中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	信息与计算科学班	专业（方向）	信息与计算科学
学号	21311359	姓名	何凯迪

一、 实验题目

在给定的文本数据集完成文本情感分类训练，在测试集完成测试，计算准确率。需要对上次给的数据集进行重新划分，训练集：测试集为 8：2。

1. 文本的特征可以使用 TF-IDF（可以使用 sklearn 库提取特征）
2. 利用 KNN 完成对测试集的分类，并计算准确率
3. 报告中需探究超参 K 对分类准确率的影响
4. 需要提交简要报告+代码
5. 提交伪代码

二、 实验内容

1. 算法原理

KNN 是一种基本的**监督学习算法**，用于分类和回归问题。该算法的**基本思想**是根据特征空间中的最近邻居进行分类或预测。

以下是 KNN 算法的**原理步骤**：

数据准备：收集带有标签的训练样本数据，其中每个样本由多个特征组成。

距离度量：选择一种距离度量方式（例如欧氏距离、曼哈顿距离等），用于计算不同样本之间的距离。

选择 K 值：确定 K 的大小，它代表在进行分类或预测时考虑的最近邻居数量。

寻找最近邻居：对于一个未标记的样本，计算它与训练集中所有样本的距离，并选择与之最近的 K 个样本作为最近邻居。

确定类别或预测值：对于分类问题，根据 K 个最近邻居的标签进行投票，并选择获得最多投票的类别作为预测结果。对于回归问题，可以计算 K 个最近邻居的平均值或加权平均值，并将其作为预测值。

KNN 算法的**优点**是简单直观，易于理解和实现。它适用于小规模数据集和特征空间维度较低的问题。

KNN 算法的**缺点**包括计算开销大、存储要求高以及对于不平衡数据集和噪声敏感等。

2. 伪代码

关于 TF-IDF 和拉普拉斯平滑的伪代码已在上一次实验中给出，故不再赘述。

KNN 伪代码：

```
def distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def KNN(training_set, test_sample, K):
    distances = []
    for training_sample in training_set:
        distance = distance(training_sample[:-1], test_sample[:-1])
        distances.append((distance, training_sample[-1]))  # 距离和对应的标签
    distances.sort(key=lambda x: x[0])  # 根据距离排序
    neighbors = distances[:K]  # 取前 K 个最近邻居
    if K == 1:
        return neighbors[0][1]  # 如果 K=1，直接返回最近邻居的标签
    votes = {}
    for neighbor in neighbors:
        label = neighbor[1]
        votes[label] = votes.get(label, 0) + 1  # 统计每个类别的票数
    predicted_label = max(votes, key=votes.get)  # 获取票数最多的类别
    return predicted_label
# 主程序
training_set = [...]  # 训练样本集，包含特征和标签
test_samples = [...]  # 测试样本集，只包含特征
K =   # 选择 K 值
for test_sample in test_samples:
    predicted_label = KNN(training_set, test_sample, K)
    print("预测结果:", predicted_label)
```

3. 关键代码展示（带注释）

将原数据集按 8:2 划分：

```
import random
def split_dataset(data, train_ratio):
    random.shuffle(data)  # 随机打乱数据集

    total_samples = len(data)
    train_samples = int(total_samples * train_ratio)

    train_data = data[:train_samples+1]
    test_data = data[train_samples+1:]
```

```
    return train_data, test_data

# 读取训练集数据
with open('train.txt', 'r') as train_file:
    train_data = train_file.readlines()
# 读取测试集数据
with open('test.txt', 'r') as test_file:
    test_data = test_file.readlines()
# 合并数据集
all_data = train_data + test_data

# 分割数据集
train_ratio = 0.8  # 训练集所占比例
train_data, test_data = split_dataset(all_data, train_ratio)

# 将训练集和测试集分别保存到不同的文件中
with open('train_split.txt', 'w') as train_file:
    train_file.writelines(train_data)
with open('test_split.txt', 'w') as test_file:
    test_file.writelines(test_data)
```

TF-IDF：

```
# 计算 TF-IDF 特征
def calculate_tfidf_features(train_data, test_data):
    train_texts = [text for _, text in train_data]
    test_texts = [text for _, text in test_data]

    # 使用 TfidfVectorizer 计算 TF-IDF 特征
    vectorizer = TfidfVectorizer()
    train_features = vectorizer.fit_transform(train_texts)
    test_features = vectorizer.transform(test_texts)

    return train_features, test_features
```

归一化处理：

```
# 归一化处理
train_features_normalized = normalize(train_features)
test_features_normalized = normalize(test_features)
```

KNN:

```python
# KNN
def knn(train_features_normalized, train_labels, test_features_normalized, k):
    train_features_normalized = train_features_normalized.toarray()
    test_features_normalized = test_features_normalized.toarray()

    predicted_labels = []

    for test_sample in test_features_normalized:
        distances = []

        for i, train_sample in enumerate(train_features_normalized):
            # 计算欧氏距离
            distance = np.linalg.norm(test_sample - train_sample)
            distances.append((distance, train_labels[i]))

            # 计算曼哈顿距离
            #distance = np.linalg.norm(test_sample - train_sample,ord=1)
            #distances.append((distance, train_labels[i]))

            # 计算切比雪夫距离
            #distance = np.max(np.abs(test_sample - train_sample))
            #distances.append((distance, train_labels[i]))

        # 根据距离进行排序
        distances.sort(key=lambda x: x[0])

        # 选择前 k 个距离最近的样本
        k_nearest = distances[:k]

        # 统计最近样本中出现最多的标签
        labels = [label for _, label in k_nearest]
        predicted_label = max(set(labels), key=labels.count)
        predicted_labels.append(predicted_label)

    return predicted_labels
```

4. 创新点&优化（如果有）

（1）添加百度的停用词表

```
# 读取停用词表
def read_stop_words(file_path):
    with open(file_path, 'r', encoding="utf-8") as file:
        stop_words = file.read().splitlines()
    return stop_words
def calculate_tfidf_features(train_data, test_data, stop_words):
    train_texts = [text for _, text in train_data]
    test_texts = [text for _, text in test_data]

    # 使用 TfidfVectorizer 计算 TF-IDF 特征
    vectorizer = TfidfVectorizer(stop_words=stop_words)
    train_features = vectorizer.fit_transform(train_texts)
    test_features = vectorizer.transform(test_texts)

    return train_features, test_features
```

```
# 计算 TF-IDF 特征
train_features, test_features = calculate_tfidf_features(train_data, test_data,
stop_words)
```

（2）使用 PCA

PCA 是一种常用的降维技术和数据分析方法。它通过线性变换将原始数据投影到一个新的特征空间，使得投影后的特征具有最大的方差。这样做的目的是为了保留原始数据中最重要的信息，并且去除不相关或冗余的特征。基本思想是寻找一组新的正交基，将原始数据映射到这个新的基上。这些新的基称为主成分，按照方差的大小依次排列。第一个主成分是原始数据中方差最大的方向，第二个主成分是与第一个主成分正交且方差次大的方向，以此类推。通过选择前 m 个主成分，我们可以实现降维，并且保留了最重要的特征。

```
from sklearn.decomposition import PCA

    # 使用 PCA 进行降维
    pca = PCA(n_components=k)  # 设置要保留的主成分个数
    train_features_pca = pca.fit_transform(train_features.toarray())
    test_features_pca = pca.transform(test_features.toarray())

    return train_features_pca, test_features_pca
```

（3）运用多种距离度量方法

用了欧几里得距离、曼哈顿距离、切比雪夫距离和余弦相似度四种方法，其中最后一种由于表现太差所以我干脆删了。

```
# 计算欧氏距离
distance = np.linalg.norm(test_sample - train_sample)
distances.append((distance, train_labels[i]))

# 计算曼哈顿距离
#distance = np.linalg.norm(test_sample - train_sample,ord=1)
#distances.append((distance, train_labels[i]))

# 计算切比雪夫距离
#distance = np.max(np.abs(test_sample - train_sample))
#distances.append((distance, train_labels[i]))
```

三、 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）

我将原数据集按 8:2 划分，得到 4 组数据，分别存于 train_split.txt, test_split.txt, train_split1.txt, test_split1.txt, train_split2.txt, test_split2.txt, train_split3.txt, test_split3.txt. 其中 train_split.txt, test_split.txt 为按原编号顺序分为 8:2 的数据组，我将其称为“标准数据组”。

对于实验结果展示，我按照第一组数据进行展示。

欧氏距离 ：

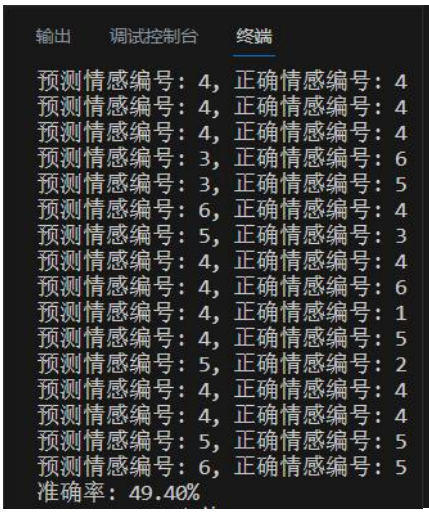
K=5

```
输出  调试控制台  终端
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 3, 正确情感编号: 6
预测情感编号: 5, 正确情感编号: 5
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 5, 正确情感编号: 3
预测情感编号: 5, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 6
预测情感编号: 3, 正确情感编号: 1
预测情感编号: 4, 正确情感编号: 5
预测情感编号: 3, 正确情感编号: 2
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 1, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 5
预测情感编号: 2, 正确情感编号: 5
准确率: 45.38%
```

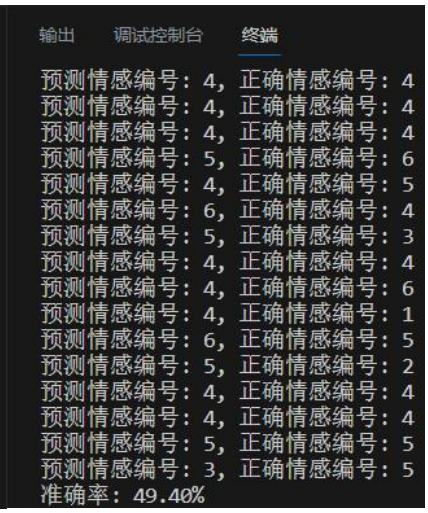
K=10

```
输出  调试控制台  终端
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 3, 正确情感编号: 6
预测情感编号: 4, 正确情感编号: 5
预测情感编号: 6, 正确情感编号: 4
预测情感编号: 5, 正确情感编号: 3
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 6
预测情感编号: 4, 正确情感编号: 1
预测情感编号: 4, 正确情感编号: 5
预测情感编号: 5, 正确情感编号: 2
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 4, 正确情感编号: 4
预测情感编号: 5, 正确情感编号: 5
预测情感编号: 6, 正确情感编号: 5
准确率: 51.41%
```

K=15



K=20

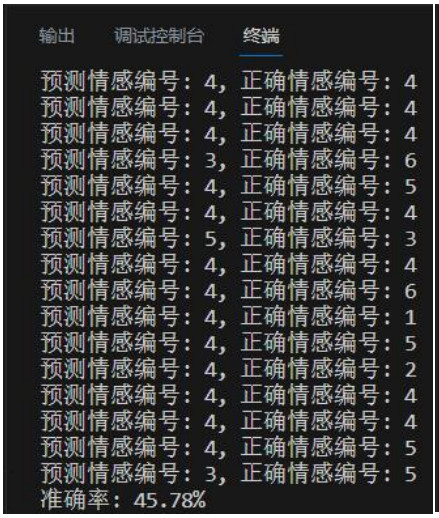


曼哈顿距离：

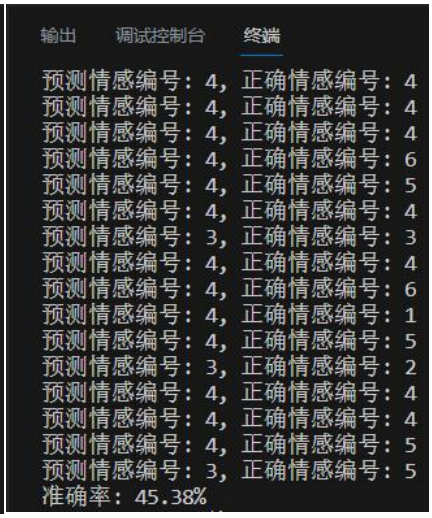
K=5

K=10

K=15



K=20



2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

基础结果：

正确率（单位：%）	数据一			
	k=5	k=10	k=15	k=20
欧氏距离	45.38	51.41	49.4	49.4
曼哈顿	51	46.99	45.78	45.38

正确率（单位：%）	数据二			
	k=5	k=10	k=15	k=20
欧氏距离	47.79	46.18	47.39	49.4
曼哈顿	42.17	45.38	43.37	37.75

正确率（单位：%）	数据三			
	k=5	k=10	k=15	k=20
欧氏距离	44.76	52.82	49.6	50.81
曼哈顿	45.97	47.98	44.76	44.76

正确率（单位：%）	标准数据			
	k=5	k=10	k=15	k=20
欧氏距离	44	46.4	44	46.8
曼哈顿	37.2	39.6	38	40

对于随机划分的数据一、二、三，数据一和数据三的训练和预测效果较好。
对于按顺序划分所得的标准数据，显然效果差强人意。
总的来说，一般情况下欧氏距离比曼哈顿距离较优，对于超参 k 的选择则受到文本分布影响。

添加停用词表的结果：

选用基础结果较优的数据一（欧氏距离）做进一步探索。

正确率（单位：%）	数据一			
	k=5	k=10	k=15	k=20
欧氏距离	45.38	51.41	49.4	49.4
曼哈顿	51	46.99	45.78	45.38
添加停用词表	40	42.4	44.4	45.6

显然，添加停用词表后的效果显著降低，原因至今未被探究出。

运用 PCA 进行降维：

选用基础结果较优的数据一（欧氏距离）做进一步探索。

正确率（单位：%）	数据一			
	k=5	k=10	k=15	k=20
欧氏距离	45.38	51.41	49.4	49.4
曼哈顿	51	46.99	45.78	45.38
PCA=100	45.78	50.6	46.18	46.99
PCA=200	48.59	47.79	48.19	48.59
PCA=300	49.4	49	48.59	48.59
PCA=400	49	52.21	50.2	51
PCA=500	49	53.01	50.6	51

使用 PCA 进行降维后，多数训练及预测效果有所改善。
当 PCA 值取 500 时，在 k=10 时甚至能达到 53.01%的正确率。

四、 参考资料

GitHub - goto456/stopwords: 中文常用停用词表（哈工大停用词表、百度停用词表等）