# 第6章

# 文本分类

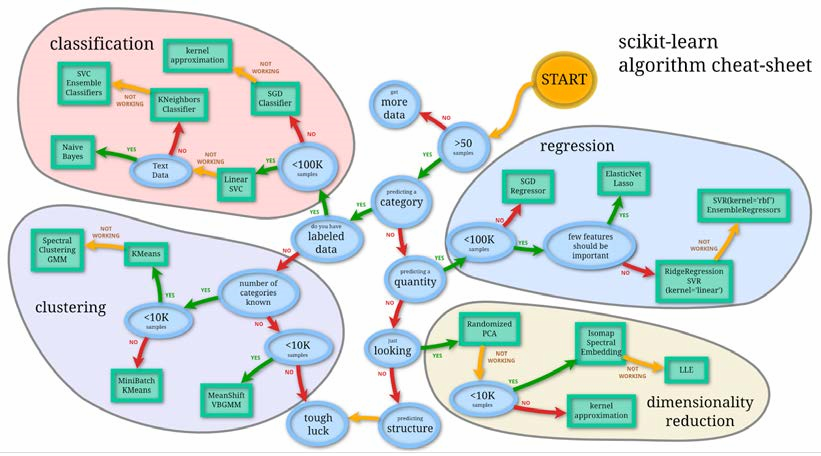
我们在上一章中对NLP领域中一些最常见的工具和预处理步骤进行了详细的讨论。在本章，我们将会利用之前所学习的大部分知识来构建一种复杂度最高的NLP应用。我们将会给出一个解决文本分类问题的通用方法，并带领您从零开始用尽可能简短的代码来构建一个文本分类器。除此之外，我们还将给出一份适用于文本分类问题的分类算法清单。

虽然我们会对部分最常见的文本算法进行一些讨论，但这些也都只是蜻蜓点水式的介绍，对于那些想要了解具体细节和相关数学北京的读者，我们也列出了许多在线资源和相关书籍，以供参考。我们将会尽力帮助读者了解他们所要了解的知识，使他们能够着手使用本章提供的代码片段。尽管，文本分类问题是NLP领域中一个很典型的用例，但在这里我们并不打算使用NLTK来做这件事，因为scikit-learn库中包含了更为广泛的分类算法，使用该库来执行文本挖掘会更为有效。

在阅读完本章之后，我们希望：

* 您应该学会所有的文本分类算法并理解它们。
* 您应该学会如何使用点对点的管道来构建文本分类器，并用scikit-learn和NLTK来实现它

下面我们来看一下scikit-learn库在机器学习应用上的功能列表：



credit : scikit-learn

我们可以将上面这个功能列表当成一个流程图来走。这样一来我们就等于有了一个明确的方向，譬如知道哪种方法对应的是哪个问题、分类器之间的迁移依赖于多大规模的标记样本等。对于构建实用程序来说，从这张流程图入手是一个不错的开始，它在大多数情况下都是适用的。在本章，我们大多数时候关注的是文本数据，尽管scikit-learn库也可以处理其他类型的数据，但我们（为了降低维度）在这里将只探讨文本中的文本分类，文本聚类以及主题检测问题，并带您构建一些炫酷的NLP应用。当然，本章不会对机器学习、分类和聚类的概念进行详细说明，因为对于这些内容，我们可以在Web上找到充足的可用资料。我们会在谈到相关语料库时给出这些概念的更多细节，不过在此之前，我们先来做个复习。

## 机器学习

机器学习技术可以被分成两大类型：监督式学习和无监督式学习：

* **监督式学习**：该技术基于若干预先标记的历史样本，它用来预测未知测试样本的算法主要有以下两类：
  + **分类算法** ：该算法主要用于预测测试样本是否属于某些类型当中的一个。如果算法中只有两个类，这就是一个二元分类问题; 否则就是一个多元分类问题。
  + **回归算法**：该算法主要用于预测某种连续性的变量，例如房价和股票指数等。
* **无监督式学习**：当我们没有任何标签数据却仍需要预测类标签时，我们就会用到这种被称之为无监督式学习的技术。如果我们需要基于相关项之间的相似性来对它们进行分组，这就是在解决聚类问题。而如果我们是需要在较低维度上表示高维数据的话，那就更多的是一个降维问题。
* **半监督式学习**：它在分类上应该属于监督式学习任务和技术，但同时也会使用未标记的数据来进行调校。从名称上就可以看出，这更像是一种介于监督式学习和无监督式学习之间的技术，我们会基于少量标记数据和大量未标记数据来构建具有预测能力的机器学习模型。
* **增强式学习**：这是一种利用奖罚机制来实现的机器学习形式，它没有指定的完成任务方式。

如果我们认为自己已经理解了这些不同的机器学习算法，就可以来猜猜看下面这些都属于哪种机器学习问题：

* 下个月的天气预报
* 从数百万笔交易中检测出欺诈行为
* Google的优先收件箱
* 亚马逊的推荐机制
* Google新闻
* 自动驾驶汽车

## 文本分类

对于文本分类，最简单的定义就是要基于文本内容来对该文本进行分类。通常情况下，目前所有的机器学习方法和算法都是根据数字/变量特征来编写的。所以这里最重要的问题之一，就是我们如何在语料库中用数字特征的形式来表示文本。各种技术文献提出了各种不同的转换方式，下面我们从最简单，使用最广泛使用的转换方式着手。

现在，为了帮助读者理解文本分类的具体过程，我们来看看垃圾邮件这个现实问题。 在WhatsApp和SMS的世界中，我们难免会收到许多垃圾邮件。下面，我们就来想想如何借助文本分类算法来解决垃圾邮件检测这个现实问题。 我们会用这个运行实例来贯穿本章的内容。首先我们要求读者手动标记一下这几则真实的SMS例文：

SMS001 ['spam', 'Had your mobile 11 months or more? U R entitled to

Update to the latest colour mobiles with camera for Free! Call The Mobile

Update Co FREE on 08002986030']

SMS002 ['ham', "I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today."]

|  |
| --- |
| 读者也可以从下面链接中下载到一份类似的已完成标注的数据集。当然，请确保您创建的是一个和上述例子显示内容相同的CSV文件。下面代码中'SMSSpamCollection'所对应的就是这个文件。  https://archive.ics.uci.edu/ml/datasets/ SMS+Spam+Collection |

我们现在要做的第一件事是按照之前几章所学到的数据清理、标识化处理以及词干提取等知识来对SMS进行清理，使其内容更简洁一些。 下面我们就来写一个基本的、用于文本清理的函数：

>>> import nltk

>>> from nltk.corpus import stopwords

>>> from nltk.stem import WordNetLemmatizer

>>> import csv

>>> def preprocessing(text):

>>> text = text.decode("utf8")

>>> # tokenize into words

>>> tokens = [word for sent in nltk.sent\_tokenize(text) for word in nltk.word\_tokenize(sent)]

>>> # remove stopwords

>>> stop = stopwords.words('english')

>>> tokens = [token for token in tokens if token not in stop]

>>> # remove words less than three letters

>>> tokens = [word for word in tokens if len(word) >= 3]

>>> # lower capitalization

>>> tokens = [word.lower() for word in tokens]

>>> # lemmatize

>>> lmtzr = WordNetLemmatizer()

>>> tokens = [lmtzr.lemmatize(word) for word in tokens]

>>> preprocessed\_text= ' '.join(tokens)

>>> return preprocessed\_text

我们在*第3章：词性标注*中已经讨论过了与标记化处理、词形还原以及停用词相关的知识。在上述代码中[[1]](#footnote-1)，我只是对SMS进行了解析并对其内容做了清理，获得了较为简洁的SMS文本。在接下来的几行代码中，我们将会创建两个列表，分别用以获取被清理之后的所有SMS内容以及类标签。用**ML（Machine learning）**术语来说就是获取所有的X和Y：

>>> smsdata = open('SMSSpamCollection') # check the structure of this file!

>>> smsdata\_data = []

>>> sms\_labels = []

>>> csv\_reader = csv.reader(sms,delimiter='\t')

>>> for line in csv\_reader:

>>> # adding the sms\_id

>>> sms\_labels.append( line[0])

>>> # adding the cleaned text We are calling preprocessing method

>>> sms\_data.append(preprocessing(line[1]))

>>> sms.close()

在继续任何下一步动作之前，我们要确保自己所用的系统中已经安装了scikit-learn库。

>>> import sklearn

|  |
| --- |
| 如果这句代码除了错，或者您在安装scikit的过程中遇到了一些错误，可以按照下面链接中的内容来安装scikit：<http://scikit-learn.org/stable/install.html> |

## 取样操作

一旦我们以列表的形式持有了整个语料库，接下来就要对其进行某种形式的取样操作。 通常来说，对语料库的整体取样方式与下图中开发调校集、开发测试集和测试集的取样方式是类似的，整个练习背后的思路是要避免调校过度。如果我们将所有数据点都反馈给该模型，那么算法就会基于整个语料库来进行机器学习，但这些算法在真实测试中针对的是不可见数据。在非常简单的词汇环境中，如果我们在模型学习过程中使用的是全体数据，那么尽管分类器在该数据上能得到很好的执行，但其结果是不稳健的。原因在于我们一直只在给定数据上执行出最佳结果，但这样它是学不会如何处理未知数据的。

（图：图中翻译

Corpus：语料库

Development set：开发集

Training set：调校集

Dev-test set：开发测试集

Test set：测试

）

要想解决此类问题，最好的办法是将整个语料库划分成两个主要集合。在建模练习中，我们应该要避开开发集和测试集，只用开发测试集来完成建模操作。在我们完成整个建模练习之后，再将其结果放到我们之前搁置的测试集合中来进行预测。这样一来，如果该模型在该集合上表现良好，我们就可以确信它对任何新的数据样本都可以进行准确而稳健的预测。

取样本来就是一个非常复杂的操作流程，机器学习社区一直在对其深入研究，它本质上是一个应对许多数据编程和调校过度问题的补救措施。当然为简单起见，本章将只进行基本取样，下面我们只对语料库进行70:30的划分：

>>> trainset\_size = int(round(len(sms\_data)\*0.70))

>>> # i chose this threshold for 70:30 train and test split.

>>> print 'The training set size for this classifier is ' + str(trainset\_ size) + '\n'

>>> x\_train = np.array([''.join(el) for el in sms\_data[0:trainset\_size]])

>>> y\_train = np.array([el for el in sms\_labels[0:trainset\_size]])

>>> x\_test = np.array([''.join(el) for el in sms\_data[trainset\_ size+1:len(sms\_data)]])

>>> y\_test = np.array([el for el in sms\_labels[trainset\_size+1:len(sms\_ labels)]])or el in sms\_labels[trainset\_size+1:len(sms\_labels)]])

>>> print x\_train

>>> print y\_train

* 如果我们将全体数据都用作调校数据，您认为情况会怎样？
* 如果我们面对的是一个非常不平衡的样本，情况又会怎样？

|  |
| --- |
| 如果想要了解更多可用的取样技术，请访问以下链接：  http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cross\_validation. |

下面我们将视线跳转到另一件事上：就是我们要讲整个文本转换成向量形式。这种形式被称之为**词汇文档矩阵（term-document matrix）**。如果我们有必要为这个给定例子构建一个词汇文档矩阵，它看起来应该像下面这样：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TDM** | **Anymore** | **Call** | **camera** | **color** | **cried** | **enough** | **Entitled** | **free** | **Gon** | **had** | **latest** | **Mobile** |
| SMS1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 1 | 0 | 3 |
| SMS2 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

当然，文本文档也可以用所谓的**BOW（Bag of Word）**来表示。这也是文本挖掘和其他相关应用中最常见的表示方法之一。基本上，我们不必去考虑这些单词到其所产生的表示方法之间的任何上下文环境。

如果想要用Python来生成一个类似词汇文档矩阵，我们就需要用到scikit中的向量化器（Vectorizer）：

>>> from sklearn.feature\_extraction.text import CountVectorizer

>>> sms\_exp=[ ] >>>for line in sms\_list:

>>> sms\_exp.append(preprocessing(line[1]))

>>> vectorizer = CountVectorizer(min\_df=1)

>>> X\_exp = vectorizer.fit\_transform(sms\_exp)

>>> print "||".join(vectorizer.get\_feature\_names()) >>>print X\_exp.toarray() array([[

1, 0, 1, 1, 1, 0, 0, 1, 2, 0, 1, 0, 1, 3, 1, 0,

0, 0, 1, 0, 0, 2, 0, 0], [ 0, 1, 0, 0, 0, 1, 1, 0,

0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, ]])

计数向量为我们开了个好头，但它在使用过程中会遇到一个问题：即较长文档所获得的平均计数值会高于较短的文档，即使在讨论主题相同的时也是如此。

|  |
| --- |
| 如果想要避免这些潜在的误差，我们只要将文档中每个单词出现的次数去除以该文档中的单词总数就行了。这个新的特征值叫做tf（即term frequencies）。 |

Tf之上还有另一个更细致的改进，那就是对语料库中许多文档中的出现的词汇进行降格加权。通过这种方式，我们就可以得到减少那些只在该语料库的某一小部分中出现的的信息。

这种降格加权我们称之为tf-idf（即**term frequency–inverse document frequency**）。幸运的是，scikit库也提供了相应的实现方式，具体如下：

>>> from sklearn.feature\_extraction.text import TfidfVectorizer

>>> vectorizer = TfidfVectorizer(min\_df=2, ngram\_range=(1, 2), stop\_ words='english', strip\_accents='unicode', norm='l2')

>>> X\_train = vectorizer.fit\_transform(x\_train)

>>> X\_test = vectorizer.transform(x\_test)

我们现在得到了一个矩阵格式的文本，它与我们在任何机器学习作业（machine learning exercise）中得到的结果是一样的。 现在，X\_train和X\_test可以被用于所有机器学习算法中的分类处理了。所以接下来我们要来看看在文本分类这个语境中，最常用的机器学习算法有哪些。

### 朴素贝叶斯法

下面就来构建我们的第一个文本分类器吧。首先来看朴素贝叶斯分类器。朴素贝叶斯分类器依赖于贝叶斯算法，它本质上是一个根据给定特征/属性，基于某种条件概率为样本赋予某个类别标签的模型。在这里，我们将用频率/伯努利数来预估先验概率和后验概率。

（公式）

朴素算法往往会假设其中所有的特征都将是相互独立的，这样对于文本环境来说看起来会直观一些。但令人惊讶的是，朴素贝叶斯算法在大多数实际用例中的表现也相当良好。

朴素贝叶斯（NB）法的另一个伟大之处在于它非常简单，实现起来很容易，评分也很简单。我们只需要将各频率值存储起来，并计算出概率。它无论在调校时还是测试（评分）时的速度都很快。基于所有的这些原因，大多数的文本分类问题都会用它来做基准

下面我们就来写一下这个分类器的实现代码：

>>> from sklearn.naive\_bayes import MultinomialNB

>>> clf = MultinomialNB().fit(X\_train, y\_train)

>>> y\_nb\_predicted = clf.predict(X\_test)

>>> print y\_nb\_predicted

>>> print ' \n confusion\_matrix \n '

>>> cm = confusion\_matrix(y\_test, y\_pred)

>>> print cm

>>> print '\n Here is the classification report:'

>>> print classification\_report(y\_test, y\_nb\_predicted) confusion\_matrix [[1205 5]

[26 156]]

（图 图中翻译：

Classified：分类

Actuals：实际情况

True positive：真阳性

False positive：假阳性

False negative：假阴性

True negative：真阴性

）

该方法将测试集中的所有1,392个样本读取到了混合矩阵中，其中有1205个真阳性判例和156个真阴性判例。但我们也预测到了5个假阴性判例和26个假阳性判例。现在，对于这样一个典型的二元分类器，我们有不同的测量指标。

下面，我们就来给出其中几个最常见的分类测量指标的定义：

（公式）

（公式）

（公式）

（公式）

现在我们来看一下分类报告：

Precision recall f1-score support

ham 0.97 1.00 0.98 1210

spam 1.00 0.77 0.87 182

avg / total 0.97 0.97 0.97 1392

有了上述定义，我们现在可以对结果一目了然了。因此事实上，上述所有的测量指标看起来都挺不错的，这意味着我们的分类器执行得准确而稳健。在这里，我会强烈建议您以更多的选项来查看该模块的测量指标，用来分析该分类器所得到的结果。这之中最重要，且最平衡的指标是f1指标（这只是关于精确率和反馈率的调和平均指标）。该指标之所以被广泛使用，是因为它给出了一个具有更高覆盖面的、优质的分类算法。另外，准确度也直观地告诉了我们真样本在所有样品中占了多少。精确率与反馈率也都有其各自的含义，精确率所讨论的是该分类器能得到多少真阳性判例以及它们的覆盖面，而反馈率则可以让我们详细了解自己能从这个关于真阳性和假阴性的判例池中得到多大的准确性。

|  |
| --- |
| 如果想了解scikit库中各种类的更多信息，请访问以下链接：  http://scikit-learn.org/stable/modules/classes. html#module-sklearn.metrics |

下面来看另一个更重要的过程，我们要根据自己理解的模型真正深入地查看这个模型，通过查看实际特征来分辨阳性和阴性的判例类。在这里，我们只写了一段非常小的代码来生成前n个特征并打印出它们。具体如下：

>>> feature\_names = vectorizer.get\_feature\_names()

>>> coefs = clf.coef\_

>>> intercept = clf.intercept\_

>>> coefs\_with\_fns = sorted(zip(clf.coef\_[0], feature\_names))

>>> n = 10

>>> top = zip(coefs\_with\_fns[:n], coefs\_with\_fns[:-(n + 1):-1])

>>> for (coef\_1, fn\_1), (coef\_2, fn\_2) in top:

>>> print('\t%.4f\t%-15s\t\t%.4f\t%-15s' % (coef\_1, fn\_1, coef\_2, fn\_2))

-9.1602 10 den -6.0396 free

-9.1602 15 -6.3487 txt

-9.1602 1hr -6.5067 text

-9.1602 1st ur -6.5393 claim

-9.1602 2go -6.5681 reply

-9.1602 2marrow -6.5808 mobile

-9.1602 2morrow -6.5858 stop

-9.1602 2mrw -6.6124 ur

-9.1602 2nd innings -6.6245 prize

-9.1602 2nd ur -6.7856 www

在上述代码中，我们只是从向量化器中读取了所有的特征名称，并获取了与给定特征相关的系数，然后将其中前10个特征打印出来。如果想要打印更多的特征，只需要修改代码中的n值即可。如果我们仔细观察这些特征，就会得到很多关于该模型的信息，以及更多与特征选择和其他参数相关的建议，譬如说关于预处理过程、单元/二元语法，词干提取、标记化处理等方面的建议。举个例子，如果我们查看垃圾过滤器的前几个特征，可以看到2morrow, 2nd innings和一些非常明显的数字。我们会看到对于阳性判例类（即垃圾信息）来说。“free”是一个非常明显被突出的词汇，许多垃圾信息里都有与免费优惠与交易相关的内容。当然，prize、www、claim等其它词汇也同样值得关注的。

|  |
| --- |
| 更多细节请参考http://scikitlearn.org/stable/ modules/naive\_bayes.html. |

### 决策树

决策树是最古老的预测建模技术之一，对于给定的特征和目标，基于该技术的算法会尝试构建一个相应的逻辑树。使用决策树的算法有很多种类，我们这里要介绍的是其中最着名和使用最广泛的算法之一：**CART**。

CART会用特性构造一些二叉树结构，并构造出一个阈值，用于从每个节点中产生大量的信息。下面我们就通过编写代码来获取一个CART分类器：

>>> from sklearn import tree

>>> clf = tree.DecisionTreeClassifier().fit(X\_train.toarray(), y\_train)

>>> y\_tree\_predicted = clf.predict(X\_test.toarray())

>>> print y\_tree\_predicted

>>> print ' \n Here is the classification report:'

>>> print classification\_report(y\_test, y\_tree\_predicted)

这里唯一的区别在于调校数据集的输入格式。我们需要将之前的稀疏矩阵格式修改成**NumPy**数组，因为scikit库树模块只接受一个NumPy数组。

通常情况下，只有在特征数量非常少时，树结构才是一个不错的选择。因此，尽管乍看之下我们在这里得到了不错的结果，但实际上人们很少会在文本分类问题上使用树结构。但从另一方面来说，树结构也有确实有一些积极面。它仍然是一个最直观的算法，简单易懂，易于实现。基于树结构来实现的分类算法也很多，譬如ID3，C4.5和C5等。scikit-learn库所采用的是CART算法的优化版本。

### 随机梯度下降

**随机梯度下降（Stochastic gradient descent，简称SGD）法**是一种既简单又非常有效的，适用于线性模型的方法。尤其在目标样本数量（和特征数量）非常庞大时，其作用会特别突出。如果参照之前那个功能列表，我们会发现SGD是许多文本分类问题的一站式解决方案。另外，由于它也能照顾到规范化问题并提供不同的损失函数，所以对于线性模型的实验工作来说它也是个很好的选择。

SGD算法有时候也被称之为**最大熵（Maximum entropy，简称MaxEnt）算法**，它会用不同的（坡面）损失函数（loss functions）和惩罚机制来适配针对分类问题与回归问题的线性模型。例如当loss = log时，它适配的是一个对数回归模型，而当loss = hinge时，它适配的则是一个线性的支持向量机（SVM）。

下面来看一个具体的SGD算法用例：

>>>from sklearn.linear\_model import SGDClassifier

>>>from sklearn.metrics import confusion\_matrix

>>>clf = SGDClassifier(alpha=.0001, n\_iter=50).fit(X\_train, y\_train)

>>>y\_pred = clf.predict(X\_test)

>>>print '\n Here is the classification report:'

>>>print classification\_report(y\_test, y\_pred)

>>>print ' \n confusion\_matrix \n '

>>>cm = confusion\_matrix(y\_test, y\_pred)

>>>print cm

其分类结果报告如下：

precision recall f1-score support

ham 0.99 1.00 0.99 1210

spam 0.96 0.91 0.93 182

avg / total 0.98 **0.98 0.98 1392**

下面是其最翔实的特征列表：

-1.0002 sir 2.3815 ringtoneking

-0.5239 bed 2.0481 filthy

-0.4763 said 1.8576 service

-0.4763 happy 1.7623 story

-0.4763 might 1.6671 txt

-0.4287 added 1.5242 new

-0.4287 list 1.4765 ringtone

-0.4287 morning 1.3813 reply

-0.4287 always 1.3337 message

-0.4287 and 1.2860 call

-0.4287 plz 1.2384 chat

-0.3810 people 1.1908 text

-0.3810 actually 1.1908 real

-0.3810 urgnt 1.1431 video

### 逻辑回归

逻辑回归（Logistic regression）是一种针对分类问题的线性模型。它在某些文献中也被称之为対元逻辑（logit regression）、最大熵（MaxEnt）分类法或对数线性分类器。在这个模型中，我们会用一个対元函数来进行建模，以概率的方式来描述单项试验的可能结果。

作为优化问题来说，这里由L2二元类所惩罚的逻辑回归可以用以下成本函数来进行最小化：

（公式）

类似地，由L1二元类所规范的逻辑回归则应该用以下函数来解决其优化问题：

（公式）

### 支持向量机

**支持向量机（Support vector machines，简称SVM）**是目前在机器学习领域中最为先进的算法。

SVM属于非概率分类器。SVM会在无限维空间中构造出一组超平面，它可被应用在分类、回归或其他任务中。直观来说，我们可以通过一个超平面来实现良好的分类划界，这个超平面应该距离最接近调校数据点的那些类最远（我们称这个距离为功能边界），因为在一般情况下，这个边界越大，分类器的规模就越小。

下面我们就用scikit库来构建一个最高级的监督式学习算法：

>>>from sklearn.svm import LinearSVC

>>>svm\_classifier = LinearSVC().fit(X\_train, y\_train)

>>>y\_svm\_predicted = svm\_classifier.predict(X\_test)

>>>print '\n Here is the classification report:'

>>>print classification\_report(y\_test, y\_svm\_predicted)

>>>cm = confusion\_matrix(y\_test, y\_pred)

>>>print cm

其分类结果报告与之前相同：

precision recall f1-score support

ham 0.99 1.00 0.99 1210

spam 0.97 0.90 0.93 182

avg / total 0.98 0.98 0.98 1392

confusion\_matrix [[1204 6] [ 17 165]]

下面是其最翔实的特征列表：

-0.9657 road 2.3724 txt

-0.7493 mail 2.0720 claim

-0.6701 morning 2.0451 service

-0.6691 home 2.0008 uk

-0.6191 executive 1.7909 150p

-0.5984 said 1.7374 www

-0.5978 lol 1.6997 mobile

-0.5876 kate 1.6736 50

-0.5754 got 1.5882 ringtone

-0.5642 darlin 1.5629 video

-0.5613 fullonsms 1.4816 tone

-0.5613 fullonsms com 1.4237 prize

这些结果绝对是我们到目前为止所试过的所有监督算法中最好的。介绍完了这个算法之后，我们对监督式分类器的介绍也就到此结束了。目前有数以百万计的书籍在介绍各种不同的机器学习算法，即使是针对个别特定算法，可供选择的书也有很多。但我们强烈建议读者务必要在对上述算法有了一个深入了解之后，然后再在实际应用中使用它们。

## 随机森林算法

随机森林是一种以不同决策树组合为基础来进行评估的合成型分类器。事实上，它比较适合用于在各种数据集的子样本上构建某种多决策树型的分类器。另外，该森林中的每个树结构都建立在一个随机的最佳特征子集上。最后，启用这些树结构的动作也为我们找出了所有随机特征子集中的最佳子集。 总而言之，随机森林是当前众多分类算法中表现最佳的算法之一。

下面来看一个具体的随机森林算法用例：

>>> from sklearn.ensemble import RandomForestClassifier

>>> RF\_clf = RandomForestClassifier(n\_estimators=10)

>>> predicted = RF\_clf.predict(X\_test)

>>> print '\n Here is the classification report:'

>>> print classification\_report(y\_test, predicted)

>>> cm = confusion\_matrix(y\_test, y\_pred)

>>> print cm

|  |
| --- |
| 对于仍然希望用NLTK库来解决文本分类问题的读者，我们推荐您阅读下面链接中的内容：<http://www.nltk.org/howto/classify.html> |

## 文本聚类

与文本相关的另一个问题族系是无监督式分类问题。关于这类问题，我们最常见的一种问题描述是“我手里有数以百万计的（非结构化数据）文档，是否能找到一种方式将它们分组，以便赋予其有意义的类别？”到目前为止，只要我们掌握了被标注的数据样本，就可以构建一个相应的监督式算法，但这些我们之前已经讨论过了。在这里，我们需要使用无监督的方式来对这些文本文档进行分组。

文本聚类法（有时也叫聚类法）是目前最为常见的无监督式分组方式之一。使用聚类法的算法有很多种选择。其中，我本人最常用的是**k均值法（k-means）**或**层次聚类法（hierarchical clustering）**。下面，我们就来分别来看看它们是如何与语料库搭配使用的。

### K-me ans

Very intuitively, as the name suggest, we are trying to find k groups around the mean of the data points. So, the algorithm starts with picking up some random data points as the centroid of all the data points. Then, the algorithm assigns all the data points to it's nearest centroid. Once this iteration is done, recalculation of the centroid happens and these iterations continue until we reach a state where the centroids don't change (algorithm saturate).

非常直观地，顾名思义，我们试图找到围绕数据点的平均值的k组。 因此，算法从拾取一些随机数据点开始，作为所有数据点的质心。 然后，算法将所有数据点分配给它最近的质心。 一旦这个迭代完成，重心的重新计算发生，并且这些迭代继续，直到我们达到质心不改变的状态（算法饱和）。

There is a variant of the algorithm that uses mini batches to reduce the computation time, while still attempting to optimize the same objective function.

有一种算法的变体，其使用迷你批次来减少计算时间，同时仍然尝试优化相同的目标函数。

|  |
| --- |
| Mini batches are subsets of the input data randomly sampled in each training iteration. These options should always be tried once your dataset is really huge and you want less training time. |

An example of K-means is as follows:

K均值的示例如下：

>>> from sklearn.cluster import KMeans, MiniBatchKMeans

>>> true\_k=5

>>> km = KMeans(n\_clusters=true\_k, init='k-means++', max\_iter=100, n\_ init=1)

>>> kmini = MiniBatchKMeans(n\_clusters=true\_k, init='k-means++', n\_init=1, init\_size=1000,

batch\_size=1000, verbose=opts.verbose)

>>># we are using the same test,train data in TFIDF form as we did in text classification

>>>km\_model=km.fit(X\_train)

>>>kmini\_model=kmini.fit(X\_train)

>>>print "For K-mean clustering "

>>>clustering = collections.defaultdict(list)

>>>for idx, label in enumerate(km\_model.labels\_):

>>> clustering[label].append(idx)

>>>print "For K-mean Mini batch clustering "

>>>clustering = collections.defaultdict(list)

>>>for idx, label in enumerate(kmini\_model.labels\_):

>>> clustering[label].append(idx)

In the preceding code, we just imported scikit-learn's kmeans / minibatchkmeans and fitted the same training data that we were using in the running examples. We can also print a cluster for each sample using the last three lines of the code.

在上面的代码中，我们刚刚导入scikit-learn的kmeans / minibatchkmeans，并且使用了我们在运行示例中使用的相同的训练数据。 我们还可以使用代码的最后三行为每个样本打印一个集群。

## 文本中的主题建模

The other famous problem in the context of the text corpus is finding the topics of the given document. The concept of topic modeling can be addressed in many different ways. We typically use **LDA** (**Latent Dirichlet allocation**) and LSI (Latent semantic indexing) to apply topic modeling text documents.

在文本语料库的上下文中的另一个着名的问题是找到给定文档的主题。主题建模的概念可以以许多不同的方式来解决。我们通常使用LDA（潜在Dirichlet分配）和LSI（潜在语义索引）来应用主题建模文本文档。

Typically, in most of the industries, we have huge volumes of unlabeled text documents. In case of an unlabeled corpus to get the initial insights of the corpus, a topic model is a great option, as it not only gives us topics of relevance, but also categorizes the entire corpus into number of topics given to the algorithm.

通常，在大多数行业中，我们有大量的未标记的文本文档。在一个未标记的语料库获得语料库的初步洞察的情况下，一个主题模型是一个伟大的选择，因为它不仅给我们相关的主题，而且将整个语料库归类到给算法的主题数量。

We will use a new Python library "gensim" that implements these algorithms for us. So, let's jump to the implementation of LDA and LSI for the same running SMS dataset. Now, the only change to the problem is that we want to model different topics in the SMS data and also want to know which document belongs to which topic. A better and more realistic use case could be to run topic modeling on the entire Wikipedia dump to find different kinds of topics that have been discussed there, or to run topic modeling on billions of reviews/complaints from customers to get an insight of the topics that people discuss.

我们将使用一个新的Python库“gensim”为我们实现这些算法。因此，让我们跳到同一个运行的SMS数据集的LDA和LSI的实现。现在，对问题的唯一改变是我们要在SMS数据中建模不同的主题，并且还想知道哪个文档属于哪个主题。更好和更现实的用例可以是在整个维基百科转储上运行主题建模以找到已经讨论过的不同类型的主题，或者对来自客户的数十亿条评论/投诉执行主题建模以获得对主题的洞察人们讨论。

### 安装gensim

One of the easiest ways to install gensim is using a package manager:

>>>easy\_install -U gensim

Otherwise, you can install it using:

>>>pip install gensim

Once you're done with the installation, run the following command:

>>>import gensim

|  |
| --- |
| If there is any error, go to  https://radimrehurek.com/gensim/install.html. |

Now, let's look at the following code:

>>>from gensim import corpora, models, similarities

>>>from itertools import chain

>>>import nltk

>>>from nltk.corpus import stopwords

>>>from operator import itemgetter

>>>import re

>>>documents = [document for document in sms\_data]

>>>stoplist = stopwords.words('english')

>>>texts = [[word for word in document.lower().split() if word not in stoplist] \ for document in documents]

We are just reading the document in our SMS data and removing the stop words. We could use the same method that we did in the previous chapters to do this. Here, we are using a library-specific way of doing things.

我们只是在我们的短信数据中阅读文档，并删除停用词。 我们可以使用我们在前面章节中所做的相同的方法来做到这一点。 在这里，我们使用一种特定于库的方式来做事情。

|  |
| --- |
| Gensim has all the typical NLP features as well provides some great way to create different corpus formats, such as TFIDF, libsvm, market matrix. It also provides conversion of one to another. |

In the following code, we are converting the list of documents to a BOW model and then, to a typical **TF-IDF** corpus:

在下面的代码中，我们将文档列表转换为BOW模型，然后转换为典型的TF-IDF语料库：

>>>dictionary = corpora.Dictionary(texts)

>>>corpus = [dictionary.doc2bow(text) for text in texts]

>>>tfidf = models.TfidfModel(corpus)

>>>corpus\_tfidf = tfidf[corpus]

Once you have a corpus in the required format, we have the following two methods, where given the number of topics, the model tries to take all the documents from the corpus to build a LDA/LSI model:

一旦你有一个所需的格式的语料库，我们有以下两种方法，给定主题的数量，模型试图从语料库中建立一个LDA / LSI模型的所有文档：

>>>si = models.LsiModel(corpus\_tfidf, id2word=dictionary, num\_topics=100)

>>>#lsi.print\_topics(20)

>>>n\_topics = 5

>>>lda = models.LdaModel(corpus\_tfidf, id2word=dictionary, num\_topics=n\_

topics)

Once the model is built, we need to understand the different topics, what kind of terms represent that topic, and we need to print some top terms related to that topic:

一旦建立了模型，我们需要理解不同的主题，什么样的术语代表该主题，我们需要打印一些与该主题相关的主要术语：

>>> for i in range(0, n\_topics):

>>> temp = lda.show\_topic(i, 10)

>>> terms = []

>>> for term in temp:

>>> terms.append(term[1])

>>> print "Top 10 terms for topic #" + str(i) + ": "+ ",

".join(terms)

Top 10 terms for topic #0: week, coming, get, great, call, good, day, txt, like, wish

Top 10 terms for topic #1: call, ..., later, sorry, 'll, lor, home, min, free, meeting

Top 10 terms for topic #2: ..., n't, time, got, come, want, get, wat, need, anything

Top 10 terms for topic #3: get, tomorrow, way, call, pls, 're, send, pick, ..., text

Top 10 terms for topic #4: ..., good, going, day, know, love, call, yup, get, make

Now, if you look at the output, we have five different topics with clearly different intent. Think about the same exercise for Wikipedia or a huge corpus of web pages, and you will get some meaningful topics that represent the corpus.

现在，如果你看看输出，我们有五个不同的主题，有明显不同的意图。 考虑维基百科或一个庞大的网页语料库的相同练习，你会得到一些有意义的主题代表语料库。

## 参考资料

* http://scikit-learn.org/
* https://radimrehurek.com/gensim/
* https://en.wikipedia.org/wiki/Document\_classification

## 本章小结

The idea behind this chapter was to introduce you to the world of text mining. We want to give you a basic introduction to some of the most common algorithms available with text classification and clustering .We know how some of these concept will help you to build really great NLP applications, such as spam filters, domain centric news feeds, web page taxonomy, and so on. Though we have not used NLTK to classify the module in our code snippets, we used NLTK for all the preprocessing steps. We highly recommend you to use scikit-learn over NLTK for any classification problem. In this chapter, we started with machine learning and the types of problems that it can address. We discussed some of the specifics of ML problems in the context of text. We talked about some of the most common classification algorithms that are used for text classification, clustering, and topic modeling. We also give you enough implementation details to get the job done. I still think you need to read a lot about each and every algorithm separately to understand the theory and to gain in-depth understanding of them.

本章背后的想法是介绍文本挖掘的世界。我们想给你一些基本的介绍一些最常见的算法可用文本分类和聚类。我们知道这些概念如何将帮助您构建真正伟大的NLP应用程序，如垃圾邮件过滤器，域中心新闻Feed，网络页面分类，等等。虽然我们没有使用NLTK在我们的代码片段中分类模块，我们使用NLTK进行所有的预处理步骤。我们强烈建议您对任何分类问题使用scikit-learn over NLTK。在本章中，我们开始了机器学习和它可以解决的问题类型。我们在文本的上下文中讨论了ML问题的一些细节。我们讨论了一些最常用的分类算法，用于文本分类，聚类和主题建模。我们还给你足够的实现细节，以完成工作。我仍然认为你需要阅读很多关于每个和每个算法单独地理解理论和获得对它们的深入理解。

We also provided you an entire pipeline of the process that you need to follow in case of any text mining problem. We covered most of the practical aspects of machine learning, such as sampling, preprocessing, model building, and model evaluation.

我们还为您提供了整个过程的流程，在任何文本挖掘问题的情况下，您都需要遵循。我们涵盖了机器学习的大多数实用方面，如采样，预处理，模型建立和模型评估。

The next chapter will also not be directly related to NLTK/NLP, but it will be a great tool for a data scientist/NLP enthusiast. In most of NLP problems, we deal with unstructured text data, and the Web is one of the richest and biggest data sources available for this. Let's learn how to gather data from the Web and how to efficiently use it to build some amazing NLP applications.

下一章也将不直接涉及NLTK / NLP，但它将是一个数据科学家/ NLP爱好者的伟大工具。在大多数NLP问题中，我们处理非结构化文本数据，而Web是可用于此的最丰富和最大的数据源之一。让我们学习如何从Web收集数据，以及如何有效地使用它来构建一些惊人的NLP应用程序。

1. 译者注：此处的原文是In the following code，但从文本实际布局来看，这里指的应该是这段文字上面的代码。疑为作者笔误。 [↑](#footnote-ref-1)