

# 实验二决策树和随机森林

20201060287 李昂

## 实验内容

根据已有数据训练决策树和随机森林

## 环境要求

Python,numpy支持多维度的数组和矩阵运算,pandas数据处理和分析工具,Matplotlib图形化工具,sklearn机器学习库

```
import pandas as pd
import sklearn.ensemble as ensemble
from sklearn.model_selection import cross_val_score, GridSearchCV, train_test_split
import sklearn.tree
import pydotplus
import pprint
```

## 任务一：决策树

### 题目：

本任务中你将使用决策树来预测泰坦尼克号的乘客是否能够生存。

通过乘客的相关信息来构建合适的决策树,以此判断乘客的生还情况,并尝试不同的决策树参数,观察其对决策树的影响（如节点分裂标准,树的最大深度等）,尝试使用网格搜索的方法找到较优的参数,将决策树可视化。

文件ex2data.csv包含我们的线性回归问题的数据集,数据参数如下:

PassengerId（乘客ID），Name（姓名），Ticket（船票信息）；

Survived（获救情况）其值为1或0，代表着获救或未获救；

Pclass（乘客等级），Sex（性别）值为male或者female；

Embarked（登船港口）其值为S，Q，C；

Age（年龄），SibSp（堂兄弟姐妹个数），Parch（父母与小孩的个数）；

Fare（票价）是数值型数据；Cabin（船舱）则为文本型数据。

决策树相关参数和网格搜索参数代码介绍在demo.py可见。

1. 请将70%的数据用作训练集，30%的数据用作测试集，使用留出法对以上模型进行验证
2. 请对生成的决策树进行剪枝，并可视化剪枝前/剪枝后的决策树，比较两者区别

### 代码：

```
# -*- coding: utf-8 -*-
# @Time : 3/27/23 16:07
# @Author : ANG

import pandas as pd
import sklearn.tree
from sklearn.model_selection import train_test_split, GridSearchCV
import pydotplus
import pprint

def loaddata(path):
    """
    本函数用于数据预处理,为训练决策树模型做准备,对于各类数据的具体处理如下:

    PassengerId（乘客ID），Name（姓名），Ticket（船票信息），Cabin（船舱）对于是否存活意义不大，不加入后续的分析；
    Survived（获救情况）变量为因变量，其值只有两类1或0，代表着获救或未获救，保持不变；
    Pclass（乘客等级），Sex（性别），Embarked（登船港口）是明显的类别型数据，保持不变；
    Age（年龄），SibSp（堂兄弟姐妹个数），Parch（父母与小孩的个数）是隐性的类别型数据，保持不变；
    Fare（票价）是数值型数据；Cabin（船舱）为文本型数据，保持不变；
    Age（年龄），和Embarked（登船港口）信息存在缺失数据，进行填充处理；

    :param path: 所需数据的绝对路径
```

```

:return: DataFrame格式的数据列表

"""
dataset = pd.read_csv(path, index_col=None)

# 删除无用的列
dataset.drop(["PassengerId", "Name", "Ticket"], axis=1, inplace=True)
dataset.drop(["Cabin"], axis=1, inplace=True)

# 将Sex（性别）列的值转换为数值型数据
dataset["Sex"].replace({"male": 0, "female": 1}, inplace=True)

# 将Embarked（登船港口）列的值转换为数值型数据
dataset["Embarked"].replace({"S": 0, "C": 1, "Q": 2}, inplace=True)

# 处理缺失值
# Age（年龄）缺失值用平均值填充
dataset["Age"].fillna(dataset["Age"].mean(), inplace=True)
# Embarked（登船港口）缺失值用众数填充
dataset["Embarked"].fillna(dataset["Embarked"].mode()[0], inplace=True)

# 划分数据集和标签集
sample = dataset.iloc[:, 1:]
label = dataset.iloc[:, 0]

# 将70%的数据用作训练集，30%的数据用作测试集
sample_train, sample_test, label_train, label_test = train_test_split(sample, label, test_size=0.3, random_state=0)

return sample_train, sample_test, label_train, label_test

def DecisionTree():
    """
    本函数用于决策树模型的训练，具体说明如下：

    :param

    criterion: 特征选择标准 【entropy, gini】
        默认gini，即CART算法

    splitter: 特征划分标准 【best, random】
        best在特征的所有划分点中找出最优的划分点，random随机的在部分划分点中找局部最优的划分点
        默认的‘best’适合样本量不大的时候，而如果样本数据量非常大，此时决策树构建推荐‘random’

    max_depth: 决策树最大深度 【int, None】
        默认值是‘None’
        一般数据比较少或者特征少的时候可以不用管这个值
        如果模型样本数量多，特征也多时，推荐限制这个最大深度，具体取值取决于数据的分布
        常用的可以取值10-100之间，常用来解决过拟合

    min_samples_split: 内部节点（即判断条件）再划分所需最小样本数 【int, float】
        默认值为2。如果是int，则取传入值本身作为最小样本数
        如果是float，则取Ceil(min_samples_split*样本数量)作为最小样本数（向上取整）

    min_samples_leaf: 叶子节点（即分类）最少样本数
        如果是int，则取传入值本身作为最小样本数
        如果是float，则取ceil(min_samples_leaf*样本数量)的值作为最小样本数
        这个值限制了叶子节点最少的样本数，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝

    min_weight_fraction_leaf: 叶子节点（即分类）最小的样本权重和 【float】
        这个值限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝
        默认是0，就是不考虑权重问题，所有样本的权重相同
        一般来说如果我们有多样本有缺失值或者分类树样本的分布类别偏差很大，就会引入样本权重，这时就要注意此值

    max_features: 在划分数据集时考虑的最多的特征值数量 【int值】
        在每次split时最大特征数；【float值】表示百分数，即（max_features*n_features）

    random_state: 【int, randomState instance, None】 默认是None

    max_leaf_nodes: 最大叶子节点数 【int, None】
        通过设置最大叶子节点数，可以防止过拟合
        默认值None，默认情况下不设置最大叶子节点数
        如果加了限制，算法会建立在最大叶子节点数内最优的决策树
        如果特征不多，可以不考虑这个值，但是如果特征多，可以加限制，具体的值可以通过交叉验证得到

    min_impurity_decrease: 节点划分最小不纯度 【float】
        默认值为‘0’ 限制决策树的生长
        节点的不纯度（基尼系数，信息增益，均方差，绝对差）必须大于这个阈值，否则该节点不再生成子节点

    min_impurity_split（已弃用）: 信息增益的阈值
        决策树在创建分支时，信息增益必须大于这个阈值，否则不分裂
        （从版本0.19开始不推荐使用：min_impurity_split已被弃用，以0.19版本中的min_impurity_decrease取代）
        （min_impurity_split的默认值将在0.23版本中从1e-7变为0，并且将在0.25版本中删除，请改用min_impurity_decrease）

    class_weight: 类别权重 【dict, list of dicts, balanced】
        默认为None（不适用于回归树，sklearn.tree.DecisionTreeRegressor）
        指定样本各类别的权重，主要是为了防止训练集某些类别的样本过多，导致训练的决策树过于偏向这些类别
    """

```

```

        balanced, 算法自己计算权重, 样本量少的类别所对应的样本权重会更高。如果样本类别分布没有明显的偏倚, 则可以不管这个参数

    presort : bool, 默认是False, 表示在进行拟合之前, 是否预先对数据来加快树的构建
        对于数据集非常庞大的分类, presort=True将导致整个分类变得缓慢
        当数据集较小, 且树的深度有限制, presort=True才会加速分类

    ccp_alpha : 将选择成本复杂度最大且小于ccp_alpha的子树
        默认情况下, 不执行修剪

    :return: 决策树模型

"""
decision_tree = sklearn.tree.DecisionTreeClassifier()
return decision_tree

def DecisionTree_self_adjust():
    """
    根据情景需求, 手动调节参数:
    本次数据为离散数据, 而entropy对离散数据的处理效果更好, 所以选择entropy作为特征选择标准;
    min_samples_split: 4, 即内部节点再划分所需最小样本数为4, 防止过拟合;
    min_samples_leaf: 2, 即叶子节点最少样本数为2, 防止过拟合;
    :return: 决策树模型

    """
    decision_tree = sklearn.tree.DecisionTreeClassifier(criterion="entropy", min_samples_split=4, min_samples_leaf=2)
    return decision_tree

def DecisionTree_adjust_by_gridsearch():
    """
    通过网格搜索调整参数
    :return: 决策树模型

    """
    decision_tree = sklearn.tree.DecisionTreeClassifier()
    parameters = {'criterion': ('gini', 'entropy'),
                  'splitter': ('best', 'random'),
                  'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                  'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                  'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}

    decision_tree = GridSearchCV(decision_tree, param_grid=parameters, return_train_score=True)
    return decision_tree

def main():
    """
    此函数用于测试决策树模型
    :return: 测试结果

    """

    sample_train, sample_test, label_train, label_test = loaddata(
        "/Users/wallanceleon/Desktop/机器学习/机器学习实验/20201060287-李昂-实验二/Dataset/ex2data.csv")

    # 生成决策树模型
    decision_tree = DecisionTree()
    decision_tree.fit(sample_train, label_train)

    # 生成主观调整参数后的决策树模型
    decision_tree_self_adjust = DecisionTree_self_adjust()
    decision_tree_self_adjust.fit(sample_train, label_train)

    # 生成通过网格搜索调整参数后的决策树模型
    decision_tree_adjust_by_gridsearch = DecisionTree_adjust_by_gridsearch()
    decision_tree_adjust_by_gridsearch.fit(sample_train, label_train)

    # 可视化决策树
    dot_data = sklearn.tree.export_graphviz(decision_tree, out_file=None)
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_pdf("decision_tree.pdf")

    # 可视化根据训练集主观调整参数后的决策树
    dot_data = sklearn.tree.export_graphviz(decision_tree_self_adjust, out_file=None)
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_pdf("decision_tree_self_adjust.pdf")

    # 可视化通过网格搜索调整参数后的决策树
    decision_tree_adjust_by_gridsearch = decision_tree_adjust_by_gridsearch.best_estimator_
    dot_data = sklearn.tree.export_graphviz(decision_tree_adjust_by_gridsearch, out_file=None)
    graph = pydotplus.graph_from_dot_data(dot_data)
    graph.write_pdf("decision_tree_adjust_by_gridsearch.pdf")

    # 打印决策树的最优参数
    print('最优参数为:')

```

```
pprint.pprint(decision_tree_adjust_by_gridsearch.get_params())

# 评估模型
print("未剪枝和预剪枝的决策树模型:")
print("决策树模型的准确率为:", decision_tree.score(sample_test, label_test))
print("根据训练集主观调整参数进行预剪枝的决策树的准确率为:",
      decision_tree_self_adjust.score(sample_test, label_test))
print("通过网格搜索调整参数进行预剪枝的决策树的准确率为:",
      decision_tree_adjust_by_gridsearch.score(sample_test, label_test))

if __name__ == '__main__':
    main()
```

## 分析：

对于本问题，由于数据集中存在缺项和不完备问题，以及部分数据与问题分析的相关性不大，因此需要对数据进行预处理，在这里使用loaddata () 函数进行数据预处理。

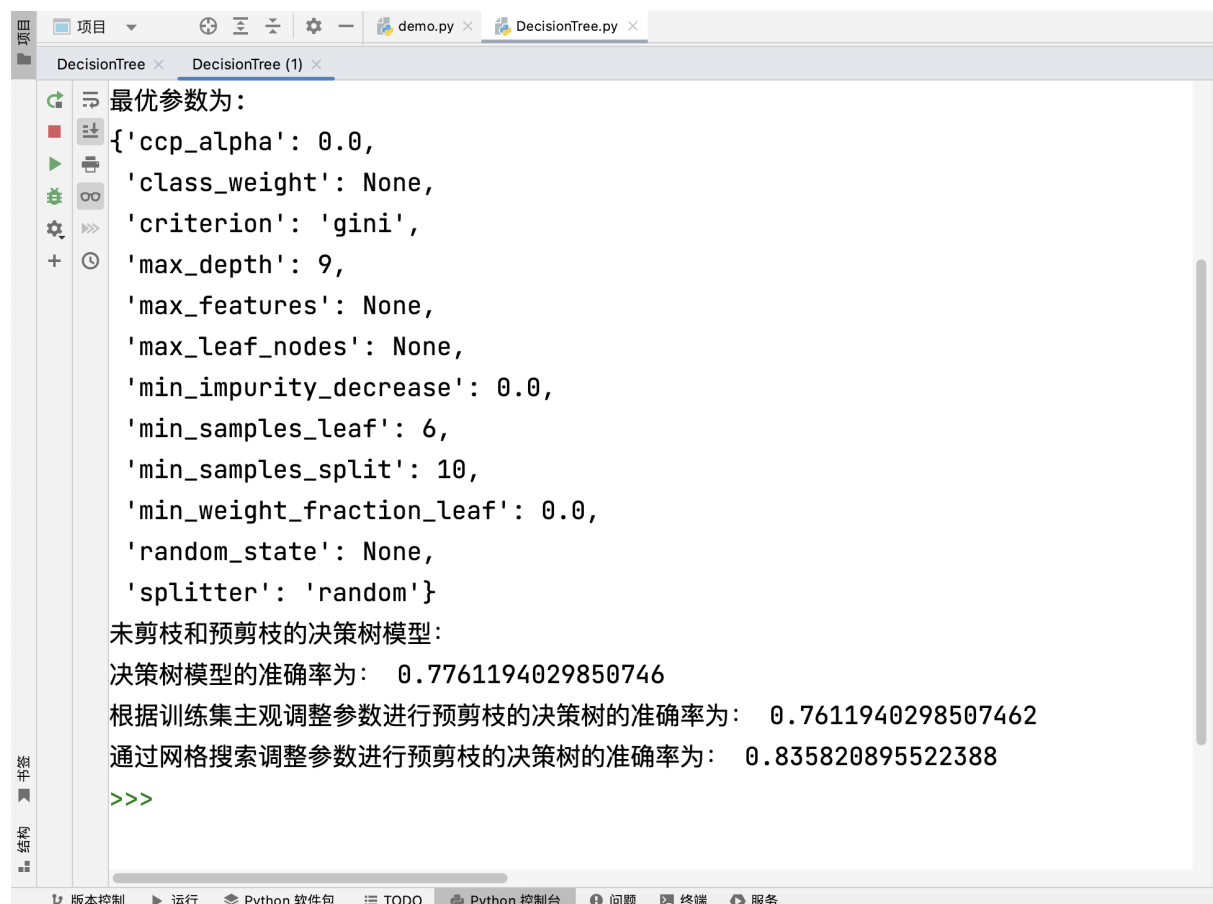
完成数据处理后，我们可以使用sklearn库中的DecisionTreeClassifier函数来构建决策树模型。决策树的构建过程中，criterion表示特征选择的标准，splitter表示特征划分标准，max\_depth表示决策树的最大深度，min\_samples\_split表示节点再划分所需最小样本数，min\_samples\_leaf表示叶子节点最少样本数，通过调整相关参数，构建决策树。

通过网格搜索的方法，对这些参数进行调整，以得到最优模型。这里需要注意max\_depth最好不要设置过高，以免出现过拟合的情况。另外，使用决策树的可视化工具，对生成的决策树进行可视化，以便于我们更好地理解模型的构建过程和决策规则。

在训练好模型后，我们需要使用测试集对模型进行验证。采用留出法来对模型进行验证，将70%的数据用作训练集，30%的数据用作测试集。对生成的决策树进行剪枝，并可视化剪枝前/剪枝后的决策树。

## 实验结果：

见附件：decisiontree.pdf、decisiontree\_self\_sdjust.pdf、decisiontree\_adjust\_by\_gridsearch.pdf



```
最优参数为:
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 9,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 6,
 'min_samples_split': 10,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'random'}
未剪枝和预剪枝的决策树模型:
决策树模型的准确率为: 0.7761194029850746
根据训练集主观调整参数进行预剪枝的决策树的准确率为: 0.7611940298507462
通过网格搜索调整参数进行预剪枝的决策树的准确率为: 0.835820895522388
>>>
```

## 任务二：随机森林

### 题目：

本任务中你将使用随机森林来预测泰坦尼克号的乘客是否能够生存。

通过乘客的相关信息来构建合适的随机森林，以此判断乘客的生还情况，并尝试不同的随机森林参数，观察其对随机森林的影响（如森林中的决策树数量，节点分裂标准，树的最大深度等），尝试使用网格搜索的方法找到较优的参数。

数据集同上。随机森林相关参数代码介绍在demo.py可见。请使用10折交叉验证法对以上模型进行验证。

### 代码

```
# -*- coding: utf-8 -*-
# @Time : 3/26/23 20:31
# @Author : ANG

import pandas as pd
import sklearn.ensemble as ensemble
from sklearn.model_selection import cross_val_score, GridSearchCV

def loaddata(path):
    """
    本函数用于数据预处理, 为训练决策树模型做准备, 对于各类数据的具体处理如下:

    PassengerId (乘客ID), Name (姓名), Ticket (船票信息), Cabin (船舱) 对于是否存活意义不大, 不加入后续的分析;
    Survived (获救情况) 变量为因变量, 其值只有两类1或0, 代表着获救或未获救, 保持不变;
    Pclass (乘客等级), Sex (性别), Embarked (登船港口) 是明显的类别型数据, 保持不变;
    Age (年龄), SibSp (堂兄弟姊妹个数), Parch (父母与小孩的个数) 是隐性的类别型数据, 保持不变;
    Fare (票价) 是数值型数据; Cabin (船舱) 为文本型数据, 保持不变;
    Age (年龄), 和Embarked (登船港口) 信息存在缺失数据, 进行填充处理;

    :param path: 所需数据的绝对路径
    :return: DataFrame格式的数据列表

    """
    dataset = pd.read_csv(path, index_col=None)

    # 删除无用的列
    dataset.drop(["PassengerId", "Name", "Ticket"], axis=1, inplace=True)
    dataset.drop(["Cabin"], axis=1, inplace=True)

    # 将Sex (性别) 列的值转换为数值型数据
    dataset["Sex"].replace({"male": 0, "female": 1}, inplace=True)

    # 将Embarked (登船港口) 列的值转换为数值型数据
    dataset["Embarked"].replace({"S": 0, "C": 1, "Q": 2}, inplace=True)

    # 处理缺失值
    # Age (年龄) 缺失值用平均值填充
    dataset['Age'].fillna(dataset['Age'].mean(), inplace=True)
    # Embarked (登船港口) 缺失值用众数填充
    dataset['Embarked'].fillna(dataset['Embarked'].mode()[0], inplace=True)

    # 划分数据集和标签集
    sample = dataset.iloc[:, 1:]
    label = dataset.iloc[:, 0]

    return sample, label

def RandomForest(sample, label):
    """
    本函数用于训练随机森林模型, 并对模型进行评估, 输出模型的评估结果
    :param sample: 数据集
    :param label: 标签集
    :return: randomforest模型

    """
    random_forest = ensemble.RandomForestClassifier()
    # 使用10折交叉验证模型评估, 输出每次验证的准确率和平均准确率
    scores = cross_val_score(random_forest, sample, label, cv=10)
    print("随机森林模型的平均准确率为: ", scores.mean())

def RandomForest_self_adjust(sample, label):
    """
    本函数用于调整随机森林模型的参数, 并对模型进行评估, 输出模型的评估结果
    对于随机森林模型的参数调整, 主要调整的参数有:
    n_estimators: 森林中决策树的数量, 默认100
    表示这是森林中树木的数量, 即基评估器的数量
    """
```

这个参数对随机森林模型的精确性影响是单调的，`n_estimators`越大，模型的效果往往越好。但是相应的，任何模型都有决策边界，`n_estimators`达到一定的程度之后，随机森林的精确性往往不在上升或开始波动。并且，`n_estimators`越大，需要的计算量和内存也越大，训练的时间也会越来越长。对于这个参数，我们是渴望在训练难度和模型效果之间取得平衡。

`criterion`：分裂节点所用的标准，可选“`gini`”，“`entropy`”，默认“`gini`”

`max_depth`：树的最大深度  
如果为`None`，则将节点展开，直到所有叶子都是纯净的（只有一个类）  
或者直到所有叶子都包含少于`min_samples_split`个样本。默认是`None`

`min_samples_split`：拆分内部节点所需的最少样本数  
如果为`int`，则将`min_samples_split`视为最小值  
如果为`float`，则`min_samples_split`是一个分数，而`ceil(min_samples_split * n_samples)`是每个拆分的最小样本数，默认是2

`min_samples_leaf`：在叶节点处需要的最小样本数  
仅在任何深度的分割点在左分支和右分支中的每个分支上至少留下`min_samples_leaf`个训练样本时，才考虑  
这可能具有平滑模型的效果，尤其是在回归中。如果为`int`，则将`min_samples_leaf`视为最小值  
如果为`float`，则`min_samples_leaf`是分数，而`ceil(min_samples_leaf * n_samples)`是每个节点的最小样本数，默认是1

`min_weight_fraction_leaf`：在所有叶节点处（所有输入样本）的权重总和中的最小加权分数  
如果未提供`sample_weight`，则样本的权重相等

`max_features`：寻找最佳分割时要考虑的特征数量：  
如果为`int`，则在每个拆分中考虑`max_features`个特征  
如果为`float`，则`max_features`是一个分数，并在每次拆分时考虑`int(max_features * n_features)`个特征  
如果为“`auto`”，则`max_features = sqrt(n_features)`  
如果为“`sqrt`”，则`max_features = sqrt(n_features)`  
如果为“`log2`”，则`max_features = log2(n_features)`  
如果为`None`，则`max_features = n_features`  
注意：在找到至少一个有效的节点样本分区之前，分割的搜索不会停止，即使它需要有效检查多个`max_features`功能也是如此

`max_leaf_nodes`：最大叶子节点数，整数，默认是`None`

`min_impurity_decrease`：如果分裂指标的减少量大于该值，则进行分裂

`min_impurity_split`：决策树生长的最小纯净度。默认是0。

`bootstrap`：是否进行bootstrap操作，`bool`，默认`True`  
如果`bootstrap=True`，将每次有放回地随机选取样本，只有在`extra-trees`中，`bootstrap=False`

`oob_score`：是否使用袋外样本来估计泛化精度，默认`False`

`n_jobs`：并行计算数，默认是`None`

`random_state`：控制bootstrap的随机性以及选择样本的随机性

`verbose`：在拟合和预测时控制详细程度，默认是0

`class_weight`：每个类的权重，可以用字典的形式传入`{class_label: weight}`  
如果选择了“`balanced`”，则输入的权重为`n_samples / (n_classes * np.bincount(y))`

`ccp_alpha`：将选择成本复杂度最大且小于`ccp_alpha`的子树  
默认情况下，不执行修剪

`max_samples`：如果`bootstrap=True`，则从`X`抽取以训练每个基本分类器的样本数  
如果为`None`（默认），则抽取`X.shape[0]`样本  
如果为`int`，则抽取`max_samples`样本  
如果为`float`，则抽取`max_samples * X.shape[0]`个样本

:param sample: 数据集  
:param label: 标记集  
:return: randomforest模型

```
"""
random_forest = ensemble.RandomForestClassifier(criterion='entropy', max_features='sqrt')
# 使用10折交叉验证模型评估，输出每次验证的准确率和平均准确率
scores = cross_val_score(random_forest, sample, label, cv=10)
print("主观调整参数后的随机森林模型的平均准确率为：", scores.mean())
```

```
def RandomForest_adjust_by_gridsearch(sample, label):
    """
    使用网格搜索调整随机森林模型参数
    :param sample: 数据集
    :param label: 标记集
    :return: 调整后的随机森林模型
    """
    # 定义网格搜索的参数
    param_grid = {
        'n_estimators': [10, 50, 100, 200],
        'criterion': ['gini', 'entropy'],
        'max_features': ['sqrt', 'log2'],
        'max_depth': [2, 4, 6, 8, 10],
        'min_samples_split': [2, 4, 6, 8, 10],
        'min_samples_leaf': [1, 2, 3, 4, 5]
```

```

}
# 定义随机森林模型
random_forest = ensemble.RandomForestClassifier()
# 使用网格搜索调整模型参数
grid_search = GridSearchCV(random_forest, param_grid, cv=10, scoring='accuracy', n_jobs=-1)
grid_search.fit(sample, label)
print("经过网格搜索调整参数后的随机森林模型的平均准确率为:", grid_search.best_score_)

def main():
    """
    主函数, 调用上述函数, 生成随机森林模型
    :return:

    """
    sample, label = loaddata(
        "/Users/wallanceleon/Desktop/机器学习/机器学习实验/20201060287-李昂-实验二/Dataset/ex2data.csv")

    RandomForest(sample, label)

    RandomForest_self_adjust(sample, label)

    RandomForest_adjust_by_gridsearch(sample, label)

if __name__ == '__main__':
    main()

```

## 分析：

首先，我们加载了一个数据集，并对数据进行了一些预处理，如删除无用的列、转换数据类型和填充缺失值。使用交叉验证评估了随机森林模型的性能，并对模型进行了参数调整`n_estimators`、`criterion`、`max_features`等。使用网格搜索进一步调整了模型的参数，并展示了最佳的模型参数和准确率。有了题目一的基础，任务二较为简单，但由于随机森林计算量较大，需要一定的运算时间。

## 实验结果：

```

/Users/wallanceleon/anaconda3/envs/MachineLearning/bin/python /Applications/PyCharm.app/Content:
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/wallanceleon/Desktop/机器学习/机器学习实验/20201060287-李昂-实验二'])

Python 控制台
随机森林模型的平均准确率为： 0.8104119850187266
主观调整参数后的随机森林模型的平均准确率为： 0.8149063670411983
经过网格搜索调整参数后的随机森林模型的平均准确率为： 0.8429463171036204

>>>

```

## 实验总结

决策树和随机森林是机器学习中常用的分类和回归算法。决策树是基于树形结构的一种分类算法，通过对数据集进行划分，构建一棵树来进行分类。随机森林是由多个决策树组成的集成算法，通过随机抽样和特征选择来减小决策树的方差和提高模型的泛化能力。

首先进行了数据预处理，包括删除无用的列、转换数据类型和填充缺失值等。然后使用交叉验证对随机森林模型进行了性能评估，并对模型进行了参数调整。接着使用网格搜索进一步调整了模型的参数，并展示了最佳的模型参数和准确率。

除了上述内容，我们还可以对随机森林进行更深入的研究，如特征重要性分析、随机森林对缺失值的处理、随机森林对不平衡数据的处理等。总之，决策树和随机森林是一种强大的机器学习算法，可以用于分类和回归任务，并具有较高的准确率和泛化能力。

