

0、概念

“**文本表示**”简单的说就是将自然语言产生的文本在计算机内表示，从而为下一步任务进行处理和计算；除此之外，有时也称作“**文本向量化**”、“**词嵌入 (word embedding)**”、“**词向量**”，概念上有一定差别，但都是转为机器能够进行计算的形式；除此之外，按照颗粒度大小，可分为字、词、句子、篇章几个层次；

如“我爱自然语言处理”这句话，人类可以很好的理解，但是计算机无法直接理解其含义，为了使计算机可以进行后续的分类、聚类等等任务，必须将文本转为机器可以识别的形式，也就是对文本进行表示；

目前已经有很多很多种方式来实现文本表示，如基础的One-hot、TF-IDF再到Word2Vec、BERT等，方法很多也很成熟了，因时间安排限制，对部分算法的原理部分做简要讲解，详细原理请自己查阅论文、博客等搜集相关资料，作为辅助理解；

1、分类

按照不同分类标准存在多种分类方式，下面论述两类主要分类方法：

- 基于one-hot、tf-idf、textrank等的bag-of-words；
- 主题模型：LSA (SVD) 、pLSA、LDA；
- 基于词向量的静态表征：word2vec、fastText、glove
- 基于词向量的动态表征：elmo、GPT、bert

2、原理解释

2.1 One-hot

One-hot简称读热向量编码，也是特征工程中最常用的方法。其步骤如下：

1. 构造文本分词后的字典，每个分词是一个比特值，比特值为0或者1。
2. 每个分词的文本表示为该分词的比特位为1，其余位为0的矩阵表示。

例如：

language	
1	John likes to watch movies. Mary likes too
2	John also likes to watch football games.

以上两句可以构造一个词典：

language	
1	{"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6, "football": 7, "games": 8, "Mary": 9, "too": 10}

每个词典索引对应着比特位。那么利用One-hot表示为

```

1 John likes to watch movies. Mary likes too
2 John:      [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
3 likes:     [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
4 to:        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
5 watch:     [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
6 movies:    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
7 also:      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
8 football:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
9 games:     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
10 Mary:     [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
11 too:      [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
12
13
14 # John also likes to watch football games.
15 John:      [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
16 likes:     [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
17 to:        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
18 watch:     [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
19 movies:    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
20 also:      [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
21 football:  [0, 1, 0, 0, 0, 0, 1, 0, 0, 0]
22 games:     [0, 1, 0, 0, 0, 0, 0, 1, 0, 0]
23 Mary:     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
24 too:      [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```

One-hot表示文本信息的 **缺点**：

- 随着语料库的增加，数据特征的维度会越来越大，产生一个维度很高，又很稀疏的矩阵。
- 这种表示方法的分词顺序和在句子中的顺序是无关的，不能保留词与词之间的关系信息。

代码

```

1 def doc2onhot_matrix(file_path):
2     """
3     文本向量化 One-Hot
4     1.文本分词
5     """
6     # (1)读取待编码的文件
7     with open(file_path, encoding = "utf-8") as f:
8         docs = f.readlines()
9
10    # (2)将文件每行分词，分词后的词语放入 words 中
11    words = []
12    for i in range(len(docs)):
13        docs[i] = jieba.lcut(docs[i].strip("\n"))
14        words += docs[i]
15
16    # (3)找出分词后不重复的词语，作为词袋，是后续 onehot 编码的维度，放入 vocab 中
17    vocab = sorted(set(words), key = words.index)
18

```

```

19         # (4)建立一个 M 行 V 列的全 0 矩阵, M 是文档样本数, 这里是行数, V 为不重复词语
20 数, 即编码维度
21     M = len(docs)
22     V = len(vocab)
23     onehot = np.zeros((M, V))
24     for i, doc in enumerate(docs):
25         for word in doc:
26             if word in vocab:
27                 pos = vocab.index(word)
28                 onehot[i][pos] = 1
29     onehot = pd.DataFrame(onehot, columns = vocab)
30     return onehot

```

2.2 词袋模型

词袋模型(Bag-of-words model), 像是句子或是文件这样的文字可以用一个袋子装着这些词的方式表现, 这种表现方式不考虑语法以及词的顺序。

文档的向量表示可以直接将各词的词向量表示加和。 例如:

```

python
1 John likes to watch movies. Mary likes too
2 John also likes to watch football games.
3
4 构造字典为:
5 {
6     "John": 1,
7     "likes": 2,
8     "to": 3,
9     "watch": 4,
10    "movies": 5,
11    "also": 6,
12    "football": 7,
13    "games": 8,
14    "Mary": 9,
15    "too": 10,
16 }
17
18 John likes to watch movies, Mary likes too. → [1, 2, 1, 1, 1, 0, 0, 0, 1, 1]
19 John also likes to watch football games. → [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
20
21 解释: [1,2,1,1,1,0,0,0,1,1], 其中的2表示likes在该句中出现了2次, 依次类推。

```

缺点:

- 词向量化后, 词与词之间是有大小关系的, 不一定词出现的越多, 权重越大。
- 词与词之间是没有顺序关系的。

代码:

```
python
1 from sklearn import CountVectorizer
2
3 count_vect = CountVectorizer(analyzer = "word")
4
5 # 假定已经读进来 DataFrame, "text"列为文本列
6 count_vect.fit(trainDF["text"])
7
8 # 每行为一条文本, 此句代码基于所有语料库生成单词的词典
9 xtrain_count = count_vect.transform(train_x)
```

2.3 TF-IDF

TF-IDF (term frequency-inverse document frequency) 是一种用于信息检索与数据挖掘的常用加权技术。TF意思是词频(Term Frequency), IDF意思是逆文本频率指数(Inverse Document Frequency)。

字词的重要性随着它在文件中出现的次数成正比增加, 但同时会随着它在语料库中出现的频率成反比下降。一个词语在一篇文章中出现次数越多, 同时所有文档中出现次数越少, 越能够代表该文章。

、

$$TF_w = \frac{\text{在某一类中词条}w\text{出现的次数}}{\text{该类中所有的词条数目}}$$

$$IDF = \log\left(\frac{\text{文档总数}}{\text{包含词条}w\text{的文档总数} + 1}\right)$$

分母之所以加1, 是为了避免分母为0。

那么, `TF-IDF=TF*IDF`, 从这个公式可以看出, 当w在文档中出现的次数增大时, 而TF-IDF的值是减小的, 所以也就体现了以上所说的了。

缺点: 还是没有把词与词之间的关系顺序表达出来

代码:

```
python
1 from sklearn import TfidfVectorizer
2 from sklearn import HashingVectorizer
3
4 # word level tf-idf
5 tfidf_vect = TfidfVectorizer(analyzer = "word", token_pattern = r"\w{1,}"
6 , max_features = 5000)
7 tfidf_vect.fit(trianDF["text"])
8 xtrain_tfidf = tfidf_vect.transform(train_x)
9
10 # n-gram level tf-idf
11 tfidf_vect_ngram = TfidfVectorizer(analyzer = "word", token_pattern =
```

```

11 r"\w{1,}", ngram_range(2, 3), max_features = 5000)
12 tfidf_vect_ngram.fit(trainDF["text"])
   xtrain_tfidf = tfidf_vect.transform(train_x)

```

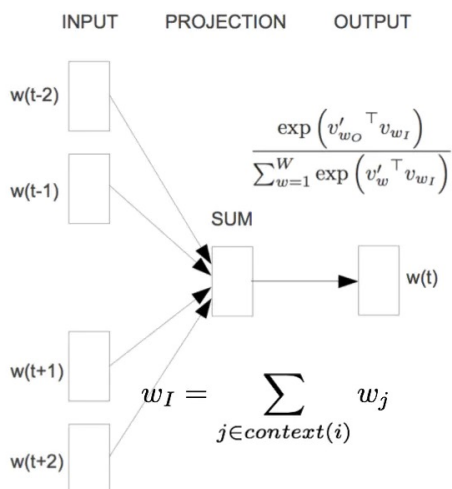
2.4 Word2Vec

谷歌2013年提出的Word2Vec是目前最常用的词嵌入模型之一。Word2Vec实际是一种浅层的神经网络模型，它有两种网络结构，分别是CBOW（Continuous Bag of Words）连续词袋和Skip-gram。

注：此部分难度较大，仅做简单介绍，代码部分将在后续任务中提及；

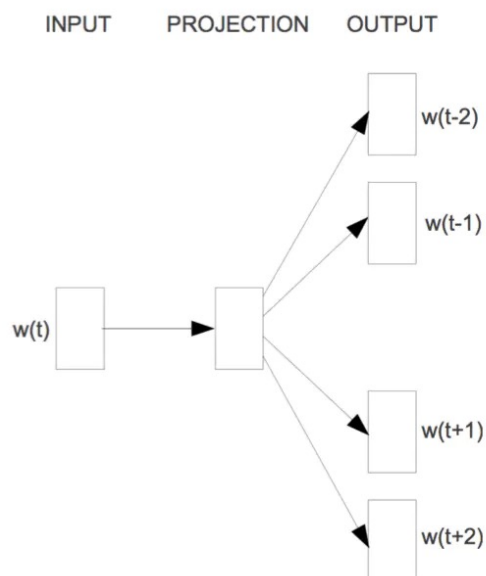
CBOW

CBOW获得中间词两边的上下文，然后用周围的词去预测中间的词，把中间词当做y，把窗口中的其它词当做x输入，x输入是经过one-hot编码过的，然后通过一个隐层进行求和操作，最后通过激活函数softmax，可以计算出每个单词的生成概率，接下来的任务就是训练神经网络的权重，使得语料库中所有单词的整体生成概率最大化，而求得的权重矩阵就是文本表示词向量的结果。



Skip-gram :

Skip-gram是通过当前词来预测窗口中上下文词出现的概率模型，把当前词当做x，把窗口中其它词当做y，依然是通过一个隐层接一个Softmax激活函数来预测其它词的概率。如下图所示：



2.5 BERT等

后续的BERT、ELMo等模型和上述的模型差别又很大了，简单的介绍是说不清楚的，大家知道文本表示除了上述这些模型外，也还有BERT等模型，且其性能更好，表现更佳；

代码使用等将在后续任务中涉及；

3、参考资料

论文：

- **Word2vec** : [\[🔗\] Parameter Learning Explained](#)
- **GloVe** : [\[🔗\] Global Vectors for Word Representation](#)
- **FastText**: [\[🔗\] Bag of Tricks for Efficient Text Classification](#)
- 综述: [\[🔗\] From static to dynamic word representations: a survey](#)

注：英文论文阅读有困难可以去知网找中文的相关论文，也是一大堆。

视频：

- [🔗 词向量汇总：word2vec、fastText、GloVe、BERT等_哔哩哔哩_bilibili](#)


4、任务

这一部分其实属于基础内容，如果直接跳过到具体任务环节，零基础的同学后面理解起来会很费劲，但是具体讲起来也没有什么实际操作的内容，都是概念性的东西；重要的是了解原理及目前可用的模型，因而不布置代码任务，但需要完成一个实验文档，帮助自己理解原理；

(word2vec及后面复杂的模型如果看不懂没关系，先自己整理一下)

要求：

- 根据介绍的文本表示模型及参考资料，写一个对文本表示的理解文档，类似文献综述，字数不限；

- 主要包括：One-Hot、词袋模型、TF-IDF、Word2Vec、FastText、Bert、GPT等，其余模型可写可不写
- 可以去找论文，也可以在知乎、公众号等搜索一些博客
 - 搜索关键词：**文本表示 词向量 词嵌入 (word Embedding)**
- 文档请用 `Markdown` 格式书写，个人建议涉及代码公式的话就不要使用word了，操作上很不方便且臃肿；markdown编辑器软件推荐 `Typora`、`MarkText`；都是“可见即所得”模式的markdown编辑器，操作也非常简单，软件上方都有文字提示，也可以自己去找找教程，如下方链接；
 -  [Typora 完全使用详解 - 少数派 \(sspai.com\)](https://sspai.com/post/3544)

提交方式

- 在QQ群里提交 `markdown文件` 或 `word文件`