DL-NLP 第四次作业

一、问题描述

从给定的语料库中均匀抽取200个段落（每个段落大于500个词）， 每个段落的标签就是对应段落所属的小说。利用LDA模型对于文本建模，并把每个段落表示为主题分布后进行分类。验证与分析分类结果。

二、背景介绍

2.1 词向量

用词向量来表示词并不是word2vec的首创，在很久之前就出现了。最早的词向量是很冗长的，它使用是词向量维度大小为整个词汇表的大小，对于每个具体的词汇表中的词，将对应的位置置为1，这种词向量的编码方式一般叫做1-of-N representation或者one hot representation。

One hot representation用来表示词向量非常简单，但是却有很多问题。最大的问题是我们的词汇表一般都非常大，比如达到百万级别，这样每个词都用百万维的向量来表示简直是内存的灾难。这样的向量其实除了一个位置是1，其余的位置全部都是0，表达的效率不高。

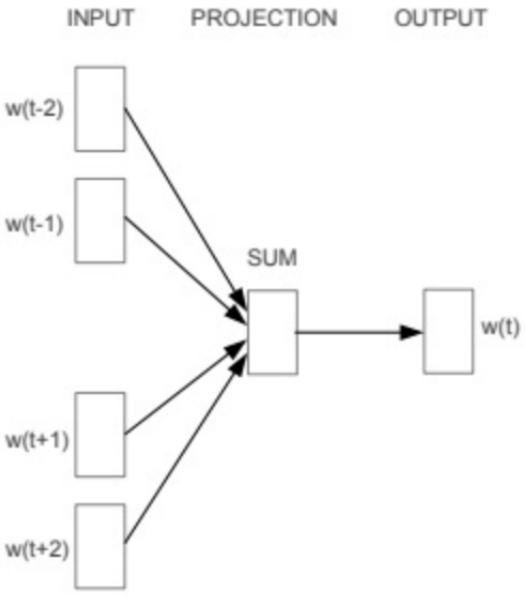
Distributed representation可以解决One hot representation的问题，它的思路是通过训练，将每个词都映射到一个较短的词向量上来。所有的这些词向量就构成了向量空间，进而可以用普通的统计学的方法来研究词与词之间的关系。这个较短的词向量维度是多大呢？这个一般需要我们在训练时自己来指定。

2.2 Word2Vec

word2vec的方法是在2013年的论文《Efficient Estimation of Word Representations inVector Space》中提出的。它是一种高效训练词向量的模型，基本出发点和Distributed representation类似:上下文相似的两个词，它们的词向量也应该相似，比如香蕉和梨在句子中可能经常出现在相同的上下文中，因此这两个词的表示向量应该就比较相似。

2.2.1 CBOW(Continuous Bag-of-Words)

CBOW模型根据某个中心词前后n个连续的词，来计算该中心词出现的概率，即用上下文预测目标词，即已知上下文，估算当前词语的语言模型。对于CBOW，**输入层**是上下文的词语的词向量，**投影层**对其求和，所谓求和，就是简单的向量加法，**输出层**输出最可能的w。由于语料库中词汇量是固定的|C|个，所以上述过程其实可以看做一个多分类问题。给定特征，从|C|个分类中挑一个。模型结构简易示意图如下：



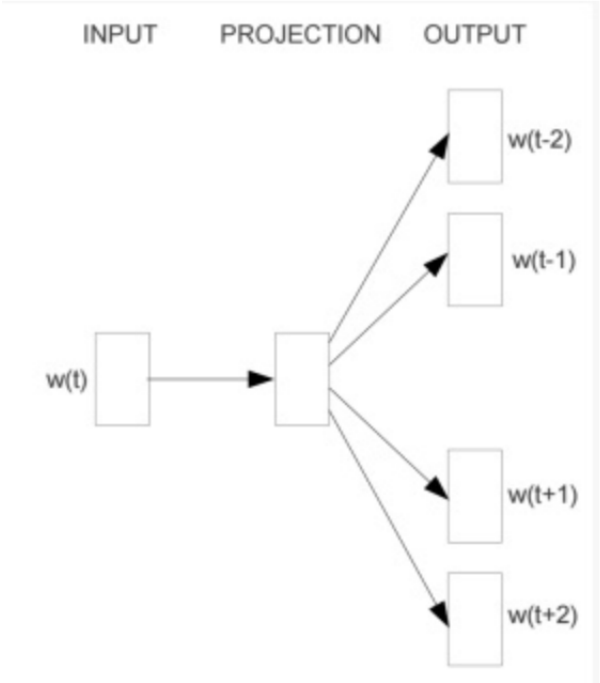
2.2.2 Skip-gram

Skip-gram只是逆转了CBOW的因果关系而已，即已知当前词语，预测上下文。下图与CBOW的两个不同在于:

1. 输入层不再是多个词向量，而是一个词向量
2. 投影层其实什么事情都没干，直接将输入层的词向量传递给输出层

跳字模型的**概念**是在每一次迭代中都取一个词作为中心词汇，尝试去预测它一定范围内的上下文词汇。

所以这个模型定义了一个**概率分布**：给定一个中心词，某个单词在它上下文中出现的概率。我们会选取词汇的向量表示，从而让概率分布值最大化。重要的是，这个模型对于一个词汇，有且只有一个概率分布，这个概率分布就是输出，也就是出现在中心词周围上下词的一个输出。



三、实验过程

3.1 数据介绍

- txt_data 文件夹下存放了：16本金庸武侠小说
- baidu_stopwords：baidu提供的禁用词（stopwords）
- train_data 文件夹下存放了对小说进行分词后的数据
- preprocessor.py：数据读取及预处理程序
- main.py：模型训练及预测分类主程序
- cbow.model：保存的 cbow 模型
- skip_gram.model：保存的 skip_gram 模型

3.2 数据预处理

读入小说文件的过程中，要将非中文字符处理掉，只考虑中文字符。为了便于后续训练过程，每一份语料所对应的标签为该小说名，将小说名索引化，id为0-15。

```
def get_texts():
    import_stopwords()
    corpus_context_dict = {}
    id_corpus_dict = {}
    id = 0
    for file in get_files():
        simple_name = str(file).split(os.sep)[1].split('.')[0]
        with open(file, 'rb') as f:
            context = f.read()
            real_encode = chardet.detect(context)['encoding']
            context = context.decode(real_encode, errors='ignore')
            new_context = ''
            for c in context:
                if is_chinese(c):
                    new_context += c
            # for sw in stopwords_list:
            #     new_context = new_context.replace(sw, '')
            corpus_context_dict[simple_name] = new_context
            id_corpus_dict[id] = simple_name
            id += 1
        print(id)
    return corpus_context_dict, id_corpus_dict
```

之后需要对其进行禁用词过滤。首先要均匀的从16个小说中进行语料采样，采集250个段落，每种语料的词数约为500（本实验初始值设置为600，考虑到禁用词过滤会过滤掉一部分），采用jieba库对每一种语料进行分词，分词之后采用baidu_stopwords进行禁用词过滤。最后将语料与索引化的标签合并成元组，进行下一步训练操作。

```
def get_dataset():
    data = []
    for i in range(para_num):
        context = corpus_context_dict[id_corpus_dict[i % 16]]
        rand_value = randint(0, len(context) - per_para_words)
        new_context = context[rand_value:rand_value + per_para_words]
```

```
            cut_list = list(jieba.cut(new_context, cut_all=False))
            filter_list = []
            for c in cut_list:
                if stopwords_list.__contains__(c):
                    continue
                filter_list.append(c)
            data.append((i % 16, filter_list))
    return data
```

3.3 Word2Vec 模型训练

gensim是一个自然语言处理工具包，主要用于主题建模，文本索引和相似度检索。实现了很多流行的算法，比如潜在语义分析LSA，LDA和word2vec等。gensim中的word2vec模块的内容，该算法包括跳字模型和连续词袋模型，都使用了层次化softmax和负采样技术进行优化。本实验采用 gensim 中的 Word2Vec 模型来进行训练。

```
#min_count是最低出现数，默认数值是5；
#size是gensim Word2Vec将词汇映射到的N维空间的维度数量（N）默认的size数是100；
#iter是模型训练时在整个训练语料库上的迭代次数，假如参与训练的文本量较少，就需要把这个参数调大一些。iter的默认值为5；
#sg是模型训练所采用的的算法类型：1 代表 skip-gram，0代表 CBOW，sg的默认值为0；
#window控制窗口，如果设得较小，那么模型学习到的是词汇间的组合性关系（词性相异）；如果设置得较大，会学习到词汇之间的聚合性关系（词性相同）。模型默认的window数值为5；
```

```
word2vec_model_cb = Word2Vec(sentences=PathLineSentences('train_data'), hs=1, min_count=10, window=5, vector_size=200, sg=0, workers=16, epochs=10)
word2vec_model_sg = Word2Vec(sentences=PathLineSentences('train_data'), hs=1, min_count=10, window=5, vector_size=200, sg=1, workers=16, epochs=10)
word2vec_model_cb.save('cbow.model')
word2vec_model_sg.save('skip_gram.model')
```

3.4 KMeans 聚类

1. K-means算法，也称为K-平均或者K-均值，一般作为掌握聚类算法的第一个算法。
2. 这里的K为常数，需事先设定，通俗地说该算法是将没有标注的 M 个样本通过迭代的方式聚集成K个簇。
3. 在对样本进行聚集的过程往往是以样本之间的距离作为指标来划分。

```
    word_vectors = []
    for tmp_word in highest_words:
        word_vectors.append(model.wv[tmp_word])
    tSNE = TSNE()
    word_embeddings = tSNE.fit_transform(word_vectors)
    classifier = KMeans(n_clusters=16)
    classifier.fit(word_embeddings)
    labels = classifier.labels_

    for i in range(len(word_embeddings)):
        plt.plot(word_embeddings[i][0], word_embeddings[i][1],
markers[labels[i]])
    plt.axis([min_left, max_right, min_bottom, max_top])
    plt.savefig("./kmeans_result.png")
```

四、实验结果

- 对测试集进行实验代码

```
word2vec_model = Word2Vec.load('skip_gram.model')
        cluster(word2vec_model)
        test_name = ['郭靖', '华山派', '蛤蟆功', '九阴真经']
        for name in test_name:
            print(name)
            for result in word2vec_model.wv.similar_by_word(name,
topn=10):
                print(result[0], '{:.3f}'.format(result[1]))
```

使用Word2Vec模型对金庸的五本小说进行了词向量的构建和聚类，结果显示与某个词(选取小说主角)相似的词在原著中也有一定的联系，在原著中有联系的一系列词(以人物为例)构建而成的词向量距离较近，使用Kmeans聚类效果良好。

-

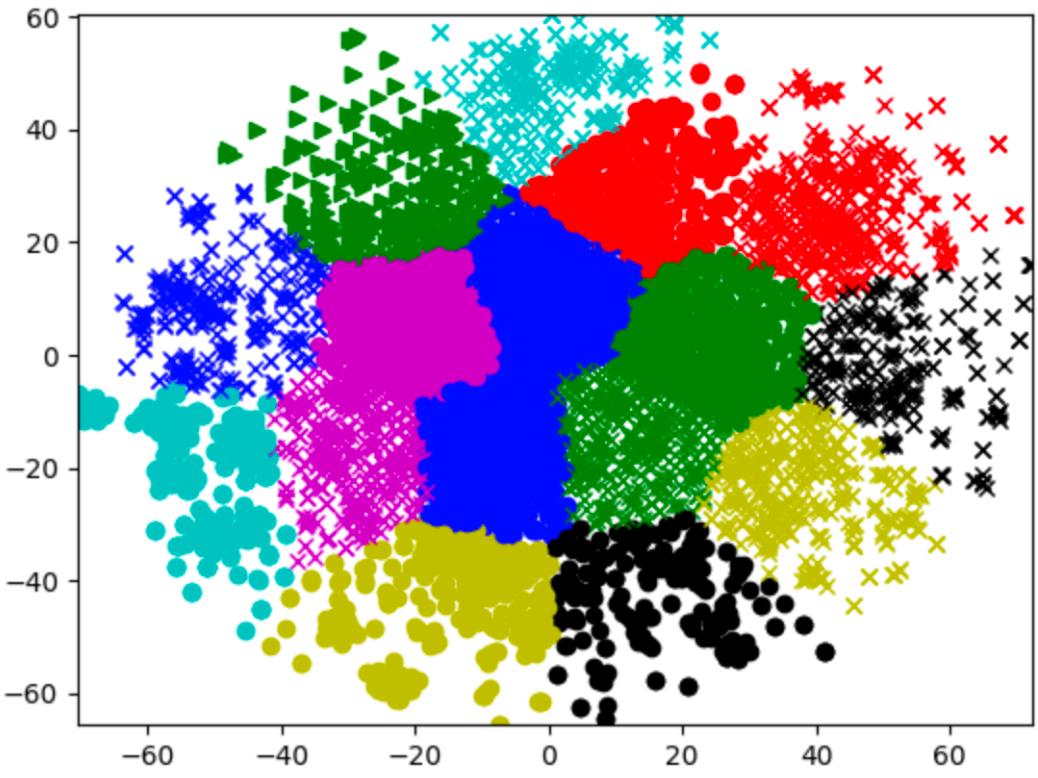
CBOW模型	Top1	Top2	Top3	Top4	Top5
郭靖	黄蓉 0.785	欧阳锋 0.697	欧阳克 0.664	黄药师 0.663	杨过 0.632
华山派	本派 0.596	青城派 0.572	峨嵋派 0.570	贵派 0.563	本门 0.542
蛤蟆功	西毒 0.540	降龙十八掌 0.491	掌力 0.460	一阳指 0.454	阳刚 0.440
九阴真经	经文 0.653	真经 0.642	葵花宝典 0.613	宝典 0.607	下卷 0.580

-

SkipGram 模型	Top1	Top2	Top3	Top4	Top5
郭靖	黄蓉 0.837	杨过 0.728	黄药师 0.681	洪七公 0.676	欧阳锋 0.674
华山派	本派 0.629	派 0.624	嵩山 0.609	门下 0.607	恒山 0.606
蛤蟆功	欧阳锋 0.621	西毒 0.566	洪七公 0.538	降龙十八掌 0.523	一阳指 0.516
九阴真经	真经 0.689	经文 0.670	下卷 0.668	中所载 0.619	总纲 0.610

- 聚类结果

  采用 K-means 聚类方法对这些词向量进行聚类，并利用 tSNE 方法将聚类结果进行可视化。通过下图可以看出，16本小说都都有着较好的聚类效果。



- 训练词向量为什么一般采用无监督的方式？还有其他办法来获得词向量吗？
  其实训练词向量有非常多的方法和结构，完全取决于你的下游任务，哪怕是文本分类任务，实体识别任务都可以，关键是最终得到全连接层的参数矩阵。虽然是这样，但是由于很多下游任务数据量很小，带来的问题就是，很容易过拟合。因此一般先采用大规模的无监督语料训练初始的词向量，减低过拟合风险。后续在下游任务再进行微调即可。

- 为什么词向量会具有相似上下文的词向量夹角余弦、向量欧氏距离更接近，体现出相似性?
  来源于Harris提出的分布假说（distributional hypothesis），即“上下文相似的词，其语义也相似”。在词向量中，由于我们的训练过程是有窗口的，每次相同窗口内的词对应的参数矩阵的相应位置会一同得到更新，这种更新会随着窗口内类似词语的共现次数的增加，而不断积累，最终训练好的词向量会非常接近。