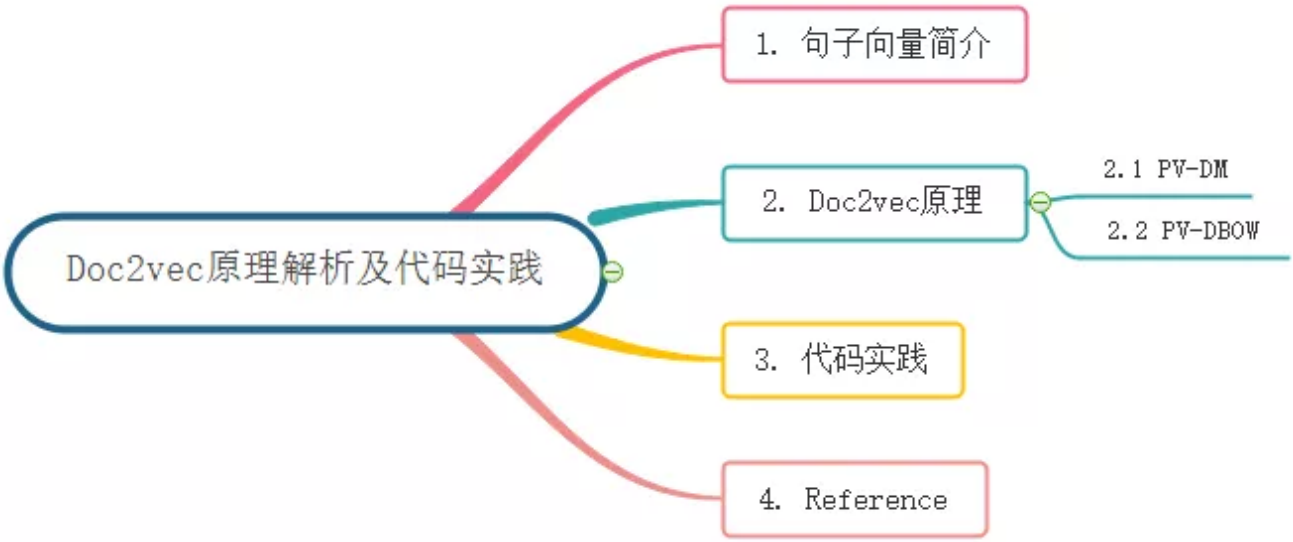
Doc2vec原理解析及代码实践

原创 Microstrong Microstrong 2020-04-24

收录于话题
#自然语言处理模型汇总

16个

本文概览：



1. 句子向量简介

Word2Vec提供了高质量的词向量，并在一些任务中表现良好。虽然Word2Vec提供了高质量的词汇向量，但是仍然没有有效的方法将它们结合成一个高质量的文档向量。对于一个句子、文档或者说一个段落，怎么把这些数据投影到向量空间中，并具有丰富的语义表达呢？过去人们常常使用以下几种方法：

- Bag of Words
- LDA
- Average Word Vectors
- TF-IDF Weighted Word Vectors

就Bag of Words而言，有如下缺点：

- 没有考虑到单词的顺序;
- 忽略了单词的语义信息;

因此这种方法对于短文本效果很差，对于长文本效果一般，通常在科研中用来做Baseline。

Average Word Vectors就是简单的对句子中的所有词向量取平均。是一种简单有效的方法，但缺点也是没有考虑到单词的顺序。

TF-IDF Weighted word vectors是指对句子中的所有词向量根据TF-IDF权重加权求和，是常用的一种计算sentence embedding的方法，在某些问题上表现很好，相比于简单的对所有词向量求平均，考虑到了TF-IDF权重，因此句子中更重要的词占得比重就更大。但缺点也是没有考虑到单词的顺序。

LDA模型就是计算出一篇文档或者句子的主题分布，也常常用于文本分类任务。

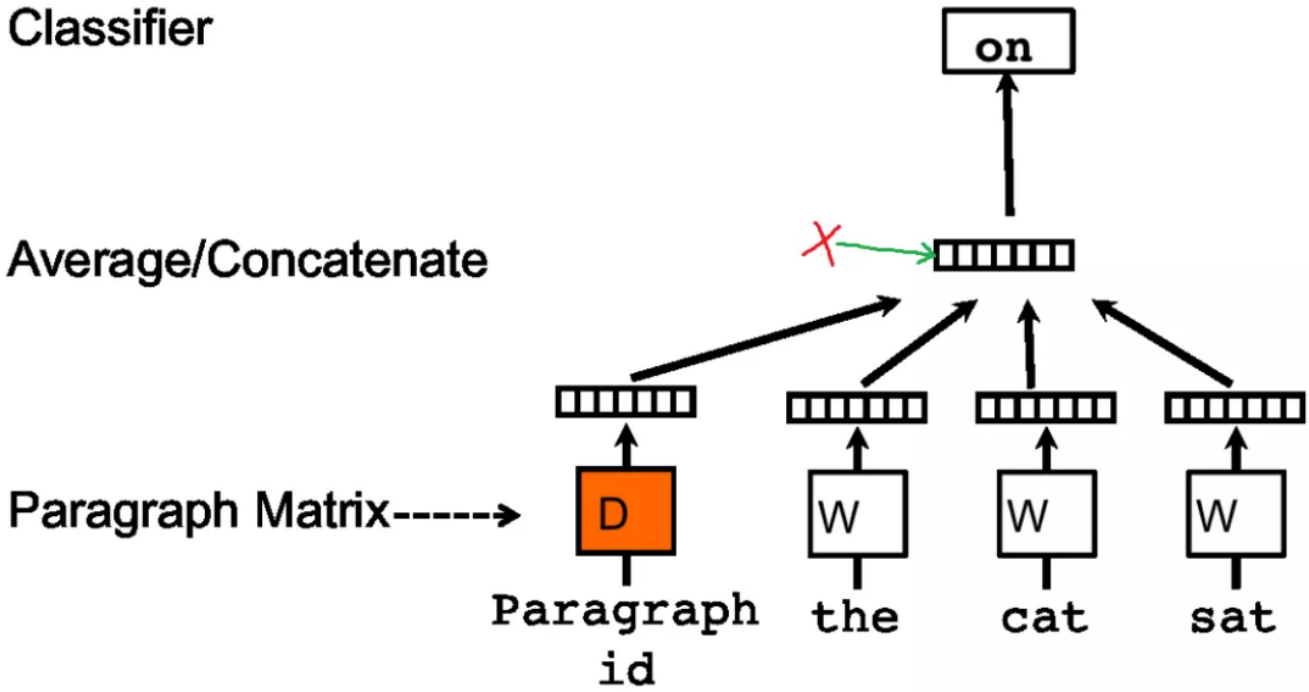
2. Doc2vec原理

Doc2vec方法是一种无监督算法，能从变长的文本（例如：句子、段落或文档）中学习得到固定长度的特征表示。Doc2vec也可以叫做 Paragraph Vector、Sentence Embeddings，它可以获得句子、段落和文档的向量表达，是Word2Vec的拓展，其具有一些优点，比如不用固定句子长度，接受不同长度的句子做训练样本。Doc2vec算法用于预测一个向量来表示不同的文档，该模型的结构潜在的克服了词袋模型的缺点。

Doc2vec模型是受到了Word2Vec模型的启发。Word2Vec预测词向量时，预测出来的词是含有词义的，Doc2vec中也是构建了相同的结构，所以Doc2vec克服了词袋模型中没有语义的缺点。假设现在存在训练样本，每个句子是训练样本，和Word2Vec一样，Doc2vec也有两种训练方式，一种是分布记忆的段落向量（Distributed Memory Model of Paragraph Vectors , PV-DM）类似于Word2Vec中的CBOW模型，另一种是分布词袋版本的段落向量（Distributed Bag of Words version of Paragraph Vector，PV-DBOW）类似于Word2Vec中的Skip-gram模型。

2.1 Distributed Memory Model of Paragraph Vectors

训练句向量的方法和词向量的方法非常类似。训练词向量的核心思想就是可以根据每个单词w_i的上下文预测w_i，也就是说上下文的单词对w_i是有影响的。那么同理，可以用同样的方法训练Doc2vec。例如对于一个句子 `s: i want to drink water` ，如果要去预测句子中的单词 `want` ，那么不仅可以根据其它单词生成feature，也可以根据其它单词和句子 `s` 来生成feature进行预测。因此Doc2vec的框架如下图所示：



在Doc2vec中，每一句话用唯一的向量来表示，用矩阵D的某一列来代表。每一个词也用唯一的向量来表示，用矩阵W的某一列来表示。每次从一句话中滑动采样固定长度的词，取其中一个词作预测词，其他的作为输入词。输入词对应的词向量Word Vector和本句话对应的句子向量Paragraph vector作为输入层的输入，将本句话的向量和本次采样的词向量相加求平均或者累加构成一个新的向量X，进而使用这个向量X预测此次窗口内的预测词。

Doc2vec相对于Word2vec不同之处在于，在输入层增添了一个新的句子向量Paragraph vector，Paragraph vector可以被看作是另一个词向量，它扮演了一个记忆角色。Average Word Vectors中，使用Word2Vec训练词向量，因为每次训练只会截取句子中一小部分词训练，而忽略了除了本次训练词以外该句子中的其他词，这样仅仅训练出来每个词的向量表达，句子只是每个词的向量累加在一起取平均的一种表达。正如上面所说的Average Word Vectors的缺点，忽略了文本的词序问题。而Doc2vec中的Paragraph vector则弥补了这方面的不足，它每次训练也是滑动截取句子中一小部分词来训练，Paragraph Vector在同一个句子的若干次训练中是共享的，所以同一句话会有多次训练，每次训练中输入都包含Paragraph vector。它可以被看作是句子的主旨，有了它，该句子的主旨每次都会被作为输入的一部分来训练。这样每次训练过程中，不光是训练了词，得到了词向量。同时随着一句话每次滑动取若干词训练的过程中，作为每次训练的输入层一部分的共享Paragraph vector，该向量表达的主旨会越来越准确。Doc2vec中PV-DM模型具体的训练过程和Word2Vec中的CBOW模型训练方式相同，这里就不在重复讲解了，不明白Word2Vec原理可以看我的这篇文章：深入理解Word2Vec原理解析。

这个段落向量或句向量也可以认为是一个单词，它的作用相当于是上下文的记忆单元或者是这个段落的主题，所以我们一般叫这种训练方法为Distributed Memory Model of Paragraph Vectors(PV-DM)。在训练的时候我们固定上下文的长度，用滑动窗口的方法产生训练集。段落向量或句向量在该上下文中共享。

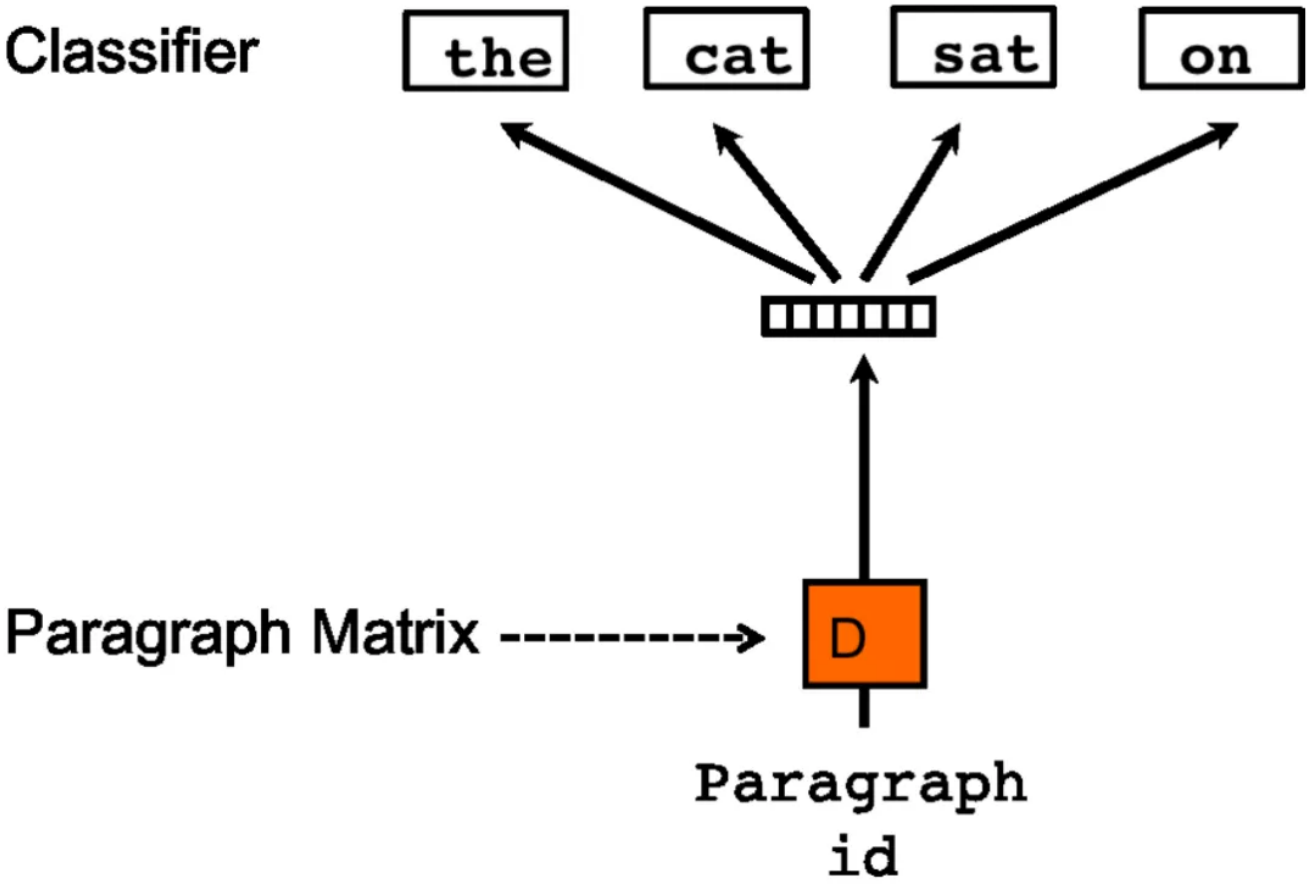
训练完了以后，就会得到训练样本中所有的词向量和每句话对应的句子向量，那么Doc2vec是怎么预测新的句子Paragraph vector呢？其实在预测新的句子的时候，还是会将该Paragraph vector随机初始化，放入模型中再重新根据随机梯度下降不断迭代求得最终稳定下来的句子向量。不过在预测过程中，模型里的词向量、投影层到输出层的softmax weights参数是不会变的，这样在不断迭代中只会更新Paragraph vector，其它参数均已固定，只需很少的时间就能计算出待预测的Paragraph vector。

总结PV-DM的过程, 主要有两步：

- 训练模型，在已知的训练数据中得到词向量W, softmax的参数U和b，以及段落向量或句向量D。
- 推断过程（inference stage），对于新的段落，得到其向量表达。具体地，在矩阵D中添加更多的列，在固定W,U,b的情况下，利用上述方法进行训练，使用梯度下降的方法得到新的D,从而得到新段落的向量表达。

2.2 Distributed Bag of Words version of Paragraph Vector

另外一种训练方法是忽略输入的上下文，让模型去预测段落中的随机一个单词。就是在每次迭代的时候，从文本中采样得到一个窗口，再从这个窗口中随机采样一个单词作为预测任务，让模型去预测，输入就是段落向量。我们称这种模型为 Distributed Bag of Words version of Paragraph Vector(PV-DBOW)，如下图所示：



在上述两种方法中，我们可以使用PV-DM或者PV-DBOW得到段落向量或者句向量。对于大多数任务，PV-DM的方法表现很好，但论文中的作者也强烈推荐两种方法相结合。

3. 代码实践

现在，我们对NLP中Doc2vec算法进行简单地实现。这个案例是使用 `gensim 3.4` 和 `python3` 实现Doc2vec模型的训练和预测，gensim库使Doc2vec的实现更加容易。本文的所有代码已经放在我的GitHub中，地址：https://github.com/Microstrong0305/WeChat-zhihu-csdnblog-code/blob/master/NLP/Doc2vec/Doc2vec.py

```
# -------------Let’s start implementing----------------------
# Import all the dependencies
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
import nltk

nltk.download()

# -------------Let’s prepare data for training our doc2vec model----------------
data = ["I love machine learning. Its awesome.",
        "I love coding in python",
        "I love building chatbots",
        "they chat amagingly well"]

tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()), tags=[str(i)]) fo

# -------------Lets start training our model----------------------
max_epochs = 100
vec_size = 20
alpha = 0.025

model = Doc2Vec(size=vec_size,
                alpha=alpha,
                min_alpha=0.00025,
                min_count=1,
                dm=1)

'''
```

```
Note: dm defines the training algorithm. If dm=1 means‘distributed memory’(PV-DM
Distributed Memory model preserves the word order in a document whereas Distribu
which doesn’t preserve any word order.
'''

model.build_vocab(tagged_data)

for epoch in range(max_epochs):
    print('iteration {0}'.format(epoch))
    model.train(tagged_data,
                total_examples=model.corpus_count,
                epochs=model.iter)
    # decrease the learning rate
    model.alpha -= 0.0002
    # fix the learning rate, no decay
    model.min_alpha = model.alpha

model.save("d2v.model")
print("Model Saved")

# -------------Lets play with it----------------------
from gensim.models.doc2vec import Doc2Vec

model = Doc2Vec.load("d2v.model")
# to find the vector of a document which is not in training data
test_data = word_tokenize("I love chatbots".lower())
v1 = model.infer_vector(test_data)
print("V1_infer", v1)

# to find most similar doc using tags
similar_doc = model.docvecs.most_similar('1')
print(similar_doc)

# to find vector of doc in training data using tags or in other words, printing
print(model.docvecs['1'])
```

实践总结：

- 使用gensim的Doc2vec进行句子、段落或文章的向量表示时，不需要进行分词；

- 使用TaggedDocument进行语料预处理；
- train训练模型，save 和 load 加载训练好的模型；
- docvecs.most_similar( )计算相似度；

4. Reference

【1】Le Q, Mikolov T. Distributed representations of sentences and documents[C]// International conference on machine learning. 2014: 1188-1196.
【2】基于Doc2vec训练句子向量 - 灰灰的文章 - 知乎 https://zhuanlan.zhihu.com/p/36886191
【3】doc2vec原理及实践，地址：https://blog.csdn.net/John_xyz/article/details/79208564
【4】https://medium.com/@mishra.thedeepak/doc2vec-simple-implementation-example-df2afbbfbad5

收录于话题 #自然语言处理模型汇总·16个

上一篇
基于知识图谱和图卷积神经网络的应用和开发

下一篇
深入浅出Word2Vec原理解析