模式识别 K 近邻法

一、最近邻、k 近邻算法介绍

1.1 介绍

以每个训练样本为一个子类，不同类的两个样本之间用最小距离作为分类准则。显然这时就没有必要事先用所有两两样本间的分类面构造出分段线性分类面，而是可以在拿到一个待分类的样本后，通过判断它到两类样本的距直来进行决策。这就是最近邻法。

最近邻法就是源于这样一种直观的想法：对于一个新样本，把它逐一与已知样本比较，找出距离新样本最近的已知样本，该样本的类别作为新样本的类别。

在很多情况下，把决策建立在一个最近的样本上有一定风险，当数据分布复杂或数据中噪声严重时尤其如此。一种很自然的改进就是引人投票机制选择前若干个离新样本最近的已知样本，用它们的类别投票来决定新样本的类别，这种方法称作 k-近邻法，因为人们习惯上把参加投票的近邻样本的个数记作 k。显然，最近邻法可以看作是 k 近邻法的特例。

1.2 近邻法的形式化表示

（1）最近邻

已知样本集$S_N = \{(\boldsymbol{x_1}, \theta_1), (\boldsymbol{x_2}, \theta_2), \dots\dots, (\boldsymbol{x_N}, \theta_N)\}$，其中，$x_i$是样本 i 的特征向量，$\theta_i$是它对应的类别，设 c 个类，即$\theta_i \in \{1,2, \dots, c\}$。定义两个样本简得距离量度 $\delta(\boldsymbol{x_i}, \boldsymbol{x_j})$，比如可以采用欧氏距离$\delta(\boldsymbol{x_i}, \boldsymbol{x_j}) = ||\boldsymbol{x_i} - \boldsymbol{x_j}||$。对未知样本$\boldsymbol{x}$，求$S_N$中与之距离最近的样本，设为$\boldsymbol{x}'$（对应的类别为$\theta'$），

即：

$$\delta(\boldsymbol{x_i}, \boldsymbol{x'}) = \min_{j=1,\dots,N} \delta(\boldsymbol{x}, \boldsymbol{x_j})$$

将$\boldsymbol{x}$决策为θ'类。

如果写成判别函数的形式，ω_i类的判别函数可以写作：

$$g_i(\boldsymbol{x}) = \min_{\boldsymbol{x_j} \in \omega_i} \delta(\boldsymbol{x}, \boldsymbol{x_j}) \quad , \quad i = 1, \dots, c$$

决策规则为各类的判别函数比较大小，即：

$$若 g_k(\boldsymbol{x}) = \min_{i=1,\dots,c} g_i(\boldsymbol{x}) \quad , \quad 则 x \in \omega_k$$

错误率：最坏不会超过两倍的贝叶斯错误率，而最好则有可能接近或达到贝叶斯错误率。

注：根据实际问题，可以采用不同的距离度量。

（2）k 近邻

设有 N 个已知样本分属于 c 个类ω_i，$i = 1,2, \dots, c$，考察新样本$\boldsymbol{x}$在这些样本中的前k个近邻，设有k_i个属于ω_i类，则ω_i类的判别函数就是：

$$g_i(\boldsymbol{x}) = k_i \quad , \quad i = 1,2, \dots, c$$

决策规则是：

$$若 g_k(\boldsymbol{x}) = \max_{i=1,\dots,c} g_i(\boldsymbol{x}) \quad , \quad 则 x \in \omega_k$$

错误率：当 k 趋于无穷大时，k 近邻法就达到了贝叶斯错误率。

注：在两分类问题中，通常需要选择 k 为奇数，以避免出现两类的票相等的情况

二、实验数据集介绍

2.1 Iris 数据集介绍

Iris 数据集			
类别	3		
维度	4		
数据长度	150	50	Iris-setosa
		50	Iris-versicolor
		50	Iris-setosa

2.2 Sonar 数据集介绍

Sonar 数据集			
类别	2		
维度	60		
数据长度	208	97	R
		111	M

三、实验设置

对两类数据集，均分别采用最近邻、k 近邻两种方法进行分类，计算准确率，最后将 k 值、测试比例进行变换，观察分类结果。

两类数据集处理方法具有相似性，因此实验设置基本一致，具体如下：

1、读取数据信息。首先从数据集文件（iris.data、sonar.all-data）中读取数据，将读取到的数据按照一定比例（例如：1/5 的数据用于测试，4/5 的数据用于训练）**随机**存放到训练集和测试集中，再将训练集中数据按标签分类（iris 分三类、sonar 分两类）。

2、求解样本间距离。依次遍历测试集中的元素，将测试数据分别与训练集中的每一组数据求解欧氏距离(还可更换其他距离)，将所有距离数据保存。

3、寻找最小距离/分类别投票，求分类结果。

（1）最近邻法：直接遍历即可求出最小距离，该最小距离对应的样本标签即为分类结果。

（2）k 近邻法：将训练集数据的标签和对应距离依据距离进行排序（递增），取前 k 项数据，统计出现次数最多的标签数，该标签即为分类结果。

4、统计所有测试数据分类的正确、错误数量，计算准确率。

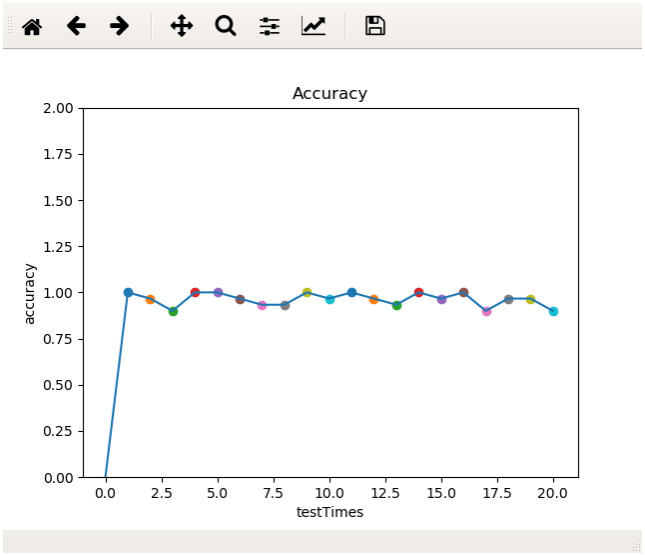
5、重复 1-4 步 20 次，计算平均准确率，绘图显示。

6、更改 k 值，观察准确率变化。

四、实验结果展示与分析

4.1 Iris 数据集分类结果分析

1、首先使用**最近邻**法将 iris 测试集进行分类，默认使用 1/5 的数据用于测试，4/5 的数据用于训练，重复 20 次后，分类精确度如下：

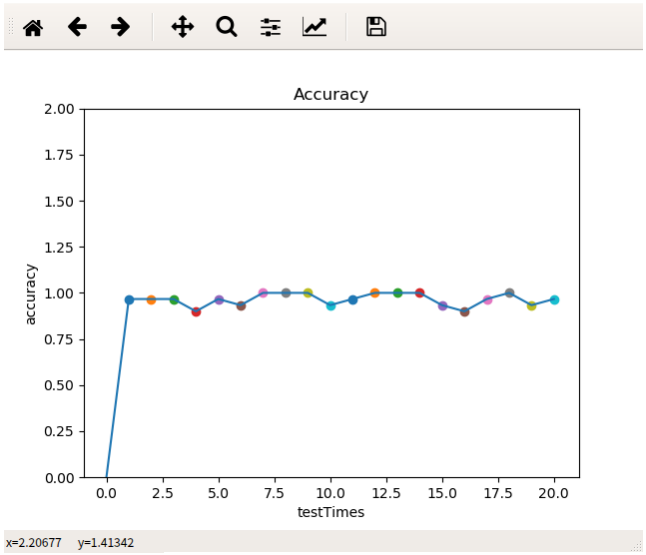



```
Accuracy_ 0 :  1.0
Accuracy_ 1 :  0.9666666666666667
Accuracy_ 2 :  0.9
Accuracy_ 3 :  1.0
Accuracy_ 4 :  1.0
Accuracy_ 5 :  0.9666666666666667
Accuracy_ 6 :  0.9333333333333333
Accuracy_ 7 :  0.9333333333333333
Accuracy_ 8 :  1.0
Accuracy_ 9 :  0.9666666666666667
Accuracy_ 10 :  1.0
Accuracy_ 11 :  0.9666666666666667
Accuracy_ 12 :  0.9333333333333333
Accuracy_ 13 :  1.0
Accuracy_ 14 :  0.9666666666666667
Accuracy_ 15 :  1.0
Accuracy_ 16 :  0.9
Accuracy_ 17 :  0.9666666666666667
Accuracy_ 18 :  0.9666666666666667
Accuracy_ 19 :  0.9
Accuracy_AVE: 0.9633333333333332
(base) deepin@deepin-PC:~/Desktop/
```

图 4.1：20 次最近邻法 iris 数据集分类结果

上述结果可以看出：20 次最近邻法 iris 数据集分类平均准确率 96.3%，由于测试样本较少，还出现了几次准确率为 1 的情况。

2、使用 **k 近邻**法将 iris 数据集分类，设置 k=5，其余设置与上述一致，分类结果如下：

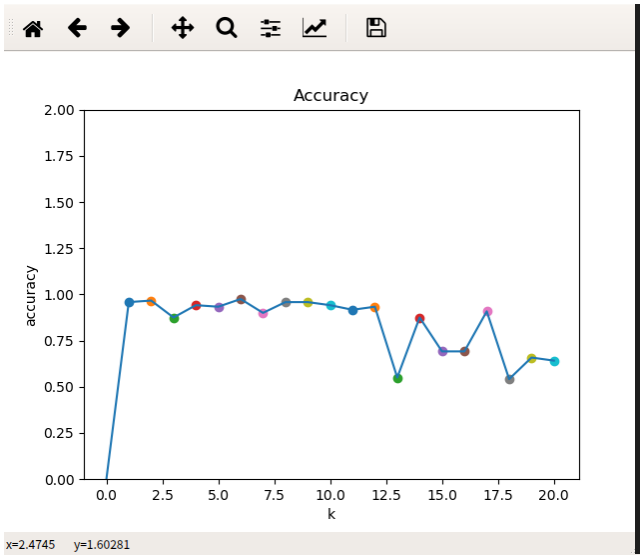



```
Accuracy_ 0 :  0.9666666666666667
Accuracy_ 1 :  0.9666666666666667
Accuracy_ 2 :  0.9666666666666667
Accuracy_ 3 :  0.9
Accuracy_ 4 :  0.9666666666666667
Accuracy_ 5 :  0.9333333333333333
Accuracy_ 6 :  1.0
Accuracy_ 7 :  1.0
Accuracy_ 8 :  1.0
Accuracy_ 9 :  0.9333333333333333
Accuracy_ 10 :  0.9666666666666667
Accuracy_ 11 :  1.0
Accuracy_ 12 :  1.0
Accuracy_ 13 :  1.0
Accuracy_ 14 :  0.9333333333333333
Accuracy_ 15 :  0.9
Accuracy_ 16 :  0.9666666666666667
Accuracy_ 17 :  1.0
Accuracy_ 18 :  0.9333333333333333
Accuracy_ 19 :  0.9666666666666667
Accuracy_AVE: 0.9649999999999999
(base) deepin@deepin-PC:~/Desktop/K
```

图 4.2：20 次 k 近邻法(k=5) iris 数据集分类结果

上述结果可以看出：使用 k 近邻法（k=5）的分类结果与最近邻法相比，精准度略有提高，但相差不多。平均准确率为 96.5%。

3、随后**改变 k** 的值（1-20，对应图中 0-19）进行测试，分类结果如下：




```
Accuracy_ 0 :  0.9583333333333334
Accuracy_ 1 :  0.9666666666666667
Accuracy_ 2 :  0.875
Accuracy_ 3 :  0.9416666666666667
Accuracy_ 4 :  0.9333333333333333
Accuracy_ 5 :  0.975
Accuracy_ 6 :  0.9
Accuracy_ 7 :  0.9583333333333334
Accuracy_ 8 :  0.9583333333333334
Accuracy_ 9 :  0.9416666666666667
Accuracy_ 10 :  0.9166666666666666
Accuracy_ 11 :  0.9333333333333333
Accuracy_ 12 :  0.55
Accuracy_ 13 :  0.875
Accuracy_ 14 :  0.6916666666666667
Accuracy_ 15 :  0.6916666666666667
Accuracy_ 16 :  0.9083333333333333
Accuracy_ 17 :  0.5416666666666666
Accuracy_ 18 :  0.6583333333333333
Accuracy_ 19 :  0.6416666666666667
```

图 4.3：20 次 k 取不同值时 iris 数据集分类结果

上述结果可以看出：k 取值在 12 之前准确率均在较高水平，由于测试集只有 30 个样本（1/5 的数据用于测试），而且包含了 3 类，所以当 k 超过一定值（实验结果 12）时，距离测试样本前 k 近的已知样本中必然包含错误类的样本，导致分类错误，可见，k 不能设置太大。

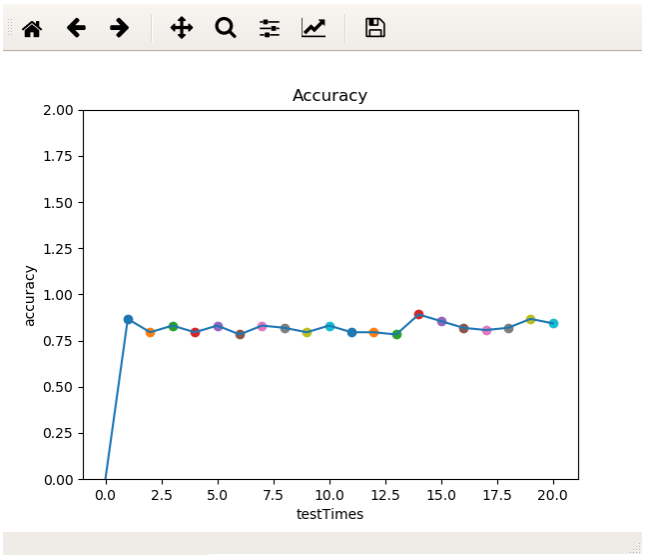
尤其，当 k=1 时，k 近邻法就等价于最近邻法，实验结果也可以看到，k=1 时的准确率与最近邻相似(有区别原因在于每次分类都会重新分配训练、测试数据，导致结果不同)。

k 很小时，准确率也很高，同时也说明 iris 数据集的噪声很小，易分类。

4.2 sonar 数据集分类结果分析

由于两类数据集处理具有相似性，这里只简要描述。

1、首先使用**最近邻**法将 sonar 测试集进行分类（1/5 测试）：




```
Accuracy_ 0 :  0.8674698795180723
Accuracy_ 1 :  0.7951807228915663
Accuracy_ 2 :  0.8313253012048193
Accuracy_ 3 :  0.7951807228915663
Accuracy_ 4 :  0.8313253012048193
Accuracy_ 5 :  0.7831325301204819
Accuracy_ 6 :  0.8313253012048193
Accuracy_ 7 :  0.8192771084337349
Accuracy_ 8 :  0.7951807228915663
Accuracy_ 9 :  0.8313253012048193
Accuracy_ 10 :  0.7951807228915663
Accuracy_ 11 :  0.7951807228915663
Accuracy_ 12 :  0.7831325301204819
Accuracy_ 13 :  0.891566265060241
Accuracy_ 14 :  0.8554216867469879
Accuracy_ 15 :  0.8192771084337349
Accuracy_ 16 :  0.8072289156626506
Accuracy_ 17 :  0.8192771084337349
Accuracy_ 18 :  0.8674698795180723
Accuracy_ 19 :  0.8433734939759037
Accuracy_AVE: 0.8228915662650602
```

图 4.4：20 次最近邻法 sonar 数据集分类结果

上述结果可以看出：20 次最近邻法 sonar 数据集分类平均准确率为 82.3%。

2、使用 **k 近邻**法将 sonar 数据集分类，设置 k=5，其余设置与上述一致，分类结果如下：

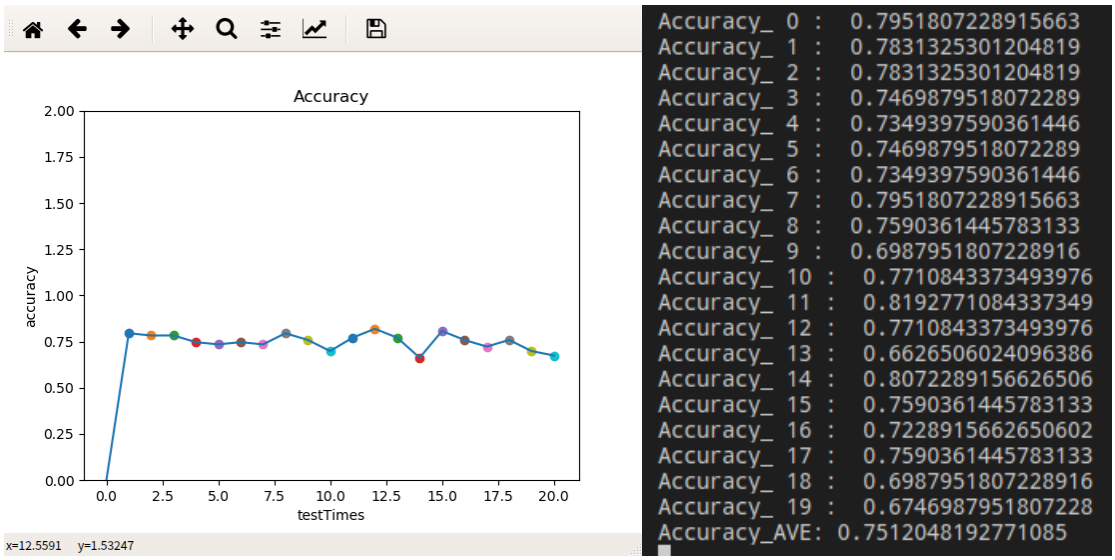


图 4.5：20 次 k 近邻法(k=5) sonar 数据集分类结果

上述结果可以看出：20 次 k 近邻法（k=5）sonar 数据集分类平均准确率为 75.1%，相比最近邻有所下降。

3、随后**改变 k** 的值（1-20，对应图中 0-19）进行测试，分类结果如下：

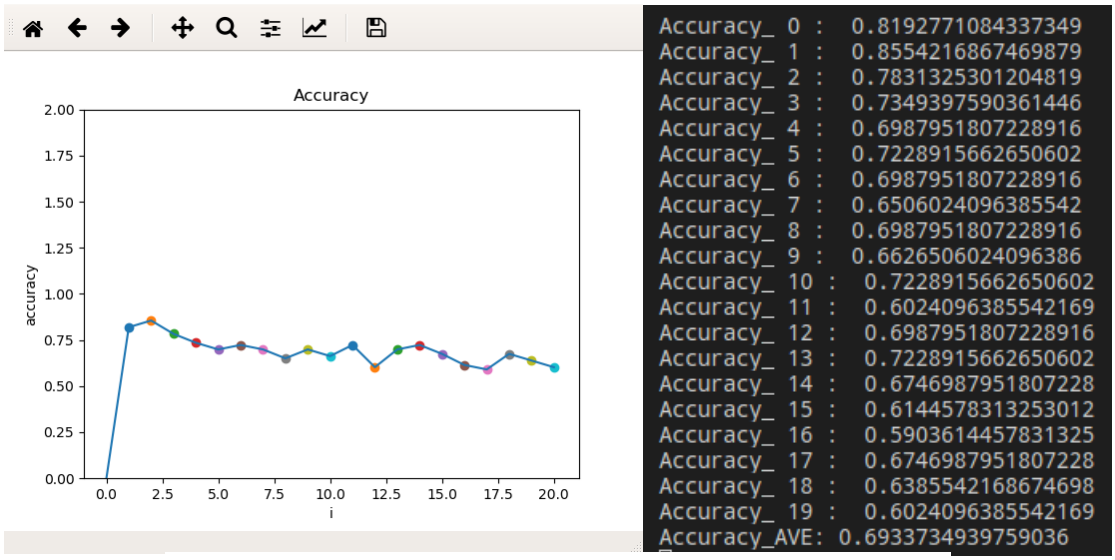


图 4.6：20 次 k 取不同值时 sonar 数据集分类结果

上述结果可以看出：k=2 时准确率最高，随 k 增加准确率下降，由于样本较多，没有 iris 数据集那样变化剧烈。

4、随后**改变测试集和训练集的比例**，可以看到随测试样本增加、已知样本数量减少，准确率逐渐降低：

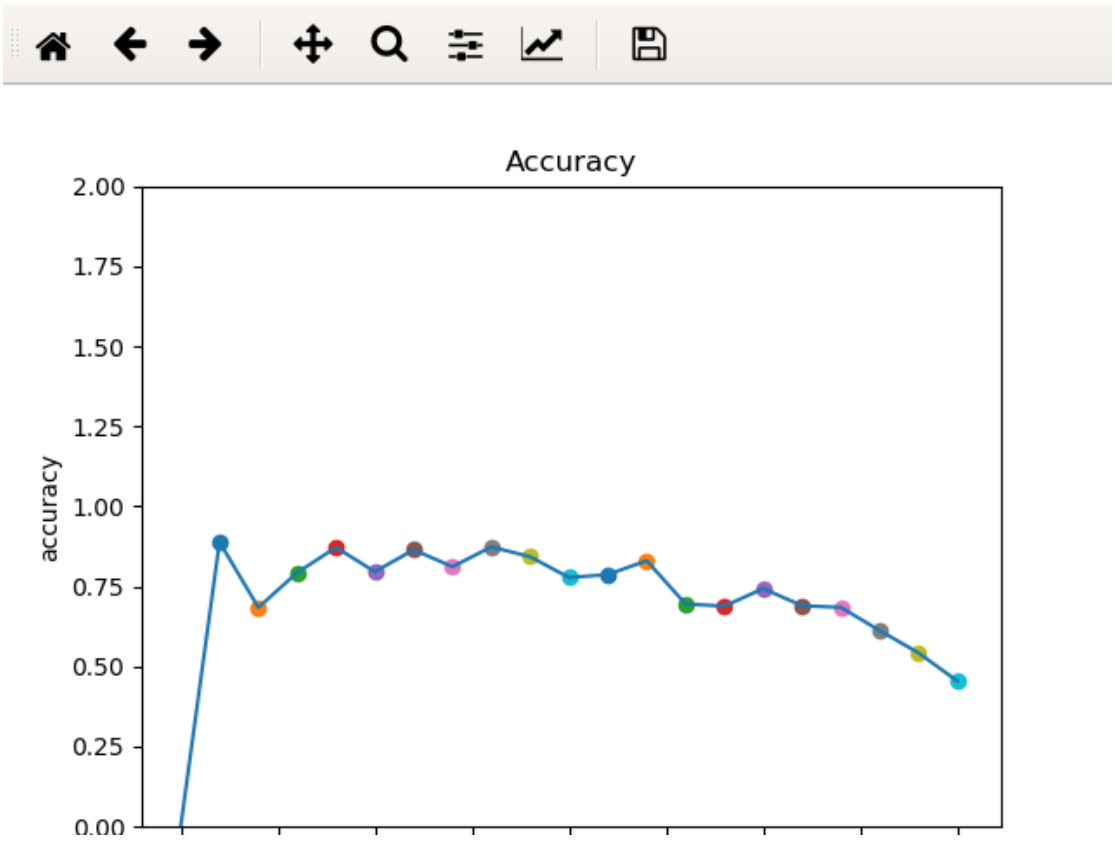


图 4.7：减少已知样本对分类准确率的影响

五、Fisher 与 k 近邻（最近邻）对比

首先，给出每种算法的分类准确率如下：

	fisher	最近邻	k 近邻(k=2)	k 近邻(k=5)
iris 数据集	96.70%	96.30%	96.60%	96.50%
sonar 数据集	72.30%	82.30%	85.50%	75.10%

可以看出，iris 数据集使用 fisher 分类效果最好，sonar 使用 k 近邻或最近邻分类效果好，可见，不同数据集应根据本身特性选用不同的分类方法。

两大类算法具体对比如下：

1、**Fisher 算法**利用训练样本求出的权向量对测试数据进行降维进行分类。

该算法需求出每两类间的判别函数，对两分类问题较为方便，但对多分类问题较为复杂；

当训练样本中有少量噪声时，对分类结果影响不大，但样本有重叠，难以通过投影方式分开时，该方法错误率较高；

随训练样本的数目增加，训练时间会有所增加，但并不影响测试数据的分类时间。

2、**近邻算法**以新样本与的已知样本的距离为依据，寻找距离最近的已知样本或前 k 个距离最近样本类别投票进行分类。

该算法对二分类、多分类问题处理方法相同，方便适用于多分类问题。

当训练样本中有少量噪声时，对最近邻的分类结果影响很大，通过 k 近邻法有所改善，但需要选取合适的 k 值，k 近邻法较 fisher 还改善了样本重叠造成的影响。

近邻法相比于 fisher，没有了训练过程，但如果已知样本数量巨大，由于需要将测试样本与所有已知样本对比一遍，测试时间将非常漫长，分类效率很低，需要通过近邻法的快速算法进行改善。

六、Python 代码

5.1 iris 数据集

```
import os
import sys
import numpy as np
from numpy import *
import random
import matplotlib.pyplot as plt

def readData(DATA_PATH,DATA_SIZE,DATA_DIMENSION,test_rate):
    test_Labels = random.sample(range(0,DATA_SIZE-
1),int(DATA_SIZE*test_rate))
    test_Data = [[0 for i in range(DATA_DIMENSION+1)] for j in range(le
n(test_Labels))]
    train_Data = [[0 for i in range(DATA_DIMENSION+1)] for j in range(D
ATA_SIZE-len(test_Labels))]
    #随机提取训练、测试数据
    f = open(DATA_PATH)
    line = f.readline()
    i = 0
    train_Label = 0
    test_Label = 0
    while line:
        temp = line.split(",",DATA_DIMENSION)
        if i in test_Labels:
            test_Data[test_Label][0:DATA_DIMENSION] = list(map(float, t
emp[0:DATA_DIMENSION]))
            test_Data[test_Label][DATA_DIMENSION] = temp[DATA_DIMENSION
]
            test_Label = test_Label+1
        else:
            train_Data[train_Label][0:DATA_DIMENSION] = list(map(float,
 temp[0:DATA_DIMENSION]))
            train_Data[train_Label][DATA_DIMENSION] = temp[DATA_DIMENSI
ON]
            train_Label = train_Label+1
        i = i+1
        line = f.readline()
    #将要训练的数据分类保存
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'Iris-setosa\n':
```

```
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-versicolor\n':
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Num = _3_Num+1
    _1_Data = mat(zeros((_1_Num,DATA_DIMENSION)))
    _2_Data = mat(zeros((_2_Num,DATA_DIMENSION)))
    _3_Data = mat(zeros((_3_Num,DATA_DIMENSION)))
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'Iris-setosa\n':
            _1_Data[_1_Num] = train_Data[i][0:DATA_DIMENSION]
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-versicolor\n':
            _2_Data[_2_Num] = train_Data[i][0:DATA_DIMENSION]
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Data[_3_Num] = train_Data[i][0:DATA_DIMENSION]
            _3_Num = _3_Num+1
    f.close()
    return train_Data,test_Data,_1_Data.T,_2_Data.T,_3_Data.T

def calcu_Eucli_Dis(x,y):
    dis = np.sqrt(np.sum(np.square(x-y)))
    return dis

def Nearest_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testR
ate = 2/5):
    #数据准备
    train_Data,test_Data,group1,group2,group3 = readData(dataPath,dataS
ize,dataDimension,testRate)
    print("**************************************************************
*********")
    print("                    train_Size:",len(train_Data))
    print("                    test_Size:",len(test_Data))
    print("**************************************************************
*********")

    wrongNum = correctNum = 0

    #遍历每一个测试元素
    for testElement in range(len(test_Data)):
        minDis = 1024
        minLabel = -1
```

```
        print("**********************testElement_",testElement,"*******
*****************")
        #每一个测试元素与训练集中的每一个数据进行距离计算
        for trainElement in range(len(train_Data)):
            testArray = np.array(test_Data[testElement][0:dataDimension
])
            trainArray = np.array(train_Data[trainElement][0:dataDimens
ion])
            #计算距离
            distance = calcu_Eucli_Dis(testArray,trainArray)
            #寻找最近邻元素
            if distance < minDis:
                minDis = distance
                minLabel = trainElement

        print("minDis:",minDis)
        print("Real:",test_Data[testElement][dataDimension],"Forecast:"
,train_Data[minLabel][dataDimension])

        if(test_Data[testElement][dataDimension] == train_Data[minLabel
][dataDimension]):
            correctNum += 1
        else:
            wrongNum += 1
    print("------------------------------------")
    print("|WrongNums:",wrongNum,"correctNums:",correctNum)
    print("------------------------------------\n\n\n")
    Accuracy = correctNum/(correctNum+wrongNum)
    return  Accuracy

def dis_Sort(allDisInfo):
    for i in range(0,len(allDisInfo)):
        for j in range(0,len(allDisInfo)-i-1):
            if allDisInfo[j][1]>allDisInfo[j+1][1]:
                temp = allDisInfo[j]
                allDisInfo[j] = allDisInfo[j+1]
                allDisInfo[j+1] = temp
    return allDisInfo

def k_Judge(allDisInfo,k):
    type_1 = type_2 = type_3 = 0
    for i in range(k):
        if allDisInfo[i][0] == "Iris-setosa\n":
            type_1 += 1
```

```
        if allDisInfo[i][0] == "Iris-versicolor\n":
            type_2 += 1
        if allDisInfo[i][0] == "Iris-virginica\n":
            type_3 += 1
    if type_1>=type_2 and type_1>=type_3:
        return "Iris-setosa\n"
    if type_2>=type_1 and type_2>=type_3:
        return "Iris-versicolor\n"
    if type_3>=type_1 and type_3>=type_2:
        return "Iris-virginica\n"

def K_Nearest_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testRate,k=5):
    #数据准备
    train_Data,test_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)
    print("*************************************************************")
    print("                    train_Size:",len(train_Data))
    print("                    test_Size:",len(test_Data))
    print("*************************************************************")

    wrongNum = correctNum = 0
    #遍历每一个测试元素
    for testElement in range(len(test_Data)):
        allDisInfo = [[0 for i in range(2)] for j in range(len(train_Data))]  #[forcast_label][distance]

        print("********************testElement_",testElement,"*********************")
        #每一个测试元素与训练集中的每一个数据进行距离计算
        for trainElement in range(len(train_Data)):
            testArray = np.array(test_Data[testElement][0:dataDimension])
            trainArray = np.array(train_Data[trainElement][0:dataDimension])
            #计算距离
            dis = calcu_Eucli_Dis(testArray,trainArray)
            #保存所有 label 的距离信息
            allDisInfo[trainElement][0] = train_Data[trainElement][dataDimension]
            allDisInfo[trainElement][1] = dis
```

```
        #对距离信息进行排序
        dis_Sort(allDisInfo)
        for i in range(len(allDisInfo)):
            print(allDisInfo[i])
        #选择 K 值进行判别
        forcast_label = k_Judge(allDisInfo,k)
        print("-----------------------------------
-> Forcast Label:",forcast_label,"-----------------------------------
-> Real Label:",test_Data[testElement][dataDimension])
        if(test_Data[testElement][dataDimension] == forcast_label):
            correctNum += 1
        else:
            wrongNum += 1

    print("-------------------------------------")
    print("|WrongNums:",wrongNum,"correctNums:",correctNum)
    print("-------------------------------------\n\n\n")
    Accuracy = correctNum/(correctNum+wrongNum)
    return  Accuracy


if __name__ == "__main__":
    Accuracy_AVE = 0
    Accuracy = []
    Accuracy.append(0)
    testTimes = 20
    for i in range(testTimes):
        temp = K_Nearest_Neighbors("iris.data",150,4,3,1/5,5)
        # temp = Nearest_Neighbors("iris.data",150,4,3,1/5)
        Accuracy.append(temp)
        plt.scatter(i+1,temp)
        Accuracy_AVE += temp
    Accuracy_AVE /= testTimes
    for i in range(testTimes):
        print("Accuracy_",i,": ",Accuracy[i+1])
    print("Accuracy_AVE:",Accuracy_AVE)
    plt.title("Accuracy")
    plt.xlabel("testTimes")
    plt.ylabel("accuracy")
    plt.ylim(0,2)
    plt.plot(Accuracy)

    plt.show()
```

5.2 sonar 数据集（代码通用，与 iris 相差无几）

```
import os
import sys
import numpy as np
from numpy import *
import random
import matplotlib.pyplot as plt

def readData(DATA_PATH,DATA_SIZE,DATA_DIMENSION,test_rate):
    test_Labels = random.sample(range(0,DATA_SIZE-
1),int(DATA_SIZE*test_rate))
    test_Data = [[0 for i in range(DATA_DIMENSION+1)] for j in range(le
n(test_Labels))]
    train_Data = [[0 for i in range(DATA_DIMENSION+1)] for j in range(D
ATA_SIZE-len(test_Labels))]
    #随机提取训练、测试数据
    f = open(DATA_PATH)
    line = f.readline()
    i = 0
    train_Label = 0
    test_Label = 0
    while line:
        temp = line.split(",",DATA_DIMENSION)
        if i in test_Labels:
            test_Data[test_Label][0:DATA_DIMENSION] = list(map(float, t
emp[0:DATA_DIMENSION]))
            test_Data[test_Label][DATA_DIMENSION] = temp[DATA_DIMENSION
]
            test_Label = test_Label+1
        else:
            train_Data[train_Label][0:DATA_DIMENSION] = list(map(float,
 temp[0:DATA_DIMENSION]))
            train_Data[train_Label][DATA_DIMENSION] = temp[DATA_DIMENSI
ON]
            train_Label = train_Label+1
        i = i+1
        line = f.readline()
    #将要训练的数据分类保存
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'R\n':
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'M\n':
```

```
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Num = _3_Num+1
    _1_Data = mat(zeros((_1_Num,DATA_DIMENSION)))
    _2_Data = mat(zeros((_2_Num,DATA_DIMENSION)))
    _3_Data = mat(zeros((_3_Num,DATA_DIMENSION)))
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'R\n':
            _1_Data[_1_Num] = train_Data[i][0:DATA_DIMENSION]
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'M\n':
            _2_Data[_2_Num] = train_Data[i][0:DATA_DIMENSION]
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Data[_3_Num] = train_Data[i][0:DATA_DIMENSION]
            _3_Num = _3_Num+1
    f.close()
    return train_Data,test_Data,_1_Data.T,_2_Data.T,_3_Data.T

def calcu_Eucli_Dis(x,y):
    dis = np.sqrt(np.sum(np.square(x-y)))
    return dis

def Nearest_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testR
ate = 2/5):
    #数据准备
    train_Data,test_Data,group1,group2,group3 = readData(dataPath,dataS
ize,dataDimension,testRate)
    print("**********************************************************
*********")
    print("                    train_Size:",len(train_Data))
    print("                    test_Size:",len(test_Data))
    print("**********************************************************
*********")

    wrongNum = correctNum = 0

    #遍历每一个测试元素
    for testElement in range(len(test_Data)):
        minDis = 1024
        minLabel = -1
        print("********************testElement_",testElement,"*******
*****************")
```

```
        #每一个测试元素与训练集中的每一个数据进行距离计算
        for trainElement in range(len(train_Data)):
            testArray = np.array(test_Data[testElement][0:dataDimension
])
            trainArray = np.array(train_Data[trainElement][0:dataDimens
ion])
            #计算距离
            distance = calcu_Eucli_Dis(testArray,trainArray)
            #寻找最近邻元素
            if distance < minDis:
                minDis = distance
                minLabel = trainElement

        print("minDis:",minDis)
        print("Real:",test_Data[testElement][dataDimension],"Forecast:"
,train_Data[minLabel][dataDimension])

        if(test_Data[testElement][dataDimension] == train_Data[minLabel
][dataDimension]):
            correctNum += 1
        else:
            wrongNum += 1
    print("------------------------------------")
    print("|WrongNums:",wrongNum,"correctNums:",correctNum)
    print("------------------------------------\n\n\n")
    Accuracy = correctNum/(correctNum+wrongNum)
    return  Accuracy


def dis_Sort(allDisInfo):
    for i in range(0,len(allDisInfo)):
        for j in range(0,len(allDisInfo)-i-1):
            if allDisInfo[j][1]>allDisInfo[j+1][1]:
                temp = allDisInfo[j]
                allDisInfo[j] = allDisInfo[j+1]
                allDisInfo[j+1] = temp
    return allDisInfo

def k_Judge(allDisInfo,k):
    type_1 = type_2 = type_3 = 0
    for i in range(k):
        if allDisInfo[i][0] == "R\n":
            type_1 += 1
        if allDisInfo[i][0] == "M\n":
```

```
            type_2 += 1
        if allDisInfo[i][0] == "Iris-virginica\n":
            type_3 += 1
    if type_1>=type_2 and type_1>=type_3:
        return "R\n"
    if type_2>=type_1 and type_2>=type_3:
        return "M\n"
    if type_3>=type_1 and type_3>=type_2:
        return "Iris-virginica\n"

def K_Nearest_Neighbors(dataPath,dataSize,dataDimension,dataTypeNum,testRate,k=5):
    #数据准备
    train_Data,test_Data,group1,group2,group3 = readData(dataPath,dataSize,dataDimension,testRate)
    print("*************************************************************")
    print("                   train_Size:",len(train_Data))
    print("                   test_Size:",len(test_Data))
    print("*************************************************************")

    wrongNum = correctNum = 0
    #遍历每一个测试元素
    for testElement in range(len(test_Data)):
        allDisInfo = [[0 for i in range(2)] for j in range(len(train_Data))]  #[forcast_label][distance]

        print("**********testElement_",testElement,"**********")
        #每一个测试元素与训练集中的每一个数据进行距离计算
        for trainElement in range(len(train_Data)):
            testArray = np.array(test_Data[testElement][0:dataDimension])
            trainArray = np.array(train_Data[trainElement][0:dataDimension])
            #计算距离
            dis = calcu_Eucli_Dis(testArray,trainArray)
            #保存所有 label 的距离信息
            allDisInfo[trainElement][0] = train_Data[trainElement][dataDimension]
            allDisInfo[trainElement][1] = dis

        #对距离信息进行排序
```

```
        dis_Sort(allDisInfo)
        for i in range(len(allDisInfo)):
            print(allDisInfo[i])
        #选择 K 值进行判别
        forcast_label = k_Judge(allDisInfo,k)
        print("-----------------------------------
-> Forcast Label:",forcast_label,"-----------------------------------
-> Real Label:",test_Data[testElement][dataDimension])
        if(test_Data[testElement][dataDimension] == forcast_label):
            correctNum += 1
        else:
            wrongNum += 1

    print("-------------------------------------")
    print("|WrongNums:",wrongNum,"correctNums:",correctNum)
    print("-------------------------------------\n\n\n")
    Accuracy = correctNum/(correctNum+wrongNum)
    return  Accuracy


if __name__ == "__main__":
    Accuracy_AVE = 0
    Accuracy = []
    Accuracy.append(0)
    testTimes = 20
    for i in range(testTimes):
        temp = K_Nearest_Neighbors("sonar.all-data",208,60,2,2/5,5)
        # temp = Nearest_Neighbors("sonar.all-data",208,60,2,2/5)
        Accuracy.append(temp)
        plt.scatter(i+1,temp)
        Accuracy_AVE += temp
    Accuracy_AVE /= testTimes
    for i in range(testTimes):
        print("Accuracy_",i,": ",Accuracy[i+1])
    print("Accuracy_AVE:",Accuracy_AVE)
    plt.title("Accuracy")
    plt.xlabel("testTimes")
    plt.ylabel("accuracy")
    plt.ylim(0,2)
    plt.plot(Accuracy)

    plt.show()
```