分类器构建

1. 导言

分类是数据挖掘中应用领域极其广泛的重要技术之一,至今已经提出很多算法。分类是根据数据集的特点构造一个分类器,利用分类器对未知类别的样本赋予类别的一种技术。构造分类器的过程一般分为训练和测试两个步骤。在训练阶段,分析训练数据集的特点,为每个类别产生一个对相应数据集的准确描述或模型。在测试阶段,利用类别的描述或模型对测试进行分类,测试其分类准确度。一般来说,测试阶段的代价远远低于训练阶段。

本次所需要的分类器是针对校园信息的文本分类，类别限定在学科竞赛、科研信息、行政信息、招生信息、招聘就业、校园活动、升学留学、生活娱乐这八类，足以归纳所有的校园信息。目的是做一个程序，输入一篇文章的内容，输出这篇文章的类型。主要过程是收集一定数量的已知类别的文章作为训练数据，用这些训练数据结合一些分类算法训练出分类器，用这个训练好的分类器去处理未分类的文章就可以得到该文章的类型。算法过程中还涉及很多具体问题，在下文中具体介绍。

1. 算法流程

分类器构建主要流程有收集数据、中文分词、构建词向量空间、设计权重策略、构建分类器。

要实现文本分类，首先要提取文章的特征，常用词向量空间表示文章的特征，词向量空间就是具有这篇文章所有词的向量，前提将文章切分成词条。词向量空间只能表层的表征文章，更准确表征文章特征的数据类型是文章中每个词条在文本中的权重，本文使用TF-IDF（后面具体介绍）算法计算词的权重，这样就可以使用简单的数据类型表征一篇文章，更重要的是，这样的数据类型可以被Python的分类器算法识别使用。构建一个文本分类器还需要建立所有文本词条的集合，类似字典，可以理解二维数据，横坐标是一个词条，纵坐标是这个词条的索引，就是将词条与数据对应起来。

* 1. 收集数据

收集数据包括训练数据和测试数据的收集，测试数据的收集成本较训练数据的收集成本低很多。

训练数据是一定数量的已知类型的文本数据，由两部分组成：数据和标签，在这里数据就是文章内容，标签就是文章类型。训练数据作为所有文章特征的体现者，数量如果太少，不足以反映出所有待分类数据的特征，如果太多，可能会使特征饱和，造成浪费（个人想法）。而且每个类型的训练数据量应该是相近的。最重要的是，训练数据的固有类型和标签一定要真实对应的，否则训练好的分类器不符合自然逻辑。

网上几乎没有现成的针对校园信息的数据集，所以训练数据的收集是手动进行的，提取350条左右的校园新闻作为训练数据，人眼判断出这篇文章的类型，这种方法不可避免主观错误。

测试数据就是待分类的文章。

* 1. 中文分词

1.1中我们得到的训练数据和测试数据都是连续的句子，现在需要对这些文本进行分词，并且去除掉了所有的换行和空格，只有这样，才能在基于单词的基础上对文档进行结构化表示。相对于英文分词，中文分词有其特有的难点。现有很多的中文分词工具，本文使用了Python第三方库jieba这个中文分词组件，可以实现文本的准确分割。

* 1. 文本变量表示

在1.2中得到了分词后的训练数据和测试数据，下面我们要把这两个数据集表示为变量，从而为下面程序调用提供服务。我们采用的是Scikit-Learn库中的Bunch数据结构来表示这两个数据集。

为了完整了表征一篇文章，需要以下信息：

1，类别。也就是所有分类类别的集合，即学科竞赛、科研信息、行政信息、招生信息、招聘就业、校园活动、升学留学、生活娱乐这些类别组成的列表，叫做target\_name。

2，文本文件名。我们可以把所有文件名集合在一起做一个列表，叫做filenames

3，文本标签（就是文本的类别），叫做label（与2中的filenames一一对应）。文本标签与1中的类别区别在于：文本标签集合里面的元素就是1中类别，而文本标签集合的元素是可以重复的，相应的，1中的类别集合元素显然都是独一无二的类别。

4，文本内容（contens）。1.1中已经成功的把文本内容进行了分词，并且去除掉了所有的换行，得到的其实就是一行词袋（词向量），每个文本文件都是一个词向量。这里的文本内容指的就是这些词向量。

我们在Bunch对象里面创建了有4个成员：

target\_name：是一个list，存放的是整个数据集的类别集合。

label：是一个list，存放的是所有文本的标签。

filenames：是一个list，存放的是所有文本文件的名字。

contents：是一个list，分词后文本文件词向量形式

可以使用下图表示：



图 1 Bunch数据结构

* 1. 权重策略--TF-IDF

上述操作将训练数据和测试数据绑定为Bunch数据类型，实现了数据集的变量表示，也就是词向量。但是在不同的文本中，词向量之间的单词个数并不相同，词向量的涵盖的单词也不尽相同。他们并不在一个空间里，换句话说，就是他们之间没有可比性，例如：词向量1：我校 聘请 德国 ….，对应的词向量空间是（我校，聘请，德国,…），可以表示为（1,1,1）；词向量2：航天 学院 党委 …，对应的词向量空间是（航天，学院，党委，…），也可以表示为（1,1,1,1）。需要将所有这些词向量统一到同一个词向量空间中，例如，在上面的例子中，我们可以设置词向量空间为（我校，聘请，德国，航天，学院，党委，…），这样，词向量1和词向量2分别可以表示为（1,1,1,0,0,0）和（0,0,0,1,1,1），这样两个向量就都在同一个空间里面了，可以进行比较和各种运算了。

上述操作会导致词向量的维数很大，要把训练集内所有出现过的单词，都作为一个维度，构建统一的词向量空间，即使是中等大小的文本集合，向量维度也很轻易就达到数十万维。为了节省空间，我们首先将训练集中每个文本中一些垃圾词汇去掉。所谓的垃圾词汇，就是指意义模糊的词，或者一些语气助词，标点符号等等，通常他们对文本起不了分类特征的意义。这些垃圾词汇我们称之为停用词。把所有停用词集合起来构成一张停用词表格，这样，以后我们处理文本时，就可以从这个根据表格，过滤掉文本中的一些垃圾词汇了。

下面目的就是要将训练集所有文本文件（词向量）统一到同一个词向量空间中。值得一提的是，在词向量空间中，事实上不同的词，它的权重是不同的，它对文本分类的影响力也不同，为此我们希望得到的词向量空间不是等权重的空间，而是不同权重的词向量空间。我们把带有不同权重的词向量空间叫做“加权词向量空间”，也有的技术文档将其称为“加权向量词袋”，一个意思。

本文使用了TF-IDF（Term Frequency-InversDocument Frequency）加权技术，这是一种常用于信息处理和数据挖掘的加权技术。该技术采用一种统计方法，根据词条在文本中出现的次数和在整个语料中出现的文档频率来计算一个词条在整个语料中的重要程度。它的优点是能过滤掉一些常见的却无关紧要本的词语，同时保留影响整个文本的重要字词。

在一份给定的文件里，词频（term frequency，TF）指的是某一个给定的词语在该文件中出现的频率。这个数字是对词数(term count)的归一化，以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词数，而不管该词语重要与否。）对于在某一特定文件里的词语来说，它的重要性可表示为：



以上式子中  是该词在文件中的出现次数，而分母则是在文件中所有字词的出现次数之和。

逆向文件频率（inverse document frequency，IDF）是一个词语普遍重要性的度量。某一特定词语的，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到：



其中，表示语料库中的文件总数，表示词条的文件数目，如果该词语不在语料库中，就会导致被除数为零，因此一般情况下使用

然后



某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

下面我们将训练集文本转换成了一个TF-IDF词向量空间，应该保证训练样本和测试样本使用同样的词向量空间，即使测试集出现了新的词汇（不是停用词），即使新的文本数据有新的词汇，只要它不是训练集生成的TF-IDF词向量空间中的词，我们就都不予考虑。这就实现了所有文本词向量空间的统一，也只有这样，训练样本和测试样本才能相互比较。

本文使用scikit-learn包计算文本TF-IDF值，Scikit-learn是一个用于数据挖掘和数据分析的简单且有效的工具，它是基于Python的机器学习模块。

构建训练样本的TF-IDF空间向量：

1. 构建tf-idf词向量空间对象

tfidfspace=Bunch(target\_name=bunch.target\_name,label=bunch.label,filenames=bunch.filenames, tdm=[], vocabulary={})

在前面几节中，我们已经介绍了Bunch，target\_name等几个成员都是我们自己定义的数据类型，前面已经讲过不再赘述。下面我们讲一下tdm和vocabulary变量：tdm存放的是计算后得到的TF-IDF权重矩阵，这也是后面分类器需要的东西，就是训练集的tdm和标签label，因此这个成员是很重要的。vocabulary是词向量空间的索引，例如，如果我们定义的词向量空间是（我校，聘请，德国），那么vocabulary就是这样一个索引字典vocabulary={"我校":0," 聘请":1," 德国":2}，你可以简单的理解为：vocabulary就是词向量空间的坐标轴，索引值相当于表明了第几个维度。我们现在就是要构建一个词向量空间，因此初始化tdm和vocabulary为空。

1. TfidfVectorizer初始化向量空间模型

vectorizer = TfidfVectorizer(stop\_words=stpwrdlst, sublinear\_tf=True, max\_df=0.5)

tfidfspace.tdm = vectorizer.fit\_transform(bunch.contents)

tfidfspace.vocabulary = vectorizer.vocabulary\_

TfidfVectorizer会构建计算词频和构建一个计算TF-IDF。stop\_words是停用词表，max\_df表示，当一个词所有文章中出现的频率大于50%就认为这个词是停用词（高频词肯定不能表现文章的特性）。

经过上述程序就得到了训练库的tdm，即TF-IDF权重矩阵；vocabulary，即词向量空间坐标。

构建测试样本的TF-IDF空间向量时，步骤与训练样本的TF-IDF空间向量构建基本一致，但要注意的是，测试样本在构建tf-idf词向量空间对象时，需要将vocabulary初始化为训练数据的vocabulary，这样就实现了训练样本和测试样本在同一词向量中比较。

* 1. 分类器构建

本文采用的是朴素贝叶斯分类器，使用Python第三方库sklearn.naive\_bayes中的MultinomialNB库。这个库有封装好的分类器函数：MultinomialNB，这个函数获取训练集的权重矩阵和标签，进行训练，然后获取测试集的权重矩阵，进行预测，即给出预测标签。

1. 训练分类器：

输入词袋向量和分类标签，alpha:0.001 alpha越小，迭代次数越多，精度越高

clf = MultinomialNB(alpha=0.001).fit(train\_set.tdm, train\_set.label)

1. 预测分类结果

predicted = clf.predict(test\_set.tdm)。

至此分类器构建结束。

1. 总结

本文主要介绍了文本分类器构造的过程，其中讲述了一些文本分类概念。其中每一步都有很多优化的方法，后续研究。经过实践，构建出的分类器的正确率目测在70%左右，还有很多改进的方法，如改变分词算法，改变分类器构建方法。

1. 参考文献

[1]Python中文文本分类，<http://blog.csdn.net/github_36326955/article/details/54891204>。