扒kaggle hotel reviews

笔记本:	computer science		
创建时间:	2019/03/24 16:12	**更新时间:**	2019/03/25 11:17
作者:	陈阳春		
URL:	https://www.kaggleusercontent.com/kf/8589608/eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q...		

load data

- .read_csv() 读数据
- reviews_df["review"] = 新建列
- .apply(lambda x: 1 if x < 5 else 0) 类似def写法, 匿名函数
- reviews_df = reviews_df[["review", "is_bad_review"]]只选新建两列
- reviews_df.head() 读取, 括号内可写几行
- lambda与apply

```
import pandas as pd

# read data
reviews_df = pd.read_csv("../input/Hotel_Reviews.csv")
# append the positive and negative text reviews
reviews_df["review"] = reviews_df["Negative_Review"] + reviews_df["Positive_Review"]
# create the label
reviews_df["is_bad_review"] = reviews_df["Reviewer_Score"].apply(lambda x: 1 if x < 5
else 0)
# select only relevant columns
reviews_df = reviews_df[["review", "is_bad_review"]]
reviews_df.head()
```

2. sample data

```
reviews_df = reviews_df.sample(frac = 0.1, replace = False, random_state=42)
```

不需要用到所有数据, 利用Pandas库中的sample。

DataFrame.sample(n=None, frac=None, replace=False, weights=None, random_state=None, axis=None)

- n是要抽取的行数。（例如n=20000时，抽取其中的2W行）
- frac是抽取的比列。（有一些时候，我们并对具体抽取的行数不关系，我们想抽取其中的百分比，这个时候就可以选择使用frac，例如frac=0.8，就是抽取其中80%）
- replace：是否为有放回抽样，取replace=True时为有放回抽样。
- weights这个是每个样本的权重，具体可以看官方文档说明。
  `str or ndarray-like, optionalDefault ‘None’ results in equal probability weighting. If passed a Series, will align with target object on index. Index values in weights`

```
not found in sampled object will be ignored and index values in sampled object not in
weights will be assigned weights of zero. If called on a DataFrame, will accept the
name of a column when axis = 0. Unless weights are a Series, weights must be same
length as axis being sampled. If weights do not sum to 1, they will be normalized to
sum to 1. Missing values in the weights column will be treated as zero. Infinite
values not allowed.
```

- random_state
  random_state 是用来复现结果的 int or numpy.random.RandomState, optionalSeed for the random number generator (if int), or numpy RandomState object.
- axis是选择抽取数据的行还是列。axis=0的时是抽取行，axis=1时是抽取列（也就是说axis=1时，在列中随机抽取n列，在axis=0时，在行中随机抽取n行）

3. clean data

- .replace(被换, 换)
- 数据里空值会显示 no Neg.. no Posi...
- import 语法

import numpy
import numpy as up
from 模块 import XX函数或命令空间
form 模块 import * (所有的)

```
# remove 'No Negative' or 'No Positive' from text
reviews_df["review"] = reviews_df["review"].apply(lambda x: x.replace("No Negative",
"").replace("No Positive", ""))
```

- 自然语言处理,

```


# return the wordnet object value corresponding to the POS tag
from nltk.corpus import wordnet
# 这段不是很懂
def get_wordnet_pos(pos_tag):
    if pos_tag.startswith('J'):
        return wordnet.ADJ
    elif pos_tag.startswith('V'):
        return wordnet.VERB
    elif pos_tag.startswith('N'):
        return wordnet.NOUN
    elif pos_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

import string
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.tokenize import WhitespaceTokenizer
from nltk.stem import WordNetLemmatizer
```

```
def clean_text(text):
    # lower text小写
    text = text.lower()
    # tokenize text and remove puncutation去标点
    text = [word.strip(string.punctuation) for word in text.split(" ")]
    # remove words that contain numbers去数字
    text = [word for word in text if not any(c.isdigit() for c in word)]
    # remove stop words 去敏感词?
    stop = stopwords.words('english')
    text = [x for x in text if x not in stop]
    # remove empty tokens 去空值
    text = [t for t in text if len(t) > 0]
    # pos tag text 分词性
    pos_tags = pos_tag(text)
    # lemmatize text 去掉s, ing之类(词性还原)
    text = [WordNetLemmatizer().lemmatize(t[0], get_wordnet_pos(t[1])) for t in 
pos_tags]
    # remove words with only one letter
    text = [t for t in text if len(t) > 1]
    # join all 用空格连在一起
    text = " ".join(text)
    return(text)

# clean text data
reviews_df["review_clean"] = reviews_df["review"].apply(lambda x: clean_text(x))
```

4. Feature engineering

1. 情感分析 .polarity_scores(x) 这个函数, 直接用

```
# add sentiment anaylsis columns
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()
reviews_df["sentiments"] = reviews_df["review"].apply(lambda x: 
sid.polarity_scores(x))
reviews_df = pd.concat([reviews_df.drop(['sentiments'], axis=1), 
reviews_df['sentiments'].apply(pd.Series)], axis=1)
```

- 输出4个值
-
    - a neutrality scorea

-
  - positivity scorea
-
  - negativity scorean
-
  - overall score that summarizes the previous scores

2. 计算句子词数与词的长度

-
  - number of characters in the text
-
  - number of words in the text

```
# add number of characters column
reviews_df["nb_chars"] = reviews_df["review"].apply(lambda x: len(x))

# add number of words column
reviews_df["nb_words"] = reviews_df["review"].apply(lambda x: len(x.split(" ")))
```

3. Doc2vec

- Doc2Vec(documents, vector_size=5, window=2, min_count=1, workers=4)内维度含义不明, 但越多肯定包含信息越多, 数据量不大就好不要做太多维度, 计算量大并且过于分散
-
  - 理解链接1, stackflow
-
  - 官方手册例子, 参数设置同本代码
-
  - 官方手册tutorial
-
  -
    - 官方tutorial解释
-
  -
    - epochs=40遍历训练40次, 最小字数2为去掉出现少的词, 大的数据量遍历越多次花时间, 而且最后到收益递减点, 小的数据遍历多次理论上可以更准确的
      Now, we'll instantiate a Doc2Vec model with a vector size with 50 words and iterating over the training corpus 40 times. We set the minimum word count to 2 in order to discard words with very few occurrences. (Without a variety of representative examples, retaining such infrequent words can often make a model worse!) Typical iteration counts in published 'Paragraph Vectors' results, using 10s-of-thousands to millions of docs, are 10-20. More iterations take more time and eventually reach a point of diminishing returns. However, this is a very very small dataset (300 documents) with shortish documents (a few hundred words). Adding training passes can sometimes help with such small datasets.

```
model = gensim.models.doc2vec.Doc2Vec(vector_size=50, min_count=2, epochs=40)
```

- enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。
- .series 创建数组
- .infer_vector 推断向量
- .column()新加列?
- .concat和并表

```
# create doc2vec vector columns
from gensim.test.utils import common_texts
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

documents = [TaggedDocument(doc, [i]) for i, doc in
enumerate(reviews_df["review_clean"].apply(lambda x: x.split(" ")))]

# train a Doc2Vec model with our text data
model = Doc2Vec(documents, vector_size=5, window=2, min_count=1, workers=4)

# transform each document into a vector data
doc2vec_df = reviews_df["review_clean"].apply(lambda x: model.infer_vector(x.split("
"))).apply(pd.Series)
doc2vec_df.columns = ["doc2vec_vector_" + str(x) for x in doc2vec_df.columns]
reviews_df = pd.concat([reviews_df, doc2vec_df], axis=1)
```

4. 词频与反向词频

并不直接计算词频, 因为词频并不表示重要性
TF计算单词在文本中出现的经典次数
IDF计算这个单词的相对重要性，这取决于可以找到单词的文本数量

- .shape()
-
    - 与Numpy一样，用shape属性来显示数据的格式

```
dimensions = food_info.shape

print(dimensions)
# 输出：(8618,36) 表示这个表格有8618行和36列的数据，其中dimensions[0]为8618,
dimensions[1]为36
```

- reshape（行，列）可以根据指定的数值将数据转换为特定的行数和列数。
- 本项目实际代码

```
# add tf-idfs columns
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(min_df = 10)
tfidf_result = tfidf.fit_transform(reviews_df["review_clean"]).toarray()
tfidf_df = pd.DataFrame(tfidf_result, columns = tfidf.get_feature_names())
```

```
tfidf_df.columns = ["word_" + str(x) for x in tfidf_df.columns]
tfidf_df.index = reviews_df.index
reviews_df = pd.concat([reviews_df, tfidf_df], axis=1)
```

Exploratory data analysis