Extreme Learn Machines
(极限学习机)

—— Python 实现

Outline

1. ELM简介
2. ELM原理
3. Python实现
4. 总结

ELM简介

极限学习机(Extreme Learning Machine) ELM，是由黄广斌教授提出来的**求解单隐层神经网络**的算法。ELM最大的特点是对于传统的神经网络，尤其是单隐层前馈神经网络，在保证学习精度的前提下比传统的学习算法速度更**快**。

该算法在网络参数的确定过程中,隐层节点参数**随机**选取, 在训练过程中无需调节，只需要设置隐含层神经元的个数，便可以获得唯一的最优解;而网络的外权（即输出权值）是通过最小化平方损失函数得到的**最小二乘解**。这样网络参数的确定过程中**无需任何迭代**步骤,从而大大降低了网络参数的调节时间。

ELM原理

针对训练数据样本x,具有 L 个隐层神经元的单隐层前向神经网络的输出函数表达式为:

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}) \qquad (1)$$

针对加法型的隐层节点

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i) \qquad (2)$$

其中ai是输入权重, bi是隐层单元的偏置

针对径向基函数神经网络的隐层节点

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \| \mathbf{x} - \mathbf{a}_i \|) \qquad (3)$$

ai和bi (bi> 0) 分别表示第 i 个径向基函数 (RBF) 节点的中心和影响因子

SLFN（单隐层前馈神经网络）

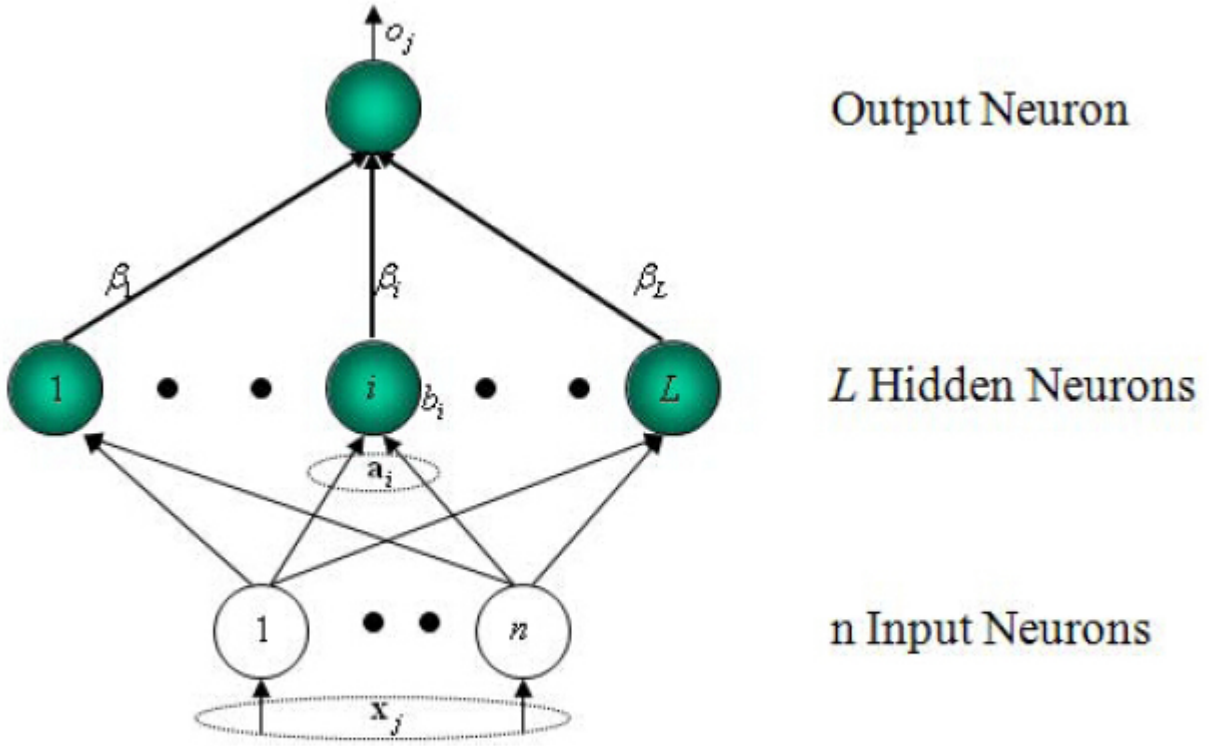


Figure 1: SLFN: additive hidden nodes

对于一个单隐层神经网络，假设有N个任意的样本 (X_i, t_i)，其中 $X_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in R^n$，$t_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^T \in R^m$，对于一个有L个隐层节点的单隐层神经网络可以表示为:

$$\sum_{i=1}^{L} \beta_i g(W_i \cdot X_j + b_i) = o_j, \ j = 1, \cdots, N \quad (1)$$

其中，g(x) 为激活函数，$W_i = [w_{i,1}, w_{i,2}, \cdots, w_{i,n}]^T$ 为输入权重，β_i 为输出权重，b_i 是第个隐层单元的偏置。

单隐层神经网络学习的目标是使得输出的误差最小，可以表示为

$$\sum_{j=1}^{N} \|o_j - t_j\| = 0 \qquad (1)$$

即存在 β_i W_i, b_i 和 ，
使得

$$\sum_{i=1}^{L} \beta_i g\left(W_i \cdot X_j + b_i\right) = t_j,\ j = 1, \cdots, N \qquad (2)$$

可以矩阵表示为 $\mathbf{H}\beta = \mathbf{T}$ (3)

其中，H 是隐层节点的输出，β 为输出权重，T 为期望输出。

可以矩阵表示为 $$\mathbf{H}\beta = \mathbf{T} \quad (3)$$

其中，H 是隐层节点的输出，β 为输出权重，T 为期望输出。

$$H(W_1, \cdots, W_L, b_1, \cdots, b_L, X_1, \cdots, X_L)$$

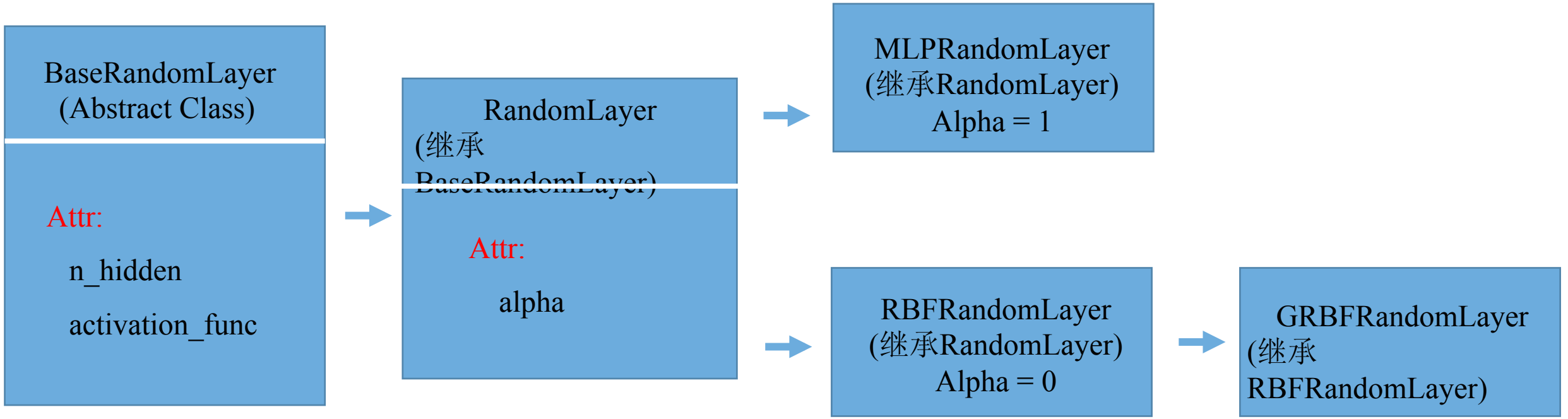
$$= \begin{bmatrix} g(W_1 \cdot X_1 + b_1) & \cdots & g(W_L \cdot X_1 + b_L) \\ \vdots & \cdots & \vdots \\ g(W_1 \cdot X_N + b_1) & \cdots & g(W_L \cdot X_N + b_L) \end{bmatrix}_{N \times L}$$

$$T = \begin{bmatrix} T_1^T \\ \vdots \\ T_N^T \end{bmatrix}_{N \times m} \qquad \beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m}$$

经推导可得 $$\hat{\beta} = \arg\min_{\beta} \| \mathbf{H}\beta - \mathbf{T} \| = \mathbf{H}^{\dagger}\mathbf{T} \quad (4)$$

Python实现

random_layer.py

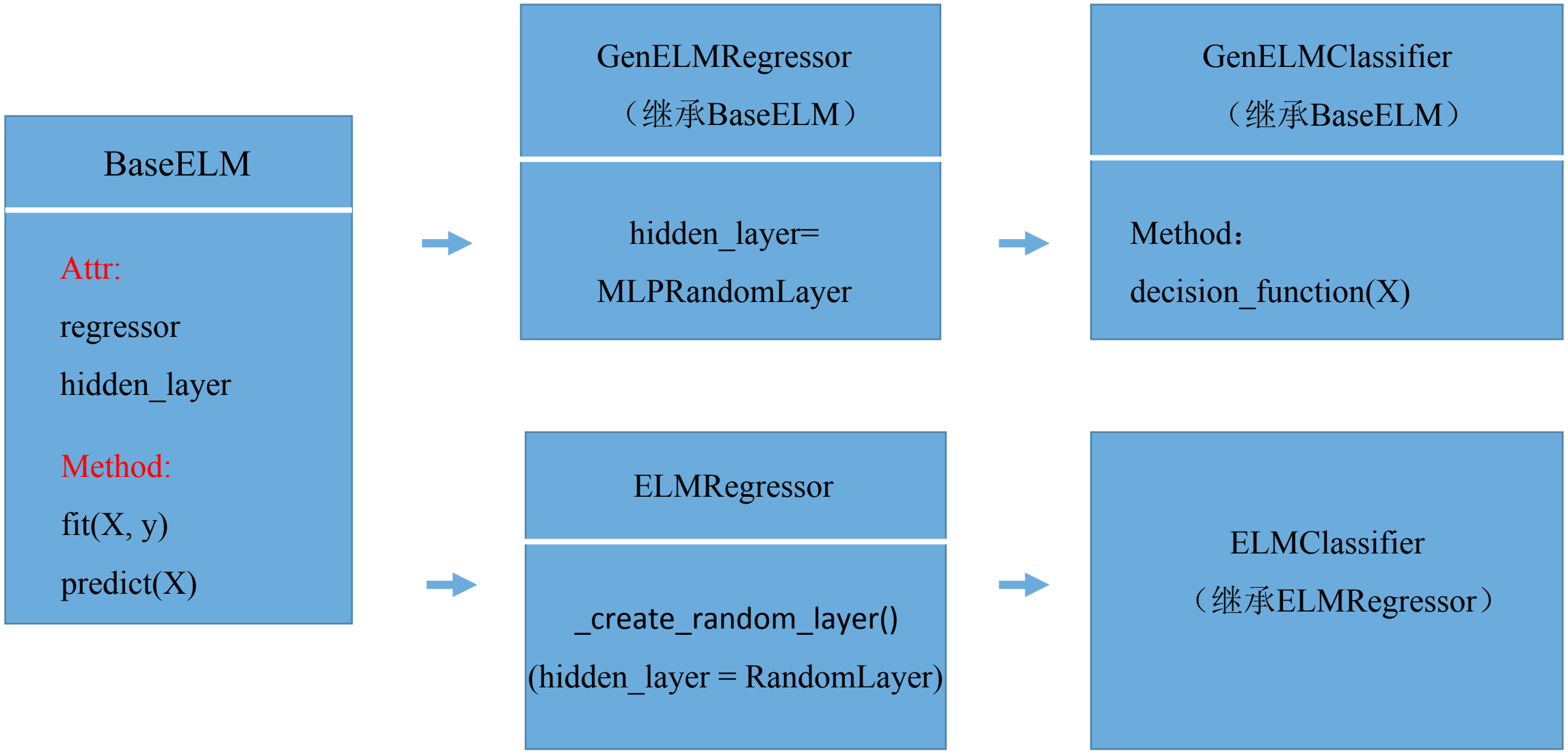


input_activation = alpha * mlp_activation + (1-alpha) * rbf_activation

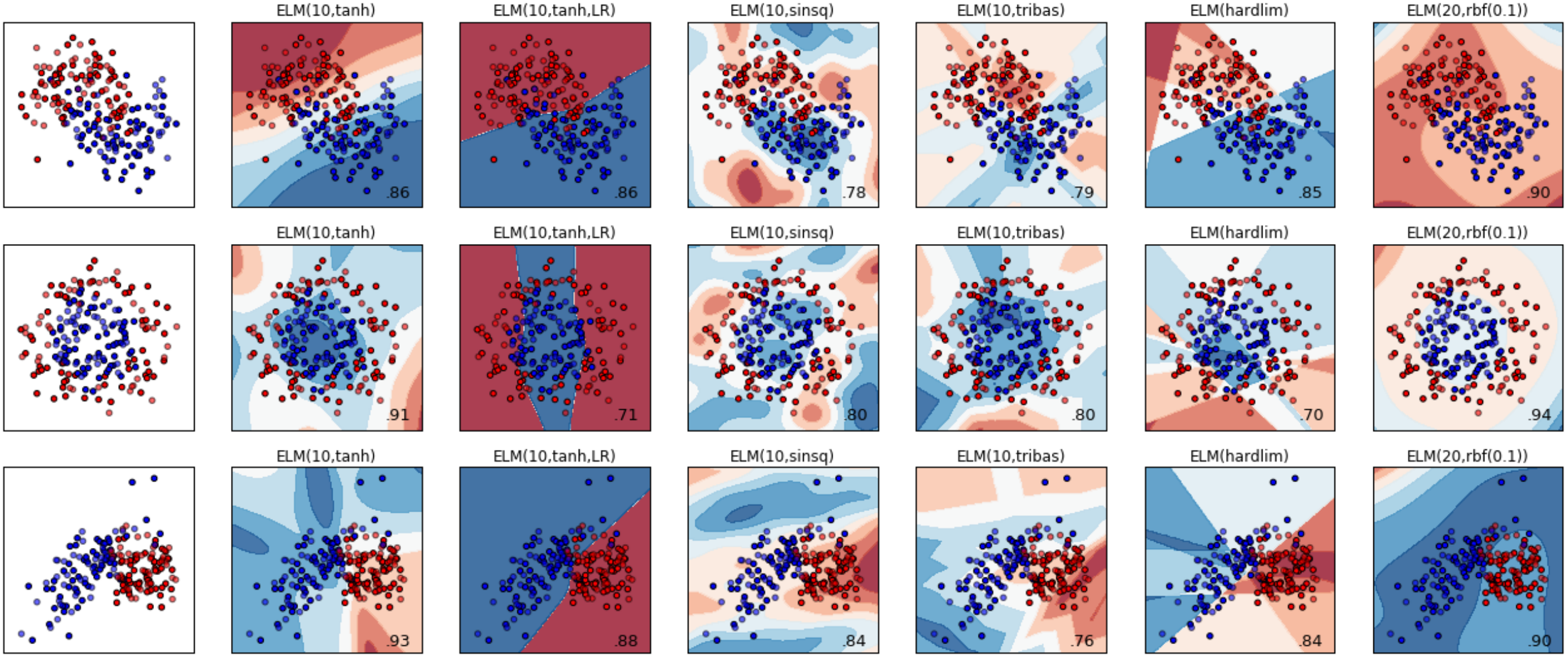
mlp_activation(x) = dot(x, weights) + bias

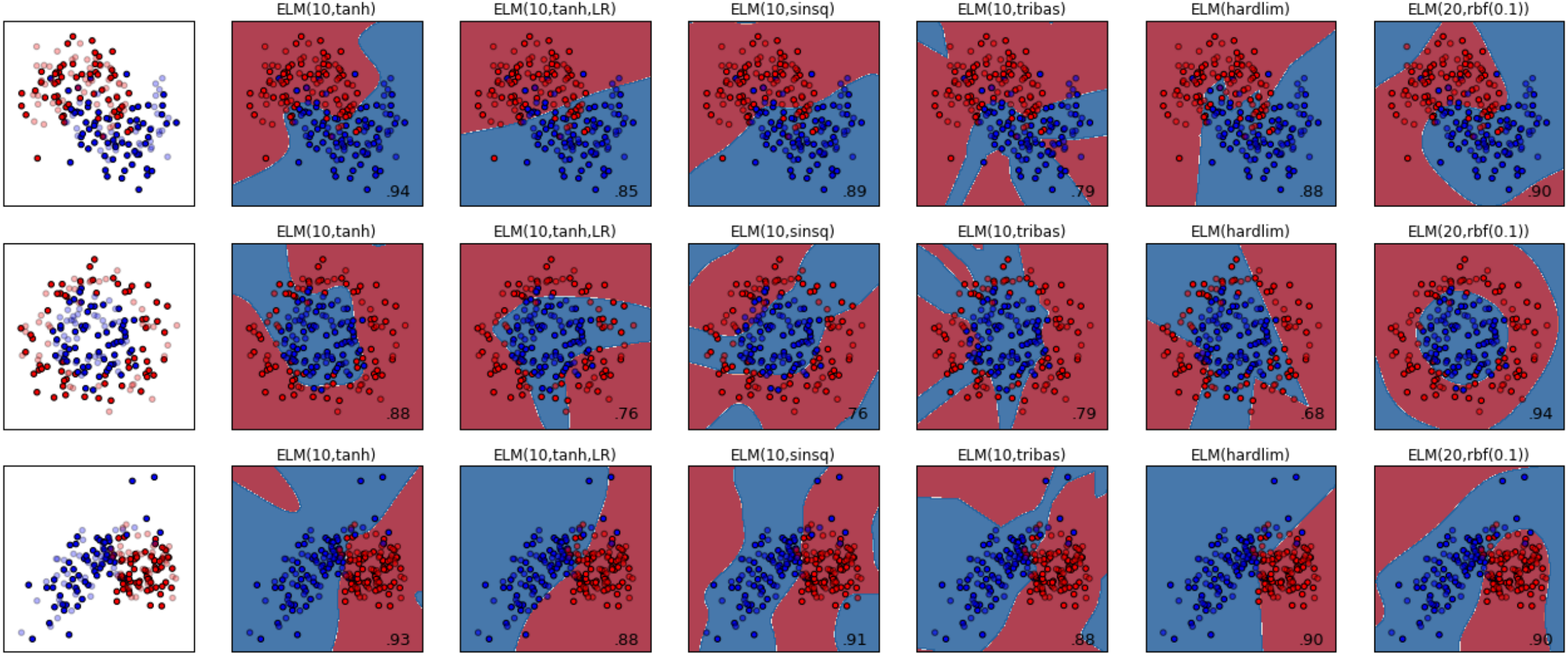
rbf_activation(x) = rbf_width * ||x - center||/radius

elm.py

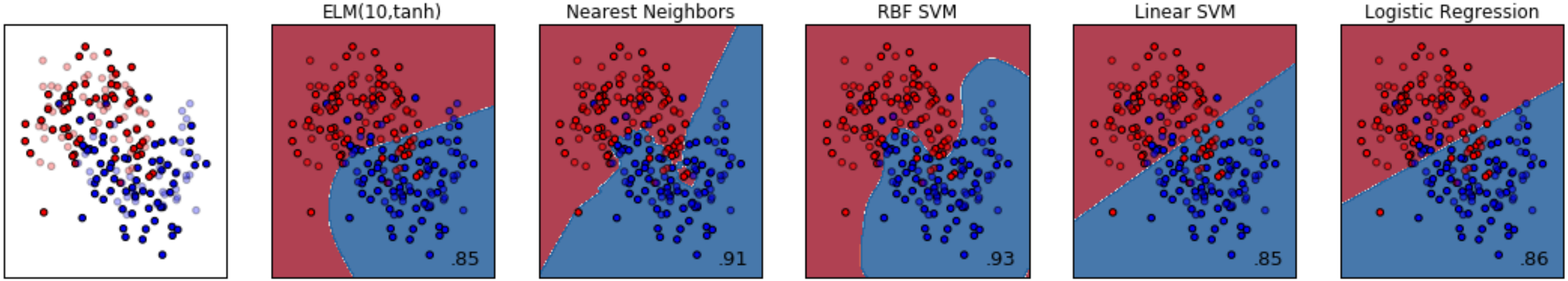

BaseELM
Attr:
regressor
hidden_layer
Method:
fit(X, y)
predict(X)
GenELMRegressor
（继承BaseELM）
hidden_layer=
MLPRandomLayer
GenELMClassifier
（继承BaseELM）
Method：
decision_function(X)
ELMRegressor
_create_random_layer()
(hidden_layer = RandomLayer)
ELMClassifier
（继承ELMRegressor）

plot_elm_comparison.py

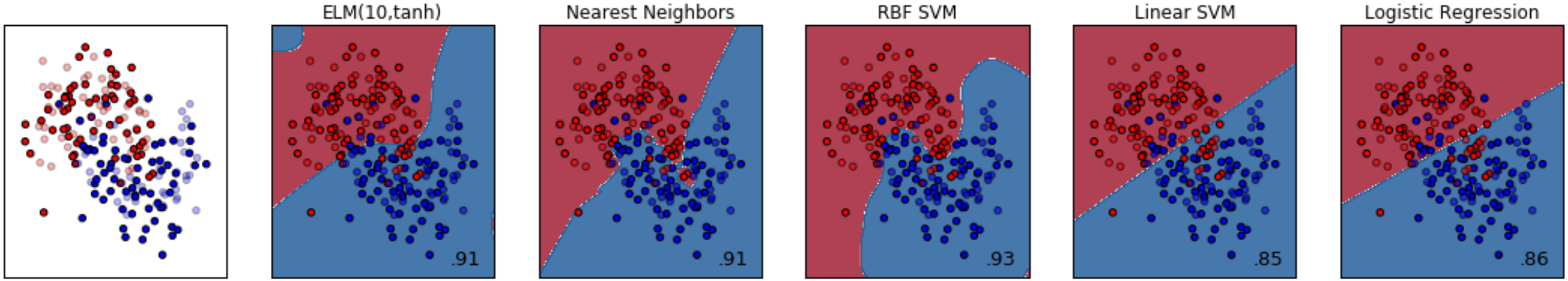



ELM(10,tanh)
ELM(10,tanh,LR)
ELM(10,sinsq)
ELM(10,tribas)
ELM(hardlim)
ELM(20,rbf(0.1))
.94
.85
.89
.79
.88
.90
ELM(10,tanh)
ELM(10,tanh,LR)
ELM(10,sinsq)
ELM(10,tribas)
ELM(hardlim)
ELM(20,rbf(0.1))
.88
.76
.76
.79
.68
.94
ELM(10,tanh)
ELM(10,tanh,LR)
ELM(10,sinsq)
ELM(10,tribas)
ELM(hardlim)
ELM(20,rbf(0.1))
.93
.88
.91
.88
.90
.90

First time



Second time



总结

- 基础知识
- 整个程序的结构
- Sklearn库

Any Question ?

Thanks for listening

References:

1. 简单易学的机器学习算法——极限学习机http://blog.csdn.net/google19890102/article/details/18222103
2. 论战Yann LeCun：谁能解释极限学习机（ELM）牛在哪里？ http://www.csdn.net/article/2015-05-07/2824636
3. 机器学习 --- 1. 线性回归与分类, 解决与区别http://blog.csdn.net/ppn029012/article/details/8775597
4. sklearn学习记录一：官方使用说明 http://blog.csdn.net/nkwangjie/article/details/17471829
5. 机器学习经典算法之-----最小二乘法 http://www.cnblogs.com/armysheng/p/3422923.html