链滴

sklearn-文本分析

作者：Zing

原文链接：https://ld246.com/article/1463123451349

来源网站：链滴

本章节的目的是通过一个实际的问题来介绍scikit-learn的主要文本分析工具。该问题是：分析有20主题的文本文件（新闻帖）。

在本章节中，我们会接触到如下内容：

- 加载文件内容和类别
- 抽取适合机器学习的特征向量
- 训练线性模型来拟合分类
- 使用网格搜索来寻找适合特征抽取和分类的参数配置

#开始

在开始该教程之前，你必须安装scikit-learn和所有需求的依赖。

安装相关的请查看installation

该教程的源码可以在你的scikit-learn文件夹下找到：

```
scikit-learn/doc/tutorial/text_analytics/
```

教程文件下，应该包含了如下文件：

- *.rst files - 使用sphinx写的教程文档
- data - 本教程将用到的数据集
- skeletons - 练习题的不完全示例脚本
- solutions - 练习题的答案

你可以将skeletons复制到你硬盘上的文件夹下，并重命名为sklearn_tut_workspace，这样你就可以辑自己的练习题解决方法，同时也不影响原来的内容：

```
% cp -r skeletons work_directory/sklearn_tut_workspace
```

机器学习算法需要数据。到每个$TUTORIAL_HOME/data 子文件价下，运行 fetch_data.py 脚本。

例如：

```
% cd $TUTORIAL_HOME/data/languages
% less fetch_data.py
% python fetch_data.py
```

#加载“Twenty Newsgroups”数据集

这是“Twenty Newsgroups”数据集的官方描述

20 Newsgroups 数据集是大约20000新闻报道文档的集合，大致覆盖了20类不同的新闻报道。这些档最初是由Ken Lang为了支撑他的论文“Newsweeder: Learning to filter netnews”收集的。20 N wsgroups 数据集很快在机器学习处理文本技术实验中流行起来，常用于文本分类和聚类。

接下来，我们将使用sklearn内建的数据集加载器加载20 newsgroups 数据集。当然，你也可以在网下载数据集，再用sklearn.datasets.load_files 指向解压出来的20news-bydate-train子文件夹。

为了节约时间，在第一个例子中，我们只是关注20类中的4类新闻报道：

```
>>> categories = ['alt.atheism', 'soc.religion.christian',
...               'comp.graphics', 'sci.med']
```

现在我们加载属于上述4类的新闻的文件：

```
>>> from sklearn.datasets import fetch_20newsgroups
>>> twenty_train = fetch_20newsgroups(subset='train',
...     categories=categories, shuffle=True, random_state=42)
```

返回的数据集是sklearn中的bunch实体：包含的字段信息和数据，可以像python中的dict或object样访问。target_names属性保存了类别：

```
>>> twenty_train.target_names
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
```

加载到内容中的文件数据存储在 data属性中。也可以使用filenames属性访问：

```
>>> len(twenty_train.data)
2257
>>> len(twenty_train.filenames)
2257
```

打印加载的第一个文件的第一行：

```
>>> print("\n".join(twenty_train.data[0].split("\n")[:3]))
From: sd345@city.ac.uk (Michael Collier)
Subject: Converting images to HP LaserJet III?
Nntp-Posting-Host: hampton

>>> print(twenty_train.target_names[twenty_train.target[0]])
comp.graphics
```

有监督学习算法在训练集中需要每个文档和对应的类别属性。在这个例子中，类别是新闻报道的类别同时也是每个文档的父文件夹的名字。类别属性用整数代表按顺序存储在target属性中：

```
>>> twenty_train.target[:10]
array([1, 1, 3, 3, 3, 3, 3, 2, 2, 2])
```

可以用以下方法还原类别的真正名称：

```
>>> for t in twenty_train.target[:10]:
...     print(twenty_train.target_names[t])
...
comp.graphics
```

```
comp.graphics
soc.religion.christian
soc.religion.christian
soc.religion.christian
soc.religion.christian
soc.religion.christian
sci.med
sci.med
sci.med
```

你可以注意到样本已经被随机洗牌，这对以下这种情况特别有用：你只是选择第一个样本来快速训练型，并以训练的结果来启发之后的正式训练。

#从文本文件抽取特征

为了对文本文件使用机器学习，首先我们需要将文本内容转化为数值特征向量。

##Bags of words

最直观的方法就是抽取有代表性的单词：

1. 为训练集中每个文档中出现的每个单词分配一个固定的数字id（建立从单词映射到数字索引的dict）
2. 对每个文档i，计算每个单词w出现的次数并存在 X[i, j]，其中特征j是单词w在1中分配的id值。

由Bags of words方法产生的向量的维度n_features是语料库中不同单词的数量：约大于100,000

若样本数量 n_samples == 10000，特征向量X用类型为float32的numpy array表示，那么需要 100 0 * 100000 * 4 bytes = 4GB 内存，即使对于目前的计算机来说也是很勉强的。

幸运的是，特征X中的大部分值为0，因为给定的文档中使用的单词不超过几千个。因此，我们认为 ba s of words 的结果 是典型的 高维系数数据集。我们可以通过只存储向量中非零的部分来节约内存。

scipy.sparse 矩阵正是为了解决这种问题设计的数据结构，sklearn内建中已经支持了这种数据结构。

##使用sklearn进行分词（Tokenizing text）

文本预处理，分词和过滤被包含在高级组件中，可以用于创建特征字典和将文档转化为特征向量：

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> count_vect = CountVectorizer()
>>> X_train_counts = count_vect.fit_transform(twenty_train.data)
>>> X_train_counts.shape
(2257, 35788)
```

CountVectorizer 支持计算 N-grams 单词或字符序列。一旦fit完成，CountVectorizer建立起特征引的dict：

```
>>> count_vect.vocabulary_.get(u'algorithm')
4690
```

单词表中单词的索引值指向其在整个训练语料库中的出现次数。

##将出现次数转化为频率

计算出现次数是一个好的开端，但是存在如下问题：更长的文档中单词的平均出现次数会比短文档的高，即使他们的主题是一致的。

为了避免出现上述可能的差异，使用文档中每个单词出现的数量除以该文档单词的总数量：这个新特称为tf （Term Frequencies，词频）。

另一个需要考虑的问题是，一个文档的语料库越小，则每个语料包含的信息量越大。因此需要削减语库大的文档中单词特征的权重。

这种削减方法是 tf-idf（Term Frequency times Inverse Document Frequency）

tf 和 tf-idf 可以通过下面代码计算：

```
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> tf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
>>> X_train_tf = tf_transformer.transform(X_train_counts)
>>> X_train_tf.shape
(2257, 35788)
```

在上面示例代码中，我们先使用 fit(...)方法使用数据调整estimator，接着使用transform(...)方法将我的计数矩阵转化为 tf-idf 表示。直接使用 fit_transform(..) 方法将这两个步骤可以合并到一起以减少些中间计算。 以下代码实现的功能和上面的代码一直：

```
>>> tfidf_transformer = TfidfTransformer()
>>> X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
>>> X_train_tfidf.shape
(2257, 35788)
```

#训练分类器

现在我们已经得到了特征，我们可以训练一个分类器并尝试预测测试数据的类别。我们将以naive Bays开始，该分类器可以为我们的问题提供一个良好的基线。sklearn包含了naive Bayes的几个变种版，其中多项式版本最适合于词频。

```
>>> from sklearn.naive_bayes import MultinomialNB
>>> clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)
```

预测新的文档的类别，我们需要用和上述几乎一直的方法抽出特征。不同的是，我们直接调用TfidfTrasformer的transform方法，而不是 fit_transform。因为之前我们已经使用训练样本fit过了。

```
>>> docs_new = ['God is love', 'OpenGL on the GPU is fast']
>>> X_new_counts = count_vect.transform(docs_new)
>>> X_new_tfidf = tfidf_transformer.transform(X_new_counts)

>>> predicted = clf.predict(X_new_tfidf)

>>> for doc, category in zip(docs_new, predicted):
...     print('%r => %s' % (doc, twenty_train.target_names[category]))
...
'God is love' => soc.religion.christian
```

```
'OpenGL on the GPU is fast' => comp.graphics
```

#建立管道

为了更加简便地使用vectorizer => transformer => classifier 工作流程，sklearn提供了Pipeline类该类类似于混合分类器：

```
>>> from sklearn.pipeline import Pipeline
>>> text_clf = Pipeline([('vect', CountVectorizer()),
...                      ('tfidf', TfidfTransformer()),
...                      ('clf', MultinomialNB()),
... ])
```

其中vect, tfidf 和 clf 是随意命名的。我们将在下面网格搜索一节中看到他们的用法。现在训练整个型（包括特征抽取、转化、分类器训练），仅仅需要通过以下命令：

```
>>> text_clf = text_clf.fit(twenty_train.data, twenty_train.target)
```

#使用测试集评估

评估模型预测的正确率是非常简单的：

```
>>> import numpy as np
>>> twenty_test = fetch_20newsgroups(subset='test',
...     categories=categories, shuffle=True, random_state=42)
>>> docs_test = twenty_test.data
>>> predicted = text_clf.predict(docs_test)
>>> np.mean(predicted == twenty_test.target)
0.834...
```

我们获得83.4%的准确率。现在我们看看能否使用线性SVM模型获得更好的结果（线性SVM被普遍地为是最好的文本分类算法，虽然比naive Bayes慢一些）。我们仅仅需要将管道中的分类器进行特换可。

```
>>> from sklearn.linear_model import SGDClassifier
>>> text_clf = Pipeline([('vect', CountVectorizer()),
...                      ('tfidf', TfidfTransformer()),
...                      ('clf', SGDClassifier(loss='hinge', penalty='l2',
...                                            alpha=1e-3, n_iter=5, random_state=42)),
... ])
>>> _ = text_clf.fit(twenty_train.data, twenty_train.target)
>>> predicted = text_clf.predict(docs_test)
>>> np.mean(predicted == twenty_test.target)
0.912...
```

此外sklearn还提供了更详细的效果评估工具：

```
>>> from sklearn import metrics
>>> print(metrics.classification_report(twenty_test.target, predicted,
...     target_names=twenty_test.target_names))
...
                    precision    recall  f1-score   support

         alt.atheism       0.95      0.81      0.87       319
       comp.graphics       0.88      0.97      0.92       389
             sci.med       0.94      0.90      0.92       396
soc.religion.christian       0.90      0.95      0.93       398

         avg / total       0.92      0.91      0.91      1502


>>> metrics.confusion_matrix(twenty_test.target, predicted)
array([[258,  11,  15,  35],
       [  4, 379,   3,   3],
       [  5,  33, 355,   3],
       [  5,  10,   4, 379]])
```

由混淆矩阵可以看出，新闻报道中的atheism主题比comp.graphics更容易被混淆。

#使用网格搜索调整参数

我们已经在TfidfTransformer中使用了一些参数，如“use_idf”。同样，分类器也会有很多参数，如ultinomialNB分类器包含平滑参数aipha，SGDClassifier包含惩罚参数alpha等等。

逐个调整pipline中的参数是不明智的，我们需要一个穷举搜索方法（exhaustive search）帮助我们找参数网格中最好的参数组合。

```
>>> from sklearn.grid_search import GridSearchCV
>>> parameters = {'vect__ngram_range': [(1, 1), (1, 2)],
...               'tfidf__use_idf': (True, False),
...               'clf__alpha': (1e-2, 1e-3),
... }
```

显然的，穷举搜索方法开销是较大的。如果我们有多核CPU，我们可以通过设置n_jobs = -1，让网搜索计算时使用所有的cpu进行并行计算：

```
>>> gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1)
```

网格搜索实例和普通的sklearn模型一样。让我们在一个较小的数据集中机械能网格搜索，以缩短计时间：

```
>>> gs_clf = gs_clf.fit(twenty_train.data[:400], twenty_train.target[:400])
```

GridSearchCV的fit方法返回一个分类器，我们可以使用它进行预测：

```
>>> twenty_train.target_names[gs_clf.predict(['God is love'])]
'soc.religion.christian'
```

但是另一方面，这个分类器是相当巨大和笨拙的。我们可以使用grid_scores_属性从该对象中获取最的参数列表。

```
>>> best_parameters, score, _ = max(gs_clf.grid_scores_, key=lambda x: x[1])
>>> for param_name in sorted(parameters.keys()):
...     print("%s: %r" % (param_name, best_parameters[param_name]))
...
clf__alpha: 0.001
tfidf__use_idf: True
vect__ngram_range: (1, 1)

>>> score
0.900...
```

练习题连接

#路在何方

以下是几点建议可以帮助你在学完本指导后，在sklearn路上走得更远：

- 尝试玩一玩CountVectorizer下的analyzer 和 token normalisation
- 如果你没有类属性，尝试使用聚类方法获得
- 如果每个文档有多个类属性，可以看看 Multiclass and multilabel section
- 尝试使用Truncated SVD 进行潜在语义分析
- 使用Out-of-core分类方法学习没办法加载到主存的数据
- 尝试使用 Hashing Vectorizer 替换CountVectorizer