实验二：KNN 近邻法

1 问题描述

使用 Fisher 线性判别和 KNN 处理分类问题，并在 MNIST 数据集上验证算法的有效性。本实验需完成两个任务：

1、在不降维的情况下，列表展示不同 K 值下的 KNN 算法精度。

2、在 LDA 降维的情况下，使用最近邻算法计算分类精度，画出精度关于纬度的折线图，并与 1 中的结果进行比对。

2 数据集说明

MNIST 数据集是一组由美国高中生和人口调查局员工手写的 70000 个数字的图片。MNIST 中每个样本都是一张长 28、宽 28 的灰度图片，其中包含一个 0-9 的数字，它为机器学习的初学者提供了一个练手的机会，可以在真实的数据上用学到的算法来解决问题。

下表为部分 MNIST 数据展示：

图 1: 部分 MNIST 数据展示

3 近邻法原理分析

3.1 KNN 算法模型

K 近邻算法是指给定一个未知标签的样本，在已有的训练样本集中，找到与该待分类的样本距离最邻近的 k 个训练样本，随后根据这 k 个训练样本的类别，通过一定的决策规则决定该未知样本的类别。具体流程如下：

设有 N 个已知样本分属于 c 个类 $w_i, i = 1, 2, ..., c$, 考察新样本 x 在这些样本中的前 k 个近邻，设其中有 k_i 个属于 w_i，则 w_i 类的判别函数就是

$$g_i(x) = k_i, i = 1, \ldots, c$$

决策规则是

$$\text{若 } g_k(x) = \max_{i=1,\ldots,c} g_i(x)\text{，则 } x \in \omega_k$$

当 k=1 时，k 近邻又称为最近邻算法。

3.2 距离度量

两个样本的特征距离反映了彼此之间的相似程度。常见的距离度量有欧式距离、具有一般性的 L_p 距离和曼哈顿距离等。

假设样本空间 X 属于 n 维实数向量空间 R^n, 则其 L_p 距离定义为：

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

其中，p 大于等于 1。当 p=2 时，称为欧式距离：

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

当 p=1 时，称为曼哈顿距离：

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

本实验中，选用的是欧氏距离。

4 数据预处理

本实验中，MNIST 数据量为 70000*28*28，若全部使用，则会导致 KNN 运行效率缓慢。

因此，本实验随机选取 MNIST 数据集中 10000 条数据作为样本空间，并对数据进行标准化。数据标准化能提升 KNN 算法的精度，并且提升算法的运行效率。

数据标准化的公式如下：

$$x^*=\frac{x-\mu}{\sigma}$$

数据标准化之后，就会服从为均值为 0，方差为 1 的正态分布。

5 实验结果

5.1 不同 k 值下的 KNN 分类准确率

本次实验对 10000 条 MNIST 数据以 80% 作训练集,20% 作训练集，计算 10 次准确率后求取平均值。选取不同 k 值，得到的准确率如下表所示：

表 1: 不同 k 值下的 KNN 分类准确率

k 值	准确率%	k 值	准确率%	k 值	准确率%	k 值	准确率%	k 值	准确率%
1	89.2	11	90.7	21	90.25	31	89.75	41	89.65
2	88.7	12	90.6	22	90.4	32	89.8	42	89.6
3	90.75	13	90.75	23	90.25	33	89.75	43	89.55
4	90.5	14	90.8	24	90.25	34	89.9	44	89.6
5	90.4	15	90.75	25	90.05	35	89.85	45	89.6
6	90.35	16	90.55	26	90.15	36	89.7	46	89.75
7	90.65	17	90.55	27	89.95	37	89.75	47	89.6
8	90.55	18	90.3	28	90	38	89.65	48	89.55
9	90.45	19	90.3	29	89.95	39	89.65	49	89.65
10	91.55	20	90.35	30	89.95	40	89.7	50	89.5

为了让结果更清晰，我们将表格中的数据绘制成图像如下图所示：

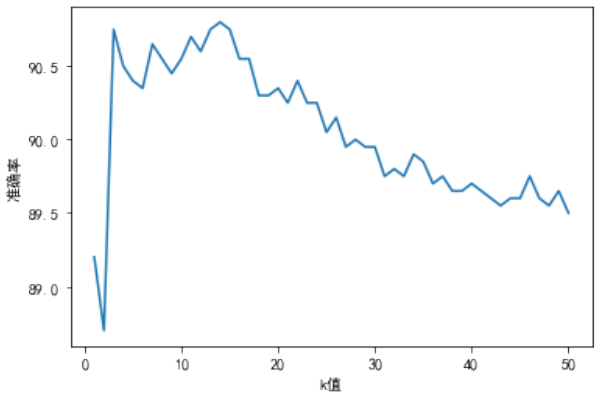


图 2: 不同 k 值下的 KNN 分类准确率

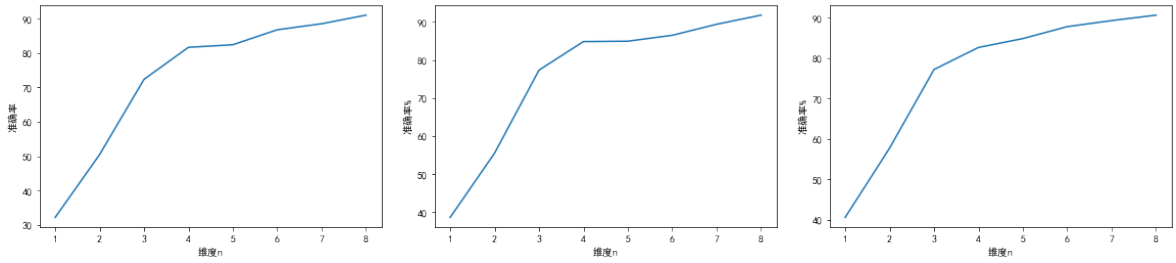
从表中可以发现当 k 为 1，2，3 时，KNN 的分类准确率变化波动较大，说明不同类样本的特征分布较为密集，导致 k 值较小时，对分类准确率影响很大。当 k 不断增大时，KNN 的分类准确率呈现减小的趋势。这从理论上也容易理解，当 k 不断接近样本总量时，不同类之间的分类区别不断减小，从而导致分类准确率的降低。

从图中还可以发现，当 k 取值在 13 左右时，分类准确率达到最高，最高准确率为 90.75%。

5.2 LDA 降维下的 KNN 分类准确率

MNIST 数据集每个样本是 28*28 的图片，数据维度为 28*28=784，但由于 LDA 本身的限制，最大降维数需保证小于类别数-1。MNIST 数据为 10 分类的数据，因此，可以使用 LDA 将原数据降到 1～8 维。

本实验分别选取 k 值为 3，13，23，绘制准确率随维度的变化图如下所示：



(a) k=3　　(b) k=13　　(c) k=23

图 3: LDA 降维准确率随维度的变化图

从图中可以发现，当 k 选取不同的取值时，准确率随维度的关系差别不大，当维度越大时，KNN 的分类准确率也越高。这很容易理解，由于 LDA 通过数据降维，必然损失了部分数据，当维度越高时，数据的损失量越小，分类准确率自然越高。当维度 n=8 时，分类准确率接近降维之前的准确率（此时分类准确率为 90.6%, 降维前的最高准确率为 90.75%），说明 LDA 降维能够在不减小分类准确率的情况下，降低数据的维度，从而提升计算效率。

从图中还可以发现，当维度 n 不断增大时，分类准确率增大的趋势不断减小，n=4 为其中的一个明显拐点。因此，对 MNIST 数据集的 LDA 降维，维度选择设置成 4 最为合适，其能在分类准确率和计算效率上做到平衡。

6 程序代码

本实验采用的开发环境为 Jupyter lab，这里仅展现核心代码，完整代码和过程见” 程序.ipynb”

```
# 数据集的导入
mnist = fetch_openml("mnist_784")
X, y = mnist['data'], mnist['target'] # X:data, y:label
print(X.shape, y)# 70000 70000

#标准化特征
scaler = StandardScaler()

shuffle_index = np.random.permutation(60000)  # 随机排列一个序列，返回一个排列的序列。
X1, y1 = X[shuffle_index[:10000]], y[shuffle_index[:10000]]
# 为了保证学习的效率，只随机取了10000个数据作为训练集进行训练

X_standardized = scaler.fit_transform(X1)

def data_division():
# 随机划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X_standardized, y1, test_size=0.2)
#     print(X_train.shape, X_test.shape)
def knn(k):
# 测试用，记录算法的时间
# begin_t = t.time()
#创建一个有5个邻居的KNN分类器对象
knn = KNeighborsClassifier(n_neighbors=k, n_jobs=-1)

#训练模型
model = knn.fit(X_train, y_train)

#预测数据
predictions = model.predict(X_test)

#测试准确率
accuracy = metrics.accuracy_score(y_test, predictions)
#     print ("k=",k)
#     print ('accuracy: %.2f%%'%(100*accuracy))
return 100*accuracy
# print("Total time: {:.2f}s".format(t.time()-begin_t))

# 使用Fisher进行降维
# 注：LDA最大降维数<分类数-1
# minsit为10分类，因此维度数可以取1-8
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
42          def mylda(X, y, demension):
43          lda = LDA(n_components=demension)
44          lda.fit(X, y)
45          result_x = lda.transform(X)
46          return result_x
47
48          acc = [] # acc用来存放不同k值下的准确率数值
49          for d in range(1,9):
50          lda_x = mylda(X_standardized,y1,d)
51          X_train, X_test, y_train, y_test = train_test_split(lda_x, y1,
                test_size=0.2)
52          # 选取不同k值的情况
53          k = 3
54          # k = 13
55          # k = 23
56          accuracy = knn(k)
57          acc.append(accuracy)
58          print ("k=",k)
59          print ('accuracy: %.2f%%'%(accuracy))
60
61          # 绘制不同k值下的准确率图线
62          plt.plot(np.arange(1, 9), acc)
63          # 解决中文显示问题
64          plt.rcParams['font.sans-serif'] = ['SimHei']
65          plt.rcParams['axes.unicode_minus'] = False
66          plt.xlabel("维度n")
67          plt.ylabel("准确率%")
68          # plt.savefig('./demension.jpg')
69          plt.show()
```