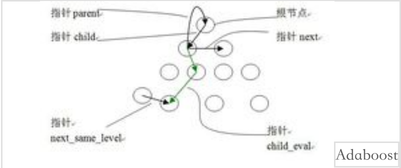
AI学习笔记--sklearn--AdaBoost算法

Adaboost是一种迭代算法，其核心思想是针对同一个训练集训练不同的分类器(弱分类器)，然后把这些弱分类器集合起来，构成一个更强的最终分类器（强分类器）。

- ## 算法简介

Adaboost是一种迭代算法，其核心思想是针对同一个训练集训练不同的分类器(弱分类器)，然后把这些弱分类器集合起来，构成一个更强的最终分类器(强分类器)。其算法本身是通过改变数据分布来实现的，它根据每次训练集之中每个样本的分类是否正确，以及上次的总体分类的准确率，来确定每个样本的权值。将修改过权值的新数据集送给下层分类器进行训练，最后将每次训练得到的分类器最后融合起来，作为最后的决策分类器。使用adaboost分类器可以排除一些不必要的训练数据特征，并放在关键的训练数据上面。



Adaboost

- ## 算法应用

对adaBoost算法的研究以及应用大多集中于分类问题，同时也出现了一些在回归问题上的应用。就其应用adaBoost系列主要解决了: 两类问题、多类单标签问题、多类多标签问题、大类单标签问题、回归问题。它用全部的训练样本进行学习。

- ## 算法原理

基于Boosting的理解，对于AdaBoost，我们要搞清楚两点：

1. 每一次迭代的弱学习$h(x;a_m)$h(x;am)有何不一样，如何学习？
2. 弱分类器权值β_mβm如何确定？

对于第一个问题，AdaBoost改变了训练数据的权值，也就是样本的概率分布，其思想是将关注点放在被错误分类的样本上，减小上一轮被正确分类的样本权值，提高那些被错误分类的样本权值。然后，再根据所采用的一些基本机器学习算法进行学习，比如逻辑回归。

对于第二个问题，AdaBoost采用加权多数表决的方法，加大分类误差率小的弱分类器的权重，减小分类误差率大的弱分类器的权重。这个很好理解，正确率高分得好的弱分类器在强分类器中当然应该有较大的发言权。

- ## 实例

为了加深理解，我们来举一个例子。有如下的训练样本，我们需要构建强分类器对其进行分类。x是特征，y是标签。

序号	1	2	3	4	5	6	7	8	9	10
x	0	1	2	3	4	5	6	7	8	9
y	1	1	1	-1	-1	-1	1	1	1	-1

令权值分布$D_1=(w_{1,1},w_{1,2},\ldots,w_{1,10})$D1=(w1,1,w1,2,…,w1,10)

并假设一开始的权值分布是均匀分布：$w_{1,i}=0.1$，$i=1,2,\ldots,10$w1,i=0.1，i=1,2,…,10

现在开始训练第一个弱分类器。我们发现阈值取2.5时分类误差率最低，得到弱分类器为：

$$G_1(x)=\begin{cases}1, & \text{x<2.5}\\ -1, & \text{x>2.5}\end{cases}$$

当然，也可以用别的弱分类器，只要误差率最低即可。这里为了方便，用了分段函数。得到了分类误差率$e_1=0.3$e1=0.3。

第二步计算$(G_1(x)$(G1(x)在强分类器中的系数$\alpha_1=\frac{1}{2}\log\frac{1-e_1}{e_1}=0.4236$α1=12log1−e1e1=0.4236，这个公式先放在这里，下面再做推导。

第三步更新样本的权值分布，用于下一轮迭代训练。由公式：

$$w_{2,i}=\frac{w_{1,i}}{z_1}exp(-\alpha_1 y_i G_1(x_i)),\ i=1,2,\dots,10$$

得到新的权值分布，从各0.1变成了:

$$D_2=(0.0715,0.0715,0.0715,0.0715,0.0715,0.0715,0.1666,0.1666,0.1666,0.0715)$$

可以看出，被分类正确的样本权值减小了，被错误分类的样本权值提高了。

第四步得到第一轮迭代的强分类器:

$$sign(F_1(x))=sign(0.4236G_1(x))$$

以此类推，经过第二轮……第N轮，迭代多次直至得到最终的强分类器。迭代范围可以自己定义，比如限定收敛阈值，分类误差率小于某一个值就停止迭代，比如限定迭代次数，迭代1000次停止。这里数据简单，在第3轮迭代时，得到强分类器:

$$sign(F_3(x))=sign(0.4236G_1(x)+0.6496G_2(x)+0.7514G_3(x))$$

的分类误差率为0，结束迭代。

$$F(x)=sign(F_3(x))$$

就是最终的强分类器。

- 算法流程

总结一下，得到AdaBoost的算法流程:

输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),(x_N,y_N)\}$，其中，$x_i\in X\subseteq R^n$，$y_i\in Y=-1,1$，迭代次数$M$

初始化训练样本的权值分布：$D_1=(w_{1,1},w_{1,2},\dots,w_{1,i}),w_{1,i}=\frac{1}{N},i=1,2,\dots,N$。

对于$m=1,2,\dots,M$

(a) 使用具有权值分布D_m的训练数据集进行学习，得到弱分类器$G_m(x)$ (b) 计算$G_m(x)$在训练数据集上的分类误差率:

$$e_m=\sum_{i=1}^{N}w_{m,i}I(G_m(x_i)\neq y_i)$$

(c) 计算$G_m(x)$在强分类器中所占的权重:

$$\alpha_m=\frac{1}{2}log\frac{1-e_m}{e_m}$$

(d) 更新训练数据集的权值分布（这里，z_m是归一化因子，为了使样本的概率分布和为1）:

$$w_{m+1,i}=\frac{w_{m,i}}{z_m}exp(-\alpha_m y_i G_m(x_i)),\ i=1,2,\dots,10$$

$$z_m=\sum_{i=1}^{N}w_{m,i}exp(-\alpha_m y_i G_m(x_i))$$

3. 得到最终分类器:

$$F(x)=sign(\sum_{i=1}^{N}\alpha_m G_m(x))$$

- 公式推导

现在我们来搞清楚上述公式是怎么来的。

假设已经经过$m-1$轮迭代，得到$F_{m-1}(x)$，根据前向分步，我们可以得到:

$$F_m(x)=F_{m-1}(x)+\alpha_m G_m(x)$$

我们已经知道AdaBoost是采用指数损失，由此可以得到损失函数:

$$Loss=\sum_{i=1}^{N}exp(-y_iF_m(x_i))=\sum_{i=1}^{N}exp(-y_i(F_{m-1}(x_i)+\alpha_m G_m(x_i)))$$

这时候，$F_{m-1}(x)$是已知的，可以作为常量移到前面去:

$$Loss=\sum_{i=1}^{N}\widetilde{w_{m,i}}exp(-y_i\alpha_m G_m(x_i))$$

其中，$\widetilde{w_{m,i}}=exp(-y_i(F_{m-1}(x)))$，敲黑板！这个就是每轮迭代的样本权重！依赖于前一轮的迭代重分配。

是不是觉得还不够像？那就再化简一下：

$$\tilde{w}_{m,i}=exp(-y_i(F_{m-1}(x_i)+\alpha_{m-1}G_{m-1}(x_i)))=\tilde{w}_{m-1,i}\,exp(-y_i\alpha_{m-1}G_{m-1}(x_i))$$

现在够像了吧？ok，我们继续化简Loss：

$$Loss=\sum_{y_i=G_m(x_i)}\tilde{w}_{m,i}\,exp(-\alpha_m)+\sum_{y_i\neq G_m(x_i)}\tilde{w}_{m,i}\,exp(\alpha_m)$$

$$=\sum_{i=1}^{N}\tilde{w}_{m,i}\left(\frac{\sum_{y_i=G_m(x_i)}\tilde{w}_{m,i}}{\sum_{i=1}^{N}\tilde{w}_{m,i}}exp(-\alpha_m)+\frac{\sum_{y_i\neq G_m(x_i)}\tilde{w}_{m,i}}{\sum_{i=1}^{N}\tilde{w}_{m,i}}exp(\alpha_m)\right)$$

公式变形之后，炒鸡激动！$\frac{\sum_{y_i\neq G_m(x_i)}\tilde{w}_{m,i}}{\sum_{i=1}^{N}\tilde{w}_{m,i}}$这个不就是分类误差率$e_m$吗？？？！重写一下，

$$Loss=\sum_{i=1}^{N}\tilde{w}_{m,i}\,exp(-\alpha_m)+e_m exp(\alpha_m))$$

Ok，这样我们就得到了化简之后的损失函数。接下来就是求导了。对α_m求偏导，令$\frac{\partial Loss}{\partial \alpha_m}=0$得到：

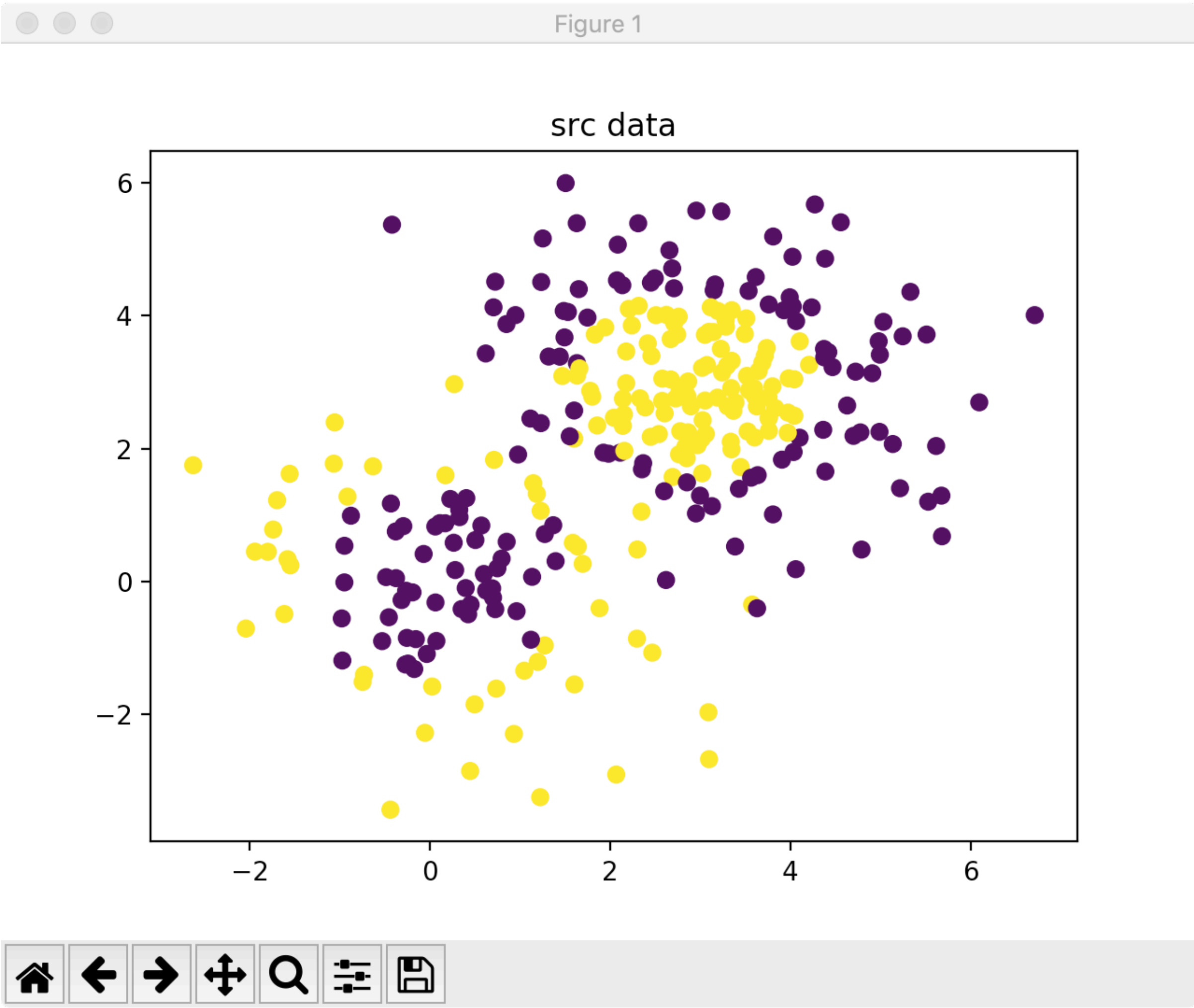
$$\alpha_m=\frac{1}{2}log\frac{1-e_m}{e_m}$$

- **算法例子**

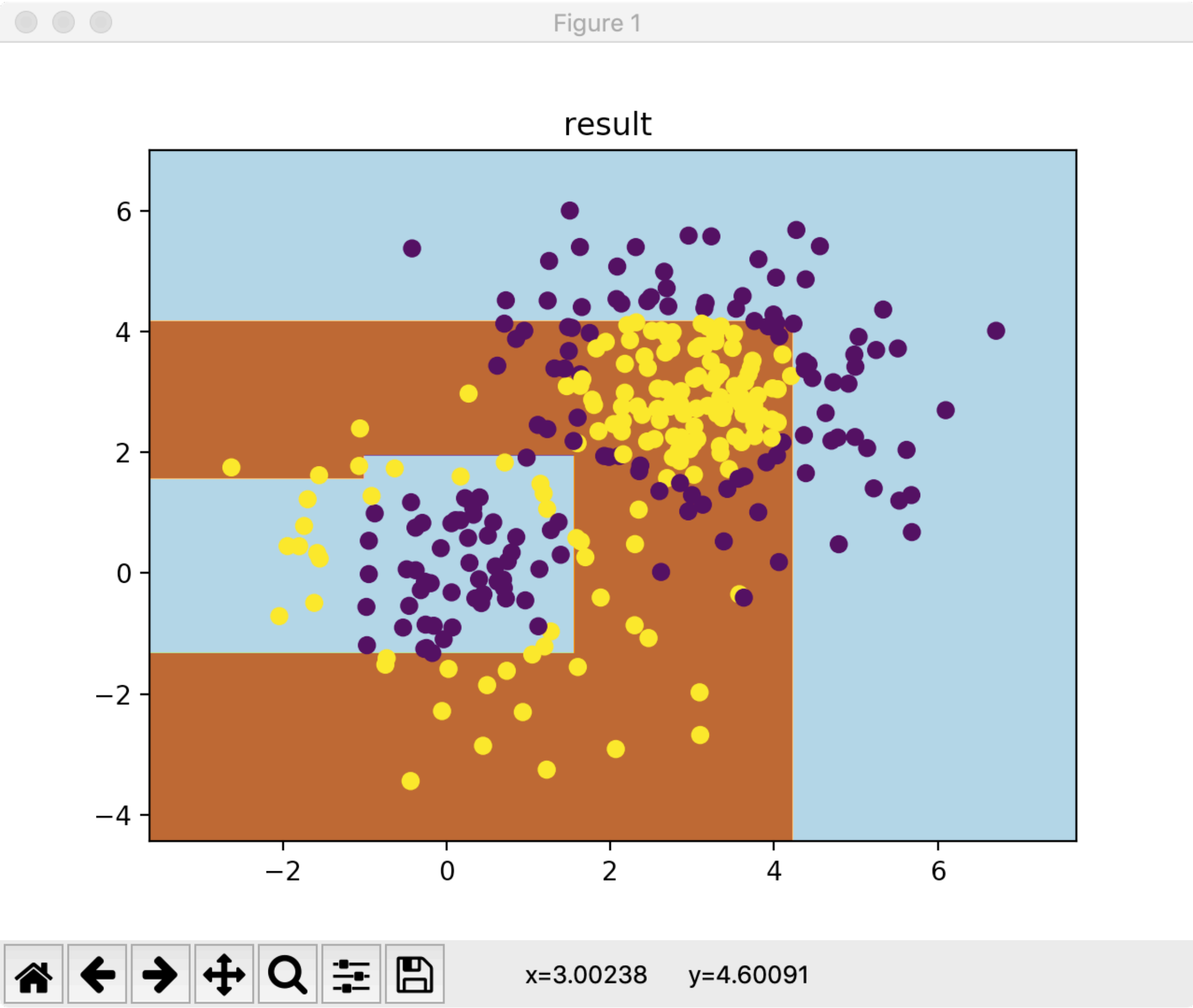
例子是Python的，最简单的例子

```
#coding=utf-8
'''
Created on 2017年11月27日
'''
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_gaussian_quantiles

#用make_gaussian_quantiles生成多组多维正态分布的数据
#这里生成2维正态分布，设定样本数100，协方差2
x1,y1=make_gaussian_quantiles(cov=2., n_samples=100, n_features=2, n_classes=2, shuffle=True, random_state=1)
#为了增加样本分布的复杂度，再生成一个数据分布
x2,y2=make_gaussian_quantiles(mean=(3,3), cov=1.5, n_samples=200, n_features=2, n_classes=2, shuffle=True,
random_state=1)
#合并
X=np.vstack((x1,x2))
y=np.hstack((y1,1-y2))
plt.scatter(X[:,0],X[:,1],c=y)
plt.title('src data')
plt.show()

#设定弱分类器CART
weakClassifier=DecisionTreeClassifier(max_depth=1)

#构建模型。
clf=AdaBoostClassifier(base_estimator=weakClassifier,algorithm='SAMME',n_estimators=200,learning_rate=0.1)
clf.fit(X, y)

#绘制分类效果
x1_min=X[:,0].min()-1
x1_max=X[:,0].max()+1
x2_min=X[:,1].min()-1
x2_max=X[:,1].max()+1
x1_,x2_=np.meshgrid(np.arange(x1_min,x1_max,0.02),np.arange(x2_min,x2_max,0.02))

y_=clf.predict(np.c_[x1_.ravel(),x2_.ravel()])
y_=y_.reshape(x1_.shape)
plt.contourf(x1_,x2_,y_,cmap=plt.cm.Paired)
plt.title('result')
plt.scatter(X[:,0],X[:,1],c=y)
plt.show()
```

执行结果：



原始生成数据和实际训练结果：

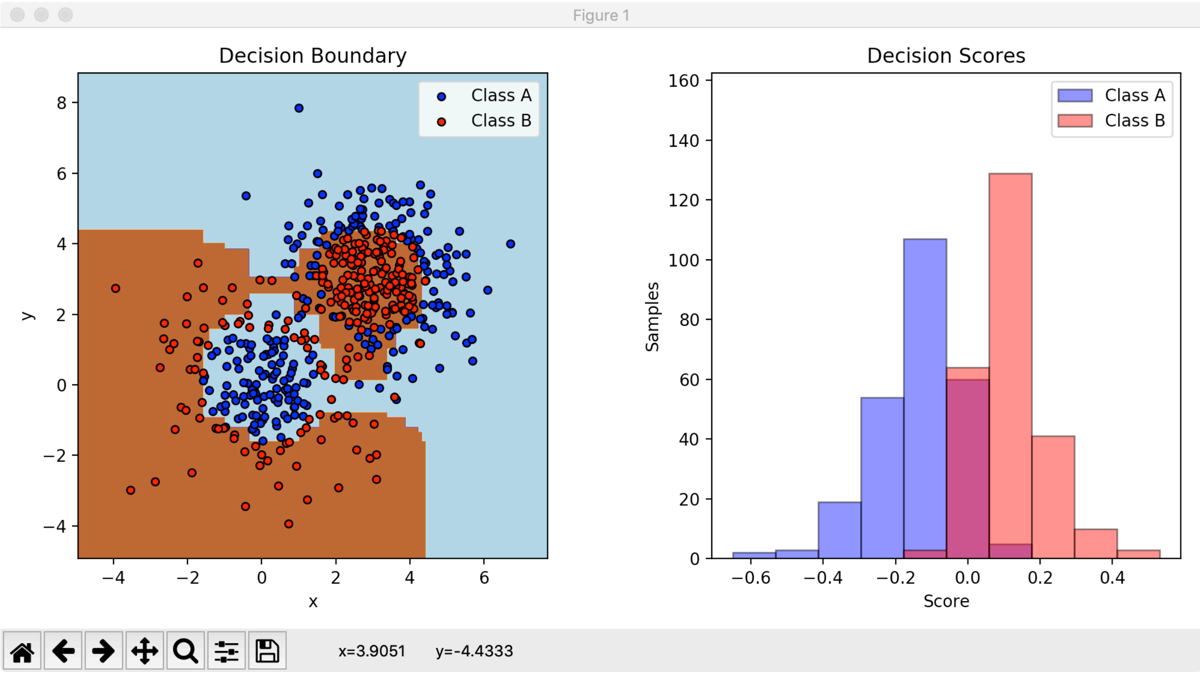


再看一个sklearn的官方例子：

```
print(__doc__)

# Author: Noel Dawe <noel.dawe@gmail.com>
#
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_gaussian_quantiles


# Construct dataset
X1, y1 = make_gaussian_quantiles(cov=2.,
                                 n_samples=200, n_features=2,
                                 n_classes=2, random_state=1)
X2, y2 = make_gaussian_quantiles(mean=(3, 3), cov=1.5,
                                 n_samples=300, n_features=2,
                                 n_classes=2, random_state=1)
X = np.concatenate((X1, X2))
y = np.concatenate((y1, - y2 + 1))

# Create and fit an AdaBoosted decision tree
bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),
                         algorithm="SAMME",
                         n_estimators=200)

bdt.fit(X, y)

plot_colors = "br"
```

```
plot_step = 0.02
class_names = "AB"

plt.figure(figsize=(10, 5))

# Plot the decision boundaries
plt.subplot(121)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

Z = bdt.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
plt.axis("tight")

# Plot the training points
for i, n, c in zip(range(2), class_names, plot_colors):
    idx = np.where(y == i)
    plt.scatter(X[idx, 0], X[idx, 1],
                c=c, cmap=plt.cm.Paired,
                s=20, edgecolor='k',
                label="Class %s" % n)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.legend(loc='upper right')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Decision Boundary')

# Plot the two-class decision scores
twoclass_output = bdt.decision_function(X)
plot_range = (twoclass_output.min(), twoclass_output.max())
plt.subplot(122)
for i, n, c in zip(range(2), class_names, plot_colors):
    plt.hist(twoclass_output[y == i],
             bins=10,
             range=plot_range,
             facecolor=c,
             label='Class %s' % n,
             alpha=.5,
             edgecolor='k')
x1, x2, y1, y2 = plt.axis()
plt.axis((x1, x2, y1, y2 * 1.2))
plt.legend(loc='upper right')
plt.ylabel('Samples')
plt.xlabel('Score')
plt.title('Decision Scores')

plt.tight_layout()
plt.subplots_adjust(wspace=0.35)
plt.show()
```

输出结果:



Sklearn 也提供了各种对比：

```
print(__doc__)
# Code source: Gaël Varoquaux
#              Andreas Müller
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

h = .02  # step size in the mesh

names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]

X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable
```

```
            ]

figure = plt.figure(figsize=(27, 9))
i = 1
# iterate over datasets
for ds_cnt, ds in enumerate(datasets):
    # preprocess dataset, split into training and test part
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=.4, random_state=42)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    if ds_cnt == 0:
        ax.set_title("Input data")
    # Plot the training points
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright,
               edgecolors='k')
    # Plot the testing points
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6,
               edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    i += 1

    # iterate over classifiers
    for name, clf in zip(names, classifiers):
        ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, x_max]x[y_min, y_max].
        if hasattr(clf, "decision_function"):
            Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
        else:
            Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)

        # Plot the training points
        ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright,
                   edgecolors='k')
        # Plot the testing points
        ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,
                   edgecolors='k', alpha=0.6)

        ax.set_xlim(xx.min(), xx.max())
        ax.set_ylim(yy.min(), yy.max())
        ax.set_xticks(())
        ax.set_yticks(())
        if ds_cnt == 0:
            ax.set_title(name)
        ax.text(xx.max() - .3, yy.min() + .3, ('%.2f' % score).lstrip('0'),
                size=15, horizontalalignment='right')
        i += 1

plt.tight_layout()
plt.show()
```

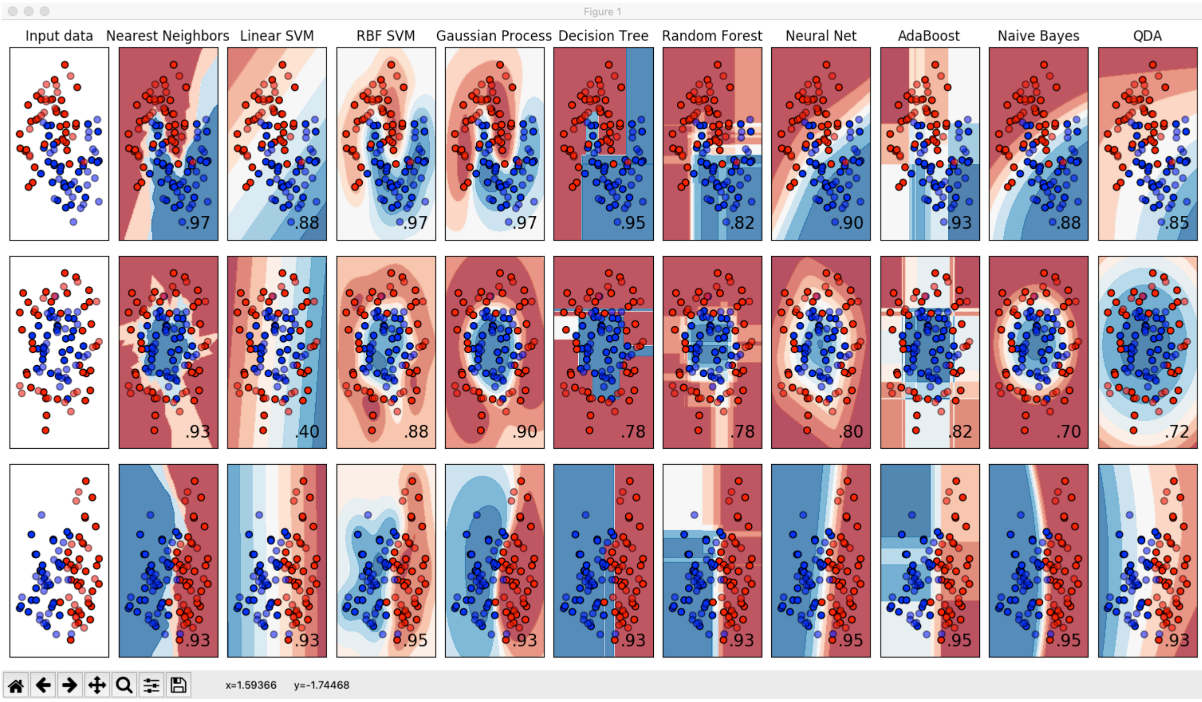


Figure 1
Input data
Nearest Neighbors
Linear SVM
RBF SVM
Gaussian Process
Decision Tree
Random Forest
Neural Net
AdaBoost
Naive Bayes
QDA
.97
.88
.97
.97
.95
.82
.90
.93
.88
.85
.93
.40
.88
.90
.78
.78
.80
.82
.70
.72
.93
.93
.95
.93
.93
.93
.95
.95
.95
.93
x=1.59366 y=-1.74468