数据挖掘 Project 2 报告

计 25 高越 2012011379

一. 语言：python

二. 工具：scikit-learn(http://scikit-learn.org)

三. 实现任务：分类，简单的层分类

四. 实验过程

1. 预处理

第一步工作是解析 xml 文件，我没有用 tools 文件夹里的 NYTCorpusDocument.java 和 NYTCorpusDocumentParser.java，而是自己使用 python 的 xml 解析工具 xml.dom.minidom 包。我将每一年的所有文件的信息都写在同一个文件里放到年目录下，方便以后处理的时候读取数据。具体解析 xml 文件的过程是这样的

```
files = os.listdir(dayPath)
for file in files:
    if file.startswith('.') or file.endswith('.txt'):
        continue
    fout.write(file + '\n')
    dom = mini.parse(dayPath + '/' + file)
    classifiers = dom.getElementsByTagName('classifier')
    classes = set()
    for c in classifiers:
        if c.getAttribute('type') == 'taxonomic_classifier':
            cla = c.firstChild.data
            arr = cla.split('/')
            if len(arr) >= 3:
                classes.add(arr[2])
            else:
                classes.add(arr[-1])
    #print classes
    for c in classes:
        fout.write(c + '\t')
    fout.write('\n')
    block = dom.getElementsByTagName('block')
    for b in block:
        if b.getAttribute('class') == 'full_text':
            paragraphs = b.getElementsByTagName('p')
            text = ''
            for p in paragraphs:
                data = p.firstChild.data.lower()
                for char in data:
                    if (char >= 'a' and char <= 'z') \
                            or (char == ' ') or (char == '\n' or char == '\r'):
                        text += char
                text += ' '
            #print text
            fout.write(text + '\n-----\n')
```

首先通过 dom = mini.parse(fileName)得到解析过的 xml 文件对象，这个函数会把 xml 文件建立成一棵树。然后用 dom.getElementsByTagName('classifier')得到

所有 classifier 标签，但是该标签下有很多内容，不是每个内容都是需要的分类属性，通过 getAttribute('type') == 'taxonomic_classifier'可以得到文件的分类属性。

解析过 xml 文件后接着很重要的一步就是确定每个文件的类。分类的难度在于每个文件都属于很多个类，并且这些类有层次上的差别，而且对于同一个文件来说，不同层次的类可能具有不同的地位，比如如下的这个文件

```
<location class="indexing_service">MASSACHUSETTS</location>
<classifier class="online_producer" type="taxonomic_classifier">Top/News</classifier>
<classifier class="online_producer" type="taxonomic_classifier">Top/News/U.S./U.S. States, Territories and Possessions/
Massachusetts</classifier>
<classifier class="online_producer" type="taxonomic_classifier">Top/News/U.S./New England</classifier>
<classifier class="online_producer" type="taxonomic_classifier">Top/News/Sports</classifier>
<classifier class="online_producer" type="taxonomic_classifier">Top/Features/Travel/Guides/Destinations/North America</
classifier>
<classifier class="online_producer" type="taxonomic_classifier">Top/Features/Travel/Guides/Destinations/North America/United
States</classifier>
<classifier class="online_producer" type="taxonomic_classifier">Top/Features/Travel/Guides/Destinations/North America/United
States/Massachusetts</classifier>
```

其中有一个 taxonomic_classifier 是 Top/News/U.S./U.S. States, Territories and Possessions/Massachusetts。按照层次分类，这个文件对应的各级层次应该是 news – US – territories and possessions – massachusetts，但是并不能说这个文件依次属于 news 类，US 类，territories and possessions 类，因为其中的 US 和 territories and possessions 并没有出现在这个文件的 taxonomic_classifier 中，所以在不同层次的分类中，第二层的 news 类可能比第三层的 US 类具有更重要的地位，同样第五层的 Massachusetts 类也比第三层的 US 类有更重要的地位，这就给文件的分类造成很大的不确定性，不知道根据什么标准确定一个文件所属的类别。为了避免分类太细导致出现的类过于繁杂，我把每个文件的分类截取到第三层，如果类属性没有三层就取最细致的一层。对于不同层次的类别忽略了包含关系，把它们看做不同的类别。以上述文件为例，Top/News 说明这个文件属于 news 类，Top/News/U.S./U.S. States, Territories and Possessions/Massachusetts 和 Top/News/U.S./New England 都说明文件属于 U.S 类，Top/News/Sports 说明文件属于 Sports 类，而 Top/Features/Travel/Guides/Destinations/North America，
Top/Features/Travel/Guides/Destinations/North America/United States 和 Top/Features/Travel/Guides/Destinations/North America/United States/Massachusetts 三个类别都说明文件属于 Travel 类，需要注意的是，虽然后面三个类属性从第三级的 Features 开始不同于前面几个类，但是由于 Top/Features 并没有单独出现在文件的类属性中，所以没有把该文件归入 Features 类

确定好文件的分类后，就要提取文件的文本内容。用 dom.getElementsByTagName('block')拿到所有 block 元素，通常第一个 block 是文本的导言 lead 部分，需要的正文内容在 full_text 部分，通过 getAttribute('class') == 'full_text'找到对应的内容。提取内容的时候只提取了英文字符、空格和换行符，去掉了所有数字和标点。

对于每一年中的所有文件，将读取到的文件名、类别和正文内容放到同一个文件下，格式是

第一行：文件名

第二行：类 1 \t 类 2 \t 类 3 ...

后面多行：正文

最后一行：-----

由于正文中包括换行符，没办法将一个文件的正文放在一行内，所以在最后一行加上-----分割每篇文章。例如

```
0000004.xml
Business
lead  company reports  canaveral international otc year to sept    revenue   net inc  b
share earns   shares outst   bafter an extraordinary gain on extinguishment of debt of
company reports  canaveral international otc year to sept    revenue   net inc  b share
earns   shares outst   bafter an extraordinary gain on extinguishment of debt of
-----
```

接着根据单词的长度和出现的频率对每篇文本进行过滤。由于长度短的单词很可能是无意义的虚词、代词、介词、冠词等，它们对分类没有帮助，所以去掉长度小于等于 3 的单词。在每一年的目录下首先遍历文件，将长度大于 3 的单词放入一个 dictionary，记录每个单词出现的次数。通过观察这些单词的出现次数发现，$\frac{1}{4}$分位数是 1，中位数是 2，$\frac{3}{4}$分位数是 8，但是均值是 102.292，说明在这些文本中，单词的出现次数两级分化严重，绝大部分单词只出现了很少次数，一小部分单词出现了非常多次。出现很少次数的单词对分类产生的影响很小，但是增加了特征的维数，可以删掉这些单词进行降维。删掉单词的阈值设置为均值 102。于是第二遍遍历的时候，只将出现次数在均值的单词输出到新文件里。

最后根据单词的 tf-idf 改造单词的权重。在很多文档中都出现过的单词对分类的作用较小，只在个别文档中才出现的单词具有代表性，权重应更大。再给每篇文档的单词的次数做标准化。这些工作都是在工具包 scikit-learn 中的 TfidfVectorizer.fit_tranform()函数中做的

2. 训练和测试

代码如下

```
x = TfidfVectorizer().fit_transform(X)
y = MultiLabelBinarizer().fit_transform(label)
print '%d documents, %d words, %d categories' % (x.shape[0], x.shape[1], y.shape[1])
classifiers = [LinearSVC(), BernoulliNB(), LogisticRegression()]
clfNames = ['svm', 'naive bayes', 'logistic regression']
scorers = ['accuracy', 'precision_weighted', 'recall_weighted']
for i in range(0, 3):
    classifier = OneVsRestClassifier(classifiers[i])
    print clfNames[i]
    for scorer in scorers:
        t1 = time.clock()
        scores = cross_val_score(classifier, x, y, cv = 5, scoring=scorer)
        t2 = time.clock()
        #print 'trained for %ds' % (t2 - t1)
        print '%s =' % scorer,
        for cvs in scores:
            print '%.1f%%' % (cvs * 100),
        print ''
```

训练的时候依次构建了 svm、伯努利朴素贝叶斯、逻辑斯蒂克回归分类器，对每种分类器使用 cross-validation，给出了 accuracy，precision 和 recall 值。结果如下

```
106104 documents, 20509 words, 5 categories
svm
accuracy = 67.5% 68.5% 66.4% 66.8% 67.2%
precision_weighted = 87.7% 87.8% 87.3% 87.4% 87.7%
recall_weighted = 87.9% 88.1% 88.0% 88.0% 87.5%
naive bayes
accuracy = 20.0% 18.6% 20.3% 19.0% 19.7%
precision_weighted = 75.8% 75.9% 74.8% 75.0% 75.5%
recall_weighted = 74.7% 74.2% 72.5% 72.9% 74.7%
logistic regression
accuracy = 68.6% 69.3% 67.6% 68.5% 68.1%
precision_weighted = 88.0% 88.1% 88.1% 88.1% 88.2%
recall_weighted = 88.1% 88.2% 88.1% 88.1% 87.7%
```

3. 结果分析

我比较了三种分类方法的准确率，看出朴素贝叶斯在多标签多类别的分类中准确率很低，SVM 和逻辑斯蒂克回归都有较高的准确率。

需要指出的是，由于进行的是多标签的分类(multlabels)，对一次正确的分类会有多种定义，可以是预测结果和真实结果完全匹配才算正确，也可以是预测结果和真实结果有交集就叫正确，上述分类中的一次正确指的是预测的结果集合完全匹配真实的结果集合才叫正确，这种情况下 accuracy 能达到 70%说明分类效果是不错的。

然后我尝试了一下采取第二种正确率的定义，找出预测集合和真实集合中的交集，代码和结果如下

```
def selfCrossValidation(x, y):
    classifiers = [LinearSVC(), BernoulliNB(), LogisticRegression()]
    clfNames = ['svm', 'naive bayes', 'logistic regression']
    for i in range(0, 3):
        classifier = OneVsRestClassifier(classifiers[i])
        print clfNames[i]
        xTrain, xTest, yTrain, yTest = train_test_split(x, y)
        classifier.fit(xTrain, yTrain)
        yPred = classifier.predict(xTest)
        yTest = yTest
        right = 0
        total = len(yPred)
        for i in range(0, yPred.shape[0]):
            for j in range(0, yPred.shape[1]):
                if yPred[i][j] == 1 and yTest[i][j] == 1:
                    right += 1
                    break
        print 'accuracy = %.1f' % (float(right) / total * 100)
```

```
106104 documents, 20509 words, 5 categories
svm
accuracy = 98.2
naive bayes
accuracy = 92.2
logistic regression
accuracy = 98.4
```

可以看出，采用这种方式定义一个分类算作正确的话，accuracy 非常高，这说明分类器虽然不能完全正确地给出每个文档属于的详细分类，但是至少能非常准确地给出一个正确的分类

五. 代码说明

extract.py 逐层解压原始的 tgz 文件
parsexml.py 解析 xml 文件，提取正文
preprocess.py 预处理，去掉出现次数很少的单词
classify.py 分类