# 使用多种分类算法判断井字棋状态

## 摘要

本文使用了多种分类算法对Tic-Tac-Toe数据集进行分类。本文采用了sklearn库中的数据集读取函数将数据集导入Python环境，并对数据集进行了处理和分析，以此研究不同分类算法的分类效果和优缺点。采用的算法包括K近邻、ID3算法和随机森林。在实验过程中，我们将数据集划分为训练集和测试集，利用训练集训练模型，并在测试集上进行性能评估。通过对算法进行调参，本文找到了每种算法的最优参数，使得其具有良好表现；通过对不同算法的比较，本文发现在该数据集上，ID3算法表现最优，其次是随机森林，K近邻算法表现最差。

目录

[使用多种分类算法判断井字棋状态 1](#_Toc137654500)

[摘要 1](#_Toc137654501)

[一、 数据集特征与数据预处理 3](#_Toc137654502)

[1.1 数据特征 3](#_Toc137654503)

[1.2 二值化处理与划分数据集 4](#_Toc137654504)

[二、 算法原理与分类实验 5](#_Toc137654505)

[2.1 最近邻算法 5](#_Toc137654506)

[2.2 ID3算法 7](#_Toc137654507)

[2.3 随机森林 8](#_Toc137654508)

[三、 实验结果与对比分析 9](#_Toc137654509)

[3.1 最近邻算法 9](#_Toc137654510)

[3.2 ID3算法 10](#_Toc137654511)

[3.3 随机森林 12](#_Toc137654512)

[3.4 评估不同分类算法 13](#_Toc137654513)

[四、 总结 14](#_Toc137654514)

[附录 14](#_Toc137654515)

[实现B+树 19](#_Toc137654516)

## 数据集特征与数据预处理

### 数据特征

Tic-Tac-Toe是一个著名的双人游戏，近年来随着计算机技术的发展，越来越多的人开始使用机器学习算法来对Tic-Tac-Toe进行分类和预测。利用机器学习算法对Tic-Tac-Toe进行分类，可以帮助我们更好地理解该游戏的规律，从而提高我们的游戏水平。同时，这项研究的应用也可以扩展到其他领域，例如棋类游戏的人机对战等。

Tic-Tac-Toe是一种二人博弈游戏，也叫井字棋。数据集是基于该游戏的结束状态（胜利、失败、平局）收集而来的，用于二分类问题的学习和训练，部分数据如下所示：

b,b,b,x,x,x,o,b,o,positive  
b,b,b,x,x,x,b,o,o,positive  
b,b,b,o,o,b,x,x,x,positive  
b,b,b,o,b,o,x,x,x,positive  
b,b,b,b,o,o,x,x,x,positive  
x,x,o,x,x,o,o,b,o,negative  
x,x,o,x,x,o,b,o,o,negative  
x,x,o,x,x,b,o,o,o,negative  
x,x,o,x,o,x,o,o,b,negative

数据集包括958个实例，每个实例有9个属性，代表井字棋棋盘上每个位置的状态，取值为"x"、"o"或者"b"，分别代表玩家1、玩家2和空白。目标变量为类别标签，分别为"positive"和"negative"，分别代表胜利和失败/平局。该数据集是一个较为干净的数据集，其存在以下特征和问题：

* 数据平衡：在Tic-Tac-Toe数据集中，根据类别1（赢）和类别0（输或平局）的比例，我们可以发现这是一个相对均衡的数据集，约65.3%为positive，因此不需要进行欠采样等数据平衡的处理。
* 不存在数据缺失值：经检查，Tic-Tac-Toe数据集不存在缺失值，这可以避免数据填充、缺失值删除或插值等操作。
* 不存在明显噪声：Tic-Tac-Toe数据集是一个经过人工处理和筛选的数据集，数据质量比较高，没有明显的噪声。也就避免了对数据集进行数据清洗、降噪滤波的操作。
* 数据集较小：Tic-Tac-Toe数据集只有958个实例，对于一些复杂的模型如神经网络或决策树的深度较高的机器学习算法而言可能会存在过拟合的问题。

### 二值化处理与划分数据集

尽管Tic-Tac-Toe数据集存在一些特点和问题，但我们仍然可以通过应用数据挖掘技术来挖掘其中的规律和信息，从而提高我们对游戏胜负的预测准确性，例如进行以下数据预处理：

* 二值化处理：对于Tic-Tac-Toe数据集，由于所有特征都是二元离散型特征，因此不需要进行特征缩放。特征缩放通常用于连续型特征，以避免某些特征对模型的影响过大。对于二元离散型特征，我们只需要对其进行二值化处理即可。在对Tic-Tac-Toe.data数据集进行处理时，可以将数据集中的非数值型特征进行二值化处理，将其转化为数值型特征，便于应用机器学习算法进行分类任务。具体的二值化处理方法可以采用如下步骤：
  + 将Tic-Tac-Toe.data数据集中的每个非数值型特征（X、O、B）分别编码为不同的整数，如'X' -> 1, 'O' -> 2, 'B' -> 3。
  + 对于每个特征，将其转化为多个二元特征。以X为例，可以将其转化为两个二元特征：一个表示格子中是否为X，另一个表示格子中是否不是X。这两个特征可以用0/1表示，其中0表示否，1表示是。
  + 重复上一步骤，对所有的非数值型特征进行二值化处理。

在实现上，可以使用Python的Pandas库来进行数据预处理和二值化处理：

1. **import** pandas as pd
3. # 读取Tic-Tac-Toe数据集
4. data = pd.read\_csv('Tic-Tac-Toe.data', header=None)
6. # 将X、O、B编码为整数
7. data.replace({'X': 1, 'O': 2, 'B': 3}, inplace=True)
9. # 对所有特征进行二值化处理
10. **for** col **in** data.columns[:-1]:
11. # 将当前特征转化为两个二元特征
12. data[col+'\_1'] = (data[col] == 1).astype(int)
13. data[col+'\_2'] = (data[col] == 2).astype(int)
14. data.drop(col, axis=1, inplace=True)
16. # 将类别标签编码为0/1
17. data.replace({'positive': 1, 'negative': 0}, inplace=True)

经过对Tic-Tac-Toe数据集的预处理，可以更好地提取数据中的信息和模式，以便分类算法能够更准确地进行分类。

由于Tic-Tac-Toe数据集较小，将其划分数据集为训练集和测试集就足够了，在这种情况下，使用训练集来训练模型并进行参数选择，然后使用测试集来评估模型的性能。以下将数据集按照8:2的比例进行数据集划分：

1. # 划分数据集
2. X = data.iloc[:, :-1].values
3. y = data.iloc[:, -1].values
4. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

## 算法原理与分类实验

### 最近邻算法

最近邻算法是一种基于实例的学习算法，其基本思想是通过寻找样本空间中与待分类样本距离最近的K个训练样本，将待分类样本划分到与之最邻近的类别中。最近邻算法没有显式地学习一个模型，而是通过搜索和计算距离来进行分类。它可以处理多分类和回归问题，常用于模式识别、图像识别、语音识别等领域。scikit-learn实现了两种不同的近邻分类器。

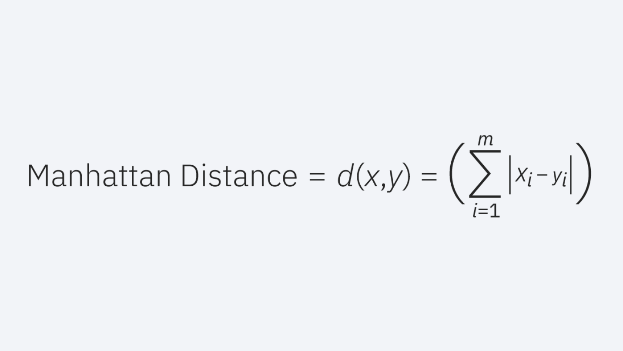
本文中采用KNeighborsClassifier()，其重要参数如下：

* weights ：

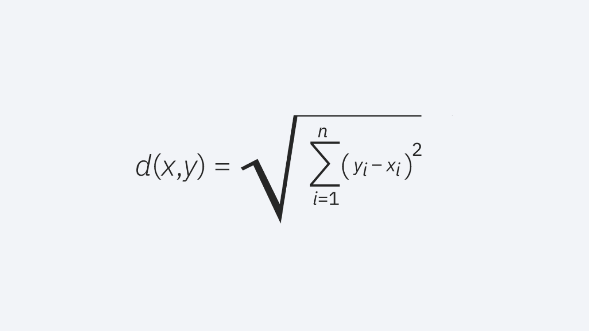
对于Tic-Tac-Toe数据集，我们可以使用Uniform weights或Distance weights来计算邻居的权重。但本文认为离查询点越近的邻居对结果的影响不会更大，所以本文使用Uniform weight，也即默认设置。

* p ：

距离度量公式。这个参数默认为2，也就是默认使用欧式距离公式进行距离度量。也可以设置为1，使用曼哈顿距离公式进行距离度量。经测试，p=1，即使用曼哈顿距离公式进行距离度量能有更高的精确度。究其原因如下：



* 曼哈顿距离适用于特征空间为网格状的情况，例如棋盘上的移动距离。在这种情况下，曼哈顿距离可以准确地测量两点之间的距离，因为在网格状空间中，只能沿着坐标轴的方向移动，不能斜向移动。



* 欧式距离适用于特征空间为连续的情况，例如物理空间中的距离。在这种情况下，欧式距离可以准确地测量两点之间的距离，因为在连续空间中，可以沿着任意方向移动。
* n\_neighbors :

默认情况下kneighbors查询使用的邻居数。就是k-NN的k的值，选取最近的k个点。对于Tic-Tac-Toe数据集，这是一个分类任务，因此需要考虑模型的准确性和泛化能力。本文使用sklearn中的网格搜索（Grid Search）进行KNN模型的参数调优。具体来说，定义了一个参数范围，这里是n\_neighbors从1到21，然后通过GridSearchCV函数来进行网格搜索，并以准确率（accuracy）作为评估指标：

1. # 定义参数范围
2. param\_grid = {"n\_neighbors": range(1, 21)}
4. # 进行网格搜索
5. grid\_search = GridSearchCV(knn, param\_grid, cv=5, scoring="accuracy")
6. grid\_search.fit(X\_train, y\_train)

最后，按照验证所确认的参数对K近邻分类器进行训练：

1. # 训练K近邻分类器
2. knn = KNeighborsClassifier(p=1, n\_neighbors=?)
3. knn.fit(X\_train, y\_train)

### ID3算法

ID3算法是一种用于构建决策树的经典算法。在ID3算法中，决策树的构建是通过选择最大信息增益的特征来进行的。信息增益是用来度量将某个特征用作划分标准时所能获得的信息增益的大小，从而选择信息增益最大的特征进行划分，从而生成一个根据输入特征对数据进行分类的模型。

ID3算法的具体步骤如下：

* 根据训练数据集中的样本特征和分类标签，计算每个特征对于分类的信息增益。
* 选择信息增益最大的特征作为当前节点的特征，将数据集按照该特征的不同取值进行划分。
* 对于每个子节点，如果样本集合已经完全属于同一类别，则将该节点标记为叶子节点，否则重复前两步，直到所有节点都为叶子节点或已满足预先设定的停止条件。将生成的决策树存储起来，并将其用于对新样本的分类。

本文采用DecisionTreeClassifier(criterion='entropy')，其重要参数如下：

* criterion ：

特征选择标准，可选参数，默认是gini，可以设置为entropy。在ID3算法中，决策树的构建是通过选择最大信息增益的特征来进行的。信息增益是用来度量将某个特征用作划分标准时所能获得的信息增益的大小，从而选择信息增益最大的特征进行划分。而信息增益是由信息熵计算得出的，信息熵用来度量样本集合的纯度和不确定性，即样本集合中不同类别样本的混杂程度。当某个特征的信息熵越小，说明该特征可以将样本集合划分得越纯，所能获得的信息增益也就越大。因此，ID3算法将criterion参数设置为"entropy"，是为了使用信息熵来评估特征的划分能力，并根据信息增益来进行特征选择和决策树的构建。

* max\_depth ：

决策树最大深，这个参数是这是树的层数的。一般来说，数据少或者特征少的时候可以不管这个值。或者如果设置了min\_samples\_slipt参数，那么直到少于min\_smaples\_split个样本为止。如果模型样本量多，特征也多的情况下，就需要限制这个最大深度。同样的，本文使用网格搜索来探索最合适的max\_depth：

1. # 定义参数范围
2. param\_grid = {"max\_depth": range(1, 21)}
4. # 进行网格搜索
5. results = []
6. **for** max\_depth in param\_grid["max\_depth"]
8. # 训练模型
9. clf = DecisionTreeClassifier(criterion='entropy', max\_depth=max\_depth)

最终本文对ID3的参数设置如下：

1. clf = DecisionTreeClassifier(criterion='entropy', max\_depth=?)

### 随机森林

随机森林是一种集成学习算法，它由多个决策树组成。在建立随机森林时，它会随机选择样本和特征，然后构建多个决策树，最后通过投票方式进行分类或者回归。

本文选用RandomForestClassifier()，其重要参数为：

* random\_state ：

控制生成随机森林的模式。并不能控制森林中的树的样式。随机性越大，模型效果越好，当然这样可能就不是很稳定，不便于调试。为了保证模型稳定，设置random\_state为42。

* n\_estimators :

这是森林中树木的数量，即基基评估器的数量。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。同样的，我们着重对该参数进行调参：

1. # 定义参数范围
2. param\_grid = {"n\_estimators": range(1, 51)}
4. # 进行网格搜索
5. grid\_search = GridSearchCV(rf, param\_grid, cv=5, scoring="accuracy")
6. grid\_search.fit(X\_train, y\_train)

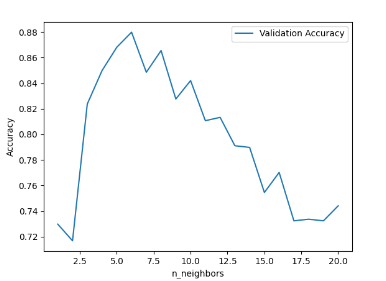
最终本文对随机森林算法的参数设置如下：

1. rf = RandomForestClassifier(random\_state=42, n\_estimators=?)

## 实验结果与对比分析

### 最近邻算法

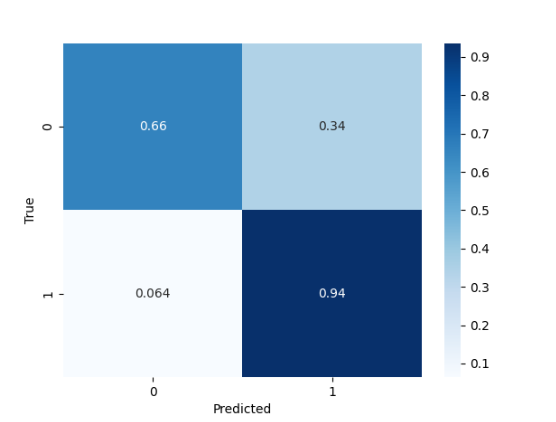
不同的n\_neighbors参数可以获得不同的结果，经过第二章中的调参，可以得到下图的结果：



由图可知，当n\_neighbors取值较小时，模型的复杂度较高，容易受到噪声的影响，但是可以更好地拟合局部特征；而当n\_neighbors取值较大时，模型的复杂度较低，可以更好地拟合全局特征，但是容易出现过拟合的问题。通常而言，将n\_neighbors取值设为奇数，这样可以避免出现平局的情况，同时也有利于提高模型的准确性，但是根据上图所示真实结果，最终本文决定选取n\_neighbors=6。

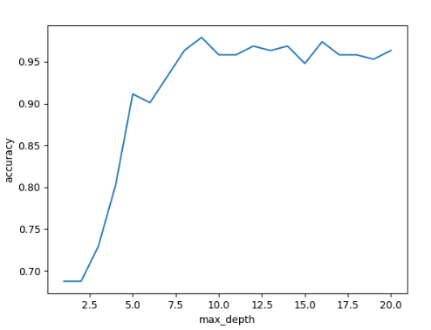
1. # 输出最优参数
2. print("Best Parameters:", grid\_search.best\_params\_)
3. print("Best Score:", grid\_search.best\_score\_)
5. # 控制台输出结果
6. Best Parameters: {'n\_neighbors': 6}
7. Best Score: 0.8798828622358034
8. Test Accuracy: 0.8385416666666666

将最优n\_neighbors=6代入，可得最近邻算法对Tic-Tac-Toe数据分类的结果：



### ID3算法

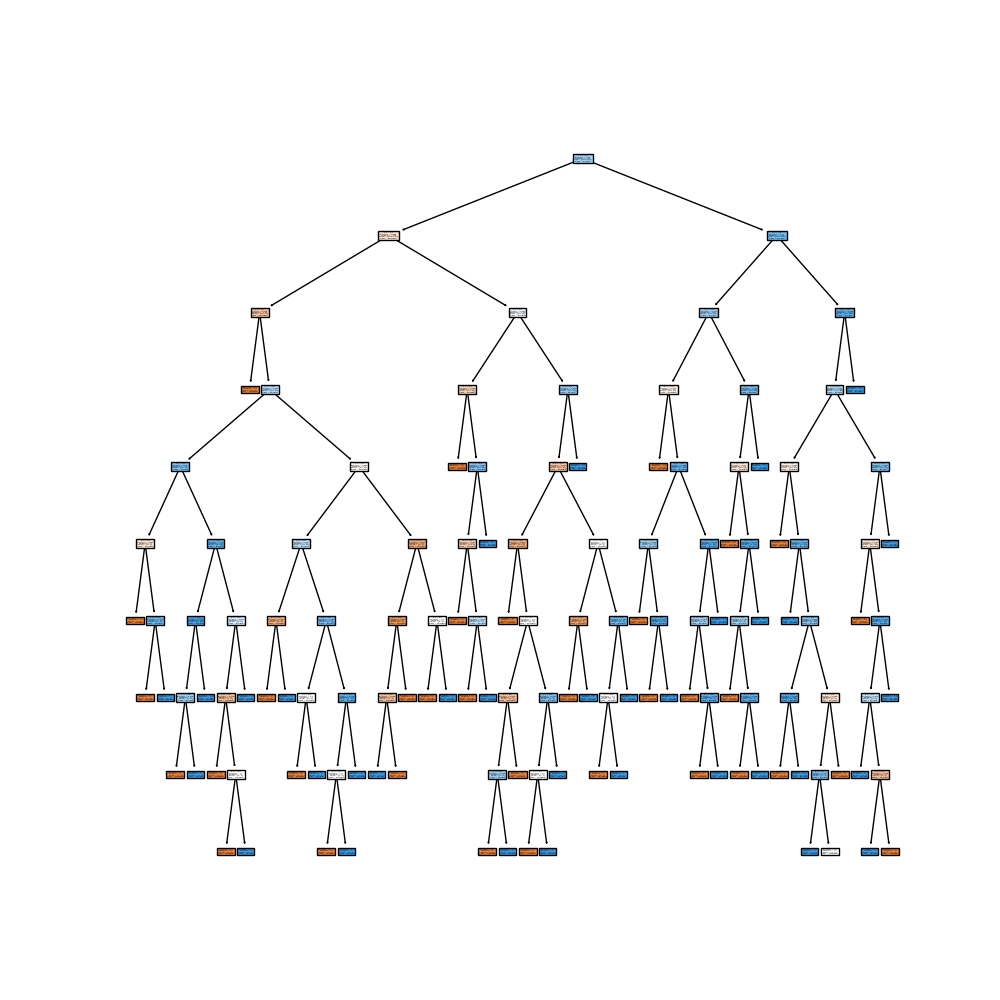
不同的max\_depth参数可以获得不同的结果，经过第二章中的调参，可以得到下图的结果：



由图可知，max\_depth的取值越大，模型的复杂度越高，训练时的准确率会更高，但是可能会出现过拟合的问题。相反，如果max\_depth取值过小，模型的准确率可能会下降，欠拟合的问题也可能出现。max\_depth=9时，ID3算法有最优精确度：

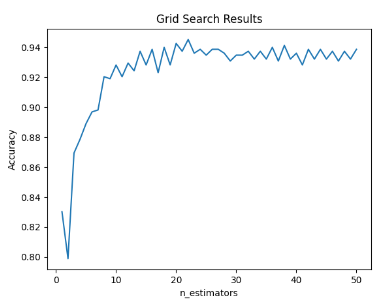
1. # 打印结果
2. print("max\_depth  accuracy")
3. **for** result in results:
4. print("{:<10d}{:<10.4f}".format(result[0], result[1]))
6. # 控制台输出结果
7. max\_depth  accuracy
8. 1         0.6875
9. 2         0.6875
10. 3         0.7292
11. 4         0.8021
12. 5         0.9115
13. 6         0.9010
14. 7         0.9323
15. 8         0.9635
16. 9         0.9792
17. 10        0.9583
18. 11        0.9583
19. 12        0.9688
20. 13        0.9635
21. 14        0.9688
22. 15        0.9479
23. 16        0.9740
24. 17        0.9583
25. 18        0.9583
26. 19        0.9531
27. 20        0.9635

将最优max\_depth=9代入，可得ID3算法对Tic-Tac-Toe数据分类的决策树：



### 随机森林

不同的n\_estimators参数可以获得不同的结果，经过第二章中的调参，可以得到下图的结果：

****

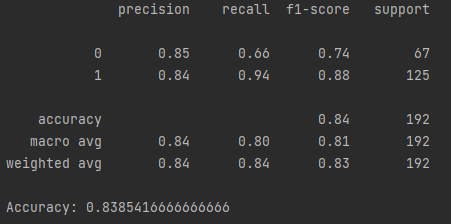
增加 n\_estimators 的值可以提高模型的准确性，但也会增加模型的计算时间和内存消耗。通常来说，增加 n\_estimators 直到达到稳定的准确性水平即可。n\_estimators =22时，随机森林算法有最优精确度：

1. # 输出最优参数
2. print("Best Parameters:", grid\_search.best\_params\_)
3. print("Best Score:", grid\_search.best\_score\_)
5. # 控制台输出结果
6. Best Parameters: {'n\_estimators': 22}
7. Best Score: 0.9452168746286393
8. Test Accuracy: 0.9270833333333334

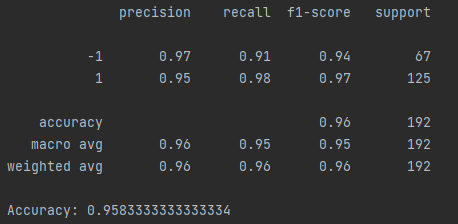
由于Tic-Tac-Toe数据集中特征维度较高，难以进行可视化展示

### 评估不同分类算法

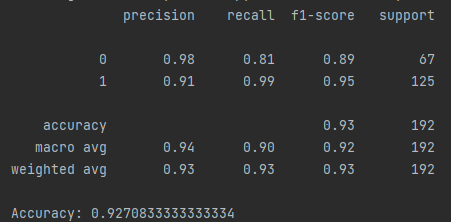
最近邻算法结果：



ID3算法结果：



随机森林算法结果：



从准确率上看，ID3算法的准确率最高，随机森林算法其次，K近邻法的准确率最低。这是因为ID3算法是一种基于信息熵的决策树算法，能够有效地对数据集进行分类。而随机森林算法是通过组合多个决策树来提高分类准确率，因此也能得到较好的分类结果。而K近邻法则是一种基于距离度量的算法，其分类结果较为依赖于训练数据的分布情况和K值的选取，因此相对于ID3算法和随机森林算法来说，K近邻法的分类准确率要低一些。

## 总结

本文主要围绕机器学习中的三种算法：ID3、随机森林和K近邻法进行了讨论。针对Tic-Tac-Toe数据集，本文详细介绍了这三种算法的执行步骤、调参方法、算法特性等，并通过对数据集进行分类得到了结果。总体而言，ID3算法在处理小规模数据集时表现良好，但在面对大规模数据时容易过拟合；随机森林算法在处理大规模数据集时表现优秀，并且能够有效避免过拟合；而K近邻法则适用于处理中等规模数据集，并且对于非线性的分类问题表现优异。在调参方面，本文介绍了各个算法的重要参数及其调参方法，并通过网格搜索的方法找到了最优参数。最后，我们根据算法在Tic-Tac-Toe数据集上的准确率对这三种算法进行了比较，其中ID3算法表现最好，随机森林次之，K近邻法表现最差。

## 附录

* 最近邻算法

1. import pandas as pd
2. from sklearn.model\_selection import train\_test\_split, GridSearchCV
3. from sklearn.neighbors import KNeighborsClassifier
4. import matplotlib.pyplot as plt
5. from sklearn.preprocessing import LabelEncoder
6. from sklearn.metrics import classification\_report
7. from sklearn.metrics import accuracy\_score
8. import seaborn as sns

11. # 加载数据集
12. data = pd.read\_csv("tic-tac-toe.data", header=None)
14. # 将类别变量转换为数值型变量
15. le = LabelEncoder()
16. data = data.apply(le.fit\_transform)
18. # 划分训练集和测试集
19. X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.iloc[:, :-1], data.iloc[:, -1], test\_size=0.2, random\_state=42)
21. # 定义参数范围
22. param\_grid = {"n\_neighbors": range(1, 21)}
24. # 实例化模型和GridSearchCV对象
25. knn = KNeighborsClassifier(p=1)
27. # 进行网格搜索
28. grid\_search = GridSearchCV(knn, param\_grid, cv=5, scoring="accuracy")
29. grid\_search.fit(X\_train, y\_train)
31. # 输出最优参数
32. print("Best Parameters:", grid\_search.best\_params\_)
33. print("Best Score:", grid\_search.best\_score\_)
35. # 可视化参数调优结果
36. plt.plot(param\_grid["n\_neighbors"], grid\_search.cv\_results\_["mean\_test\_score"], label="Validation Accuracy")
37. plt.xlabel("n\_neighbors")
38. plt.ylabel("Accuracy")
39. plt.legend()
40. plt.show()
42. # 输出模型在测试集上的准确率
43. print("Test Accuracy:", grid\_search.score(X\_test, y\_test))
45. # 以下为获取到最优参数后的计算
47. # 训练K近邻分类器
48. # knn = KNeighborsClassifier(p=1, n\_neighbors=6)
49. # knn.fit(X\_train, y\_train)
51. # 预测测试集结果
52. # y\_pred = knn.predict(X\_test)
54. # 输出分类报告
55. # print(classification\_report(y\_test, y\_pred))
57. # 计算准确率
58. # accuracy = accuracy\_score(y\_test, y\_pred)
59. # print('Accuracy:', accuracy)
61. # 绘制混淆矩阵热力图
62. # confusion\_matrix = pd.crosstab(y\_test, y\_pred, rownames=['True'], colnames=['Predicted'], normalize='index')
63. # sns.heatmap(confusion\_matrix, annot=True, cmap='Blues')
64. # plt.show()

* ID3算法

1. import pandas as pd
2. from sklearn.model\_selection import train\_test\_split
3. from sklearn.tree import DecisionTreeClassifier
4. from sklearn.metrics import accuracy\_score
5. import matplotlib.pyplot as plt
6. from sklearn.metrics import classification\_report
7. from sklearn.tree import plot\_tree
9. # 读取数据集
10. data = pd.read\_csv('tic-tac-toe.data', header=None)
12. # 将标签编码为数字
13. data = data.replace({'x': 1, 'o': -1, 'b': 0, 'positive': 1, 'negative': -1})
15. # 划分特征和标签
16. X = data.iloc[:, :-1]
17. y = data.iloc[:, -1]
19. # 划分训练集和测试集
20. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
22. # 定义参数范围
23. param\_grid = {"max\_depth": range(1, 21)}
25. # 进行网格搜索
26. results = []
27. **for** max\_depth in param\_grid["max\_depth"]:
28. # 训练模型
29. clf = DecisionTreeClassifier(criterion='entropy', max\_depth=max\_depth)
30. clf.fit(X\_train, y\_train)
31. # 预测结果
32. y\_pred = clf.predict(X\_test)
33. # 计算准确率
34. acc = accuracy\_score(y\_test, y\_pred)
35. results.append((max\_depth, acc))
37. # 输出结果
38. print("max\_depth  accuracy")
39. **for** result in results:
40. print("{:<10d}{:<10.4f}".format(result[0], result[1]))
42. # 绘制可视化结果
43. plt.plot(param\_grid["max\_depth"], [x[1] **for** x in results])
44. plt.xlabel("max\_depth")
45. plt.ylabel("accuracy")
46. plt.show()
48. # 以下为获取到最优参数后的计算
50. # # 训练决策树分类器
51. # dtc = DecisionTreeClassifier(criterion='entropy', max\_depth=9)
52. # dtc.fit(X\_train, y\_train)
53. #
54. # # 预测测试集结果
55. # y\_pred = dtc.predict(X\_test)
56. #
57. # # 输出分类报告
58. # print(classification\_report(y\_test, y\_pred))
59. #
60. # # 计算准确率
61. # accuracy = accuracy\_score(y\_test, y\_pred)
62. # print('Accuracy:', accuracy)
63. #
64. # # 绘制决策树
65. # plt.figure(figsize=(10, 10))
66. # plot\_tree(dtc, filled=True, feature\_names=data.columns[:-1], class\_names=['negative', 'positive'])
67. # plt.show()

* 随机森林

1. import pandas as pd
2. from sklearn.ensemble import RandomForestClassifier
3. from sklearn.model\_selection import train\_test\_split, GridSearchCV
4. import matplotlib.pyplot as plt
5. from sklearn.preprocessing import LabelEncoder
6. from sklearn.metrics import classification\_report
7. from sklearn.metrics import accuracy\_score
9. # 读取数据集
10. data = pd.read\_csv('tic-tac-toe.data', header=None)
12. # 将类别变量转换为数值型变量
13. le = LabelEncoder()
14. data = data.apply(le.fit\_transform)
16. X = data.iloc[:, :-1]
17. y = data.iloc[:, -1]
19. # 划分训练集和测试集
20. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
22. # 定义模型
23. rf = RandomForestClassifier(random\_state=42)
25. # 定义参数范围
26. param\_grid = {"n\_estimators": range(1, 51)}
28. # 进行网格搜索
29. grid\_search = GridSearchCV(rf, param\_grid, cv=5, scoring="accuracy")
30. grid\_search.fit(X\_train, y\_train)
32. # 输出最优参数
33. print("Best Parameters:", grid\_search.best\_params\_)
34. print("Best Score:", grid\_search.best\_score\_)
35. print("Test Accuracy:", grid\_search.score(X\_test, y\_test))
37. # 可视化结果
38. results = pd.DataFrame(grid\_search.cv\_results\_)
39. plt.plot(results['param\_n\_estimators'], results['mean\_test\_score'])
40. plt.title("Grid Search Results")
41. plt.xlabel("n\_estimators")
42. plt.ylabel("Accuracy")
43. plt.show()
45. # 以下为获取到最优参数后的计算
47. # # 训练随机森林分类器
48. # rfc = RandomForestClassifier(random\_state=42, n\_estimators=22)
49. # rfc.fit(X\_train, y\_train)
50. #
51. # # 预测测试集结果
52. # y\_pred = rfc.predict(X\_test)
53. #
54. # # 输出分类报告
55. # # 由于Tic-Tac-Toe数据集中特征维度较高，难以进行可视化展示
56. # print(classification\_report(y\_test, y\_pred))
57. # # 计算准确率
58. # accuracy = accuracy\_score(y\_test, y\_pred)
59. # print('Accuracy:', accuracy)