



数据分析算法

北京理工大学计算机学院 孙新

2019年1月

▣ 实验目的

- 熟悉Weka平台，学习掌握KNN算法，利用Weka和不同参数设置进行分类分析，对比结果，得出结论，对问题进行总结实验过程

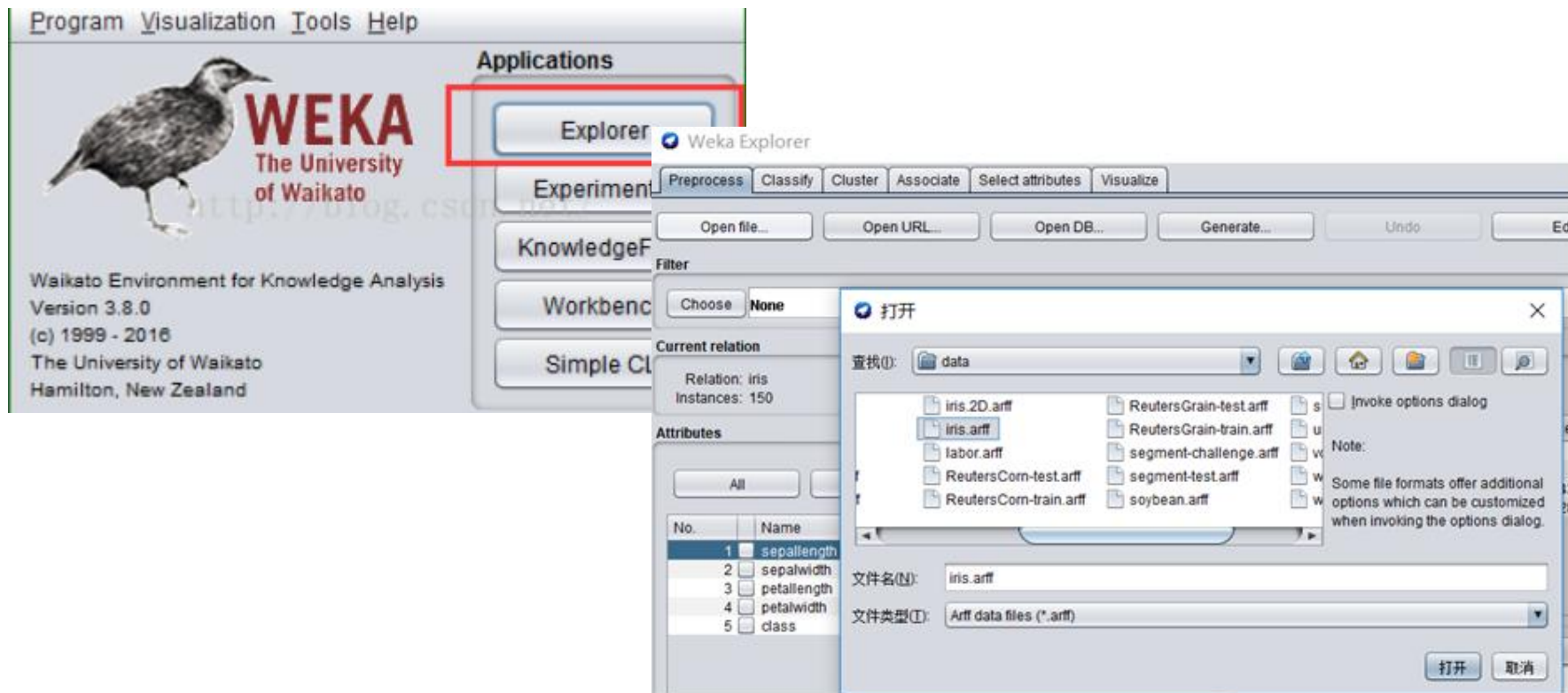
▣ 实验内容

- (1) 打开Weka，并导入数据
- (2) KNN算法分类操作步骤
- (3) 运行观察结果

使用Weka实现KNN分类算法的示例

4.3 KNN算法

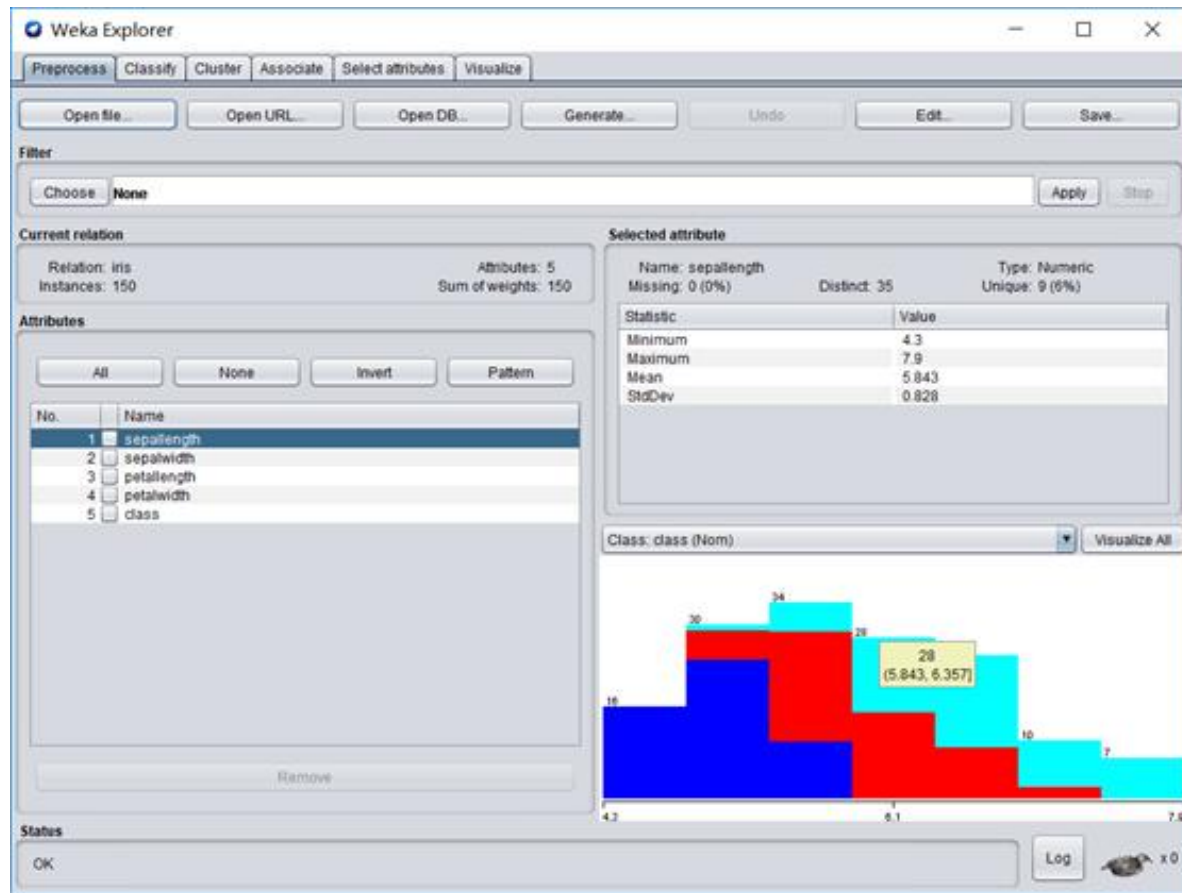
- 打开Weka自带的数据集的“iris.arff”。



使用Weka实现KNN分类算法的示例

4.3 KNN算法

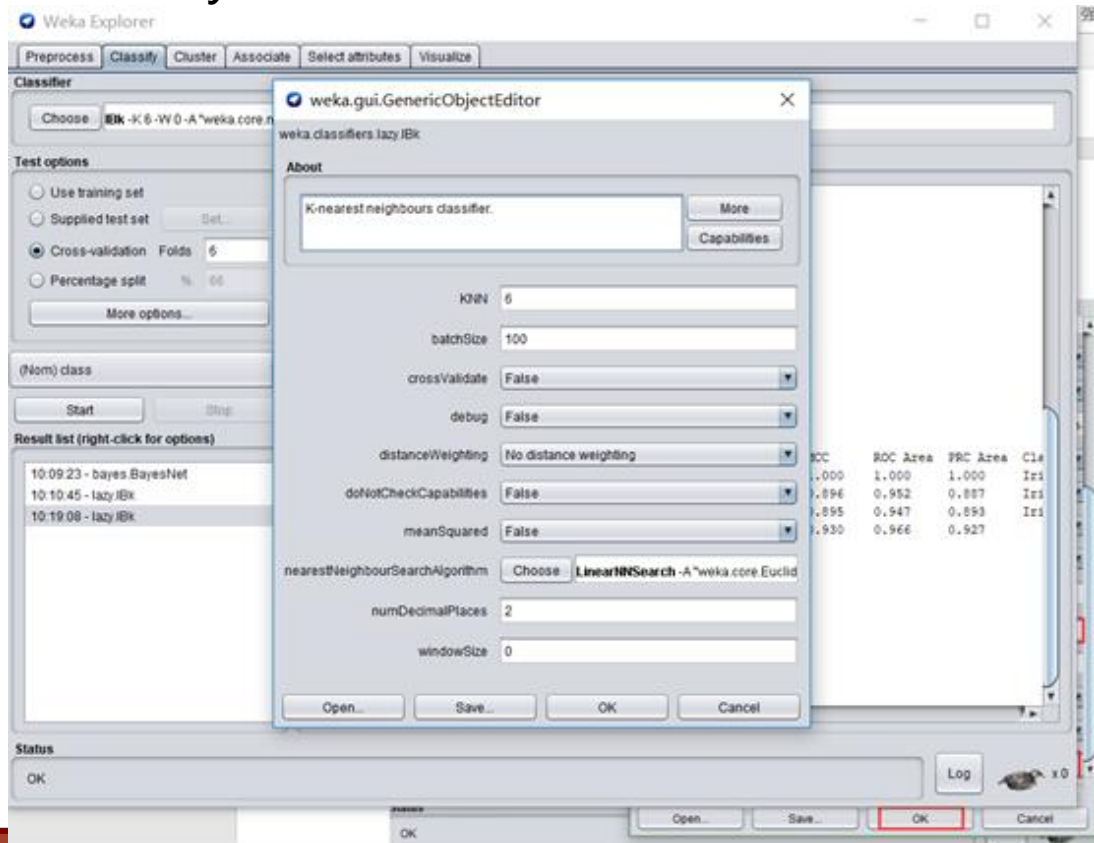
- 打开数据集后，界面出现该数据集的相关描述



使用Weka实现KNN分类算法的示例

4.3 KNN算法

- 点击“Choose”按钮，切换到“Classify”，选择目录下的“IBK”，这是WEKA中实现K最近邻的算法。
 - 点击“Choose”旁的文本框，修改参数“KNN”为6
 - 选中“Test Options”的“Cross-Validation”，选择“10”，



- 点击 “Start” 按钮之后，观察右边窗口 “Classifier output” 给出的分类结果。
- 也可以在左下角 “Result list” 中这次产生的结果上点右键，选择 “View in separate window”，然后在新窗口中浏览结果

```
Time taken to build model: 0 seconds

Stratified cross-validation ===
Summary ===

Correctly Classified Instances      145           96.6667 %
Incorrectly Classified Instances     5           3.3333 %
Kappa statistic                    0.95
Mean absolute error                 0.0391
Mean squared error                  0.137
Root mean squared error             8.789 %
Relative squared error              29.0555 %
Total Number of Instances          150

Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
-----
      1.000    0.000    1.000    1.000    1.000    1.000    1.000    1.000    Iris-setosa
      0.980    0.040    0.925    0.980    0.951    0.927    0.994    0.985    Iris-versicolor
      0.920    0.010    0.979    0.920    0.948    0.925    0.994    0.985    Iris-virginica
Weighted Avg.    0.967    0.017    0.968    0.967    0.967    0.951    0.996    0.990

Confusion Matrix ===

  b  c  <-- classified as
  0  0  |  a = Iris-setosa
 49  1  |  b = Iris-versicolor
  4 46  |  c = Iris-virginica
```

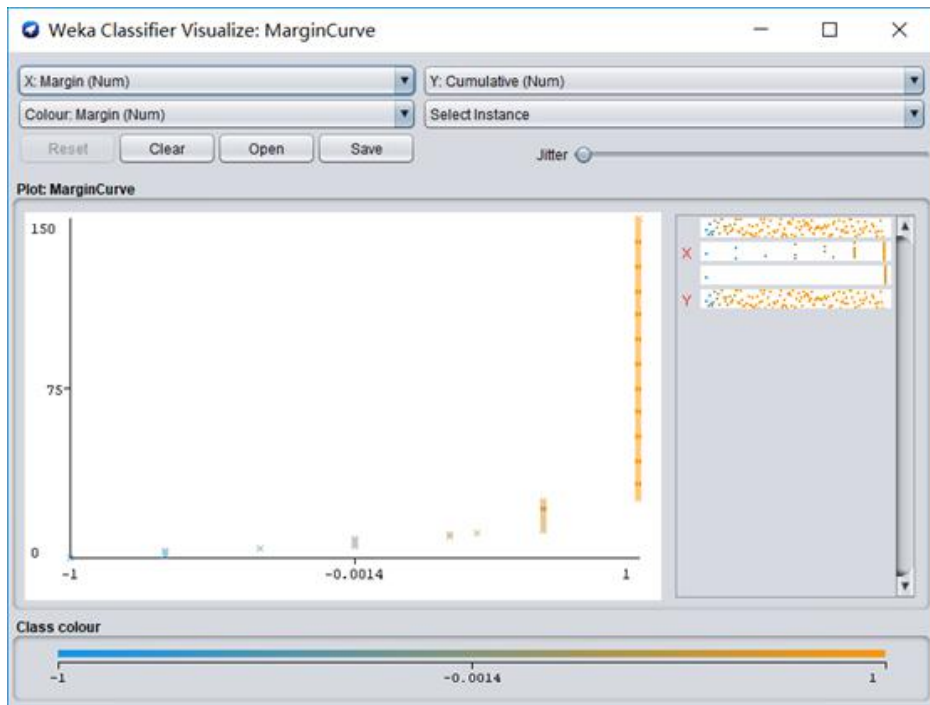
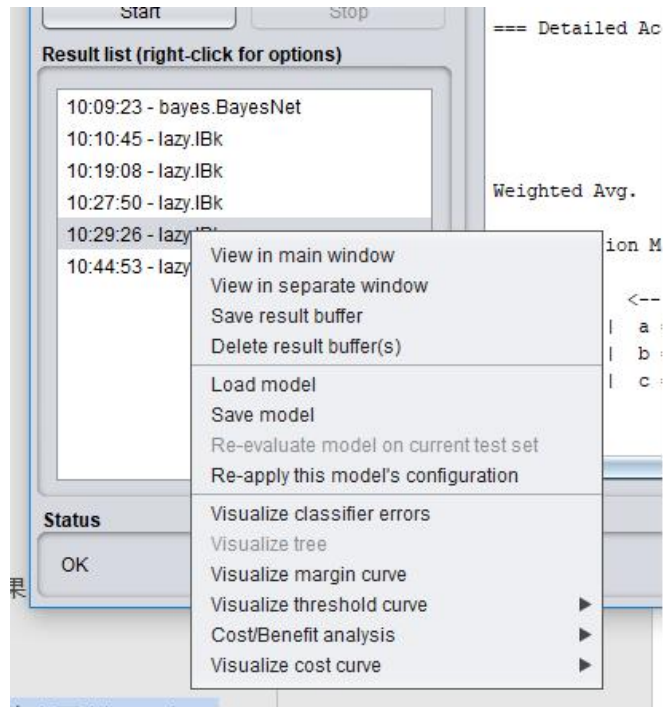
- 修改参数值观察运行结果
- 将“KNN”值修改为“3”，再次运行。
- 可以发现：与上面的运行过程正确率变小，
- 说明在一定范围内，k值越大正确率会越高

Correctly Classified Instances	145	96.6667 %
Incorrectly Classified Instances	5	3.3333 %
Kappa statistic	0.95	
Mean absolute error	0.0391	
Root mean squared error	0.137	
Relative absolute error	8.789 %	
Root relative squared error	29.0555 %	
Total Number of Instances	150	

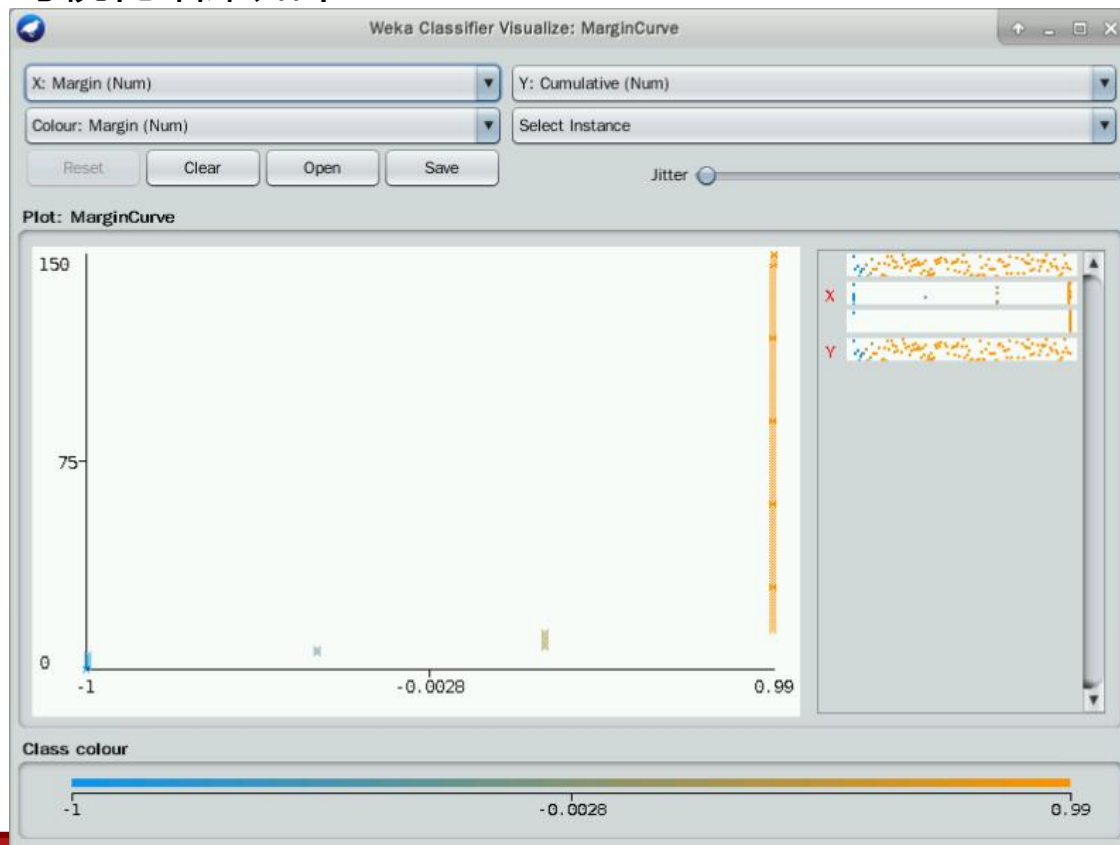
Correctly Classified Instances	143	95.3333 %
Incorrectly Classified Instances	7	4.6667 %
Kappa statistic	0.93	
Mean absolute error	0.04	
Root mean squared error	0.1703	
Relative absolute error	9.0013 %	
Root relative squared error	36.1192 %	
Total Number of Instances	150	

0 4 46 | c = Iris-virginica

- 为了观察可视化的分类结果，我们在左下方“Result list”列出的结果上右击，点“Visualize margin curve”。
- 当K=6时，可视化结果如下



- 当 $K=3$ 时，可视化结果如下

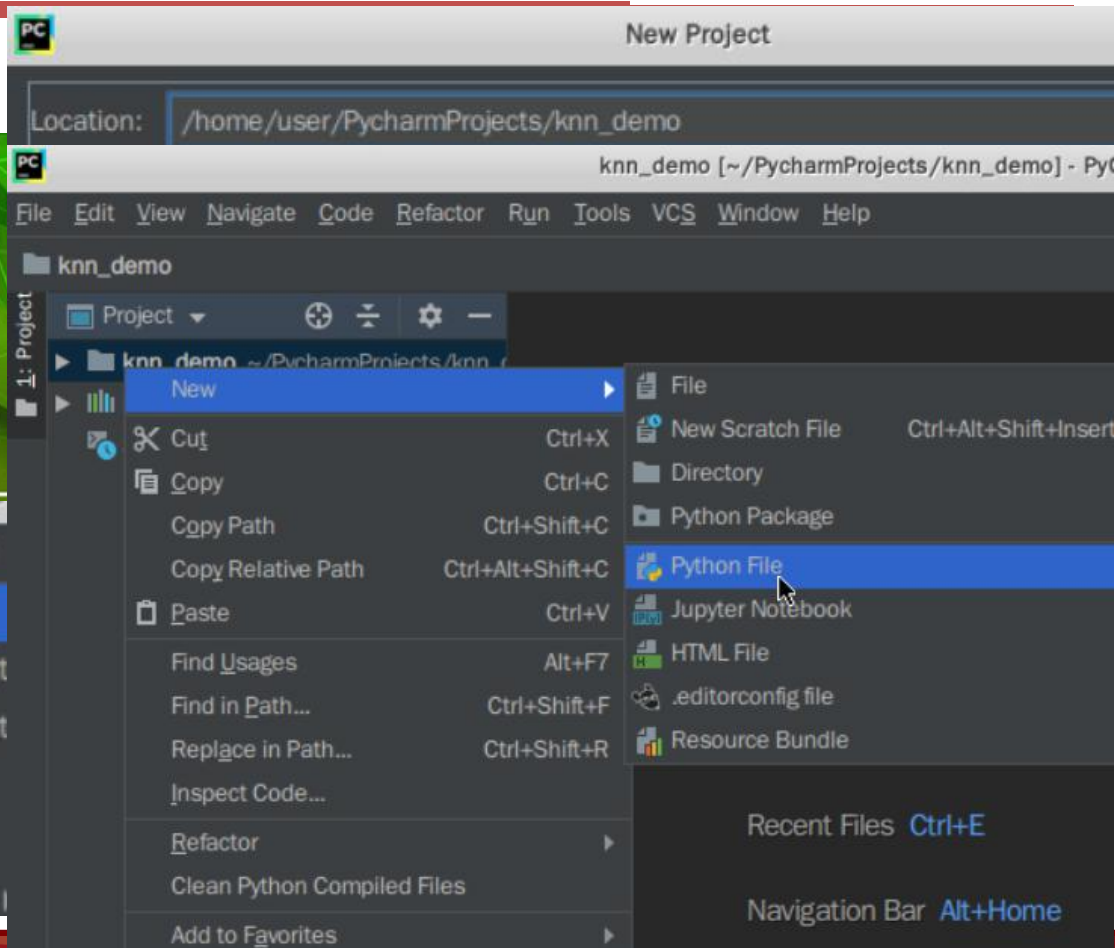
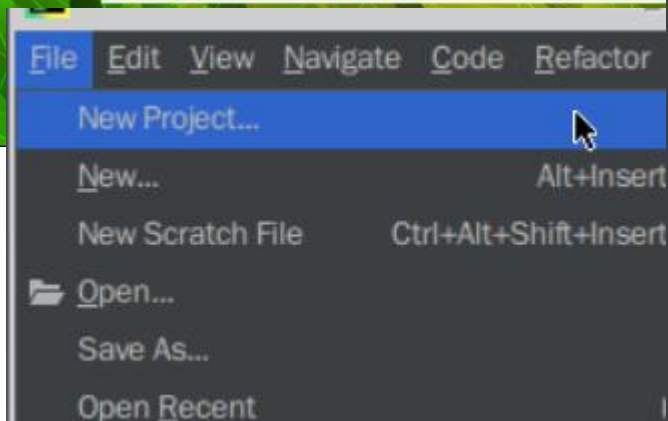


- ❑ 实验目的：编写KNN.py文件，该程序基于对简单数据的分类实现KNN算法的核心思想。
- ❑ 实验内容
 - ❑ (1) 创建脚本文件
 - ❑ (2) 编写KNN算法程序
 - ❑ (3) 运行观察结果

基于python的KNN算法简单实现

4.3 KNN算法

■ (1) 创建脚本文件



- (2) 编写KNN算法程序。具体实验数据如下图所示。

需求：有以下先验数据，

属性1	属性2	类别
1.0	0.9	A
1.0	1.0	A
0.1	0.2	B
0.0	0.1	B

使用KNN算法对未知类别数据分类

属性1	属性2	类别
1.2	1.0	?
0.1	0.3	?

编辑KNNTest.py文件 进行KNN算法思想的 验证实现

```
# 创建一个数据集, 包含2个类别共4个样本
def createDataSet():
    # 生成一个矩阵, 每行表示一个样本
    group = array([[1.0, 0.9], [1.0, 1.0], [0.1, 0.2], [0.0, 0.1]])
    # 4个样本分别所属的类别
    labels = ['A', 'A', 'B', 'B']
    return group, labels
```

```
# KNN分类算法函数定义
def KNNClassify(newInput, dataSet, labels, k):
    numSamples = dataSet.shape[0]
    # shape[0]表示行数
    ## step 1: 计算距离
    # tile(A, reps): 构造一个矩阵, 通过A重复reps次得到
    # the following copy numSamples rows for dataSet
    diff = tile(newInput, (numSamples, 1)) - dataSet # 按元素求差值
    squaredDiff = diff ** 2 # 将差值平方
    squaredDist = sum(squaredDiff, axis = 1) # 按行累加
    distance = squaredDist ** 0.5 # 将差值平方和求开方, 即得距离
```

step 2: 对距离排序

```
# argsort() 返回排序后的索引值
sortedDistIndices = argsort(distance)
classCount = {} # define a dictionary (can be append element)
for i in range(k):
```

step 3: 选择k个最近邻

```
    voteLabel = labels[sortedDistIndices[i]]
```

step 4: 计算k个最近邻中各类别出现的次数

```
# when the key voteLabel is not in dictionary classCount, get()
# will return 0
classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
```

step 5: 返回出现次数最多的类别标签

```
maxCount = 0
for key, value in classCount.items():
    if value > maxCount:
        maxCount = value
        maxIndex = key

return maxIndex
```

- (2) 编写KNN算法程序。
- 编辑KNNTest.py文件进行KNN算法思想的验证实现

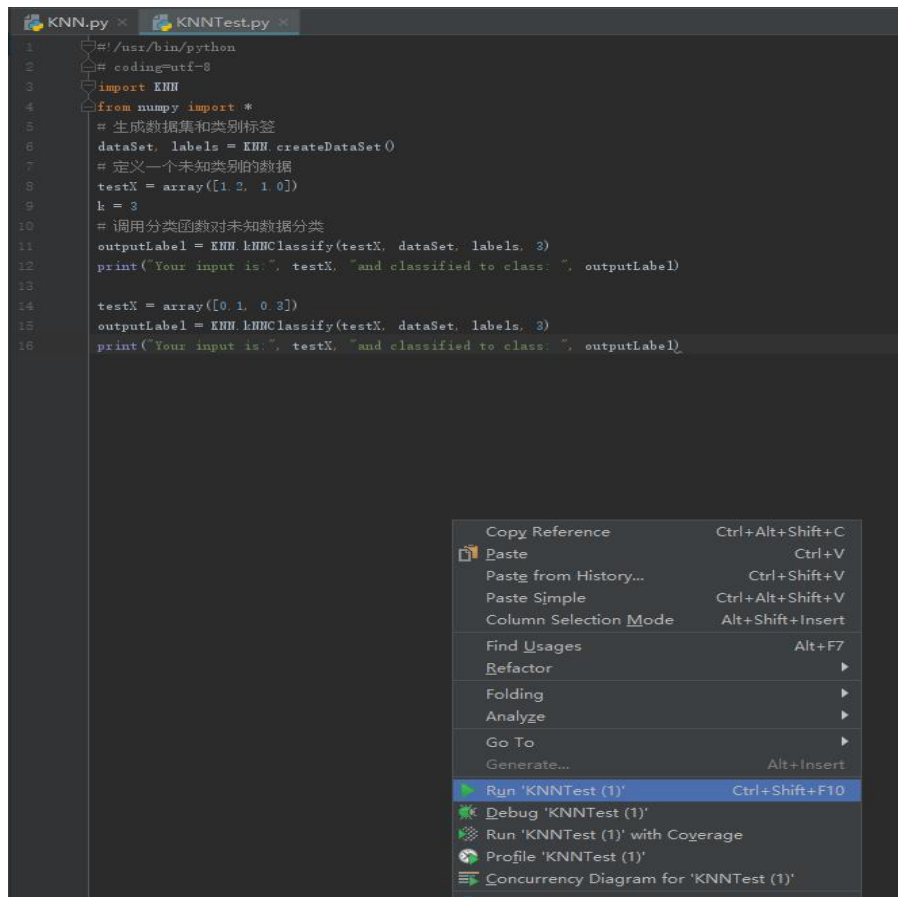
```
KNN.py × KNNTest.py ×
1  #!/usr/bin/python
2  # coding=utf-8
3  import KNN
4  from numpy import *
5  # 生成数据集和类别标签
6  dataSet, labels = KNN.createDataSet()
7  # 定义一个未知类别的数据
8  testX = array([1.2, 1.0])
9  k = 3
10 # 调用分类函数对未知数据分类
11 outputLabel = KNN.kNNClassify(testX, dataSet, labels, 3)
12 print("Your input is:", testX, "and classified to class: ", outputLabel)
13
14 testX = array([0.1, 0.3])
15 outputLabel = KNN.kNNClassify(testX, dataSet, labels, 3)
16 print("Your input is:", testX, "and classified to class: ", outputLabel)
```

- ❑ (3) 运行观察结果：在KNNTTest.py文件空白处，右键，点击Run 'KNNTTest'，运行KNNTTest.py文件
- ❑ 得到输出结果：
 - 数据[1.2 1]分类结果为A;
 - 数据[0.1 0.3]分类结果为B。



```
Your input is: [1.2 1.] and classified to class: A
Your input is: [0.1 0.3] and classified to class: B

Process finished with exit code 0
```



```
KNN.py KNNTTest.py
1 #!/usr/bin/python
2 # coding=utf-8
3 import KNN
4 from numpy import *
5 # 生成数据集和类别标签
6 dataSet, labels = KNN.createDataSet()
7 # 定义一个未知类别的数据
8 testX = array([1.2, 1.0])
9 k = 3
10 # 调用分类函数对未知数据分类
11 outputLabel = KNN.kNNClassify(testX, dataSet, labels, 3)
12 print('Your input is ', testX, "and classified to class: ", outputLabel)
13
14 testX = array([0.1, 0.3])
15 outputLabel = KNN.kNNClassify(testX, dataSet, labels, 3)
16 print('Your input is ', testX, "and classified to class: ", outputLabel)
```

Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Paste from History... Ctrl+Shift+V
Paste Simple Ctrl+Alt+Shift+V
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Refactor
Folding
Analyze
Go To
Generate... Alt+Insert
Run 'KNNTTest (1)' Ctrl+Shift+F10
Debug 'KNNTTest (1)'
Run 'KNNTTest (1)' with Coverage
Profile 'KNNTTest (1)'
Concurrency Diagram for 'KNNTTest (1)'

谢 谢

Thank you for your attention!