

机器学习作业-随机森林

软件 51 庞建业 2151601012

<https://github.com/sherjy/Notes-RL>

本次主要针对随机森林 (Random Forest) 的实现的学习和复习

随机森林是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的，决策树实际上是将空间用超平面进行划分的一种方法，每次分割的时候，都将当前的空间一分为二，利用多个分类树对数据进行判别与分类的方法，它在对数据进行分类的同时，还可以给出各个变量（基因）的重要性评分，评估各个变量在分类中所起的作用。

环境：

Ubuntu16.04

Atom+Anaconda3.6

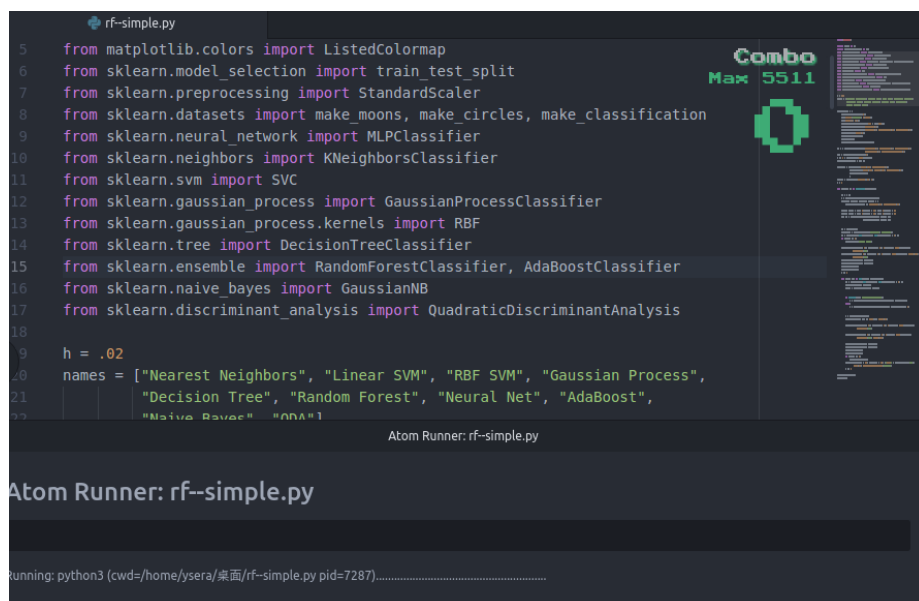
代码：

(1) .这个例子是在人工数据集 make_moons 上使用不同分类器来跑的，对不同分类器会形成不同的分类边界，之所以说是人造数据集是因为只是为了表现在低维数据上各个分类器的不同效果的显现，因为在高维数据上往往更容易被线性分类器达到很好的效果，比如线性 svm，朴素 bayes 等等。

见文件夹中 rf-simple.py

使用到的分类器有

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

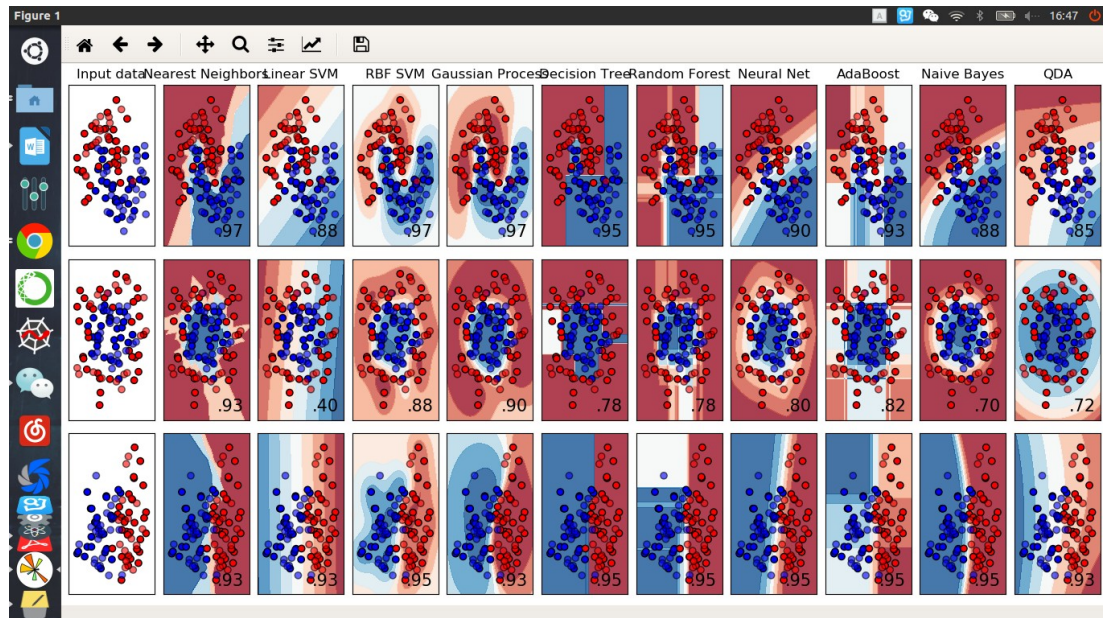


```
rf-simple.py  
5 from matplotlib.colors import ListedColormap  
6 from sklearn.model_selection import train_test_split  
7 from sklearn.preprocessing import StandardScaler  
8 from sklearn.datasets import make_moons, make_circles, make_classification  
9 from sklearn.neural_network import MLPClassifier  
10 from sklearn.neighbors import KNeighborsClassifier  
11 from sklearn.svm import SVC  
12 from sklearn.gaussian_process import GaussianProcessClassifier  
13 from sklearn.gaussian_process.kernels import RBF  
14 from sklearn.tree import DecisionTreeClassifier  
15 from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier  
16 from sklearn.naive_bayes import GaussianNB  
17 from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis  
18  
19 h = .02  
20 names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",  
21         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",  
22         "Naive Bayes", "QDA"]  
23  
Atom Runner: rf-simple.py  
Atom Runner: rf-simple.py  
Running: python3 (cwd=/home/ysera/桌面/rf-simple.py pid=7287).....
```

数据集链接: make_moons, make_circles, make_classification

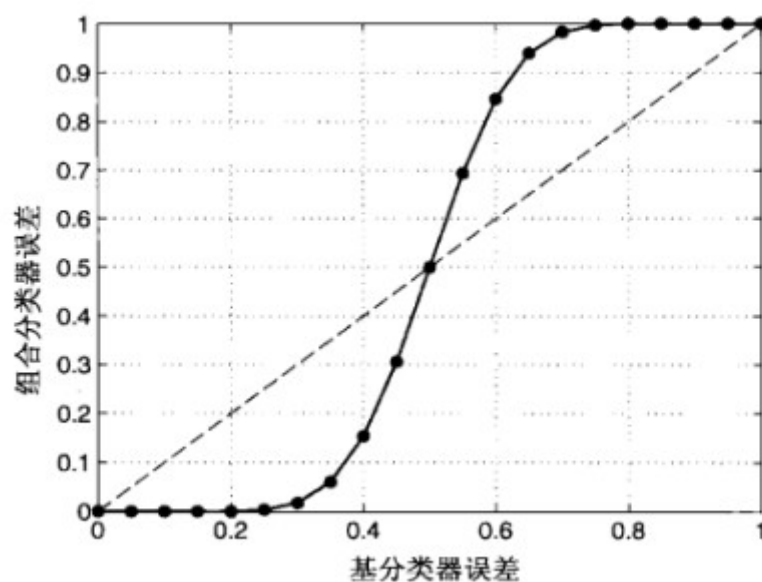
http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html

各个分类器的分类效果:



上面的图是依据每次使用随机森林的时候只使用一种分类器,

有一张很经典的图来说明组合分类器和基本分类器的关系:



上图, 虚线表示所有基分类器都一样, 实线表示所有基分类器都独立。可以看出, 当基分类器的错误率大于 0.5 时候, 组合分类器的性能不比基分类器的性能好, 当然这个 0.5 是一个概念上的数字。

组合分类器的性能优于基分类器的条件:

- 1、基分类器应该是独立的。
- 2、基分类器应当好于随机猜想。

(2).这个例子跑的是对鸢尾花 iris 数据集进行特征分类（这里的特征是特征子集，为了可视化方便，所以只选择部分特征做三种花的分类），我们知道随机森林属于集成学习，并且基于决策树，所以在代码中包含了与随机森林、树分类模型思想最贴近的四种分类器，并且测试使用单一种时的分类效果和得分

见文件夹中 rf-simple1.py

在此附上该 demo 的代码链接

http://scikit-learn.org/dev/auto_examples/ensemble/plot_forest_iris.html#sphx-glr-auto-examples-ensemble-plot-forest-iris-py

```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn.datasets import load_iris
from sklearn.ensemble import (RandomForestClassifier, ExtraTreesClassifier,
                              AdaBoostClassifier)
from sklearn.tree import DecisionTreeClassifier

n_classes = 3
n_estimators = 30
cmap = plt.cm.RdYlBu
plot_step = 0.02
plot_step_coarser = 0.5
RANDOM_SEED = 13

iris = load_iris()

plot_idx = 1

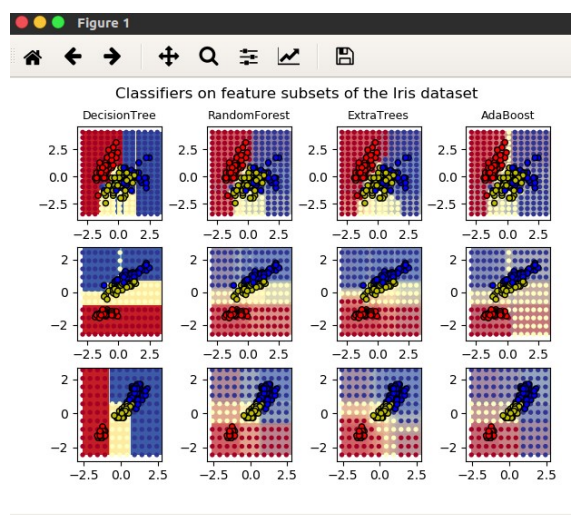
models = [DecisionTreeClassifier(max_depth=None),
          RandomForestClassifier(n_estimators=n_estimators),
          ExtraTreesClassifier(n_estimators=n_estimators),
          AdaBoostClassifier(DecisionTreeClassifier(max_depth=3),
                              n_estimators=n_estimators)]

for pair in ([0, 1], [0, 2], [2, 3]):
    for model in models:
```

iris.csv 数据集我宕下来了 放在文件夹中

以鸢尾花的特征作为数据来源，数据集包含 150 个数据集，分为 3 类，每类 50 个数据，每个数据包含 4 个独立的属性,这些属性变量测量植物的花朵,比如萼片长度, 萼片宽度,花瓣长度, 花瓣宽度.

三类分别为:setosa, versicolor, virginica



在这个 demo 生成的图中，有四列图片，展示除了随机森林还是用了其他一些分类方法做对比实验

第一行生成的图片使用 sepal width 和 sepal length 特征

第二行生成的图片使用 petal length 和 sepal length 特征

第三行生成的图片使用 petal width 和 petal length 特征

我们可以看出来分类结果差异很明显，在验证集中我们使用 4 种特征，30 个决策树（这里我们加快计算减少复杂度 只使用 30 棵树 并且由于使用的是 CART 算法 对于鸢尾花这种数据很少的数据集很容易过拟合，所以树的数量可以设置的少一些），10 次 10 折交叉验证（将数据集分成十分，轮流将其中 9 份做训练 1 份做测试，10 次的结果的均值作为对算法精度的估计），打分如下：

```
Atom Runner: rf-simple.py

None
DecisionTree with features [0, 1] has a score of 0.926666666667
RandomForest with 30 estimators with features [0, 1] has a score of 0.926666666667
ExtraTrees with 30 estimators with features [0, 1] has a score of 0.926666666667
AdaBoost with 30 estimators with features [0, 1] has a score of 0.84
DecisionTree with features [0, 2] has a score of 0.993333333333
RandomForest with 30 estimators with features [0, 2] has a score of 0.993333333333
ExtraTrees with 30 estimators with features [0, 2] has a score of 0.993333333333
AdaBoost with 30 estimators with features [0, 2] has a score of 0.993333333333
DecisionTree with features [2, 3] has a score of 0.993333333333
RandomForest with 30 estimators with features [2, 3] has a score of 0.993333333333
ExtraTrees with 30 estimators with features [2, 3] has a score of 0.993333333333
AdaBoost with 30 estimators with features [2, 3] has a score of 0.993333333333
```

注意：值得一提的是，根据我之前使用 sklearn 的经验，RandomForest 的分类类是 RandomForestClassifier，回归类是 RandomForestRegressor（之前的 demo 都是做分类任务的），sklearn 中的随机森林默认的分类器应该是 CART（因为决策树默认为 CART 算法），所以需要调参的参数包括两部分，第一部分是 Bagging 框架的参数，第二部分是 CART 决策树的参数（树高 depth、树的个数 numbers 等等）。由于随机森林是 bagging 框架的，所以是一个固定某些参数需要调节的算法。

在此放上一些随机森林特有的、可调且对结果较有用的参数：

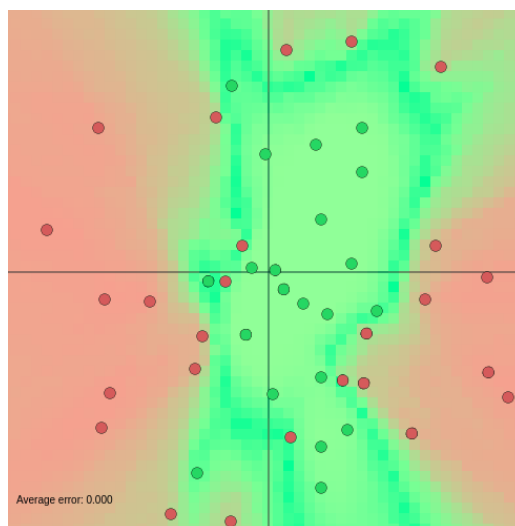
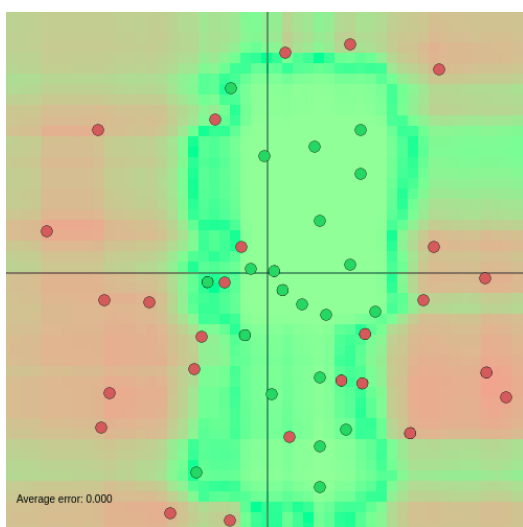
n_estimators=100	决策树的个数，越多越好，但是性能就会越差，至少 100 左右,可以达到可接受的性能和误差率。
bootstrap=True	是否有放回的采样。
oob_score=False	oob（outof band，带外）数据，即：在某次决策树训练中没有被 bootstrap 选中的数据。多单个模型的参数训练，我们知道可以用

	crossvalidation (cv) 来进行，但是特别消耗时间，而且对于随机森林这种情况也没有大的必要，所以就用这个数据对决策树模型进行验证，算是一个简单的交叉验证。性能消耗小，但是效果不错。
warm_start=False	热启动，决定是否使用上次调用该类的结果然后增加新的。
class_weight=None	各个 label 的权重。

(3) .然后我们来看一个好玩的

这个是我最早学习 randomforest 的时候玩的一个 toolkit，主要用于二维数据点的随机森林可视化

我改写参数和随机生成点后生成的一张图：



上面两个图分别从一维和二维的角度去呈现

参数调整的为：

max_depth:6

numTries :10

number of trees:150(刚才我们说过，因为我们知道至少要达到 100 次才能保证结果是有效的)

(4) .使用随机森林对引入噪声的近似余弦做多周期 multi-period 回归任务 见文件夹中 **rf1.py**

```
fast_oscillation = np.sin(5 * x)
```

```
slow_oscillation = np.sin(0.5 * x)
```

```
noise = sigma * np.random.randn(len(x))
```

```
def model(x, sigma=0.3):
    fast_oscillation = np.sin(5 * x)
    slow_oscillation = np.sin(0.5 * x)
    noise = sigma * np.random.randn(len(x))

    return slow_oscillation + fast_oscillation + noise

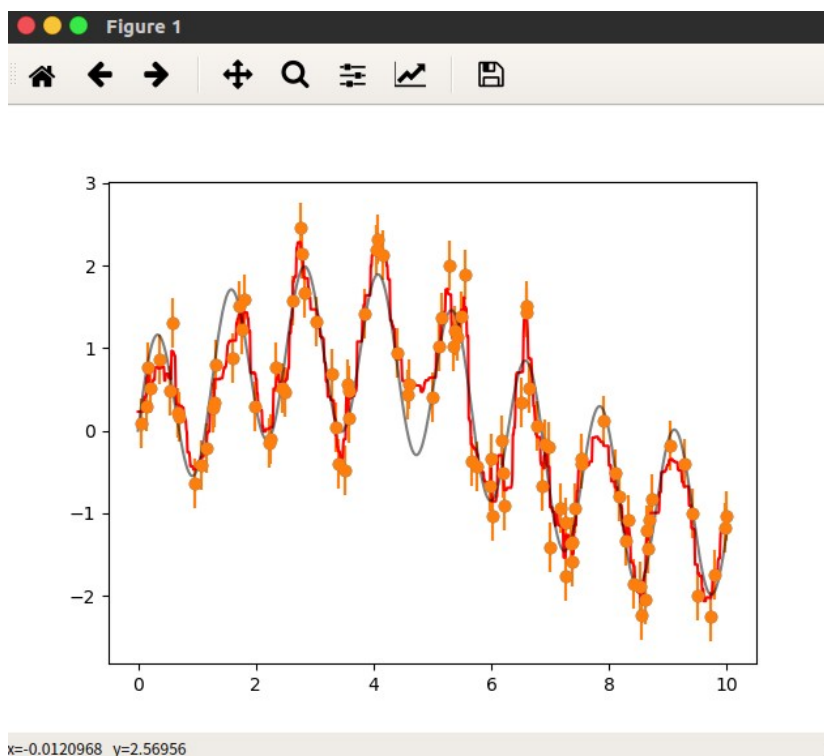
y = model(x)
plt.errorbar(x, y, 0.3, fmt='o');
xfit = np.linspace(0, 10, 1000)
yfit = RandomForestRegressor(100).fit(x[:, None], y).predict(xfit[:, None])
ytrue = model(xfit, 0)

plt.errorbar(x, y, 0.3, fmt='o')
plt.plot(xfit, yfit, '-r')
plt.plot(xfit, ytrue, '-k', alpha=0.5)
plt.show()
```

Atom Runner: rf-simple.py

Atom Runner: rf1.py

Running: python3 (cwd=/home/ysera/桌面/rf1.py
pid=23408)



可以看出来 对随机点的拟合效果还是非常好的

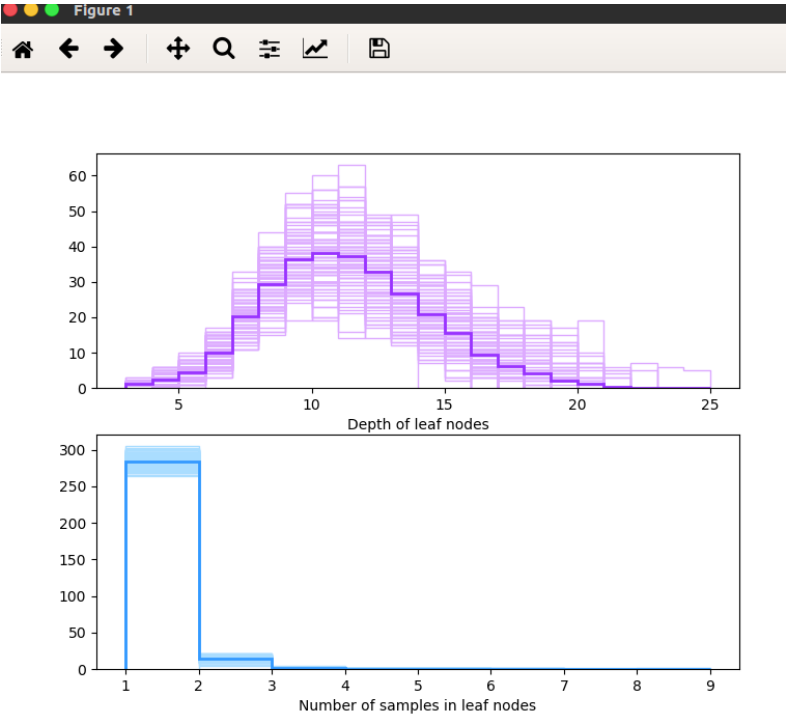
(5) .使用 sklearn 的数据集 load_boston 对叶节点信息（叶节点深度和 sample 数）的统计可视化

见文件夹中 rf2.py

```
6
7 import matplotlib.pyplot as plt
8
9 from sklearn.utils import check_random_state
10
11 def leaf_depths(tree, node_id = 0):
12     left_child = tree.children_left[node_id]
13     right_child = tree.children_right[node_id]
14
15     if left_child == _tree.TREE_LEAF:
16         depths = np.array([0])
17
18     else:
19         left_depths = leaf_depths(tree, left_child) + 1
20         right_depths = leaf_depths(tree, right_child) + 1
21
22
23
24
```

Atom Runner: rf-simple.py

Atom Runner: rf2.py



scikit-learn 自带波士顿房价集,该数据集来源于 1978 年美国某经济学杂志上。该数据集包含若干波士顿房屋的价格及其各项数据，每个数据项包含 14 个数据，分别是房屋均价及周边犯罪率、是否在河边等相关信息，其中最后一个数据是房屋均，这个数据集很经典，我最初也是在别的用于可视化例子中见到使用。

随机森林相关小结：

(1) .Bagging 和 Boosting

Bagging 算法的全称应该是 Bootstrap aggregating。

它有两个步骤组成：Bootstrap 和 aggregating。

Bootstrap 方法是有放回的抽样，即从初始训练集中有放回可重复的随机取出 N 条数据（这个值需要事先设定，可以是初始数据集的 80%、70% 都随意）组成新的数据集。

另一个步骤就是 aggregating，通常用的是投票法。训练分类器的算法往往有稳定和不稳定两类，通常是使用不稳定的学习算法，因为不稳定的学习算法可以因为数据集微小的变化而导致结果的改变，因此有助于我们生成若干具有一定差异性的分类器集合。

Boosting 主要是 Adaboost (Adaptive Boosting)，它与 Bagging 的不同在于他将权重赋予每个训练元组，生成基分类器的过程为迭代生成。每当训练生成一个分类器 $M(i)$ 时要进行权重更新，使得 $M(i+1)$ 更关注被 $M(i)$ 分类错误的训练元组。最终提升整体集合的分类准确率，集成规则是加权投票，每个分类器投票的权重是其准确率的函数。

这两种集成算法主要区别在于“加没加权”。Bagging 的训练集是随机生成，分类器相互独立；而 Boosting 的分类器是迭代生成，更关注上一轮的错分元组。因此 Bagging 的各个预测函数可以并行生成；而 Boosting 只能顺序生成。因此对于像一些比较耗时的分类器训练算法（如神经网络等）如果使用 Bagging 可以极大地节约时间开销。

但是通过在大多数数据集的实验，Boosting 的准确率多数会高于 Bagging，但是也有极个别数据集使用 Boosting 反倒会退化。

(2) .GBDT 和 xgboost

GBDT 的核心就在于：每一棵树学的是之前所有树结论和的残差，这个残差就是一个加预测值后能得真实值的累加量。

Xgboost 相比于 GBDT 来说，更加有效应用了数值优化，最重要是对损失函数（预测值和真实值的误差）变得更复杂。目标函数依然是所有树的预测值相加等于预测值。

传统 GBDT 以 CART 作为基分类器，xgboost 还支持线性分类器，这个时候 xgboost 相当于带 L1 和 L2 正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。

决策树优点及局限性：

优点：

a. 在数据集上表现良好，两个随机性的引入，使得随机森林不容易陷入过拟合

- b. 在当前的很多数据集上，相对其他算法有着很大的优势，两个随机性的引入，使得随机森林具有很好的抗噪声能力
- c. 它能够处理很高维度（feature 很多）的数据，并且不用做特征选择，对数据集的适应能力强：既能处理离散型数据，也能处理连续型数据，数据集无需规范化
- d. 可生成一个 Proximities= (pij) 矩阵，用于度量样本之间的相似性： $p_{ij}=a_{ij}/N$, a_{ij} 表示样本 i 和 j 出现在随机森林中同一个叶子结点的次数，N 随机森林中树的颗数
- e. 在创建随机森林的时候，对 generlization error 使用的是无偏估计
- f. 训练速度快，可以得到变量重要性排序 两种：基于 OOB 误分率的增加量和基于分裂时的 GINI 下降量
- g. 在训练过程中，能够检测到 feature 间的互相影响
- h. 容易做成并行化方法
- i. 实现比较简单

局限性：

- 随机森林算法在训练和预测时都比较慢。
- 如果需要区分的类别十分多，随机森林的表现并不会很好。
- 当我们需要推断超出范围的独立变量或非独立变量，随机森林做得并不好，我们最好使用如 MARS 的算法。

参考

[1]Awesome Random Forest

<https://github.com/kjw0612/awesome-random-forest>

[2]forest.js

<https://github.com/tarobjtu/decision-tree>

[3]Ensemble methods

http://scikit-learn.org/dev/auto_examples/index.html#ensemble-methods

[4]Random forest

https://en.wikipedia.org/wiki/Random_forest

[5]随机森林 RF

<https://zhuanlan.zhihu.com/p/32989337>

