# 第6章

# 文本分类

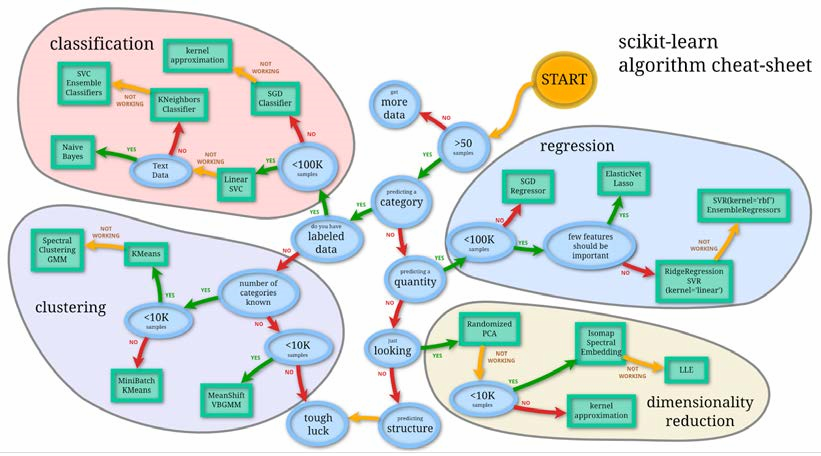
我们在上一章中对NLP领域中一些最常见的工具和预处理步骤进行了详细的讨论。在本章，我们将会利用之前所学习的大部分知识来构建一种复杂度最高的NLP应用。我们将会给出一个解决文本分类问题的通用方法，并带领您从零开始用尽可能简短的代码来构建一个文本分类器。除此之外，我们还将给出一份适用于文本分类问题的分类算法清单。

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库在机器学习方面的功能列表：



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：监督学习和无监督学习：

* **监督型学习**：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。
* **半监督学习**：它在分类上应该属于监督学习任务和技术，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学习和无监督学习之间的技术，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vectorizer模块：

>>> 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。

### 朴素贝叶斯方法

Let's build your first text classifier. Let's start with a Naive Bayes classifier. Naive Bayes relies on the Bayes algorithm and essentially, is a model of assigning a class label to the sample based on the conditional probability class given by features/attributes. Here we deal with frequencies/bernoulli to estimate prior and posterior probabilities.

让我们建立你的第一个文本分类器。 让我们从朴素贝叶斯分类器开始。 朴素贝叶斯依赖贝叶斯算法，本质上是一个基于由特征/属性给出的条件概率类别向样本分配类别标签的模型。 这里我们处理频率/ bernoulli来估计先验和后验概率。

（公式）

The naive assumption here is that all features are independent of each other, which looks counter intuitive in the case of text. However, surprisingly, Naive Bayes performs quite well in most of the real-world use cases.

这里的天真假设是所有特征都是彼此独立的，这在文本的情况下看起来是直观的。 然而，令人惊讶的是，朴素贝叶斯在大多数真实世界的用例中表现相当好。

Another great thing about NB is that it's too simple and very easy to implement and score. We need to store the frequencies and calculate the probabilities. It's really fast in case of training as well as test (scoring). For all these reasons, in most of the cases of text classification, it serves as a benchmark.

NB的另一个伟大的事情是，它太简单，很容易实现和得分。 我们需要存储频率并计算概率。 它在训练和测试（评分）的情况下真的很快。 由于所有这些原因，在大多数文本分类的情况下，它作为基准。

Let's write some code to achieve this classifier:

让我们编写一些代码来实现这个分类器：

>>>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]]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | Classified | | |  | | --- | | A c t u a l s | | |  |  |  | | --- | --- | --- | | True Positive |  | False Negarive | | | | |  |  |  | | --- | --- | --- | | False Positive |  | True Negarive | | | |

The way to read the confusion matrix is that from all the 1,392 samples in the test set, there were 1205 true positives and 156 true negative cases. However, we also predicted 5 false negatives and 26 false positives. There are different ways of measuring a typical binary classification.

读取混淆矩阵的方法是，从测试集中的所有1,392个样本中，有1205个真阳性和156个真阴性病例。 然而，我们还预测了5个假阴性和26个假阳性。 有不同的方法来测量典型的二进制分类。

We have given definitions of some of the most common measures used in classification measures:

我们给出了在分类测量中使用的一些最常见的措施的定义：

（公式）

（公式）

（公式）

（公式）

Here is the classification report:

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

With the preceding definition, we can now understand the results clearly. So, effectively, all the preceding metrics look good, which means that our classifier is performing accurately, and is robust. I would highly recommend that you look into the module metrics for more options to analyze the results of the classifier. The most important and balanced metric is the f1 measure (which is nothing but the harmonic mean of precision and recall), which is used widely because it gives a better picture of the coverage and the quality of the classification algorithms. Accuracy intuitively tells us how many true samples have been covered from all the samples. Precision and recall both have significance, while precision talks about how many true positives it got and what else got covered, hand recall gives us details about how accurate we are from the pool of true positives and false negatives.

根据上述定义，我们现在可以清楚地了解结果。 因此，有效地，所有前面的度量看起来不错，这意味着我们的分类器执行准确，是鲁棒的。 我强烈建议您查看模块指标以了解更多选项，以分析分类器的结果。 最重要的和平衡的度量是f1度量（其仅仅是精度和回忆的调和平均值），其被广泛使用，因为它给出了分类算法的覆盖和质量的更好的图片。 准确度直观地告诉我们所有样品中覆盖了多少真实样品。 精确度和回忆都有意义，而精确性谈论有多少真正的阳性和什么被覆盖，手回忆让我们详细了解我们从真阳性和假阴性池的准确性。

|  |
| --- |
| For more information on various scikit classes visit the following link: http://scikit-learn.org/stable/modules/classes. html#module-sklearn.metrics |

The other more important process we follow to understand our model is to really look deep into the model by looking at the actual features that contribute to the positive and negative classes. I just wrote a very small snippet to generate the top *n* features and print them. Let's have a look at them:

另一个更重要的过程，我们了解我们的模型是真正深入的模型，通过查看有助于积极和消极类的实际特征。 我只是写了一个非常小的代码片段来生成前n个功能并打印它们。 让我们看看他们：

>>>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

In the preceding code, I just read all the feature names from the vectorizer, got the coefficients related to the given feature, and then printed the first-10 features. If you want more features, just modify the value of *n*. If we look closely just at the features, we get a lot of information about the model as well as more suggestions about our feature selection and other parameters, such as preprocessing, unigrams/bigrams, stemming, tokenizations, and so on. For example, if you look at the top features of ham you can see that 2morrow, 2nd innings, and some of the digits are coming very significantly. We can see on the positive class (spam ) term "free" comes out a very significant term which is intuitive while many spam messages will be about some free offers and deal. Some of the other terms to note are prize, www, claim.

在上面的代码中，我只是从向量化器读取所有的特征名称，得到与给定特征相关的系数，然后打印前10个特征。 如果你想要更多的功能，只需修改n的值。 如果我们仔细观察特征，我们得到很多关于模型的信息，以及关于特征选择和其他参数的更多建议，例如预处理，单字符/双字母，词干，标记化等等。 例如，如果你看看火腿的顶部特征，你可以看到2morrow，第二局，和一些数字是非常显着。 我们可以看到积极的类（垃圾邮件）术语“免费”出来一个非常重要的术语是直观的，而许多垃圾邮件将是一些免费的优惠和交易。 其他一些要注意的术语是奖品，www，claim。

|  |
| --- |
| For more details, refer to http://scikitlearn.org/stable/ modules/naive\_bayes.html. |

### 决策树

Decision trees are one of the oldest predictive modeling techniques, where for the given features and target, the algorithm tries to build a logic tree. There are multiple algorithms that exist for decision trees. One of the most famous and widely used algorithm is **CART**.

决策树是最古老的预测建模技术之一，其中对于给定的特征和目标，算法尝试构建逻辑树。 存在用于决策树的多个算法。 最着名和广泛使用的算法之一是CART。

CART constructs binary trees using this feature, and constructs a threshold that yields the large amount of information from each node. Let's write the code to get a CART classifier:

CART使用此功能构造二叉树，并构造一个阈值，从每个节点产生大量的信息。 让我们编写代码以获取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)

The only difference is in the input format of the training set. We need to modify the sparse matrix format to a **NumPy** array because the scikit tree module takes only a NumPy array.

唯一的区别是在训练集的输入格式。 我们需要将稀疏矩阵格式修改为NumPy数组，因为scikit树模块只需要一个NumPy数组。

Generally, trees are good when the number of features are very less. So, although our results look good here, people hardly use trees in text classification. On the other hand, trees have some really positive sides to them. It is still one the most intuitive algorithms and is very easy to explain and implement. There are many implementations of tree-based algorithms, such as ID3, C4.5, and C5. scikit-learn uses an optimized version of the CART algorithm.

通常，当特征的数量非常少时，树是好的。 所以，虽然我们的结果在这里看起来不错，人们很少在文本分类中使用树。 另一方面，树对他们有一些真正积极的一面。 它仍然是一个最直观的算法，很容易解释和实现。 有许多基于树的算法的实现，例如ID3，C4.5和C5。 scikit-learn使用CART算法的优化版本。

### 随机梯度下降

**Stochastic gradient descent** (**SGD**) is a simple, yet very efficient approach that fits linear models. It is particularly useful when the number of samples (and the number of features) is very large. If you follow the cheat sheet, you will find SGD to be the one-stop solution for many text classification problems. Since it also takes care of regularization and provides different losses, it turns out to be a great choice when experimenting with linear models.

随机梯度下降（SGD）是一种简单，但非常有效的方法，适合线性模型。 当样本的数量（和特征的数量）非常大时，这是特别有用的。 如果你按照备忘单，你会发现SGD是许多文本分类问题的一站式解决方案。 由于它也负责正则化和提供不同的损失，它是一个伟大的选择，当实验线性模型。

SGD, also known as **Maximum entropy** (**MaxEnt**), provides functionality to fit linear models for classification and regression using different (convex) loss functions and penalties. For example, with loss = log, fits a logistic regression model, while with loss = hinge, it fits a linear support vector machine (SVM).

SGD，也称为最大熵（MaxEnt），提供了使用不同（凸）损失函数和惩罚来拟合线性模型以用于分类和回归的功能。 例如，损失= log，拟合逻辑回归模型，而损失=铰链，它适合线性支持向量机（SVM）。

An example of SGD is as follows:

SGD的示例如下：

>>>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

Here is the classification report:

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**

Most informative features:

-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

Logistic regression is a linear model for classification. It's also known in the literature as logit regression, maximum-entropy classification (MaxEnt), or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logit function.

逻辑回归是分类的线性模型。 它在文献中也称为logit回归，最大熵分类（MaxEnt）或对数线性分类器。 在这个模型中，描述单个试验的可能结果的概率使用logit函数建模。

As an optimization problem, the L2 binary class' penalized logistic regression minimizes the following cost function:

作为优化问题，L2二进制类的惩罚逻辑回归使以下成本函数最小化：

（公式）

Similarly, L1 the binary class' regularized logistic regression solves the following optimization problem:

类似地，L1二进制类的正则逻辑回归解决以下优化问题：

（公式）

### 支持向量机

**Support vector machines** (**SVM**) is currently the-state-of-art algorithm in the field of machine learning.

支持向量机（SVM）是目前在机器学习领域的最先进的算法。

SVM is a non-probabilistic classifier. SVM constructs a set of hyperplanes in an infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by a hyperplane that has the largest distance to the nearest training data point of any class (the so-called functional margin), since in general, the larger the margin, the lower the size of classifier.

SVM是非概率分类器。 SVM在无限维空间中构造一组超平面，可用于分类，回归或其他任务。 直观地，通过具有到任何类的最近训练数据点的最大距离（所谓的功能裕度）的超平面来实现良好的分离，因为一般来说，裕度越大，分类器的尺寸越小。

Let's build one of the most sophisticated supervised learning algorithms with scikit:

让我们用scikit构建一个最复杂的监督学习算法：

>>>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

Here is the classification report for the same:

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]]

The most informative features:

-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

These are definitely the best results so far from all the supervised algorithms we have tried. Now with this, I will stop with supervised classifiers. There are millions of books available related to the different machine learning algorithms; even for individual algorithms, there are many books that are available for you. I would highly recommend you to have a deep understanding of any of the preceding algorithms before you use them for any of the real-world applications.

这些绝对是我们试过的所有监督算法的最好的结果。 现在有了这个，我将停止与监督分类器。 有数百万本书可用于与不同的机器学习算法相关; 即使对于个别算法，也有很多书可供您使用。 我强烈建议您对任何前面的算法有深入的了解，然后再将它们用于任何真实应用程序。

## 随机森林算法

A random forest is an ensemble classifier that estimates based on the combination of different decision trees. Effectively, it fits a number of decision tree classifiers on various subsamples of the dataset. Also, each tree in the forest built on a random best subset of features. Finally, the act of enabling these trees gives us the best subset of features among all the random subsets of features. Random forest is currently one of best performing algorithms for many classification problems.

随机森林是基于不同决策树的组合进行估计的合成分类器。 实际上，它适合于在数据集的各种子样本上的多个决策树分类器。 此外，森林中的每个树都建立在一个随机最佳的特征子集上。 最后，启用这些树的行为为我们提供了所有随机子集的特征中最好的子集。 随机森林当前是许多分类问题的最佳表现算法之一。

An example of Random forest is as follows:

随机森林的一个例子如下：

>>>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

|  |
| --- |
| People who still want to work with NLTK for text classification. Please go through the following link:  <http://www.nltk.org/howto/classify.html> |

## 文本聚类

The other family of problems that can come with text is unsupervised classification. One of the most common problem statements you can get is "I have these millions of documents (unstructured data). Is there a way I can group them into some meaningful categories?". Now, once you have some samples of tagged data, we could build a supervised algorithm that we talked about, but here, we need to use an unsupervised way of grouping text documents.

可以带有文本的另一个问题族是无监督分类。 你可以得到的最常见的问题陈述之一是“我有这些数百万的文档（非结构化数据）。有一种方法，我可以将它们分组成一些有意义的类别？ 现在，一旦你有一些标记数据的样本，我们可以构建一个监督的算法，我们谈论，但在这里，我们需要使用无监督的方式分组文本文档。

Text clustering is one of the most common ways of unsupervised grouping, also known as, clustering. There are a variety of algorithms available using clustering. I mostly used **k-means** or **hierarchical** clustering. I will talk about both of them and how to use them with a text corpus.

文本聚类是无监督分组（也称为聚类）的最常见的方法之一。 使用聚类有多种算法可用。 我大多使用k-means或层次聚类。 我将谈论他们和如何使用它们与文本语料库。

### K-means

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)