词嵌入的直觉理解：从计数向量到Word2Vec

2017.07.15 杨德杰 数据科学 阅读量：104

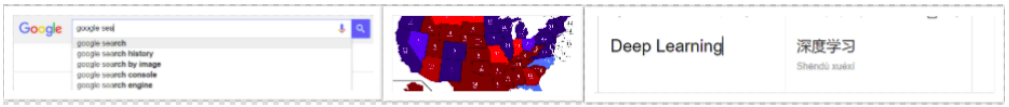
Instruction

我们在开始之前，看看下面的例子：

1.你打开Google搜索一篇关于正在进行的冠军奖杯比赛的新闻文章，你会得到它返回的数以百计的搜索结果。

2.Nate Silver分析了数百万条推文，并在2008年美国总统选举中正确地预测出了50个州中的49个州的结果。

3.你可以在 Google翻译中用英语输入一个句子，并得到一个同义的中文转换。

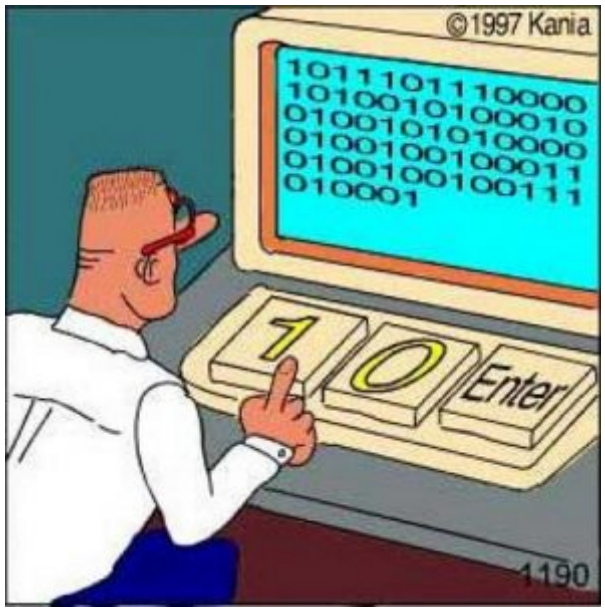


那么上面的例子有什么共同点呢？

菜单

文章目录

扩展又不高效。



由于我们知道在执行处理字符串、文本或者任何大量结果时通常是低效的，那么我们如何使今天的计算机在文本数据上执行聚类，分类等？

当然，电脑可以匹配两个字符串，并告诉你是否相同。 但是，当您搜索梅西（Messi）时，我们如何让电脑告诉您足球或罗纳尔多（Ronaldo）？ 如何

并且所有✓的这些都是通过使用词嵌入或文本的数字表示来实现的，以便计算机可以处理它们。

下面我们将正式看到词嵌入及其不同类型，以及我们如何在实际中来实现它们来执行诸如返回高效的Google搜索结果等任务。

目录

1.什么是词嵌入？

狭义上，词嵌入是将文本转化为数字，并且相同文本可能用不同数字表示。但是在我们深入了解词嵌入的细节之前，应该问下面的问题 - 为什么我们需要词嵌入？

事实证明，许多机器学习算法和几乎所有的深度学习框架都无法处理原始形式的字符串或普通文本。 广义上，他们需要数字作为输入，以执行任何类型的工作，无论是分类，回归等。 而且在具有文本格式存在的大量数据，必定要从中提取知识并构建应用程序，诸如一些现实世界的文本应用像Amazon评论的情感分析，Google的文档或新闻的分类或聚类等。

现在让我们正式定义词嵌入。 词嵌入通常会尝试使用字典将字词映射到向量。 让我们把这个句子分解成更精细的细节，以便有一个清晰的看法。

看看这个例子 - **sentence**（句子）=”Word Embeddings are Word converted into numbers”

这句话中的一个**word**(单词)可能是“Embeddings”或“numbers”等。

这只是一个非常简单的方法来表示向量形式的单词。 我们来看看不同类型的词嵌入（或词向量化）及其优缺点。

2.不同类型的词嵌入

不同类型的词嵌入可以大致分为两类：

基于频率的词嵌入
基于预测的词嵌入
让我们尝试详细了解这些方法。

2.1基于频率的词嵌入

在这个类别下，我们通常会遇到三种类型的向量。

- 计数向量
- 使用TF-IDF的向量
- 使用共生矩阵的向量

让我们详细研究这些向量化方法。

让我们用一个简单的例子来理解这个。

D1: He is a lazy boy. She is also lazy.

D2: Neeraj is a lazy person.

创建的字典可以是语料库中的具有唯一标记的单词：
['He','She','lazy','boy','Neeraj','person']

这里，D = 2，N = 6

大小为2×6的计数矩阵M将被表示为 -

	He	She	lazy	boy	Neeraj	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

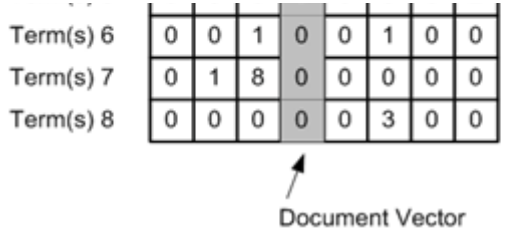
现在，列也可以被理解为矩阵M中的对应单词的词向量。例如，上述矩阵中的“lazy”的词向量是[2,1]等等。这里的行对应于 语料库和列中的文档对应于

为什么呢？ 因为在现实世界的应用中，我们可能会有一个包含数百万个文档的语料库。 并且在数百万的文档中，我们可以提取数亿个独特的词。 所以基本上，上面准备的矩阵将是非常稀疏的，对于任何计算都是低效的。 所以使用每一个独特的单词作为字典元素的替代方法是根据频率来选择上一个万字，然后准备一个字典。

2.每个单词的计数方式。

我们可以采取频率（单词在文档中出现的次数）或存在（将文档中出现的单词？）作为计数矩阵M中的条目。但通常，频率方法优于后者。

下面是矩阵M的表示图像，以便于理解。



2.1.2TF-IDF 向量化

这是基于频率方法的另一种方法，但它与计数向量化不同，在于它不仅考虑单个文档中的单词而是在整个语料库中的出现。那么这背后的理由是什么呢？让我们试着去了解一下。

一个对文档中像‘is’, ‘the’, ‘a’等常用单词往往要比很重要的单词频繁。例如，与其他文件相比，Lionel Messi的档件A将会包含更多的“Messi”一词。但是，几乎每个文档中，像“the”等这样的常用词也将以更高的频率存在。

理想情况下，我们想要的是减少几乎所有文件中都会出现的常见单词，并更加重视出现在文档子集中的其他单词。

TF-IDF通过对这些常用词进行降低权重来忽略他们同时重视特定文档中的像Messi等这样的词。

关闭

目录 分类 标签 项目 友链 关于 搜索

Term	Count
This	1
is	1
about	2
Messi	4

Term	Count
This	1
is	2
about	1
Tf-idf	1

现在我们来定义一些与TF-IDF相关的术语。

TF = (词项t在文档中出现的次数) / (在文档中的总词项数)

所以，TF (This，Document1) = 1/8

TF (This，Document2) = 1/5

它表示文字对文件的贡献，即与文件相关的词应该是频繁的。 例如：关于Messi的文件应该包含大量的“Messi”这个词。

IDF = log (N / n) ，其中，N是总文档数量，n是出现词项t的文档数量。

菜单 其中N是文档数量，n是术语t出现的文档数量。(原文档是不是排版错误)

让我们计算“Messi”一词的IDF。

IDF(Messi) = log(2/1) = 0.301.

现在，我们将TF-IDF与一个通用单词“This”和似乎与文献1相关的“Messi”进行比较。

TF-IDF(This,Document1) = (1/8) * (0) = 0

TF-IDF(This, Document2) = (1/5) * (0) = 0

TF-IDF(Messi, Document1) = (4/8)*0.301 = 0.15

因为，您可以看到Document1，TF-IDF方法严重忽略“This”这个词，但是赋予“Messi”更大的权重。 所以，这可以被理解为“Messi”是整个语料库上下文在Document1中的重要词。

2.1.3具有固定上下文边界（Context Window）的同现矩阵

大的思想（The Big Idea） - 类似的词汇往往会发生在一起，并将具有类似的上下文，例如 - 苹果是一个水果。 芒果是水果。

一起出现的次数。

上下文边界 - 上下文边界由数字和方向指定。 那么2（周围）的上下文边界是什么意思？ 下面我们来看一个例子，

绿色单词是“Fox”一词的2（周围）上下文边界，并且为了计算共现，只会计算这些单词。 让我们看看“Over”这个词的上下文边界。

现在我们来举个例子来计算一个同现矩阵。

语料库 = He is not lazy. He is intelligent. He is smart.

关闭

目录 分类 标签 项目 友链 关于 搜索

intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

让我们通过看到上表中的两个例子来理解这个同现矩阵。 红色和蓝色框。

红色框 - 在上下文边界2中出现了“He”和“is”的次数，可以看出，这个数字是4。下表将帮助您显示计数。

He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart
He	is	not	lazy	He	is	intelligent	He	is	smart

而“Lazy”一词在上下文边界中从未出现过“intelligent”，因此在蓝盒中已被赋值为0。

同现矩阵的变化

菜单

效词等的不相关词来获得。这仍然是非常大的并且存在计算困难。
但是，请记住，这种同现矩阵通常不用于词向量表示。相反，该同现矩阵使用诸如PCA，SVD等技术被分解成因子，并且这些因子的组合形成了词向量的表示。

让我更清楚地说明这一点。例如，您在上述VXV大小的矩阵上执行PCA。您将获得V个主要组件。您可以从这些V个组件中选择k个组件。所以，新的矩阵将是V X k的形式。

而且，一个单词，将被表示为k维而不是V 维，同时仍然能捕捉几乎相同的语义信息。 k通常是数百的数量级。

那么PCA在后面要做的是将同现矩阵分解为三个矩阵U，S和V，其中U和V都是正交矩阵。重要的是U和S的点积给出了词向量的表示，V给出了单词上下文的表示。

$$\underset{m \times n}{\hat{X}} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix} \approx \underset{m \times r}{U} \begin{pmatrix} u_{11} & \dots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix} \underset{r \times r}{S} \begin{pmatrix} s_{11} & 0 & \dots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix} \underset{r \times n}{V^{\mathsf{T}}} \begin{pmatrix} v_{11} & \dots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}$$

4.它只需被计算一次，并且之后就可以随时使用。 在这个意义上，它比其他的快。

同现矩阵的缺点

1.它需要巨大的内存来存储同现矩阵。
但是，例如在Hadoop集群将矩阵从系统中分解出来等可以避免这个问题，并且可以被保存。

2.2基于预测的词嵌入

先决条件：本部分假设您了解神经网络工作原理的知识以及神经网络中权重更新的机制。如果您是神经网络的新人，我建议您通过Sunil的这篇令人敬畏的文章，了解神经网络的工作原理。

到目前为止，我们已经看到确定性的方法来确定词向量。但是，这些方法已经被证明在他们的词表示中会受到限制，直到Mitolov等人将word2vec引入NLP社区。这些方法是基于某些意义上的预测，如它们为单词提供概率，这些方法并被证明是处理像词类比和词相似性等任务的最新技术。他们也能够

两种方法，并获得深入理解他们的工作。

2.2.1 CBOW（连续的词袋）

CBOW的工作方式是倾向于在给定上下文情况下预测单词的概率。上下文可以是单个单词或一组单词。但为了简单起见，我将采用单个的上下文单词，并尝试预测单个目标词。

假设我们有一个语料库C =“Hey, this is sample corpus using only one context word”，并且我们定义了一个上下文边界1.该语料库可以转换为如下CBOW模型的训练集合。输入如下所示。下图中右侧的矩阵包含从左侧输入的独热编码。

Input	Output			Hey	This	is	sample	corpus	using	only	one	context	word
Hey	this		Datapoint 1	1	0	0	0	0	0	0	0	0	0
this	hey		Datapoint 2	0	1	0	0	0	0	0	0	0	0
is	this		Datapoint 3	0	0	1	0	0	0	0	0	0	0
is	sample		Datapoint 4	0	0	1	0	0	0	0	0	0	0
sample	is		Datapoint 5	0	0	0	1	0	0	0	0	0	0
sample	corpus		Datapoint 6	0	0	0	1	0	0	0	0	0	0
corpus	sample		Datapoint 7	0	0	0	0	1	0	0	0	0	0
corpus	using		Datapoint 8	0	0	0	0	1	0	0	0	0	0
using	corpus		Datapoint 9	0	0	0	0	0	1	0	0	0	0
using	only		Datapoint 10	0	0	0	0	0	1	0	0	0	0
only	using		Datapoint 11	0	0	0	0	0	0	1	0	0	0
only	one		Datapoint 12	0	0	0	0	0	0	1	0	0	0
one	only		Datapoint 13	0	0	0	0	0	0	0	1	0	0
one	context		Datapoint 14	0	0	0	0	0	0	0	1	0	0
context	one		Datapoint 15	0	0	0	0	0	0	0	0	1	0
context	word		Datapoint 16	0	0	0	0	0	0	0	0	1	0
word	context		Datapoint 17	0	0	0	0	0	0	0	0	0	1

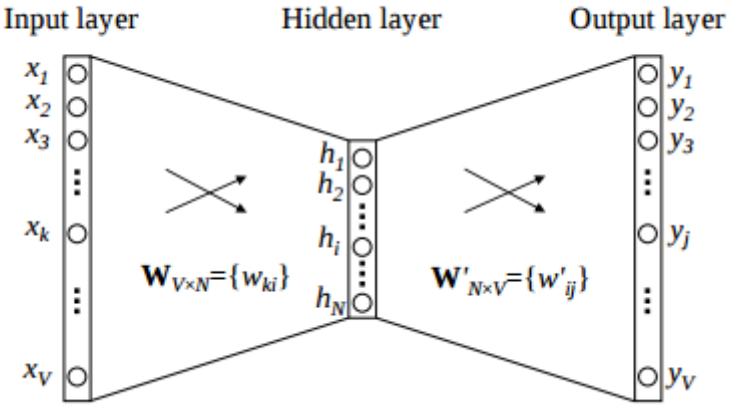
使用单个数据点的目标表示数据点4如下所示

关闭

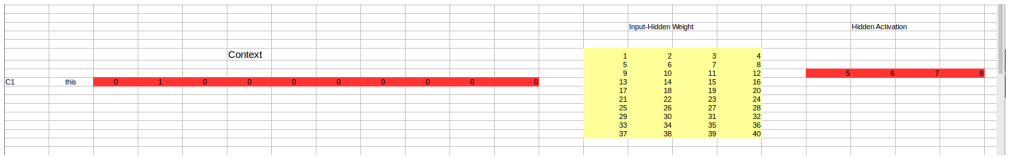
目录 分类 标签 项目 友链 关于 搜索

现在让我们看看正向传播如何用于计算隐藏层激活。

我们首先看看CBOW模型的图解表示。



用单个数据点表示上述图像的矩阵如下。

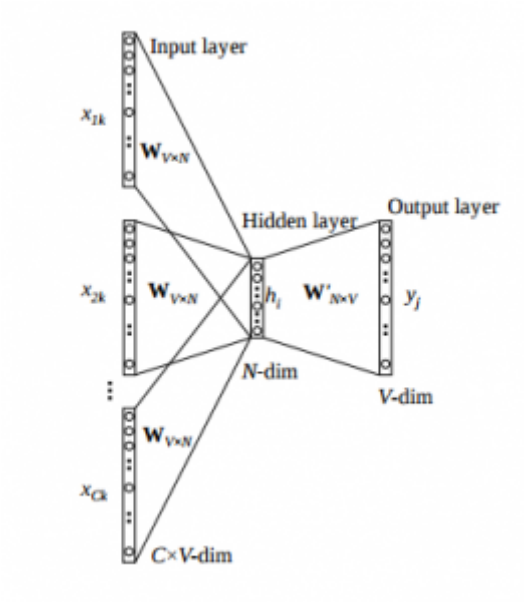


菜单

外，N是隐藏层中的神经元数量。这里，N = 4。

3. 任何层之间都没有激活功能（我的意思是它不是线性激活）
4. 输入被乘以输入-隐藏层权重，并称为隐藏激活。它只是复制输入-隐藏层矩阵中的相应行。
5. 隐藏层输入乘以隐藏层输出的权重并计算输出。
6. 输出和目标之间的误差被计算出来并传播回来重新调整权重。
7. 隐藏层和输出层之间的权重取为单词的词向量表示。

我们看到上述步骤是为单个上下文单词的情况。现在，如果我们有多个上下文单词呢？下面的图片描述了多个上下文单词的架构。



输入层平会三个读热编码向量，如上所示为红色，蓝色和绿色。

因此，输入层将在输入中具有3个 [1 X V]矢量，如上所示，输出层中有1个[1 XV]。架构的其余部分与单上下文单词的CBOW相同。

步骤保持不变，只有隐藏激活的计算更发生了变化。不是将输入-隐藏的权重矩阵的相应行复制到隐藏层，而是取矩阵的所有相应行的平均值。我们可以用上图来理解。计算的平均向量成为隐藏激活。因此，如果我们对单个目标词处理需要三个上下文单词，那么我们将有三个初始隐藏激活，然后对元素进行平均得到最终激活。

在单个上下文单词和多个上下文单词中，因为CBOW与简单MLP网络不同，我已经展示了直到隐藏激活的计算的图像。隐藏层计算后的步骤与本文所述MLP的步骤相同 - 从头开始理解和编码神经网络。

下文澄清MLP和CBOW之间的差异：

1. MLP中的目标函数是一个MSE（均方差），而在CBOW中，给定一组上下文（即-log（p（wo / wi））的单词的负对数似然，其中给出了p（wo / wi）如

wi：上下文单词

相对于隐藏-输出的权重，输入-隐藏权重的误差梯度是不同的，因为MLP具有S形激活（通常），但CBOW具有线性激活。然而，计算梯度的方法与MLP相同。

CBOW的优势：

1. 概率是自然的，它应该优于确定性方法（一般）。
2. 耗费内存低。 它不需要像共生矩阵那样需要存储三个巨大的矩阵的巨大的RAM要求。

CBOW的缺点：

1. CBOW取一个单词的上下文的平均值（如上面在隐藏-激活的计算中所见）。例如，苹果可以是一个水果和一个公司，但是CBOW只能在一个水果的集群和公司的集群之间进行平均的上下文的分析。
2. 如果不适当优化，要从头开始训练CBOW。

2.2.2 Skip - Gram模型

Skip - gram遵循与CBOW相同的拓扑。它只是颠覆了CBOW的架构。skip-gram的目的是预测给定一个单词的上下文。让我们对同样的语料库C=”Hey,

sample	is	corpus
corpus	sample	corpus
using	corpus	only
only	using	one
one	only	context
context	one	word
word	context	<padding>

Skip - Gram的输入向量将与单上下文单词的CBOW模型相似。此外，隐藏激活的计算将是相同的。差异是在目标变量中。由于我们已经在两边定义了一个单个上下文单词的边界，所以在图像的蓝色部分中可以看到“ **“2”个独热编码的目标变量**和“ **“2”个对应的输出**。

对于两个目标变量计算出两个单独的误差，并且获得的两个误差向量也被逐个地添加以获得最终误差向量，这里使用了向后传播。

输入和隐藏层之间的权重作为训练后的词向量的表示。损失函数或目标函数与CBOW模型类型相同。

Skip-Gram架构如下所示。

让我们分解上面的图像。

输入层大小 - [1 X V]，输入-隐藏权重矩阵大小 - [VXN]，隐层中的神经元数量N，隐藏 - 输出权重矩阵大小 - [N X V]，输出层大小 - C [1 XV]

在上述例子中，C是上下文单词的数量= 2，V = 10，N = 4

1. 红色的行是对应于输入独热编码向量的隐藏激活。它基本上是对应的输入隐藏矩阵行。
2. 黄色矩阵是隐层和输出层之间的权重。
3. 蓝色矩阵通过隐藏激活和隐藏输出权重的矩阵乘法获得。将为两个目标单词（上下文）计算两行。
4. 将蓝色矩阵的每一行分别转换成其softmax概率，如绿色框所示。
5. 灰色矩阵包含两个上下文单词（目标）的独热编码向量。
6. 通过从绿色矩阵（输出）的第一行逐元素的减去灰色矩阵（目标）的第一行元素来计算错误。下一行重复这一步。因此，对于**n个** 目标语境单词，我们将有**n个**错误向量。
7. 对所有误差向量进行元素和求和以获得最终误差向量。
8. 该错误向量被传播回来以更新权重。

Skip-Gram模型的优点

1. Skip-gram模型可以捕获单个单词的两个语义。即它将苹果用两个向量表示。一个为公司和一个为水果。

关闭

目录 分类 标签 项目 友链 关于 搜索

由于词嵌入或词向量化是单词之间的上下文相似性的数值表示，因此可以对其进行操纵并执行令人惊奇的任务，如 -

1. 找出两个词之间的相似度。
   model.similarity('woman','man')
   0.73723527
2. 找出奇怪的一个。
   model.doesnt_match('breakfast cereal dinner lunch';.split())
   'cereal'
3. 惊人的东西，像womam+king=queen
   model.most_similar(positive=['woman','king'],negative=['man'],topn=1)
   queen: 0.508
4. 在模型下计算文本的概率
   model.score(['The fox jumped over the lazy dog'.split()])
   0.21

以下是word2vec的一个有趣的可视化。

菜单

上述图像是t-SNE的2维词向量表示，您可以看到苹果的两个上下文已被捕获。一个是水果，另一个是公司。

5.可用于执行机器翻译。

关闭

目录 分类 标签 项目 友链 关于 搜索

约1000亿字节训练的300万个词汇的词汇。这个模型的downlaod链接是这样的。当心这是一个1.5 GB的下载。

```
from gensim.models import Word2Vec
#loading the downloaded model
model = Word2Vec.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True, norm_only=True)
#the model is loaded. It can be used to perform all of the tasks mentioned above.
# getting word vectors of a word
dog = model['dog']
#performing king queen magic
print(model.most_similar(positive=['woman', 'king'], negative=['man']))
#picking odd one out
print(model.doesnt_match("breakfast cereal dinner lunch".split()))
#printing similarity index
print(model.similarity('woman', 'man'))
```

5.训练你自己的词向量

我们将在自定义语料库上训练我们自己的word2vec。对于训练模型，我们将使用gensim，步骤如下图所示。

菜单

关闭

目录 分类 标签 项目 友链 关于 搜索

```
#training word2vec on 3 sentences
model = gensim.models.Word2Vec(sentence,
min_count=1,size=300,workers=4)
```

让我们尝试了解这个模型的参数。

sentence - 我们的语料库列表

min_count = 1 - 词的阈值。频率大于此值的词将被包含在模型中。

size = 300 - 我们希望它代表我们的单词的维度数。这是词向量的大小。

worker = 4 - 用于并行化

```
#using the model
#The new trained model can be used similar to the pre-trained ones.
#printing similarity index
print(model.similarity('woman', 'man'))
```

6.总结

词嵌入是一个活跃的试图找出比现有的更好的词表示的研究领域。但随着时间的推移，数量越来越多，复杂程度越来越大。本文旨在简化这些嵌入模型

菜单

关闭

目录 分类 标签 项目 友链 关于 搜索

菜单