第 3 章

图书类目自动标引系统

21 世纪以来，随着信息资源量的不断增长，世界各地的图书馆普遍使用大量数字资源进行数字化建设，如何对数字资源进行加工整理成为数字化图书馆建设的重要方向之一。为了使数字资源像纸质文献一样能够被快速按类别进行检索，数字资源也需要进行标引。

无论是纸质资源还是数字资源，其分类都不是与生俱来的，图书文献的标引人员需要经过培训，即使是经验丰富的图书标引人员也要根据纸质资源或数字资源的主要内容，参照《中图分类法》的分类规则进行分类标引。目前数字资源在图书馆馆藏资源中所占的比例已经越来越大，数字资源的标引工作也变得越来越重要，如何在数字资源种类和规模都在迅速增长的情况下仍然兼顾标引的质量和速度，是任何一个数字化图书馆都不可忽视的重要项目。

3.1 业务背景分析

目前对于图书馆收录的数字资源，大部分图书馆仍然在采取人工分类的方式对数字资源进行标引，这种方法需要经验非常丰富的标引人员耗费大量时间才能完成。因此数字资源的自动标引方法不仅可以节省人力和财力，而且还能够大大提高数字资源标引的速度，缩短资源上架周期，被读者更好地利用，有利于知识的传播。而目前图书馆所能够使用的数字资源自动标引系统均较为陈旧，其算法依赖词表和知识库的构建，且并未使用近年来机器学习和自然语言处理领域的最新成果。这些系统的标引准确率低下，且对于部分数字资源需要人工参与进行协助分类或者检验，并不能从真正意义上解放人力资源，达不到自动标引的要求。而近年来快速发展的基于机器学习和自然语言处理的算法，并没有在数字资源标引系统上有效应用。

3.2 数据提取

这里将使用某市图书馆提供的 F 经济大类馆藏数字资源作为语料素材。数字资源的文献标题、期刊或会议名称、作者、单位、时间、文献摘要和作者给出的关键词组成了全部数字资源的索引数据库部分,而数字资源的全文则以二进制大文件的形式单独进行存储。

由于多数字段空值比例较高,从中选择部分字段作为机器标引的输入特征,经过筛选,选择标题、出版社、关键词、摘要作为后续分类标引的依据,如图 3.1 所示。

	title	publisher	pubcode	keywords	category	abstract
0	论清末湖北地方政府军政外债的影响	沈阳工程学院学报:社科版	46272	/清后期/财政史/湖北/外债	F812.952	清末,湖北地方政府为弥补财政亏空举借了各种用途的军政外债共10笔,这些外债对债权方和湖北地区...
1	中心城区区域经济发展模式的选择与分析	沈阳工程学院学报:社科版	46272	/城市经济/经济发展/中国	F299.2	随着经济持续发展,各大城市中心城区的经济发展模式日渐成熟,由于中心城区发展的特殊性和广泛的示...
2	中国股市高投机性的制度机理研究	沈阳工程学院学报:社科版	46272	/股票市场/制度/中国	F832.5	投机性是市场所具有的天然属性.然而,中国股市投资者的过度投机行为在市场波动中所反映出来的极端...
3	西部农村减缓贫困的进展	中国农村观察	4276	/地方农业经济/中国	F327	本文在对改革初期中国西部农村贫困特点回顾的基础上,综述了改革开放以来西部地区的反贫困措施。经...
4	交易成本与中国农村的基础设施治理结构选择:以灌溉、电力、公路和饮用水设施为例	中国农村观察	4276	/农业建设/农业经济发展/中国	F323	本文运用交易成本理论指出,农村基础设施治理的最优化是其有效供给的充分条件,不同规模设施的最优...

图 3.1 待标引文献数据示例

图书馆提供的初始数据库文件为 Access 数据库,文件类型为 mdb,一共有 74 万条样本数量。首先安装 Access 数据驱动以及 pyobdc 工具包,连接 Access 数据库并将数据导出为 csv 文件。在 Windows 系统上运行以下代码:

```
import pyodbc
print([x for x in pyodbc.drivers() if x.startswith('Microsoft Access Driver')])
```

如果看到一个空列表,那么正在运行 64 位 Python,并且需要安装 64 位版本的 ACE 驱动程序。如果只看到['Microsoft Access Driver (*.mdb)']并且需要使用.accdb 文件,那么需要安装 32 位版本的 ACE 驱动程序。

数据提取部分的代码见 extract.py,其中没有抽取原本数据库中全部的字段,只使用了对于分类最重要的几个字段,即正文地址、target、title、abstract、keyword。

```
import pyodbc
import csv

path = 'D:\\data\\'
cnxn = pyodbc.connect(r'DRIVER={Microsoft Access Driver (*.mdb, *.accd b)};DBQ=' + path
+ 'F 大类 08 到 18 年数据.mdb')
crsr = cnxn.cursor()
for table_info in crsr.tables(tableType='TABLE'):
    print(table_info.table_name)

rows = crsr.execute("SELECT Fulltext_store_path, attribute_string_14, attribute_string_1, a
ttribute_string_13, attribute_text_1 FROM F 数据")

csv_writer = csv.writer(open('F08_18.csv', 'w', newline='', encoding='utf8'))
```

```
for row in rows:
    list = []
    for item in row:
        if item != None:
            list.append(item)
        else:
        list.append('')
    csv_writer.writerow(list)
```

其中，首先读取所有表的名称，然后再执行 SQL 游标查询(crsr.execute)，逐行读取并将其写到文本文件中(csv.writer)。

如果是苹果操作系统，需要通过 Homebrew 安装 unixodbc，安装方法为 brew install unixodbc，然后安装 mdbtools(brew install mdbtools)，使用命令“mdb-export F 大类 08 到 18 年数据.mdb 'F08-18 数据'> output_file.csv”即可导出为 csv 格式。

3.3　数据预处理

对数据进行分析后发现约有 5%的文献关键词缺失，约有 20%的文献摘要缺失，仅有约 30%的文献存在正文部分。

对数据中的文献标题、摘要使用 jieba 分词，并删去对分类显然没有帮助的词性。对数据中作者给出的关键词不做处理。使用 jieba 分词的示例方法如下：

```
import jieba
import jieba.posseg as pseg
abstract = getAbstract()                    # 文献摘要
words = pseg.cut(abstract)
for word, flag in words:
    print('%s %s' % (word, flag))
```

对于分词后的词语，在停用词表中进行搜索，删去纯数字以及在停用词表中的词。停用词是指在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据之前或之后会自动过滤掉某些字或词，这些字或词被称为停用词。对中文文本分类任务来说大部分是助词、副词、介词、连接词，本身无实际含义。预处理部分的代码见 pre.py，经预处理后得到 F08_18_pre.csv。

然后对数据中的文献正文部分，使用中文维基语料库训练 N-Gram 模型，训练语言模型的代码如下：

```
import pickle
file = open('ngram_char.txt', 'r', encoding = 'utf8')
dict1 = {}
dict2 = {}
num = 0
for line in file:num += 1
    if num % 10000 == 0:
    print(num)
words = line.strip().split(' ')
```

```
for i in range(len(words)):
    word = words[i]
    if word not in dict1:
        continue
    dict1[word] += 1
for i in range(1, len(words)):
    word1 = words[i-1]
    word2 = words[i]
    if (word1, word2) not in dict2:
        dict2[(word1, word2)] = 1
        continue
    dict2[(word1, word2)] += 1
picklestring1 = pickle.dump(dict1, open('ngram1.pkl', 'wb'), pickle.HIGH EST_PROTOCOL)
picklestring2 = pickle.dump(dict2, open('ngram2.pkl', 'wb'), pickle.HIGH EST_PROTOCOL)
dict1[word] = 1
```

使用语言模型过滤后,尝试提取其中对分类有帮助的词。使用语言模型过滤的代码如下:

```
def is_sentence(s):
    global charList, dict1, dict2
    p = 0
    words_cut = jieba.cut(s)
    words = ['<b>']
    for item in words_cut:
        if item not in charList and item != '「':
           words.append(item)
           words.append('<e>')
    for i in range(1, len(words)-1):
        if (words[i-1], words[i]) not in dict2:
            num1 = 1
        else:
            num1 = dict2[(words[i-1], words[i])] + 1
    if words[i] not in dict1:
        num2 = len(dict2)
    else:
        num2 = dict1[words[i]] + len(dict2)
    p = p + math.log(num1 / num2, 10)
    print(''.join(words), '', s, '', p)
    #设置阈值
    if p > -6 * len(words):
        return True
    else:
        return False
```

然后使用过滤后的正文部分提取关键词,实现过程包括以下几步:

(1) 利用提供的关键词构建关键词表,在文献正文中进行搜索,选择出现次数大于阈值的词,加入到数据集中。关键代码如下:

```
keywords = ''
for keyword in keyword_dict.keys():
```

```
c = content.count(keyword)
#设置阈值
if c > 5:
    keywords += '' + keyword
```

(2) 使用 TF-IDF 算法从文献正文提取关键词,加入到数据集中。使用 jieba 的 TF-IDF 的关键词提取方法如下:

```
import jieba
import jieba.analyse
sentence = getText()                          #文献全文部分
keywords = jieba.analyse.extract_tags(text, withWeight = True)
for item in keywords:
    print(item[0],item[1])
```

最后将所有数据打乱顺序后分为训练集、验证集和测试集,其中训练集占 90%,验证集和测试集各占 5%。

对 F 大类下的二级分类数量进行可视化,结果如图 3.2 所示。

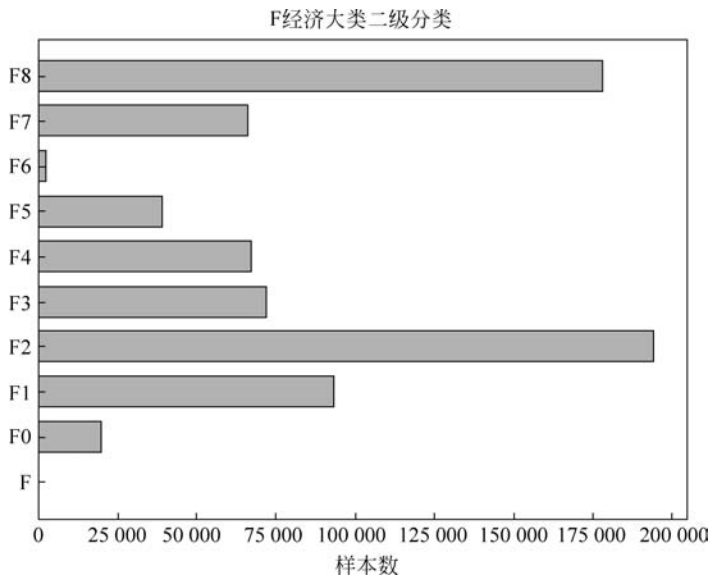

图 3.2　F 大类中二级类目样本数量

给定的数据存在样本不平衡问题,例如在 F 大类二级分类中出现的 F8 财政、经济类有 17 万多条数据,而出现最少的 F6 邮电经济仅有两千多条数据。为了增加出现较少类别数据的数据量,同时增加噪声防止过拟合并提升泛化能力。对出现次数较多类别的数据进行随机的欠采样,且在对某条数据进行欠采样时,随机删去该条数据的部分词语。这种随机删除训练集中词的方法相当于从数据源头采集了更多的数据,也可以防止过拟合。

然后,考虑使用词向量进行消歧,使用 gensim 训练词向量的方法如下:

```
import os
from gensim.models import word2vec
```

```
print("word2vec 模型训练中...")
#加载文件
sentence = word2vec.Text8Corpus('wiki_segmented.txt')
#训练模型
model = word2vec.Word2Vec(sentence, size=400, window=5, min_count=5, work ers=4, sg=1)
#保存模型
model.save('models/wiki.zh.text.model')
model.wv.save_word2vec_format('models/wiki.zh.text.vector', binary=False)
print("Word2vec 模型已存储完毕")
```

将文献的标题、作者给出的关键词、摘要三部分分别使用 TF-IDF 提取特征后，再使用贝叶斯分类进行训练和测试，准确率如表 3.1 所示。

表 3.1　不同数据来源准确率

数 据 来 源	Acc(准确率)
标题	0.71%
关键词	0.76%
摘要	0.70%

从表 3.1 中可以看到，准确率从高到低为作者给出的关键词、标题、摘要，可见不同部分数据的质量也有所不同。

3.4　基于贝叶斯分类的文献标引

文本分类是自然语言处理最重要也是最基础的应用之一。20 世纪 90 年代以来，随着信息资源量的不断增长，可用于训练的语料库越来越多，为基于统计方法的分类算法提供了大量的数据。基于统计学习的算法如贝叶斯分类逐渐成为文本分类的主要算法。

贝叶斯理论的基本思想最早由英国著名数学家 Thomas Bayes 于 1764 年提出，直至 20 世纪，信息论和统计决策理论的发展推动了贝叶斯理论的进一步发展。贝叶斯方法用概率表示不确定性，概率规则表示推理或学习，随机变量的概率分布表示推理或学习的最终结果。贝叶斯理论现已被应用到人工智能的众多领域，针对很多领域的核心的分类问题，大量卓有成效的算法都是基于贝叶斯理论设计。这里要解决的问题即为典型的文本分类问题，考虑使用贝叶斯分类解决该问题。

这里使用的机器学习库为 scikit-learn。scikit-learn 是一个功能强大的通用机器学习库，封装了大量常用的机器学习算法，包括各种特征工程以及分类算法，非常适合像该项目一样需要对数据进行大量处理的项目。这里使用的 TF-IDF 特征提取、卡方检验以及贝叶斯分类都可以利用该机器学习库较为容易地实现。

使用 TF-IDF 提取特征后，使用贝叶斯分类器进行分类，二级分类准确率仅为 76%，全部代码见 bayes1.py：

```
def train(train_data, train_target):
    #TfidfVectorizer 中默认的 token_pattern 不包括单个字的词
    #但考虑到中文中单个字对分类也有帮助,需要对其进行修改
```

```
tfidf = TfidfVectorizer(token_pattern = r"(?u)\b\w + \b")
tfidf_train = tfidf.fit_transform(train_data)

mnb = MultinomialNB(alpha = 1.0)
l = tfidf_train.shape[0]
#因数据量过大,需要使用 partial_fit 进行增量学习
for i in range(0, l, 100000):
    data = tfidf_train[i:min(i + 100000, l)]
    label = train_target[i:min(i + 100000, l)]
    mnb.partial_fit(data, label, classes = classes)

    return tfidf, mnb

def pre(tfidf, mnb, test_data, test_target):
    tfidf_test = tfidf.transform(test_data)
    predict = mnb.predict(tfidf_test)
    count = 0
    for left, right in zip(predict, test_target):
        if left == right:
            count += 1
    return count / len(test_target)
```

具体的分类指标如表 3.2 所示。

表 3.2　不同样本数的精确率、召回率、F1 分值对比 1

类　　别	Precision （精确率）	Recall （召回率）	F1-Score （F1 分值）	Support （样本数）
F	0	0	0	14
F0	0.96	0.14	0.24	961
F1	0.69	0.71	0.7	4296
F2	0.66	0.89	0.76	8959
F3	0.88	0.8	0.84	3338
F4	0.91	0.53	0.67	3011
F5	0.97	0.65	0.78	1805
F6	0	0	0	123
F7	0.86	0.57	0.68	2942
F8	0.82	0.9	0.86	8233
avg/total	0.79	0.76	0.75	33 682

其中，Precision 为分类的精确率，Recall 为分类的召回率，F1-Score 为 Precision 和 Recall 的调和平均数，Support 为样本数。在两个样本数最少的分类 F 与 F6 上的精确率和召回率均为 0，说明并没有测试集上的样本被标引为 F 或 F6。在样本数较少的分类 F0 与 F5 上虽然精确率很高，但召回率极低，说明标引为这两个分类的样本大部分是分类正确的，但还有很多属于它们的样本划分到了别的分类。二级分类精确率甚至低于 80%，远远没有达到该项目的预期，可能需要补充更多的数据或者使用更好的算法。下面从训练过程、特征降维以及权重调节三个角度提出三种不同的算法，用于提高类别标引的精确率。

3.4.1 增量训练

增量训练指的是机器学习方法不仅可以保留之前已经学习过的知识,也可以从新的样本中学习新的知识,这种训练方法的学习是可以逐步进行的。增量学习不仅可以及时利用新的数据,也可以避免因数据过大导致 MemoryError 的错误。

对于贝叶斯分类器的初次训练,训练数据使用的是训练集中各部分数据拼接经过 TF-IDF 特征提取器后获得的特征向量。在初步训练结束后,将训练数据的特征向量在训练好的贝叶斯分类器上进行预测,若预测结果与实际结果不一致,则将该条数据加入到新的训练集中,之后将所有训练集中预测失败的数据作为新的训练数据进行增量训练,以上过程重复多次。该算法的思想是增加难以预测的训练集样本的权重,其实现的核心代码如下:

```
for item in range(0, iterNum):                                    #迭代次数
    print(item, pre(tfidf, mnb, test_data, test_target))          #在验证集上预测
    tfidf_train = tfidf.transform(train_data)
    predict_target = mnb.predict(tfidf_train)                     #在训练集上预测
    for i in range(0, l, 100000):                                 #增量训练
        data = tfidf_train[i:min(i + 100000, l)]
        label = train_target[i:min(i + 100000, l)]
        predict = predict_target[i:min(i + 100000, l)]
        num = 0
        weight = []
        for a, b in zip(predict, label):
            if a != b:                          #若预测错误则将其加入到训练集中重新训练
              weight.append(1)
            else:                               #若预测正确则将其权重设置为 0 不进行训练
              weight.append(0)
            num += 1
        mnb.partial_fit(data, label, sample_weight = weight, classes = classes)
    print('res', pre(tfidf, mnb, test_data, test_target))
```

经过上述过程之后,将迭代次数分别在训练集和验证集上的准确率变化趋势可视化出来,结果如图 3.3 所示。

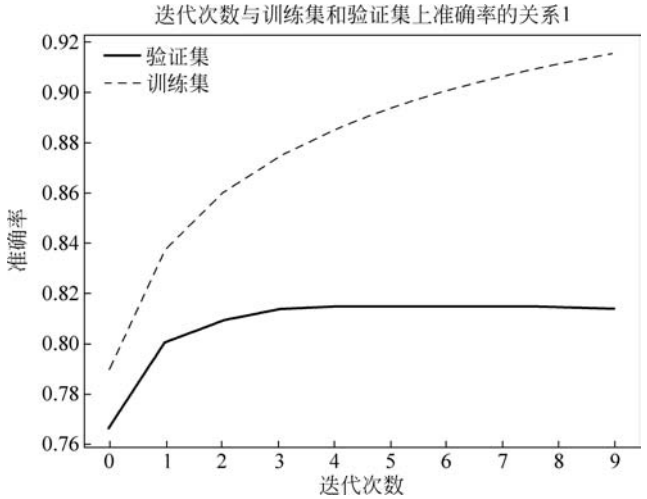

图 3.3 迭代次数与训练集和验证集上准确率的关系

从图 3.3 中每次迭代贝叶斯分类器在训练集和验证集上的准确率可以看出，在训练集上的准确率在每轮增量训练时均在上升，而在验证集上的准确率在前几轮增量训练时同步上升，而在后面的迭代时不再上升，反而在第 8 次迭代后略有下降。在验证集上的准确率在第一次增量训练时大大提升至接近 80%，在第 5 次到第 6 次增量训练时达到最大值。但如图所示显然这种方法会出现过拟合的情况，随着增量训练的轮数增加，在训练集上的准确率高于在验证集上的准确率，而在验证集上的准确率也会随之下降。

考虑将数据增强方法用于增量学习的每次迭代中，每次迭代都根据原训练数据获取不同的新的训练数据，若某条数据所属分类出现次数较少，则将该条数据随机删去部分词得到的新数据加入到新的训练数据中，该过程可能随机进行多次；若某条数据所属分类出现次数较多，则可能将该条数据随机删去部分词得到的新数据加入到新的训练数据中，也可能将其直接删除，实现代码如下：

```
#根据原训练集生成新的训练集,thr 为每个词保留的概率
def chuli(train_data, train_target, thr):
#某些样本较少的数据可能生成多条数据,某些样本较多的数据可能不生成数据
    gcy = {'F': 100, 'F0': 100, 'F1': 100, 'F2': 100, 'F3': 100, 'F4': 100,
'F5': 100, 'F6': 100, 'F7': 100, 'F8': 50}

    new_train_data = []
    new_train_target = []

    for i in range(len(train_data)):
      text = train_data[i]
      new_text = ''
      rand = random.randint(1, 100)
      k = gcy[train_target[i]]
      #判断是否生成或是否多次生成某条数据
      while rand <= k:
          for word in text.split():
          #对某个词语有 thr 的概率将其保留
              if random.randint(1, 100) <= thr:
                  new_text += '' + word
        new_train_data.append(new_text)
        new_train_target.append(train_target[i])
        k - = 100
    return new_train_data, new_train_target
```

通过这种数据增强与增量训练的学习方式使贝叶斯分类的准确率提升至 82%左右，迭代次数与在训练集和验证集上准确率的关系如图 3.4 所示。

从图 3.4 中可以看出虽然也存在一定的过拟合，但过拟合的情况远没有之前那么明显。且准确率远高于不使用数据增强的准确率。

3.4.2　特征降维与消歧

图书馆提供的数字资源量高达 70 多万，将训练集中各部分数据拼接后词语的种类即特征维度高达 40 多万，需要进行特征降维。数据降维既可以去除一些与分类关系不大的无关

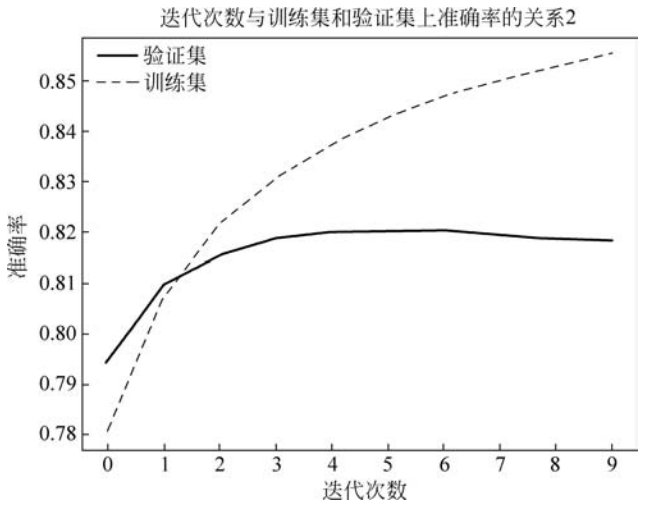

图 3.4　迭代次数与训练集和验证集上准确率的关系

特征,以便获取更有价值的信息,也可以大大降低算法的复杂度。

将卡方检验用于特征降维,对于所有数据使用 TF-IDF 特征提取方法提取出的特征,使用卡方检验的方法检验每个特征与分类的相关性,根据卡方值排序后的结果保留排名靠前的词加入到词表中,将词表中的词作为保留的特征。该方法对分类准确率有一定程度的提升。实现代码如下:

```
def train(train_data, train_target):
    tfidf = TfidfVectorizer(token_pattern = r"(?u)\b\w + \b")
    tfidf_train = tfidf.fit_transform(train_data)

    #建立新词表
    words_set = set()
    dict = {}
    for item in tfidf.vocabulary_:
        dict_2[tfidf.vocabulary_[item]] = item

    selectKBest = SelectKBest(chi2, k = 1)            #选择 k 个最佳特征
    selectKBest.fit_transform(tfidf_train, train_target)
    max_score = np.argsort(selectKBest.scores_)[::- 1]
     #得到新词表
     for i in range(maxNum):
         words_set.add(dict_2[max_score[i]])
     max_label = []
     for item in new_words_set:
         max_label.append(tfidf.vocabulary_[item])
     new_tfidf_train = tfidf_train[:, max_label]
     mnb = MultinomialNB(alpha = 1.0)
     l = tfidf_train.shape[0]
     for i in range(0, l, 100000):
         data = new_tfidf_train[i:min(i + 100000, l)]
         label = train_target[i:min(i + 100000, l)]
```

```
            mnb.partial_fit(data, label, classes = classes)
        return tfidf, mnb, max_label
def pre(tfidf, mnb, test_data, test_target, max_label):
    tfidf_test = tfidf.transform(test_data)
    predict = mnb.predict(tfidf_test[:, max_label])
    count = 0
    for left, right in zip(predict, test_target):
        if left == right:
            count += 1
    return count / len(test_target)
```

对于将卡方检验用于特征降维，考虑将卡方检验与前文中提到的数据增强方法相结合，设计了一种更好的算法进一步提升特征的质量。每次迭代时使用数据增强方法利用原训练集构建新的不同的训练集，之后使用上文中的卡方检验方法计算卡方值后排序，提取与分类关系较大的部分的特征并加入词表中，之后在验证集上进行预测。每次迭代用于训练的新的训练集各不相同，因此提取出的排名靠前的词也并不相同，每次迭代后若在验证集上进行预测的准确率有提升则保留该词表，否则删去本次迭代中加入的词并重新构造训练集进行训练。

算法开始时词表最初为空集，每次迭代时根据原训练集 traindata 构建新的训练集，并获取卡方值靠前的词，之后与原词表合并。若在验证集上使用合并后词表进行预测的准确率有提升则更新词表和准确率，若准确率若干次均为上升则停止迭代。

当每次选取卡方值前 36 000 个词时，词表和准确率的变化如图 3.5 所示。

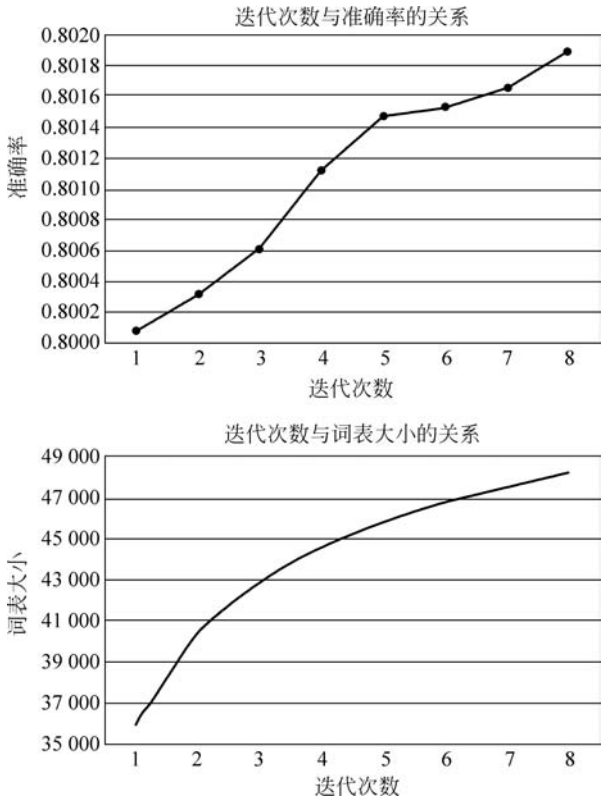

图 3.5　词表和准确率随迭代次数变化情况

获取最终词表后,考虑使用训练好的词向量进行消歧,对于某条数据中的某个词,若与其距离小于阈值的词在最终的词表中,则将词表中的词加入到该条数据中。

该算法效果使准确率略微提升,具体的分类性能指标如表 3.3 所示。

表 3.3 不同样本数的准确率、召回率、F1 分值对比 2

类 别	Precision (精确率)	Recall (召回率)	F1-Score (F1 分值)	Support (样本数)
F	0	0	0	14
F0	0.65	0.57	0.6	961
F1	0.69	0.82	0.75	4296
F2	0.85	0.78	0.81	8959
F3	0.84	0.89	0.86	3338
F4	0.79	0.83	0.81	3011
F5	0.92	0.93	0.92	1805
F6	0.86	0.68	0.76	123
F7	0.77	0.8	0.78	2942
F8	0.9	0.85	0.87	8233
avg/total	0.82	0.82	0.82	33 682

从表 3.3 中可以看出大部分类别的召回率均有小幅增长。

3.4.3 权重调节

在 scikit-learn 中的 TfidfVectorizer 提供了多种权重调节方法,通过设置 min_df 与 max_df 参数可以过滤掉在训练集中出现比例低于或高于该值的词语。虽然此方法在初期很有效,但在特征降维之后没有必要对此参数进行调整。

清华同方自动标引系统与 ST_index 自动标引系统都考虑到了给予不同部分的数据不同的权重。从数据预处理过程中也可以看出单独使用某部分数据进行训练,关键词和标题得到的准确率远高于摘要部分。最初尝试调节不同部分数据的词频,该方法虽然有一定的效果但调节起来费时费力,这种方式只是简单地将不同部分所占的权重进行调整,需要设计更优的算法对其权重进行调整取代这种手动调节权重的方法。

考虑到不同部分的数据中词语的分布不同,在不做特征降维之前拼接后特征数高达 40 多万,文章的标题、摘要和通过 TF-IDF 算法提取的关键词特征数高达 20 多万,但作者给出的关键词特征数仅为 4 万左右。各部分数据特征间有大量不重合的地方,不能简单地将这些词语进行拼接。

使用不同的 TF-IDF 特征提取器和贝叶斯分类器,分别对标题、人工提取的关键词、摘要、正文搜索得到的关键词、正文使用 TF-IDF 算法提取得到的关键词以及这些词语拼接后的结果进行训练,之后对在验证集上得到的属于不同分类的概率使用不同的权重相加,得到最后的结果。

将词语拼接部分的初始权重设置为 1,其他权重设置为 0,可以保证得到的结果不会比原来的结果即简单拼接的结果差。每次迭代随机增减每部分的权重,若得到的结果好于最好的结果则修改权重。若一次迭代中得到的结果均比最好的结果差,则减小每次增减权重

的值，直至使该值小于阈值。权重调整的框架代码如下：

```
def pre(tfidfs, mnbs, max_labels, test_data, test_target):
    probas = []
    #tfidfs,mnbs,max_labels 为在不同部分数据上不同的特征提取器、分类器和词表
    for i in range(data_num):
      proba = pre_part(tfidfs[i], mnbs[i], max_labels[i], test_data[i])
      probas.append(proba)

    accs = get_partly_acc(probas, test_data, test_target)
    for item in accs:
      print(item)
    if mode == 'train':
      #训练模式,使用验证集获取 w
      w = get_w(probas, test_data, test_target)
      print(w)
      return 0
    if mode == 'test':
      #测试模式,使用在验证集上获取的 w 在测试集上进行测试
      w = [1.25, 1.5, 0.31640625, 0.453125, 0.40625, 0.71875]
    return get_acc_use_w(probas, test_data, test_target, w)
```

其中，函数的参数 tfidfs 是 TfidfVectorizer 的对象数组；参数 mnbs 是 MultinomialNB 对象数组；参数 max_labels 是 TF-IDF 的词表(tfidf.vocabulary_)。pre_part 方法是将词表在分类器上进行验证，多组数据的验证结果存于 probas 对象中。函数 get_partly_acc 主要用于计算各组数据(test_data)的准确率。pre_part 函数的具体实现过程如下：

```
def pre_part(tfidf, mnb, max_label, test_data):
    tfidf_test = tfidf.transform(test_data)
    #predict = mnb.predict(tfidf_test)                    #在测试集上预测结果
    if max_label is not None:
        proba = mnb.predict_proba(tfidf_test[:, max_label])
    else:
        proba = mnb.predict_proba(tfidf_test)
    return proba
```

使用 TF-IDF 算法可以快速找到在验证集准确率的局部最大值，且可以保证结果不会差于使用将所有部分词语拼接得到的结果。

对于初始权重，当拼接部分权重为 1，其他部分权重为 0 时在验证集上得到的准确率即为不使用该权重调节算法时得到的准确率。将该准确率记为暂时的最大准确率，对于每次迭代调用 getRandomPlace()获取一个将 6 个不同位置打乱的列表，如将原位置[0,1,2,3,4,5]打乱为[3,5,1,0,2,4]，按这个顺序对这 6 个位置对应的数据进行权重调整，权重增减的值为 num。若增减权重后准确率提升则调整权重，若对于这 6 个位置增减权重准确率均未提高则将 num 减半。当 num 小于阈值时退出迭代。

当使用验证集得到对应不同部分数据不同的权重后，将该权重应用在测试集上，大大提升了在测试集上的准确率，具体的分类指标如表 3.4 所示。

表 3.4 权重调节之后各项指标对比结果

类 别	Precision（精确率）	Recall（召回率）	F1-Score（F1 分值）	Support（样本数）
F	0	0	0	14
F0	0.83	0.54	0.65	961
F1	0.77	0.84	0.8	4296
F2	0.83	0.87	0.85	8959
F3	0.89	0.88	0.89	3338
F4	0.89	0.82	0.85	3011
F5	0.95	0.92	0.93	1805
F6	0.95	0.58	0.72	123
F7	0.84	0.83	0.84	2942
F8	0.91	0.91	0.91	8233
avg/total	0.86	0.86	0.86	33 682

从表 3.4 中可以看到，经过权重调节，模型对二级分类的平均精确率、召回率和 F1 分值均有提升，达到 86%。

3.5 贝叶斯分类性能评估

使用训练集进行训练后在测试集上预测准确率提升至 86%。综合以上结果，该基于朴素贝叶斯的智能标引系统流程如下：预处理时根据训练集中的数据使用数据增强方法生成新的数据，使用特征降维算法获取新的词表，每轮增量训练时都将预测失败的数据增强后重新训练，最终对不同部分的数据使用权重调节算法对预测的概率进行调节。

对于小样本的分类问题，虽然使用了数据增强方法但效果有限，在 F 大类上的精确率和召回率仍为 0，可能需要在模型之前或之后人为地增加一些规则，从而满足小样本的关键特征，这样便可以最大限度地减小小样本的错误概率。若想进一步提升准确率需要结合深度学习等方法，这也是目前文本分类的主要研究方向。

3.6 基于 BERT 算法的文献标引

近年来深度学习方法在自然语言处理方面的研究和应用取得了显著的成果。2013 年 Word2Vec 的出现把文本数据从高维度、高稀疏变成了连续稠密的数据。基于 CNN 和 RNN 的分类方法在分类任务中效果显著。Attention 机制直观地给出每个词对结果的贡献。谷歌 AI 团队发布的 BERT 模型在 11 种不同的自然语言处理任务中创出佳绩，为自然语言处理带来里程碑式的改变，也是自然语言处理领域近期重要的进展。

BERT 是一种对语言表征进行预训练的方法，即经过大型文本语料库（如维基百科）训练后获得的通用“语言理解”模型，该模型可用于自然语言处理下游任务（如自动问答）。BERT 之所以表现得比过往的方法要好，是因为它是首个用于自然语言处理预训练的无监督、深度双向系统。BERT 的优势是能够轻松适用多种类型的自然语言处理任务。

3.6.1　数据预处理

图书馆给出了一个“中图法 F 大类第五版与第四版删改类目对照表”，此表中存在 2 个 sheet，分别为“需删除分类号数据”以及“四五版对比”，其示例如图 3.6 所示。

原分类号	修订注释	处理方式	修改后分类号
F014.6	两大部类及部门间关系，5版改入F264有关各类	删除分类号为“F014.6”，且年份为2015年以前的数据	
F031.1	停用；5版改入F031	改号	F031
F035.1	停用；5版改入F035	改号	F035
F035.2	停用；5版改入F264有关各类	删除分类号为“F035.2”的全部数据	
F035.3	停用；5版改入F036.1	改号	F036.1
F036.5	停用；5版改入C913.3	删除分类号为“F036.5”的全部数据	
F037.1	停用；5版改入F037	改号	F037
F037.3	停用；5版改入F037	改号	F037
F041.1	停用；5版改入F041	改号	F041
F041.2	停用；5版改入F041	改号	F041
F041.8	停用；5版改入F041	改号	F041
F045.1	停用；5版改入F045	改号	F045
F046.2	停用；5版改入F046	改号	F046
F046.3	停用；5版改入F046	改号	F046
F047.2	停用；5版改入F047.1	改号	F047.1
F047.5	停用；5版改入C913.3	删除分类号为“F047.5”的全部数据	
F048.1	停用；5版改入F048	改号	F048
F048.2	停用；5版改入F264有关各类	删除分类号为“F048.2”的全部数据	
F114.42	停用；5版改入F114.4	改号	F114.4
F114.45	停用；5版改入F114.4	改号	F114.4
F121.29	地下经济，5版改入F264.9	将分类号为“F121.29”，且主题词中包含“地下经济”的“相关数据分类号改为“F264.9”，其余不变	
F123.11	停用；5版改入F123.1	改号	F123.1
F123.13	停用；5版改入F123.1	改号	F123.1
F124.7	扶贫问题，5版改入F126	将分类号为“F124.7”主题词中包含“扶贫”的数据分类号改为“F126”，其余不变	

需删除分类号
F001.5
F014.309
F014.359
F014.7
F016.5
F019.349
F0-27
F033.3
F046.32
F046.33
F058
F059
F061
F061.9
F062
F063
F065
F069
F069.4
F069.6
F074.1

图 3.6　“四五版对比”的示例

“需删除分类号数据”中主要是分类号的列表，如图中右侧显示的样式，将表中分类号对应的数据全部删除（此部分数据分类号错误），再根据“四五版对比”表中的处理方式，对剩余数据进行处理。然后读取“四五版对比”表，根据给定规则处理剩余标签，主要有以下几种规则来处理标签。

- rule1：五版停用，但还是属于 F 经济大类，故直接修改为五版对应的分类号。
- rule2：五版停用，但不属于 F 经济类，直接删除此类数据。
- rule3：将此类分类号下的某些包含特定主题词的样本改为其他分类号。
- rule4：将此类分类号下的某些包含特定主题词的样本删除。
- rule5：删除特定年份之前的分类号数据。

首先构建这 5 项修改规则的字典，对于 rule1 来说，只需匹配处理方式中为“改号”的记录，将修改后的分类号添加到 rule1 的字典；对于 rule2，匹配处理方式中的“删除”以及“全部数据”两个字符串即可找到要删除的分类号；对于 rule3，稍微复杂一些，由于处理方式一栏的语言组成不标准，这里增加了对于关键词的判断，匹配了“主题”（肯定存在）以及“改为”或“入”（两者存在其一即可）；对于 rule4，匹配“删除”与“主题”，并使用正则表达式，将包含的主题词解析出来用于删除判断；对于 rule5，匹配“年份”以及“删除”即可。实现过程代码如下：

```
import sys
import xlrd
import pandas as pd
ExcelFile = xlrd.open_workbook('4version_2_5version.xlsx')
sheet_name = ExcelFile.sheet_names()
compare = ExcelFile.sheet_by_name(sheet_name[0])
delete = ExcelFile.sheet_by_name(sheet_name[1])
delete_cols = delete.col_values(0)
```

```
label_needDel = []
for item in delete_cols:
    label_needDel.append(item)
label_needDel.remove('需删除分类号')
print("read file…")
df = pd.read_csv('F08-18.csv', encoding='utf8')
print("read finish!")
```

其中,先读取两个 sheet,将需要比较的四、五版分类号读到 compare 列表中,并将需要删除的分类号读到 delete_cols 中。然后依次提取 1～5 项规则并执行,实现的核心代码如下:

```
rule1 = {}
rule2 = {}
rule3 = {}
rule4 = {}
rule5 = {}
pattern_rule4 = re.compile(r'.*?[""]+(.+?)[""]+')
print("create rules…")
for i,item in enumerate(rules):
    if item == "改号":
        rule1[original_label[i].strip()] = final_label[i]
    elif '删除' in item and '年份' in item and '2015' in item:
        rule5[original_label[i].strip()] = 'delete'
    elif '删除' in item and '全部数据' in item :
        rule2[original_label[i].strip()] = 'delete'
    elif '删除' in item and '主题' in item:
        theme = pattern_rule4.findall(item)
        th = ""
        for j in range(1,len(theme)-1):
            th = th + theme[j] + "/"
        th = th + theme[-1]
        rule4[original_label[i].strip()] = th
    elif '主题' in item and ('改为' in item or '入' in item):
        theme = pattern_rule4.findall(item)
        th = ""
        for j in range(0,len(theme)-1):
            th = th + theme[j] + "/"
        th = th + theme[-1]
        rule3[original_label[i].strip()] = th
```

从代码中能看到,主要是通过判断规则的关键词,例如,出现“改号”标记时,则将要改的原类别号和目标类别号记录到 rule 列表中,需要删除的类别号则记录到 rule2 中,以此类推。经过处理之后的结果如下,其中对于 rule1 得到的分类号为:

```
{'F031.1': 'F031', 'F035.1': 'F035', 'F035.3': 'F036.1', 'F037.1': 'F037', 'F037.3': 'F037', 'F041.1': 'F041',
'F041.2': 'F041', 'F041.8': 'F041', 'F045.1':
'F045', 'F046.2': 'F046', 'F046.3': 'F046', 'F047.2': 'F047.1', 'F048.1': 'F048',
'F114.42': 'F114.4', 'F114.45': 'F114.4', 'F1 23.11': 'F123.1', 'F123.13':
'F123.1', 'F213.1': 'F213', 'F213.2': 'F213', 'F213.3': 'F213', 'F213.4':…}
```

对于 rule2 得到的分类号为：

{'F035.2': 'delete', 'F036.5': 'delete', 'F047.5': 'delete', 'F048.2': 'd elete', 'F213.5': 'delete', 'F249.15': 'delete', …}

对于 rule3 得到的结果为：

{'F031.1': 'F243.3/劳动定额/F243.1', 'F035.2': 'F243.5/劳动纪律/纪律/生产责任制/F243.1', 'F124.7': 'F062.9/产业经济学/产业经济/产业定位/产业发展/产业规模/产业化/ 产业化经营/产业经济学/产业社会学/产业组织/产业组织理论/F260/F260/产业经济/政府管制/规制经济学/F262/F260/产业集群/产业带/产业链/产业市场/产业一体化/F263', 'F046.2': 'F270/企业管理/管理理论/企业经营管理/企业行为/企业行为学/F270－0/F270/企业文化/企业形象/企业精神/企业责任/企业信用/F272－05', 'F014.6': 'F279.15/…}

对于 rule4 得到的结果为：

{'F302.5': '农业数据', 'F743.1': '国际贸易组织'}

对于 rule5 得到的结果为：

{'F014.6': 'delete', 'F293': 'delete', 'F293.33': 'delete'}

至此，所有规则创建成功，接下来便是遍历原始数据文件，根据上面的五个规则更改标签，最终将第四版分类规则全部统一为第五版规则体系下，将数据保存为 F08-1.tsv。

3.6.2　构建训练集

在 BERT 模型训练时，需要按其要求对数据格式进行转化，并且构建训练集和测试集，首先读取 3.6.1 节中预处理完成的 csv 到 DataFrame 中。

```
df = pd.read_csv("./F08－1.tsv", header=0,\
    usecols=['attribute_string_1', 'attribute_string_5', 'attribute_string_6',\
    'attribute_string_13','attribute_text_1','attribute_string_14'],sep='\t')
```

对读到的 DataFrame 提取前 5 条进行查看，结果如图 3.7 所示。

```
df.head(5)
```

	attribute_string_1	attribute_string_5	attribute_string_6	attribute_string_13	attribute_string_14	attribute_text_1
0	论清末湖北地方政府军政外债的影响	沈阳工程学院学报:社科版	46272.0	/清后期/财政史/湖北/外债	F812.952	清末,湖北地方政府为弥补财政亏空举借了各种用途的军政外债共10笔,这些外债对债权方和湖北地区...
1	中心城区区域经济发展模式的选择与分析	沈阳工程学院学报:社科版	46272.0	/城市经济/经济发展/中国	F299.2	随着经济持续发展,各大城市中心城区的经济发展模式日渐成熟,由于中心城区发展的特殊性和广泛的示...
2	中国股市高投机性的制度机理研究	沈阳工程学院学报:社科版	46272.0	/股票市场/制度/中国	F832.5	投机性是市场所具有的天然属性.然而,中国股市投资者的过度投机行为在市场波动中所反映出来的极端...
3	西部农村减缓贫困的进展	中国农村观察	4276.0	/地方农业经济/中国	F327	本文在对改革初期中国西部农村贫困特点回顾的基础上，综述了改革开放以来西部地区的反贫困措施。经...
4	交易成本与中国农村的基础设施治理结构选择:以灌溉、电力、公路和饮用水设施为例	中国农村观察	4276.0	/农业建设/农业经济发展/中国	F323	本文运用交易成本理论指出，农村基础设施治理的最优化是其有效供给的充分条件，不同规模设施的最优...

图 3.7　训练数据示例

可以看到标题、出版社、关键词等字段名为原始库中带的列名，为了方便辨识，将其转化为易于阅读的属性名称，实现方法如下：

```
df.columns = ['title','publisher','pubcode','keywords','category','abstract']
```

其中,category 是文献的类目编号,即未来需要进行预测的标签列,对数据进行简单分析,查看其类别数量和前 10 类目。

```
print('\nnumber of different class: ', len(list(set(df.category))))
print(list(set(df.category))[:10])
```

运行之后得到结果如下:

```
number of different class: 4170
['F726.722', 'F552.9', 'F812.934', 'F550.7', nan, 'F811.2', 'F535.51', 'F269.338', 'F272.91',
'F147.6']
```

可以看到其中 F 大类下总的类目数量为 4170 种,使用 set 方法获取唯一的类目编号,发现其中有空值的编号,需要将其过滤。当前任务的目标是对 4 级类目进行分类,所以还要对类目(category)进行长度截取,获得 3 级标签和 4 级标签,并将其分别命名为 level3 和 level4,实现代码如下:

```
df = df[df.category.notnull()]
df['level3'] = df.category.str[:3]
df['level4'] = df.category.str[:4]
```

然后,对标题、关键词和摘要内容进行合并作为输入,并使用 thulac(pip3 install thulac)对其进行分词,实现代码如下:

```
import thulac
thu1 = thulac.thulac(seg_only = True)
df['content'] = df['title'] + df['publisher'] + df['keywords'] + df['abstract']
for index, row in df_columns_all.iterrows():
    try:
        seg_list = thu1.cut( row['content'])
        seg_list1 = [w[0] for w in seg_list if w[0].strip() not in stopwords]
        df_columns_all.at[index,'content1'] = " ".join(seg_list1)
        if row['level4'][-1] == '-':
            df_columns_all.at[index,'level4'] = row['level4'][:-1]
        if index % 1000 == 0:print(str(index))
    except:
        print(row['content'],row['level4'])
        print(index)
        break
```

由于是强制截断类目编号,而编号中会有 F53—32 这种带“—”字符的情况,会使 level4 中末尾字符可能存在“—”,需要将其去掉,另外,数据量较大,每处理 1000 条则输出处理的进度,最后得到的训练集示例如图 3.8 所示。

下一步划分训练集、验证集和测试集,实现方法如下:

```
df_columns_all = df
msk = np.random.rand(len(df_columns_all)) < 0.9
train = df_columns_all[msk]
dev_test = df_columns_all[~msk]
```

level3	level4	content1
F81	F812	论 清末 湖北 地方 政府 军政 外债 影响 沈阳 工程学院 学报 社科版 清后期 财政史 ...
F29	F299	中心 城区 区域 经济 发展 模式 选择 与 分析 沈阳 工程学院 学报 社科版 城市 经济...
F83	F832	中国 股市 高投机性 制度 机理 研究 沈阳 工程学院 学报 社科版 股票市场 制度 中国...
F32	F327	西部 农村 减缓 贫困 进展 中国 农村 观察 地方 农业 经济 中国 本文 在 对 改革 ...
F32	F323	交易 成本 与 中国 农村 基础 设施 治理 结构 选择 以 灌溉 电力 公路 和 饮用水 ...
F32	F323	关于 农业 基础 建设 制度 变迁 内在 机理 制度 分析 基于 徐闻县 案例 调查 中国 ...

图 3.8　分词之后的训练样本示例

```
msk = np.random.rand(len(dev_test)) < 0.5
dev = dev_test[msk]
test = dev_test[~msk]
train.to_csv ('train.tsv', sep = '\t', index = None, header = None)
dev.to_csv ('dev.tsv', sep = '\t',index = None, header = None)
test.to_csv ('test.tsv', sep = '\t',index = None, header = None)
```

其中，由于总数据集的样本量较大，所以最后训练集占总样本量的 90%，而验证集占比为 5%，测试集占比为 5%，分别将其保存为 train.tsv、dev.tsv 和 test.tsv 三个文本文件，供后续 BERT 模型的训练和测试。

3.6.3　模型实现

BERT 模型采用 Google 开源项目，下载地址为 https://github.com/google-research/bert，只需要在 run_classifier.py 中建立自定义的样本处理类（DataProcessor）即可实现对文本的分类，其实现代码如下：

```
class SHLibProcessor(DataProcessor):

  def_init_(self):
      lable_file_path = os.path.join(FLAGS.data_dir, "level4_labels.txt")
      self.static_label_list = self.load_labels(lable_file_path)

  def get_train_examples(self, data_dir):
    return self._create_examples(
      self._read_tsv(os.path.join(data_dir, "train.tsv")), "train")

  def get_dev_examples(self, data_dir):
    return self._create_examples(
      self._read_tsv(os.path.join(data_dir, "dev.tsv")), "dev")

  def get_test_examples(self, data_dir):
    return self._create_examples(
      self._read_tsv(os.path.join(data_dir, "test.tsv")), "test")
```

```
  def get_labels(self):
    """example ['F575', 'F759', 'F495','F140', 'F460', 'F410', 'F765', 'F615']"""
    return self.static_label_list

  def _create_examples(self, lines, set_type):
    examples = []
    for (i, line) in enumerate(lines):
      guid = "%s-%s" % (set_type, i)
      if set_type == "test":
        text_a = tokenization.convert_to_unicode(line[0])
        label = tokenization.convert_to_unicode(line[2])
      else:
        text_a = tokenization.convert_to_unicode(line[0])
        label = tokenization.convert_to_unicode(line[2])
      examples.append(InputExample(guid=guid, text_a=text_a, text_b=None, label=label))
    return examples

  def load_labels(self, label_file_path):
    with open(label_file_path,'r') as label_file:
      static_label_list = list(set(label_file.read().splitlines()))
    return static_label_list
```

其中,get_train_examples、get_dev_examples、get_test_examples 三个方法中只需要指定上一步骤中的训练集、验证集和测试集文件名即可。get_labels 方法中需要返回所有样本的标签值列表,编写方法 load_labels 实现标签列表的加载,static_label_list 中存储的标签格式为['F575','F759','F495','F140','F460','F410','F765','F615']。

在_create_examples 方法中,由于输入格式和之前有少许不同,需要更改训练集和测试集文件中对应的输入和标签列列号,这与 train.tsv 各列的排列有关。由于在生成训练集时,第 1 列为分词后的文本内容,第 2 列为 3 级类目,第 3 列为 4 级类目,所以 text_a 指定为 line[0],而 label 指定为 line[2]。

由于这里是分类任务,而不是训练词向量,所以不需要指定 text_b 的值,即将其赋值为 None,为了在训练过程中使用前面定义的样本集处理类,需要在 main 方法中增加处理类的 key 值,我们命名为“shlib”,代码如下:

```
processors = {
    "cola": ColaProcessor,
    "mnli": MnliProcessor,
    "mrpc": MrpcProcessor,
    "xnli": XnliProcessor,
    "shlib":SHLibProcessor
}
```

最后,在 run_classifier.py 的起始设置好参数,或者在命令行指定,具体参数如下:

```
export BERT_BASE_DIR=/root/chinese_L-12_H-768_A-12
export GLUE_DIR=/root/Lib

nohup python36 -u run_classifier.py \
  --task_name=shlib\
```

```
--do_train=true \
--do_eval=true \
--data_dir=$GLUE_DIR/data \
--vocab_file=$BERT_BASE_DIR/vocab.txt \
--bert_config_file=$BERT_BASE_DIR/bert_config.json \
--init_checkpoint=$BERT_BASE_DIR/bert_model.ckpt \
--max_seq_length=128 \
--train_batch_size=32 \
--learning_rate=2e-5 \
--num_train_epochs=10 \
--output_dir=$GLUE_DIR/output>log &
```

其中，BERT_BASE_DIR 为中文预先训练的 BERT 字向量模型，task_name 是构建训练集和标签的方法，max_seq_length 是模型输入的文本长度，num_train_epochs 指定迭代的回合数为 10 次，除此之外还可以指定每隔多少步保存模型(save_checkpoints_steps)，目前设为 1000 步，默认情况下程序会保存最近 5 个 checkpoints 的模型。训练过程存入 log 文件中。

运行后便可进入训练过程，首先输出训练集示例，结果如图 3.9 所示。

```
INFO:tensorflow:Writing example 0 of 249720
INFO:tensorflow:*** Example ***
INFO:tensorflow:guid: train-0
INFO:tensorflow:tokens: [CLS] 论 清 末 湖 北 地 方 政 府 军 政 外 债 影 响 沈 阳 工 程 学 院 学 报 社
 科 版 清 后 期 财 政 史 湖 北 外 债 清 末 湖 北 地 方 政 府 为 弥 补 财 政 亏 空 举 借 各 种 用 途 >
军 政 外 债 共 10 笔 这 些 外 债 对 债 权 方 和 湖 北 地 区 均 产 生 双 重 影 响 体 现 外 债 二 重 性
 对 债 权 国 而 言 各 帝 国 主 义 国 家 为 获 得 湖 北 债 务 放 贷 权 而 矛 盾 加 剧 同 时 他 们 通 >
过 对 [SEP]
INFO:tensorflow:input_ids: 101 6389 3926 3314 3959 1266 1765 3175 3124 2424 1092 3124 1912 965 2512 1
510 3755 7345 2339 4923 2110 7368 2110 2845 4852 4906 4276 3926 1400 3309 6568 3124 1380 3959 1266 19
12 965 3926 3314 3959 1266 1765 3175 3124 2424 711 2477 6133 6568 3124 755 4958 715 955 1392 4905 450
0 6854 1092 3124 1912 965 1066 8108 5011 6821 763 1912 965 2190 965 3326 3175 1469 3959 1266 1765 127
7 1772 772 4495 1352 7028 2512 1510 860 4385 1912 965 753 7028 2595 2190 965 3326 1744 5445 6241 1392
 2370 1744 712 721 1744 2157 711 5815 2533 3959 1266 965 1218 3123 6587 3326 5445 4757 4688 1217 1196
 1398 3198 800 812 6858 6814 2190 102
INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
INFO:tensorflow:segment_ids: 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 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 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
INFO:tensorflow:label: F812 (id = 0)
INFO:tensorflow:*** Example ***
```

图 3.9　训练过程中的样本示例

从中可以看到，总的训练样本为 24 万 +，基于已经预训练好的中文 BERT 模型，会将中文的每个字映射为数值，并解析出标签。

接下来会进入 BERT 模型的训练过程，并不断输出模型的训练性能指标数据，其输出结果如图 3.10 所示。

```
INFO:tensorflow:examples/sec: 53.8993
INFO:tensorflow:global_step/sec: 1.68454
INFO:tensorflow:examples/sec: 53.9052
INFO:tensorflow:global_step/sec: 1.68428
INFO:tensorflow:examples/sec: 53.8969
INFO:tensorflow:global_step/sec: 1.68547
INFO:tensorflow:examples/sec: 53.935
INFO:tensorflow:global_step/sec: 1.68566
INFO:tensorflow:examples/sec: 53.9412
INFO:tensorflow:global_step/sec: 1.68565
INFO:tensorflow:examples/sec: 53.9409
INFO:tensorflow:Saving checkpoints for 13000 into ../output/model.ckpt.
```

图 3.10　模型训练过程

从中可以看到平均每秒处理样本数为53条,每隔1000步将结果进行保存,在训练结束后使用测试集进行验证,得到结果如图3.11所示。

```
INFO:tensorflow:Restoring parameters from ../output/model.ckpt-39018
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2019-05-14-19:15:58
INFO:tensorflow:Saving dict for global step 39018: eval_accuracy = 0.7766962, eval_loss = 0.87970275
 global_step = 39018, loss = 0.8796229
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 39018: ../output/model.ckpt-39018
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 0.7766962
INFO:tensorflow:  eval_loss = 0.87970275
INFO:tensorflow:  global_step = 39018
INFO:tensorflow:  loss = 0.8796229
```

图3.11 模型训练后的测试结果

可以看到最终4级类目的分类准确率约为77.67%。

另外,采用相同的训练过程,对3级类目进行训练,与4级分类不同之处在于样本处理器(SHLibProcessor)的方法_create_examples中,修改标签列的序号,将如下代码

```
label = tokenization.convert_to_unicode(line[2])
```

修改为:

```
label = tokenization.convert_to_unicode(line[1])
```

然后再运行训练程序,等模型训练完成,可以从日志中看到模型的验证结果如下:

```
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 19512: ../output/model.ckpt-
19512
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow: ***** Eval results *****
INFO:tensorflow: eval_accuracy = 0.85591197
INFO:tensorflow: eval_loss = 0.5436569
INFO:tensorflow: global_step = 19512
INFO:tensorflow: loss = 0.5433417
```

可以看到3级类目的测试集准确率达到了85.59%,相较于前述贝叶斯算法中的2级类目准确率有明显的性能提高。

目前图书馆所能够采用的数字资源自动标引系统较为陈旧,其算法未利用近几年来在机器学习、自然语言处理方面的新成果。这些系统的标引准确率低下,且需要人工参与进行协助分类或者检验,不能从真正意义上解放人力资源,达不到自动标引的要求。而近年来快速发展的基于机器学习和深度学习的自然语言处理算法并未有在数字资源标引系统上的应用。在这一领域中我们可以看到强烈的需求与落后的产品技术之间的差距。同时,将现有的多种NLP技术恰当地组合,应用于数字资源自动标引这一任务上。

这里利用多种机器学习、深度学习算法进行实践,最终与人工分类结果进行对比,在分类上获得与人工标引相当甚至更高的准确率,解决了大数据背景下的智能分类问题。

思考题

1. 在分类问题中,讨论对特征降维的方法。

2. 阐述卡方检验用于特征降维的理由。
3. 在分类性能指标中，如何平衡模型精确度和召回率的取值？
4. 如何度量文本中的词的重要性？
5. 与贝叶斯网络等算法相比，BERT 算法用在文本分类有什么优势？
6. 怎样理解文献标引是典型的文本分类问题？
7. 在文献标引中，BERT 算法的输入和输出分别是什么？
8. 把识别错误的检验样本加入训练样本是否可以提高训练模型的质量？