题目　Dimensionality Reduction

姓名 骆克云 学号 MG1633052 邮箱 streamer.ky@foxmail.com 联系方式18115128082

(南京大学 计算机科学与技术系, 南京　210093)

1　实现细节

1.1　　读取文件，返回训练集、测试集的数据和标签（projectutil.py）

基本的 Python 读写文件：将每一行数据按逗号分隔，最后一个作为标签，前面的作为数据

```
def get_reduction_dataset(name="sonar", types="train"):
 """
 获取数据集
 :param name: 数据名称：sonar/splice
 :param types:数据类型：train/test
 :return: 数据，标签
 """
 file_train = get_dataset_dir() + name + "-train.txt"
 file_test = get_dataset_dir() + name + "-test.txt"
 file = file_train if types == "train" else file_test

 data = []
 label = []

 with open(file, "r", encoding="utf-8") as f:
    for line in f:
        line = line.split(",")
        data.append(line[:-1])
        label.append(line[-1].strip("\n"))
 return data, label
```

1.2　　knn算法实现(assignment2/knn.py)

该文件主要将本次任务所使用的近邻算法独立出来实现共用。本文件实现了两个方法 knn 和 onenn。knn 可以实现 K 近邻，返回测试集的标签；onenn 则由于是特殊情况，中间结果不需要排序，速度快，故后面 PCA、SVD、ISOMAP 默认都是采用 onenn,但在 PCA 算法中测试了 knn 的正确性。

1.2.1　knn

对于测试集中的每一个点，寻找其在训练集中的 K 个近邻，并且将这 K 个近邻的标签累加，取最大的那个标签作为测试集中该点的预测值。

1.2.2　onenn

基本同 knn，不过由于是最近邻，取测试集中的点和训练集中与其最近的点的标签作为其标签。另外考

虑到通用性，将最终的结果输出也集成在这个方法中。

```
def onenn(projections, labels, name, algorithm):
    """
    最近邻计算
    :param projections: projection_trains, projection_tests 元组
    :param labels: train_label, test_label
    :param name: 文件名
    :param algorithm: 使用的算法
    :return: 无
    """

    print("====算法：%s ==== \n%s 文件测试结果：" % (algorithm, name))
    projection_trains, projection_tests = projections
    train_label, test_label = labels

    # 分三种情况(k=10,20,30)计算正确率
    for i in range(3):

        row_test, _ = projection_tests[i].shape
        row_train, _ = projection_trains[i].shape
        count = 0
        for row_index in range(row_test):
            dist_row = []
            for rowtrain in range(row_train):
                # 欧式距离
                dist_row.append(np.linalg.norm(projection_tests[i][row_index] - projection_trains[i][rowtrain]))
            # 取距离最短的点的标签
            result = train_label[dist_row.index(min(dist_row))]

            if result == test_label[row_index]:
                count += 1

        # 输出结果
        if i == 0:
            print("k=10,正确率：{0}/{1}={2}%".format(count, row_test, 1.0 * count / row_test * 100))

        if i == 1:
            print("k=20,正确率：{0}/{1}={2}%".format(count, row_test, 1.0 * count / row_test * 100))

        if i == 2:
            print("k=30,正确率：{0}/{1}={2}%".format(count, row_test, 1.0 * count / row_test * 100))
```

1.3 PCA算法（assignment2/pca.py）

将高维数据投影到低维空间，使投影后的数据方差最大化或最小化投影误差。

数据为 N*M，M 个特征，采样 N 次

基本步骤：

1） 计算样本数据中每一维的均值，然后让每个数据减去每一维的均值，去中心化

2） 对去中心化后的数据求解协方差矩阵，对协方差矩阵进行特征值分解

3） 得到特征值和特征向量后，对特征值从大到小排序，前 k 个特征值索引所对应的特征向量列组成投影向量

4） 投影训练数据，测试数据

5） 运行 onenn，计算结果

伪代码：

1. self.train, self .train_label, self .test,self.test_label = get_reduction_dataset(filename)
2. meanVal=np.mean(self.train, axis=0), return self.train - self.meanVal
3. cov_mat, eig_values, eig_vectors = np.linalg.eig(cov_mat)
4. Sort eig_values,select the k-largest: eig_values_index_sort = np.argsort(-eig_values), k_index = eig_values_index_sort[:k], projection_matrix.append(eig_vectors[:, k_index]), return projection_matrix
5. Projection: projection_trains.append(self.train.dot(matrix)), projection_tests.append(self.test.dot(matrix))
6. One-nn: onenn(projections, labels, self.name, "PCA")

```
train, train_label, test, test_label = get_reduction_dataset(name)
meanVal=np.mean(train, axis=0)
train_zero = train - meanVal
cov_mat = np.cov(train_zero, rowvar=False)
eig_values, eig_vectors = np.linalg.eig(cov_mat)
eig_values_index_sort = np.argsort(-eig_values)
projection_matrix = []
for k in [10, 20, 30]:
    k_index = eig_values_index_sort[:k]
    projection_matrix.append(eig_vectors[:, k_index])

projection_trains = []
projection_tests = []
for matrix in projection_matrix:
    projection_trains.append(train.dot(matrix))
    projection_tests.append(test.dot(matrix))

projections = (projection_trains,projection_tests)
labels = (train_label, test_label)
onenn(projections, labels, name, "PCA")
```

1.4 SVD算法（assignment2/svd.py）

奇异值分解可以舍弃较小的奇异值(特征值)对应的特征向量实现降维。。

数据为 N*M，M 个特征，采样 N 次

基本步骤：

1) 将训练数据转置，进行奇异值分解，得到 u, sigma, vt，然后进行奇异值的对角化，按照 k=10,20,30 保留分解结果。
2) 投影：对于训练集，使用 Sigma 点积 Vt 后转置即可得到投影后的数据，对于训练集，使用 U.T 点积测试集的转置，再将结果转置
3) 将投影后的训练集，测试集，标签传入 onenn 算法得出结果

（伪）代码如下：

```
train, train_label, test, test_label = get_reduction_dataset(name)
u, sigma, vt = np.linalg.svd(train.T)

S = np.zeros(self.train.shape)
sig_shape = sigma.shape[0]
S[:sig_shape, :sig_shape] = np.diag(sigma)

projection_trains = []
projection_tests = []
for k in [10, 20, 30]:
    U = u[:, :k]
    Sigma = S[:k, :k]
    Vt = vt[:k]
    projection_trains.append(np.dot(Sigma, Vt).T)
    projection_tests.append(np.dot(U.T, self.test.T).T)

projections = (projection_trains,projection_tests)
labels = (train_label, test_label)
onenn(projections, labels, name, "SVD")
```

1.5 ISOMAP算法(assignment2/isomap.py)

ISOMAP 等距映设算法使用测地距离解决流式空间欧氏距离不准确的问题，使得降维后的点两两之间的距离和原始空间点的距离尽可能小。

基本步骤：

1) 获取数据集：将训练集和测试集融合，得到总数据。
2) 计算欧式距离：对总数据集计算两两之间的欧式距离矩阵。
3) 构建图模型：初始化图矩阵为无穷大，选取每个点的 K(k_nn+1,包括自身)个近邻，其距离作为图矩阵的距离，并且将图矩阵对称化，即 graph_matrix[i, j] = graph_matrix[j, i] =min(graph_matrix[i, j] ,graph_matrix[j, i])。
4) 计算任意两点间的距离：使用 Floyd 算法，并利用 numpy 优化简化计算：graph_matrix = np.minimum(graph_matrix, np.add.outer(graph_matrix[:, i], graph_matrix[i, :]))。

5） 计算 MDS：首先要消除不连通分量，否则无法进行特征值分解，即图矩阵中存在的无穷大值，无穷大值表示这两个点之间没有连接，很有可能是孤立点。具体做法是找到图矩阵中最大的连通行，保存连通的索引和非连通索引，**将图矩阵中所有的非连通索引对应的行和列去除，得到连通的图矩阵**：conn_graph_matrix，对连通的图矩阵计算距离矩阵 D= -0.5 * conn_graph_matrix**2,中心矩阵 H = np.eye(N) - np.ones((N, N)) / N，进而计算出映设矩阵 S = H.dot(D).dot(H)，对其进行对称矩阵特征值分解，得到特征值和特征向量，对特征值和特征向量进行从大到小排序，返回特征值，特征向量，非连通索引。

6） 投影：首先判断非连通索引是否为空，如果不为空，判断其在训练集和测试集中的位置，在训练集和测试集中剔除这些数值和标签，得到新的数据集。在新的数据集中，令 U 为特征向量的前 k 列，delta 为前 k 列特征值对角化后的矩阵，则总投影矩阵为 Y = U.dot(delta),训练集投影为 Y[:train_len],测试集投影为 Y[train_len:]。

7） 运行 onenn，计算结果。

代码含注释，顺序执行即可：

```
class ISOMAP:
  """ISOMAP 流式学习降维算法"""
  def __init__(self, name="sonar", k_nn=4):
    """初始化：输入文件名：sonar/splice"""
    data_train, label_train = get_reduction_dataset(name, "train")
    data_test, label_test = get_reduction_dataset(name, "test")
    self.name = name
    self.k_nn = k_nn
    self.train = np.array(data_train).astype(float)
    self.train_label = np.array(label_train).astype(int)
    self.test = np.array(data_test).astype(float)
    self.test_label = np.array(label_test).astype(int)
    self.data = np.append(self.train, self.test, axis=0)
    self.length = len(self.data)

  def distance_matrix(self):
    """计算原始点之间的距离矩阵"""
    dis_points = np.zeros((self.length, self.length))
    for i, line_i in enumerate(self.data):
      for j, line_j in enumerate(self.data):
        dis_points[i, j] = np.linalg.norm(line_i-line_j) if i != j else 0
    return dis_points

  def build_graph(self):
    """构建图模型：每个点和其周围最近的 k 个点有连接"""
    # 点到自身的距离为 0,故 k+=1
    k = self.k_nn + 1
    N = self.length
    dis_matrix = self.distance_matrix()
```

```
    # 图矩阵,初始化为无穷大
    graph_matrix = np.full((N, N), fill_value=np.inf, dtype=float)

    # 计算 K 近邻,并且对称化
    for i in range(N):
        index_sort = np.argsort(dis_matrix[i])
        for j in index_sort[:k]:
            graph_matrix[i, j] = dis_matrix[i, j]

    return np.minimum(graph_matrix, graph_matrix.T)

def shortest_path(self, algorithm="floyd"):
    """
    floyd 任意两点间距离
    采用 numpy 外积运算
    """
    graph_matrix = self.build_graph()

    for i in range(self.length):
        graph_matrix = np.minimum(graph_matrix, np.add.outer(graph_matrix[:, i], graph_matrix[i, :]))

    return graph_matrix

def mds(self):
    """
    经典 MDS 算法：
    S= -1/2*H*D*H
    H = I - 11^T/N
    :return: eig_vectors:特征向量， eig_values：特征值
    """
    graph_matrix = self.shortest_path()

    # 消除不连接分量
    isINF = np.less(graph_matrix,np.inf)
    MAX = [-np.inf, -1]
    connected = []
    for i in range(len(graph_matrix)):
        conn = np.nonzero(isINF[i])[0]
        connected.append(conn)
        if MAX[0] < len(conn):
            MAX = [len(conn), i]
    connected_index = connected[MAX[1]]
    # 非连通量
```

```
    erase_value = [index for index, _ in enumerate(connected_index) if index not in connected_index]
    # 全连通距离矩阵
    conn_graph_matrix = np.take(np.take(graph_matrix, connected_index, 0), connected_index, 1)

    #print(graph_matrix[0])
    N = len(conn_graph_matrix)#self.length

    D = -0.5 * conn_graph_matrix**2
    # 中心矩阵
    H = np.eye(N) - np.ones((N, N)) / N
    S = H.dot(D).dot(H)
    # 对称矩阵特征值分解
    eig_values, eig_vectors = np.linalg.eigh(S)
    # 对特征值，特征向量排序
    indexes = np.argsort(- eig_values)
    eig_values = eig_values[indexes]
    eig_vectors = eig_vectors[:, indexes]

    return eig_values, eig_vectors, erase_value

def projection(self):
    """投影训练集，测试集"""
    projection_trains = []
    projection_tests = []
    eig_values, eig_vectors, erase_value = self.mds()
    # 仅使用正的特征值计算坐标
    indexes, = np.where(eig_values > 0)

    if len(indexes) < 30:
        print("!!!正的特征值不足三十个，无法进行纵向比较!!!")
        return -1, -1
    # 判断非连通量，即孤立噪声点所在的数据集
    train_len = len(self.train)
    erase_train = []
    erase_test = []
    if len(erase_value) != 0:
        for i in erase_value:
            if i < train_len:
                erase_train.append(i)
            else:
                erase_test.append(train_len-i)
        train_len -= len(erase_train)
        print("===ISOMAP  文件:%s:出现孤立点===" % self.name)
```

```
        print("训练集位置:%s,测试集位置:%s" % (str(erase_train), str(erase_test)))

    for k in [10, 20, 30]:
        U = eig_vectors[:, indexes[:k]]
        delta = np.diag(np.sqrt(eig_values[indexes[:k]]))
        Y = U.dot(delta)
        projection_trains.append(Y[:train_len])
        projection_tests.append(Y[train_len:])

    return projection_trains, projection_tests, erase_train, erase_test

  def run(self):
    project = self.projection()
    projections = project[:2]
    erase_train, erase_test = project[2:]
    if projections[0] != -1:
        train_label = np.delete(self.train_label, erase_train)
        test_label = np.delete(self.test_label, erase_test)
        labels = (train_label, test_label)
        onenn(projections, labels, self.name, "ISOMAP-k"+str(self.k_nn))
```

1.6 运行

对于每个文件（"sonar", "splice"），分别运行 PCA，SVD，ISOMAP(k_nn=4,6,8,10)。

```
def run():
  for file in ["sonar", "splice"]:
    pca = PCA(file)
    pca.run()
    svd = SVD(file)
    svd.run()

    isomap = ISOMAP(file, 4)
    isomap.run()
    isomap = ISOMAP(file, 6)
    isomap.run()
    isomap = ISOMAP(file, 8)
    isomap.run()
    isomap = ISOMAP(file, 10)
    isomap.run()
```

2 结果

2.1 实验设置

数据来源：BinaryDatasets,包含如下文件：sonar-test.txt，sonar-train.txt，splice-test.txt，splice-train.txt

数据预处理：projectutil.py 中 get_reduction_dataset(name="sonar", types="train")可给定文件名返回数据及标签。

2.2 实验结果

1. 正确率比较

	PCA	SVD	ISOMAP-k4	ISOMAP-k6	ISOMAP-k8	ISOMAP-k10
K=10(sonar)	58.25%	59.22%	46.60%	41.75%	45.63%	52.43%
K=20(sonar)	56.31%	58.25%	54.37%	41.75%	46.60%	51.46%
K=30(sonar)	56.31%	56.31%	54.37%	43.69%	45.63%	54.37%
K=10(splice)	75.82%	75.86%	68.97%	67.31%	68.41%	68.83%
K=20(splice)	76.28%	76.41%	68.82%	69.89%	69.84%	69.89%
K=30(splice)	73.56%	73.56%	68.46%	72.06%	70.16%	70.02%

2. 使用 sklearn 对比

本实验中我同时使用了 sklearn 中的相关算法做了对比，差别不大，结果如下：

	PCA	SVD	ISOMAP-k4	ISOMAP-k6	ISOMAP-k8	ISOMAP-k10
K=10(sonar)	58.25%	59.22%	45.63%	41.75%	45.63%	52.43%
K=20(sonar)	56.31%	58.25%	56.31%	41.75%	46.60%	51.46%
K=30(sonar)	56.31%	56.31%	55.34%	43.69%	45.63%	54.37%
K=10(splice)	75.68%	75.31%	67.95%	67.90%	67.72%	69.19%
K=20(splice)	74.25%	76.05%	69.28%	68.83%	68.92%	70.21%
K=30(splice)	72.64%	73.20%	68.55%	69.75%	70.11%	69.61%

3. ISOMAP 说明

ISOMAP-k4 “sonar”文件中出现了孤立点（训练集位置:[83, 84, 85, 86, 87, 88],测试集位置:[]），无法进行对称矩阵特征值分解，故投影时舍弃了这几个值以及对应的标签。

4. 结果反思

利用降维算法与近邻算法进行分类的结果好坏与数据集密切相关。

在本次数据集中 SVD 分解总体好于另两个算法。

ISOMAP 中出现异常点直接删除了，本题中刚好出现在训练集中，对测试集影响不大，若出现在测试集中，则未找到比较好的度量方法。并且该算法运行缓慢，有待优化。

注意：

1. 最终提交的报告最好保存为 pdf 格式
2. 压缩格式为 zip 格式，请勿使用需要安装特定软件才能打开的压缩方式
3. 作业的文件夹目录请按照网页要求，代码、结果放在不同子文件夹中。作业网页上给出的数据不需要再次提交