随机森林

随机森林（Random Forest）是一种集成学习算法，通过构建多个决策树来提高模型的性能和泛化能力。它是 Bagging（Bootstrap Aggregating）的一种特定形式，通过在训练过程中引入随机性，减少了过拟合的风险，并且通常在处理大量特征和高维数据时表现良好。

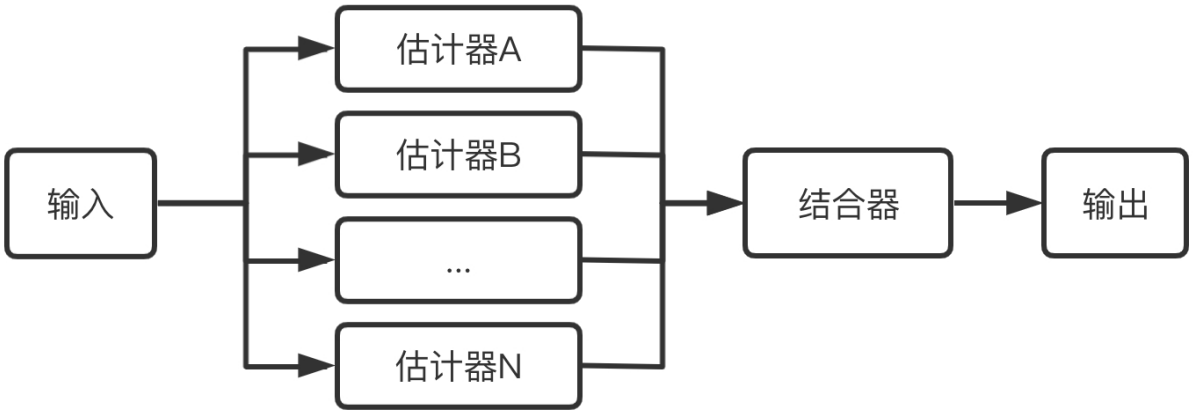
以下是随机森林的主要特点和工作原理：

1. **Bootstrap抽样：** 对于给定的训练数据，随机森林使用Bootstrap抽样（有放回抽样）来创建多个不同的训练子集。这意味着每棵决策树都是在不同的数据子集上训练的。
2. **随机特征选择：** 在每次划分决策树的节点时，随机森林不考虑所有特征，而是从特征集中随机选择一部分特征用于划分。这有助于降低树之间的相关性，增加模型的多样性。
3. **构建多个决策树：** 随机森林由多个决策树组成，每个树都是在不同的训练数据和随机特征子集上训练的。这些决策树是相互独立的。
4. **投票或平均：** 对于分类问题，随机森林的预测结果通常是所有树的投票结果；对于回归问题，预测结果可以是所有树的平均值。

有一个成语叫集思广益，指的是集中群众的智慧，广泛吸收有益的意见。在机器学习算法中也有类似的思想，被称为集成学习（Ensemble learning）。

集成学习

集成学习通过训练学习出多个估计器，当需要预测时通过结合器将多个估计器的结果整合起来当作最后的结果输出



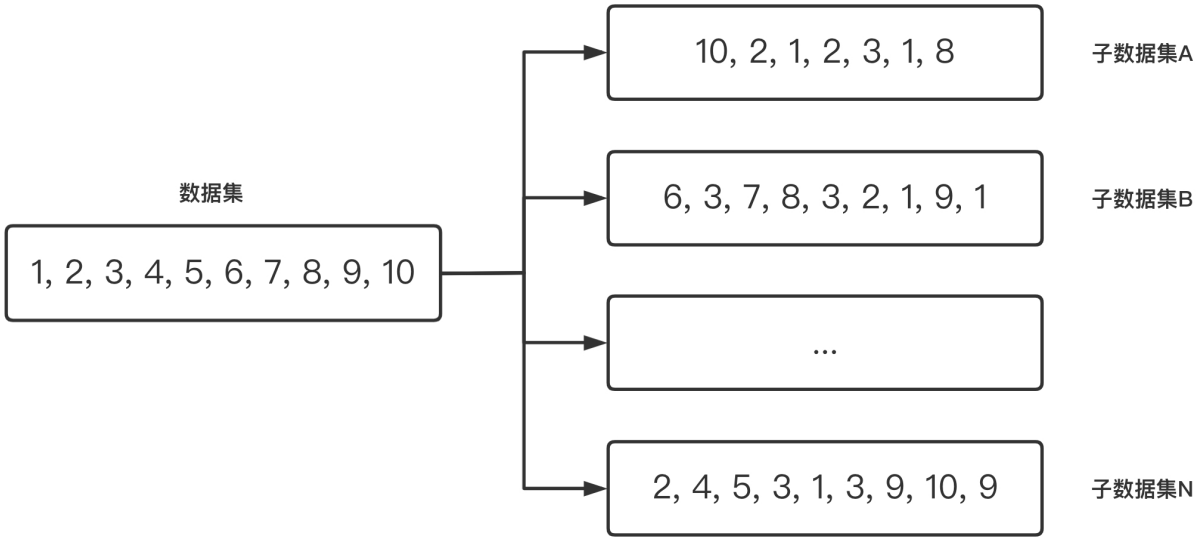
集成学习的优势是提升了单个估计器的通用性与鲁棒性，比单个估计器拥有更好的预测性能。集成学习的另一个特点是能方便的进行并行化操作。

agging算法

Bagging 算法3是一种集成学习算法，其全称为自助聚集算法（Bootstrap aggregating），顾名思义算

法由 Bootstrap 与 Aggregating 两部分组成。

图 2-2 展示了Bagging 算法使用自助取样（Bootstrapping4）生成多个子数据的示例



算法的具体步骤为：假设有一个大小为 N 的训练数据集，每次从该数据集中有放回的取选出大小为 M 的子数据集，一共选 K 次，根据这 K 个子数据集，训练学习出 K 个模型。当要预测的时候，使用这 K 个模型进行预测，再通过取平均值或者多数分类的方式，得到最后的预测结果。

随机森林

将多个决策树结合在一起，每次数据集是随机有放回的选出，同时随机选出部分特征作为输入，所以该算法被称为随机森林算法。可以看到随机森林算法是以决策树为估计器的Bagging算法。

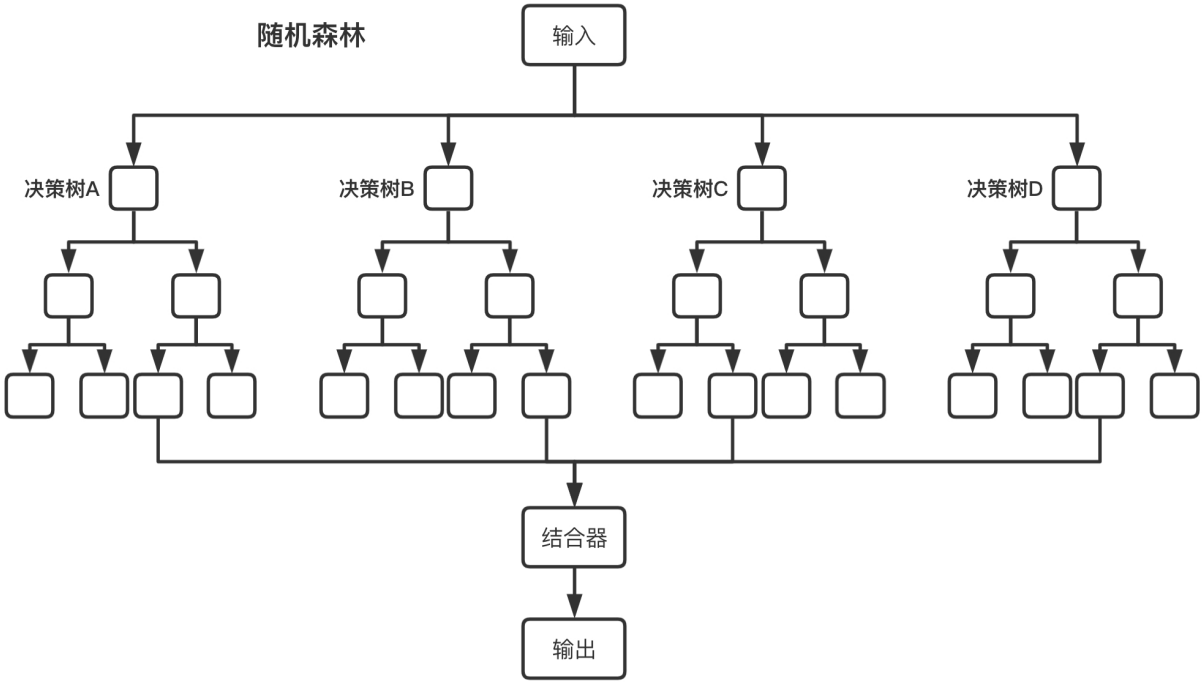


图2-3展示了随机森林算法的具体流程，其中结合器在分类问题中，选择多数分类结果作为最后的结果，在回归问题中，对多个回归结果取平均值作为最后的结果。

使用Bagging算法能降低过拟合的情况，从而带来了更好的性能。单个决策树对训练集的噪声非常敏感，但通过Bagging算法降低了训练出的多颗决策树之间关联性，有效缓解了上述问题。

算法

假设训练集 T 的大小为 N ，特征数目为 M ，随机森林的大小为 K ，随机森林算法的具体步骤如下：

遍历随机森林的大小 K 次：

从训练集 T 中有放回抽样的方式，取样N 次形成一个新子训练集 D

随机选择 m 个特征，其中 m < M

使用新的训练集 D 和 m 个特征，学习出一个完整的决策树

得到随机森林

优缺点

优点：

1. 对于很多种资料，可以产生高准确度的分类器
2. 可以处理大量的输入变量

3. 可以在决定类别时，评估变量的重要性
4. 在建造森林时，可以在内部对于一般化后的误差产生不偏差的估计
5. 包含一个好方法可以估计丢失的资料，并且如果有很大一部分的资料丢失，仍可以维持准确度
6. 对于不平衡的分类资料集来说，可以平衡误差

缺点

1. 牺牲了决策树的可解释性
2. 在某些噪音较大的分类或回归问题上会过拟合
3. 在多个分类变量的问题中，随机森林可能无法提高基学习器的准确性

代码实现：

使用 Python 实现随机森林分类：

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier

class rfc:
    """
    随机森林分类器
    """

    def __init__(self, n_estimators = 100, random_state = 0):
        # 随机森林的大小
        self.n_estimators = n_estimators
        # 随机森林的随机种子
        self.random_state = random_state

    def fit(self, X, y):
        """
        随机森林分类器拟合
        """
        self.y_classes = np.unique(y)
        # 决策树数组
        dts = []
        n = X.shape[0]
        rs = np.random.RandomState(self.random_state)
        for i in range(self.n_estimators):
            # 创建决策树分类器
            dt = DecisionTreeClassifier(random_state=rs.randint(np.iinfo(np.int32).max), max_features = "auto")
            # 根据随机生成的权重，拟合数据集
            dt.fit(X, y, sample_weight=np.bincount(rs.randint(0, n, n), minlength = n))
            dts.append(dt)
        self.trees = dts

    def predict(self, X):
        """
        随机森林分类器预测
        """
        # 预测结果数组
        probas = np.zeros((X.shape[0], len(self.y_classes)))
```

```
        for i in range(self.n_estimators):
            # 决策树分类器
            dt = self.trees[i]
            # 依次预测结果可能性
            probas += dt.predict_proba(X)
        # 预测结果可能性取平均
        probas /= self.n_estimators
        # 返回预测结果
        return self.y_classes.take(np.argmax(probas, axis = 1), axis = 0)
```

使用 Python 实现随机森林回归：

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

class rfr:
    """
    随机森林回归器
    """

    def __init__(self, n_estimators = 100, random_state = 0):
        # 随机森林的大小
        self.n_estimators = n_estimators
        # 随机森林的随机种子
        self.random_state = random_state

    def fit(self, X, y):
        """
        随机森林回归器拟合
        """
        # 决策树数组
        dts = []
        n = X.shape[0]
        rs = np.random.RandomState(self.random_state)
        for i in range(self.n_estimators):
            # 创建决策树回归器
            dt = DecisionTreeRegressor(random_state=rs.randint(np.iinfo(np.int32).max), max_features = "auto")
            # 根据随机生成的权重，拟合数据集
            dt.fit(X, y, sample_weight=np.bincount(rs.randint(0, n, n), minlength = n))
            dts.append(dt)
        self.trees = dts

    def predict(self, X):
        """
        随机森林回归器预测
        """
        # 预测结果
        ys = np.zeros(X.shape[0])
        for i in range(self.n_estimators):
            # 决策树回归器
            dt = self.trees[i]
            # 依次预测结果
            ys += dt.predict(X)
        # 预测结果取平均
        ys /= self.n_estimators
        return ys
```

用sklearn

scikit-learn 实现随机森林分类：

```
from sklearn.ensemble import RandomForestClassifier

# 随机森林分类器
clf = RandomForestClassifier(n_estimators = 100, random_state = 0)
# 拟合数据集
clf = clf.fit(X, y)
```

scikit-learn 实现随机森林回归：

```
from sklearn.ensemble import RandomForestRegressor

# 随机森林回归器
clf = RandomForestRegressor(n_estimators = 100, random_state = 0)
# 拟合数据集
clf = clf.fit(X, y)
```