基于 KNN/KD-Tree 对鸢尾花分类的比较和参数调优

李国夏，16051216

1　问题提出

1.1　鸢尾花数据集的介绍

Iris 也称鸢尾花卉数据集，是一类多重变量分析的数据集。通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

数据集来源：http://archive.ics.uci.edu/ml/datasets/Iris

```
from sklearn.datasets import load_iris
iris = load_iris() #载入数据集
iris_data = iris.data #特征矩阵
iris_target = iris.target #标签向量
```

鸢尾花数据集只有 150 个样本，每个样本只有 4 个特征，容易将其可视化

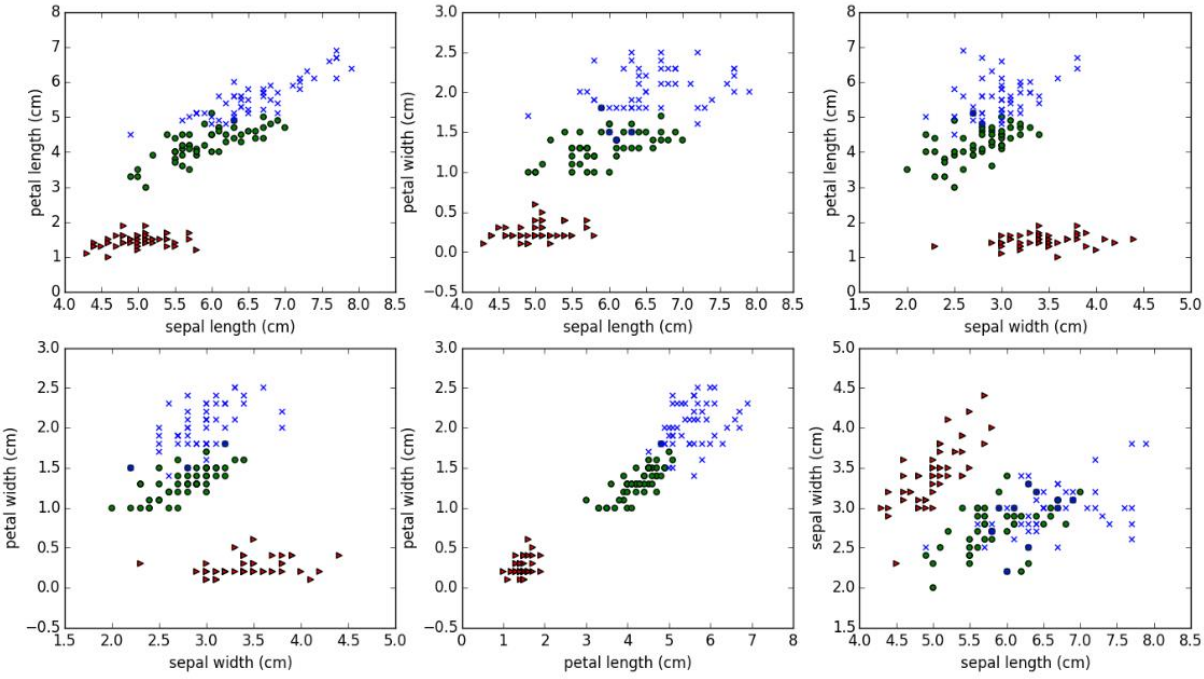

图 1　两两特征之间的可视化

1.2 问题的提出

- 我们利用 sklearn 中自带的数据集(鸢尾花数据集),并通过 KNN 算法实现对鸢尾花的分类。
- 由于手工调试参数比较麻烦。我们使用交叉验证和网格搜索来确定最好的参数

2　KNN 算法简介及 Python 实现

2.1 KNN 算法简介

- 工作原理

存在一个样本数据集合，并且样本集中每个数据都存在标签。输入没有标签的新数据后，将新数据的每个特征和样本集中特征进行比较，然后算法提取样本集中特征最相似的数据（最近邻）的分类标签。

一般来说，只选择样本数据集中前 N 个最相似的数据。K 一般不大于 20，最后选择 k 个中出现次数最多的分类（多数投票原则），作为新数据的分类。

- KNN 算法的一般流程

收集数据：使用任何方法
准备数据：距离计算所需要的数值
分析数据：可以使用任何方法
测试算法：计算错误率
使用算法：首先需要输入样本数据和结构化输出结果，运行 KNN 算法判定输入数据属于哪一类，最后对计算出的分类进行后续处理

- 距离度量：欧式距离
- 算法实现伪代码

对未知类别属性的数据集中每个点依次执行以下操作：

1. 计算已知类标数据集中的点和当前点之间的距离
2. 按照距离递增次序排序
3. 选取与当前点距离最小的 k 个点
4. 确定前 k 个点所在类别的出现频率
5. 返回前 k 个点出现频率最高类别作为当前点的预测分类

2.2 KNN 算法的 Python 实现

```
from numpy import *
import operator
def KNN_Classify(X, data, labels, k):
    dataSize = data.shape[0]
    diffMat = tile(X, (data, 1))-data
    dist = (diffMat**2).sum(axis=1)**0.5 #距离计算
    sortedDistIndex = dist.argsort()
    classCount = {}
    for i in range(k):                    #选取距离最小的 k 个点
        voteLabel = labels[sortedDistIndex[i]]
```

```
            classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
        sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True) #排序
        return sortedClassCount[0][0]
```

3 准备数据：获取鸢尾花数据

```
from sklearn.datasets import load_iris

def get_iris_data(self):
        iris = load_iris()
        iris_data = iris.data # 鸢尾花特征值
        iris_target = iris.target # 鸢尾花目标值

        return iris_data, iris_target
```

4 数据处理：测试/训练集划分

```
from sklearn.model_selection import train_test_split

# x_train 训练集特征值 x_test 测试集特征值 y_train 训练集目标值 y_test 测试集目标值 test_size=0.25 表示 25%的数据用于测试
x_train, x_test, y_train, y_test = train_test_split(iris_data, iris_target, test_size=0.25)
```

5 数据处理：数据标准化

- 由于每个特征的大小，取值范围等不一样，这样会导致每个特征的权重不一样，而实际上是一样的。通过对原始数据进行变换把数据变换到均值为 0,方差为 1 范围内，以便于计算机处理。

```
from sklearn.preprocessing import StandardScaler

std = StandardScaler()
x_train = std.fit_transform(x_train)
x_test = std.transform(x_test)
```

6 创建分类器：KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=5) # 创建一个 KNN 分类器
knn.fit(x_train, y_train) # 将测试集送入算法
y_predict = knn.predict(x_test) # 获取预测结果
```

7 创建分类器：Kd-Tree

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, algorithm='kd-tree') # 创建一个 Kd-Tree 分类器
knn.fit(x_train, y_train) # 将测试集送入算法
y_predict = knn.predict(x_test) # 获取预测结果
```

8 分类结果展示

KNN	Kd-Tree
第 1 次测试:真实值:山鸢尾 预测值:山鸢尾	第 1 次测试:真实值:山鸢尾 预测值:山鸢尾
第 2 次测试:真实值:山鸢尾 预测值:山鸢尾	第 2 次测试:真实值:山鸢尾 预测值:山鸢尾
第 3 次测试:真实值:变色鸢尾 预测值:变色鸢尾	第 3 次测试:真实值:虹膜锦葵 预测值:虹膜锦葵
第 4 次测试:真实值:变色鸢尾 预测值:变色鸢尾	第 4 次测试:真实值:虹膜锦葵 预测值:虹膜锦葵
第 5 次测试:真实值:虹膜锦葵 预测值:虹膜锦葵	第 5 次测试:真实值:山鸢尾 预测值:山鸢尾
第 6 次测试:真实值:虹膜锦葵 预测值:虹膜锦葵	第 6 次测试:真实值:山鸢尾 预测值:山鸢尾
…	…

9 算法性能比较及评价

分类器	近邻数 k	准确率	时限
KNN	5	0.973684210526315	21.9 ms
Kd-Tree	8	0.973684210526315	17 ms

- 实现 k 近邻法时，最简单实现是线性扫描，这时要计算输入实例与每一个训练实例的距离，当训练集很大时，计算非常耗时
- 为了提高 k 近邻搜索的效率，可以考虑使用特殊的结构存储训练数据，这里的 Kd-Tree 就是一种对 k 维空间中的实例点进行存储以便对其进行快速搜索的树形数据结构

10 利用交叉验证和网格搜索进行参数调优

上述算法实现中的参数 n_neighbors 是我们手动设置的，实际上我们可以使用交叉验证和网格搜索进行自动参数调优。

- 交叉验证：将数据分为 n 份，将其中一份作为验证集，经过 n 组测试，每组测试时都要更换验证集。这样得到 n 组模型的结果，取平均值作为最终的结果。

- 网格搜索：通常情况下，有很多参数是需要手动指定的，这种叫超参数。但是手动过程繁杂，所以需要对模型预设几种超参数组合。每组超参数都采用交叉验证来进行评估。最后选出最优参数组合建立模型。

```
    def evaluate(self):
        iris_data, iris_target = self.get_iris_data()
        # 训练集/测试集划分
        x_train, x_test, y_train, y_test = train_test_split(iris_data, iris_target, test_size=0.25)
        # 生成 knn 估计器
        kd_tree = KNeighborsClassifier(algorithm='kd_tree')
        # 构造超参数值
        params = {"n_neighbors":range(1,11)}
        # 进行网格搜索
        gridCv = GridSearchCV(kd_tree, param_grid=params, cv=5)
        gridCv.fit(x_train,y_train) # 输入训练数据
        # 预测准确率
        print("准确率：",gridCv.score(x_test, y_test))
        print("交叉验证中最好的结果：",gridCv.best_score_)
        print("最好的模型：", gridCv.best_estimator_)
        return gridCv.best_estimator_
```

运行结果：

```
准确率： 0.9736842105263158
```

```
交叉验证中最好的结果： 0.9642857142857143
最好的模型： KNeighborsClassifier(algorithm='kd_tree', leaf_size=30, metric='minkowski',
                metric_params=None, n_jobs=1, n_neighbors=10, p=2,
                weights='uniform')
```

11 附录：完整代码

```
# -*- coding: utf-8 -*-
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

class KdTree(object):
    # 获取鸢尾花数据
    def get_iris_data(self):
        iris = load_iris()
        iris_data = iris.data
        iris_target = iris.target
        return iris_data, iris_target

    def run(self):
        # 数据准备
        iris_data, iris_target = self.get_iris_data()
        # 训练集/测试集划分
        x_train, x_test, y_train, y_test = train_test_split(iris_data, iris_target, test_size=0.25)
        # 数据标准化
        std = StandardScaler()
        x_train = std.fit_transform(x_train)
        x_test = std.transform(x_test)
        # 创建分类器
        kd_tree = self.evaluate()
        kd_tree.fit(x_train, y_train) # 将测试集送入算法
        y_predict = kd_tree.predict(x_test) # 获取预测结果
        # 预测结果展示
```

```
        labels = ["山鸢尾","虹膜锦葵","变色鸢尾"]
        for i in range(len(y_predict)):
            print("第%d 次测试:真实值:%s\t 预测值:%s"%((i+1),labels[y_predict[i]],labels[y_test[i]]))
        print("准确率：",kd_tree.score(x_test, y_test))
    def evaluate(self):
        iris_data, iris_target = self.get_iris_data()
        # 训练集/测试集划分
        x_train, x_test, y_train, y_test = train_test_split(iris_data, iris_target, test_size=0.25)
        # 生成 knn 估计器
        kd_tree = KNeighborsClassifier(algorithm='kd_tree')
        # 构造超参数值
        params = {"n_neighbors":range(1,11)}
        # 进行网格搜索
        gridCv = GridSearchCV(kd_tree, param_grid=params, cv=5)
        gridCv.fit(x_train,y_train) # 输入训练数据
        # 预测准确率
        print("准确率：",gridCv.score(x_test, y_test))
        print("交叉验证中最好的结果：",gridCv.best_score_)
        print("最好的模型：", gridCv.best_estimator_)
        return gridCv.best_estimator_

class KNN(object):

    # 获取鸢尾花数据
    def get_iris_data(self):
        iris = load_iris()
        iris_data = iris.data
        iris_target = iris.target

        return iris_data, iris_target

    def run(self):
        # 数据准备
        iris_data, iris_target = self.get_iris_data()
        # 训练集/测试集划分
```

```
        x_train, x_test, y_train, y_test = train_test_split(iris_data, iris_target, test_size=0.25)
        # 数据标准化
        std = StandardScaler()
        x_train = std.fit_transform(x_train)
        x_test = std.transform(x_test)
        # 创建分类器
        knn = KNeighborsClassifier(n_neighbors=5)
        knn.fit(x_train, y_train) # 将测试集送入算法
        y_predict = knn.predict(x_test) # 获取预测结果
        # 预测结果展示
        labels = ["山鸢尾","虹膜锦葵","变色鸢尾"]
        for i in range(len(y_predict)):
            print("第%d 次测试:真实值:%s\t 预测值:%s"%((i+1),labels[y_predict[i]],labels[y_test[i]]))
        print("准确率：",knn.score(x_test, y_test))
    def evaluate(self):
        iris_data, iris_target = self.get_iris_data()
        # 训练集/测试集划分
        x_train, x_test, y_train, y_test = train_test_split(iris_data, iris_target, test_size=0.25)
        # 生成 knn 估计器
        knn = KNeighborsClassifier()
        # 构造超参数值
        params = {"n_neighbors":range(1,11)}
        # 进行网格搜索
        gridCv = GridSearchCV(knn, param_grid=params, cv=5)
        gridCv.fit(x_train,y_train) # 输入训练数据
        # 预测准确率
        print("准确率：",gridCv.score(x_test, y_test))
        print("交叉验证中最好的结果：",gridCv.best_score_)
        print("最好的模型：", gridCv.best_estimator_)

if __name__ == '__main__':
    kd_tree = KdTree()
    kd_tree.run()
```