论文｜Doc2vec的算法原理、代码实现及应用启发

原创 Thinkgamer 搜索与推荐Wiki 2020-12-15

收录于话题
#论文笔记

18个

点击标题下「*搜索与推荐Wiki*」可快速关注

▼ 相关推荐 ▼

1、美团点评｜深度学习在推荐中的实践
2、近100页的《常见的五种神经网络》汇总电子书
3、以DSSM为例说明深度学习模型训练中的若干问题
4、聊一聊海量公众号下我是如何进行筛选和内容消费的

万物皆可Embedding系列会结合论文和实践经验进行介绍，前期主要集中在论文中，后期会加入实践经验和案例，目前已更新：

- **万物皆可Vector之语言模型：从N-Gram到NNLM、RNNLM**
- **万物皆可Vector之Word2vec：2个模型、2个优化及实战使用**
- **Item2vec中值得细细品味的8个经典tricks和thinks**
- Doc2vec的算法原理、代码实现及应用启发

后续会持续更新Embedding相关的文章，欢迎持续关注「搜索与推荐Wiki」

Doc2vec是Mikolov2014年提出的论文，也被成为Paragraph Vector，下面的内容分为三方面进行介绍，分别为：

- Doc2vec的原理
- Doc2vec在推荐系统中的应用启发
- Doc2vec的算法实现

1、Doc2vec的算法原理

如何学习得到Word的Vector表示

一个非常流行的学习Word Vector的方法如下图所示：

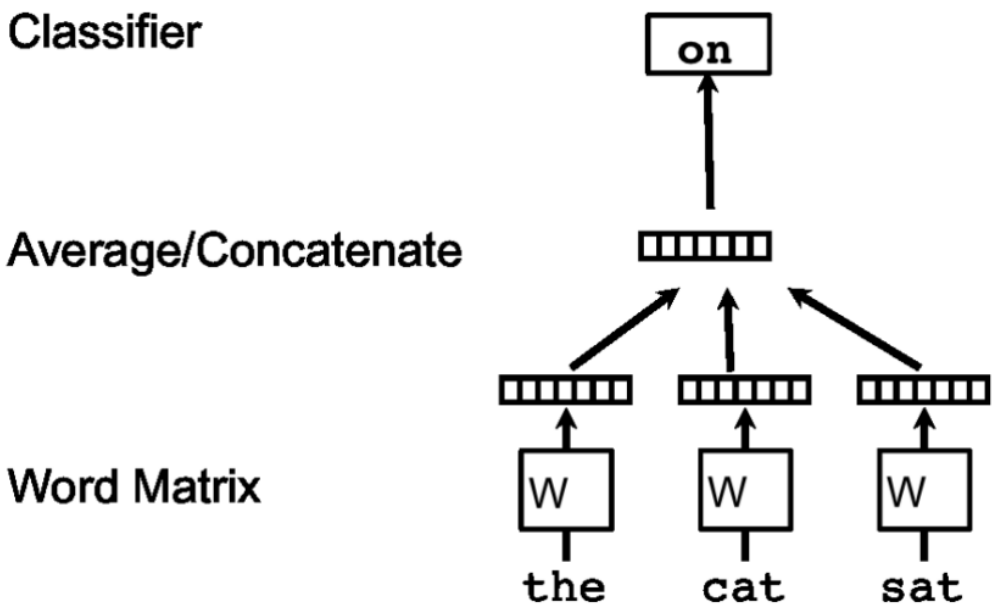

Figure 1. A framework for learning word vectors. Context of three words ("the," "cat," and "sat") is used to predict the fourth word ("on"). The input words are mapped to columns of the matrix W to predict the output word.

一个非常流行的学习Word Vector的方法

在上图中，每个Word都被映射成一个唯一的vector编码，其组合成了矩阵W，其中每列表示的就是一个Word，对于给定的序列单词$w_1, w_2, \ldots, w_T$，得到word vector的目标函数是最大化平均概率：

$$\frac{1}{T}\sum_{t=k}^{T-k} log\, p(w_t|w_{t-k}, \ldots, w_{t+k})$$

预测任务通常是通过多分类（比如：$softmax$）完成的，因此可得：

$$p(w_t|w_{t-k}, \ldots, w_{t+k}) = \frac{e^{y_{wt}}}{\sum_i e^{y_i}}$$

其中y_i表示第i个输出的word未归一化的log概率值，为：

$$y = b + Uh(w_{t-k}, \ldots, w_{t+k}; W)$$

其中U, b表示的是$softmax$参数，h表示的是矩阵W中word vector的连接方式（平均或者连接）。

论文中提出，求解word、paragraph vector使用的是hierarical softmax，和word2vec中的优化方法一致，这样可以很大程度上在丢失少量精确度的情况下，加快模型的训练速度

Doc2vec的两种算法

Doc2vec其实包含了两种算法：

- PV-DM（Distributed Memory Model of Paragraph Vector）
- PV-DBOW（Distributed Bag of Words version of Paragraph Vector）

PV-DM 」

PV-DM类似于Word2vec中的CBOW模型，其结构图如下所示：

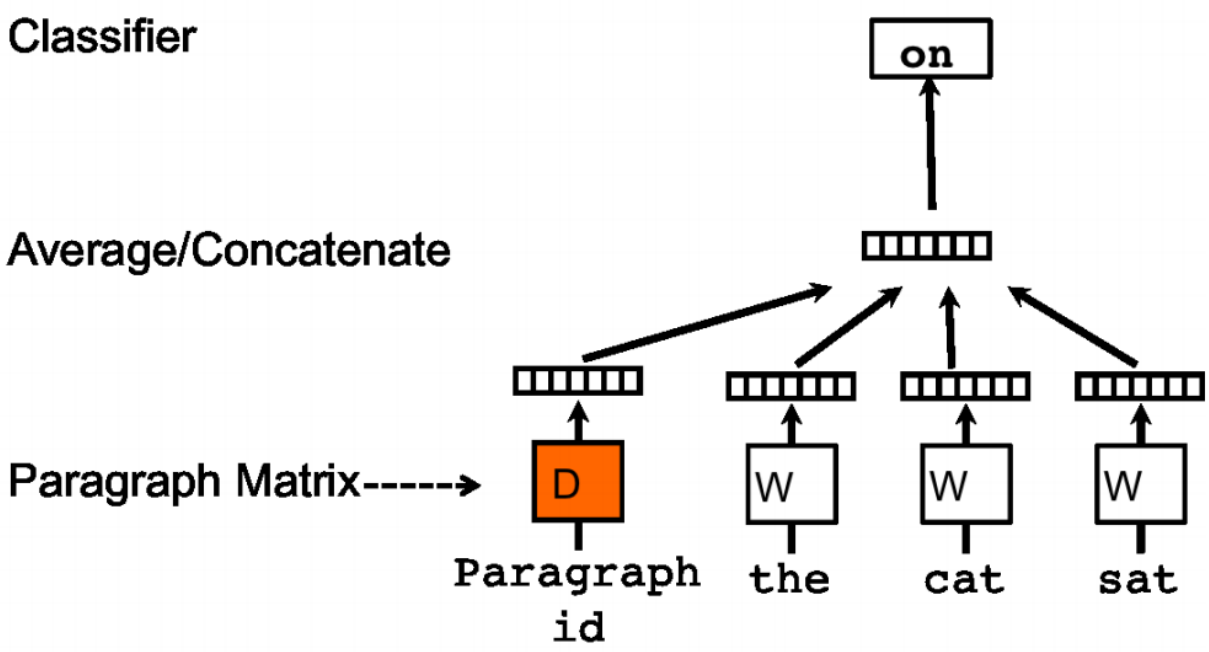

Figure 2. A framework for learning paragraph vector. This framework is similar to the framework presented in Figure 1; the only change is the additional paragraph token that is mapped to a vector via matrix D. In this model, the concatenation or average of this vector with a context of three words is used to predict the fourth word. The paragraph vector represents the missing information from the current context and can act as a memory of the topic of the paragraph.

PV-DM

和上面的图中区别是，增加了段落的vector，即 函数 h 利用了矩阵W 和D中的向量，矩阵D 表示的段落向量矩阵，W 表示的是word的向量矩阵。

Paragraph vector在这里扮演的是一个记忆的角色，因此为在词袋模型中，每次训练只会截取段落的一小部分进行训练，而忽略本次训练之外的单词，这样仅仅训练出来每个词的向量表达，段落只是每个词的向量累加在一起表达的，这里使用Paragraph vector可以在一定程度上弥补词袋模型的缺陷。

PV-DM模型的输入是固定长度的，其从段落的上下文的滑动窗口中进行采样，这一点和Word2vec是一致的，在基于同一段落构造好的样本中， 段落的向量是共享的，但是在基于不同段落构造好的样本中，段落的向量是不共享的。在所有的样本中，word的向量是一致的（共享的）。

在整个训练过程中，paragraph vector能够记忆整个句子的意义，word vector则能够基于全局部分学习到其具体的含义。

PV-DBOW」

PV-DBOW类似于Word2vec中的Skip-gram模型，其结构图如下所示：

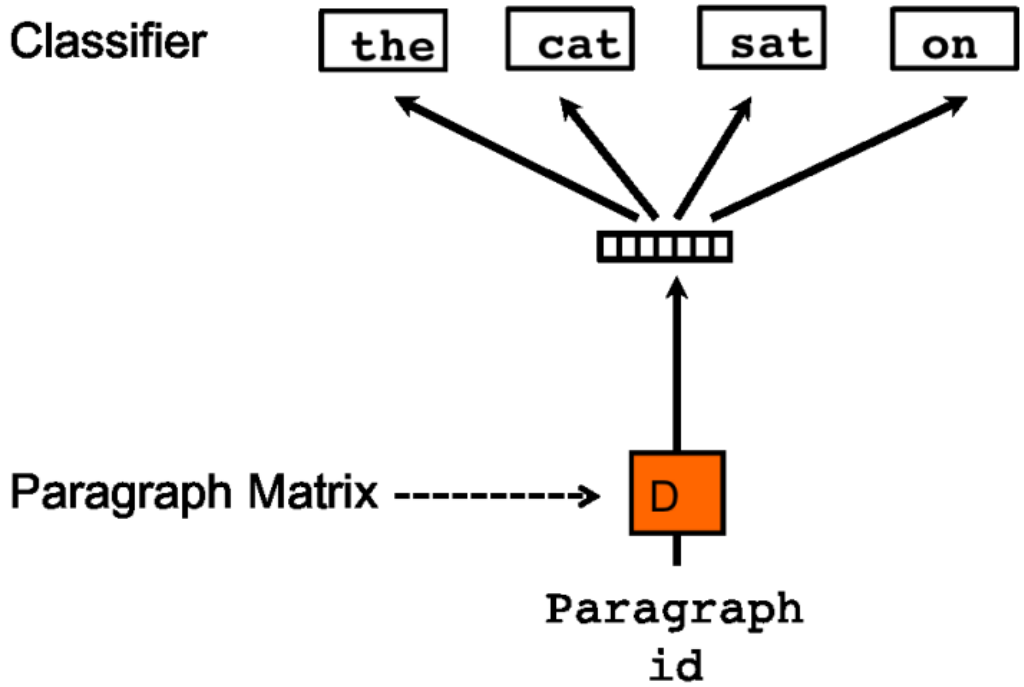

Figure 3. Distributed Bag of Words version of paragraph vectors. In this version, the paragraph vector is trained to predict the words in a small window.

PV-DBOW

和PV-DM不同的是，这里使用的是段落的向量来预测单词。

如何预测新句子的vector」

模型完成训练之后，可以得到段落的向量、word的向量和相关的参数，对于需要预测段落，会将paragram vecotr进行随机的初始化，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段落向量。

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，其他参数均已固定，只需很少的时间就能计算出带预测的Paragraph vector。

实验中注意的点

- word vector之间的聚合使用的是连接
- window size 设置值为8
- vector size 设置的是400
- 论文中进行实验使用的是PV-DM和PV-DBOW向量连接

2、在推荐系统中的应用启发

结合不同模型的vector

之前在别的文章中也看到过说 使用不同模型产出的vector，比如针对word2vec中CBOW和Skip-gram模型，产出两套vector，在使用时，可以进行求平均或者进行连接。

同样在这篇论文中，作者也给出了拼接方式的效果验证，效果是要优于单独使用一种的。

Doc2vec迁移到 User2vec

将NLP相关的知识应用在推荐系统中本身已经司空见惯了，但是我们可以转变一种思路，将这种思想迁移到user-item上，比如针对用户的点击序列，可以理解为段落里边的一个个word，用户本身可以理解为段落，通过这种方式便可以构造出user和

item的向量，因为其是在一个向量空间下的，可以直接通过余弦相似度进行user到items的召回

vector表示用作他用

针对产出的word或者doc vector，可以将其用户分类、聚类其中的特征，这一点可以参考Item2vec内容介绍中的相关内容。

3、Doc2vec的算法实现

这里使用gensim的中的Doc2vec，具体演示代码为：

```
import gensim
import os
from collections import defaultdict, Counter

print("gensim version is: {}".format(gensim.__version__))

# 数据集介绍：使用数据为新闻广播数据，来自于澳大利亚新闻广播选择的314个文档
test_data_dir = os.path.join(gensim.__path__[0], "test", "test_data")
lee_train_file = os.path.join(test_data_dir, "lee_background.cor")
lee_test_file = os.path.join(test_data_dir, "lee.cor")

# 定义函数，读取数据
import smart_open
def read_corpus(fname, tokens_only=False):
    with smart_open.open(fname, encoding="iso-8859-1") as f:
        for i, line in enumerate(f):
            tokens = gensim.utils.simple_preprocess(line)
            if tokens_only:
                yield tokens
            else:
                yield gensim.models.doc2vec.TaggedDocument(tokens,[i])

train_corpus = list(read_corpus(lee_train_file))
# print("train_corpus: \n {}".format(train_corpus[:1]))
test_corpus = list(read_corpus(lee_test_file, tokens_only=True))
# print("test corpus: \n {}".format(test_corpus[:1]))

# 创建模型并进行模型训练
model = gensim.models.doc2vec.Doc2Vec(vector_size = 50, min_count = 2, epochs=40)
```

```
model.build_vocab(train_corpus)
model.train(documents=train_corpus, total_examples=model.corpus_count, epochs=model.epochs)

# 输出每个文档的向量 model.docvecs[id]
doc_vector_dict = defaultdict(list)
for one in train_corpus:
    doc_vector_dict[one.tags[0]] = model.docvecs[one.tags[0]]
# print(doc_vector_dict)

# 计算每个文档最相似的文档 model.docvecs.most_similar
for doc_id, doc_vector in doc_vector_dict.items():
    sim_docs = model.docvecs.most_similar([doc_vector], topn=10)
    # print(sim_docs)

# 推断新文档的向量 model.infer_vector
print(train_corpus[0].words)
infer_vector = model.infer_vector(train_corpus[0].words)
print(infer_vector)

# 根据自相似性进行模型的效果评判，假设文档都是新文档，推断每个文档的向量，并计算其与所有文档的相似度，看本
ranks = list()
for doc_id, doc_vector in doc_vector_dict.items():
    sim_docs = model.docvecs.most_similar([doc_vector], topn=len(model.docvecs))
    sim_docs_rank = [_id for _id, sim in sim_docs].index(doc_id)
    ranks.append(sim_docs_rank)

print(ranks)
counter = Counter(ranks)
print(counter)

# 模型保存
model.save("files/doc2vec.model")
# 模型加载
gensim.models.doc2vec.Doc2Vec.load("files/doc2vec.model")
```