随机森林

1 随机森林

1.1 Bagging

AdaBoost 和梯度提升树都属于提升算法，即使用一系列的弱分类器**先后**进行训练，并将其进行组合，构成一个强分类器。在这些基分类器的训练过程中，后一个分类器需要前一个分类器的结果才可继续进行训练。

提升方法是**集成**方法的一种，集成方法还包括了 Bagging，Bagging 则是每次从数据集中采样出一部分的数据，在这些数据上训练出一个基分类器，最后将所有基分类器组合以得到一个强分类器。

其中需要注意的是：

1. 为了保证基分类器具有一定的差异，Bagging 使用了随机采样的方式，只从数据集中选取一部分数据进行基分类器的训练
2. 为了保证基分类器具有一定的学习能力，Bagging 对数据的采样方式为有交叠的采样子集，往往采用自助法
3. 与 Boosting 方法需要从前至后训练不同，Bagging 方法可以同时对基分类器进行训练，达到并行以加快训练速度
4. 使用自助法进行采样时，会有一部分样本未被抽取出，该部分样本可作为验证数据，进行超参数调试、决策树剪枝等工作

1.2 随机森林

随机森林是以决策树为基分类器构建 Bagging 集成的基础上，进一步在决策树训练过程中引入了随机属性选择。即对于决策树的每个节点，其先从该节点的属性集合中随机选择一个包含 k 个属性的子集，并从这个子集选取一个最优属性进行划分。

通过样本的随机采样和属性的随机选择，随机森林中的基分类器的多样性大大增加，其泛化能力因此也有提升。随机森林的算法如下：

输入：数据集 $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$，其中，$x_i \in R^n, y_i \in Y, i = 1, 2, ..., N$，基分类器个数 M，样本采样率 α 和属性采样率 $beta$。

1. 对 $m = 1, 2, ..., M$ 并行进行操作：

- 对 T 中的数据按概率 α 进行采样得到当前轮次的训练数据集 T_m
- 在数据集 T_m 上训练一个决策树，注意在对决策树节点进行划分时，需要先按 β 随机选择属性，再在剩下的属性选择最优的分割点
- 得到当前轮次的决策树模型 $f_m(x)$

2. 组合基分类器，得到最终分类器

$$F(x) = arg \max_{y \in Y} \sum_{m=1}^{M} I(f_m(x) = y)$$

输出：分类器 $F(x)$。

2 代码实现

考虑的数据集为 glass 数据集，具体可见 csv 文件。

2.1 sklearn 实现

准备数据：

```
import pandas as pd
import numpy as np
data = pd.read_csv('glass.csv')
X = data.iloc[:,:-1].values
Y = data.iloc[:,-1].values
```

随机森林的接口位于 sklearn.ensemble.RandomForestClassifier, 其文档可见**此处**。其中较为重要的参数有：

- n_estimators：基分类器的个数
- max_samples：样本采样比例
- max_features：属性采样比例
- 其余部分关于决策树的参数

```
from sklearn.ensemble import RandomForestClassifier
clf = [RandomForestClassifier(n_estimators=i,max_samples=0.8,max_features=0.8)\
       for i in range(5,101,5)]
for c in clf:
    c.fit(X,Y)
```

```
for c in clf:
    print('基分类器个数：',c.n_estimators,'准确率：',round(np.sum(c.predict(X)==Y)/len(Y),3))
```

```
## 基分类器个数： 5 准确率： 0.925
## 基分类器个数： 10 准确率： 0.958
## 基分类器个数： 15 准确率： 0.981
## 基分类器个数： 20 准确率： 0.977
## 基分类器个数： 25 准确率： 0.995
## 基分类器个数： 30 准确率： 0.995
## 基分类器个数： 35 准确率： 0.991
## 基分类器个数： 40 准确率： 1.0
## 基分类器个数： 45 准确率： 0.991
## 基分类器个数： 50 准确率： 0.995
## 基分类器个数： 55 准确率： 0.995
## 基分类器个数： 60 准确率： 1.0
## 基分类器个数： 65 准确率： 0.995
## 基分类器个数： 70 准确率： 1.0
## 基分类器个数： 75 准确率： 1.0
## 基分类器个数： 80 准确率： 1.0
## 基分类器个数： 85 准确率： 1.0
## 基分类器个数： 90 准确率： 1.0
## 基分类器个数： 95 准确率： 0.995
## 基分类器个数： 100 准确率： 1.0
```

2.2 随机森林

随机森林需要用到决策树，且该决策树在分裂节点时，只需计算部分属性的数值。从 utils 文件中导入该特定决策树即可。

```
class RandomForest:

    def __init__(self,n_estimators=100,subsample=1,colsample=1,\
                 max_depth=float('inf'),min_samples_leaf=1):
        self.n_estimators = n_estimators
        self.subsample = subsample
        self.colsample = colsample
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf
```

```
    def fit(self,X,Y):
        self.clf_list = []
        for m in range(self.n_estimators):
            idx = np.random.permutation([i for i in range(len(Y))])
            idx = list(idx)[:round(self.subsample*len(Y))]
            X_m, Y_m = X[idx,:],Y[idx]
            clf = ClassificationTree(max_depth=self.max_depth,min_samples_leaf=self.min_samples_le
            clf.fit(X_m,Y_m,col_num=round(X.shape[1]*self.colsample))
            self.clf_list.append(clf)

    def find_most_frequent(self,x):
        return np.bincount(list(x)).argmax()

    def predict(self,new_X):
        out = np.concatenate([clf.predict(new_X) for clf in self.clf_list],axis=1)
        return np.apply_along_axis(self.find_most_frequent,axis=1,arr=out)
```

结果如下：

```
from utils import ClassificationTree
clf = [RandomForest(n_estimators=i) for i in range(1,10)]
for c in clf:
    c.fit(X,Y)
    print('基分类器个数: ',c.n_estimators,'准确率: ',round(np.sum(c.predict(X)==Y.reshape(-1))/len(Y
```

```
## 基分类器个数:  1 准确率:  0.972
## 基分类器个数:  2 准确率:  0.888
## 基分类器个数:  3 准确率:  0.972
## 基分类器个数:  4 准确率:  0.981
## 基分类器个数:  5 准确率:  1.0
## 基分类器个数:  6 准确率:  0.981
## 基分类器个数:  7 准确率:  0.995
## 基分类器个数:  8 准确率:  0.995
## 基分类器个数:  9 准确率:  0.995
```