E6_22336216

人工智能实验

中山大学计算机学院

人工智能

本科生实验报告

（2023学年春季学期）

课程名称：Artificial Intelligence

教学班级	DCS315	专业（方向）	计算机科学与技术（系统结构）
学号	22336216	姓名	陶宇卓

1 实验题目

实验六：KNN & 朴素贝叶斯

一、算法原理

- **算法原理**

K最近邻（K-Nearest Neighbors，简称KNN）是一种简单而直观的监督学习算法，用于分类和回归问题。其基本原理是基于特征空间中最近邻居的投票进行分类或预测。

算法原理如下：

- **数据表示：**首先，将训练数据集中的每个样本表示为一个在特征空间中的点，特征空间中的每个维度代表一个特征。例如，如果有两个特征，则每个样本将被表示为特征空间中的一个二维点。
- **距离度量：**定义一个距离度量方法，通常使用欧氏距离或曼哈顿距离。这个度量方法用于计算样本之间的距离，以便找到最近的邻居。
- **选择邻居：**对于给定的未知样本，计算它与训练集中每个样本的距离，并选取与其最近的K个样本作为邻居。
- **投票决策：**对于分类问题，采用多数投票的方法，将K个最近邻居中出现最频繁的类别标签作为未知样本的类别标签。对于回归问题，通常采用平均值法，将K个最近邻居的输出的平均值作为未知样本的预测值。
- **分类或预测：**根据投票结果或平均值，对未知样本进行分类或预测。
- **参数选择：**选择合适的K值，通常使用交叉验证等方法来确定最优的K值。

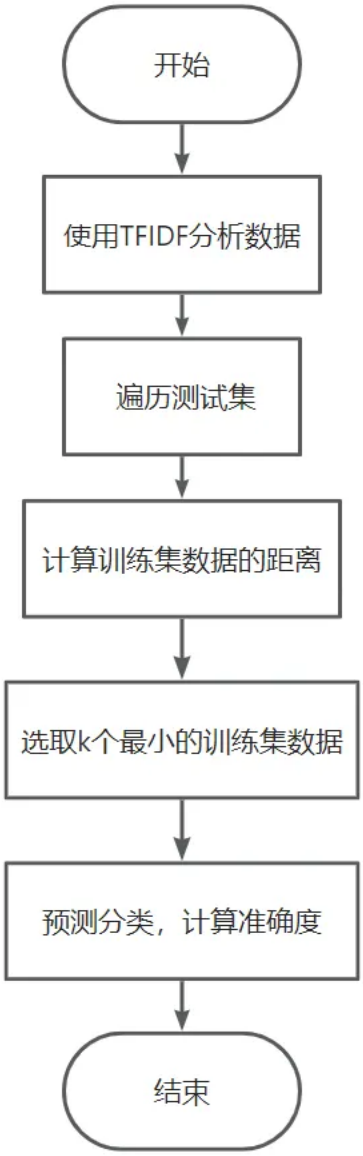
KNN算法的优点包括简单易懂、无需训练过程、适用于多分类问题等。

然而，它也有一些缺点，比如对于大规模数据集的计算开销大、对于高维数据效果可能不佳等。

- **文本情感分类**

- **初始化计数器：**correct_num用于记录分类正确的样本数量。
- **遍历测试集：**对测试集中的每个样本进行遍历。
- **计算距离：**计算当前测试样本与所有训练样本之间的距离，使用传入的距离度量函数来计算。
- **选择最近邻：**根据计算得到的距离，选择距离最近的K个训练样本。
- **统计标记：**统计这K个最近邻样本中各个标记的数量，这里是统计了每个情感类别的个数。
- **投票选出标记：**使用优先队列，按照情感类别的数量和距离之和，选出数量最多的情感类别作为当前测试样本的预测标记。
- **计算准确率：**将预测标记与实际标记进行比较，若相符则分类正确，计数器加1。
- **输出结果：**打印结果，包括距离度量方式、耗时和准确率等信息。

- **流程图**


开始
使用TFIDF分析数据
遍历测试集
计算训练集数据的距离
选取k个最小的训练集数据
预测分类，计算准确度
结束

二、伪代码

```
从各个库中导入所需的函数和类
定义函数 sklearn_KNN(train_array, test_array, train_message, test_message, k):
    创建KNN分类器对象knn
    从训练消息中提取训练集和测试集的标签
    使用训练数据训练KNN分类器
    对测试数据进行预测
    计算预测准确率
    打印准确率信息

定义函数 calculate_rmse(x_train, x_test, train_message, test_message):
    从训练消息中提取训练集和测试集的标签
    初始化存储不同k值下均方根误差的列表rmse_val
    循环k值从1到100:
        创建KNN回归模型对象model，设置邻居数量为K
        使用训练数据训练模型
        对测试数据进行预测
        计算预测值与真实值之间的均方根误差
        将均方根误差存储到rmse_val列表中
    绘制均方根误差曲线

定义函数 plot_data(test_data, train_data, train_message, distance_func):
    根据情绪类别设置颜色映射
    使用PCA将数据降至二维
    计算测试数据与训练数据之间的距离，并绘制训练数据点
    找出最近的15个训练数据点，并绘制包含这些点的圆
    绘制测试数据点，并添加标题

定义函数 Euclidean_distance(X, Y):
    计算欧氏距离并返回结果

定义函数 Manhattan_distance(X, Y):
    计算曼哈顿距离并返回结果

定义函数 Minkowski_distance(X, Y):
    计算闵氏距离并返回结果

定义函数 KNN_cosine_similarity(train, test, k, train_message, test_message):
    初始化正确分类数量correct_num为0
    记录开始时间
    获取测试数据与训练数据之间的余弦相似度
```

```
    对每个测试数据点进行遍历:
        找出与当前测试数据最相似的k个训练数据点的索引
        统计这k个训练数据点的情绪类别和相似度
        使用优先队列选取最多的类别
        如果最多类别与测试数据类别相同，则正确分类数量加一
    记录结束时间并计算花费时间
    打印结果信息

定义函数 KNN_general(train_array, test_array, k, train_message, test_message, distance_func):
    初始化正确分类数量correct_num为0
    记录开始时间
    获取训练集和测试集的大小
    对每个测试数据点进行遍历:
        计算测试数据与每个训练数据之间的距离
        找出与当前测试数据最近的k个训练数据点的索引
        统计这k个训练数据点的情绪类别和距离
        使用优先队列选取最多的类别
        如果最多类别与测试数据类别相同，则正确分类数量加一
    记录结束时间并计算花费时间
    打印结果信息
    调用 plot_data 函数绘制数据图

定义函数 read_file(f):
    初始化消息列表message和句子列表sentence
    对文件f中的每一行进行遍历:
        将每行数据按空格分割为临时列表tmp
        如果tmp的第一个元素是"documentId"，则跳过该行（表头）
        将文档序号、情绪编号、情绪标签和单个句子拼接为字符串并存入sentence列表
        将文档序号、情绪编号、情绪标签和单个句子作为列表存入message列表
    返回消息列表message和句子列表sentence

定义主函数 main():
    初始化距离函数列表distance_funcs为欧氏距离、曼哈顿距离和闵氏距离函数
    设置k值为16
    打印k值信息
    从train.txt文件中读取训练数据和句子
    从test.txt文件中读取测试数据和句子
    使用TF-IDF提取文本特征，得到训练数据和测试数据
    调用 calculate_rmse 函数计算不同k值下的均方根误差并绘制曲线
    打印训练数据和测试数据的形状信息
    调用 KNN_cosine_similarity 函数使用余弦相似度进行KNN分类并打印结果信息
    对每个距离函数进行遍历:
        调用 KNN_general 函数使用通用的KNN分类方法并打印结果信息
    调用 sklearn_KNN 函数使用Sklearn库中的KNN分类器并打印结果信息

```

```
86
    如果当前脚本是主模块，则执行主函数 main()
```

三、关键代码展示

> 完整代码见../code

- 三种距离计算函数

distance Python

```python
def Euclidean_distance(X, Y):  # 欧氏距离
    return np.sqrt(np.sum((X - Y) ** 2))


def Manhattan_distance(X, Y):  # 曼哈顿距离
    return np.sum(abs(X - Y))


def Minkowski_distance(X, Y):  # 闵氏距离
    return pow((np.sum(abs(X - Y) ** (len(X)))), 1 / len(X))
```

- 三种距离对应的KNN，原理见伪代码

KNN_general Python

```python
def KNN_general(train_array, test_array, k, train_message, test_message, distance_func):
    correct_num = 0

    time_start = time.time()
    m = len(train_message)
    n = len(test_message)

    for i in range(n):  # 遍历测试集
        distance = []
        for j in range(m):  # 遍历训练集
            distance.append(distance_func(test_array[i], train_array[j]))  # 计算距离
        indices = np.argsort(distance)[:k]  # 选取距离最小的k个的索引
        emotion = np.zeros(6)  # 存放每个情绪类别的个数（0: anger 1: disgust 2: fear 3: joy 4: sad 5: surprise）
        distance_tmp = np.zeros(6)  # 存放每个情绪类别的距离
        for j in range(k):  # 统计k个最近邻的类别
            emotion[train_message[indices[j]][1] - 1] += 1  # 统计每个类别的个数
            distance_tmp[train_message[indices[j]][1] - 1] += distance[indices[j]]  # 统计每个类别的距离
        # 使用优先队列选取最多的类别
        queue = []
        for p in range(0, 6):
            heapq.heappush(queue, (-emotion[p], -distance_tmp[p], p))
        _, _, max_id = heapq.heappop(queue)
        if max_id == test_message[i][1] - 1:
            correct_num += 1

    time_end = time.time()
    spend_time = time_end - time_start
    print("++++++++++++++++++++++++++++++++++++")
    print("\033[94mDistance metric:\033[0m", distance_func.__name__)
    print("\033[94mCost time:\033[0m", spend_time, "s")
    print("\033[94mAccuracy:\033[0m {:.4f}%".format(correct_num / n * 100))
    print("++++++++++++++++++++++++++++++++++++")
    plot_data(test_array[1], train_array, train_message, distance_func)  # 绘制第一个测试集的数据
```

- 应用余弦相似度的KNN（余弦相似度的k近邻分类，余弦相似度越大越好）

KNN_cosine_similarity Python

```python
def KNN_cosine_similarity(train, test, k, train_message, test_message):
    correct_num = 0
    time_start = time.time()
    n = len(test_message)
    similarity = cosine_similarity(test, train)  # 计算余弦相似度
    for i in range(n):  # 遍历测试集
        indices = np.argsort(similarity[i])[-k:]  # 选取相似度最大的k个训练集元素的索引（余弦相似度越大，向量越相似）
        emotion = np.zeros(6)
        distance = np.zeros(6)
        for j in range(k):
            emotion[train_message[indices[j]][1] - 1] += 1
            distance[train_message[indices[j]][1] - 1] += similarity[i][indices[j]]
        # 使用优先队列选取最多的类别
        queue = []
        for p in range(6):
            heapq.heappush(queue, (-emotion[p], -distance[p], p))
        _, _, max_id = heapq.heappop(queue)
        if max_id == test_message[i][1] - 1:
            correct_num += 1

    time_end = time.time()
    spend_time = time_end - time_start
    print("++++++++++++++++++++++++++++++++++++")
    print("\033[94mDistance metric:\033[0m Cosine_similarity")
    print("\033[94mCost time:\033[0m", spend_time, "s")
    print("\033[94mAccuracy:\033[0m {:.4f}%".format(correct_num / n * 100))
    print("++++++++++++++++++++++++++++++++++++")
```

- 读取数据

read_file Python

```python
def read_file(f):
    message = []  # 存放对应的信息特征
    sentence = []  # 存放文本信息
    for line in f:
        tmp = line.strip().split(" ")
        if tmp[0] == "documentId":  # 跳过表头
            continue
        documentId = int(tmp[0])  # 文本序号
        emotionId = int(tmp[1])  # 情绪编号
        emotion = tmp[2]  # 分类标签
        single_sentence = ' '.join(tmp[3:])  # 提取每一句的单词
        sentence.append(single_sentence)  # 存放文本信息
        # 0: documentId, 1: emotionId, 2: emotion, 3: sentence
        # 其中，emotionId-1即为情绪类别（0: anger 1: disgust 2: fear 3: joy 4: sad 5: surprise）
        message.append([documentId, emotionId, emotion, single_sentence])
    return message, sentence
```

- 画图

plot_data Python

```python
def plot_data(test_data, train_data, train_message, distance_func):
    # 创建一个映射，将情绪标签映射到颜色
    emotion_to_color = {1: 'g', 2: 'r', 3: 'c', 4: 'm', 5: 'y', 6: 'k'}
    """
    'b': 蓝色 (blue)
    'g': 绿色 (green) -> anger
    'r': 红色 (red) -> disgust
    'c': 青色 (cyan) -> fear
    'm': 品红色 (magenta) -> joy
    'y': 黄色 (yellow) -> sad
    'k': 黑色 (black) -> surprise
    """
    pca = PCA(n_components=2)
    train_data_2d = pca.fit_transform(train_data)

    if len(test_data.shape) == 1:
        test_data = test_data.reshape(1, -1)
    test_data_2d = pca.transform(test_data)

    distances = []
    labels = []
    for i in range(len(train_data)):  # 遍历训练集
        dist = distance_func(test_data_2d[0], train_data_2d[i])  # 计算距离
        distances.append(dist)
        labels.append(train_message[i][1])
        # 使用映射来确定颜色
        plt.scatter(train_data_2d[i, 0], train_data_2d[i, 1], color=emotion_to_color[labels[i]], s=5)
    # 找出最近的15个点
    distances.sort()
    nearest_15 = distances[:15]

    # 画出包含最近的15个点的圆
    circle_radius = nearest_15[-1]  # 最远的那个最近点的距离，即圆的半径
    circle = plt.Circle((test_data_2d[0, 0], test_data_2d[0, 1]), circle_radius, fill=False)
    plt.gca().add_patch(circle)

    plt.scatter(test_data_2d[0, 0], test_data_2d[0, 1], color='blue', label='Test data', s=25)
    # 添加标题，标注使用的距离函数
```

```
39      plt.title('Data Plot with Distance Function: {}'.format(distance_
    func.__name__))
40      plt.show(block=False)
41      while True:
42          if plt.waitforbuttonpress():
43              plt.close()
44              break
```

- 主函数

main Python

```python
def main():
    distance_funcs = [Euclidean_distance, Manhattan_distance, Minkowski_distance]
    k = 16  # 选取k值(根号n)
    print("++++++++++++++++++++++++++++++++++++")
    print("\033[94mValue of k:\033[0m", k)

    f = open(r".\Classification\train.txt", 'r')
    train_message, train_sentence = read_file(f)  # 读训练集
    f = open(r".\Classification\test.txt", 'r')
    test_message, test_sentence = read_file(f)  # 读测试集

    # TF-IDF 提取文本特征
    t = TfidfVectorizer()
    train = t.fit_transform(train_sentence)  # 读取训练集特征，此时返回一个sparse矩阵
    test = t.transform(test_sentence)  # 读取测试集特征，此时返回一个sparse矩阵

    train_array = np.array(train.toarray())
    test_array = np.array(test.toarray())
    # print("\033[94mTest_array[0]:\033[0m", test_array[0])

    calculate_rmse(train_array, test_array, train_message, test_message)  # 计算不同k值下的均方根损失
    # 打印train和test的基本信息
    print("\033[94mTrain data shape:\033[0m", train.shape)
    print("\033[94mTest data shape:\033[0m", test.shape)
    print("++++++++++++++++++++++++++++++++++++")

    KNN_cosine_similarity(train, test, k, train_message, test_message)  # 使用余弦相似度进行knn分类
    for distance_func in distance_funcs:
        KNN_general(train_array, test_array, k, train_message, test_message, distance_func)  # 使用其他距离进行knn分类
    sklearn_KNN(train_array, test_array, train_message, test_message, k)  # 使用sklearn中的KNN分类器
```

四、创新点&优化

- **多种距离度量方法选择**

欧氏距离，曼哈顿距离，闵氏距离

- **采用Numpy加速距离计算**

Numpy计算更快的原因主要有以下几点：

1. 底层优化： Numpy中的许多函数都是使用C语言编写的，并且针对高效的底层算法进行了优化。这些底层算法通常比Python的原生实现更快。
2. 向量化操作： Numpy支持向量化操作，允许在整个数组上执行操作而无需编写显式的循环。这意味着很多操作可以一次性应用于整个数组，而不是逐个元素处理，从而减少了Python解释器的开销。
3. 连续内存分配： Numpy数组在内存中是连续存储的，这意味着数据元素在内存中是依次存放的。这种连续的内存布局有利于CPU缓存的利用，减少了缓存未命中的次数，从而提高了访问速度。
4. 更少的类型检查： Numpy数组是静态类型的，其元素的类型在创建数组时就已经确定。这减少了在运行时进行类型检查的开销，提高了计算速度。
5. 并行计算： Numpy中的一些操作可以利用多核处理器进行并行计算，加快了运行速度。

- **对于不同k值计算均方根误差**

通过计算不同k值下的均方根误差和对不同距离度量方法的KNN分类器性能进行评估，虽然调用sklearn库中的KNN分类器时所计算的均方根误差对于自定义的几种距离函数并不是正相关，但是也可以提供一定的参考信息。

- **可视化多维数据集**

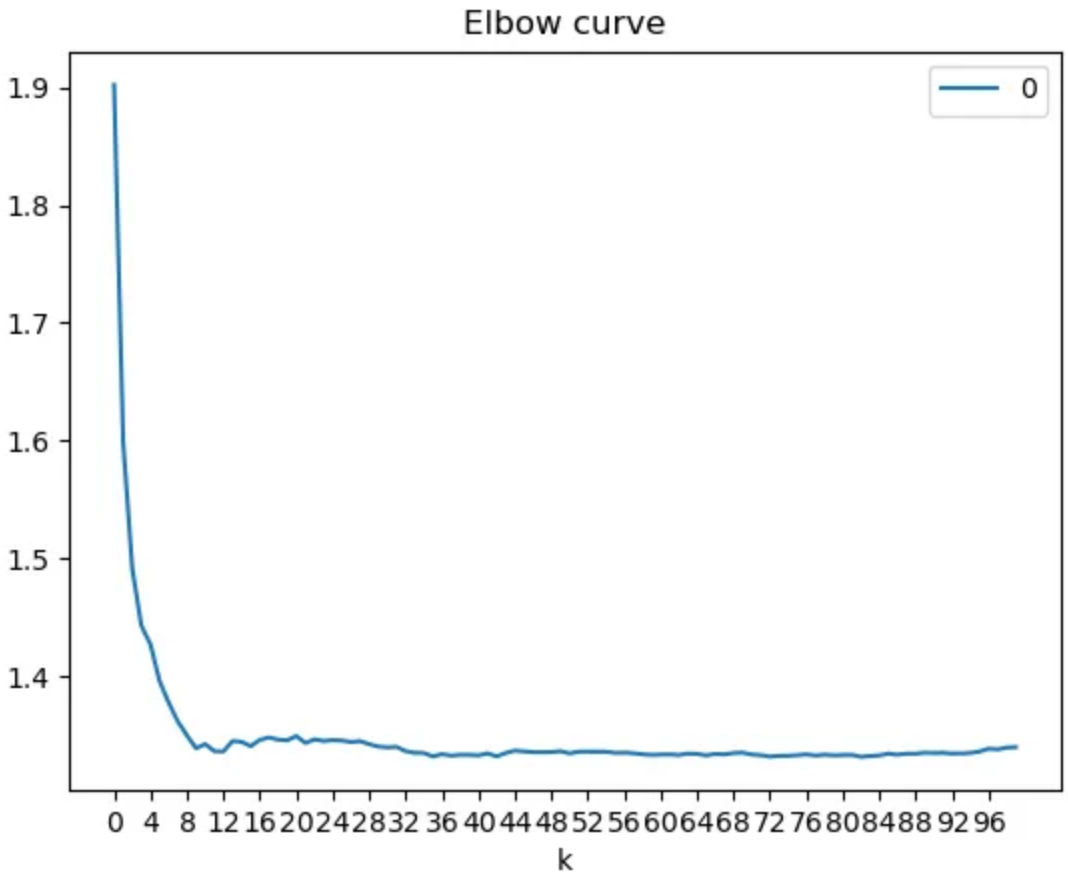
通过绘制数据图，我展示了测试数据与训练数据之间的关系，包括最近的训练数据点、测试数据点和包含最近训练数据点的圆形区域，使得数据的分布和分类结果更直观。

> 但是我发现了一定的问题，使用主成分分析PCA降维的时候散点的位置并不准确（？），故仅供参考

3 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

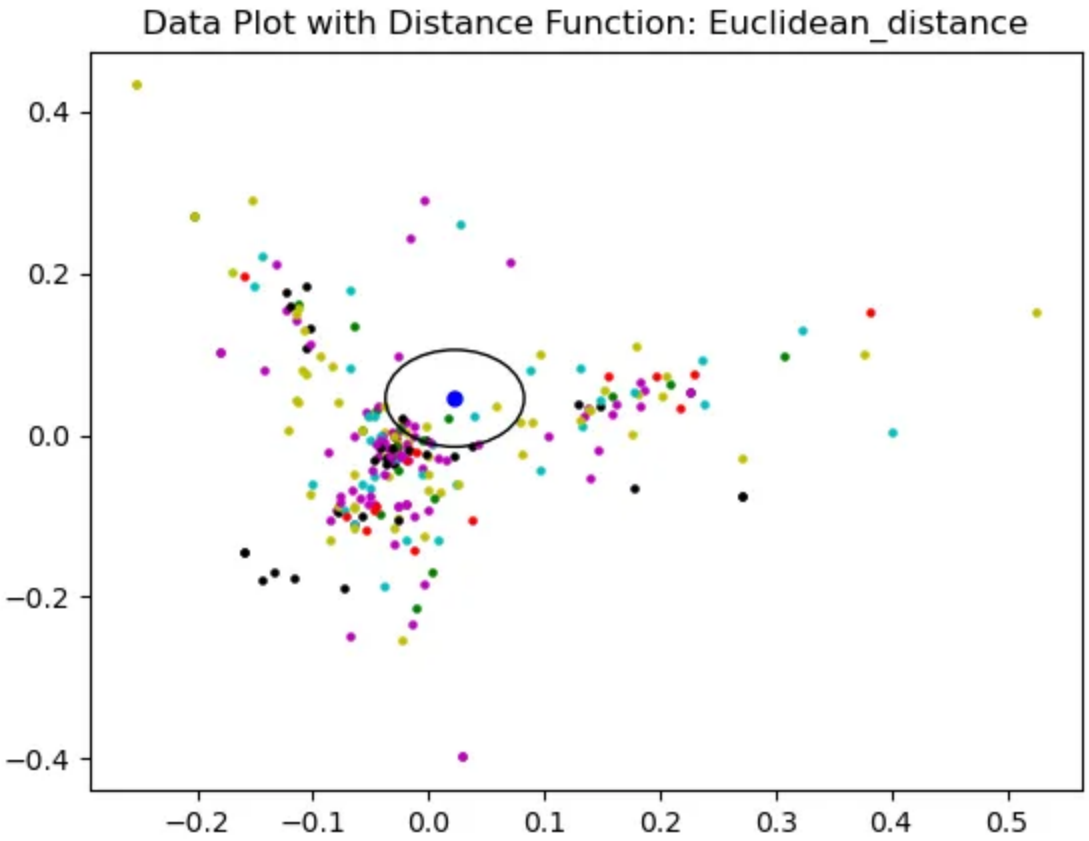
首先程序输出了不同k值下的KNN分类器的均方根误差以供参考，可见k在10之前均方根误差损失的非常快，到之后变化并不是特别明显。基于图表和PPT内的内容，我选择k=16。

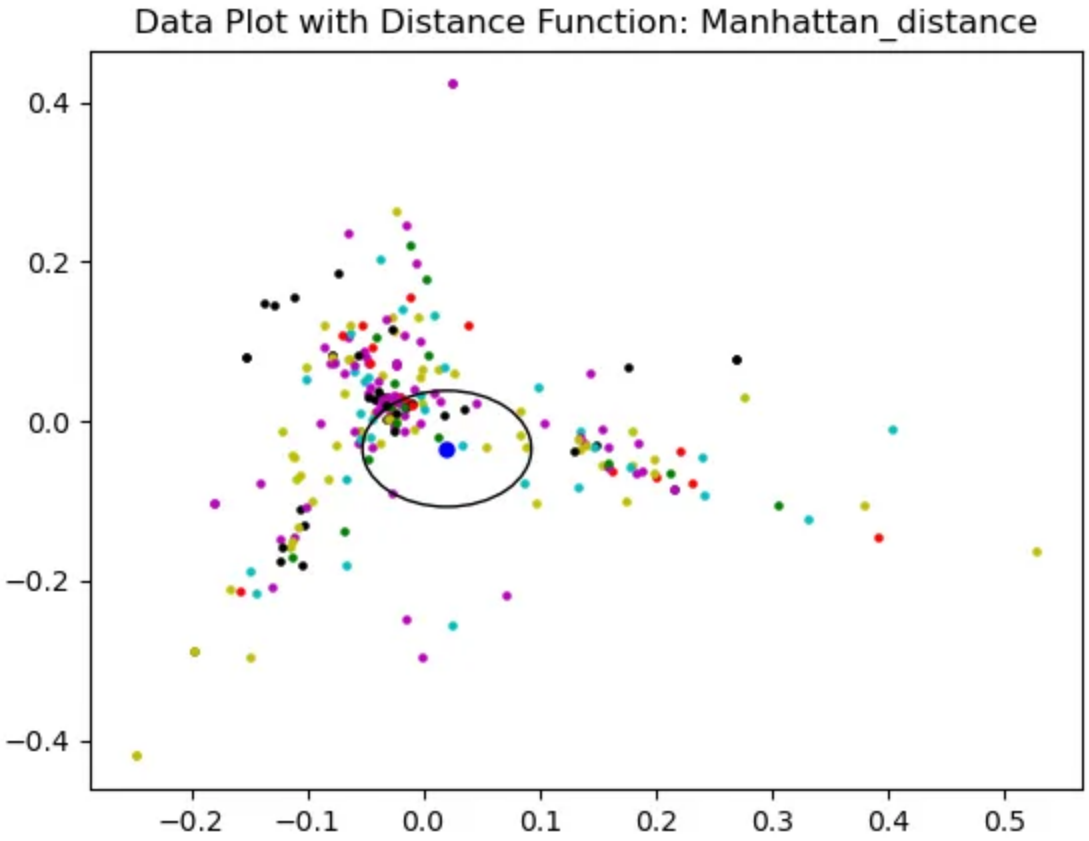


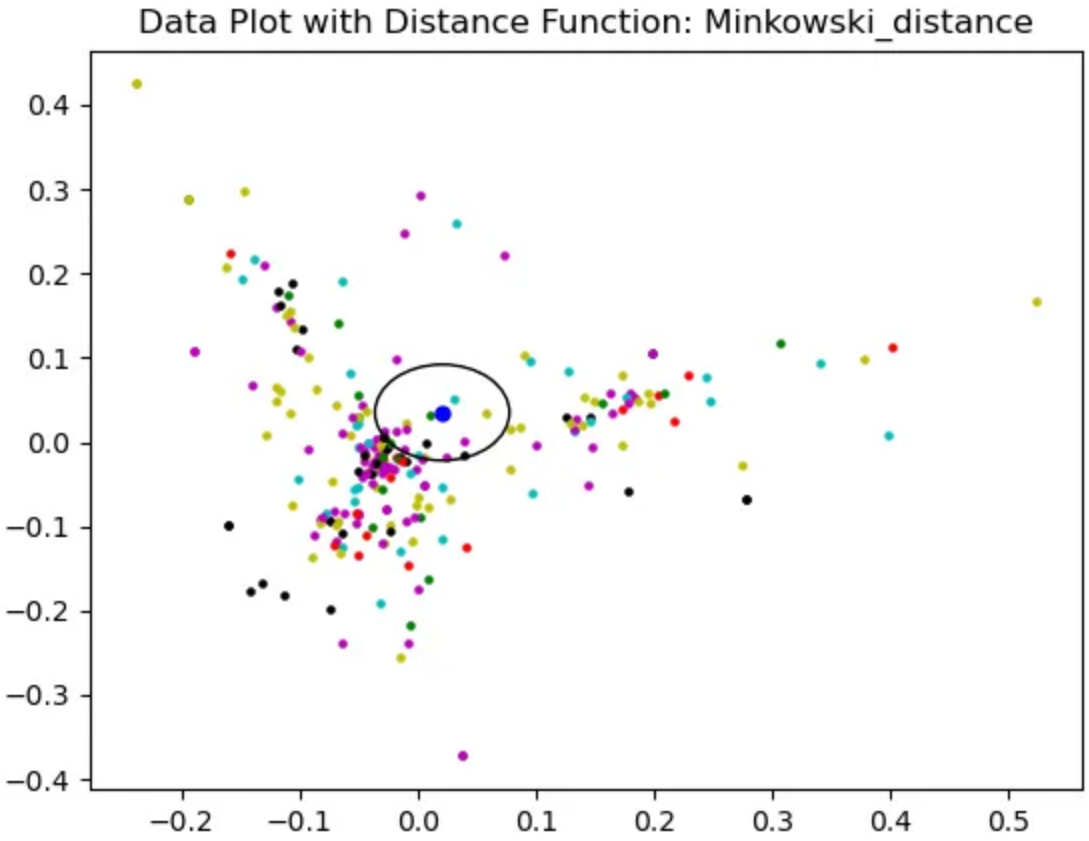
然后程序根据余弦相似度，欧氏距离，曼哈顿距离，闵氏距离来进行KNN分类。最后调用库内的KNN分类器分类，和前面四种比较。输出如下：

```
++++++++++++++++++++++++++++++++++++
Value of k: 16
Train data shape: (246, 896)
Test data shape: (1000, 896)
++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++
Distance metric: Cosine_similarity
Cost time: 0.04993247985839844 s
Accuracy: 38.1000%
++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++
Distance metric: Euclidean_distance
Cost time: 2.669344425201416 s
Accuracy: 32.6000%
++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++
Distance metric: Manhattan_distance
Cost time: 2.368488311767578 s
Accuracy: 38.3000%
++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++
Distance metric: Minkowski_distance
Cost time: 5.265030145645142 s
Accuracy: 28.3000%
++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++
KNN Classifier from sklearn:
Accuracy: 36.400000%
++++++++++++++++++++++++++++++++++++
```

可视化数据图如下：


Data Plot with Distance Function: Euclidean_distance
0.4
0.2
0.0
−0.2
−0.4
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5


Data Plot with Distance Function: Manhattan_distance
0.4
0.2
0.0
−0.2
−0.4
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5



2. 评测指标展示及分析

   k=16时各项准确率最高，分别为38.1%,32.6%,38.3%，调用KNN分类器时的准确率为36.4%。

   其他k值准确率均在20%–40%。

- **时间复杂度**：KNN分类的时间复杂度取决于训练数据的大小n、测试数据的大小m以及特征的维度d，通常情况下为O(n * m * d)，其中n为训练数据大小，m为测试数据大小，d为特征维度。
- **空间复杂度**：KNN分类的空间复杂度主要取决于训练数据的大小n，通常为O(n)，因为需要存储训练数据的特征向量和标签。

4 思考题

无

5 参考资料

实验课PPT