基于朴素贝叶斯的电影评论情感分类

实验手册

**刘东洋**

1. 实验目标

使用朴素贝叶斯算法构建电影评论的情感分类模型,实现对正面和负面评论的分类，并研究超参数对模型准确率的影响。

1. 实验过程
   1. 环境配置

本实验在PyCharm上运行，同时需要安装TensorFlow框架和sklearn工具包。

其中TensorFlow的安装前面小组已经有详细的介绍，没有安装的同学可以移步去相应的实验手册。

sklearn工具包的安装：pip install scikit-learn

本次实验所需要的库：

|  |
| --- |
| from tensorflow.keras.datasets import imdb  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.preprocessing.sequence import pad\_sequences  import numpy as np  import matplotlib.pyplot as plt  from sklearn.feature\_extraction.text import CountVectorizer  from sklearn.metrics import accuracy\_score, precision\_score, f1\_score |

* 1. 数据集

本实验采用Keras内置的IMDB电影评论数据集(即:from tensorflow.keras.datasets import imdb)。该数据集包含50,000条电影评论,其中25,000条为正面评论,25,000条为负面评论。数据集已经过预处理：将每条评论的具体单词序列转化为词库里的整数序列，其中每个整数代表该单词在词库里的位置。例如，整数104代表该单词是词库的第104个单词。为实验简单词库仅仅保留了10000个最常出现的单词，低频词汇被舍弃。每条评论都具有一个标签，0表示为负面评论，1表示为正面评论。

|  |
| --- |
|  |

我们可以尝试将已经处理过的整数样本转化为影评文本,代码见Dataset2.py

|  |
| --- |
|  |

* 1. 数据集的加载和预处理

以上功能在函数def preprocessing(train\_ratio, max\_features):中实现。

其中train\_ratio是训练集所占比例，max\_features是词袋模型的最大特征数，也是后面要研究对于模型准确率的影响的两个超参数。

从Keras中加载IMDB数据集。由于IMDB数据集已经预处理成整数编码表示,每个样本是一个整数列表,代表词汇表中的单词索引，且数据集自动划分为训练集和测试集两个部分。返回值是：

train\_data:训练集样本数据,每个样本是一个整数列表。

train\_labels:训练集样本对应的标签,0为负面评论,1为正面评论。

test\_data:测试集样本数据,与训练集格式一致。

test\_labels:测试集样本对应的标签。

|  |
| --- |
| # 加载IMDB电影评论数据集  (train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words=num\_words) |

样本1：加载后打印

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

但是为了后续的预处理，需要先把划分好的训练集和测试集合并。

这样处理会"消除"他们作为不同子集的身份标识。然后再按比例划分,就保证了:训练和测试情景一致,测试集依然对模型完全"陌生"。

|  |
| --- |
| # 合并训练集和测试集以进行预处理  data = np.concatenate((train\_data, test\_data), axis=0)  labels = np.concatenate((train\_labels, test\_labels), axis=0) |

IMDB数据集本身划分了50%比例的训练集，但是为了后续研究训练集比例对准确率的影响，以及避免测试集信息泄露，需要合并原来的训练集和测试集，重新划分比例。

这里数据经历了整数序列-字符串-向量整数序列的转化。

原数据集的整数序列需要转换为字符串，经过fit\_on\_texts()函数提取词汇特征，再经过texts\_to\_sequences()函数转化为便于后续模型输入和词袋模型构建的向量序列。

分词前是按照原文本顺序排列的字符串。

分词后是根据词汇表统计出来的词频顺序排列的整数序列。

样本1：将整数序列转换为字符串后打印

1 14 22 16 43 530 973 1622 1385 65 458 4468 66 3941 4 173 36 256 5 25 100 43 838 112 50 670 2 9 35 480 284 5 150 4 172 112 167 2 336 385 39 4 172 4536 1111 17 546 38 13 447 4 192 50 16 6 147 2025 19 14 22 4 1920 4613 469 4 22 71 87 12 16 43 530 38 76 15 13 1247 4 22 17 515 17 12 16 626 18 2 5 62 386 12 8 316 8 106 5 4 2223 5244 16 480 66 3785 33 4 130 12 16 38 619 5 25 124 51 36 135 48 25 1415 33 6 22 12 215 28 77 52 5 14 407 16 82 2 8 4 107 117 5952 15 256 4 2 7 3766 5 723 36 71 43 530 476 26 400 317 46 7 4 2 1029 13 104 88 4 381 15 297 98 32 2071 56 26 141 6 194 7486 18 4 226 22 21 134 476 26 480 5 144 30 5535 18 51 36 28 224 92 25 104 4 226 65 16 38 1334 88 12 16 283 5 16 4472 113 103 32 15 16 5345 19 178 32

样本1：分词后打印

[30, 12, 20, 14, 41, 515, 1055, 1469, 1307, 66, 461, 4163, 64, 4050, 1, 175, 35, 255, 3, 23, 100, 41, 818, 111, 48, 612, 2, 7, 34, 493, 293, 3, 148, 1, 169, 111, 166, 2, 363, 386, 38, 1, 169, 4864, 940, 15, 540, 36, 11, 433, 1, 194, 48, 14, 4, 146, 1908, 17, 12, 20, 1, 1917, 5216, 476, 1, 20, 72, 80, 10, 14, 41, 515, 36, 74, 13, 11, 1227, 1, 20, 15, 528, 15, 10, 14, 615, 16, 2, 3, 60, 379, 10, 6, 306, 6, 105, 3, 1, 2048, 5962, 14, 493, 64, 3591, 32, 1, 129, 10, 14, 36, 627, 3, 23, 120, 50, 35, 133, 45, 23, 1437, 32, 4, 20, 10, 208, 26, 77, 51, 3, 12, 408, 14, 83, 2, 6, 1, 107, 122, 5734, 13, 255, 1, 2, 5, 3327, 3, 842, 35, 72, 41, 515, 468, 24, 405, 314, 43, 5, 1, 2, 1013, 11, 103, 86, 1, 407, 13, 290, 95, 31, 2081, 55, 24, 140, 4, 193, 7193, 16, 1, 225, 20, 19, 134, 468, 24, 493, 3, 144, 27, 6216, 16, 50, 35, 26, 223, 91, 23, 103, 1, 225, 66, 14, 36, 1322, 86, 10, 14, 291, 3, 14, 4542, 116, 102, 31, 13, 14, 5474, 17, 178, 31]

|  |
| --- |
| # 将整数序列转换为字符串  data = [' '.join([str(word) for word in sample]) for sample in data]  # 对文本进行分词  tokenizer = Tokenizer(num\_words=num\_words)  tokenizer.fit\_on\_texts(data)  sequences = tokenizer.texts\_to\_sequences(data) |

接下来需要对序列进行长度统一，即填充和截断。长度统一更方便后续处理。

样本1：填充后打印

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 30 12 20 14

41 515 1055 1469 1307 66 461 4163 64 4050 1 175 35 255

3 23 100 41 818 111 48 612 2 7 34 493 293 3

148 1 169 111 166 2 363 386 38 1 169 4864 940 15

540 36 11 433 1 194 48 14 4 146 1908 17 12 20

1 1917 5216 476 1 20 72 80 10 14 41 515 36 74

13 11 1227 1 20 15 528 15 10 14 615 16 2 3

60 379 10 6 306 6 105 3 1 2048 5962 14 493 64

3591 32 1 129 10 14 36 627 3 23 120 50 35 133

45 23 1437 32 4 20 10 208 26 77 51 3 12 408

14 83 2 6 1 107 122 5734 13 255 1 2 5 3327

3 842 35 72 41 515 468 24 405 314 43 5 1 2

1013 11 103 86 1 407 13 290 95 31 2081 55 24 140

4 193 7193 16 1 225 20 19 134 468 24 493 3 144

27 6216 16 50 35 26 223 91 23 103 1 225 66 14

36 1322 86 10 14 291 3 14 4542 116 102 31 13 14

5474 17 178 31]

|  |
| --- |
| # 显示分词后的单词数量  word\_count = [len(s.split()) for s in data]  # 对序列进行填充和截断  padded\_sequences = pad\_sequences(sequences, maxlen=max\_length)  # 显示填充前后的序列长度  seq\_len = [len(s) for s in sequences] |

根据设置的超参数训练比例train\_ratio,计算划分点split的位置。将padded\_sequences序列特征和labels标签,从划分点split开始分成两部分。前split部分作为新的训练集,后部分作为测试集。

|  |
| --- |
| # 划分训练集和测试集  split = int(50000 \* train\_ratio / 100)  train\_data = padded\_sequences[:split]  test\_data = padded\_sequences[split:]  train\_labels = labels[:split]  test\_labels = labels[split:] |

划分好训练集和测试集之后，进一步利用词袋模型处理数据，得到更适合训练模型的数据。

朴素贝叶斯基于词袋假设,两个模型匹配程度高，通过词频向量可以很好地将文本标准化为算法可以直接处理的特征矩阵形式。

这里又经历了整数序列-文本序列-字符串数据-稀疏词频向量的转化，目的是为了得到适合使用词袋模型的数据形式，以及最终输出便于训练朴素贝叶斯模型的数据。

样本1：

将文本序列转换为字符串后打印

1 14 22 16 43 530 973 1622 1385 65 458 4468 66 3941 4 173 36 256 5 25 100 43 838 112 50 670 2 9 35 480 284 5 150 4 172 112 167 2 336 385 39 4 172 4536 1111 17 546 38 13 447 4 192 50 16 6 147 2025 19 14 22 4 1920 4613 469 4 22 71 87 12 16 43 530 38 76 15 13 1247 4 22 17 515 17 12 16 626 18 2 5 62 386 12 8 316 8 106 5 4 2223 5244 16 480 66 3785 33 4 130 12 16 38 619 5 25 124 51 36 135 48 25 1415 33 6 22 12 215 28 77 52 5 14 407 16 82 2 8 4 107 117 5952 15 256 4 2 7 3766 5 723 36 71 43 530 476 26 400 317 46 7 4 2 1029 13 104 88 4 381 15 297 98 32 2071 56 26 141 6 194 7486 18 4 226 22 21 134 476 26 480 5 144 30 5535 18 51 36 28 224 92 25 104 4 226 65 16 38 1334 88 12 16 283 5 16 4472 113 103 32 15 16 5345 19 178 32

样本1：将文本序列转换为字符串后打印

1 14 22 16 43 530 973 1622 1385 65 458 4468 66 3941 4 173 36 256 5 25 100 43 838 112 50 670 2 9 35 480 284 5 150 4 172 112 167 2 336 385 39 4 172 4536 1111 17 546 38 13 447 4 192 50 16 6 147 2025 19 14 22 4 1920 4613 469 4 22 71 87 12 16 43 530 38 76 15 13 1247 4 22 17 515 17 12 16 626 18 2 5 62 386 12 8 316 8 106 5 4 2223 5244 16 480 66 3785 33 4 130 12 16 38 619 5 25 124 51 36 135 48 25 1415 33 6 22 12 215 28 77 52 5 14 407 16 82 2 8 4 107 117 5952 15 256 4 2 7 3766 5 723 36 71 43 530 476 26 400 317 46 7 4 2 1029 13 104 88 4 381 15 297 98 32 2071 56 26 141 6 194 7486 18 4 226 22 21 134 476 26 480 5 144 30 5535 18 51 36 28 224 92 25 104 4 226 65 16 38 1334 88 12 16 283 5 16 4472 113 103 32 15 16 5345 19 178 32

词袋表示后打印

(0, 44) 3

(0, 132) 6

(0, 66) 11

(0, 363) 4

(0, 456) 3

(0, 477) 2

(0, 394) 1

(0, 478) 2

(0, 81) 1

(0, 286) 4

(0, 172) 2

(0, 165) 4

(0, 1) 1

(0, 14) 2

(0, 436) 2

(0, 275) 1

(0, 418) 3

(0, 203) 1

(0, 56) 1

(0, 80) 2

(0, 74) 1

(0, 260) 1

(0, 314) 1

(0, 319) 1

(0, 77) 3

: :

(24999, 502) 1

(24999, 187) 3

(24999, 30) 1

(24999, 509) 2

(24999, 472) 1

(24999, 154) 1

(24999, 188) 1

(24999, 63) 1

(24999, 433) 1

(24999, 484) 1

(24999, 17) 1

(24999, 149) 1

(24999, 13) 1

(24999, 134) 1

(24999, 10) 1

(24999, 312) 1

(24999, 2) 1

(24999, 115) 3

(24999, 170) 1

(24999, 120) 1

(24999, 153) 1

(24999, 261) 1

(24999, 163) 1

(24999, 226) 1

(24999, 336) 2

这个输出结果表示的是词袋模型转换后的稀疏矩阵表示。

可以解释为:

- 每一行表示一个样本文档

- 列索引表示词汇表中的词语索引号

- 值表示该词在该样本中出现的频率

例如:

(0, 44) 3

表示样本0中词汇表中索引为44的词出现了3次。

一般大多数样本中只出现少量词,所以转换后的稀疏矩阵很大部分值都为0,这就是稀疏矩阵的特征。

通过这样的稀疏词频表示,我们把文本转化为机器易于处理的数值向量,去除了文本本身的顺序和结构性信息,侧重考察词汇出现的统计规律,这就符合朴素贝叶斯分类模型的假设。

|  |
| --- |
| # 将整数序列转换回文本序列  train\_text = tokenizer.sequences\_to\_texts(train\_data)  test\_text = tokenizer.sequences\_to\_texts(test\_data)  # 将文本序列转换为字符串数组  train\_text = np.array(train\_text)  test\_text = np.array(test\_text)  # 创建一个词袋模型  vectorizer = CountVectorizer(max\_features=max\_features)  # 将整数序列转换为词袋向量表示  train\_features = vectorizer.fit\_transform(train\_text)  test\_features = vectorizer.transform(test\_text)  # 使用词袋特征进行分类  return train\_features, train\_labels, test\_features, test\_labels |

至此，数据集加载和预处理部分完成。

* 1. 模型的训练和准确率指标的计算

以上功能在函数calculate\_indicators(train\_features, train\_labels, test\_features, test\_labels):中实现。

其中train\_features, train\_labels, test\_features, test\_labels由preprocessing函数返回得到。

首先定义class NaiveBayesClassifier:

首先定义类属性self.alpha,用于存储拉普拉斯平滑参数α的值。默认为1。

初始化类属性self.class\_probabilities和self.feature\_probabilities为None。

这两个属性分别将存储类别先验概率和特征条件概率分布,是模型训练的主要输出。

|  |
| --- |
| def \_\_init\_\_(self, alpha=1):  self.alpha = alpha  self.class\_probabilities = None  self.feature\_probabilities = None |

朴素贝叶斯模型需要知道每个类别的先验概率和后验概率来计算后验概率。

使用np.bincount统计每个类别的样本数class\_counts,求出总样本数total\_samples

计算每个类的先验概率公式:

(样本数+α) / (总样本数+α\*类别数)

这里使用了拉普拉斯平滑,即加入小值α来避免某些类别概率为0。

|  |
| --- |
| # 计算先验概率  def calculate\_class\_probabilities(self, y\_train):  class\_counts = np.bincount(y\_train)  total\_samples = y\_train.shape[0]  class\_probabilities = (class\_counts + self.alpha) / (total\_samples + self.alpha \* len(class\_counts))  return class\_probabilities |

初始化一个全0数组来存储条件概率结果。

对每个类别:取出这个类别的所有样本,统计每个特征的计数总和,计算条件概率公式:(特征值计数+α) / (样本总计数+α\*特征总数),将每个类别的条件概率填入预先准备的结果数组,返回最终条件概率矩阵。

|  |
| --- |
| # 计算条件概率  def calculate\_feature\_probabilities(self, X\_train, y\_train):  num\_classes = len(np.unique(y\_train))  num\_features = X\_train.shape[1]  feature\_probabilities = np.zeros((num\_classes, num\_features))  for class\_label in range(num\_classes):  class\_samples = X\_train[y\_train == class\_label]  total\_count = class\_samples.sum()  feature\_probabilities[class\_label] = (class\_samples.sum(axis=0) + self.alpha) / (  total\_count + self.alpha \* num\_features)  return feature\_probabilities |

拟合朴素贝叶斯模型的fit()函数

该函数在训练数据上拟合朴素贝叶斯模型,学习模型参数，主要参数就包括先验概率和条件概率。记录学习得出的概率分布,为后续预测提供依据。

|  |
| --- |
| def fit(self, X\_train, y\_train):  # 计算先验概率,概率可输出观察  self.class\_probabilities = self.calculate\_class\_probabilities(y\_train)  # 计算条件概率  self.feature\_probabilities = self.calculate\_feature\_probabilities(X\_train, y\_train) |

接下来是predict()函数，该函数利用一个训练好的朴素贝叶斯模型,来为测试数据集X\_test进行预测。它遍历每个测试样本,并计算该样本属于每个类别的后验概率分数。这里利用了贝叶斯分类法中的公式, calculate posterior probability for each class然后利用np.argmax返回后验概率最大的那类,作为该样本的预测类别。结果存入predictions数组中,这个数组中的元素类型是整数,表示类别编号。

|  |
| --- |
| # 利用已训练好的朴素贝叶斯模型进行预测  def predict(self, X\_test):  # 根据类的个数和测试样本数,初始化预测结果数组  num\_classes = len(self.class\_probabilities)  num\_samples = X\_test.shape[0]  predictions = np.zeros(num\_samples)  # 计算该样本在每个类中的后验概率分数  for i in range(num\_samples):  sample = X\_test[i]  class\_scores = np.log(self.class\_probabilities) + np.log(  self.feature\_probabilities[:, sample.nonzero()[1]]).sum(axis=1)  predictions[i] = np.argmax(class\_scores)  return predictions |

至此，class NaiveBayesClassifier中的所有函数编写完成。

接下来调用这些函数:

定义NaiveBayesClassifier类的实例NB

调用NB的fit方法,使用训练数据训练模型,学习参数

调用NB的predict方法,使用训练好的模型进行预测,得到预测结果y\_predict

调用sklearn中的metrics来计算:

accuracy:预测正确率,测试集预测结果与真实标签一致的比例(示例代码中只返回了accuracy)

precision:精确率,正确预测正例的比例

f1:F1 score,考虑准确率和召回率的综合指标

|  |
| --- |
| NB = NaiveBayesClassifier()  NB.fit(train\_features, train\_labels)  y\_predict = NB.predict(test\_features)  accuracy = accuracy\_score(test\_labels, y\_predict)  precision = precision\_score(test\_labels, y\_predict)  f1 = f1\_score(test\_labels, y\_predict)  return accuracy |

至此，朴素贝叶斯模型的构建、训练，和分类效果指标的计算都已经完成。

* 1. 分析数据集划分比例和词袋最大特征数对分类效果的影响

设置不同的训练集验证集比例(如50%,60%等)

对每个比例:

分割数据集为训练和测试集

对训练和测试集分别进行预处理,构建词袋特征

训练朴素贝叶斯模型

计算模型在测试集上的准确率

绘制训练集比例和准确率曲线图

|  |
| --- |
| # 数据集划分对结果影响  plt.subplot(2, 2, 1)  train\_ratio = [50, 60, 70, 80, 90]  for i in range(5):  train\_features, train\_labels, test\_features, test\_labels = preprocessing(train\_ratio[i], 512)  accuracy = calculate\_indicators(train\_features, train\_labels, test\_features, test\_labels)  plt.scatter(train\_ratio[i], accuracy)  plt.xlabel("train ratio %")  plt.ylabel("accuracy")  plt.show() |

Output:

Train Ratio: 10 Accuracy: 0.8078666666666666

Train Ratio: 20 Accuracy: 0.810775

Train Ratio: 30 Accuracy: 0.8106

Train Ratio: 40 Accuracy: 0.8127

Train Ratio: 50 Accuracy: 0.813

Train Ratio: 60 Accuracy: 0.8123

Train Ratio: 70 Accuracy: 0.8140666666666667

Train Ratio: 80 Accuracy: 0.8168

Train Ratio: 90 Accuracy: 0.8188

Train Ratio: 95 Accuracy: 0.8272

整体来看,随着训练集比例从10%增加到95%,分类准确率也在整体上升,表明更多训练数据能提高模型性能;

在训练集比例低于50%时,每增加10%训练集,准确率提升幅度相对较小,主要在0.0013范围内波动;

训练比例从50%开始继续增加到95%,准确率整体呈上升趋势。重要的是,此时每次增加的准确率提升幅度开始增大:

60-70%间提升幅度最大,约0.0017

80-90%间提升幅度下降但保持在0.0008-0.0015范围

最明显的是,90-95%这一步,准确率提升幅度最大,高达0.0084

通过对比不同阶段的准确率提升幅度,我们可以看出:随着更多训练数据的增多,从50%开始每次增加的影响量呈逐步增大趋势,至90-95%时影响最大。

|  |
| --- |
| Figure_4 |

词袋最大特征数目对特征的影响

1. 设置不同的词袋模型最大特征数(如256,512等)

2. 对每个特征数:

- 设置训练集测试集比例为50%

- 对数据集进行词袋预处理,构建特征

这里的最大特征数代表词袋中保留词频最高的特征数目

- 训练模型并计算测试准确率

3. 绘制最大特征数和准确率关系图

目的在于:

- 探索词袋模型最大特征数如何影响结果

- 特征数越多,可以捕捉的文本信息就越全面

- 但特征数过多会增加模型复杂度和过拟合现象

- 寻找一个平衡特征数能获得较好效果

所以这与上一部分类似,通过调整不同超参数,分析其对结果的影响,

帮助找到词袋建模时一个合适的最大特征数设置。

|  |
| --- |
| # 词袋最大特征数目对结果影响  plt.subplot(2, 2, 2)  max\_features = [256, 512, 1024, 2048, 4096]  for i in range(5):  train\_features, train\_labels, test\_features, test\_labels = preprocessing(50, max\_features[i])  accuracy = calculate\_indicators(train\_features, train\_labels, test\_features, test\_labels)  plt.scatter(max\_features[i], accuracy)  plt.xlabel("max features")  plt.ylabel("accuracy")  plt.show() |

Output:

|  |
| --- |
| Max Features: 256 Accuracy: 0.75244  Max Features: 512 Accuracy: 0.813  Max Features: 1024 Accuracy: 0.83568  Max Features: 2048 Accuracy: 0.84572  Max Features: 4096 Accuracy: 0.84844  Max Features: 8192 Accuracy: 0.84692  Max Features: 10000 Accuracy: 0.8462  Max Features: 15000 Accuracy: 0.8462  Max Features: 20000 Accuracy: 0.8462  Max Features: 30000 Accuracy: 0.8462  Figure_5  Figure_6 |

1. 实验结果

实验验证了朴素贝叶斯算法可以很好地实现电影评论情感分类这一文本分类任务。利用词频特征表示文本,采用贝叶斯训练分类模型预测 sentiments,取得了较高的分类精度。

训练集比例对结果影响很大。低于50%时每增加10%训练数据效果明显提高;50%-90%间效果越来越明显,尤其是90%-95%间提升最明显。但超过一定比例后,模型可能会过拟合。

词袋大小也是影响重要参数。适度增加词汇表可以表达样本信息更完整,但过大则会带来数据冗余和噪声。实验表明,15000特征点能取得较优成绩。

停用词和词性还原预处理可以去除离散信息,提取关键词,有利提高模型分析文本的能力。这对某些任务也许并非必要,但对情感分类效果确实有一定正面作用。

本实验结果可为将贝叶斯模型应用于其他文本分类任务提供参考。同时,也可以考虑结合深层学习模型 ，比如RNN/CNN来挖掘文本中的结构信息,如句子结构、上下文关系等,这可能有助于进一步提升分类效果。