LDA模型

1 任务描述

利用给定语料库（或者自选语料库），利用神经语言模型（如：Word2Vec， GloVe等模型）来训练词向量，通过对词向量的聚类或者其他方法来验证词向量的有效性。

2 算法原理

2.1 word2vec

句子之中相近的词之间是有联系的，比如今天后面经常出现上午、下午。所以它的基本思想就是用词来预测词。准确的说，word2vec仍然是一种编码方式，将一个个的词给编码成向量，但是被他编码而成的向量并不是随便生成的，而是能够体现这些单词之间的关系（如相似性等）。

word2vec主要包含两个模型：

- 跳字模型（skip-gram）：

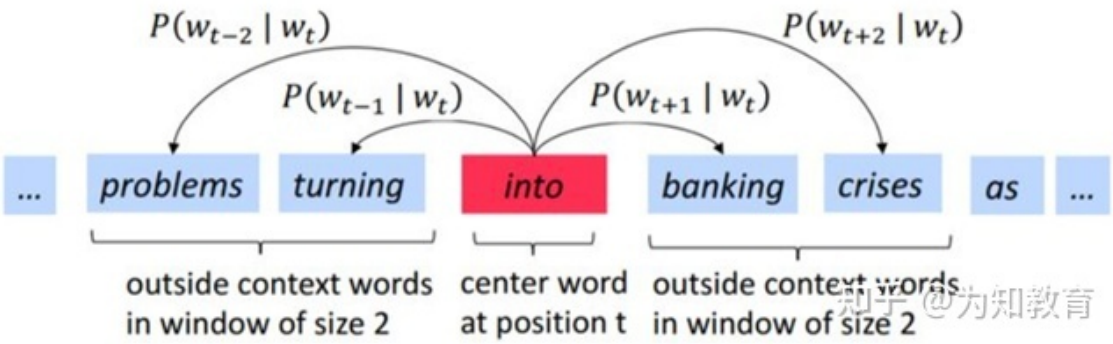
用当前词来预测上下文。相当于给你一个词，让你猜前面和后面可能出现什么词。

- 连续词袋模型（CBOW，continuous bag of words）：

通过上下文来预测当前值。相当于一句话中扣掉一个词，让你猜这个词是什么。

2.1.1 Skip-gram（跳字模型）

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。



拿到一个文本，遍历文本中所有的位置，对于文本中的每个位置，我们都会定义一个围绕中心词汇大小为2m的窗口，这样就得到了一个概率分布，可以根据中心词汇给出其上下文词汇出现的概率。

2.1.2 CBOW（连续词袋模型）

连续词袋模型与跳字模型类似，最大的不同在于，连续词袋模型假设基于某中心词在文本序列前后的背景词来生成该中心词。

例如：‘我’，‘爱’，‘红色’，‘这片’，‘土地’，窗口大小为2，就是用‘我’，‘爱’，‘这片’，‘土地’这四个背景词，来预测生成‘红色’这个中心词的条件概率，即：P（红色|我，爱，这片，土地）

给定一个长度为T的文本序列，设时间步t的词为W(t),背景窗口大小为m。则连续词袋模型的目标函数（损失函数）是由背景词生成任一中心词的概率。

$$\prod_{t=1}^{T} P\left(w^{(t)} \mid w^{(t-m)}, \cdots, w^{(t-1)}, w^{(t+1)}, \cdots, w^{(t+m)}\right)$$

因为连续词袋模型的背景词有多个，我们将这些背景词向量取平均，然后使用和跳字模型一样的方法来计算条件概率。

2.2 K-Means

K-Means 是我们最常用的基于欧式距离的聚类算法，其认为两个目标的距离越近，相似度越大。

算法步骤为：

1. 选择初始化的 k 个样本作为初始聚类中心 $a = a_1, a_2, \ldots a_k$；
2. 针对数据集中每个样本 x_i 计算它到 k 个聚类中心的距离并将其分到距离最小的聚类中心所对应的类中；
3. 针对每个类别 a_j，重新计算它的聚类中心 $a_j = \frac{1}{|c_i|} \sum_{x \in c_i} x$（即属于该类的所有样本的质心）；
4. 重复上面 2 3 两步操作，直到达到某个中止条件（迭代次数、最小误差变化等）。

3 任务过程

首先使用`jieba`对16本金庸小说的合集进行分词，将分词结果写入本地的文件中。

```
def word_seg(path):
    files = os.listdir(path)
    for file in files:
        position = path + '\\' + file
        with open(position, "r", encoding="ANSI") as f:
            data = f.read()
            f.close()
        text = cut_sentences(data)
        with open(file_w, "a", encoding="utf-8") as f:
            for sentence in text:
                sentence = re.sub('[^\u4e00-\u9fa5]+', '', sentence)
                f.write(" ".join(jieba.lcut(sentence, use_paddle=True,
cut_all=False)) + '\n')
```

接着用`gensim`中的`word2vec`训练词向量，并将单词对应的词向量标准化，打包输出成键为单词，值为词向量的字典。

```
def vec_gen(path):
    train_data = word2vec.LineSentence(path)
    model = word2vec.Word2Vec(train_data,
                              vector_size=100,
                              window=5,
                              workers=4)
    model.wv.vectors = model.wv.vectors / (np.linalg.norm(model.wv.vectors,
axis=1).reshape(-1, 1))
    vec_dist = dict(zip(model.wv.index_to_key, model.wv.vectors))
    with open('vec_dist', 'wb') as f:
        pkl.dump(vec_dist, f)
```

最后使用 K-Means 对指定单词的词向量进行聚类分析，并将词向量使用 PCA 降维后绘图。

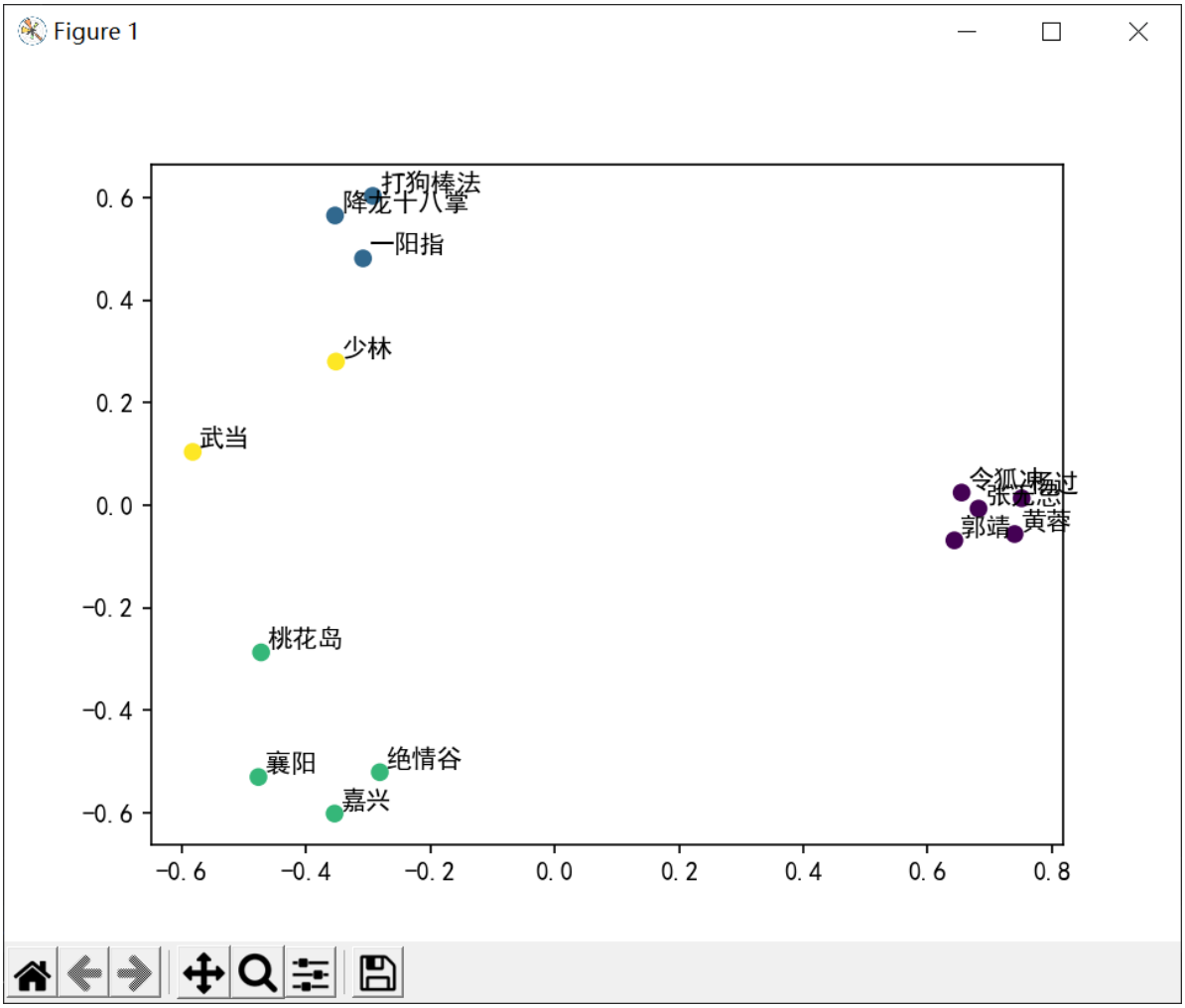
```
def cluster(data):
    with open('vec_dist', 'rb') as f:
        vec_dist = pkl.load(f)
    vec = []
    for d in data:
        vec.append(vec_dist[d])
    center, label, inertia = k_means(vec, n_clusters=4)
    vec = PCA(n_components=2).fit_transform(vec)

    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.scatter(vec[:, 0], vec[:, 1], c=label)
    for i, w in enumerate(data):
        plt.annotate(text=w, xy=(vec[:, 0][i], vec[:, 1][i]),
                     xytext=(vec[:, 0][i] + 0.01, vec[:, 1][i] + 0.01))
    plt.show()
```

4 任务结果

输入以下词汇：

```
['郭靖', '黄蓉', '杨过', '嘉兴', '襄阳', '张无忌', '令狐冲', '桃花岛', '绝情谷', '少林',
'武当', '降龙十八掌', '打狗棒法', '一阳指']
```

将K-Means的簇数设为4，聚类结果如下：



可以看到，所有的词汇总共被划分成了人名、地名、门派、武功四类，结果显示用神经语言模型Word2Vec训练词向量效果较好。