第9章　集成学习

刘家锋

哈尔滨工业大学

第9章　集成学习

9.1 分类器性能评价

性能评价指标

- **分类错误率**
  - 分类器在样本集$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$上的错误率

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

其中，$\mathbb{I}(\alpha)$为指示函数：

$$\mathbb{I}(\alpha) = \begin{cases} 1, & \alpha = \text{true} \\ 0, & \alpha = \text{false} \end{cases}$$

性能评价指标

- **查准率和查全率**
  - 检索任务可以看作是两分类问题，相关内容称为正例，无关内容称为反例
  - 分类的结果可以用混淆矩阵表示
    - 真正例(TP)：真实的正例被分类为正例的数量
    - 假反例(FN)：真实的正例被分类为反例的数量
    - 假正例(FP)：真实的反例被分类为正例的数量
    - 真反例(TN)：真实的反例被分类为反例的数量

真实类别	预测结果	
	正例	反例
正例	TP(真正例)	FN(假反例)
反例	FP(假正例)	TN(真反例)

性能评价指标

- **检索任务的性能评价指标**
  - 查准率：检索出来的结果中，真实正例所占的比例

$$P = \frac{TP}{TP + FP}$$

  - 查全率：所有真实正例中，被检索出来的比例，也被称为召回率

$$R = \frac{TP}{TP + FN}$$

  - 查全率和查准率是相互矛盾的，一般与检出的总数有关
    - 检索结果数量多，查全率高，查准率低
    - 检索结果数量少，查全率低，查准率低

性能评价指标

- **检索任务的性能指标**
  - F_1度量：综合考虑查全率和查准率，取两者的几何平均

$$F_1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{n + TP - TN}$$

  - F_β度量：通过参数β调节对查全率和查准率的关心程度

$$F_\beta = \frac{(1 + \beta^2) \times P \times R}{(\beta^2 \times P) + R}$$

$\beta > 1$更关心查全率，$\beta < 1$更关心查准率；

性能评价的方法

- **训练集和测试集**
  - 评价最好的方法是使用训练集S学习模型，使用另外的测试集T评估模型的性能
  - 数据集D既用于训练，又用于测试，需要适当的划分

$$D = S \cup T, \qquad S \cap T = \Phi$$

- **常用的方法**
  - 留出法(hold-out)
  - 交叉验证法(cross validation)
  - 自助法(bootstrap)

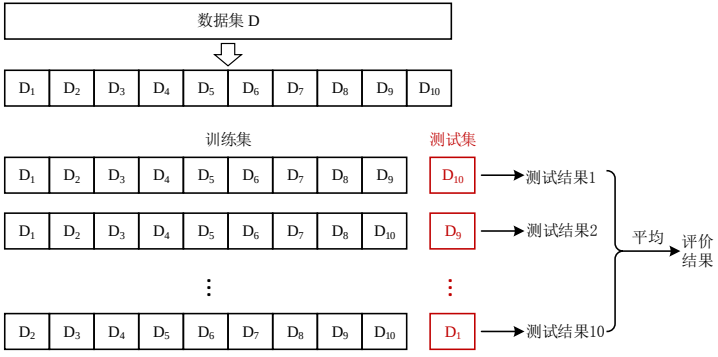
留出法

- **数据集的划分原则**
  - 保持数据分布一致性，不同类别样本的比例应该是一致的
  - 测试集不宜太大或太小，一般选择样本总数的20% ~ 30%

- **评价过程**
  - 按比例将数据集D随机划分为S和T
  - 训练集S学习模型，T测试模型的性能
  - 重复划分和测试若干次，计算平均性能

交叉验证法

- **k-折交叉验证法**
  - 数据集D随机划分为k个子集
  - $k-1$个子集用于训练，1个用于测试



自助法(Bootstrap)

- **自助法的过程**
  - 从数据集D中有放回地随机抽样n个样本，构成训练集S
  - 从数据集D中有放回地随机抽样n个样本，构成测试集T
  - 重复若干次，计算平均的评估结果
- **自助法的优点**
  - 数据集D中约有30%的样本没有出现在训练集S中

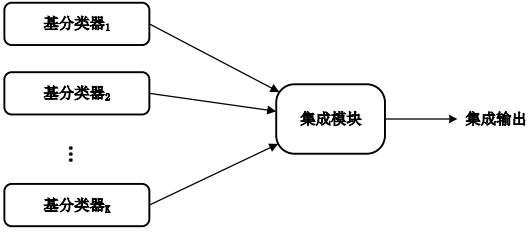
$$P(\mathbf{x} \notin S) \approx \lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.368, \quad \forall \mathbf{x} \in D$$

  - S和T的样本数与D相同，测试可以进行任意多次
  - 评估结果更接近最终模型的性能
  - 自助法更适用于样本数n较小时，样本数多时留出法和交叉验证更常用

9.2 分类器集成

多分类器集成

- **个体与集成**
  - 个体分类器：由一个现有的学习算法从训练数据产生，也称为“基分类器”
  - 集成学习：组合多个个体分类器，取得比个体分类器更好的性能，也称为“多分类器系统”等



集成分类器的性能

- **集成学习如何能够获得更好的性能？**
  - 要求基分类器有一定的“准确性”和“多样性”
  - 例如：三个不同的分类器g_1, g_2, g_3，在三个不同测试样本$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$上的分类结果
  - 采用多数投票的方式集成分类器，性能可能提升也可能下降

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
g_1	✓	✓	×
g_2	×	✓	✓
g_3	✓	×	✓
集成	✓	✓	✓

(a) 性能提升

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
g_1	✓	✓	×
g_2	✓	✓	×
g_3	✓	✓	×
集成	✓	✓	×

(b) 性能不变

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
g_1	✓	×	×
g_2	×	✓	×
g_3	×	×	✓
集成	×	×	×

(c) 性能下降

基分类器的生成

- **多样性学习算法**
  - 相同的训练样本集
  - 不同的学习算法，包括学习算法的不同超参数设置
- **多样性训练集**
  - 相同的学习算法，不同的训练样本集
  - 随机划分训练集，或者Bootstrap抽样训练集

9.3 Bagging和随机森林

Bagging算法

Algorithm 1 Bagging算法

Input: 训练集$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$，基分类器学习算法$\mathcal{L}$，基分类器数量$K$

Output: 集成分类器$G(\mathbf{x})$

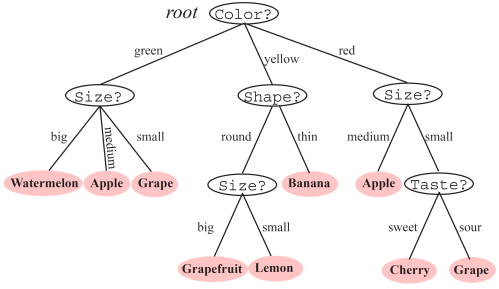
1: **procedure** BAGGING
2: **for** $k = 1, \cdots, K$ **do**
3: $D_{bs} = bootstrap(D)$ ▷ bootstrap抽样
4: $g_k = \mathcal{L}(D_{bs})$ ▷ 训练基学习器
5: **end for**
6: **return** $G(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{k=1}^{K} \mathbb{I}(g_k(\mathbf{x}) = y)$ ▷ 集成预测
7: **end procedure**

决策树

- **决策树**
  - 决策树是一种常用的模式分类方法，适用于离散特征分类问题，也可用于连续特征分类
  - 以决策树作为基分类器的Bagging算法称为随机森林
- **决策树的结构**
  - 叶节点：对应一个类别
  - 中间节点：对应某个特征，节点下的分支对应该特征的某个取值

决策树分类

- **决策树的识别过程**
  - 从根节点开始，测试节点的特征，根据待识样本的特征，决定下行分支
  - 在叶节点得到样本的类别
  - 决策树的每个分支对应一个决策规则



$(\text{color} = \text{green}) \wedge (\text{size} = \text{big}) \longrightarrow \text{Watermelon}$

$\cdots \cdots$

$(\text{color} = \text{red}) \wedge (\text{size} = \text{medium}) \longrightarrow \text{Apple}$

$(\text{color} = \text{red}) \wedge (\text{size} = \text{small}) \wedge (\text{taste} = \text{sweet}) \longrightarrow \text{Cherry}$

$(\text{color} = \text{red}) \wedge (\text{size} = \text{small}) \wedge (\text{taste} = \text{cour}) \longrightarrow \text{Grape}$

判定树的学习问题

- **学习问题**
  - 给定训练样本集合：$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$
  - 给定特征集合：$A = \{a_1, \cdots, a_d\}$
  - 构造判定树，既能够分类训练集D中的样本，也能够分类未来的测试样本$\mathbf{x}$
- **判定树的构造**
  - 采用“分而治之”的策略，在每个节点选择一个特征，将训练集分成若干子集，每个子集对应一个特征取值
  - 直到子集中只包含属于某一类的样本为止，将其作为叶节点，标注为该类别

基本算法

Algorithm 2 判定树学习算法

Input: 训练集$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$，特征集$A = \{a_1, \cdots, a_d\}$
Output: 以node为根节点的一棵决策树

function TreeGenerate(D, A)
    生成节点node;
    if 满足递归停止条件 **then** 标记节点；**return**
    end if
    从A中选择最优划分特征a_*
    for a_*的每一个值a_*^v **do**
        为node生成一个分支，D_v表示D中a_*取值a_*^v的样本子集
        if $D_v = \Phi$ **then**
            分支标记叶节点为D中样本最多类；**return**
        else
            以TreeGenerate($D_v, A \setminus \{a_*\}$)为分支节点
        end if
    end for
end function

判定树的构造

- **判定树学习算法**
  - 判定树的构造是一个递归的过程
  - 递归的终止条件：
    1. 输入样本集都属于同一类别，节点标记为该类别
    2. 输入属性集为空，或样本的所有属性相同，标记为样本最多的类别
  - 判定树的构造，关键是如何选择“最优”的特征a^*

最优特征

- **划分特征选择的原则**
  - 判定树根节点对应的样本集合包含所有类别的样本，而叶节点只包含某一个类别的样本
  - 判定树的构建过程可以看作是一个使得样本集合越来越“纯净”的过程
  - 如果能够定义一个度量集合“纯度”的指标，在每个节点上应该选择使得“纯度”增加最快的特征来划分样本集合

Information Entropy

- **信息熵**
  - 信息熵是度量样本集合纯度最常用的一个指标
  - 样本集合D中第k类样本所占比例为p_k，D的信息熵定义为

$$\mathsf{Ent}(D) = -\sum_{k=1}^{c} p_k \log_2 p_k$$

  - $\mathsf{Ent}(D)$的值越小，表示纯度越高，当所有样本属于一个类别时，取得最小值$\mathsf{Ent}(D) = 0$；（约定$p = 0$，$p \log_2 p = 0$）
  - $\mathsf{Ent}(D)$的最大值为$\log_2 c$

ID3算法

- **信息增益**
  - 离散特征a有V个可能的取值：$\{a^1, \cdots, a^V\}$
  - D^v表示D中特征$a = a^v$的样本集合，以特征a对数据集D进行划分的信息增益为

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

  - 第1项$\mathsf{Ent}(D)$是划分前数据集的信息熵
  - 第2项是使用特征a划分之后的信息熵，$|D^v|/|D|$是每个分支的权重，样本越多的分支权重越大
  - ID3算法依据信息增益选择最优的划分特征

$$a_* = \arg\max_{a \in A} \mathsf{Gain}(D, a)$$

CART算法

- **基尼指数**
  - 基尼值是样本集D纯度的另外一种度量

$$\text{Gini}(D) = \sum_{k=1}^{c} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{c} p_k^2$$

  - 基尼值反映了从样本集D中随机抽取两个样本，其类别不一致的概率
  - 基尼值越小纯度越高，越大纯度越低
  - CART算法选择基尼指数最小的特征作为划分特征

$$\text{Gini\_index}(D, a) = \sum_{v=1}^{V} \frac{|D^v|}{|D|} \text{Gini}(D^v)$$

连续值特征

- **连续值的处理**
  - 基本思路：连续特征离散化
  - 二分法：连续特征a设定划分点t，将样本集D划分为两个子集，$a \geq t$的样本作为D_t^+，$a < t$的样本作为D_t^-
  - 特征a在D中有m个取值，由小到大排列为$\{a^1, \cdots, a^m\}$
  - $t \in [a^i, a^{i+1})$的任意取值，划分效果相同，包含$m-1$个候选划分点的集合

$$T_a = \left\{ \frac{a^i + a^{i+1}}{2} \mid 1 \leq i \leq m-1 \right\}$$

连续值特征

- **连续值特征的信息增益**
  - 特征a以t为划分点的信息增益

$$\mathsf{Gain}(D,a,t)=\mathsf{Ent}(D)-\sum_{\lambda\in\{-,+\}}\frac{|D_t^\lambda|}{|D|}\mathsf{Ent}(D_t^\lambda)$$

  - 选择最优的划分点t来划分，特征a的信息增益

$$\begin{aligned}\mathsf{Gain}(D,a)&=\max_{t\in T_a}\mathsf{Gain}(D,a,t)\\&=\max_{t\in T_a}\left\{\mathsf{Ent}(D)-\sum_{\lambda\in\{-,+\}}\frac{|D_t^\lambda|}{|D|}\mathsf{Ent}(D_t^\lambda)\right\}\end{aligned}$$

- **连续值特征的选择**
  - 每个节点按照$\mathsf{Gain}(D,a)$来选择划分特征
  - 与离散特征不同，使用过的连续特征仍可作为后代节点的划分特征

随机森林

- **随机森林(Random Forrest)**
  - 以决策树作为基分类器，采用bagging算法集成学习
  - 为了增加基分类器的多样性，除了Bootstrap抽样训练集的随机性之外，增加了划分特征的随机性
  - 对于每个节点，先从特征集合中随机选择包含k个元素的子集，在子集中选择最优特征用于划分
  - 决策树学习过程中，一般不做剪枝处理

例9.1 随机森林

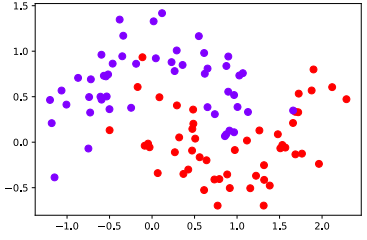
- **仿真数据实验**
  - 生成100个两分类随机数据，数据分布在2维空间的两个交叠半圆形区域
  - 随机划分数据集为训练集和测试集
  - 学习包含5个决策树分类器器的随机森林
  - 测试每个决策树以及集成的随机森林在训练集和测试集上的分类正确率
  - 显示每个决策树以及随机森林的分类边界

生成仿真数据

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X,y,stratify=y,random_state=42)

plt.scatter(X[:,0],X[:,1], c=y, s=50, cmap='rainbow')
plt.show()
```



学习随机森林分类器

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=5, random_state=79)
forest.fit(X_train, y_train)

for tree in forest.estimators_:
    print(tree)
```

```
DecisionTreeClassifier(max_features='sqrt', random_state=2869591)
DecisionTreeClassifier(max_features='sqrt', random_state=1715508413)
DecisionTreeClassifier(max_features='sqrt', random_state=2010334282)
DecisionTreeClassifier(max_features='sqrt', random_state=1982403395)
DecisionTreeClassifier(max_features='sqrt', random_state=3360916)
```

测试随机森林

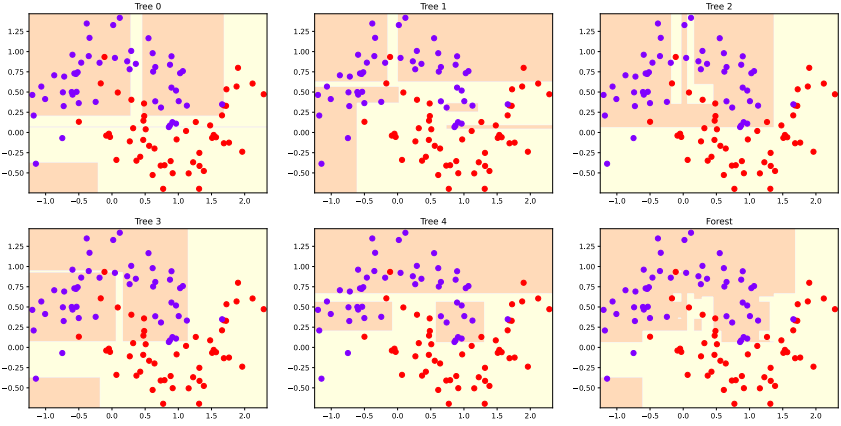
```
from plot_decision_boundary import plot_decision_boundary

fig, axes = plt.subplots(2, 3, figsize=(18, 8))
for ax, tree, i in zip(axes.reshape([6,1]),forest.estimators_,range(5)):
    plot_decision_boundary(tree,axis=[x_min,x_max,y_min,y_max],ax=ax[0])
    ax[0].scatter(X[:,0], X[:,1], c=y.reshape(-1,1), s=50, cmap='rainbow')
    ax[0].set_title("Tree %d" %(i))
    print("Tree %d: train acc = %4.3f, test acc = %4.3f" \
          % (i,tree.score(X_train,y_train),tree.score(X_test,y_test)))

plot_decision_boundary(forest,axis=[x_min,x_max,y_min,y_max],ax=axes[-1,-1])
axes[-1,-1].scatter(X[:,0], X[:,1], c=y.reshape(-1,1), s=50, cmap='rainbow')
axes[-1,-1].set_title( "Forest")
print("Forest: train acc = %4.3f, test acc = %4.3f" \
      % (forest.score(X_train,y_train),forest.score(X_test,y_test)))

plt.show()
```

测试随机森林




```
Tree 0: train acc = 0.893, test acc = 0.840
Tree 1: train acc = 0.907, test acc = 0.920
Tree 2: train acc = 0.907, test acc = 0.920
Tree 3: train acc = 0.907, test acc = 0.880
Tree 4: train acc = 0.947, test acc = 0.920
Forest: train acc = 0.973, test acc = 0.960
```

例9.2 随机森林分类Breast Cancer数据集

- **Breast Cancer Dataset**
  - University of Wisconsin的乳腺肿瘤数据
  - 原始属性数据来自于肿瘤细胞核图像的测量数据
    - 半径，纹理，周长，面积
    - 平滑性，紧致性，凹度，凹点数，对称性，分形维度
  - 由原始属性的均值、最大值、最小值等衍生出30维分类属性
  - 类别属性
    - Malignant：恶性
    - Benign：良性

例9.2 随机森林分类Breast Cancer数据集

- **数据集**
  - Breast Cancer数据集，分类良性、恶性细胞
  - 属性：30维
  - 数据随机划分为训练集和测试集
- **对比随机森林和决策树**
  - 分别使用训练集，学习决策树和随机森林
  - 随机森林集成100棵随机树
  - 分别测试决策树和随机森林在训练集和测试集上分类正确率

数据准备

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_csv("BreastCancer.csv")

class_mapping = {'M':0,'B':1}
data['diagnosis'] = data['diagnosis'].map(class_mapping)

X = data.iloc[:,2:-1].to_numpy(); y = data.iloc[:,1].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

print("Shape of train data:", X_train.shape)
print("Shape of test data:", X_test.shape)
```

```
Shape of train data: (426, 30)
Shape of test data: (143, 30)
```

学习和测试决策树

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=7,random_state=0,criterion='entropy')
tree.fit(X_train, y_train)

print( "Decision Tree Classifier:")
print("\t Accuracy on training set: ", tree.score(X_train, y_train))
print("\t Accuracy on test set: ", tree.score(X_test, y_test))
```

```
Decision Tree Classifier:
        Accuracy on training set:  1.0
        Accuracy on test set:  0.958041958041958
```

学习和测试随机森林

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)

print( "Random Forest Classifier:")
print("\t Accuracy on training set: ", forest.score(X_train, y_train))
print("\t Accuracy on test set: ", forest.score(X_test, y_test))
```

```
Random Forest Classifier:
        Accuracy on training set:  1.0
        Accuracy on test set:  0.972027972027972
```

9.4 Boosting

序列化集成学习方法

- **Boosting**
  - Bagging算法属于并行化的集成学习方法，基分类器之间互无关联
  - Boosting算法是序列化的集成学习，下一个基分类器的学习需要根据之前的学习结果来调整
  - 这其中最有代表性的是AdaBoost算法
- **AdaBoost集成分类器**
  - AdaBoost一般用于二分类问题，类别标记$y_i \in \{-1, +1\}$
  - 学习K个基分类器$\{h_k(\mathbf{x})\}$，加权线性组合

$$H(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})$$

  - 基分类器的权重α_k是算法学习得到的参数

AdaBoost算法

- **样本集的加权重采样**
  - AdaBoost算法中基分类器的训练集也是由样本集D重采样得到的
  - 样本是按照不断变化的分布，而不是均匀分布来抽样的
  - 简单地说，为每个样本赋予一个权重w_i，以w_i为概率决定是否抽样相应样本
  - 学习得到一个新的基分类器$h_k(\mathbf{x})$之后，对权重调整，被正确分类的样本权重降低，错误分类的权重增加

$$w_i \leftarrow \frac{w_i}{z_k} \times \begin{cases} e^{-\alpha_k}, & h_k\text{正确分类}\mathbf{x}_i \\ e^{\alpha_k}, & h_k\text{错误分类}\mathbf{x}_i \end{cases}$$

其中，z_k为归一化因子，保证$\sum_{i=1}^{n} w_i = 1$

AdaBoost算法

- **基分类器的权重**
  - 基分类器$h_k(\mathbf{x})$的权重α_k体现分类性能，与错误率ϵ_k有关
  - 错误率ϵ_k同样是在一个加权重采样样本集上统计的，而不是初始训练集D
  - 令依据权重$\{w_i\}$抽样的数据集为D_ϵ，错误率为

$$\epsilon_k = \frac{1}{n} \sum_{\mathbf{x} \in D_\epsilon} \mathbb{I}(h_k(\mathbf{x}) \neq y)$$

  - 基分类器$h_k(\mathbf{x})$的权重

$$\alpha_k = \frac{1}{2} \ln \left(\frac{1 - \epsilon_k}{\epsilon_k} \right)$$

AdaBoost算法

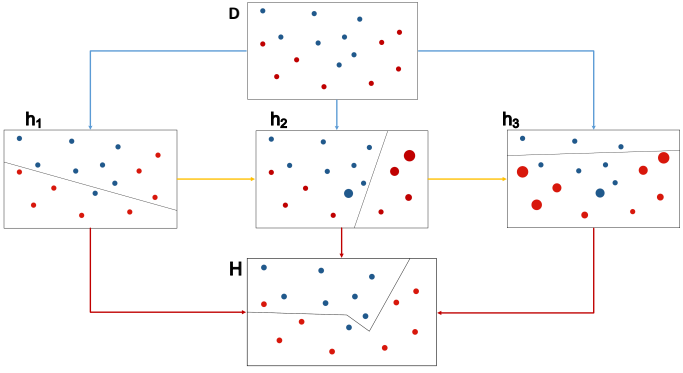
Algorithm 3 AdaBoost算法

Input: 训练集$D = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$，基分类算法$\mathcal{L}$，基分类器数量$K$
Output: 集成分类器$H(\mathbf{x})$

procedure AdaBoost
  $w_i = 1/n, i = 1, \cdots, n$ ▷ 初始化样本权重
  for $k = 1, \cdots, K$ **do**
    $D_w = bootstrap(D, \{w_i\})$ ▷ 加权抽样
    $D_\epsilon = bootstrap(D, \{w_i\})$
    $h_k = \mathcal{L}(D_w)$ ▷ 训练基分类器
    $\epsilon_k = \frac{1}{n}\sum_{\mathbf{x}\in D_\epsilon} \mathbb{I}(h_k(\mathbf{x}) \neq y)$ ▷ 计算分类误差
    if $\epsilon_k > 0.5$ **then break**
    $\alpha_k = \frac{1}{2}\ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right), w_i \leftarrow \frac{w_i}{z_k} \times \begin{cases} e^{-\alpha_k}, & h_k\text{正确分类}\mathbf{x}_i \\ e^{\alpha_k}, & h_k\text{错误分类}\mathbf{x}_i \end{cases}$
  end for
  return $H(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^{K} \alpha_k h_k(\mathbf{x})\right)$ ▷ 集成预测
end procedure

AdaBoost算法

- **AdaBoost示例**
  - AdaBoost算法学习3个基分类器的集成，基分类器采用线性分类器



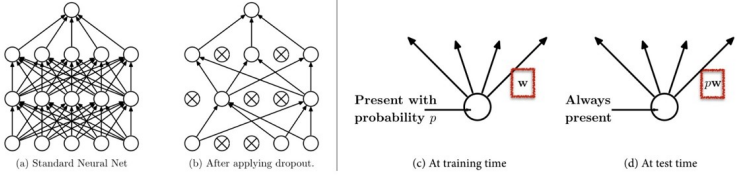
9.5 神经网络的集成

神经网络的集成方法

- **一般的集成方法**
  - Bagging算法生成多个神经网络，集成网络的分类结果
  - 设置不同的网络层数、不同的隐含层神经元数量，可以学习得到不同结构的网络
  - 相同结构的网络，设置不同的参数初始值，不同的学习率、收敛条件，可以得到不同参数的网络
  - 随机梯度学习算法每次学习，产生不同的网络学习结果
- **直接集成的缺点**
  - 计算效率低，神经网络的训练和预测耗时长，难以应用于大数据集训练的深度神经网络
  - 集成神经网络的数量不会很多

Drop Out方法

- **学习时**
  - 每一轮迭代，以一定的概率p（如0.5）随机地保留网络中的一部分神经元，屏蔽其它神经元
  - 被屏蔽的神经元，在前馈和反馈中都不起作用
- **分类时**
  - 每个神经元的输出乘以p



Drop Out的解释

- **学习时**
  - 相当于随机抽取了一个网络进行学习
  - 每一次抽取的网络权值是共享的，但连接结构是随机的
- **分类时**
  - 可以想象为，按照同样的方式随机地抽取一个网络，计算神经网络的输出
  - 重复随机抽取无穷多个网络，所有网络的输出结果以几何平均的方式集成