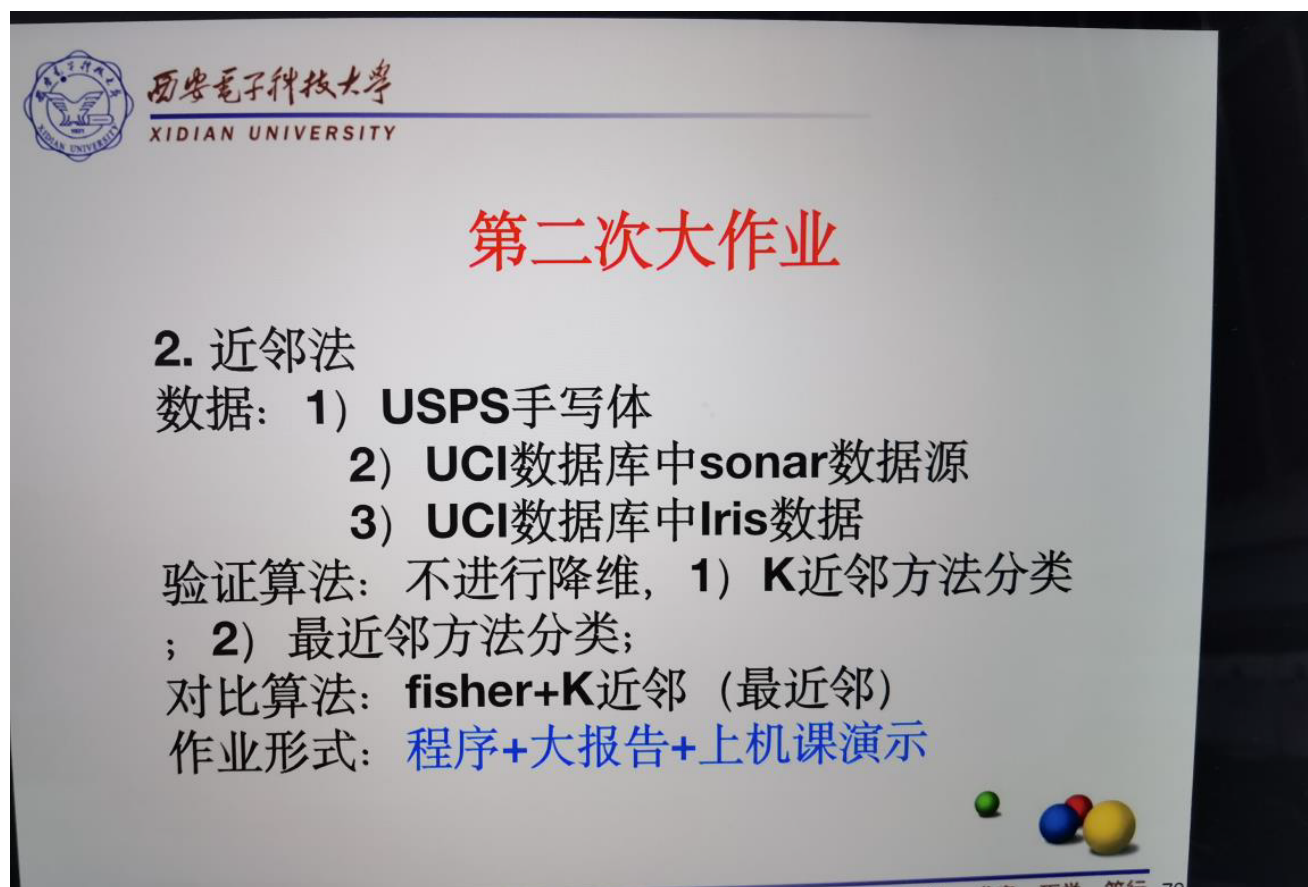


模式识别第二次大作业

姓名：游霄童 学号：21009200158

created: 2023/11/8

一、问题描述



The slide is from Xi'an Jiaotong University (XIDIAN UNIVERSITY) and contains the following information:

- 第二次大作业** (Second Major Assignment)
- 2. 近邻法** (2. Nearest Neighbor Method)
- 数据:** (Data)
 - 1) **USPS** 手写体
 - 2) **UCI** 数据库中 **sonar** 数据源
 - 3) **UCI** 数据库中 **Iris** 数据
- 验证算法:** (Validation Algorithm) 不进行降维, 1) **K** 近邻方法分类; 2) 最近邻方法分类;
- 对比算法:** (Comparison Algorithm) **fisher+K** 近邻 (最近邻)
- 作业形式:** (Assignment Format) 程序+大报告+上机课演示

二、数据集简述:

UCI_Iris数据集:

Iris数据集是一组用于分类问题的常用数据集,来自UCI机器学习库。Iris数据集包含了3种类型鸢尾花的4个特征信息。Iris数据集特征数量少,样本数量适中,适合作为分类算法的「Hello World」示例。

[illegible]

```
[ [5.1 3.5 1.4 0.2]
  [4.9 3.  1.4 0.2]
  [4.7 3.2 1.3 0.2]
  [4.6 3.1 1.5 0.2]
  [5.  3.6 1.4 0.2]
  [5.4 3.9 1.7 0.4]
  [4.6 3.4 1.4 0.3]
  [5.  3.4 1.5 0.2]
  [4.4 2.9 1.4 0.2]
```

基本信息

- 样本数量:150个
- 类别:3类
 - Iris Setosa(山鸢尾)
 - Iris Versicolour(杂色鸢尾)
 - Iris Virginica(维吉尼亚鸢尾)
- 每类样本数量:50个
- 特征数:4个
 - 花萼长度
 - 花萼宽度
 - 花瓣长度
 - 花瓣宽度

数据描述

Iris数据集包含了3种类型鸢尾花的4个特征信息。这是一个多分类问题,常被用来测试分类算法的性能。数据集中的每个样本都属于其中一类,类别是均衡的,每个类型50个样本。4个特征表示花萼和花瓣的长度和宽度,都是正实数。

使用场景

Iris数据集特征数量少,样本数量适中,适合作为分类算法的「Hello World」示例。常用来测试分类算法的效果,如KNN、SVM、决策树等。也可用来比较不同算法的分类性能。由于样本量小,可视化分类结果。

UCI_Sonar数据集

UCI 机器学习库 (UCI Machine Learning Repository) 是一个广泛用于机器学习和数据挖掘研究的资源，其中包含了许多开源数据集，Sonar数据集就是其中之一。Sonar数据集（也称为声纳数据集）是一个经典的二分类问题数据集，用于声纳信号处理和目标检测领域的研究。

网址：<https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/sonar.all-data>

	attribute_1	attribute_2	...	attribute_58	attribute_59	attribute_60	Class
1	0.02	0.0371	...	0.0084	0.009	0.0032	Rock
2	0.0453	0.0523	...	0.0049	0.0052	0.0044	Rock
3	0.0262	0.0582	...	0.0164	0.0095	0.0078	Rock
4	0.01	0.0171	...	0.0044	0.004	0.0117	Rock
...
208	0.026	0.0363	...	0.0036	0.0061	0.0115	Mine

基本信息

- 样本数量:208
- 类别:2类
 - Rock(矿石)
 - Metal(金属)
- 特征数:60
- 特征信息:声纳返回信号在60个频带中的能量强度

数据描述

这是一个二分类问题,根据声纳返回信号区分矿石和金属。数据来自实际的声纳信号采集。相对Iris数据集,它有更多的特征数,样本不均衡,是一个更难的二分类问题。

使用场景

Sonar数据集可以用来测试二分类算法在高维特征上的表现。样本量适中,可用于比较不同二分类算法的效果。也可研究特征工程技术在高维数据上的应用。因为样本不均衡,还可研究分类算法处理样本不均衡的技巧。

Usps 手写体数据集

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

(k)

(l)

(m)

(n)

(o)

(p)

(q)

(r)

USPS手写体数据集（United States Postal Service Database）是一个广泛用于机器学习和模式识别研究的手写数字数据集。该数据集由美国邮政服务（USPS）提供，用于邮政信封上的邮政编码自动识别系统的训练和测试。

该数据集包含7,291张16x16像素的手写数字图像，涵盖了数字0到9。这些图像已经被预处理和标准化，使得它们适合用于机器学习算法的训练和测试。每张图像都被展平成256个像素值的向量，用于作为特征输入。

USPS手写体数据集通常被用来进行数字识别（手写数字识别）的实验和性能评估。研究人员和学生可以使用这个数据集来测试他们的算法在处理手写数字识别任务上的表现。

这个数据集在机器学习和模式识别领域广泛被使用，特别是在测试各种分类算法（如支持向量机、神经网络等）的性能时。由于它是一个相对较小的数据集，它通常被用来进行算法的快速原型开发和调试。

三、算法分析

1、KNN算法

邻近算法，或者说 K 近邻(kNN, k-NearestNeighbor)分类算法是数据挖掘分类技术中最简单的方法之一。所谓 K 近邻，就是 k 个最近的邻居的意思，说的是每个样本都可以用它最接近的 k 个邻居来代表。KNN 算法本身简单有效，它是一种 lazy-learning 算法，分类器不需要使用训练集进行训练，训练时间复杂度为 0，KNN 分类的计算复杂度和训练集中的样本数目成正比。

由于 KNN 方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他方法更为适合。

当 KNN 算法的超参数 $k = 1$ 时，算法直接由与测试样本最近的训练集样本决定测试样本类别，此时算法又称为最近邻算法。k 值的选取是影响 KNN 算法准确率的重要因素。1、KNN 中的距离度量

两个样本特征向量之间的距离反映了其相似程度。常见的距离有欧氏距离、曼哈顿距离和 L_p 距离等。

设样本空间 $X \subseteq R^n$ ，n 为数据特征向量维度。定义 L_p 为

$$L_p(x_i, x_j) = \left(\sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|^p \right)^{1/p}$$

其中 $p \geq 1$ ，当 $p = 2$ 时，称为欧氏距离

$$L_2(x_i, x_j) = \left(\sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|^2 \right)^{1/2}$$

当 $p = 1$ 时，称为曼哈顿距离

$$L_1(x_i, x_j) = \sum_{l=1}^n |x_i^{(l)} - x_j^{(l)}|$$

当 $p = \infty$ 时，它代表各个坐标距离的最大值，即

$$L_\infty(x_i, x_j) = \max_l |x_i^{(l)} - x_j^{(l)}|$$

2、KNN 中的 k 值选择

KNN 算法最重要的超参数就是 k 值。其选取会显著影响算法的准确性。如果 k 值较小，则表示在分类时依靠的样本数更少，因此更容易造成过拟合。如果 k 值较大，与预测样本点距离较远的样本也会参与投票，这样会导致错误预测的概率更大。

一般情况下，k 值从较小值开始尝试，然后通过交叉验证选取最优的 k 值。另外，k 值一般不取偶数，因为这样在投票表决过程中会产生矛盾。

3、KKN 中的分类决策准则

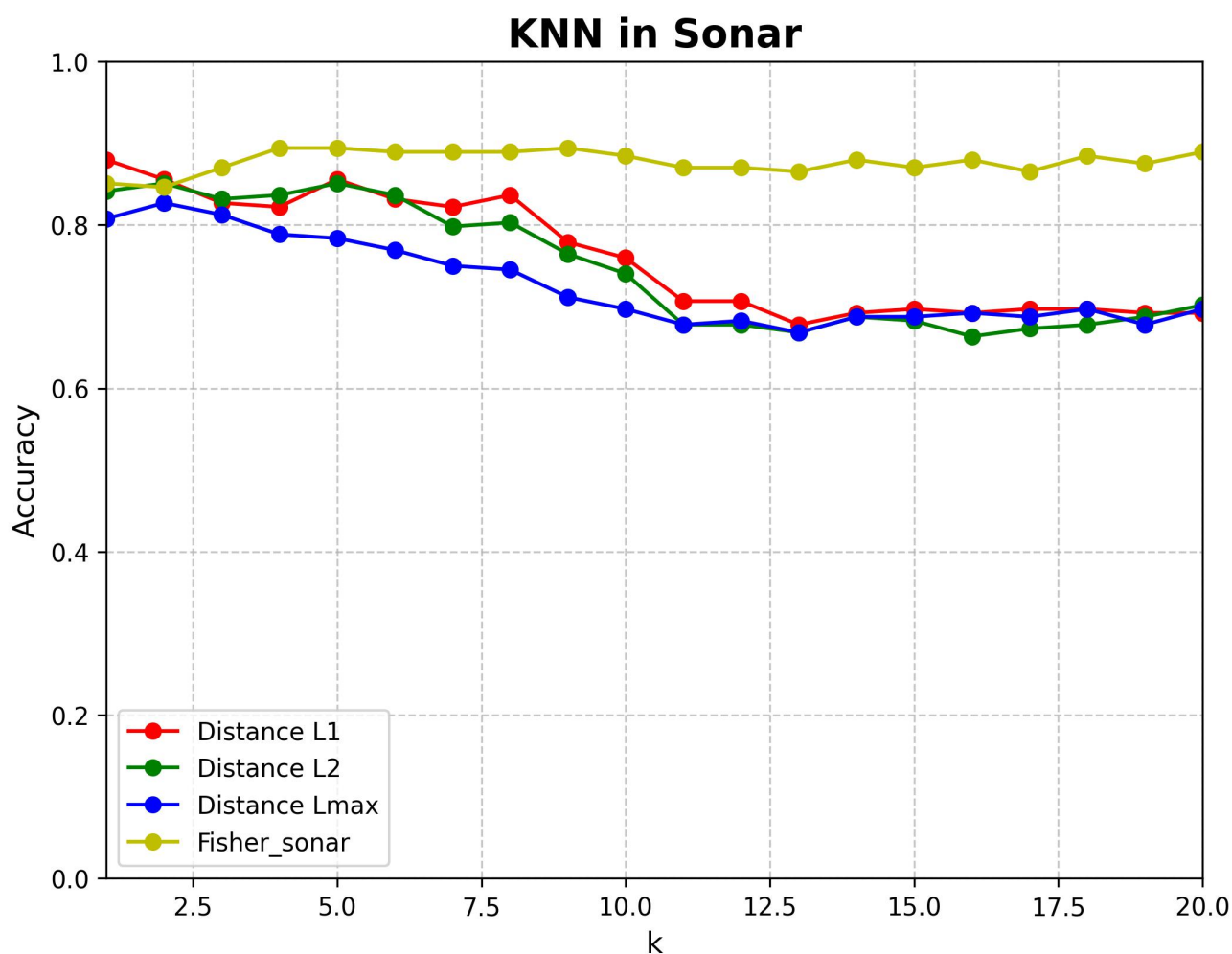
KNN 算法使用多数表决来决定测试样本分类：已知样本的 k 个近邻训练样本，统计其中最多的类别作为测试样本的类别，少数服从多数。

4、Fisher+KNN对比

将数据首先进行Fisher降维后再进行KNN分类算法，与只进行KNN算法进行对比。

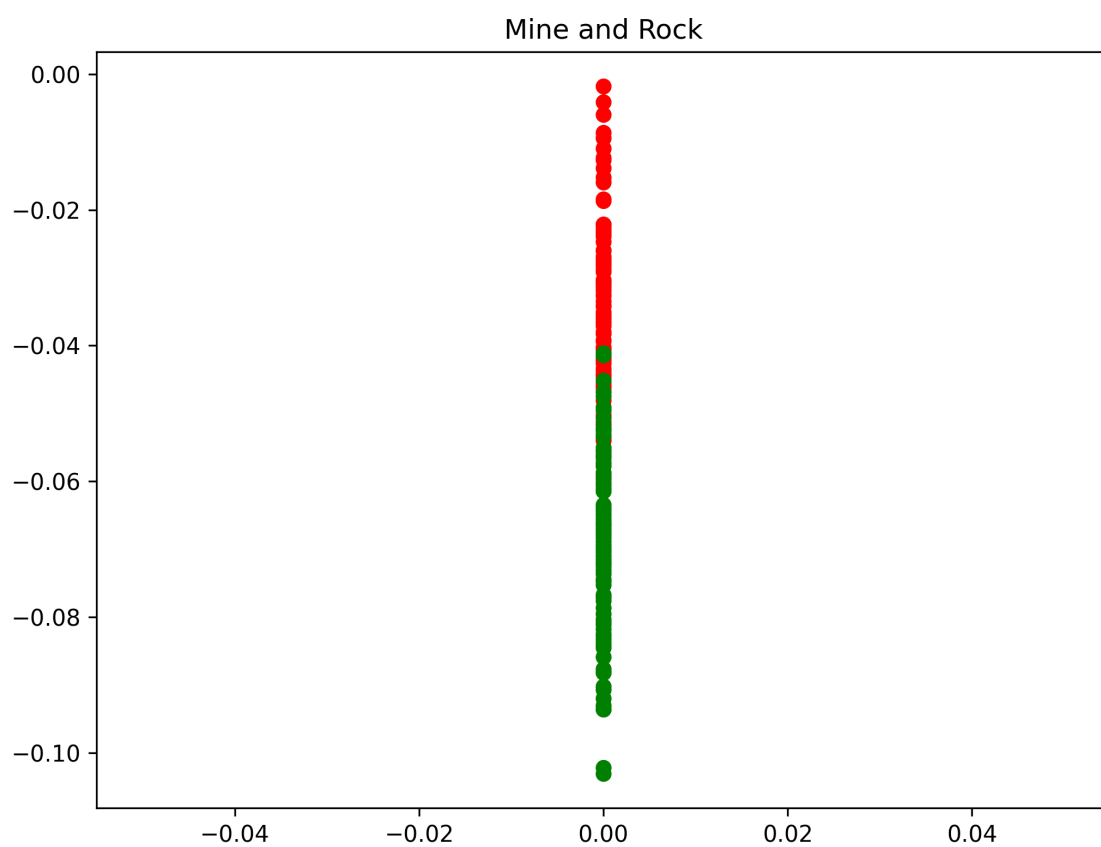
四、实验结果

1、Sonar 数据集



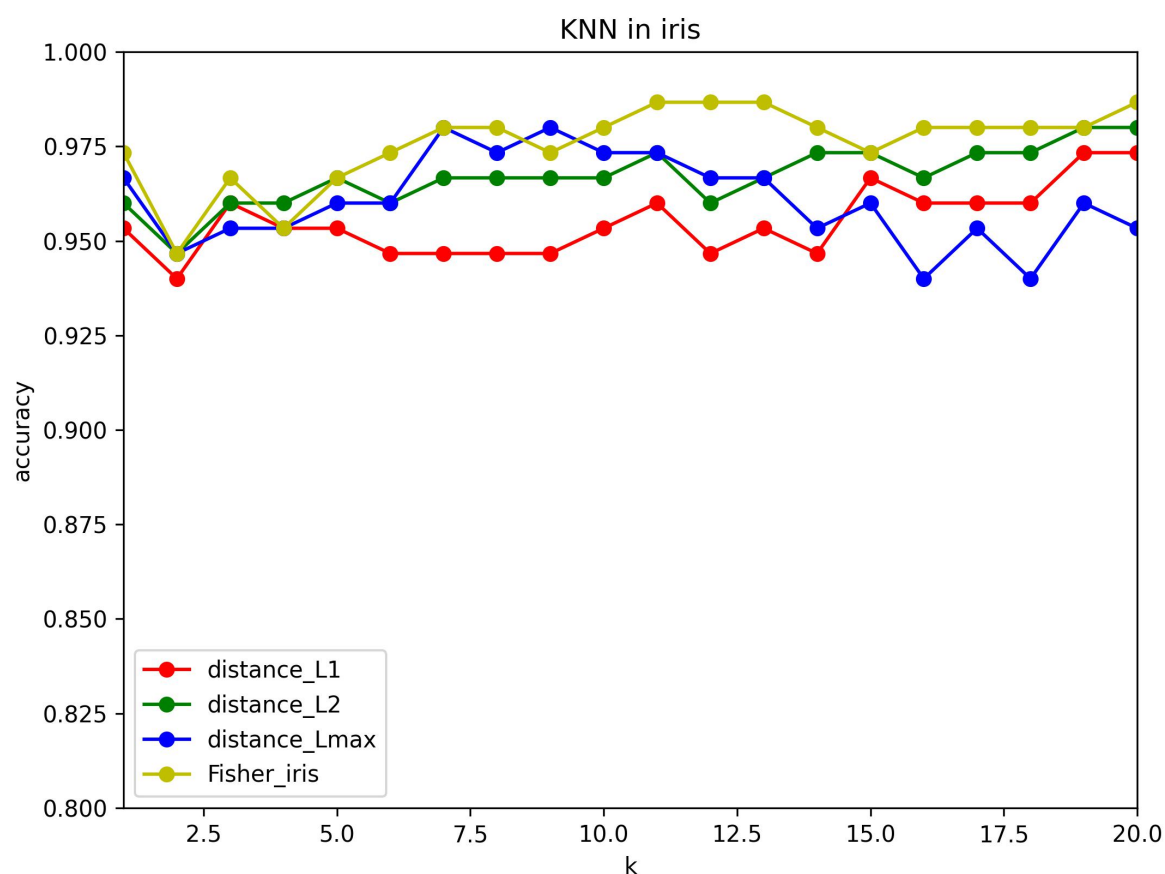
其中L1、L2指一范数与二范数即曼哈顿距离和欧几里得距离，Lmax为距离的最大值，Fisher_sonar为先进行fisher降低为一维后在进行KNN的结果。

横坐标为K即选取的最近邻的个数，纵坐标为准确率。



该图为sonar数据集降低为一维的结果。

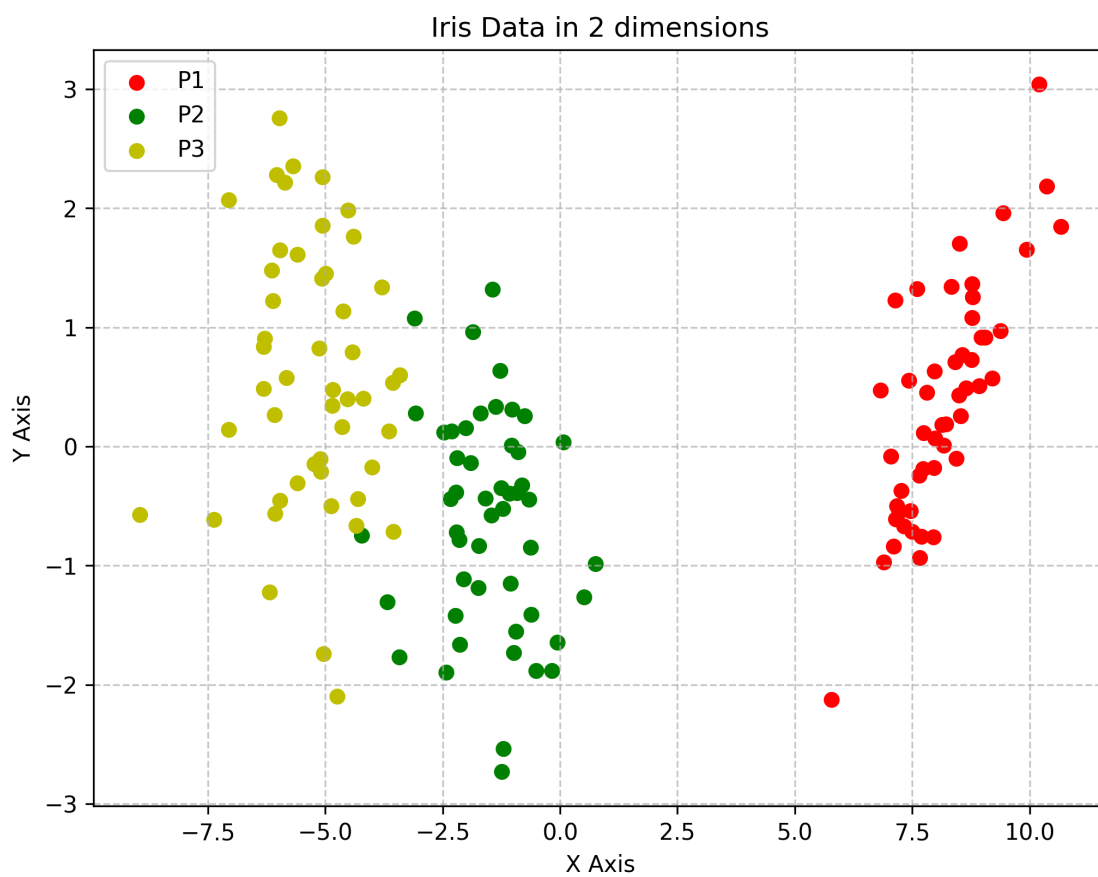
2、Iris 数据集



其中L1、L2指一范数与二范数即曼哈顿距离和欧几里得距离，Lmax为距离的最大值，Fisher_iris为先进行fisher降低为一维后在进行KNN的结果。

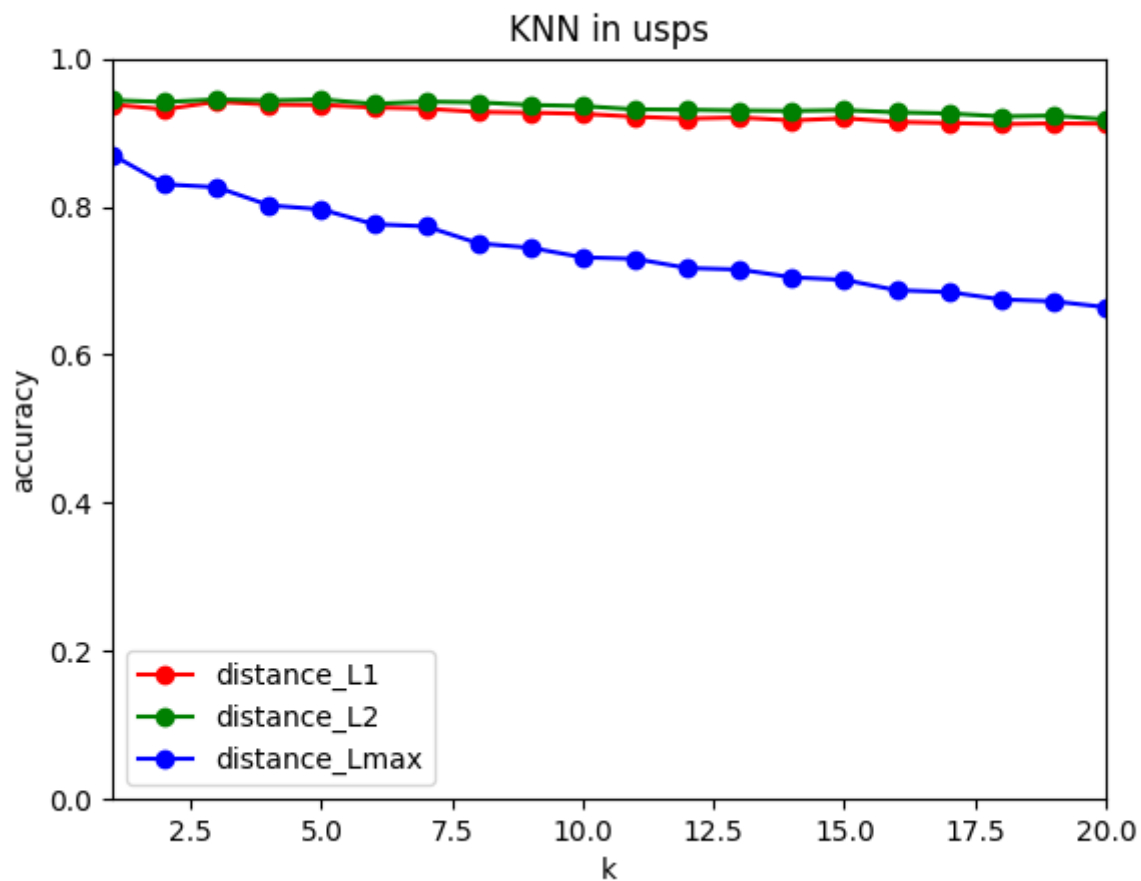
横坐标为K即选取的最近邻的个数，纵坐标为准确率。

由于iris数据集相对简单，准确率都逼近0.95+，故将纵坐标设置在0.8-1.0，更能看清四者的区别。



该图为iris数据集在二维平面上的结果。

3、Usps 数据集



其中L1、L2指一范数与二范数即曼哈顿距离和欧几里得距离，Lmax为距离的最大值。

横坐标为K即选取的最近邻的个数，纵坐标为准确率。

可以看出最大值距离在Usps数据集上表现不佳，而曼哈顿距离和欧几里得距离相近。

五、源代码如下

1、Sonar

```
###
from tqdm import tqdm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
###
sonar = pd.read_csv('sonar.all-data.csv', header=None, sep=',')
sonar1 = sonar.iloc[0:208,0:60]
sonar2 = np.array(sonar1)
y1 = np.zeros(104)
y2 = np.ones(104)
y = np.append(y1, y2)
###
def KNN(k, X_tr, y_tr, X_te, y_te, distance_type):
    """
```

κ最近邻分类器函数

参数:

k (int): 最近邻数目

X_tr (numpy.ndarray): 训练集特征矩阵

y_tr (numpy.ndarray): 训练集标签数组

X_te (numpy.ndarray): 测试集特征矩阵

y_te (numpy.ndarray): 测试集标签数组

distance_type (str): 距离度量类型, 如 'euclidean', 'manhattan' 等

返回:

accuracy (float): 分类器在测试集上的准确率

"""

```
accuracy = 0 # 初始化准确率
```

```
for i in range(X_te.shape[0]): # 遍历测试集中的每个样本
```

```
    distance = np.zeros((2, X_tr.shape[0])) # 创建一个二维数组, 用于存储  
    距离和对应的标签
```

```
    clas = np.zeros(2) # 创建一个数组, 用于统计最近邻中不同类别的数量
```

```
    for j in range(X_tr.shape[0]): # 遍历训练集中的每个样本
```

```
        # 计算测试样本与训练样本之间的距离
```

```
        distance[1, j] = np.linalg.norm(X_te[i] - X_tr[j],  
ord=distance_type)
```

```
        distance[0, j] = y_tr[j] # 存储对应的训练样本的标签
```

```
    index = np.lexsort(distance) # 使用lexsort进行多级排序, 返回排序后的  
    索引
```

```
    for l in range(k): # 获取最近的k个样本
```

```
        clas[int(distance[0, index[l]])] += 1 # 统计各类别的数量
```

```
    prediction = clas.argmax() # 找出数量最多的类别, 作为预测结果
```

```
    #若有类别投票结果相同, 那么根据argmax函数方法, 返回第一个作为结果
```

```
    if prediction == y_te[i]: # 如果预测结果与真实标签一致
```

```
        accuracy += 1 # 增加准确率计数
```

```
    accuracy = accuracy / X_te.shape[0] # 计算准确率, 即正确预测的样本数除以  
    总样本数
```

```
    return accuracy # 返回准确率
```

```
%%%
```

```
accuracy_all = np.zeros((4, 20))
```

```
%%%
```

```
def KNN_sonar(k, y, sonar2, distace_type):
```

```
    acc = 0
```

```
    for i in range(208):
```

```
        X_te = sonar2[i].reshape((1, 60))
```

```
        X_tr = np.delete(sonar2, i, axis=0)
```

```

        y_te = y[i].reshape(1)
        y_tr = np.delete(y, i, axis=0)
        acc += KNN(k, X_tr, y_tr, X_te, y_te, distace_type)
    return acc/208

#%%
for m in tqdm(range(20)):
    k = m + 1
    #1为一范数2为二范数, inf为无穷范数, 即吗, 曼哈顿距离, 欧氏距离, 绝对值最大
    accuracy_all[0][m] = KNN_sonar(k, y, sonar2, 1)
    accuracy_all[1][m] = KNN_sonar(k, y, sonar2, 2)
    accuracy_all[2][m] = KNN_sonar(k, y, sonar2, np.inf)
    pass

#%%
def Fisher(X1, X2, n): #Fisher线性判别过程
    X1 = X1[:, 0:n]
    X2 = X2[:, 0:n]
    m1 = (np.mean(X1, axis=0))
    m2 = (np.mean(X2, axis=0))
    m1 = m1.reshape(n, 1) # 将行向量转换为列向量以便于计算
    m2 = m2.reshape(n, 1)

    # 计算类内离散度矩阵
    S1 = np.zeros((n, n))
    S2 = np.zeros((n, n))
    for i in range(0, X1.shape[0]):
        S1 += (X1[i].reshape(n, 1) - m1).dot((X1[i].reshape(n, 1) -
m1).T)
    for i in range(0, X2.shape[0]):
        S2 += (X2[i].reshape(n, 1) - m2).dot((X2[i].reshape(n, 1) -
m2).T)
    # 计算总类内离散度矩阵S_w
    S_w = S1 + S2

    # 计算最优投影方向 w
    W = np.linalg.inv(S_w).dot(m1 - m2)
    # 在投影后的一维空间求两类的均值
    m_1 = (W.T).dot(m1)
    m_2 = (W.T).dot(m2)

    # 计算分类阈值 w0 (为一个列向量)
    W0 = 0.5 * (m_1 + m_2)

    return W, W0

def Classify(X, W, W0, n):
    y = (W.T).dot(X[0:n, :]) - W0
    return y

```

```

P1 = sonar2[0:104, 0:60]
P2 = sonar2[104:208, 0:60]

result = np.zeros(208)
Accuracy = np.zeros(60)

for n in range(1, 61):
    count = 0
    for i in range(208):
        if i <= 103:
            test = P1[i]
            test = test.reshape(60, 1)
            train = np.delete(P1, i, axis=0)
            W,W0 = Fisher(train, P2, n)
            if (Classify(test, W, W0, n)) >= 0:
                count += 1
                result[i] = Classify(test, W, W0, n)
        else:
            test = P2[i-104]
            test = test.reshape(60, 1)
            train = np.delete(P2, i-104, axis=0)
            W,W0 = Fisher(P1, train, n)
            if (Classify(test, W, W0, n)) < 0:
                count += 1
                result[i] = Classify(test, W, W0, n)
    Accuracy[n-1] = count/208
    #print("分类准确率在维数取%d时为:%.3f" % (n, Accuracy[n-1]))
    ###
    #class1 and class2
    answer1=P1.dot(W)
    answer2=P2.dot(W)
    plt.figure(figsize=(8, 6), dpi=300) # 调整图形大小和分辨率
    plt.title("Mine and Rock")
    plt.plot(answer1.T,'r-o')
    plt.plot(answer2.T,'g-o')
    plt.savefig('Fisher_Sonar')
    plt.show()
    ###
    sonar2.shape,W.shape
    ###
    fisher_sonar2=sonar2.dot(W)
    fisher_sonar2
    ###
def KNN_sonar_fisher(k, y, sonar2, distace_type):
    acc = 0
    for i in range(208):
        X_te = sonar2[i]
        X_tr = np.delete(sonar2, i, axis=0)

```

```

        y_te = y[i].reshape(1)
        y_tr = np.delete(y, i, axis=0)
        acc += KNN(k, X_tr, y_tr, X_te, y_te, distance_type)
    return acc/208

for m in tqdm(range(20)):
    k = m + 1
    #1为一范数2为二范数, inf为无穷范数, 即吗, 曼哈顿距离, 欧氏距离, 绝对值最大
    accuracy_all[3][m] = KNN_sonar_fisher(k, y, fisher_sonar2, 1)
    pass

#%%
x = np.arange(1, 21, 1)
# 设置图形参数
plt.figure(figsize=(8, 6), dpi=300) # 调整图形大小和分辨率
plt.title("KNN in Sonar", fontsize=16, fontweight='bold') # 设置标题
plt.xlabel('k', fontsize=12) # 设置x轴标签
plt.ylabel('Accuracy', fontsize=12) # 设置y轴标签
plt.xticks(fontsize=10) # 设置x轴刻度文字大小
plt.yticks(fontsize=10) # 设置y轴刻度文字大小
plt.xlim((1, 20)) # 设置x轴范围
plt.ylim((0, 1.0)) # 设置y轴范围
plt.grid(True, linestyle='--', alpha=0.7) # 添加网格线

# 绘制曲线
plt.plot(x, accuracy_all[0], 'r-o', label="Distance L1") # 绘制第一条曲线, 红色圆点线条
plt.plot(x, accuracy_all[1], 'g-o', label="Distance L2") # 绘制第二条曲线, 绿色圆点线条
plt.plot(x, accuracy_all[2], 'b-o', label="Distance Lmax") # 绘制第三条曲线, 蓝色圆点线条
plt.plot(x, accuracy_all[3], 'y-o', label="Fisher_sonar")
# 添加图例
plt.legend(fontsize=10)

# 保存和显示图形
plt.savefig('result_of_KNN_in_Sonar.jpg', dpi=300, bbox_inches='tight')
# 保存图形并设置边界框紧凑
plt.show()
#%%

```

2、Iris

```

#%%
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from tqdm import tqdm

```

```

#%%
iris = datasets.load_iris()
X = iris.data[0:150, 0:4]
y1 = np.zeros(50)
y2 = np.ones(50)
y3 = y2*2
y = np.append(np.append(y1, y2), y3)
#%%
def KNN(k, X_tr, y_tr, X_te, y_te, distace_type):
    accuracy = 0
    for i in range(X_te.shape[0]):
        distance = np.zeros((2, X_tr.shape[0]))
        clas = np.zeros(3)
        for j in range(X_tr.shape[0]):
            distance[1, j] = np.linalg.norm(X_te[i] -
X_tr[j], distace_type)
            distance[0, j] = y_tr[j]
        index = np.lexsort(distance)
        for l in range(k):
            clas[int(distance[0,index[l]])] +=1
        prediction = clas.argmax()
        if prediction == y_te[i]:
            accuracy +=1

    accuracy = accuracy/X_te.shape[0]
    return accuracy
#%%
accuracy_all = np.zeros((4,20))
#%%
def KNN_iris(k, y, X, distace_type):
    acc = 0
    for i in range(150):
        X_te = X[i].reshape((1,4))
        X_tr = np.delete(X, i, axis=0)
        y_te = y[i].reshape(1)
        y_tr = np.delete(y, i, axis=0)
        acc += KNN(k, X_tr, y_tr, X_te, y_te, distace_type)
    return acc/150
#%%
for m in tqdm(range(20)):
    k = m + 1
    accuracy_all[0][m] = KNN_iris(k, y, X, 1)
    accuracy_all[1][m] = KNN_iris(k, y, X, 2)
    accuracy_all[2][m] = KNN_iris(k, y, X, np.inf)
    pass
#%%
x = np.arange(1,21,1)
plt.figure(figsize=(8, 6), dpi=300) # 调整图形大小和分辨率
plt.title("KNN in iris")

```

```

plt.xlabel('k')
plt.ylabel('accuracy')
plt.xlim((1, 20))
plt.ylim((0, 1.0))
plt.plot(x, accuracy_all[0], 'r-o', label = "distance_L1")
plt.plot(x, accuracy_all[1], 'g-o', label = "distance_L2")
plt.plot(x, accuracy_all[2], 'b-o', label = "distance_Lmax")
plt.legend()
plt.savefig('result of KNN in iris.jpg')
plt.show()

###

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# 加载Iris数据集
iris = load_iris()
X = iris.data
y = iris.target
P1 = iris.data[0:50, 0:4]
P2 = iris.data[50:100, 0:4]
P3 = iris.data[100:150, 0:4]
# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# 初始化Fisher LDA模型，设置降维后的维数为n_components
n_components = 2 # 指定降维后的维数
lda = LinearDiscriminantAnalysis(n_components=n_components)

# 训练模型并降维
X_train_lda = lda.fit_transform(X_train, y_train)
X_lda = lda.transform(X)
P1_lda = lda.transform(P1)
P2_lda = lda.transform(P2)
P3_lda = lda.transform(P3)
###

X_lda
###

plt.figure(figsize=(8, 6), dpi=300) # 调整图形大小和分辨率
plt.scatter(P1_lda[:,0],P1_lda[:,1], color='r', label='P1')
plt.scatter(P2_lda[:,0],P2_lda[:,1], color='g', label='P2')
plt.scatter(P3_lda[:,0],P3_lda[:,1], color='y', label='P3')

# 添加标题和坐标轴标签
plt.title('Iris Data in 2 dimensions')
plt.xlabel('X Axis')

```



```

plt.ylabel('Y Axis')
plt.grid(True, linestyle='--', alpha=0.7) # 添加网格线
# 显示图例
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7) # 添加网格线
plt.savefig('Iris Data in 2 dimensions')
# 显示散点图
plt.show()
#%%
def KNN_iris2(k, y, X, distace_type):
    acc = 0
    for i in range(150):
        X_te = X[i].reshape((1,2))
        X_tr = np.delete(X, i, axis=0)
        y_te = y[i].reshape(1)
        y_tr = np.delete(y, i, axis=0)
        acc += KNN(k, X_tr, y_tr, X_te, y_te, distace_type)
    return acc/150
#%%
for m in tqdm(range(20)):
    k = m + 1
    accuracy_all[3][m] = KNN_iris2(k, y, X_lda, np.inf)
    pass
#%%
x = np.arange(1,21,1)
plt.figure(figsize=(8, 6), dpi=300) # 调整图形大小和分辨率
plt.title("KNN in iris")
plt.xlabel('k')
plt.ylabel('accuracy')
plt.xlim((1, 20))
plt.ylim((0.8, 1.0))
plt.plot(x, accuracy_all[0], 'r-o', label = "distance_L1")
plt.plot(x, accuracy_all[1], 'g-o', label = "distance_L2")
plt.plot(x, accuracy_all[2], 'b-o', label = "distance_Lmax")
plt.plot(x, accuracy_all[3], 'y-o', label="Fisher_iris")
plt.legend()
plt.savefig('result of KNN in iris.jpg')
plt.show()
#%%

```

3、Usps

```

#%%
import h5py
from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt
#%%

```

```

with h5py.File('usps.h5', 'r') as hf:
    train = hf.get('train')
    X_tr = train.get('data')[:]
    y_tr = train.get('target')[:]
    test = hf.get('test')
    X_te = test.get('data')[:]
    y_te = test.get('target')[:]

###
train
###
X_tr.shape, y_tr.shape, X_te.shape, y_te.shape
###
def KNN(k, X_tr, y_tr, X_te, y_te, distace_type):
    accuracy = 0
    for i in tqdm(range(X_te.shape[0])):
        distance = np.zeros((2, X_tr.shape[0]))
        clas = np.zeros(10)
        for j in range(X_tr.shape[0]):
            distance[1, j] = np.linalg.norm(X_te[i] -
X_tr[j], distace_type)
            distance[0, j] = y_tr[j]
        index = np.lexsort(distance)
        for l in range(k):
            clas[int(distance[0, index[l]])] +=1
        prediction = clas.argmax()
        if prediction == y_te[i]:
            accuracy +=1
        pass
    accuracy = accuracy/X_te.shape[0]
    return accuracy

###
accuracy_all = np.zeros((3,20))
###
for m in range(20):
    k = m + 1
    accuracy_all[0][m] = KNN(k, X_tr, y_tr, X_te, y_te, 1)
    accuracy_all[1][m] = KNN(k, X_tr, y_tr, X_te, y_te, 2)
    accuracy_all[2][m] = KNN(k, X_tr, y_tr, X_te, y_te, np.inf)

###
x = np.arange(1,21,1)
plt.figure(figsize=(8, 6), dpi=300) # 调整图形大小和分辨率
plt.title("KNN in usps", fontsize=16, fontweight='bold') # 设置标题
plt.xlabel('k')
plt.ylabel('accuracy')
plt.xlim((1, 20))
plt.ylim((0, 1.0))
plt.grid(True, linestyle='--', alpha=0.7) # 添加网格线

plt.plot(x, accuracy_all[0], 'r-o', label = "distance_L1")

```

```
plt.plot(x, accuracy_all[1], 'g-o', label = "distance_L2")
plt.plot(x, accuracy_all[2], 'b-o', label = "distance_Lmax")
plt.legend()
plt.savefig('result of KNN in usps, jpg')
plt.show()
```