

36讲文本聚类：如何过滤冗余的新闻



你好，我是黄申。

前两节，我讲了向量空间模型，以及如何在信息检索领域中运用向量空间模型。向量空间模型提供了衡量向量之间的距离或者相似度的机制，而这种机制可以衡量查询和被查询数据之间的相似程度，而对于文本检索来说，查询和文档之间的相似程度可作为文档的相关性。

实际上，除了文档的相关性，距离或者相似度还可以用在机器学习的算法中。今天，我们就来聊聊如何在聚类算法中使用向量空间模型，并最终实现过滤重复文章。

聚类算法

在概率统计模块中，我们介绍了分类（Classification/Categorization）和回归（Regression）这两种监督式学习（Supervised Learning）。监督式学习通过训练资料学习并建立一个模型，并依此模型对新的实例进行预测。

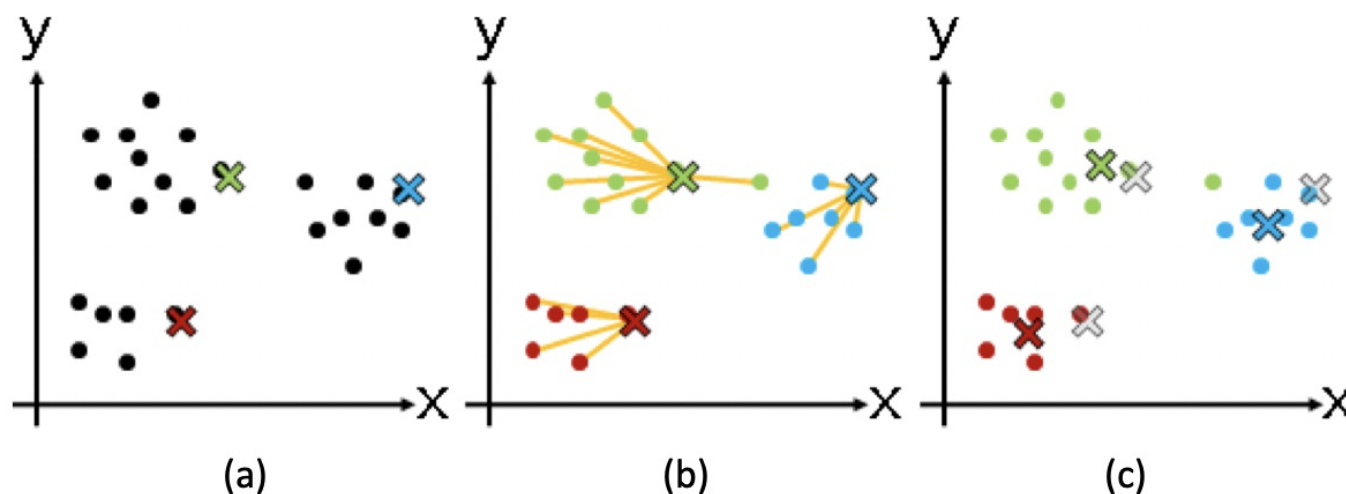
不过，在实际场景中，我们常常会遇到另一种更为复杂的情况。这时候不存在任何关于样本的先验知识，而是需要机器在没人指导的情形下，去将很多东西进行归类。由于缺乏训练样本，这种学习被称为“非监督学习”（Unsupervised Learning），也就是我们通常所说的聚类（Clustering）。在这种学习体系中，系统必须通过一种有效的方法发现样本的内在相似性，并把数据对象以群组（Cluster）的形式进行划分。

谈到相似性，你可能已经想到了利用特征向量和向量空间模型，这确实是可行的方法。不过，为了让你全面了解在整个非监督式学习中，如何运用向量空间，让我先从一个具体的聚类算法开始。

这个算法的名称是K均值（K-Means）聚类算法，它让我们可以在一个任意多的数据上，得到一个事先定好群组数量（K）的聚类结果。这种算法的中心思想是：尽量最大化总的群组内相似度，同时尽量最小化群组之间的相似度。群组内或群组间的相似度，是通过各个成员和群组质心相比较来确定的。想法很简单，但是在样本数量达到一定规模后，希望通过排列组合所有的群组划分，来找到最大总群组内的相似度几乎是不可能的。于是人们提出如下的求近似解的方法。

1. 从N个数据对象中随机选取k个对象作为质心，这里每个群组的质心定义是，群组内所有成员对象的平均值。因为是第一轮，所以第i个群组的质心就是第i个对象，而且这时候我们只有这一个组员。
2. 对剩余的对象，测量它和每个质心的相似度，并把它归到最近的质心所属的群组。这里我们可以说距离，也可以说相似度，只是两者呈现反比关系。
3. 重新计算已经得到的各个群组的质心。这里质心的计算是关键，如果使用特征向量来表示的数据对象，那么最基本的方法是取群组内成员的特征向量，将它们的平均值作为质心的向量表示。
4. 迭代上面的第2步和第3步，直至新的质心与原质心相等或相差之值小于指定阈值，算法结束。

我以二维空间为例子，画张图来展示一下数据对象聚类的过程。



在这张图中，(a)、(b)、(c)三步分别展示了质心和群组逐步调整的过程。我们一一看。(a)步骤是选择初始质心，质心用不同颜色的x表示；(b)步骤开始进行聚类，把点分配到最近的质心所在的组；(c)步骤重新计算每个群组的质心，你会发现x的位置发生了改变。之后就是如此重复，进入下一轮聚类。

总的来说，K均值算法是通过不断迭代、调整K个聚类质心的算法。而质心或者群组的中心点，是通过求群组所包含的成员之平均值来计算的。

使用向量空间进行聚类

明白了K均值聚类算法的核心思想，再来理解向量空间模型在其中的运用就不难了。我还是以文本聚类为例，讲讲如何使用向量空间模型和聚类算法，去除重复的新闻。

我们在看新闻的时候，一般都希望不断看到新的内容。可是，由于现在的报道渠道非常丰富，经常会出现热点新闻霸占版面的情况。假如我们不想总是看到重复的新闻，应该怎么办呢？有一种做法就是对新闻进行聚类，那么内容非常类似的文章就会被聚到同一个分组，然后对每个分组我们只选择1到2篇显示就够了。

基本思路确定后，我们可以把整个方法分为三个主要步骤。

第一步，把文档集合都转换成向量的形式。这块我上一节讲过了，你要是不记得了，可以自己回去复习一下。

第二步，使用K均值算法对文档集合进行聚类。这个算法的关键是如何确定数据对象和分组质心之间的相似度。针对这点，我们有两个点需要关注。

- 使用向量空间中的距离或者夹角余弦度量，计算两个向量的相似度。

- 计算质心的向量。K均值里，质心是分组里成员的平均值。所以，我们需要求分组里所有文档向量的平均值。求法非常直观，就是分别为每维分量求平均值，我把具体的计算公式列在这里：

$$x_i = \arg \operatorname{avg}_{j=1}^n (x_{ij})$$

其中， $x_{\{i\}}$ 表示向量的第*i*个分量， $x_{\{ij\}}$ 表示第*j*格向量的第*i*个分量，而 $j=1,2,\dots,n$ 表示属于某个分组的所有向量。

第三步，在每个分类中，选出和质心最接近的几篇文章作为代表。而其他的文章作为冗余的内容过滤掉。

下面，我使用Python里的sklearn库，来展示使用欧氏距离的K均值算法。

Python中的K均值算法

在尝试下面的代码之前，你需要看看自己的机器上是不是已经安装了scikit-learn。Scikit-learn是Python常用的机器学习库，它提供了大量的机器学习算法的实现和相关的文档，甚至还内置了一些公开数据集，是我们实践机器学习算法的好帮手。

首先，我使用sklearn库中的CountVectorizer，对一个测试的文档集合构建特征，也就是词典。这个测试集合有7句话，2句关于篮球，2句关于电影，还有3句关于游戏。具体代码如下：

```
from sklearn.feature_extraction.text import CountVectorizer

#模拟文档集合
corpus = ['I like great basketball game',
          'This video game is the best action game I have ever played',
          'I really really like basketball',
          'How about this movie? Is the plot great?',
          'Do you like RPG game?',
          'You can try this FPS game',
          'The movie is really great, so great! I enjoy the plot']

#把文本中的词语转换为词典和相应的向量
vectorizer = CountVectorizer()
vectors = vectorizer.fit_transform(corpus)

#输出所有的词条（所有维度的特征）
print('所有的词条（所有维度的特征）')
print(vectorizer.get_feature_names())
print('\n')

#输出(文章ID, 词条ID) 词频
print('(文章ID, 词条ID) 词频')
print(vectors)
print('\n')
```

从运行的结果中，你可以看到，整个词典里包含了哪些词，以及每个词在每个文档里的词频。

这里，我们希望使用比词频 tf 更好的 $tf-idf$ 机制，`TfidfTransformer`可以帮助我们做到这点，代码和注释如下：

```
from sklearn.feature_extraction.text import TfidfTransformer

#构建tfidf的值
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))

# 输出每个文档的向量
tfidf_array = tfidf.toarray()
words = vectorizer.get_feature_names()

for i in range(len(tfidf_array)):
    print ("*****第", i + 1, "个文档中，所有词语的tf-idf*****")
    # 输出向量中每个维度的取值
    for j in range(len(words)):
        print(words[j], ' ', tfidf_array[i][j])
    print('\n')
```

运行的结果展示了每个文档中，每个词的 $tfidf$ 权重，你可以自己手动验算一下。

最后，我们就可以进行K均值聚类了。由于有篮球、电影和游戏3个类别，我选择的K是3，并在`KMeans`的构造函数中设置`n_clusters`为3。

```

from sklearn.cluster import KMeans

#进行聚类，在我这个版本里默认使用的是欧氏距离
clusters = KMeans(n_clusters=3)
s = clusters.fit(tfidf_array)

#输出所有质心点，可以看到质心点的向量是组内成员向量的平均值
print('所有质心点的向量')
print(clusters.cluster_centers_)
print('\n')

#输出每个文档所属的分组
print('每个文档所属的分组')
print(clusters.labels_)

#输出每个分组内的文档
dict = {}
for i in range(len(clusters.labels_)):
    label = clusters.labels_[i]
    if label not in dict.keys():
        dict[label] = []
        dict[label].append(corpus[i])
    else:
        dict[label].append(corpus[i])
print(dict)

```

为了帮助你的理解，我输出了每个群组的质心，也就是其中成员向量的平均值。最后，我也输出了3个群组中所包含的句子。在我机器上的运行结果显示，系统可以把属于3个话题的句子区分开来。如下所示：

```
{2: ['I like great basketball game', 'I really really like basketball'], 0: ['This video game is the best action game I ever played'], 1: ['I love playing video games', 'I love playing video games']}
```

不过，由于KMeans具体的实现可能不一样，而且初始质心的选择也有一定随机性，所以你看到的结果可能稍有不同。

总结

这一节，我介绍了如何在机器学习的聚类算法中，使用向量空间模型。在聚类中，数据对象之间的相似度是很关键的。如果我们把样本转换为向量，然后使用向量空间中的距离或者夹角余弦，就很自然的能获得这种相似度，所以向量空间模型和聚类算法可以很容易的结合在一起。

为了给你加深印象，我介绍了一个具体的K均值算法，以及向量空间模型在其中所起到的作用，并通过Python的sklearn代码演示了几个关键的步骤。

向量空间模型和K均值算法的结合，虽然简单易懂，但是一开始怎样选择这个群组的数量，是个关键问题。我今天演示的测试数据很小，而且主题划分的也非常明显。所以我选择聚类的数量为3。

可是在实际项目中，对于一个新的数据集，选择多少比较合适呢？如果这个K值取得太小，群组可能切分太细，每个之间区别不大。如果K值取得太大，群组的粒度又太粗，造成群组内差异比较明显。对非监督式的学习来说，这个参数确实难以得到准确预估。我们可以事先在一个较小的数据集上进行尝试，然后根据结果和应用场景确定一个经验值。

思考题

今天我使用的是sklearn里的KMeans包，它使用了向量间的欧氏距离来进行聚类。你可以尝试实现自己的K均值聚类，并使用向量间的夹角余弦作为相似度的度量。

欢迎留言和我分享，也欢迎你在留言区写下今天的学习笔记。你可以点击“请朋友读”，把今天的内容分享给你的好友，和他一起精进。



程序员的数学基础课

在实战中重新理解数学

黄申

LinkedIn 资深数据科学家



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

精选留言



mickey

encoding=utf-8

```
from sklearn.feature_extraction.text import CountVectorizer
```

#模拟文档集合

```
corpus = ['I like great basketball game',  
'This video game is the best action game I have ever played',  
'I really really like basketball',  
'How about this movie? Is the plot great?',  
'Do you like RPG game?',  
'You can try this FPS game',  
'The movie is really great, so great! I enjoy the plot']
```

#把文本中的词语转换为词典和相应的向量

```
vectorizer = CountVectorizer()
```

```
vectors = vectorizer.fit_transform(corpus)

from sklearn.feature_extraction.text import TfidfTransformer

#构建tfidf的值
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus))

# 输出每个文档的向量
tfidf_array = tfidf.toarray()
words = vectorizer.get_feature_names()

from Bio.Cluster import kcluster

#进行聚类，使用向量间的夹角余弦作为相似度的度量
clusterid, error, nfound = kcluster(tfidf_array, nclusters=3, dist='u')
print(clusterid)
'''
Kmeans 的欧式距离:[0 2 0 1 2 2 1]
Bio 的夹角余弦:[2 0 0 2 0 2 1]
'''
```

2019-03-08 11:29