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	计科 1 班	专业（方向）	计算机科学与技术
学号	21307077	姓名	凌国明

一、实验题目

机器学习

实现朴素贝叶斯分类与 KNN 分类，处理文本的情感分类问题
尝试多种算法及算法中的不同策略/参数，并进行结果对比分析

	ID	text	class label
训练集	1	good,thanks	joy
	2	No impressive, thanks	sad
	3	Impressive good	joy
测试集	4	No, thanks	?

本次实验，我实现了

1. 朴素贝叶斯分类与 KNN 分类
2. RNN 循环神经网络分类

二、实验内容

1、算法原理

朴素贝叶斯

设文本为 x，长度为 m 的文本中每个单词为 $x_j\ (1 \leq j \leq m)$

文本的标签记为 y，共 n 个类别，标签的每个类别记为 $y_i\ (1 \leq i \leq n)$

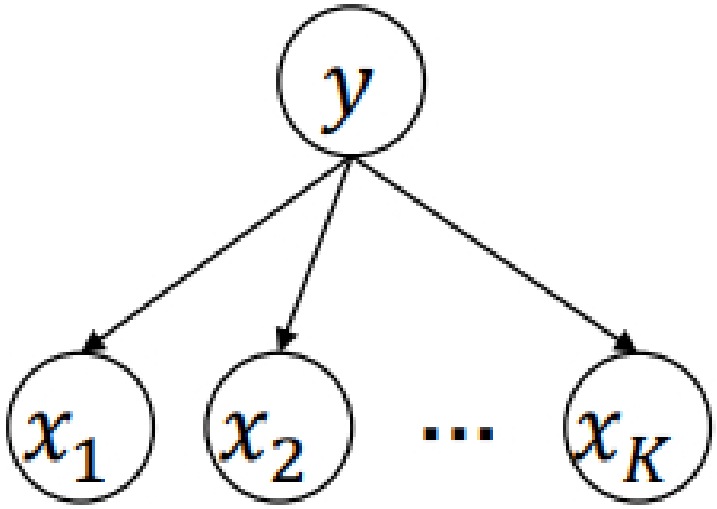
文本 x 属于类别 y 的概率是 $p(y|x) = \frac{p(xy)}{p(x)} = \frac{p(y)\cdot p(x|y)}{p(x)} \propto p(y) \cdot p(x|y)$

所以，**计算给定文本属于某个情感的概率，转化为了计算各个情感下出现此文本的概率**

$$p(y_i|x) = \frac{p(y_i)\cdot p(x|y_i)}{p(x)} \propto p(y_i) \cdot p(x|y_i) = p(y_i) \cdot \prod_{j=1}^{m} p(x_j|y_i)$$

我们需要估计的是 $\arg\max_{y_i} p(y_i|x)$，由上面的式子，转化为估计 $\arg\max_{y_i} p(y_i) \cdot \prod_{j=1}^{m} p(x_j|y_i)$

这里的 $p(x|y_i) = \prod_{j=1}^{m} p(x_j|y_i)$ 用到了**朴素贝叶斯的条件独立性假设：“假设在给定类别的条件下，特征变量间是独立的”**



$p(x_j|y_i)$ 的建模

伯努利模型

$n_{y_i}(x_j)$ 表示类别为 y_i 的文本中出现 x_i 的文本数量；
N_{y_i} 表示类别为 y_i 的文本数量；N 表示全部文本的数量。

$$p(x_j|y_i) = n_{y_i}(x_j)/N_{yi} \qquad p(y_i) = N_{y_i}/N$$

标签为 y_i 的文本中出现词 x_j 的概率 = （训练集中）含 x_j 且 标签为 y_i 的文本的数量 / 标签为 y_i 的文本的数量

多项式模型

$t_{y_i}(x_j)$ 表示类别为 y_i 的文本中 出现词 x_j 的总次数
t_{y_i} 表示类别为 y_i 的文本中 单词的总数
T_{y_i} 表示训练集中类别为 y_i 的文本的数量
T 表示训练集中 文本的数量

$$p(x_j|y_i) = t_{y_i}(x_j)/T_{yi} \qquad p(y_i) = T_{y_i}/T$$

标签为 y_i 的文本中出现词 x_j 的概率 = （训练集中）标签为 y_i 的所有文本中 x_j 出现的总次数 / 标签为 y_i 的所有文本的总词数

平滑处理

当测试时遇到训练集中没有见过的单词时，计算 $p(x|y_i) = \prod_{j=1}^{m} p(x_j|y_i)$ 时会遇到概率为 0 的情况，这个时候分类器就失效了。

拉普拉斯平滑是一种常用的平滑方法，它的思想是在估计概率时，给每个特征的出现次数加上一个固定的常数 k，避免出现 0 的情况。这样可以保证估计的概率不会为 0，并且避免在测试数据中出现未知特征导致分类器失效。

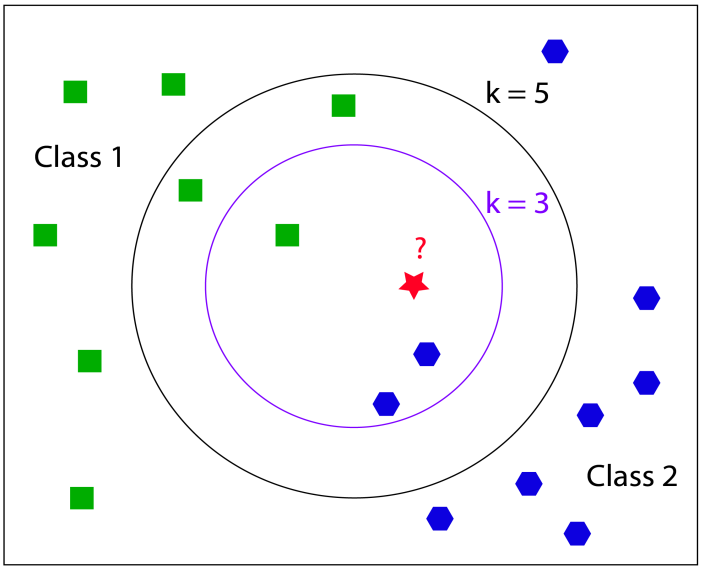
在多项式模型中使用拉普拉斯平滑

计算公式为 $p(x_j|y_i) = (t_{y_i}(x_j) + k)/(T_{yi} + k \cdot |V|)$

其中 $|V|$ 表示特征总数（即词汇表大小；k 为平滑参数，通常取值为 1

KNN

$KNN(K-NearestNeighbor)$ 算法的核心思想是：如果一个样本在特征空间中的 k 个最临近的样本中的大多数属于某一个类别，则该样本也属于这个类别。
KNN 中分类决策方式一般较为固定："多数表决，如果数量一致则看最接近的"。KNN 中最关键的是度量距离的方式，其次是参数 K。



k 的选取

一般根据样本的分布，选择一个较小的值，可以通过交叉验证选择一个合适的 k 值，K 通常是不大于 20 的整数。
选择较小的 k 值，相当于用小区域中的训练实例进行预测，训练误差会减小，但是泛化误差更可能增大。k 值的减小就意味着整体模型变得复杂，容易发生过拟合。
选择较大的 k 值，相当于用较大区域的训练实例进行预测，优点是可以减少泛化误差，但是训练误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且K值的增大就意味着整体的模型变得简单。

距离的度量方式

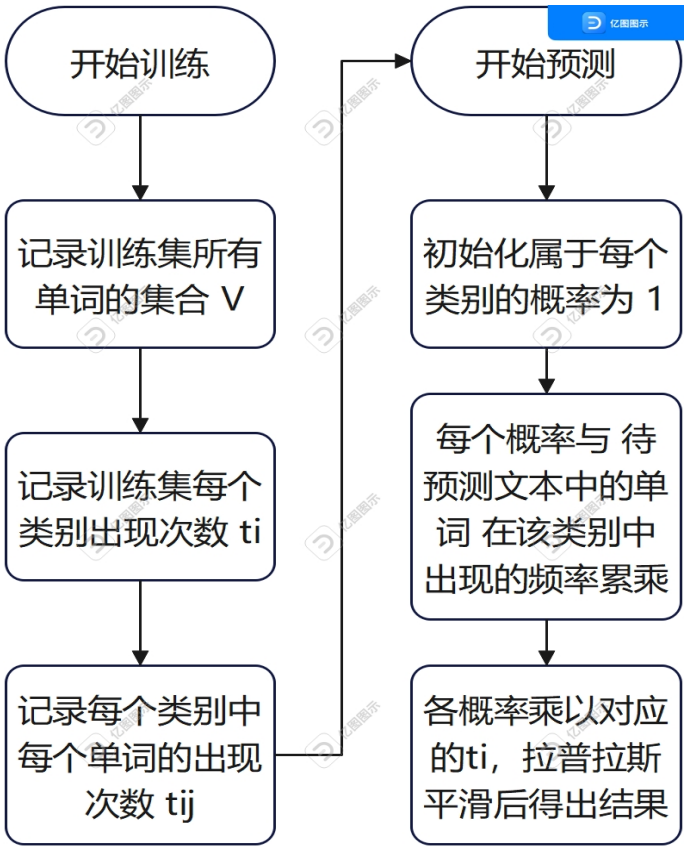
距离公式有曼哈顿距离，欧式距离，切比雪夫距离等等，这些都是闵氏距离 $(\sum_{i=1}^{n}|x_i-y_i|^p)^{1/p}$ (x,y是向量) 的特例

衡量向量的相似度时，还可以用到余弦相似度，点积相似度，皮尔逊相关系数等等

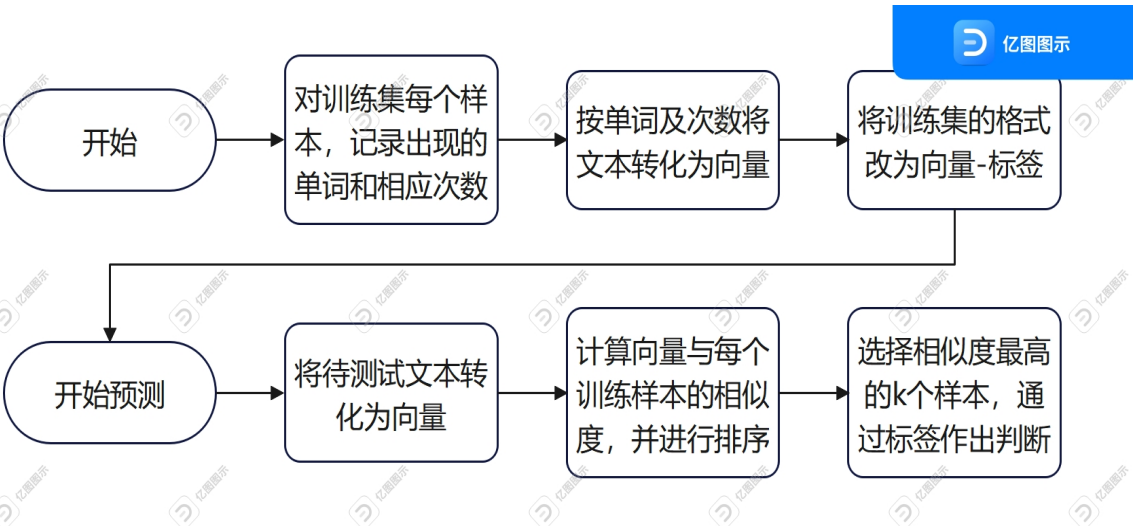
本实验的 KNN 用的是余弦相似度，公式为 $(\sum_{i=1}^{n}x_i\cdot y_i)/(\sqrt{\sum_{i=1}^{n}x_i^2}\cdot\sqrt{\sum_{i=1}^{n}y_i^2})$

2、伪代码，流程图

朴素贝叶斯



KNN



3、关键代码展示（带注释）

贝叶斯训练

```
# 计算每个类别中每个单词出现的次数
def fit(self, dataset, labels):
    for i in range(len(dataset)):
        text = dataset[i]
        label = labels[i]
        # 统计类别出现的次数（未见过的类别次数置0）
        self.category_count[label] = self.category_count.get(label, 0) + 1
        # 将文本数据转换为单词列表
        words = self.text_parse(text)
        for word in words:
            # 将单词添加到词汇表中
            self.vocabulary.add(word)
            # 统计每个类别中每个单词出现的次数
            if label not in self.word_count:
                self.word_count[label] = {}
            self.word_count[label][word] = self.word_count[label].get(word, 0) + 1
```

贝叶斯预测

```
# 计算单词在类别中出现的概率
def word_prob(self, word, category):
    # 计算类别中单词出现的总次数
    category_word_count = sum(self.word_count[category].values())
    # 计算单词在类别中出现的次数
    word_count = self.word_count[category].get(word, 0)
    # 计算概率，并使用拉普拉斯平滑避免出现概率为0的情况
    return (word_count + 2) / (category_word_count + 2*len(self.vocabulary))

# 计算文本数据属于每个类别的概率
def predict_prob(self, text):
    # 将文本数据转换为单词列表
    words = self.text_parse(text)
    # 初始化概率为1，避免出现概率为0的情况
    prob = {category: 1 for category in self.category_count.keys()}
    for category in self.category_count.keys():
        for word in words:
            # 计算单词在类别中出现的概率，并累乘
            prob[category] *= self.word_prob(word, category)
        # 计算文本数据属于该类别的概率（归一化）
        prob[category] *= self.category_count[category] / sum(self.category_count.values())
    return prob
```

文本向量化表示 (KNN)

```
# 将文本转换为向量表示
def text2vec(text):
    words = text.split()
    vec = {}
    for word in words:
        vec[word] = vec.get(word, 0) + 1
    return vec
```

计算两个向量的余弦相似度

```
# 计算两个向量之间的余弦相似度
def cosSimilarity(vec1, vec2):
    dotProduct = 0.0
    norm1 = 0.0
    norm2 = 0.0
    for key in vec1:
        # key in vec2 表示对应元素相乘，如果不对应则一方为 0，忽略
        if key in vec2:
            # Sigma(xi * yi)
            dotProduct += vec1[key] * vec2[key]
        norm1 += vec1[key] ** 2  # Sigma(xi^2)
    for key in vec2:
        norm2 += vec2[key] ** 2  # Sigma(yi^2)
    if norm1 == 0.0 or norm2 == 0.0:
        return 0.0
    else:
        # Sigma(xi*yi) / (sqrt(Sigma(xi^2) + sqrt(Sigma(yi^2) )
        return dotProduct / (math.sqrt(norm1) * math.sqrt(norm2))
```

KNN 预测

```
# k-近邻算法
def knnClassify(inputVec, dataset, labels, k):
    similarities = []
    for i in range(len(dataset)):
        sim = cosSimilarity(inputVec, dataset[i])  # 计算相似度
        similarities.append((sim, labels[i]))
    similarities.sort(reverse=True)  # 按照相似度降序排序
    classCount = {}
    for i in range(k):  # 获取最近的 k 个样本
        voteLabel = similarities[i][1]
        classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
    sortedClassCount = sorted(classCount.items(), key=lambda x:x[1], reverse=True)
    return sortedClassCount[0][0]
```

4、创新点&优化

KNN 中文本向量化的优化

训练集中有两千多个 $token$，采用词袋模型表示的话，向量维数两千多维，这在计算相似度时会很耗时间
考虑到每个句子的单词数都不多，可以**采用字典来表示词袋模型**，具体为 {word : times}，字典的底层是用**哈希**实现的，时间复杂度为 $O(1)$这一点大大加速了 KNN 的预测

rnn 的实现

实现了循环神经网络预测，首先是一层 $embedding$ 层，将文本的 $onehot$ 编码转化为词向量编码，然后经过一个带隐藏层的简单循环神经网络得到输出。$nn1$ 文件中的输出采用的是循环神经网络的最后一次输出，$nn2$ 文件中的输出采用的是循环神经网络多次输出的平均向量

```
训练集准确率:  0.9966555183946488
验证集准确率:  0.4358974358974359
```

三、实验结果与分析

1、实验结果展示示例

在训练集上训练，在验证集上调参后，在测试集的结果如下：

模型	nb	knn	rnn
准确率	0.474358	0.403846	0.435897

2、运行时间分析

假设训练集有N个样本，测试集有M个样本，每个样本是一个V维的向量。
使用线性搜索的话，那么k-NN的时间花销就是O(N*M*V)

四、思考题

伯努利模型和多项式模型

伯努利模型：

优点：

1. 适用于文本分类等二元变量的场景。
2. 能够处理缺失数据。
3. 可以使用稀疏矩阵存储，节省内存空间。

缺点：

1. 伯努利模型假设特征之间是相互独立的，这在实际应用中不一定成立。
2. 对于长文本的处理，伯努利模型可能会低估某些词汇的重要性。

多项式模型：

优点：

1. 能够处理多元离散变量的场景。
2. 多项式模型中特征之间可以有相关性，更符合实际应用中的情况。

缺点：

1. 对于缺失数据，多项式模型表现不如伯努利模型。
2. 多项式模型需要花费更多的存储空间。

IDF数值有什么含义？TF-IDF数值有什么含义？

IDF (Inverse Document Frequency)数值表示一个词语在整个文本集合中的重要性。它是由该词语在整个文本集合中出现的文档数目的倒数 logarithm。IDF 数值越高，表示该词语越重要。

TF-IDF(Term Frequency-Inverse Document Frequency)数值是指将TF (Term Frequency)和IDF两个值结合在一起，用于衡量一个词语在单个文档中的重要性。TF指该词语在单个文档中出现的频率，IDF指该词语在整个文本集合中的重要性。TF-IDF数值越高，表示该词语在该文档中越重要，同时在整个文本集合中的重要性也越高。

例如，如果一个词语在单个文档中出现的频率很高，但在整个文本集合中出现的文档数目也很多，那么它的TF-IDF数值可能并不高。相反，如果一个词语在单个文档中出现的频率较低，但在整个文本集合中

出现的文档数目很少，那么它的TF-IDF数值可能会很高。这是因为它在该文档中出现的频率很低，但对于该文档来说却是非常重要的词语。

TF-IDF表示一个词在这个文档中的重要程度。如果词w在一篇文档d中出现的频率高，并且在其他文档中很少出现，则认为词w具有很好的区分能力，适合用来把文章d和其他文章区分开来。词频高在文章中往往是停用词，“的”，“是”，“了”等，这些在文档中最常见但对结果毫无帮助、需要过滤掉的词，用TF可以统计到这些停用词并把它们过滤。当高频词过滤后就只需考虑剩下的有实际意义的词。

课件中计算相似度时，为什么使用倒数？如果要求同一测试样本的各个情感概率总和为1，应该如何处理？

因为相似度和距离是负相关的，距离越大，相似度越低，距离越小，相似度越高

五、参考资料

https://blog.csdn.net/zgcr654321/article/details/85219121 KNN原理
https://www.zhihu.com/question/272195313 两组向量的相似度