|  |  |
| --- | --- |
| **自然语言处理及应用**  **实验报告** | |
|  | |
|  |  |
| **名称** | 文本分类 |
| **姓名** | 黄隆宁 |
| **班级** | 硕1224班 |
| **学号** | 3121155049 |
| Email | hln18773372567@stu.xjtu.edu.cn |
| **日期** | 2020-10-22 |

# 实验目的

使用KNN和Naïve bayes实现文本分类器，评估分类效果，并分析结果。

# 实验环境

Ubuntu 18.04

Python3.8

调用build-in库：

re：正则表达式用于文本预处理

os：由于数据文件扫描及加载

heapq：实现基于堆的距离筛选

collections：词权重统计及预测结果分析

pip依赖：

numpy：绝大部分的数据表示和数学计算

sklearn：数据集划分及预测结果分析

# 实验方法

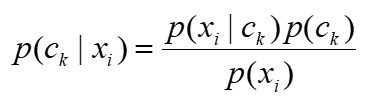
**所有的实验代码都更新在我的Github，可在Github在线查看：https://github.com/HLNN/NLP**

**因为本次实验的内容较多，设计预处理、建模、评估和超参数调试多个部分。因此实验方法和实验结果部分，展示最简单最基础的数学表示和代码实现。算法不足及实际遇到的问题在第五部分进行阐述。**

**原理分析：**

**Naïve bayes：**

贝叶斯分类器利用贝叶斯公式，将后验概率转化为条件概率和先验概率之积。



在实际算法设计中，可以对上述公式进行一些数学运算，转化为矩阵形式，提高计算机的运算速度。具体的实验代码完全按照PPT中的逻辑编写。



其中，P(ck)为类别的先验概率，直接计算各类数据占比得到。 P(tj|ck)为单词tj在ck 类别中出现的概率，统计每个类别中各单词的出现频率即可得到。nij为预测数据中单词出现的次数，对预测数据中各单机记数可得。

**KNN：**

KNN的基本思路非常简单，即从词向量空间中，找出和预测数据距离最近的k个向量，用这些向量中出现最多的类别，表示预测数据的类别，其中k为超参数。

这个问题可以分解为以下几个部分：

* 1. 词向量空间的表示方法。
  2. 向量距离的表示方法及最小距离向量的计算方法。
  3. 实验结果的评估方式及超参数k的选择方式。

词向量空间表示方面，直接复用Naïve bayes中的词频统计代码，即使用tf权重，将数据投影到词向量空间中。距离计算初步使用最为常见的欧几里得距离，使用Python内建库heapq实现基于堆的距离筛选。实验结果分析使用sklearn生成预测结果的混淆矩阵。

**代码实现：**

代码整体逻辑结构分为两层：

----text\_classification.py 父层，选择分类器类型及参数

----knn.py 子层，KNN算法实现

----naïve\_bayes.py 子层，NB算法实现

----util.py 子层，其他工具函数

knn.py和naive\_bayes.py的结构类似，KNN类和Naïve\_bayes类，都包括三个必要函数，分别为\_\_init\_\_，fix，predict。分别实现分类器的初始化、训练和预测。

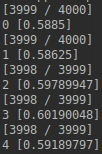
训练函数fix接受形如(X, y)的输入数据，其中X，y均为numpy数组，大小均为(n, 1)，n为输入数据个数。

预测函数predict接受形如(X)的输入数据，X为大小为(n, 1)的numpy数组，返回结果尺寸与输入数据相同，n同为输如数据的个数。

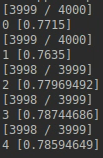
数据集自动下载、数据集扫描及数据集加载等工作由util.py完成，load\_data函数返回符合fix和predict输入要求的数据内容。

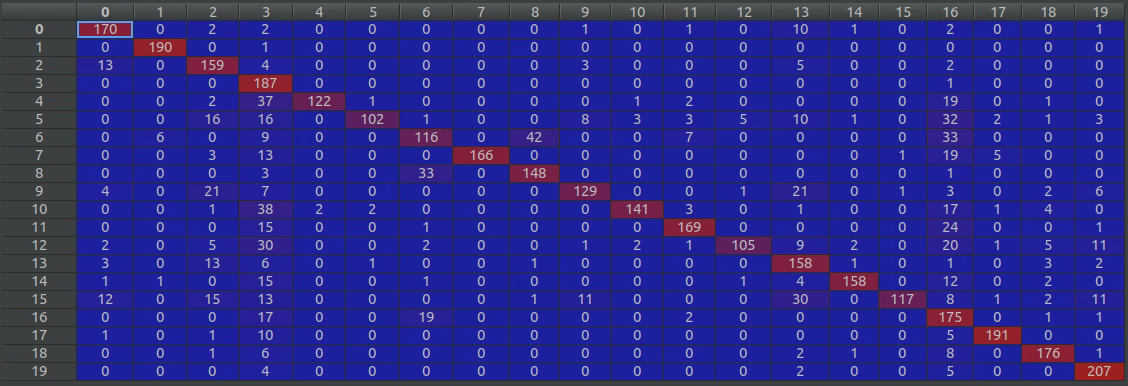
1. **实验结果**

在初步的实验中，KNN算法也采用n-fold的方式交叉验证，因为我自己的knn代码实现完全采用python完成，预测一个数据大约需要2s的时间，完成一次n=5的n-fold的时间过长。初始设置超参数k=20，以下是完成5次交叉验证的结果，**平均准确率约为59.32%**，结果尚可接受。单论交叉验证大约用时2h，5轮共用时10h，计算时间过长。



同样采用n-fold对NB分类器进行测试，**平均准确率约为77.76%**，分类准确率优于KNN分类器。且NB分类器的预测速度远高于KNN，完成5轮的n-fold测试仅需两分钟左右。



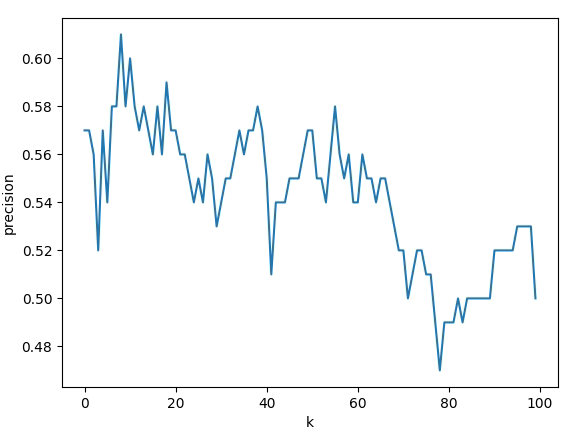


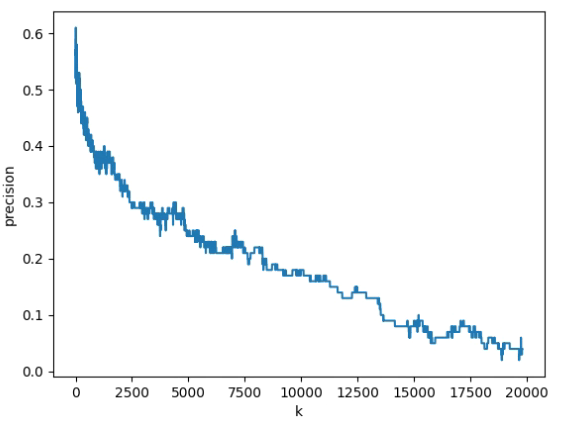
在KNN算法中，超参数k的设置直接影响到分类器的分类效果，因此采用保留测试（Hold out test）的方式选择合适的k值。

在KNN的代码实现过程中，其实我尝试了两种不同的算法。其中基于numpy的矩阵方式，虽然计算速度更慢，但一次计算就可以得到预测数据和所有的训练数据的距离排序，非常适合用于超参数k的选择。

因为已经得到了相对训练数据的距离排序，只要按此顺序，逐个将训练数据加入Counter中进行记数，即可得到不同k值时，预测的准确度。而无需对不同的k值，重复预测过程。

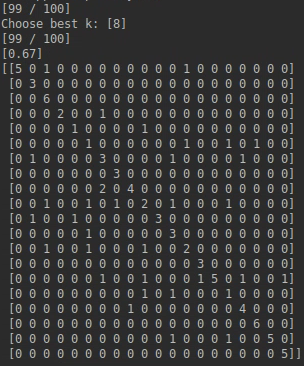
将原始数据分为训练集、验证集、测试集三个部分，因为KNN计算非常耗时，因此设置相对较小的验证集和测试集比例，三者的比例为0.99：0.005：0.005。即验证集和测试集各100个数据样本。





前期随着k的增大，分类精度增加；后期随着k继续增大，分类精度下降。结果完全符合预期。

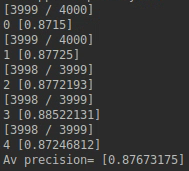
使用训练集生成模型之后，通过验证集的预测结果选择合适的超参数k，并最终使用该k值测试在测试集上的分类准确率，并计算混淆矩阵，结果如下：



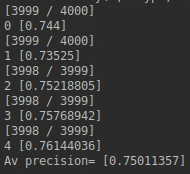
**经过超参数k的选择之后，分类精度达到67%，相较前述人工选取超参数k时约59.32%的分类精度，KNN分类器的效果有明显的提升。**

2021.9.26 更新

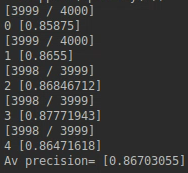
最后对数据预处理部分进行优化。使用nltk包，删除数据中的停用词并进行词干化，在其他与前述NB算法相同，并使用相同的随机种子的情况下，优化效果如下：



仅删除停用词，正确率提升至约87.67%。



仅使用词干化，平均正确率为75.01%，分类精度不如之前不使用时



同时使用停用词和词干化处理，平均正确率86.70%，不如但使用停用词处理。

1. **遇到问题及解决思路**

KNN实现的速度太慢，初始使用Python内建\*\*函数实现乘方开方，heapq实现的距离排序；改为使用numpy矩阵计算完成距离计算，sort函数快排实现距离排序。修改之后发现速度反而大幅度下降。

但这种基于矩阵的计算方法，可以一次性得到，预测数据和所有训练数据的排序关系。该结果可以用于KNN超参数的筛选，只要计算一次之后，不断增加入选的训练数据数量即可，不用对每一个k都重新进行整个计算。