

2020.06 By Kinghao

一、项目名称

亚马逊在线市场的用户评论预测与情感分析模型

——基于自然语言处理与深度学习

二、摘要

本项目是基于python语言与自然语言处理算法实现的，主要实现了三个功能：一是根据亚马逊市场近年来产品的评论构建对应的语言库；二是利用市场中用户已打分评论训练生成分类器，去预测未知评分评论的好坏；三是通过特定评论词提取用户情感。

三、项目背景及目的

选择这样一个研究主题的想法来源于2020年数学建模美赛的C题，这是一个较为典型的数据分析题目，除了多数论文中使用的可视化分析和统计学函数以外，我认为这道题目更适合用自然语言处理的方式训练用户评论数据，并从中提取情感建立一个语料库，用于对后面未评级的评论打标签。同时我还利用了一些自己学习的NLP处理方法，在这个项目中加入了相关词分析等题目之外的功能，相当于搭建了完整的评论分析语料库。

问题需求大致概述如下：亚马逊在其创建的在线市场中为顾客提供了对购买产品进行评级的机会，在评级之外用户还可以追加“评论”

来进一步表达对产品的意见和信息。公司希望了解用户评论的内容是否与他们的满意度具有关系，并且他们的评论内容是否会对销售策略产生影响，以及希望提取其中一些关键性的信息来改善产品发展。公司计划在线上市场上推出和销售三种新产品：微波炉，婴儿奶嘴和吹风机，对应着这三个产品题中也给了对应的评论数据，这部分数据也被用于本项目的训练与测试。

四、项目环境

Python 3.6

numpy 1.18.1

pandas 1.0.1

beautifulsoup4 4.8.2

nltk 3.4.5

matplotlib 3.1.3

scikit – learn 0.22.2

gensim 3.8.1

Pillow 7.0.0

五、项目原理

(1) *word2vec*

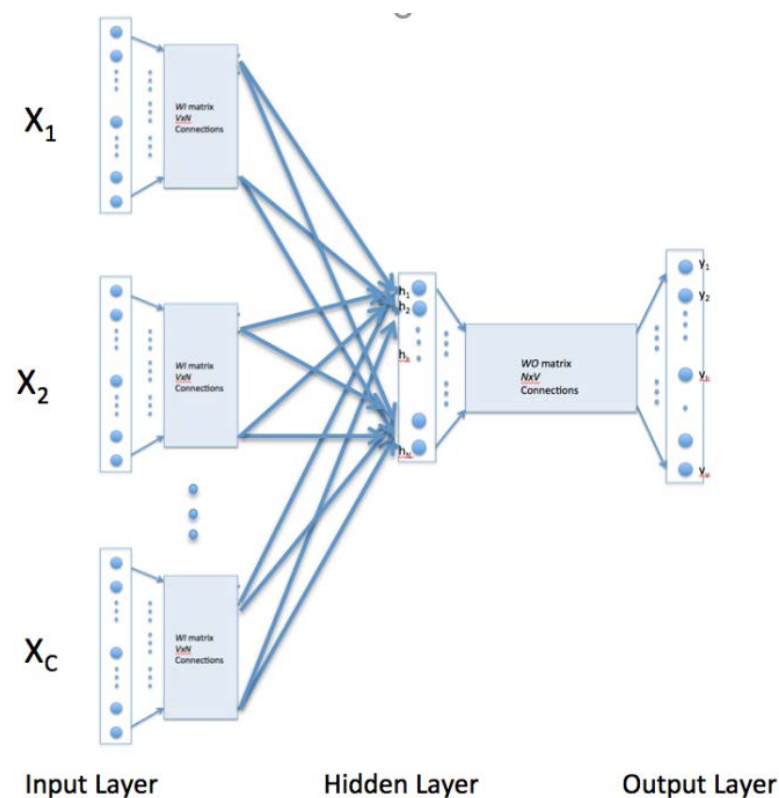
*word2vec*是一种把词转到某种向量空间的方法，在新的向量空间，词之间的相互关系，上下文关系都以某种程度被表征出来，使用

*word2vec*库也是本项目代码的重要部分。

*CBOW*模型是其中一种转换方法的训练输入是某一个特征词的上下文相关的词对应的词向量,而输出就是这特定的一个词的词向量。比如下面这段话,我们的上下文大小取值为4,特定的这个词是"*Learning*",也就是我们需要的输出词向量,上下文对应的词有8个,前后各4个,这8个词是我们模型的输入。由于*CBOW*使用的是词袋模型,因此这8个词都是平等的,也就是不考虑他们和我们关注的词之间的距离大小,只要在我们上下文之内即可。

...an efficient method for learning high quality distributed vector ...
context focus word context

*CBOW*结构图如下:



*Skip-Gram*模型的思路是反着来的,即输入是特定的一个词的词向

量，而输出是特定词对应的上下文词向量。还是上面的例子，我们的上下文大小取值为4，特定的这个词"*Learning*"是我们的输入，而这8个上下文词是我们的输出。我们的输入是特定词，输出是softmax概率排前8的8个词，对应的*Skip - Gram*神经网络模型输入层有1个神经元，输出层有词汇表大小个神经元。隐藏层的神经元个数我们可以自己指定。通过DNN的反向传播算法，我们可以求出DNN模型的参数，同时得到所有的词对应的词向量。这样当我们有新的需求，要求出某1个词对应的最可能的8个上下文词时，我们可以通过一次DNN前向传播算法得到概率大小排前8的softmax概率对应的神经元所对应的词即可。

(2) *RandomForest*

随机森林就是由多棵分类回归树(*CART*)构成的。对于每棵树，它们使用的训练集是从总的训练集中有放回采样出来的，这意味着，总的训练集中的有些样本可能多次出现在一棵树的训练集中，也可能从未出现在一棵树的训练集中。在训练每棵树的节点时，使用的特征是从所有特征中按照一定比例随机地无放回的抽取的，假设总的特征数量为 M ，这个比例可以是 \sqrt{M} , $1/2\sqrt{M}$, $2\sqrt{M}$ 。

因此，随机森林的训练过程可以总结如下：(1)给定训练集 S ，测试集 T ，特征维数 F 。确定参数：使用到的*CART*的数量 t ，每棵树的深度 d ，每个节点使用到的特征数量 f ，终止条件：节点上最少样本数 s ，节点上最少的信息增益 m ，对于第 $1 - t$ 棵树， $i = 1 - t$ ；(2)从 S 中有放回的抽取大小和 S 一样的训练集 $S(i)$ ，作为根节点的样本，从根节点

开始训练；(3)如果当前节点上达到终止条件，则设置当前节点为叶子节点，如果是分类问题，该叶子节点的预测输出为当前节点样本集合中数量最多的那一类 $c(j)$ ，概率 p 为 $c(j)$ 占当前样本集的比例；如果是回归问题，预测输出为当前节点样本集各个样本值的平均值。然后继续训练其他节点。如果当前节点没有达到终止条件，则从 F 维特征中无放回的随机选取 f 维特征。利用这 f 维特征，寻找分类效果最好的一维特征 k 及其阈值 th ，当前节点上样本第 k 维特征小于 th 的样本被划分到左节点，其余的被划分到右节点。继续训练其他节点。有关分类效果的评判标准在后面会讲；(4)重复(2)(3)直到所有节点都训练过了或者被标记为叶子节点。(5)重复(2),(3),(4)直到所有CART都被训练过。

利用随机森林的预测过程如下：对于第 $1-t$ 棵树， $i = 1-t$ ：(1)从当前树的根节点开始，根据当前节点的阈值 th ，判断是进入左节点($< th$)还是进入右节点($\geq th$)，直到到达，某个叶子节点，并输出预测值。(2)重复执行(1)直到所有 t 棵树都输出了预测值。如果是分类问题，则输出为所有树中预测概率总和最大的那一个类，即对每个 $c(j)$ 的 p 进行累计；如果是回归问题，则输出为所有树的输出的平均值。

六、实现过程与代码核心部分介绍

(1) 首先需要完成的代码是训练生成词向量库的`TR_model.py`，我先将拿到的评论数据分为三个部分，

```
def load_dataset(name, nrows=None):
    datasets = {
        "unlabeled_train": "unlabeledTrainData.tsv",
        "labeled_train": "labeledTrainData.tsv",
        "test": "testData.tsv"
    }
```

第一个是没有打情感标签的数据集`unlabeledTrainData.tsv`，这部分数据用于生成基础的`word2vec`词向量库，`word2vec`是一类神经网络模型，在给定无标签的语料库的情况下，为语料库的单词产生一个能表达语义的向量，与之对应的`tf-idf`分数决定的方式，能告诉我一个单词在评论中的相对重要性，而不能包含评论中的语义。

这部分数据集比较大，评论条数有将近五万条，在利用评论信息生成词向量库之前先要对评论做数据清洗，因为评论中的标点，缩写等字符对我们利用机器学习处理文本并没有很大的帮助。

```
eng_stopwords = {}.fromkeys([ line.rstrip() for line in open('../stopwords.txt')])
def clean_text(text):
    text = BeautifulSoup(text, 'html.parser').get_text()
    text = re.sub(r'^a-zA-Z', ' ', text)
    words = text.lower().split()
    words = [w for w in words if w not in eng_stopwords]
    return ' '.join(words)
```

这里的`BeautifulSoup`工具包作用是清除爬下来数据中的`html`标签，`stopwords.txt`是自定义的停用词文件，也是我们要忽略的一些没有重要语义的单词，`clean_text`的实现是一段单词拆分，最后一句代码将单词再一次组成一段文本，这是为了接下来在词袋模型中能更好地应用它。

```
from gensim.models.word2vec import Word2Vec
model = Word2Vec(sentences_list, workers=num_workers, size=nu
model.init_sims(replace=True)
model.save(os.path.join('../', 'models', model_name))
```

生成的词向量模型还包括了词之间的关联，这也是为什么上一步要重新整合句子，因为拆解的单词具有一定的情感向量，而拼接成句子可能意思想法，比如一个好词*good*前面加上*not*就是相反的意思，所以*word2vec*必须包含词前后出现的频率向量，综合判断是否语义相符。这里调用*Word2Vec*库生成词向量模型并将模型单独保存，这个词模型后面训练分类器的基础，最初在这部分代码与训练分类器是写在一个文件中，但由于题目中有多类产品都要测试分类器，而训练词向量模型的时间很长，避免每次运行都需要重新生成词向量库，这里将两部分代码解耦。

(2) 接下来是*Classification.py*，这部分训练的对象是有情感标签的*labeledTrainData.tsv*，数据加载和清洗的函数和第一个文件相同

```
df=load_dataset("labeled_train")
df['clean_review'] = df.review.apply(clean_text)
vectorizer = CountVectorizer(max_features = 5000) #对所有关键词的term
train_data_features = vectorizer.fit_transform(df.clean_review).toarr

forest = RandomForestClassifier(n_estimators=100)#随机森林分类
forest = forest.fit(train_data_features, df.sentiment)#开始数据训练
```

这部分用到了随机森林分类器，这是一种很灵活实用的方法，它具有很好的准确性并且适用于数据集比较大的分类，能够评估各个特征在分类问题上的重要性。要将一个输入样本进行分类，需要将输入样本输入到每棵树中进行分类。每棵树的生成规则是：如果训练集大小为 N ，对于每棵树而言，随机且有放回地从训练集中的抽取 N 个训练样本，作为该树的训练集。

(3) 训练好*forest*之后，即可以引入待分类的数据测试训练结果，

上面数据加载函数中的`testData.tsv`是从第一批数据集中拆解出的一部分用作测试。

这部分还需要考虑题目中的特殊情况，即大量评论信息中，可能存在伪好评和恶意差评的情况，所以第一次训练时我们不希望出现过拟合的情况，因为这样会导致词的语义出现偏差，比如说一个好评的词语被联系到差评

```
test_data_features = vectorizer.transform(df.clean_review).toarray()
test_data_features.shape

result = forest.predict(test_data_features) # 预测结果
output = pd.DataFrame({'id': df.id, 'sentiment': result})
print(output.head())
```

这部分进行了二分类，输出文件中以01表示好差评价，列出了产品ID和情感，输出均保存在`output`文件夹下。

A1				
id				
	A	B	C	D
1	id	sentiment		
2	'RY52KZABZK8QF'	0		
3	'R3GCOEV4HYZG2I'	0		
4	'R1V2OPPNLOQGCE'	1		
5	'R9Q0QDTLKV567'	1		
6	'R3DL7HYC3QTWNI'	1		
7	'D2M006707VC6MII'	1		

这部分测试数据是从`labeled`数据中取出的一部分，将评论之前对应的星级评价进行保存，之后将结果与输出的`output`对比，大部分数据情感分析都是正确的，结论是训练的分类模型在这个测试集上表现较好，于是可以将我们需要的`microwave.tsv`替换`testData.tsv`输入到分类器中


```

datafile = os.path.join('..', 'data', 'microwave.tsv')
df = pd.read_csv(datafile, sep='\t', escapechar='\\')

print('Number of reviews: {}'.format(len(df)))
df['clean_review'] = df.review.apply(clean_text)
print(df.head())

test_data_features = vectorizer.transform(df.clean_review).toarray()
test_data_features.shape

result = forest.predict(test_data_features) # 预测结果
output = pd.DataFrame({'id': df.id, 'sentiment': result})
print(output.head())

```

	A	B	C
1	id	sentiment	
2	'RY52KZABZK8QF'	0	
3	'R3GCOEV4HYZG2I'	0	
4	'R1V2OPPNLOQGC'	1	
5	'R9QOQDTLKV567'	1	
6	'R3DL7HYC3QTWNI'	1	
7	'R3M88678ZYC6WI'	1	
8	'R2G20T7N6L3HO1'	0	
9	'R29F0E6EJCEEM'	1	
10	'R1WU4A3QFKJ0HT'	0	
11	'R2Q061UH1EJHI'	1	
12	'R2F0TUS62C3ZCN'	1	
13	'R2X9MNOB0MVUZT'	1	
14	'R35P4D0NHKAGKE'	1	
15	'R2V6PX8BC9ZPKZ'	1	
16	'R1MTXPHNTEOEM'	0	
17	'RR48906IDDC7R'	1	
18	'R3BRGLMPYWF52N'	1	
19	'R1FB7BN6ZOL3XO'	1	
20	'R3U4EMQXXN5ZWA'	0	
21	'RCFH27W77OTPH'	1	
22	'R2R63TFP7G9JL'	1	
23	'RLQM6VC4WYV08'	0	

测试输出结果有将近两千条，由于亚马逊网站上对于微波炉的评论是按星级打分，我们的数据源将其按照星级进行二分，比对 *microwave.tsv* 里评论和输出的分类结果，可以看到 *sentiment* 的结果基本合理。

(4) 接下来的一个任务是判断特定词语是否会对整体评论产生明显影响，这一方面可以验证我们前面训练的模型是否准确，另一方面也能看出判断出模型是否稳定，有的词语可能会对一句话的语义产生明显影响，而有的则影响不大。

以题目中给出的 *enthusiastic* 为例：

```
datafile = os.path.join '..', 'data', 'microwave_withenthushu.tsv')
df = pd.read_csv(datafile, sep='\t', escapechar='\\')

print('Number of reviews: {}'.format(len(df)))
df['clean_review'] = df.review.apply(clean_text)
print(df.head())
```

未拼接单词前好评数量：

output_unchanged
在 1615 条记录中找到 1227 个

拼接*enthusiastic*后好评数量：

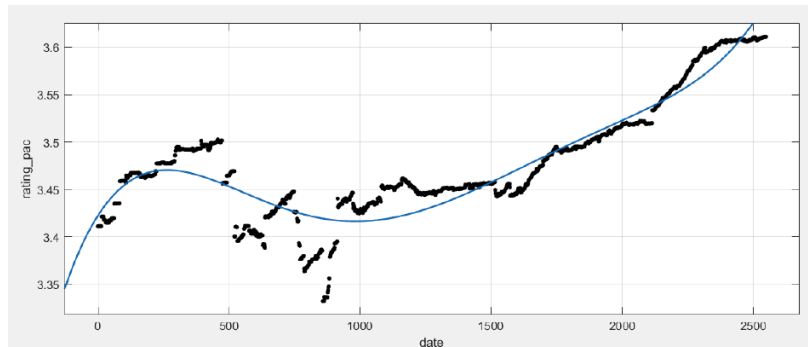
output_changed_enthu
在 1615 条记录中找到 1509 个

将微波炉对应的数据用*excel*进行拼接处理，将单词*enthusiastic*加入句子中，重新用*forest*对处理后的评论进行分类，之后统计这一次中好评的数量为 Rev_i ，并计算相比于之前的评论分类结果，改变的比例记为 G_i ，根据 G_i 的大小去比较我们加入时的期望值（与插入的位置也有关），在这个基础上可以主观性地得出一个结论，表示所测试的单词是否有明显影响，如果有则标记为*valid*。

部分单词测试结果：

word	Rev_i	G_i	position	θ_i	mark
enthusiastic	1509	0.93	rear	26.2	valid
like	1222	0.75	front	3.5	invalid
wonderful	1499	0.93	middle	24.4	valid
favorite	1241	0.77	middle	4.2	invalid
disappointed	701	0.43	front	0.9	valid

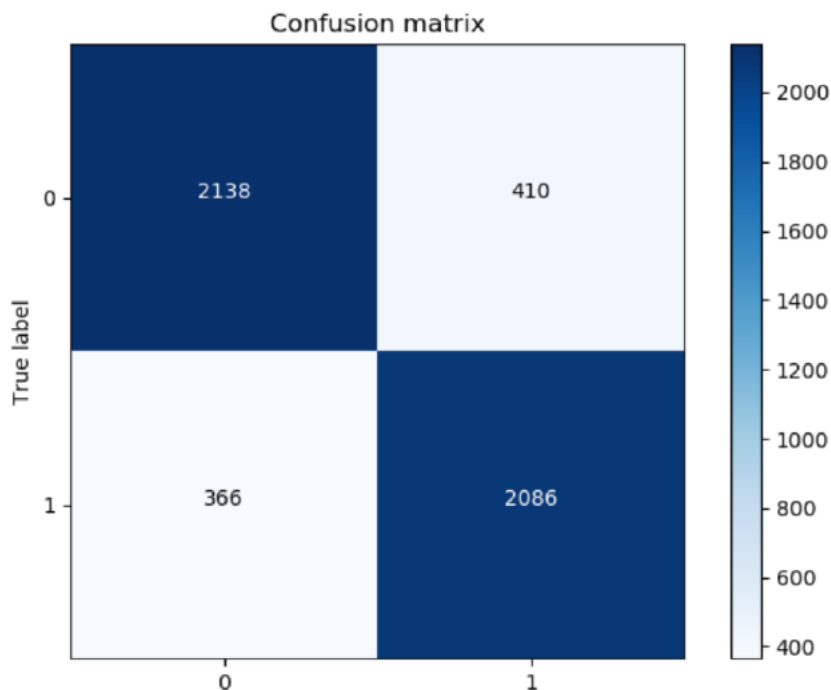
(5) 前面两个文件是拆解出来的两个训练代码，*Merged.py*是最初的完整版代码，其中包括用*matplotlib*库可视化数据的部分



二分类时混淆矩阵绘制：

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
```



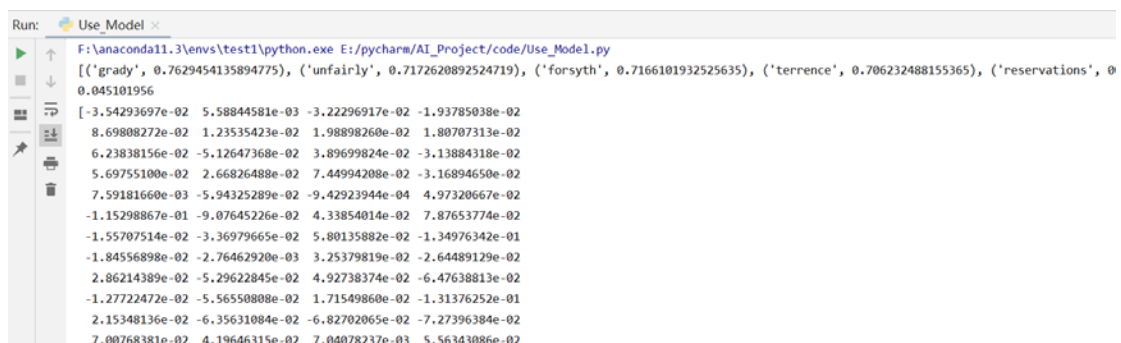
训练得到word2vec词向量库之后，可以直接查看一些单词的关联词

以及它们对应的向量，见`Use_Model.py`

比如输出与"*enthusiastic*"词义相关的评论词，或输出"*enthusiastic*"一词与“产品”是否有关，相关性等。

```
print(model.most_similar("enthusiastic"))#输出这个词相近的词
print(model.similarity("enthusiastic","product"))#输出两个词的相关性
print(model.wv["disappointed"])#输出这个词的词向量
```

首行输出各词按相关性由大到小的顺序，后面的数字表示两个单词的相关性（范围在 0 到 1）；第二行输出的数字表示在我们定义的词向量库中两个评论词的相关性；后面输出的向量表示在库中该代评论词对应的向量值，如下图。



```
Run: Use_Model x
F:\anaconda11.3\envs\test1\python.exe E:\pycharm\AI_Project\code\Use_Model.py
[('grady', 0.7629454135894775), ('unfairly', 0.7172620892524719), ('forsyth', 0.7166101932525635), ('terrence', 0.706232488155365), ('reservations', 0.045101956)
[-3.54293697e-02  5.58844581e-03 -3.22296917e-02 -1.93785038e-02
 8.69808272e-02  1.23535423e-02  1.98898260e-02  1.80707313e-02
 6.23838156e-02 -5.12647368e-02  3.89699824e-02 -3.13884318e-02
 5.69755100e-02  2.66826488e-02  7.44994208e-02 -3.16894650e-02
 7.59181660e-03 -5.94325289e-02 -9.42923944e-04  4.97320667e-02
 -1.15298867e-01 -9.07645226e-02  4.33854014e-02  7.87653774e-02
 -1.55707514e-02 -3.36979665e-02  5.80135882e-02 -1.34976342e-01
 -1.84556898e-02 -2.76462920e-03  3.25379819e-02 -2.64489129e-02
 2.86214389e-02 -5.29622845e-02  4.92738374e-02 -6.47638813e-02
 -1.27722472e-02 -5.56550808e-02  1.71549860e-02 -1.31376252e-01
 2.15348136e-02 -6.35631084e-02 -6.82702065e-02 -7.27396384e-02
 7.00768381e-02  4.19646315e-02  7.04078237e-03  5.56343086e-02]
```

七、总结

回顾整个项目代码，其中`word2vec`词向量库是贯穿整个项目的核心部分，生成词向量的方式也是大多数自然语言处理所采用的；在准备数据方面其实也花了不少时间，可以说`excel`也是数据分析必备的工具了。最后项目代码完整性与预期的功能都达到了，在设题点上也具有性很强的创新性，其中亚马逊市场的分析需求源于美赛原题，也是解决了真实存在的实际问题；整个模型的复杂程度适中，核心处理过程拆解在两个文件中，实现过程参考了网上很多博客的代码实现，

对库函数的使用也查阅了一些官方文档，最终完整的实现了功能。

八、附录

word2vec 参考文档：

<https://radimrehurek.com/gensim/models/word2vec.html>

scikit-learn 参考文档：

<https://scikit-learn.org/dev/modules/ensemble.html>

kaggle 情感分析参考实例：

<https://www.kaggle.com/c/word2vec-nlp-tutorial/data>

word2vec 词向量库博客：

https://blog.csdn.net/hgy413/article/details/81704185?utm_source=blogxgwz2

https://blog.csdn.net/weixin_41370083/article/details/82766030