实验题目

- One-hot编码实验
- Word2vec词向量训练

实验内容

- 实现 one-hot 编码

```
x=['体育', '军事', '娱乐', '教育', '文化', '时尚', '科技', '财经']
onehot_encode(x)
```

- 基于Python和gensim框架实现Word2vec在Wikipedia语料集上面的应用，并且获取词的词向量以及寻找相近词。

遇到和解决的问题

本次实验遇到的问题就是不太了解 `OneHotEncoder` 和 `Word2Vec` 的使用方法。本次实验过程也是搞清楚了这两个函数的一些用法，均在实验步骤中进行了说明，不在此赘述了。

实验步骤

one-hot

基础知识介绍

sklearn.preprocessing.OneHotEncoder

```
class sklearn.preprocessing.OneHotEncoder(*, categories='auto',
drop=None, sparse=True, dtype=<class 'numpy.float64'>,
handle_unknown='error')
```

将特征类别编码成 `one-hot` 数值数组

这个转换器的输入应当是**数值型数组或字符串数组（二维的）**，表示分类（离散）特征所取的值。这些特征被使用 `one-hot` 策略编码。这将为每个类别创建二进制列。每一列认为是一个 `feature`。默认情况下， `encoder` 根据每个特征中的唯一值生成类别。你也可以手动指定类别 `categories`。针对每个 `feature` 的二进制列，只有一个位置为1，其余位置都是 0。

我认为，下面的参数和属性是常用的，其余参数没有详细和属性没有详细介绍

handle_unknown	{'error', 'ignore'}, default='error' 在转换过程中遇到未知分类特征时，是引发错误还是忽略（默认为引发）。当此参数设置为"ignore"并且在转换过程中遇到未知类别时，这一特征的 `one-hot` 编码列将全置为 0。在逆变换中，未知类别将表示为 `None`
参数	说明

属性	说明
categories_	list of arrays 拟合期间确定的每个特征的类别（按X中特征的顺序，并与转换的输出相对应）。这包括下拉列表中指定的类别（如果有）。

使用前需要先导入

```
from sklearn.preprocessing import OneHotEncoder
```

用法

数值型整数

- 整数

```
encoder = OneHotEncoder()
encoder.fit([
    [0, 2, 1, 12],
    [2, 3, 5, 3],
    [1, 3, 2, 12],
    [1, 2, 4, 3]
])
encoded_vector = encoder.transform([[2, 3, 5, 3]]).toarray()
print("\n Encoded vector =", encoded_vector)
>> Encoded vector = [[0. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0.]]
```

  - 默认情况下，将每一列认为是一个 `feature`，一列中的每个不同的值，都认为是一个 `category`
  - 第一列中有 `[0, 1, 2]`，三种值，因为需要三位数，我们将第一列编码为
    - 0: `[1, 0, 0]`
    - 1: `[0, 1, 0]`
    - 2: `[0, 0, 1]`
  - 第三列中有 `[1, 2, 4, 5]` 四种值，因此需要四位数，我们将第三列编码为
    - 1: `[1, 0, 0, 0]`

        - 2：`[0, 1, 0, 0]`
        - 4：`[0, 0, 1, 0]`
        - 5：`[0, 0, 0, 1]`

    同时，我们可以查看 `encoder` 的属性 `categories:代表拟合期间确定的每个feature的categories`

    ```
    print(encoder.categories_)
    [array([0, 1, 2]), array([2, 3]), array([1, 2, 4, 5]), array([ 3,
    12])]>>
    ```

    - 根据 `categories_` 的输出，和上面的例子可以看到，每个 `feature` 的 `category` 是从小到大排列，并进行编码的。每个 `feature` 中的数在 `categories_` 中的位置就是该数编码后，1的位置。
- 小数：同上面相同，这里证明，也可以针对小数进行编码

    ```
    encoder = OneHotEncoder()
    encoder.fit([
        [0, 2.1, 1, 12],
        [1, 3.2, 5, 3],
        [2, 3.3, 2, 12],
        [1, 2.1, 4, 3]
    ])
    encoded_vector = encoder.transform([[2, 3.2, 5, 3]]).toarray()
    print("\n Encoded vector =", encoded_vector)
    >>  Encoded vector = [[0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0.]]
    ```

字符串型数组

```
encoder = OneHotEncoder()
encoder.fit([['体育', '军事'],
             ['计科','开心'],
             ['娱乐', '军事']])
encoded_vector = encoder.transform([['计科', '开心']]).toarray()
print("\n Encoded vector =", encoded_vector)
print(encoder.categories_)
>>Encoded vector = [[0. 0. 1. 0. 1.]]
>>[array(['体育', '娱乐', '计科'], dtype=object), array(['军事', '开
心'], dtype=object)]
```

- 可以看到，字符串型与数值型类似，编码方式也相同

handle_unknown

- 默认情况下，`handle_unknow = error`，当遇到 `transform` 时遇到 `fit` 中没有出现过的特征类别时，会直接报错
- `handle_unknown = ignore`

```
encoder = OneHotEncoder(handle_unknown='ignore')
encoder.fit([['体育', '军事'],
             ['计科','开心'],
             ['娱乐', '军事']])
encoded_vector = encoder.transform([['计科', '难过']]).toarray()
print("\n Encoded vector =", encoded_vector)
print(encoder.categories_)
>>Encoded vector = [[0. 0. 1. 0. 0.]]
>>[array(['体育', '娱乐', '计科'], dtype=object), array(['军事', '开心'], dtype=object)]
```

  - 可以看到，`transform` 时遇到了 `fit` 第二列中没有出现过的 `难过`，因此，将第二列编码的位置都置为 0。

反向transform

通过编码，来解码出 `feature` 对应的类别

```
encoder = OneHotEncoder(handle_unknown='ignore')
encoder.fit([['体育', '军事'],
             ['计科','开心'],
             ['娱乐', '军事']])
encoded_vector = encoder.transform([['计科', '难过']]).toarray()
print(encoder.inverse_transform([[0, 0, 1, 0, 0]]))
>>[['计科' None]]
```

- 可以看到，在 `handle_unknown='ignore'` 时，将没见过的编码解码成了 `None`，默认 `handle_unknown` 下，直接报错

```
encoder = OneHotEncoder(handle_unknown='ignore')
encoder.fit([['体育', '军事'],
             ['计科','开心'],
             ['娱乐', '军事']])
encoded_vector = encoder.transform([['计科', '开心']]).toarray()
print(encoder.inverse_transform([[0, 0, 1, 0, 1]]))
>>[['计科' '开心']]
```

- 这是正常情况

实验情况

- 代码

```
def oneHot(x):
    encoder = OneHotEncoder(handle_unknown='ignore')
    encoder.fit([['体育'],
                 ['军事'],
                 ['娱乐'],
                 ['教育'],
                 ['文化'],
                 ['时尚'],
                 ['科技'],
                 ['财经']])
    ans_encoder = {}
    ans_category = {}
    for i in x:
        temp = encoder.transform([[i]]).toarray()
        ans_encoder[i] = temp[0]
        ans_category[i] = encoder.categories_[0].tolist().index(i)
    ans = (ans_encoder, ans_category)
    print(ans)

x=['体育', '军事', '娱乐', '教育', '文化', '时尚', '科技', '财经']
oneHot(x)
```

- 结果

```
(
{'体育': array([1., 0., 0., 0., 0., 0., 0., 0.]),
 '军事': array([0., 1., 0., 0., 0., 0., 0., 0.]),
 '娱乐': array([0., 0., 1., 0., 0., 0., 0., 0.]),
 '教育': array([0., 0., 0., 1., 0., 0., 0., 0.]),
 '文化': array([0., 0., 0., 0., 1., 0., 0., 0.]),
 '时尚': array([0., 0., 0., 0., 0., 1., 0., 0.]),
 '科技': array([0., 0., 0., 0., 0., 0., 1., 0.]),
 '财经': array([0., 0., 0., 0., 0., 0., 0., 1.])},
{'体育': 0, '军事': 1, '娱乐': 2, '教育': 3, '文化': 4, '时尚': 5,
'科技': 6, '财经': 7})
```

缺点

- 编码结果长度与 `feature` 中的类别个数相同，编码结果很容易变得过长，占用存储空间，影响运算速度
- 编码结果稀疏，信息密度低

word2vec

获取数据

- 下载 `zhwiki-latest-pages-articles.xml.bz2`，直接访问网址https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-articles.xml.bz2 即可。
- 代码

```
input_file_name = 'zhwiki-latest-pages-articles.xml.bz2'  # 数据路径
output_file_name = 'corpus_cn.txt'  # 输出数据路径
# 加载数据
input_file = WikiCorpus(input_file_name, dictionary={})
with open(output_file_name, 'w', encoding="utf8") as output_file:
    # 使用WikiCorpus类中的get_texts()方法读取文件，每篇文章转换为一行文本，并去掉标签符号等内容
    count = 0
    for text in input_file.get_texts():
        output_file.write(' '.join(text) + '\n')
        count = count + 1
        if count % 10000 == 0:
            print('已处理%d条数据' % count)
    print('处理完成！')

# 查看处理结果
with open('corpus_cn.txt', "r", encoding="utf8") as f:
    print(f.readlines()[:1])
```

  - 下载后的文件没法直接打开，我们使用 `WikiCorpus` 来进行读取。`lemmatize` 参数已经被舍弃，这里我们直接删除

数据预处理

```
def find_chinese(file):  # 使用正则表达式,来将不是中文的词都置为空
    pattern = re.compile(r'[^\u4e00-\u9fa5]')  # 返回一个 pattern 对象
    chinese = re.sub(pattern, '', file)  # sub是substitute的缩写，表示替换
    print(chinese)


cnt = 0  # 记录已经处理完的个数,方便观察程序运行状态
input = open('input.txt', "w", encoding="utf8")
with open('corpus_cn.txt', "r", encoding="utf8") as f:
    for line in f.readlines():
        cnt += 1
```

```
            line = convert(line, 'zh-cn')  # 将繁体转化为简体
            words = jieba.cut(line)  # 分词
            ans = []
            for i in words:
                pattern = re.compile(r'[^\u4e00-\u9fa5]')  # 去除不是中文的
词
                i = re.sub(pattern, '', i)  # 不是中文的都被置为空 ''
                if len(i) > 0:  # 不是空
                    ans.append(i)
            input.write(' '.join(ans) + '\n')  # 将处理好的数据按行写入输出文
件中,这将是下一步的输入
            if cnt % 10000 == 0:
                print(f'{cnt} preprocess finished')

input.close()
```

- 这里数据预处理之后，会将每一个 `ans` 中的词，以空格分隔，写入 `input.txt` 中。这是因为后面我们在训练模型时，输入参数需要使用 `LineSentence` 函数，该函数要求这样的数据格式。

这里，函数 `join()` 的使用方法可见 Python join()与os.path.join()介绍_长命百岁的博客-CSDN博客

模型训练(使用了所有的训练数据，2G+)

添加 `log` 信息，方便观察程序的运行状态和输出

```
program = os.path.basename(sys.argv[0])
logger = logging.getLogger(program)
#1.format: 指定输出的格式和内容，format可以输出很多有用信息，
#%(asctime)s: 打印日志的时间
#%(levelname)s: 打印日志级别名称
#%(message)s: 打印日志信息
logging.basicConfig(format='%(asctime)s: %(levelname)s: %(message)s')
logging.root.setLevel(level=logging.INFO)
#打印这是一个通知日志
logger.info("running %s" % ' '.join(sys.argv))
```

模型训练

```
inp = 'input.txt'  # inp:分好词的文本路径
outp1 = 'model.model'  # outp1:训练好的模型保存路径
outp2 = 'model.vector'  # outp2:得到的词向量保存路径
w2vModel = word2vec.Word2Vec(LineSentence(inp),vector_size=256,
window=5, min_count=5,workers=multiprocessing.cpu_count(),epochs=50)
```

Word2Vec函数参数简要介绍

```
classgensim.models.word2vec.Word2Vec(sentences=None, corpus_file=None,
vector_size=100, alpha=0.025, window=5, min_count=5,
max_vocab_size=None, sample=0.001, seed=1, workers=3,
min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75,
cbow_mean=1, hashfxn=<built-in function hash>, epochs=5, null_word=0,
trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False,
callbacks=(), comment=None, max_final_vocab=None, shrink_windows=True)
```

- `sentence`：是可迭代的，可以是包含 `token` 列表的列表，但是对于大型语料库，考虑是一个迭代对象，可以直接从磁盘/网络传输句子。See `BrownCorpus`, `Text8Corpus` or `LineSentence` in `word2vec` module for such examples。如果您不提供 `sentence`，则模型将保持未初始化状态
- `vector_size`：word vector 的维度，默认是 100
- `window`： 表示句子中当前和预测词之间的最大距离（扫描窗口大小），默认是 5
- `min_count`： 表示词的最小出现频率，出现总次数小于该值的词都将被忽略。默认是 5
- `workers`：训练过程中使用的线程的个数，默认是 3
- `sg`：训练时使用的算法，1 代表 `skip-gram`，否则是 `CBOW`。默认是 0
- `epochs`： 迭代次数
- `Word2Vec` 函数的输入应该是处理好的可迭代对象

实验过程中具体用法

- 如果是列表，要求进行分词

  ```
  sentences = [["Python", "深度学习", "机器学习"], ["NLP", "深度学习",
  "机器学习"]]
  model = Word2Vec(sentences, min_count=1)
  ```

- 但是一般语料文件都很大，因此我们使用的都是语料文件，而不是将其作为列表一直存在内存中。需要保证语料文件内部每一行对应一个句子（已经分词，以空格隔开）
  - 对于单个文件语料，我们可以使用 `LineSentence: 对包含句子的文件进行迭代:每行都是一句话，单词必须讲过预处理，并由空格分隔`。 `LineSentence` 直接

将语料转换成我们需要的形式。

- 拿到了分词后的文件，在一般的NLP处理中，会需要去停用词。由于word2vec的算法依赖于上下文，而上下文有可能就是停词。因此对于word2vec，我们可以不用去停词。

训练过程中一些 log 信息

```
2022-04-04 14:33:44,414: INFO: EPOCH - 50 : training on 227776772 raw
words (211443600 effective words) took 296.4s, 713450 effective
words/s
2022-04-04 14:33:44,415: INFO: Word2Vec lifecycle event {'msg':
'training on 11388838600 raw words (10572077834 effective words) took
14809.6s, 713866 effective words/s', 'datetime': '2022-04-
04T14:33:44.414960', 'gensim': '4.1.2', 'python': '3.6.13 |Anaconda,
Inc.| (default, Jun  4 2021, 14:25:59) \n[GCC 7.5.0]', 'platform':
'Linux-3.10.0-1160.el7.x86_64-x86_64-with-centos-7.9.2009-Core',
'event': 'train'}
2022-04-04 14:33:44,415: INFO: Word2Vec lifecycle event {'params':
'Word2Vec(vocab=751307, vector_size=256, alpha=0.025)', 'datetime':
'2022-04-04T14:33:44.415199', 'gensim': '4.1.2', 'python': '3.6.13
|Anaconda, Inc.| (default, Jun  4 2021, 14:25:59) \n[GCC 7.5.0]',
'platform': 'Linux-3.10.0-1160.el7.x86_64-x86_64-with-centos-7.9.2009-
Core', 'event': 'created'}
2022-04-04 14:33:44,415: INFO: Word2Vec lifecycle event
{'fname_or_handle': 'model.model', 'separately': 'None', 'sep_limit':
10485760, 'ignore': frozenset(), 'datetime': '2022-04-
04T14:33:44.415422', 'gensim': '4.1.2', 'python': '3.6.13 |Anaconda,
Inc.| (default, Jun  4 2021, 14:25:59) \n[GCC 7.5.0]', 'platform':
'Linux-3.10.0-1160.el7.x86_64-x86_64-with-centos-7.9.2009-Core',
'event': 'saving'}
2022-04-04 14:33:44,416: INFO: storing np array 'vectors' to
model.model.wv.vectors.npy
2022-04-04 14:33:44,861: INFO: storing np array 'syn1neg' to
model.model.syn1neg.npy
2022-04-04 14:33:45,248: INFO: not storing attribute cum_table
2022-04-04 14:33:45,621: INFO: saved model.model
2022-04-04 14:33:45,621: INFO: KeyedVectors lifecycle event
{'fname_or_handle': 'word2vec.wordvectors', 'separately': 'None',
'sep_limit': 10485760, 'ignore': frozenset(), 'datetime': '2022-04-
04T14:33:45.621476', 'gensim': '4.1.2', 'python': '3.6.13 |Anaconda,
Inc.| (default, Jun  4 2021, 14:25:59) \n[GCC 7.5.0]', 'platform':
'Linux-3.10.0-1160.el7.x86_64-x86_64-with-centos-7.9.2009-Core',
'event': 'saving'}
2022-04-04 14:33:45,621: INFO: storing np array 'vectors' to
word2vec.wordvectors.vectors.npy
2022-04-04 14:33:46,387: INFO: saved word2vec.wordvectors
```

训练结果保存

保存模型可以使用

```
model.save("word2vec.model")
```

如果有保存好的模型，我们还可以接着训练

```
model = Word2Vec.load("word2vec.model")
model.train([["hello", "world"]], total_examples=1, epochs=1)
```

训练好的 `word vector` 被存在 `KeyedVectors` 实例中，as `model.wv`

```
vector = model.wv['computer']  # get numpy vector of a word
sims = model.wv.most_similar('computer', topn=10)  # get other similar
words
```

将训练好的 `vector` 分离出来，单独存在 `KeyedVectors` 中是因为：如果不需要继续训练模型，可以不存储模型，只存储 `vector` 和对应的 `key` 就好了

这将产生一个更小、更快的对象，在映射时可以快速加载，并在进程之间共享RAM中的向量

```
# Store just the words + their trained embeddings. 保存
word_vectors = model.wv
word_vectors.save("word2vec.wordvectors")

# Load back with memory-mapping = read-only, shared across processes.
加载
wv = KeyedVectors.load("word2vec.wordvectors", mmap='r')
vector = wv['computer']  # Get numpy vector of a word
```

详细信息可见官方文档 [models.word2vec – Word2vec embeddings — gensim (radimrehurek.com)]

实验过程中使用的模型保存

```
model.save('model.model')  # 保存模型
word_vectors = model.wv
word_vectors.save("word2vec.wordvectors")  # 保存
```

模型测试（vector有一些长，就没放在这里）

- 测试1

```
wv = KeyedVectors.load("word2vec.wordvectors", mmap='r')
# vector = wv['北京大学']   # Get numpy vector of a word
print(wv.most_similar('清华大学', topn=10))
>>
[('北京大学', 0.8103098273277283), ('清华', 0.7029939889907837), ('浙江
大学', 0.6582461595535278), ('武汉大学', 0.6538274复旦大学',
0.6521641612052917), ('台湾大学', 0.6375373005867004), ('南京大学',
0.6282556056976318), ('同济大学', 0.624813大学', 0.622948169708252),
('东南大学', 0.6197461485862732)]
```

- 测试2

```
wv = KeyedVectors.load("word2vec.wordvectors", mmap='r')
print(wv.most_similar('山东', topn=10))
>>
[('山东省', 0.7513332962989807), ('济南', 0.7045312523841858), ('济宁',
0.6650033593177795), ('烟台', 0.6563841700553894)0.6358231902122498),
('河南', 0.6331247091293335), ('郓城', 0.6325219869613647), ('山西',
0.629845380783081), ('菏泽', 0.77958679), ('蒙阴', 0.5834060311317444)]
```

- 测试3

```
wv = KeyedVectors.load("word2vec.wordvectors", mmap='r')
print(wv.most_similar('自然语言', topn=10))
>>
[('语法', 0.5727444291114807), ('语义', 0.5635175704956055), ('语言',
0.5608547925949097), ('计算机程序', 0.5583971738815掘',
0.5534152388572693), ('编程语言', 0.5530121326446533), ('语法分析',
0.5513348579406738), ('程序语言', 0.549354493618', 0.5483173131942749),
('机器翻译', 0.5446789860725403)]
```

可以看出，模型结果是非常不错的，因为使用了全部的训练数据，且训练了 50 个 epochs

实验结果

两个实验的实验结果均在各自实验步骤中展示。在此不再赘述。两个实验的结果都比较不错，两种算法都是很有效的。

实验总结

one-hot编码

本次实验，学习了 `OneHotEncoder` 的使用方法，需要注意的就是要搞清楚列是 `feature`，每一列转换成一个二进制序列，转换时的 `feature` 数应该和 `fit` 时相同

但是 `ont-hot` 的缺点是太占用空间，而且信息密度低，导致矩阵比较稀疏。这在数据量大，特征数目多的情况下是很难使用的。

word2vec

本次实验，实现了从数据集获取，数据预处理到word2vec训练，获得结果的整个流程。熟悉了常说的 word embedding 的实现方式和流程，有了更好的理解。本次实验使用数据集大小 2G+，训练 50 epochs，实验效果不错，看到了 embedding 的结果和一些 similarity 的匹配功能。