上海大学

SHANGHAI UNIVERSITY

《Python 计算》期末综合实验

题　　目　课内主题 A 数据分析综合应用

学　　号　**20122998**

姓　　名　金怀德

日　　期　**2022 年 5 月 22 日**

一、实验目的与要求

1.1 实验要求

(1）以文本文件格式读入文件夹\dataanalysisMlabel\下的 MTL_*.dat,CMTL_*.dat, CEMTL_*.dat(*表示 White 或者 Male,选择其中一种处理即可)中数据，并且分别读入 numpy 数组 MTLLabel、CMTLLabel 或者 CEMTLLabel 中，对各个数组取绝对值后按照降序排序，并且记录数据元素排序前的下标号;

(2）以文本文件格式读入文件夹 ldataanalysisltrain\下的 MTL_*_train.dat(*表示 White 或者 Male，选择其中一种处理即可)中的数据，并且读入 numpy'矩阵 TrainSample 中，计算矩阵的行列数（该矩阵包含了 1000 个维数为 3304 的样本的观测值，第 1-500 个样本属于第一类，第 501-1000 个样本属于第二类，每类含 500 个样本顺序保存在文件中)。根据(1)中数组的排序(3 个数组分别实验)，选择最大的 k 个值(k 取 200，400，600，800，1000，...3304 维）对应的维度，把 TrainSample 中的 1000 个样本降维为 k 维，并保存到新的矩阵中 TrainSub 中;

(3）对于\dataanalysistest\下文件作和（2）相同的处理(其中数据矩阵包含了 800 个维数为 3304 的样本，第 1-400 个样本属于第一类，第 4014800 个样本属于第二类，每类含 400 个样本顺序保存在文件中);

(4）阅读和学习 knnexample 下面关于最近邻分类算法 Knn 的实现，用(2)中数据训练分类模型，用（3）中数据测试分类结果，统计错误率。

1.2 实验目的

通过 KNN 算法，即 K Nearest Neighbours 近邻算法实现数据分类，进行数据分析。通过实验，积累对于 python 中 numpy 数组编程的经验，深入 python 语法特性，并借助 KNN 实现作为机器学习的入门。最后使用 KNN 对于图像分类进行了尝试，并且与 CNN 神经网络进行对比学习。

二、实验环境

2.1 pytorch 中 Torchvision 模块主要功能介绍

pytorch 中的 Torchvision 函数是 torch 的一部分，主要要提供一些 torch 的辅助功能：

模块	模块主要作用
torchvision.datasets	包含比较常用的一些数据集的下载和加载功能，例如 MINST、CIFAR10 等数据集。
torchvision.models	封装了目前比较常见的经典的网络模型，可直接将预训练好的模型加载过来使用。
torchvision.transforms	图像裁剪、图像仿射变换、图像旋转、颜色调整、灰度变换等等，同时还支持各中操作一定概率的随机发生。
torchvision.utils	制作图像网格、图像保存以及图像外框绘制等

2.2 Numpy 库

本实验使用的模块功能函数

使用的函数及属性	函数功能或属性含义
numpy.size	数组元素总个数
numpy.shape	数组形状，用于得到矩阵每维的大小
numpy.power(x,y)	进行 numpy 数组或者简单数字的多维幂运算

注：和 numpy 不同，pytorch 中的 size 和 shape 都用于得出数组维度。

2.3 matplotlib 绘图库

使用 matplotlib 下的 pyplot 包进行数据可视化分析。

三、实验内容

3.1 第一到第三题

首先使用 create_LabelArg 函数完成第一题，将排好序的标签放入 argList 列表。在读取 TrainSample 和 TrainSample 后分别用 To_Sub 函数根据 argList 排序好的下标，选择特定维度完成降维。降维后对 TrainSample 使用 numpy.size 可以获取矩阵的行列数，

3.2 第四题

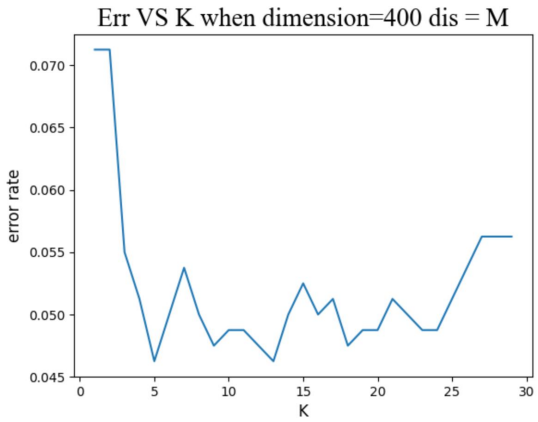
使用 KNN_classify_getErr 函数通过 TrainSub 训练集，对 TestSub 中的数据进行预测，通过 KNN 算法得出分类结果，并将测试分类结果和真实数据比较，进行统计后得出错误率。

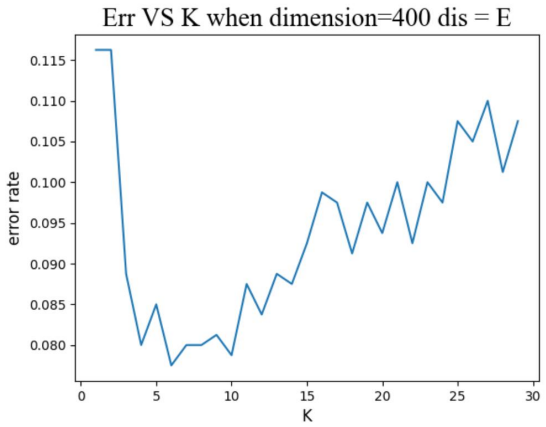
使用 KNN 时，当样本不平衡时容易导致结果错误，而对于各个样本数量差距大的数据集可以采用加权的方法（和该样本距离小的邻居权值大）来改进。但本样例中 knn 的同类样本数量同样为 500，所以不加权的误差率反而较低。

3.3 数据分析

主要分析了错误率和 K 以及维数 Dim 的关系。

① 以下是使用 MTL_Male.dat 作为标签，在 Errorrate_vs_k 函数中分别以曼哈顿（dis = M）以及欧氏距离（dis= E）作为度量方式，在降维维数为 400 时分别得到的数据分析图像。

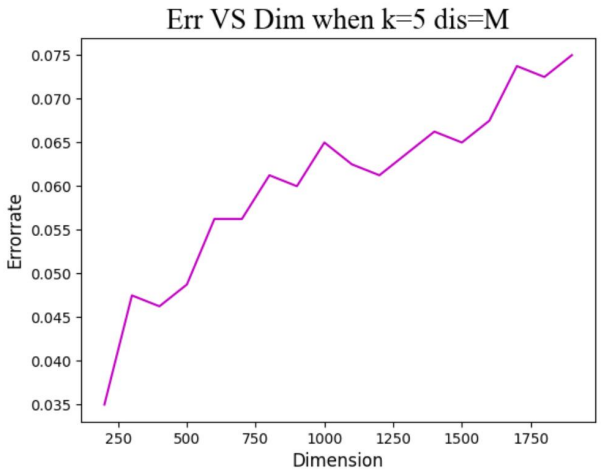


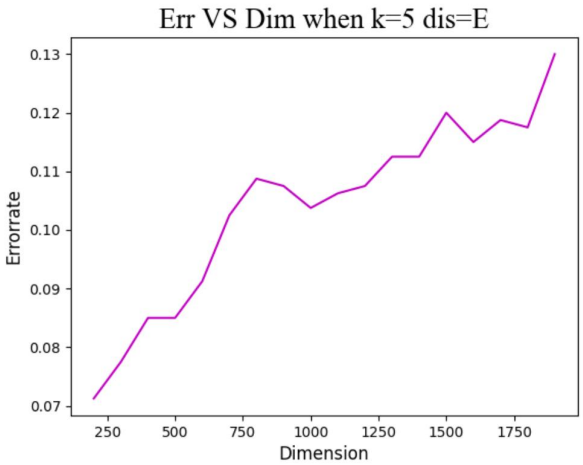


分析错误率和 K 关系图后，首先可以发现当 K 过小，样本预测容易受到个例影响，发生过拟合；K 过大，容易受到距离较远的特殊数据影响，发生欠拟合，所以 K 值选取不当错误率会提高，应该由问题自身和数据集大小决定最适合的值；并且通过图线可发现，错误率在 K 为奇数时为谷，为偶数时为锋，这是因为若把 K 设置为偶数，K 个邻别中 2 种类数量平局的现象很常见，于是产生了平局现象错判。这就说明，对于二分类问题，K 尽量选择奇数来避免平票。

总体上，以曼哈顿距离作为距离度量的预测错误率低于欧式距离，并且 K 增大时，欧式距离下的欠拟合现象也更加明显，区分能力有很大下降。

② 以下是使用 MTL_Male.dat 作为标签，在 Errorrate_vs_dim 函数中分别以曼哈顿（dis = M）以及欧氏距离（dis= E）作为度量方式，在 K 近邻值为 5 时分别得到的数据分析图像。



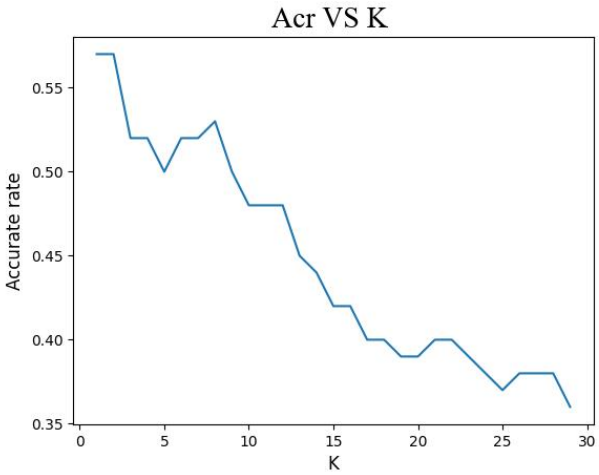


从 200 递增到 2000 维度的过程中，曼哈顿距离距离始终比欧式距离的预测错误率低。并且随着维数增加，两者都产生了维度灾难现象，即当维数过高时会使得样本在该维数的空间中的密度分布不断下降，使得分类变得越来越困难，过多的特征导致了过拟合现象，为了维持正确率需要的样本数量会增多（训练样本的数量和特征的维数呈指数关系）。在机器学习中，对于非线性决策边界的分类器，比如 KNN，神经网络，决策树等，也更容易发生过拟合。

4.3 KNN 图像识别

由于本学期学习了 pytorch 搭建的神经网络，我尝试使用 KNN，用与课题一致的分类方法对 MNIST 数据集进行图像识分类，并进行对比学习。

从数据集中将图片特征矩阵和标签分别在载入到变量中后，直接把每个 28*28 像素点图像用 reshape 展平成一个一维的向量，转化成了 TrainSub 矩阵相同的形式，每个样本包含的 28*28 个维度。以下时使用曼哈顿距作为决策距离方式，对不同的 k 作为近邻值，从验证集中每次抽取 500 个样本后进行预测，画出的精度曲线。



可以发现最高准确率仅百分之 50 左右，准确比较低。通过比较 KNN 和 CNN 的算法特性后，我认为其主要原因为 KNN 识别图像时，无法像深度学习卷积神经网络一样去提取图片的特定特征，比如鼻子耳朵等，它只会比较每个像素点之间相似度，作为距离排序，查看像素点之间接近程度后进行预测。据了解，KNN 算法可以对手写数字进行较高准确度的图像识别，但由于没有办法描述图像中特定的特征，一旦图像复杂则可能不太适用图像识别。可以得出的结论是，KNN 更适合对序列中为属性的数据，比如表格数据进行分类。

四、实验设计与实现

4.1 数据分析 A

① create_LabelArg 函数

对于数字文件，可使用 numpy.loadtxt('filename.txt')直接得取一个全为数字的 numpy 数组，使用参数 delimiter 作为分割符号,并将数据转为 np.float64。接着使用 abs 函数取绝对值后,以 np.argsort 以及切片倒置的方法获得数据元素排序前的下标号，并添加到列表中 argList 中。

② To_Sub 函数

在读取训练和验证样本 TrainSample 和 TestSample 后通过 to_sub 函数中，的列表推导式，根据 argLis 排序好的下标，在 train 和 test 中选择特定维度，进行降维操作。用参数 dim 控制降维的维数，使用列表推导式返回降维后的 TrainSub 和 TestSub 矩阵。

③ KNN_classify_getErr 函数

使用 KNN_classify_getErr 函数对样本进行预测计算出错误率。函数中传入 TrainSub 和 TestSub，参数 k 表示邻近点的个数，以及 dis 参数控制距离度量方式，规定只能传入 E 或 M，对应与欧式和曼哈顿距离，否则使用 assert 触发异常。

之后对于每个测试集样本，进行 K 近邻排序，用出现频率最高者和真实数据比较得到错误率。具体步骤：

1）对距离的关系进行排序后选距离最接近的 k 个样本的下标。具体操作是沿着 axis 为 1，也就是沿着 numpy 纵轴用距离度量公式计算测试数样本与各个训练数据间的距离。其中时使用了 power 函数进行 numpy 的开方和求次方操作。

2）对距离的使用 argsort 排序后结合切片，获取最接近的样本的 k 个数据的标签（类别）

3）使用字典 vote 统计这 k 个点所在所在类别的出现次数，统计完毕后用 max 返回前 k 个点中出现频率最高的类别作为预测分类，并且与真实分类进行比较，如果预测正确则 score 加 1

4）反复遍历 testsample 得到最后的 score 分数，接着经过计算得出 KNN 算法的错误率。

④ Errorrate_vs_k & Errorrate_VS_dim 函数

使用 matplotlib.pyplot 进行数据分析可视化。

4.2 KNN 进行图像分类

① KNN_classify_get 和 Accuracy_rate_vs_k 函数

使用 KNN 算法返回一个训练集中预判最大可能元素的列表，并分析准确度和 K 的关系，实现方式和数据分析解题方法相似。

② 对数据集图像的处理

使用 torchvision.datasets 将 MNIST 数据集保存到 data 根目录，并且使用 train 参数标注是训练集还是测试集。

规定每个 batch 即批次中的图像数为 00，然后使用 torch.utils.data.Dataloader，通过 batch_size 和 shuffle 参数将数据打包成每批 100 张的乱序图片集。

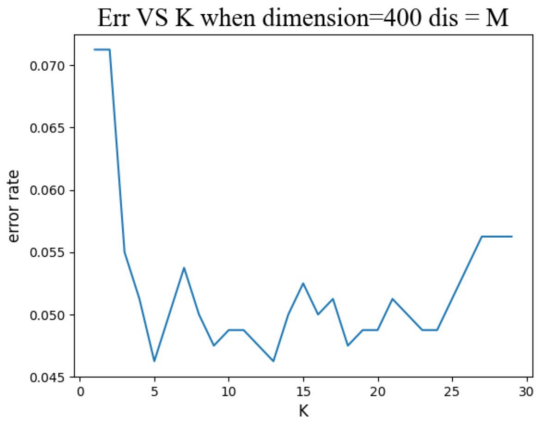
在从数据集中将图片特征矩阵和标签分别载入到变量中后转化为 numpy 数组后，会针对图像特征矩阵，把每个 28*28 像素点的图像使用 reshape 展平成一个一维的向量，把样本转化成了 TrainSub 矩阵相同的形式，每个样本包含的 28*28 个维度，共 train.shape[0]个样本。

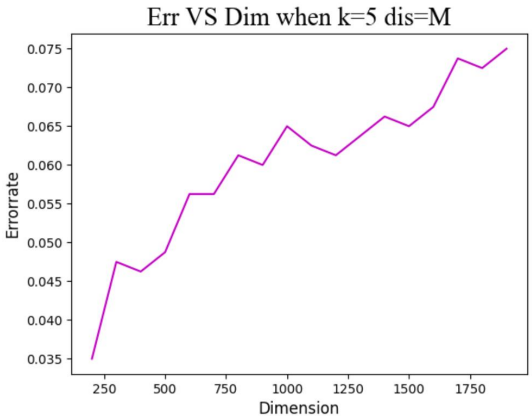
对验证集进行切片选取其中 500 张图片进行预测。

五、测试用例

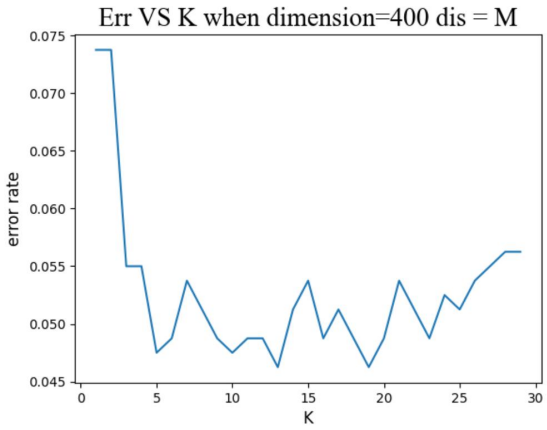
数据分析 A 中测试用例为① MTL_Male.dat ② CMTL_Male.dat ③ CEMTL_ Male.dat 。测试出的数据图像分别如下（使用曼哈顿数据）

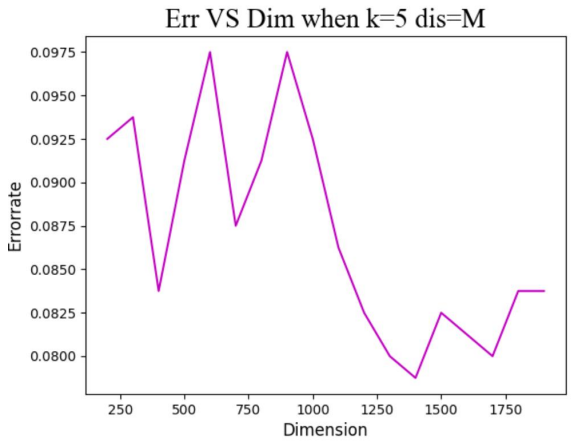
① MTL_Male.dat 作为测试数据



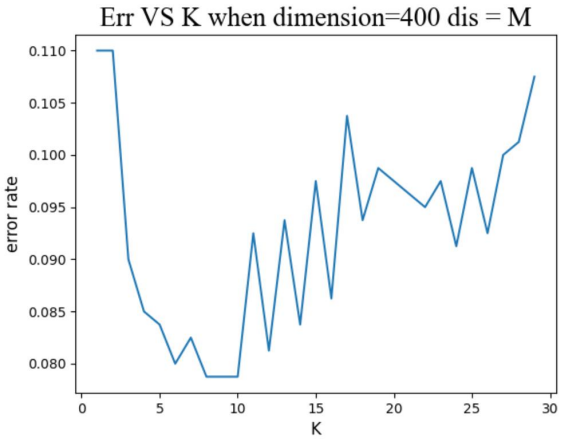


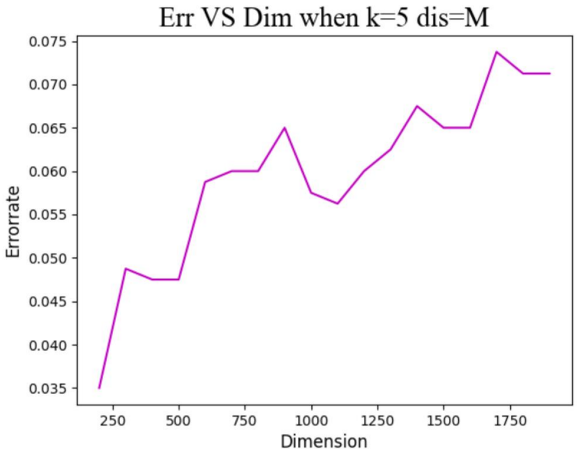
② CMTL_Male.dat 作为测试数据





③ CEMTL_Male.dat 作为测试数据





总结来说，除了对于 CMTL_Male.dat 数据较 MTL 和 CEMTL 稍有偏差外，结果大致符合预期。使用 White 作为测试时的错误率约为 Male 的两倍，没有多做测试。

六、收获与体会

本次实验实现机器学习的入门 KNN 算法，和自学 pytorch 搭建神经网络时感受比较不同：编写 KNN 算法时，并没有涉及很多难懂的库，相较之下 pytorch 搭建的更多是比较框架性的代码，而本实验更像时在构建底层代码，去实现原理，需要更多对原理的了解，以及了解影响 KNN 训练的各种因素，比如 K，维度，距离度量方式对预测准确度的影响。

总的来说，这次的作业让我了解了 python 中数据分析的方法，并深入学习了 KNN 算法，进入了机器学习的大门。机器学习是一门很深的课程，需要我们不断去理解、探索，去不断地实践。而回顾这短促但充实的一个学期，我通过马尔科夫链，排序，字符串操作，数据分析等等主题的实验中认识了 python 语言特性，并且和 C++语言进行了对比学习。从 python 语法出发，在打下牢固基础的同时，结合数据结构，去更多体会 python 语言特性，积累经验。除此之外。老师在将同学们领进 python 学习的大门后，也提醒了我们学习编程语言时还应该对哪些潜在问题，比如字符串的操作，内存的机制等细节，等有一定的积累意识。我想这就是这门课程的精髓。

七、核心代码

7.1 KNN 数据分析

① create_LabelArg 函数

```
def create_LabelArg(labels):
    """ 读取数据元素排序前的下标号，结果放入列表"""
    arglist = []
    for i in range(len(labels)):
        path = "../dataanalysis/label/" + labels[i]  # 文件路径拼接
        datalabel = np.loadtxt(path, dtype=np.float64, delimiter="\n")
        labelargs = np.argsort(abs(datalabel))[::-1]  # 取绝对值， argsort 记录排序前的下标号
        arglist.append(labelargs)
    return arglist
```

② To_Sub 函数

```
def To_Sub(train, arglist, d):
    """根据 argList 选择 dim 个维度降维"""
    return np.array([train[i][arglist[j]] for i in range(train.shape[0]) for j in range(d)]).reshape(train.shape[0], d)
```

③ KNN_classify_getErr 函数

```
def KNN_classify_getErr(train, test, k, dis):  # di 为决策距离
    """ 根据 k 个近邻点进行分类，返回预测错误率 """
    assert dis == 'E' or dis == 'M', '距离决策方式必须为 E 或 M，E 代表欧氏距离，M 代表曼哈顿距离'
    test_num = test.shape[0]
    train_num = train.shape[0]
    score = 0
    if dis == 'E':  # 欧氏距离
        for i in range(test_num):
            dist = np.power((np.power(train[:] - test[i], 2)).sum(axis=1), 0.5)
            topK = np.argsort(dist)[:k]  # 选距离最接近的 k 个样本
            votes = {}
            for m in topK:
```

```
            label = classify(m, train_num)
            votes.setdefault(label, 0)
            votes[label] += 1
        if classify(i, test_num) == max(votes, key=votes.get):
            score += 1
    elif dis == 'M':  # 曼哈顿距离
        for i in range(test_num):
            dist = np.sum(np.abs(train[:, :-1] - test[i, :-1]), axis=1)  # 曼哈顿
距离
            topK = np.argsort(dist)[:k]  # 选距离最接近的 k 个样本
            votes = {}
            for m in topK:
                label = classify(m, train_num)
                votes.setdefault(label, 0)
                votes[label] += 1
            if classify(i, test_num) == max(votes, key=votes.get):
                score += 1
    err = 1 - score / test_num
    return err
def classify(index, num):  # 只有两个分类
    return index < num / 2
```

④ 错误率和 K 值的分析

```
def Errorrate_vs_k(trainsub, testsub, dis):
    """ 比较错误率和 K 的关系"""
    x = []
    y = []
    for i in range(1, 30):  # 一般 K 值取 1-20，此处测试 1 到 30 邻接点
        x.append(i)
        err = KNN_classify_getErr(trainsub, testsub, i, dis)
        y.append(err)
    plt.title('Err VS K when dimension={} dis = {}'.format(testsub.shape[1], dis),
fontsize=20,
              fontname='Times New Roman')
    plt.xlabel("K", fontsize="large")
    plt.ylabel("error rate", fontsize="large")
    plt.plot(x, y)
    plt.show()
```

7.2 KNN 运用到图像分类

①下载数据集后对于训练集做的操作

```
# 分批次（batch）训练
batch_size = 100
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)  # 将数据打乱

# 预处理
train = train_loader.dataset.train_data.numpy()  # 获取训练集图像，转为 numpy 数组
train = train.reshape(train.shape[0], 28 * 28)
# 需要 reshape 之后才能放入 knn 分类器,图像转成一个 28*28 的维度 train.shape[0]个样本的矩阵
train_label = train_loader.dataset.train_labels.numpy()  # 训练集获取标签
```

八、课堂研讨

录制的研讨视频截图 ：