

Data Mining 实验报告

Homework 1 : VSM and KNN

一、实验要求

- 1、对于 20 个新闻组数据集，进行必要的预处理之后，建立每个文本的空间向量模型。
- 2、按照 8:2 的比例划分训练集和测试集（要求测试数据均匀覆盖各个类别），应用 KNN 算法对测试数据进行分类，统计分类准确率。

二、实验步骤

1、实验数据预处理

需要对每个类别里的每个文件的文本进行预处理，使之转换为更容易处理的格式。需要进行分词操作，将句子划分为单词组成的列表，实验中使用 python 的 re 模块，来进行分词；并且需要删除无实际意义的停用词、字母转换为小写。然后通过文件读写操作将处理后的数据进行保存。处理后每个单词占一行，格式如图 1 所示。

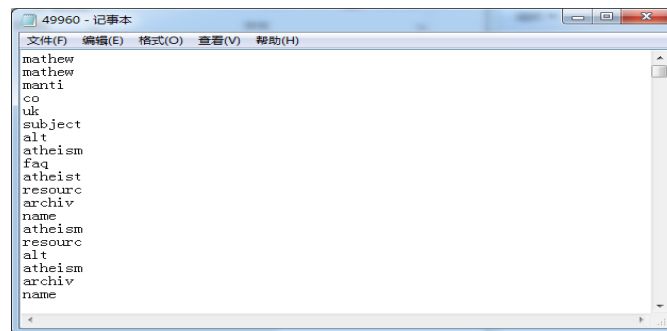


图 1 预处理后的文档格式

2、数据集的划分

要求按照 8:2 的比例划分训练集和测试集且测试数据均匀覆盖各个类别，实验中在遍历各个类别的文件时，首先计算该类别内的文件总数，遍历各文件时进行计数，使用 python 中的 shutil 模块

进行文件的复制，将占该类别总数的 80% 文件拷贝到训练集的指定路径下，20% 拷贝到测试集路径下。

3、 VSM

空间向量模型基本思想是将一个文档看做向量空间中的一个点。文档中出现的词的重要程度是不同的，用词权重来度量。实验中用 TF_IDF 算法来计算词权重。

首先需要为数据集建立词典（用字典存储，key 为 word，value 为 idf 值）：遍历每一个文档时，使用 Counter 函数得到该文档中每个词的词频字典，从而得到整个数据集的词频字典以及 df 字典，利用词频减小字典长度，只保留词频大于某个值的词，并计算这些词的 idf 值。

然后为每一个文档建立 tf_idf 字典：需要注意的是词频 (tf) 是针对一个文档而言的，所以为每个文档建立一个词频字典。遍历某个文档中所有单词，首先判断该单词是否出现在 idf 字典中（如果不出现，则没有计算其 tf 值的必要），若出现，则统计该词的词频，并计算该词的 tf_idf 值，最终得到一个文档的 tf_idf 字典（{ 'word1' : tf_idf1, 'word2' : tf_idf2, }）

Tf_idf 算法取一个文档中 tf_idf 值较大的前 50 个词作为该文档的关键词。由于 KNN 算法需要使用每个文档的类别属性，所以最终创建的向量形式为 ['label' , { 'word1' : tf_idf1, 'word2' : tf_idf2, }]

4、 KNN

KNN 算法的实现步骤：首先将训练集和测试集的每个文档都表示成向量，对于测试集的每一个向量，计算其与每一个训练向量的相似度，将训练向量的类型和相似度作为二元组存储在列表中，取列表中相似度最大的 K 个元组（需要排序操作），统计元组中类型出现次数，选取出现次数最多的类别作为该测试向量的分类结果。分类结果与测试向量的类名进行比较，统计分类正确的次数与分类错误的次数，计算分类正确率。

三、实验结果分析

1、多次修改 k 值观察其对分类准确率的影响，发现 k 取 10 时，实验准确率最高，约为 83.697%。如图 2 所示。

```
the performance of KNN:
('total test numbers:', 3754)
('number of success:', 3142)
('number of failure:', 612)
('the success rate:', 0.83697389451252)
```

图 2 实验结果 (k=10)

2、字典建立的规模也会影响准确率和运行时间，字典规模太小会使得分准确率下降，规模太大则运行时间会大大增加。建立字典时，通过更改过滤词频大小来观察字典的规模，选择能得到合适规模的设置。

四、感受体会

1、整个实验使用 python 语言进行编程，由于之前没有使用 python 进行编程的经验，所以花费了一些时间去学习 python 的语法以及数据结构。实验中切实感受到了 python 语言的简洁性以及许多模块和函数的方便，使用 python 的 nltk、os、math 模块在预处理等步

骤上极大地减少了代码编写量，编程中要合理地选择列表、元组、字典等数据结构实现目的。

- 2、 VSM 以及 KNN 的实现，首先要根据思想确定每一步的步骤，弄清楚每一个步骤要达到的效果，分模块进行编程。在建立空间向量模型时，一开始对 tf 的理解出现了一些偏差。后来明白为数据集建立词典时，我们需要统计整个数据集内单词的出现次数来对出现在词典的单词进行过滤。而为每一个文档建立向量时，需要得到该文档中的每一单词的 tf_idf 值，此时的 tf 指的是该单词在该文档中的标准化词频，它的值等于某个词在该文档中出现的次数/该文出现次数最多的词出现的次数。
- 3、 实验的过程是不断发现问题与解决问题的过程，遇到问题要结合网上相关资料进行思考，弄清楚问题出现的原因以及解决办法。

Homework 2 : 朴素贝叶斯分类器的实现

一、 实验要求

按照 8:2 的比例对 20 个新闻组数据集划分为训练集和测试集（要求测试数据均匀覆盖各个类别），应用朴素贝叶斯分类器对测试数据进行分类，统计分类准确率。

二、 朴素贝叶斯分类器

- i. 基本思想：对于待分类项，求解在该项出现的条件下各个类别出现的概率，哪个概率最大，就认为该分类项属于哪个类别。
- ii. 正式定义

1、设 $x = \{a_1, a_2, \dots, a_m\}$ 为一个待分类项，而每个 a 为 x 的一个特征属性。

2、有类别集合 $C = \{y_1, y_2, \dots, y_n\}$ 。

3、计算 $P(y_1|x), P(y_2|x), \dots, P(y_n|x)$ 。

4、如果 $P(y_k|x) = \max\{P(y_1|x), P(y_2|x), \dots, P(y_n|x)\}$ ，则 $x \in y_k$ 。

关键就是如何计算 3 中的各个条件概率。由条件概率可知：

$$P(y_i|x) = \frac{P(x|y_i)P(y_i)}{P(x)}$$

忽略分母，分子中的 $P(x|y_i)$ 转换为某文档中每个词在各个类别中出现概

率的乘积。即分子 =
$$P(y_i) \prod_{j=1}^m P(a_j|y_i)$$

iii. 平滑技术

$$P(\text{"发票"}|S) = \frac{\text{每封垃圾邮件中出现“发票”的次数的总和}+1}{\text{每封垃圾邮件中所有词出现次数（计算重复次数）的总和}+\text{被统计的词典的词数}}$$

多项式平滑技术： $P(\text{词 } w|\text{类 } A) = (\text{词 } w \text{ 在类 } A \text{ 中出现的总次数}+1)$

$/ (\text{类 } A \text{ 中所有词出现的次数}+\text{类 } A \text{ 词典数})$

iv. 具体实现

1. 为训练集建立向量

将训练集内每一个类别表示成一个向量，格式为[类名，类中单词总数，类出现的概率，类的单词字典]，训练集为一个 list，每个元素是各个类别的向量。

③ 由于每个类别出现的概率 = (该类中的文档总数) / (训练集中文档总数)，

所以编写函数 `count_files()`，用来计算训练集内文档总数。运行程序可知：训练集文档总数：15056。

②遍历训练集内的每一个文件夹(即类别)时,需要统计类中单词总数,这需要遍历每个文档时进行累加;类的词典应包括该类中每个文档中出现的单词以及它们出现的总次数。值得注意的是类中单词总数以及类的字典要在遍历完该类中所有文件之后再追加进该类的列表中。运行该函数后生成的部分训练集向量如图 3 所示。

```
第15个向量:
['sci.space', 145231, 0.052404357066950055, {'yellow': 5, 'interchang': 1, 'four': 48, 'prefix': 16, 'tr
第16个向量:
['soc.religion.christian', 161991, 0.052935706695005316, {'parenthetc': 1, 'osiri': 11, 'foul': 9, 'inte
第17个向量:
['talk.politics.guns', 148867, 0.048352816153028694, {'children': 1, 'yellow': 3, 'four': 28, 'authorit'
第18个向量:
['talk.politics.mideast', 203207, 0.04994686503719448, {'fawn': 2, 'zurueckfuehren': 1, 'convic': 3, 'fo
第19个向量: |
['talk.politics.misc', 150318, 0.04117959617428268, {'orthogon': 2, 'clerali': 1, 'woodi': 9, 'osiri': 1
第20个向量:
['talk.religion.misc', 95065, 0.033342189160467585, {'moskowitz': 2, 'osiri': 4, 'yellow': 2, 'four': 20
```

图 3 创建的部分训练集向量

1. 为训练集建立向量

将测试集的每一个文档表示成向量, [类名, 以该文档所有单词为元素的列表]。运行该函数后生成的部分测试集向量如图 4 所示。

```
第3770个向量:
['talk.religion.misc', ['bill', 'world', 'would', 'better', 'connect', 'pharvey', 'propheci', 'form',
第3771个向量:
['talk.religion.misc', ['among', 'individu', 'true', 'massada', 'cunyvm', 'gerri', 'common', 'truth',
第3772个向量:
['talk.religion.misc', ['bill', 'someth', 'mif', 'tek', 'gag', 'bil', 'kfu', 'reign', 'funni', 'thi',
```

图 4 创建的部分测试集向量

2. 为训练集建立向量

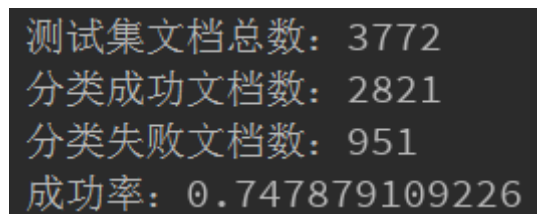
实现 NBC 的主要操作是计算后验概率 $P\{c_i|x\}$ (其中 C_i 指某一类别, x 指待分类的文档)。该后验概率的计算已转换为计算乘积: $P\{C_i\}$ *每个词在该类别中出现的概率。

为了避免乘积结果超出计算机计算的下限,采用取对数操作,将乘法转换为

加法。

对于每一个文档，要计算该文档出现时各个类别出现的概率，所以每个文档有一个列表，列表元素为元组，元组元素分别为类别名和概率。使用 sort 函数对列表根据概率值进行排序，排序后列表第一个元组中的类名即朴素贝叶斯算法对该文档的分类结果。统计所有文档的成功次数与失败次数，计算准确率。

运行该函数截图如图 5 所示。



```
测试集文档总数: 3772
分类成功文档数: 2821
分类失败文档数: 951
成功率: 0.747879109226
```

图 5 NBC 算法性能

三、实验总结

朴素贝叶斯算法与其他机器学习中的算法不同，像如 KNN 算法等，均是直接学习出特征输出 Y 和特征 X 之间的关系；但是朴素贝叶斯是直接找出特征输出 Y 和特征 X 的联合分布 $P(X,Y)$ ，然后用 $P(Y|X)=P(X,Y)/P(X)$ 得出。朴素贝叶斯思想简单直观，计算量也不大，在文本分类中具有不错的分类效果。

值得注意的是，朴素贝叶斯模型给定输出类别的情况下，假设属性之间相互独立，这个假设在实际应用中往往是不成立的，在属性个数比较多或者属性之间相关性较大时，分类效果不好。而在属性相关性较小时，朴素贝叶斯性能最为良好。

Homework 3 : clustering

一、实验要求

- i. 测试 sklearn 中聚类算法在数据集上的聚类效果；

- ii. 使用 NMI(Normalized Mutual Information)作为评价指标。

二、数据集

- 1、数据集 Tweets 以 Json 格式存储数据，格式如下：`{"text": "centrepont winter white gala london", "cluster": 65}`。需要将其进行预处理，将文本和 label 存储在列表中，然后对文本进行分词处理，后续算法会用到 tf_idf 矩阵，可以用过 sklearn 中的 TfidfVectorizer 方法得到。

三、实验过程

- i. 安装 sklearn

使用 `pip install` 命令进行安装，安装 sklearn 模块之前要确保已经安装了 numpy 和 scipy。

- ii. 对于实验要求指定的 8 种算法, 通过 scikit-learn 官方文档进行学习，通过文档以及示例熟悉 8 种算法的使用方法。

1. K-Means 算法

算法的目的是将 n 个向量分别归属到 K 个中心点里面去。算法首先会随机选择 K 个中心向量，然后通过迭代计算以及重新选择 K 个中心向量，使得 n 个向量各自被分配到距离最近的 K 中心点，并且所有向量距离各自中心点的和最小。

值得注意的是：有两个地方是需要算法使用者去自己选择的：第一个就是 K 的值，简而言之就是数据集应该被分成多少个类，在 K-Means 算法里面是要求先给出 K 值的；第二个就是距离函数，即如何计算向量和向量或者向量和中心点之间的距离，最常见的自然是欧式距离，也可以用余弦相似度来作为距离函数。

2. Affinity Propagation (AP 近邻传播聚类算法)

思想：根据 n 个点之间的相似度进行聚类（相似度可以用两个点之间的欧式距离来衡量）。两个点之间的相似度可以相同也可以不同，构成一个 $n \times n$ 的相似度矩阵。该算法不需要事先指定聚类数目，它将所有的数据点都作为潜在的聚类中心。

以 S 矩阵的对角线上的数值 $s(k, k)$ 作为 k 点能否成为聚类中心的评判标准,这意味着该值越大,这个点成为聚类中心的可能性也就越大,这个值又称作参考度 p (preference)。聚类的数量受到参考度 p 的影响,如果认为每个数据点都有可能作为聚类中心,那么 p 就应取相同的值。如果取输入的相似度的均值作为 p 的值,得到聚类数量是中等的。如果取最小值,得到类数较少的聚类。

3. MeanShift 算法

算法原理：在 n 个数据点中随机选择一个点作为圆心，找出距离圆心小于等于 $bandwidth$ 的所有点（即以该点为圆心，以 $bandwidth$ 为半径的园内所有点），这些点属于和圆心相同类别的概率要+1，然后计算出一个漂移向量，使圆沿着漂移向量移动，移动后的圆重复概率+1 操作。直至所有的点均被标记类别。漂移向量的计算方法是将园内所有点到圆心的向量之和。这使得圆会向着点密集区域移动。也可以理解成该算法是沿着密度上升的方向寻找属于一个簇的点。

编程过程中将 `tfidf_matrix` 作为参数传递时会报如下错误：
TypeError: A sparse matrix was passed, but dense data is required.

Use `X.toarray()` to convert to a dense numpy array. 于是将参数改为 `tfidf_matrix.toarray()`。

4. spectral_clustering 算法

它是一种基于图论的聚类方法（这点上跟 AP 类似，而 K-Means 是基于点与点的距离计算），它能够识别任意形状的样本空间且收敛于全局最有解，其基本思想是利用样本数据的相似矩阵进行特征分解后得到的特征向量进行聚类。

5. agglomerative_clustering 算法

凝聚是一开始将每个样本当做一个聚类，接着通过计算将距离最近的两个聚类合并，成为新聚类，每次合并聚类总数减少一个，不断循环合并操作，直到所有聚类合并成一个聚类或当聚类数量到达某预定值或当聚类直接距离达到某阈值后停止合并。而分裂则与凝聚相反，一开始将所有样本当做一个聚类，每次分裂一个聚类，直到满足某条件。

6. GaussianMixture 算法

混合高斯模型是用高斯概率密度函数（正态分布曲线）精确地量化事物，将一个事物分解为若干的基于高斯概率密度函数（正态分布曲线）形成的模型。

7. DBSCAN 算法

基于密度的方法的特点是不依赖于距离，而是依赖于密度，从而克服基于距离的算法只能发现“球形”聚簇的缺点。

DBSCAN 的核心思想是从某个核心点出发，不断向密度可达的

区域扩张，从而得到一个包含核心点和边界点的最大化区域，区域中任意两点密度相连。就是我们先找到一个核心对象，从它出发，确定若干个直接密度可达的对象，再从这若干个对象出发，寻找它们直接密度可达的点，直至最后没有可添加的对象了，那么一个簇的更新就完成了。我们也可以说，簇其实就是所有密度可达的点的集合。

8. birch 算法

BIRCH 算法利用了一个树结构来帮助我们快速的聚类，这个数据结构类似于平衡 B+ 树，一般将它称之为聚类特征树(Clustering Feature Tree，简称 CF Tree)。这颗树的每一个节点是由若干个聚类特征(Clustering Feature，简称 CF)组成。聚类特征树中，每个节点包括叶子节点都有若干个 CF，而内部节点的 CF 有指向孩子节点的指针，所有的叶子节点用一个双向链表链接起来。

四、实验结果

```
使用kmeans算法进行聚类，准确率为：0.7765083908130822  
使用AP近邻传播聚类算法进行聚类，准确率为：0.7624742003503602  
使用MeanShift算法进行聚类，准确率为：0.721245221852964  
使用spectral_clustering算法进行聚类，准确率为：0.6738977154419252  
使用agglomerative_clustering算法进行聚类，准确率为：0.7868056884757556  
使用GaussianMixture算法进行聚类，准确率为：0.7868056884757556  
使用DBSCAN算法进行聚类，准确率为：0.669611144599745  
使用birch算法进行聚类，准确率为：0.7685330121788387
```

图 6 各种聚类算法结果