-i} \leq \exp\left(-\frac{1}{2} T (1-2\varepsilon)^{2}\right)$$

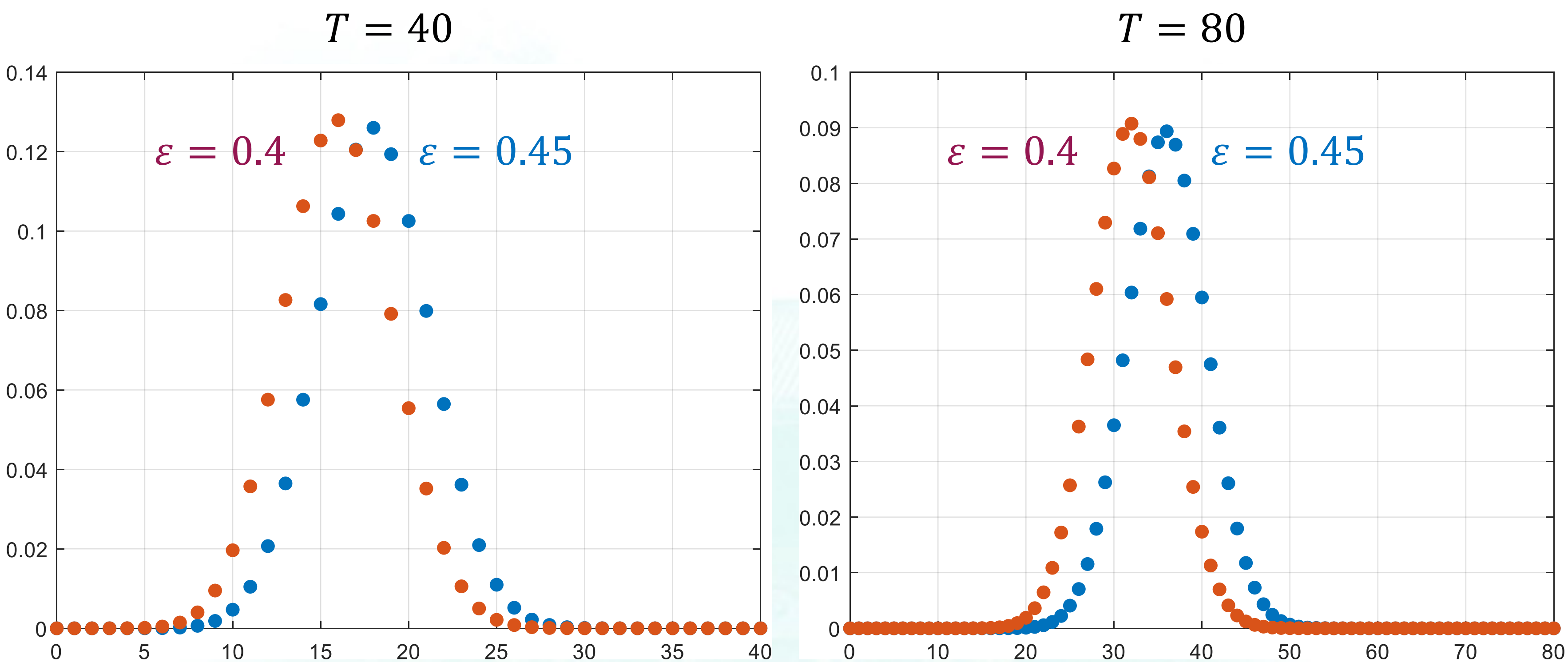
By Hoeffding's inequality

X为T个分类器中错误分类的数目

  - 随着T增大，集成的错误率将指数下降

从臭皮匠到诸葛亮？

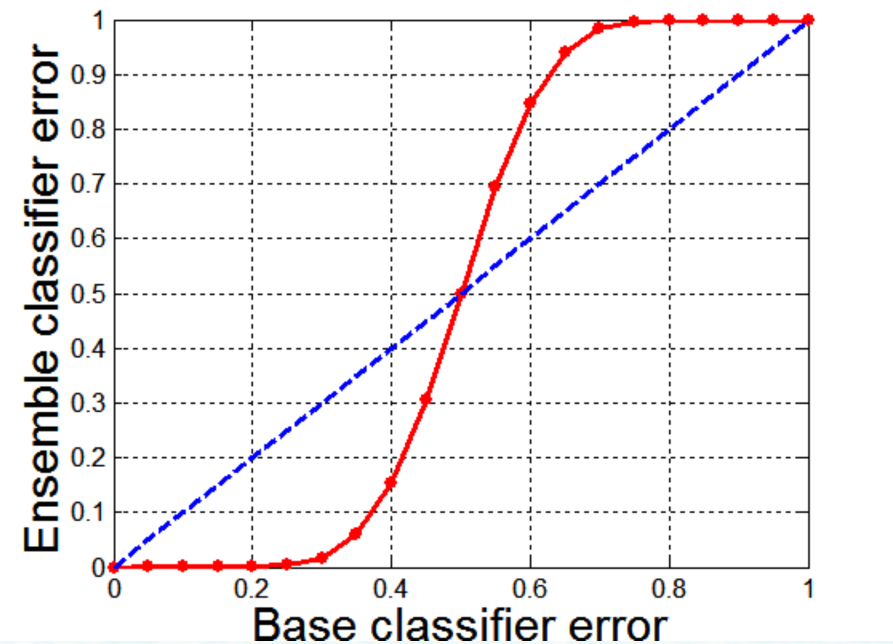
- 二项分布



- X近似地服从$N\big(T\varepsilon, T\varepsilon(1-\varepsilon)\big)$

从臭皮匠到诸葛亮？

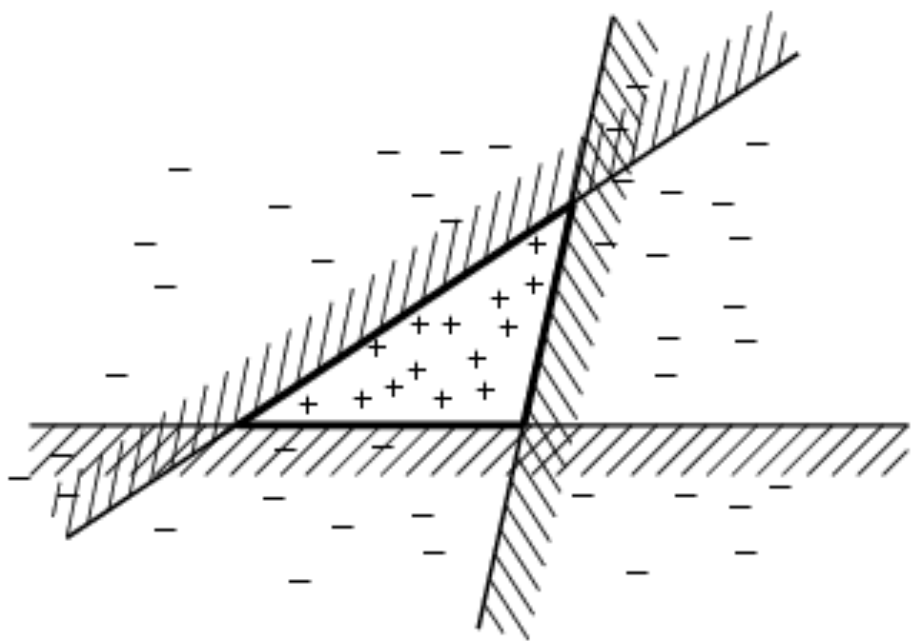
- 例如有25个分类器，错分率ε=0.35



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

从臭皮匠到诸葛亮？

- 集成后的线性分类器可区分非线性可分数据



如何获得好的集成分类器：误差-分歧分解

- 第i个分类器C_i与集成分类器C的分歧（ambiguity）：

$$A_i = (C_i - C)^2$$

- 集成分歧的加权平均（α_i为集成的权重）：

$$\bar{A} = \sum_{i=1}^{T} \alpha_i A_i = \sum_{i=1}^{T} \alpha_i (C_i - C)^2$$

- 第i个分类器C_i与集成分类器C的误差：

$$E_i = (C_i - Y)^2$$

$$E = (C - Y)^2$$

如何获得好的集成分类器：误差-分歧分解

- 个体分类器误差的加权平均：

$$\bar{E} = \sum_{i=1}^{T} \alpha E_i = \sum_{i=1}^{T} \alpha_i (C_i - Y)^2$$

- 集成分歧的加权平均写成：

$$\bar{A} = \sum_{i=1}^{T} \alpha_i [(C_i - Y) - (C - Y)]^2$$

$$= \bar{E} - E$$

- 集成分类器C的误差： $E = \bar{E} - \bar{A}$

集成学习成功的要素

- 如何获得一组好的分类器
  - 每个分类器错分率ε<50%
  - 各个分类器之间有差异（错分率相互独立）
    - 使用不同的训练样本
      - 控制样本分布：bagging，boosting
      - 控制输入特征：随机森林
      - 数据增强：翻转、随机裁剪、颜色变换等
    - 使用不同的训练算法
      - 不同的分类模型：决策树、神经网络等
      - 不同的初始参数

集成学习成功的要素

- 如何将分类器以合适的方式进行集成
  - 独立假设下的集成策略
    - 乘积法、均值法、最大值法、最小值法、中值法、多数投票法
  - 非独立假设下的集成策略
    - Stacking：将每个分类器的输出看作特征进行次级（元）学习（meta-learner）
    - 利用概率分布性质建立关联性模型

集成学习的成功应用

- KDDCup07：第一名 “… Decision Forests and …”
- KDDCup08：挑战1第一名利用Bagging；挑战2第一名 “… Using an Ensemble Method ”
- KDDCup09：Fast Track第一名 “Ensemble … ”；Fast Track第二名 “… bagging … boosting tree models …”；Slow Track第二名 “Boosting … ”；Slow Track第二名 “Stochastic Gradient Boosting”
- KDDCup10：第一名 “… Classifier ensembling”；第二名 “… Gradient Boosting machines … ”

集成学习的成功应用

- KDDCup11：Track1第一名 "A Linear Ensemble … "；Track1第二名 "Collaborative filtering Ensemble"；Track2第一名 "Ensemble …"；Track2第二名 "Linear combination of …"
- KDDCup12：Track1第一名 "Combining … Additive Forest …"；Track2第一名 "A Two-stage Ensemble of…"
- KDDCup13：Track1第一名 "Weighted Average Ensemble"；Track1第二名 "Gradient Boosting Machine"；Track2第一名 "Ensemble the Predictions"

集成学习的成功应用

- KDDCup14：第一名 “ensemble of GBM, ExtraTrees, Random Forest …” 和 “the weighted average”；第二名 “use both R and Python GBMs”；第三名 “gradient boosting machines ... random forests” 和 “the weighted average of…”

- Netflix奖：
  - 2007进步奖：集成学习
  - 2008进步奖：集成学习
  - 2009百万特等奖：集成学习

概览

- 集成学习理论
- 传统集成学习方法
- 基于多核学习的集成
- 独立性假设下的集成与依赖性建模
- 深度学习中的集成

传统集成学习方法

- Boosting
- Bagging
- 随机森林
- Stacking

Boosting

- 自适应地改变样本分布进行采样，每次迭代更关注之前被错分的样本
  - 初始化样本权值相等
  - 更新选取样本的规则：每次迭代中改变样本权值（区别于Bagging）
  - 重复以上过程得到若干分类器
  - 采用加权平均决定最后的分类预测

Boosting

- 每次迭代：
  - 增加错误分类样本的权值
  - 减小正确分类样本的权值

- 例如每次迭代选取的样本如下：

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

  - 样本4难以被正确分类
  - 其权值增加，因此每次迭代中更可能被选中

代表算法：AdaBoost

2003年哥德尔奖（Gödel Prize）

Freund & Schapire, A decision theoretic generalization of on-line learning and an application to Boosting. *Journal of Computer and System Sciences*, 1997, 55: 119-139.

- 优点：
  - 准确的分类
  - 十分简单（“just 10 lines of code” – Schapire）
  - 广泛成功的应用
  - 合理的理论基础
  - … …

AdaBoost算法介绍

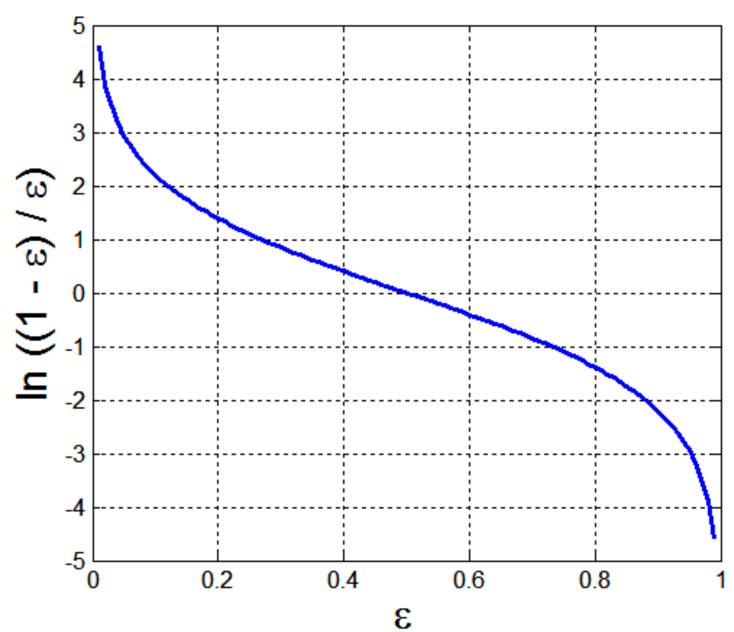
- 基分类器：$C_1, C_2, \ldots, C_T$

- 错分率：

$$\varepsilon_j = \sum_{i=1}^{N} w_i \delta\left(C_j(x_i) \neq y_i\right)$$

- 每个分类器的权值

$$\alpha_j = \frac{1}{2}\ln\left(\frac{1-\varepsilon_j}{\varepsilon_j}\right)$$



AdaBoost算法介绍

- 最小化指数损失函数

$$l(\alpha_j) = \sum_{i=1}^{N} w_i \exp(-\alpha_j C_j(x_i) y_i)$$

$$= \sum_{i=1}^{N} w_i \exp(-\alpha_j C_j(x_i) y_i) \left[\delta\left(C_j(x_i) = y_i\right) + \delta\left(C_j(x_i) \neq y_i\right)\right]$$

$$= \sum_{i=1}^{N} w_i \left[\exp(-\alpha_j)\, \delta\left(C_j(x_i) = y_i\right) + \exp(\alpha_j)\, \delta\left(C_j(x_i) \neq y_i\right)\right]$$

$$= \exp(-\alpha_j)\,(1 - \varepsilon_j) + \exp(\alpha_j)\, \varepsilon_j$$

$$\frac{dl(\alpha_j)}{d\alpha_j} = -\exp(-\alpha_j)\,(1 - \varepsilon_j) + \exp(\alpha_j)\, \varepsilon_j \quad \alpha_j = \frac{1}{2}\ln\left(\frac{1-\varepsilon_j}{\varepsilon_j}\right)$$

AdaBoost算法介绍

- 权值更新：

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp(-\alpha_j) & \textbf{if } C_j(x_i) = y_i \\ \exp(\alpha_j) & \textbf{if } C_j(x_i) \neq y_i \end{cases}$$

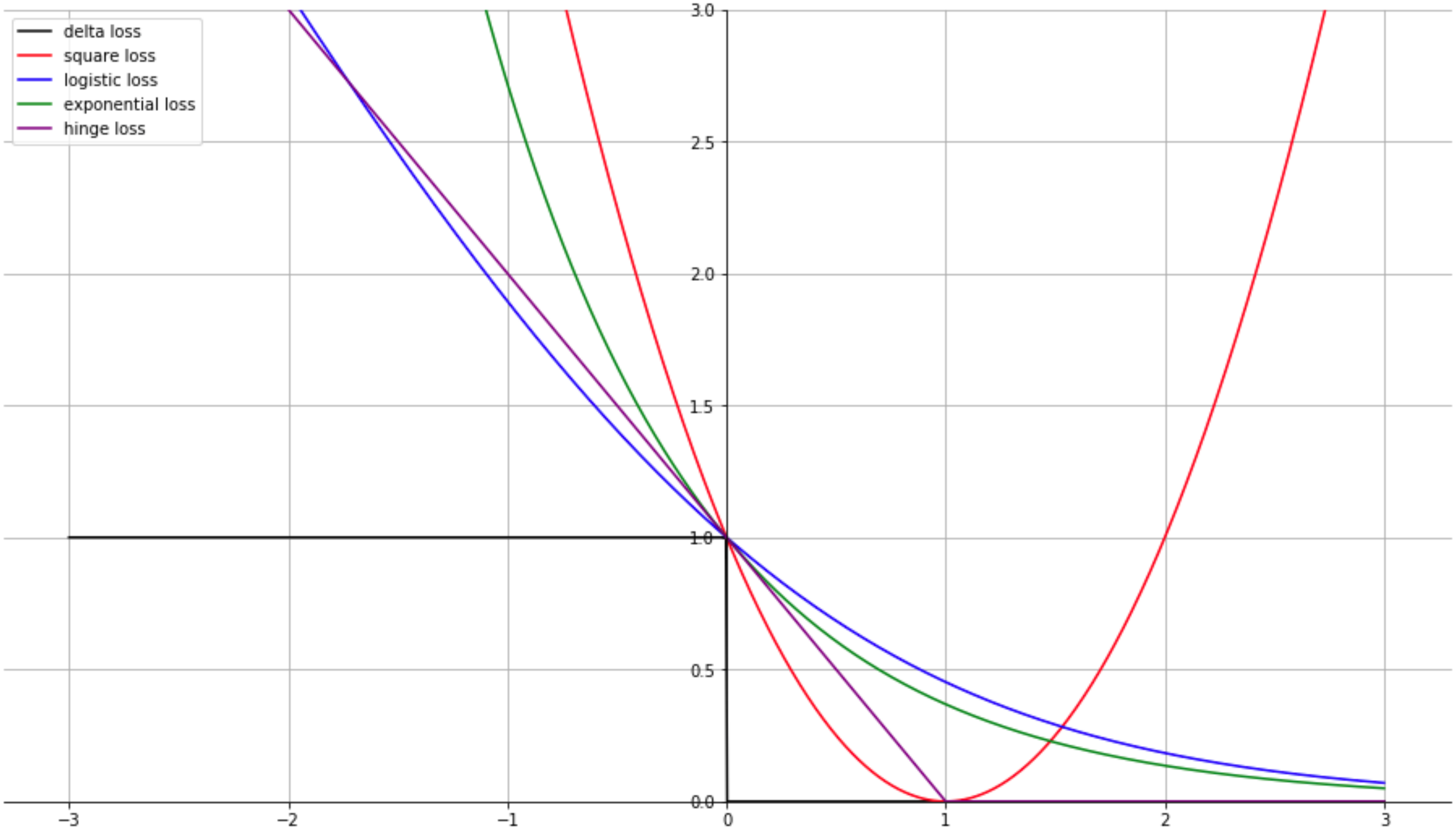
这里Z_j是归一化因子

- 如果某次迭代的错分率大于50%，权值重新分配为1/N，并重复取样过程（避免过早停止）
- 集成分类器：

$$C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

AdaBoost算法介绍

- 最小化指数损失函数



AdaBoost算法

Algorithm 5.7 AdaBoost Algorithm

1: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Initialize the weights for all n instances.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all instances in the original training set, D.
7: $\epsilon_i = \left[\sum_j w_j\, \delta(C_i(x_j) \neq y_j)\right]$ {Calculate the weighted error}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/n \mid j = 1, 2, \cdots, n\}$. {Reset the weights for all n instances.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each instance according to equation (5.88).
14: **end for**
15: $C^*(\mathbf{x}) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y)$.

优点：降低集成分类器的偏差（可选用弱学习器）

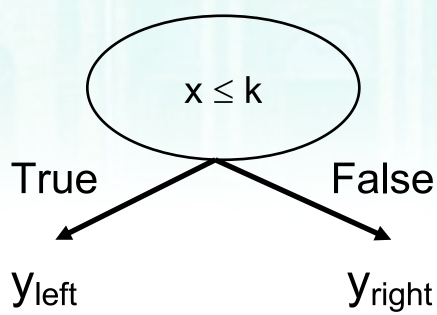
AdaBoost算法示例

- 考虑简单的一维数据集：

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- 每个基分类器是一个简单的决策树
  - 决策规则：$x \le k$ VS $x > k$
  - 分割点k通过信息熵选取



AdaBoost算法示例

- 前3次boosting迭代：

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- 基分类器与相应权值：

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

AdaBoost算法示例

- 权值更新：

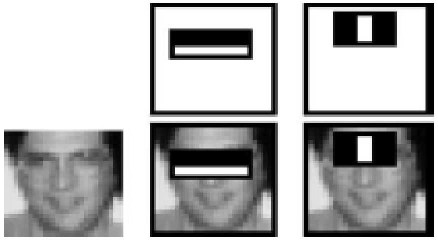
Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

- 集成分类结果：

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

AdaBoost算法应用

- 目标检测：
  - Viola-Jones检测器

    Longuet-Higgins奖（2011）

    Viola & Jones, Rapid object detection using a Boosted cascade of simple features. *CVPR*, 2001.
  - 同等准确率，（当时）比state-of-the-art速度快15倍
- 目标跟踪：
  - 基于集成学习的目标跟踪算法

    Avidan, Ensemble Tracking. *IEEE TPAMI*, 29(2):261-271, 2007.

其他Boosting算法

- Boosting算法的一般框架

Input: Sample distribution $\mathcal{D}$;
Base learning algorithm $\mathfrak{L}$;
Number of learning rounds T.

Process:
1. $\mathcal{D}_1 = \mathcal{D}$.   % Initialize distribution
2. **for** $t = 1, \ldots, T$:
3. $h_t = \mathfrak{L}(\mathcal{D}_t)$;   % Train a weak learner from distribution $\mathcal{D}_t$
4. $\epsilon_t = P_{\boldsymbol{x} \sim D_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;   % Evaluate the error of h_t
5. $\mathcal{D}_{t+1} = Adjust\_Distribution(\mathcal{D}_t, \epsilon_t)$
6. **end**

Output: $H(\boldsymbol{x}) = Combine\_Outputs(\{h_1(\boldsymbol{x}), \ldots, h_t(\boldsymbol{x})\})$

- AdaBoost.M1, AdaBoost.MR, FilterBoost, GentleBoost, GradientBoost, MadaBoost, LogitBoost, LPBoost, MultiBoost, RealBoost, RobustBoost, …

Gradient Boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma).$$

2. For m = 1 to M:
   1. Compute so-called *pseudo-residuals*:

   $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

   $$\gamma_m = \arg\min_\gamma \sum_{i=1}^n L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

   $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

XGBoost

Input: training set $\{(x_i, y_i)\}_{i=1}^{N}$, a differentiable loss function $L(y, F(x))$, a number of weak learners M and a learning rate α.

Algorithm:

1. Initialize model with a constant value:

$$\hat{f}_{(0)}(x) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta).$$

2. For m = 1 to M:

   1. Compute the 'gradients' and 'hessians':

$$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

$$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

   2. Fit a base learner (or weak learner, e.g. tree) using the training set $\{x_i, -\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\}_{i=1}^{N}$ by solving the optimization problem below:

$$\hat{\phi}_m = \arg\min_{\phi\in\mathbf{\Phi}} \sum_{i=1}^{N} \frac{1}{2}\hat{h}_m(x_i)\left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i)\right]^2.$$

$$\hat{f}_m(x) = \alpha\hat{\phi}_m(x).$$

   3. Update the model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x).$$

3. Output $\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x).$

传统集成学习方法

- Boosting
- Bagging
- 随机森林
- Stacking

Bagging（Bootstrap AGGregatING）

- 并行式集成学习方法的代表
- “有放回”的取样：

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- 每个采样集训练一个分类器
- 每个样本被选择的概率：

$$1-\left(1-\frac{1}{n}\right)^{n} \rightarrow 1-\frac{1}{e} \approx 63.2\%$$

Bagging算法

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \arg\max_y \sum_i \delta\big(C_i(x) = y\big)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

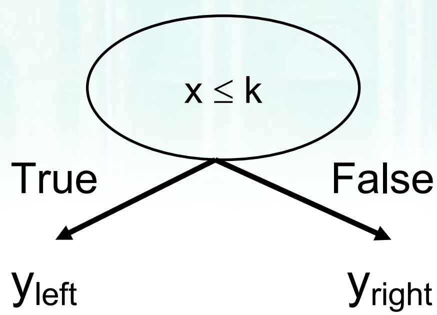
Bagging算法示例

- 考虑简单的一维数据集：

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- 每个基分类器是一个简单的决策树
  - 决策规则：$x \le k$ VS $x > k$
  - 分割点k通过信息熵选取



Bagging算法示例

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging算法示例

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging算法示例

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 → y = -1
x > 0.75 → y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 → y = 1
x > 0.05 → y = 1

Bagging算法示例

- 所有基分类器：

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging算法示例

- 利用多数投票准则集成分类结果：

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Bagging算法的优缺点

- Bagging算法集成分类器的期望和方差

$$\mathrm{E}(C)=\mathrm{E}\left(\frac{1}{T}\sum_{i=1}^{T}C_i\right)=\frac{1}{T}\sum_{i=1}^{T}\mathrm{E}(C_i)=\mathrm{E}(C_i)=\mu$$

$$\mathrm{D}(C)=\mathrm{D}\left(\frac{1}{T}\sum_{i=1}^{T}C_i\right)=\frac{\sigma^2}{T}+\frac{T-1}{T}\rho\sigma^2$$

其中μ和σ^2为基分类器的均值和方差，ρ为基分类器间的相关系数

Bagging算法的优缺点

- Bagging集成分类器的偏差与基分类器的偏差近似
  - 因此，Bagging通常选用偏差低的强学习器
- Bagging算法主要是减少分类模型的方差
  - Bagging的抽样是有放回抽样，相关系数$0 < \rho < 1$
  - Bagging算法一般用于训练算法对于训练数据较为敏感的场合（如：决策树）

传统集成学习方法

- Boosting
- Bagging
- 随机森林
- Stacking

随机森林算法

Algorithm 1 Random Forest

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

function RANDOMFOREST(S, F)
    $H \leftarrow \emptyset$
    for $i \in 1, \ldots, B$ **do**
        $S^{(i)} \leftarrow$ A bootstrap sample from S
        $h_i \leftarrow$ RANDOMIZEDTREELEARN($S^{(i)}$, F)
        $H \leftarrow H \cup \{h_i\}$
    end for
    return H
end function
function RANDOMIZEDTREELEARN(S, F)
    At each node:
        $f \leftarrow$ very small subset of F
        Split on best feature in f
    return The learned tree
end function

随机森林算法的优点

- 随机选择特征的好处
  - 通过随机选择的特征训练的基分类器（决策树）相关性更低（ρ更小）
  - 由于特征数的减少，在给定的时间内学习更多的基分类器（决策树），提高泛化能力
- 老当益壮的Leo Breiman，1928-2005
  - Breiman, Random forests. *Machine learning* 45(1):5-32, 2001.

传统集成学习方法

- Boosting
- Bagging
- 随机森林
- Stacking

Stacking算法

算法 2（Stacking）

数据D，第一层训练算法$L_1, L_2, \dots, L_T$，第二层训练算法 L'

1. **for** i=1 to T:

2. $h_t = L_i(D)$

3. **end**

4. $D' = \emptyset$

5. **for** j=1 to m

6. **for** t=1 to T

7. $z_{it} = h_t(x_i)$

8. **end**

9. $D' = D' \cup \{((z_{i1}, z_{i2}, \dots, z_{iT}), y_i)\}$

10. **end**

11. $h' = L'(D')$

12.输出：$H(x) = h'(h_1(x), \dots, h_T(x))$

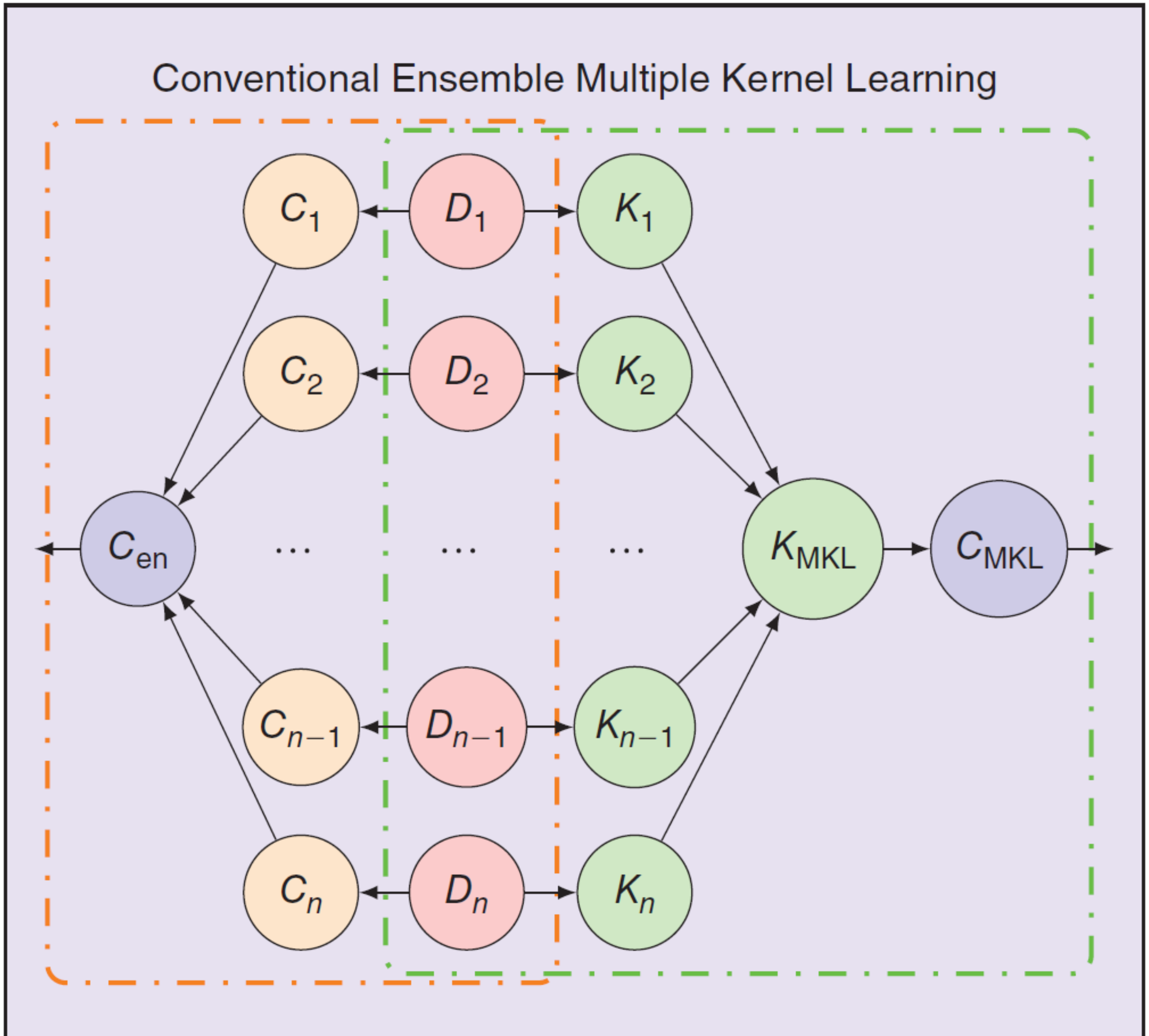
概览

- 集成学习理论
- 传统集成学习方法
- 基于多核学习的集成
- 独立性假设下的集成与依赖性建模
- 深度学习中的集成

多核学习背景

- 提出多核学习（Multiple Kernel Learning, MKL）的动机
- 特征融合
  - 每个样本可提取多种特征
  - 例如，图像分类中可提取SIFT, HOG, GIST
  - 有效地选择组合多种能提升预测准确率
- 多核融合
  - 如何选择不同类型的核函数，如线性核、高斯核、多项式核等等？
  - 不同参数的核函数，如何选择参数？

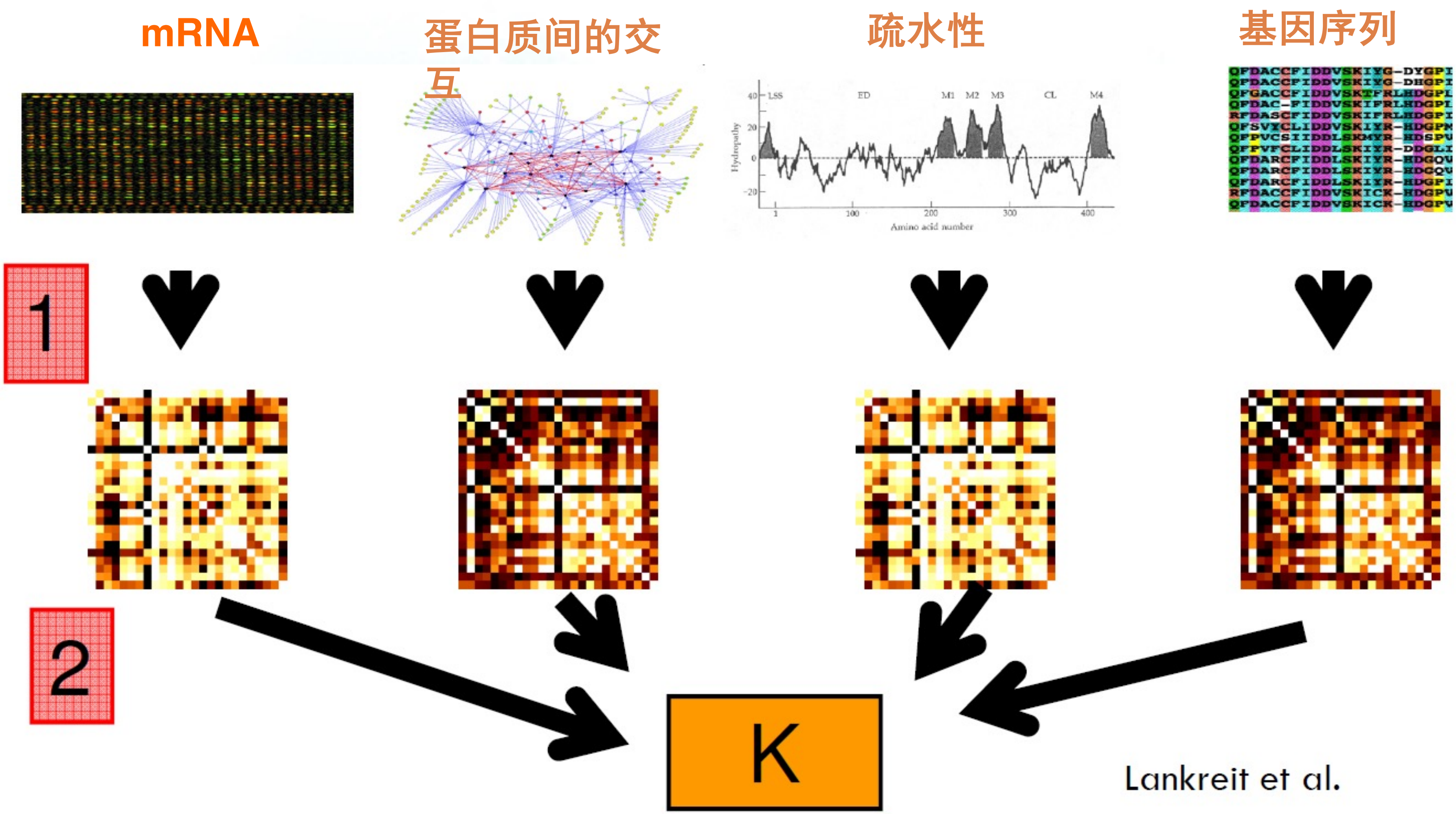
基于多核学习的集成



Conventional Ensemble Multiple Kernel Learning
C_1
D_1
K_1
C_2
D_2
K_2
C_{en}
K_{MKL}
C_{MKL}
C_{n-1}
D_{n-1}
K_{n-1}
C_n
D_n
K_n

多核学习应用举例

- 蛋白质分类



多核学习的损失函数

- 多核集成的分类器

$$f_{\mathbf{w},\mathbf{b},\mathbf{d}}(x)=\sum_{m=1}^{P}d_m<\mathbf{w}_m,\varphi_m(x)>+b$$

- 最大间距损失函数（类似于SVM）

$$\min_{\mathbf{w},\mathbf{d},b,\xi}\frac{1}{2}\sum_{m=1}^{P}d_m\|\mathbf{w}_m\|^2+C\sum_i\xi_i$$

$$\text{s.t.}\quad y_i(\sum_{m=1}^{P}d_m<\mathbf{w}_m,\varphi_m(x)>+b)\geq 1-\xi_i$$

$$\xi_i\geq 0\qquad d_m\geq 0\qquad \forall i$$

求解多核学习的最优化问题

- 对偶问题是带二次约束的二次规划问题
  - 可利用现成求解算法（如CVXOPT，Mosek，CPLEX)
- 转化为半无限线性规划问题（semi-infinite linear program，SILP）
  - 通过列生成（Column Generation）技术求解[Sonnenburg et al., 2005]
- 次梯度（Subgradient）下降 [Rakotomamonjy et al. 2007]
- ……

概览

- 集成学习理论
- 传统集成学习方法
- 基于多核学习的集成
- 独立性假设下的集成与依赖性建模
- 深度学习中的集成

独立性假设下的集成

- 乘积法

$$\Pr(\omega_l|\boldsymbol{x}_1,\dots\boldsymbol{x}_M) \propto \prod_{m=1}^{M} \Pr(\omega_l|\boldsymbol{x}_m)$$

- 均值法
  - 假设分类器为弱分类器：$\Pr(\omega_l|\boldsymbol{x}_m) = \Pr(\omega_l)\,(1+\delta_{lm})$

$$\Pr(\omega_l|\boldsymbol{x}_1,\dots\boldsymbol{x}_M) \propto \prod_{m=1}^{M} \Pr(\omega_l)\,(1+\delta_{lm})$$

$$\propto \sum_{m=1}^{M} \delta_{lm} \propto \sum_{m=1}^{M} \Pr(\omega_l|\boldsymbol{x}_m)$$

依赖性建模

- 加入依赖性项[Ma et al., TPAMI 2013]

$$\Pr(\omega_l|\boldsymbol{x}_1,\dots\boldsymbol{x}_M) \propto \prod_{m=1}^{M}\left[\Pr(\omega_l|\boldsymbol{x}_m) + \underbrace{\boxed{\alpha_{lm}}}_{\text{Dependency weight}}\underbrace{\boxed{\delta_{lm}}}_{\text{Small number}}\underbrace{\boxed{\Pr(\omega_l)}}_{\text{Prior}}\right]$$

- 利用解释函数表示联合后验概率[Ma et al., IJCV 2014]

$$\Pr(\omega_l|\boldsymbol{x}_1,\dots\boldsymbol{x}_M) \propto \sum_{k=0}^{\infty}\sum_{|n_1+\cdots+n_M|=k}\alpha_{(n_1,\dots,n_M)}\prod_{m=1}^{M}[\Pr(\omega_l|\boldsymbol{x}_m)]^{n_m}$$

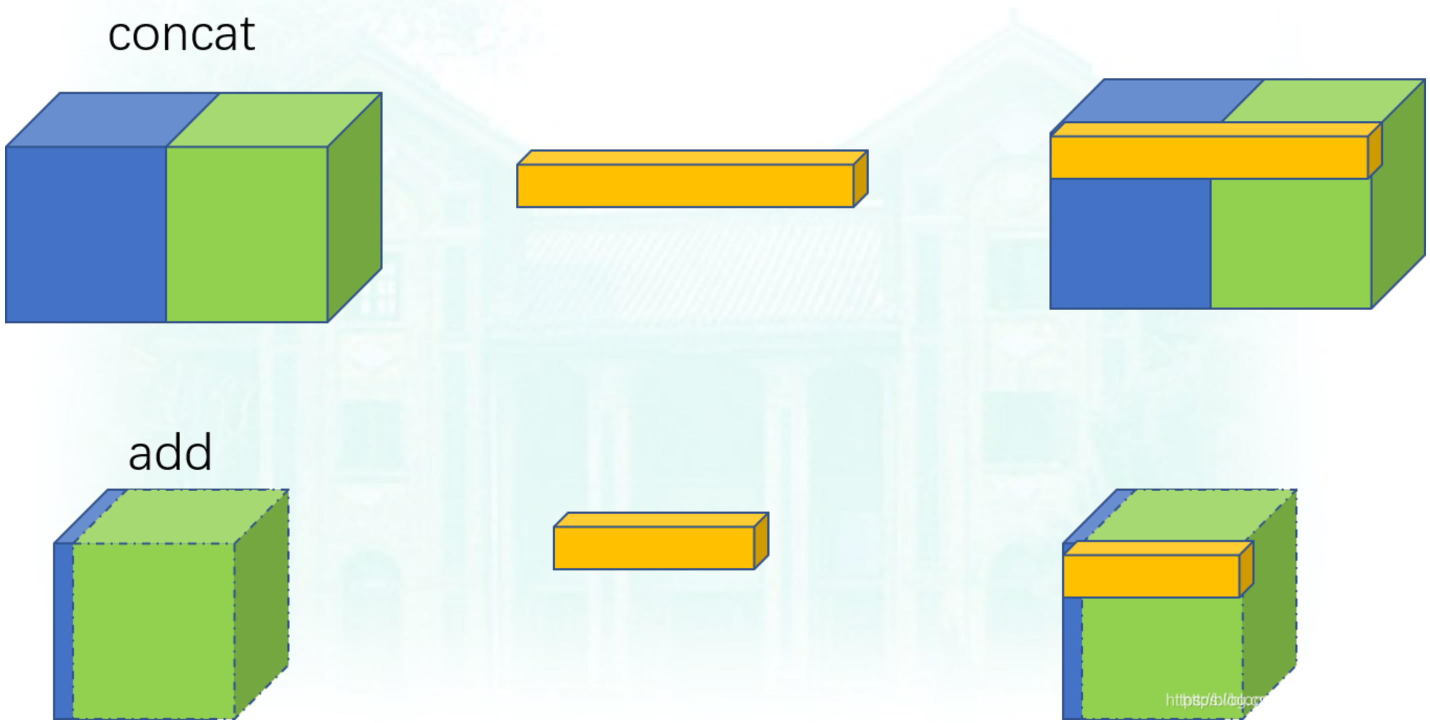
概览

- 集成学习理论
- 传统集成学习方法
- 基于多核学习的集成
- 独立性假设下的集成与依赖性建模
- 深度学习中的集成

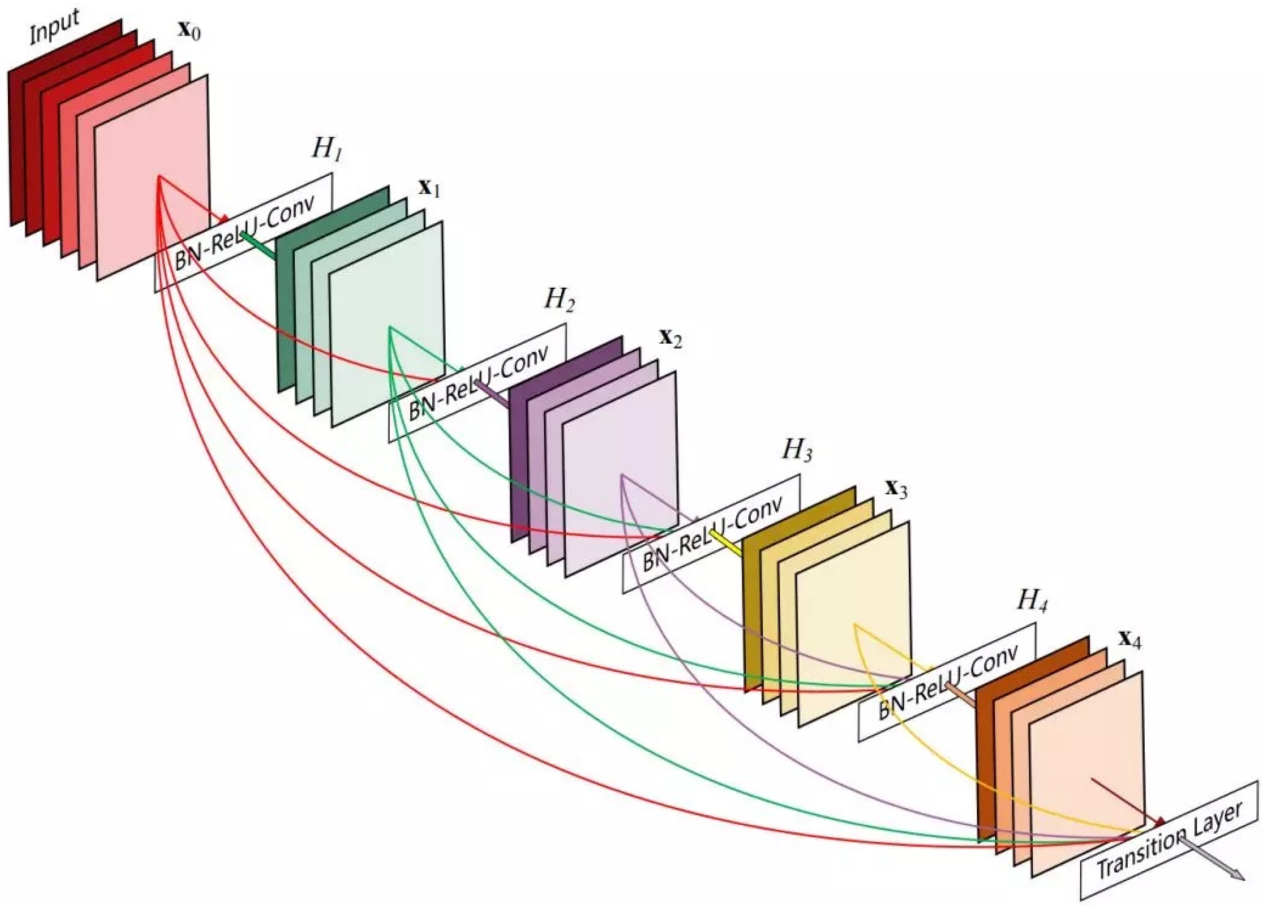
基本方法

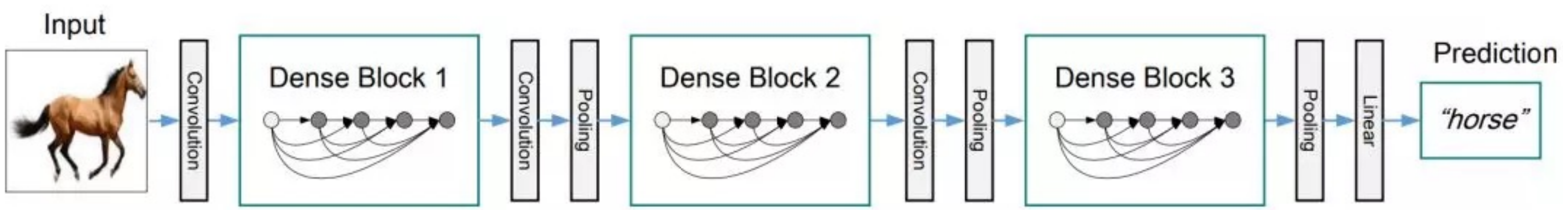
连接（concatenation）：通道数的增加

相加：特征图相加，通道数不变

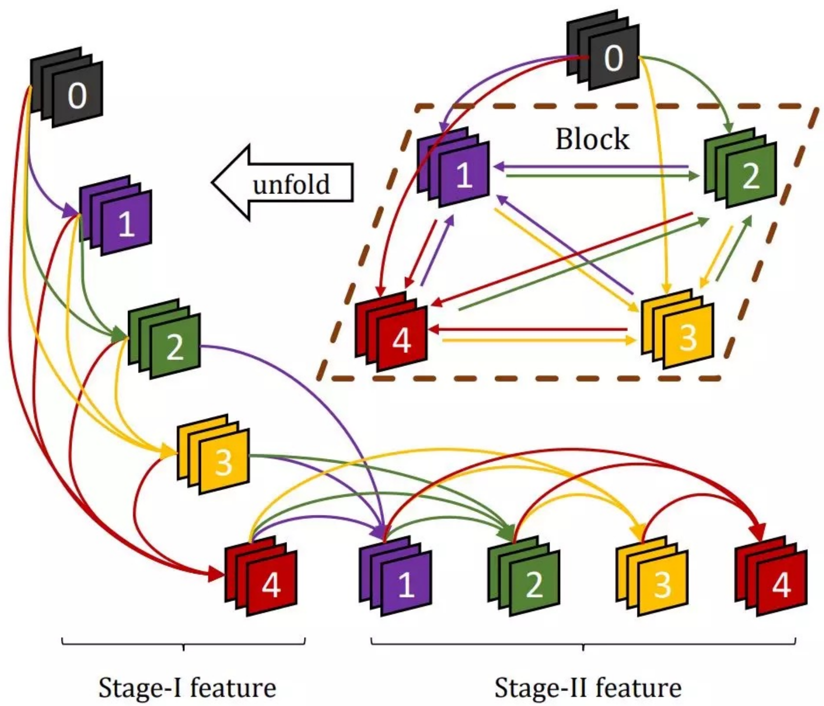


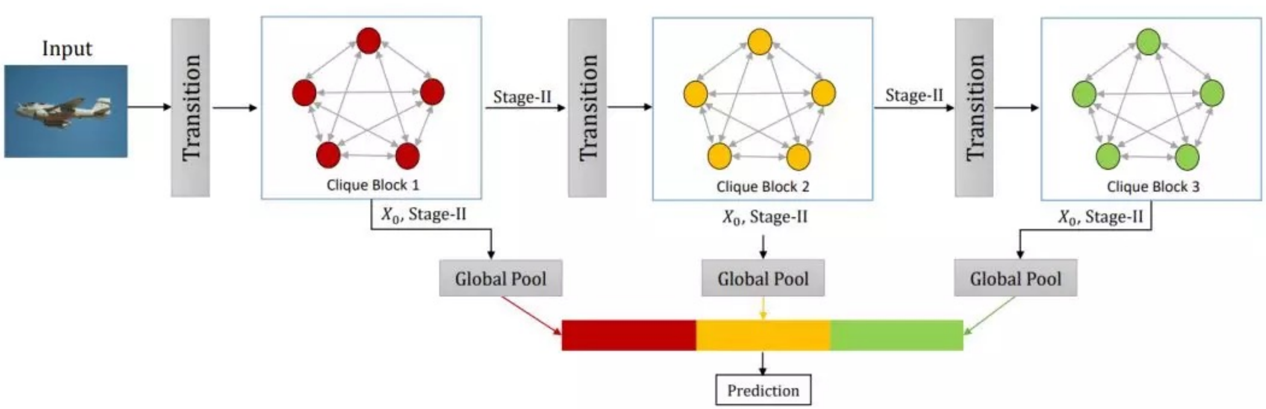
DenseNet [Huang et al., CVPR 2017]



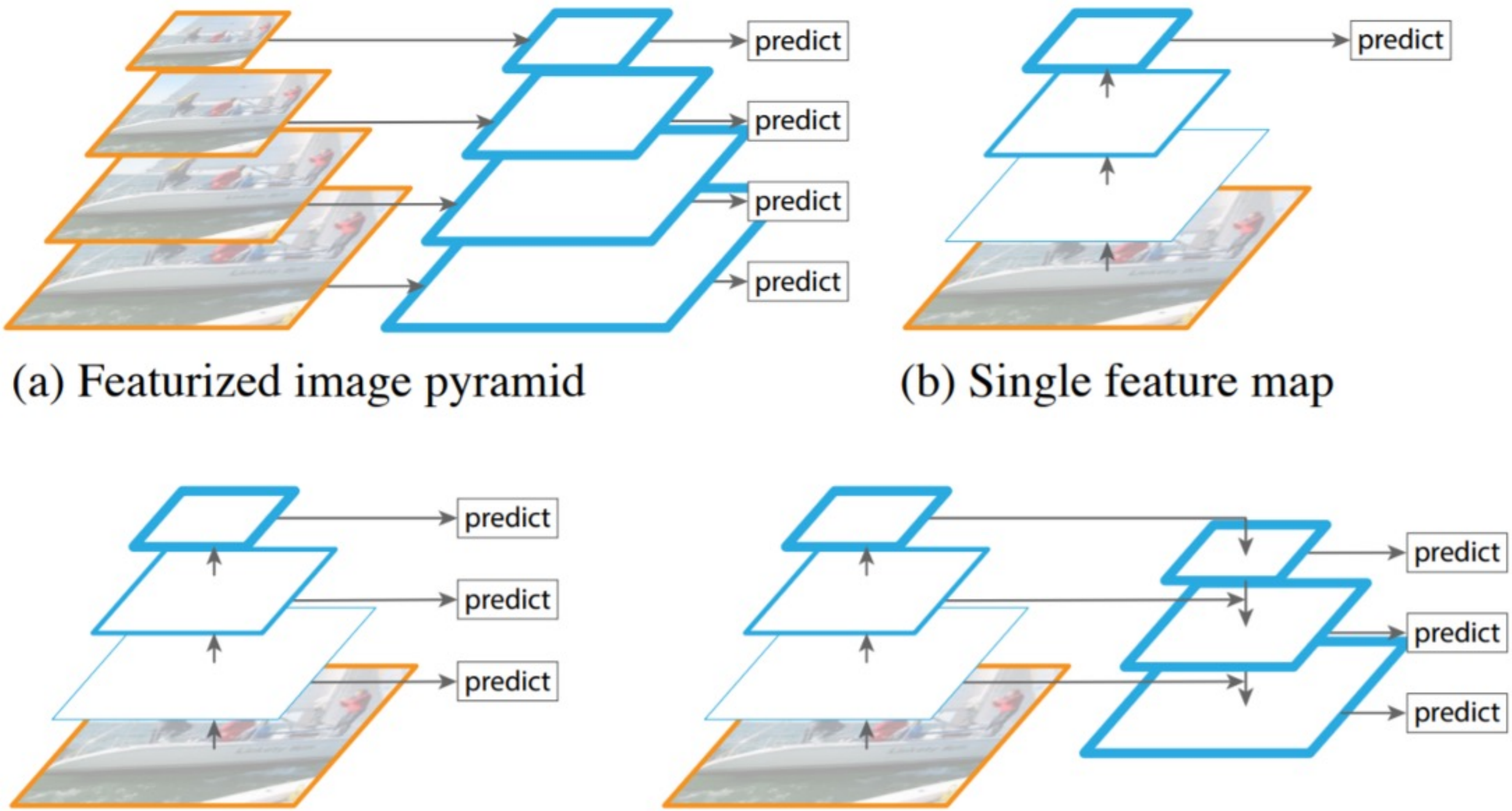


CliqueNet [Chen et al., CVPR 2018]





FPN [Lin et al., CVPR 2017]



(a) Featurized image pyramid

(b) Single feature map

(c) Pyramidal feature hierarchy

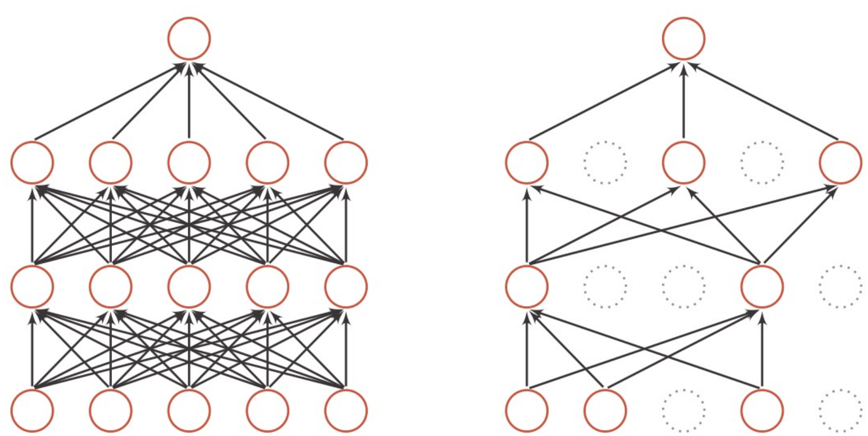
(d) Feature Pyramid Network

丢弃法（Dropout Method）

- 对于一个神经层$y = f(Wx + b)$，引入一个丢弃函数$d(\cdot)$使得$y = f(Wd(x) + b)$。

$$d(\mathbf{x}) = \begin{cases} \mathbf{m} \odot \mathbf{x} & \text{当训练阶段时} \\ p\mathbf{x} & \text{当测试阶段时} \end{cases}$$

其中$m \in \{0,1\}^d$ 是丢弃掩码，通过以概率为p的贝努力分布随机生成



(a) 标准网络　　(b) Dropout 后的网络

Dropout意义

- 集成学习的解释
  - 每做一次丢弃，相当于从原始的网络中采样得到一个子网络。如果一个神经网络有n个神经元，那么总共可以采样出2^n个子网络
- 贝叶斯学习的解释

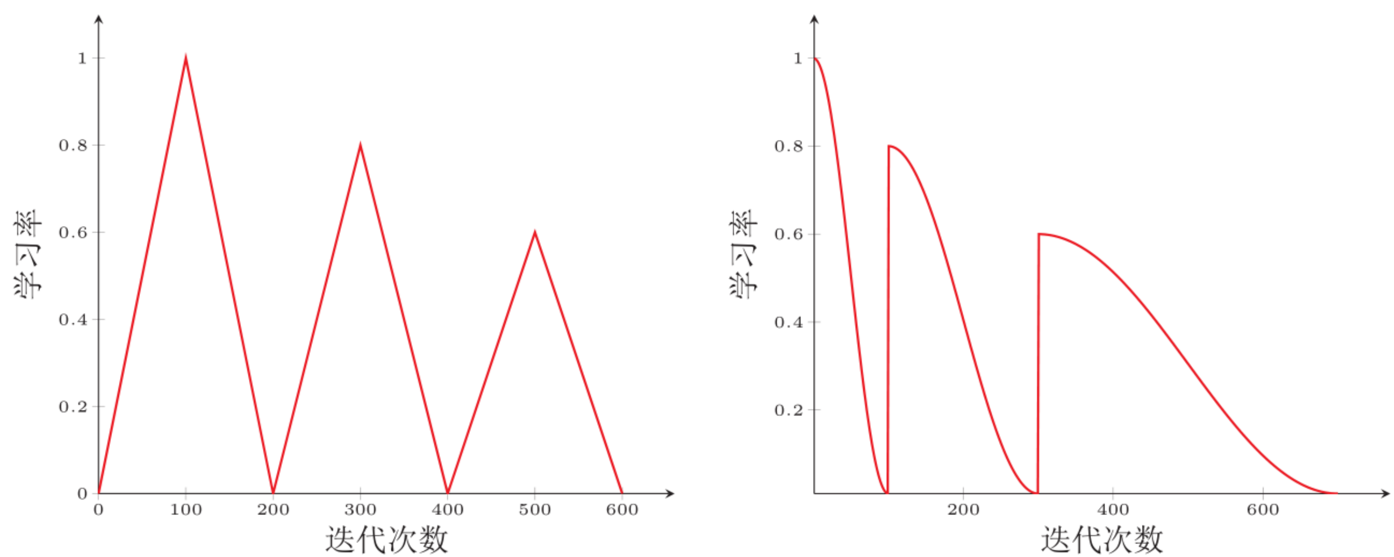
$$\mathbb{E}_{q(\theta)}[y] = \int_q f(\mathbf{x}, \theta) q(\theta) d\theta$$

$$\approx \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}, \theta_m),$$

其中$f(x,\theta_m)$为第m次应用丢弃方法后的网络

周期性(Cyclical)学习率调整

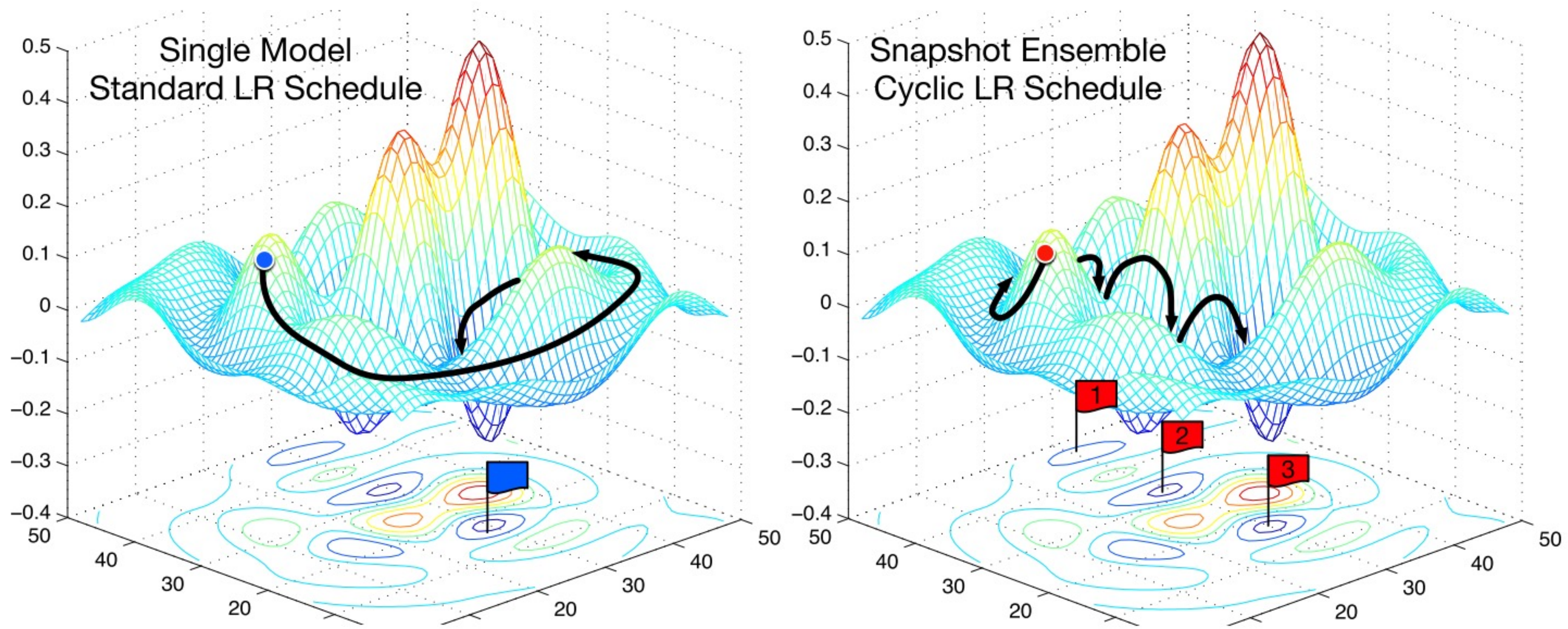
- 当参数处于尖锐最小值附近时，增大学习率有助于逃离尖锐最小值
- 当参数处于平坦最小值附近时，增大学习率依然有可能在该平坦最小值的吸引域（Basin of Attraction）内



(a) 三角循环学习率　　(b) 带热重启的余弦衰减

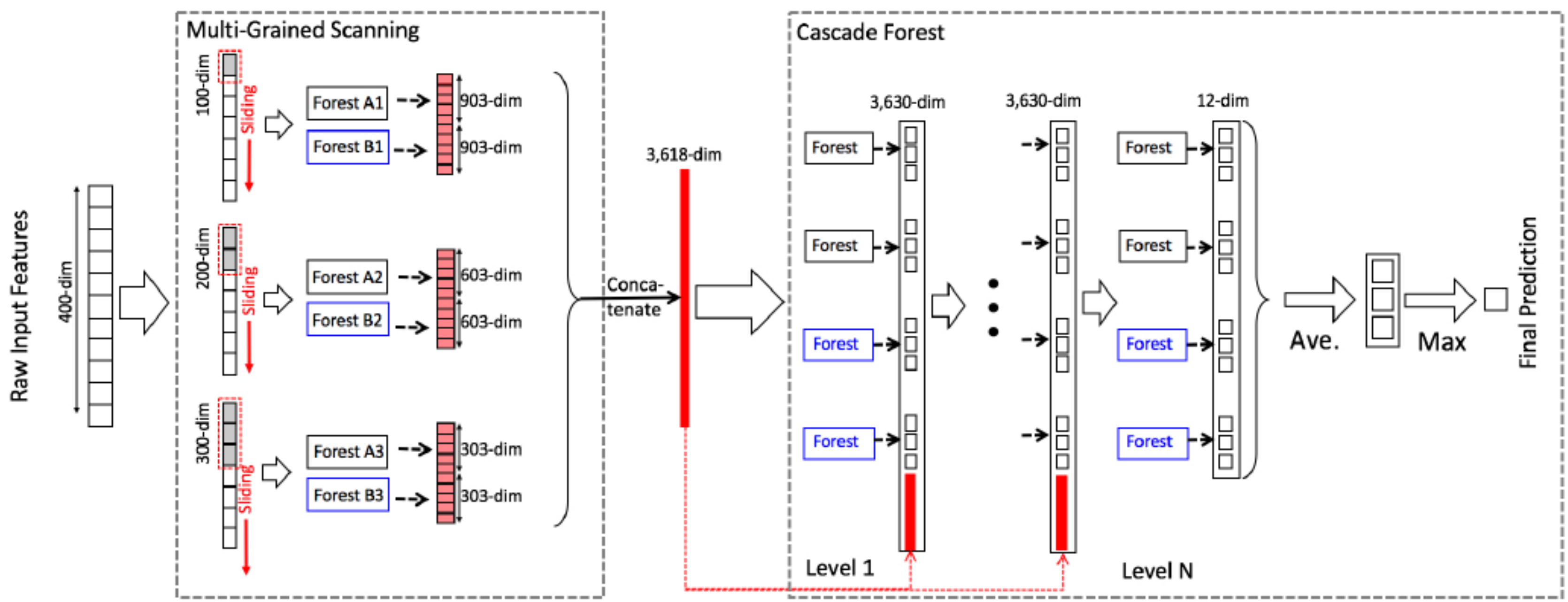
Cyclical Learning Rates [Huang et al.,2017]

- Left: converge to a minimum at the end
- Right: converges to and escape from multiple local minima for test-time ensemble



Deep Forest

- 对小数据分析的鲁棒性比深度学习更好



https://doi.org/10.48550/arXiv.1702.08835