目录

W1_138王泽群

1 720 1

TAIL好未来AI实战训练营-自然语言处理方向 团队NLP-Noob

138-王泽群-NLP LV2 1

Fork 点赞

内容 数据集 Fork记录 1 评论

内容

版本 2 2018-02-02 22:41

目录

写在前面的话

这是我第一次接触nlp，也是第一次参加比赛。比赛的过程非常惊心动魄，感谢主办方的支持。 但是因为没有经验，NOTEBOOK写得很乱，而且还分了两个。我尽量把notebook合在一起，但是部分cell可能没有办法给出OUTPUT.

数据处理流程

In [3]:

```
# 加载数据分析常用库
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
import re
import nltk
# data= pd.read_pickle('data131-2141.pic')

% matplotlib inline
```

In [718]:

```
test = pd.read_table('k-lab-data/test_ai-lab.txt', names=['id','s1','s2'])
test.head()
```

Out[718]:

In [719]:

```
train = pd.read_table('k-lab-data/train_ai-lab.txt', names=['id','s1','s2','sim'])
train.head()
```

Out[719]:

正常地对句子进行预处理

pos tagging/tokenizing/lemmatizing

```
data = pd.concat([train, test]).set_index('id')
data['pos1'] = data['s1'].apply(lambda x:nltk.pos_tag(nltk.word_tokenize(x)))
data['pos2'] = data['s2'].apply(lambda x:nltk.pos_tag(nltk.word_tokenize(x)))
data['v1'] = data['pos1'].apply(lambda x:[i[0] for i in x if i[1].startswith('V')])
data['v2'] = data['pos2'].apply(lambda x:[i[0] for i in x if i[1].startswith('V')])
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize('working', pos='v'))
data['v1_lem'] = data['v1'].apply(lambda x:[lemmatizer.lemmatize(i, pos='v') for i in x])
data['v2_lem'] = data['v2'].apply(lambda x:[lemmatizer.lemmatize(i, pos='v') for i in x])
```

这个时候发现，v1_lem这一列把be作为动词也统计进去了，写一个函数去掉'be'这个字

```
def remove_be(v_lst):
    if 'be' in v_lst:
        v_lst.remove('be')
    return v_lst
data['v1_lem'] = data['v1_lem'].apply(lambda x:remove_be(x))
data['v2_lem'] = data['v2_lem'].apply(lambda x:remove_be(x))
```

stemming

一开始并不知道lemmatizing和stemming的用法，事实上到现在也不知道stemming有什么用。因为事实上stemming之后的某些词，在训练的word2vec模型中是没有的。 所以虽然刚开始对句子的所有词进行了stemming，但是最后却没有用到这一列。

```
from nltk.stem import PorterStemmer
pst = PorterStemmer()
data['v1_stem'] = data['v1'].apply(lambda x:[pst.stem(i) for i in x])
data['v2_stem'] = data['v2'].apply(lambda x:[pst.stem(i) for i in x])

data.head()
```

目录

parsing

我在讨论区发过一片帖子 https://www.kesci.com/apps/home/competition/forum/5a7267f2fd64f22bb119910c (https://www.kesci.com/apps/home/competition/forum/5a7267f2fd64f22bb119910c) 写了我在parsing过程中的心路历程。简单说来就是安装可以对句子进行语块分解的包的难度实在太大，所以尝试着去找api,最后却只找到了在线的解析器。 运用了一些简单的爬虫技巧，可以获得这个内容。但是因为获得的字符串（树状）解析起来太过困难，最后无奈放弃。

```
def get_parse(sentence):
    res = requests.post('http://nlp.stanford.edu:8080/parser/index.jsp',data={'query':sentence})
    parsed = re.findall('<pre id="parse" class="spacingFree">(.*?)</pre>',res.text, re.S)
    # print(parsed)
    return parsed

data['s1_parse'] = data['s1'].apply(lambda x:get_parse(x))
```

可以得到类似

```
['(ROOT\n  (NP\n    (NP (CD two) (JJ big) (JJ brown) (NNS dogs))\n    (VP (VBG running)\n      (PP (IN through)\n
['(ROOT\n  (S\n    (NP (DT A) (NN woman))\n    (VP (VBZ is)\n      (VP (VBG peeling)\n        (NP (DT a) (NN potatc
['(ROOT\n  (S\n    (NP (DT A) (JJ small) (NN boy))\n    (VP (VBZ is)\n      (VP (VBG building)\n        (PP (IN wit
['(ROOT\n  (S\n    (NP (NNP Soccer) (NNS players))\n    (VP (VBP are)\n
```

的字符串。

个人的标注尝试

因为无奈，又因为在这方面因为没有什么经验，就参考了第二天买的一本NLTK Essentials的书上的ChunkRule，对名词短语和动词短语进行标注。

```
from nltk.chunk.regexp import *
import re
def get_np(pos):
    rule_np = ChunkRule(r'(<DT>?<RB>?)<JJ|CD>*(<JJ|CD><,>)*(<NN.*>)+', 'Chunk NPs')
    parser_np = RegexpChunkParser([rule_np], chunk_label='NP')
    raw_phrase = [i[3:-1] for i in re.findall(r'NP .*?:?\)', str(parser_np.parse(pos)))]
    words = [[j.split('/')[0] for j in i.split()] for i in raw_phrase]
    phrase = [' '.join(i) for i in words]
    return phrase

def get_vp(pos):
    rule_vp = ChunkRule(r'(<VB.*>)?(<VB.*>)+(<PRP>)?', 'Chunk VPs')
    parser_vp = RegexpChunkParser([rule_vp], chunk_label='VP')
    raw_phrase = [i[3:-1] for i in re.findall(r'VP .*?:?\)', str(parser_vp.parse(pos)))]
    words = [[j.split('/')[0] for j in i.split()] for i in raw_phrase]
    phrase = [' '.join(i) for i in words]
    return phrase

data['np1'] = data['pos1'].apply(lambda x:get_np(x))
data['np2'] = data['pos2'].apply(lambda x:get_np(x))
data['vp1'] = data['pos1'].apply(lambda x:get_vp(x))
data['vp2'] = data['pos2'].apply(lambda x:get_vp(x))
data['S1'] = data['np1'].apply(lambda x:x[0])
data['S2'] = data['np2'].apply(lambda x:x[0])
data['bin1'] = data['np1'].apply(lambda x:x[1:])
data['bin2'] = data['np2'].apply(lambda x:x[1:])
```

但是后期应用时发现，这样标注并不很准确。最后使用的的其实是第一个动词之前的内容作为所谓的"subject"，最后一个动词后内容作为所谓的'object'，然后对各个语块进行解析（事后发现动词后的内容（主要是宾语部分）对于相似度来说非常重要。）

```
from nltk.corpus import stopwords
stoplist = stopwords.words('english')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

def get_subject(sentence,v):
    tokens = sentence.split(' ')
```

目录

```
    v_index = 0
    for i,word in enumerate(tokens):
        if v:
            if word == v[0]:
                v_index = i
    bag = [tokens[i].lower() for i in range(v_index) if tokens[i].lower() in model if tokens[i].lower() not in stop
    return bag

def get_object(sentence,v):
    tokens = sentence.split(' ')
    v_index = 0
    for i,word in enumerate(tokens):
        if v:
            if word == v[-1]:
                v_index = i
    bag = [tokens[i].lower().strip('.') for i in range(v_index+1, len(tokens)) if tokens[i].lower().strip('.') in m
    return bag

data['sub_bag1'] = data.apply(lambda x:get_subject(x['s1'], x['v1']), axis=1)
data['sub_bag2'] = data.apply(lambda x:get_subject(x['s2'], x['v2']), axis=1)
data['ob_bag1'] = data.apply(lambda x:get_object(x['s1'], x['v1']), axis=1)
data['ob_bag2'] = data.apply(lambda x:get_object(x['s2'], x['v2']), axis=1)
```

训练Word2Vec词向量

为了使用词向量模型，一开始使用了爬虫爬了维基百科的一些页面，后来发现解析起来也是非常困难，而且其实维基百科是提供下载的。就下载了大约700M的文章，训练了一个1G左右的词向量模型。因为这个模型太大没办法上传上去，所以就选择了在本地做这个项目。

```
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence


sentences = LineSentence('enwiki2.txt')
model = Word2Vec(sentences)
model.wv.save_word2vec_format('final_model.vector')
```

整理词袋

整理词袋的过程中，并不知道到底袋子里装什么比较合适，所以把袋子分为了两部分。在去除了停用词之后，一个袋子装所有小写的句子中的词，另外一个袋子装词汇的原型。这就是data['bag1']和data['bag1_lem']。当然因为词袋是为了模型使用的，所以检查了每一个单词是否在w2v model里面。

```
from nltk.corpus import stopwords
stoplist = stopwords.words('english')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def get_word_bag_lem(sentence):
    bag = [word.lower().strip('.') for word in sentence.split(' ') if word.lower() not in stoplist if word.lower()
    return [lemmatizer.lemmatize(word) for word in bag if lemmatizer.lemmatize(word) in model]
def get_word_bag(sentence):
    bag = [word.lower().strip('.') for word in sentence.split(' ') if word.lower() not in stoplist if word.lower()
    return [word for word in bag if word in model]

data['bag1'] = data.apply(lambda x:get_word_bag(x['s1']), axis=1)
data['bag2'] = data.apply(lambda x:get_word_bag(x['s2']), axis=1)
data['bag1_lem'] = data.apply(lambda x:get_word_bag_lem(x['s1']), axis=1)
data['bag2_lem'] = data.apply(lambda x:get_word_bag_lem(x['s2']), axis=1)


data.to_pickle('data131-1824.pic')
```

特征提取

特征提取

第一次接触NLP，也第一次参加比赛，很大的失误在于没有明白词频的重要性。看到大佬光是用词频就拿到了0.8，这个必须要学习一番，争取后面的比赛可以用得到。还有tf-idf这样的内容也完全没有使用。

目录

编辑距离

第一天因为实在是不懂，所以选择了主语位于编辑距离作为特征。参考了网上对编辑距离的算法。获得了0.3左右的准确度。（还挂在排行榜的最后一名一整天T T） 因为最后drop掉了这个特征，代码也修改得面目全非找不到了。在这就不做记录了

词袋所代表的向量

第二次尝试是使用了每一个词袋所代表的向量。具体算法是用$\frac{\Sigma V_i}{n}$，获得每一个词袋的平均向量。 这两个平均向量可以用一系列的相关系数来描述。而对于两个句子而言，我们也可以寻找他们所对应的向量差的特征。是一个相当浅显的特征，但是效果很明显，上升到了0.5左右。

```
from gensim.models import Word2Vec
import gensim
model = gensim.models.KeyedVectors.load_word2vec_format('finalmodel.vector')
def vec_mean(bag):
    """
    返回向量和的平均值
    """
    # 建立一个全是0的array
    res =np.zeros([256])
    count = 0 # 记录词汇数
    for word in bag:
        res += model[word]  # 在模型中查找单词所对应的向量
        count += 1
    return res/count

def vec_diff(bag1, bag2):
    return vec_mean(bag1)-vec_mean(bag2)
# data['sub_edit_d'] = data.apply(lambda x:edit_distance(x['S1'], x['S2']), axis=1)
data['bag1_vec'] = data['bag1'].apply(lambda x:vec_mean(x))
data['bag2_vec'] = data['bag2'].apply(lambda x:vec_mean(x))
data['bag1_lem_vec'] = data['bag1_lem'].apply(lambda x:vec_mean(x))
data['bag2_lem_vec'] = data['bag2_lem'].apply(lambda x:vec_mean(x))

# 计算两个向量的差

data['bag_diff'] = data['bag1_vec'] - data['bag2_vec']
data['bag_lem_diff'] = data['bag1_lem_vec'] - data['bag2_lem_vec']
```

In [54]:

```
def get_bag_vec(bag):
    if bag==[]:
        print('no bag')
        return np.zeros((13,256))
    vec = np.zeros((13,256))

    for index, word in enumerate(bag):
        vec[index] = model[word]
    return vec

data['s1_bag_vec'] = data['bag1'].apply(lambda x:get_bag_vec(x))
data['s2_bag_vec'] = data['bag2'].apply(lambda x:get_bag_vec(x))
```

```
no bag
```

还尝试着直接把这个向量差作为特征，直接送到模型里去拟合。这也是那个0.58的来源。我还是把这个这么多的特征给drop掉了。

In [364]:

```
for i in range(256):
    data[i] = data['bag_lem_diff'].apply(lambda x:x[i])
# data.info()
```

开启疯狂构建特征模式

这里是我作为新手不太懂的地方。我不知道需要构建什么样的特征，也不知道怎么去构建特征。一开始没有阅读论文的时候，在网上查阅很多资料，也自己

“发明”了很多新的（没有用的）特征。

最后被放弃掉的主谓宾的相似度

这一次的函数构造的就非常差劲，而且本身因为使用的是最开始的正则语块分解，成功率堪忧。最后被放弃掉了。这部分特征最后得到了0.58左右的得分。

```
def word_sim(w_lst1, w_lst2):
    sim = []
    for i in w_lst1:
        for j in w_lst2:
            try:
                sim.append(model.similarity(i,j))
            except:
                pass
    if sim:
        if np.mean(sim)>0.5:
            return max(sim)
        else:
            return np.mean(sim)
    else:
        return 0
```

```
data['v_sim'] = data.apply(lambda x:word_sim(x['v1_lem'], x['v2_lem']),axis=1)
data['S1_rs'] = data['S1'].apply(lambda x:remove_stop(x))
data['S2_rs'] = data['S2'].apply(lambda x:remove_stop(x))
data['bin1_rs'] = data['bin1'].apply(lambda x:remove_stop(x))
data['bin2_rs'] = data['bin2'].apply(lambda x:remove_stop(x))
data['s_sim'] = data.apply(lambda x:word_sim(x['S1_rs'],x['S2_rs']), axis=1)
data['bin_sim'] = data.apply(lambda x:word_sim(x['bin1_rs'],x['bin2_rs']), axis=1)
```

又一次主谓宾相似度的尝试

上面说到了因为语块分解的问题，生成的特征不是特别好。这次完全放弃了语块分解（上面也说到了这次分解的方式），采用动词前词袋、动词词袋、动词后词袋这样的分解方式。然后根据gensim给出的model.n_similarity(a, b)直接计算每一个词袋的相似度，放弃了自己构建函数的想法。 在最后的随机森林特征选择中，发现宾语（data['o_sim']）所占的权重非常大，达到了30%~40%之间。说明对于这次训练集来说，动词后面的（姑且看作）宾语的相似程度，大大影响了整个句子的相似程度。而主语所占的比重非常小，因为大部分两个句子的主语都很一致。动词也有一些权重，但是和宾语比起来差远了。

```
count = 0
def n_similarity(a,b):
    global count
    if not a or not b:
        return 1
    else:
        try:
            return model.n_similarity(a,b)
        except KeyError:
            count += 1
            return 0.5
```

```
data['s_sim'] = data.apply(lambda x:n_similarity(x['sub_bag1'],x['sub_bag2']), axis=1)
data['o_sim'] = data.apply(lambda x:n_similarity(x['ob_bag1'],x['ob_bag2']), axis=1)
data['v_sim'] = data.apply(lambda x:n_similarity(x['v1_lem'],x['v2_lem']), axis=1)
data['n_sim'] = data.apply(lambda x:n_similarity(x['bag1'],x['bag2']), axis=1)
```

作死的LSTM

作为一个新手玩家，去尝试深度学习（咱们不说w2v，这个调包太简单了）本身就是一个找打的想法。我看了一整天LSTM是个啥，又看了半天的论文，大致明白了那篇论文 http://www.mit.edu/~jonasm/info/MuellerThyagarajan_AAAI16.pdf (http://www.mit.edu/~jonasm/info/MuellerThyagarajan_AAAI16.pdf) 的大致想法。 可是因为keras的api也不太懂，也没怎么接触过深度学习的图的搭建，所以参考了网上很多人的代码，跑了无数遍，见了不知道多少个error，最后跑出来的结果惨不忍睹。 后来我也放弃了直接拟合这个想法。但是考虑了一下，又加回来了这部分：我可以把评分分为positive和negative，2.5以上和2.5以下用1和0表示，用LSTM来拟合这部分。然后再把它作为特征加进去。最后的结果好像有些用处，但是感觉有点循环论证的嫌疑。而且最后我也没学会怎么在融合的过程中使用曼哈顿距离这种二元的函数。Anyway，算是一次失败的尝试，但是也看了一整天的LSTM和一整夜的keras文档，姑且算是学到了点东西吧……

In [835]:

```
import keras
from keras.models import Model, Input
from keras.layers import merge, Embedding,Dropout, BatchNormalization,Dense, Activation
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import EarlyStopping,ModelCheckpoint
import keras.backend as K
from keras.optimizers import Adadelta

from keras.layers.core import Activation, Dense
from keras.layers.recurrent import LSTM
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
```

In [527]:

```
def positive_negetive(sim):
    if sim >= 2.5:
        return 1
    else:
        return 0

data['p_n'] = data.apply(lambda x:positive_negetive(x['sim']), axis=1)
data.to_pickle('data-02020133.pic')
```

In [823]:

```
list_word=[]
dict_word={}
for column in [data['s1'],data['s2']]:
    for sent in column:
        words = [i.strip('.').lower() for i in sent.split(' ')]
        for i,word in enumerate(words):
            if word not in dict_word:
                dict_word[word]=len(dict_word)
```

In [824]:

```
id2w={dict_word[w]:w for w in dict_word}
```

In [825]:

```
embedding_size=256
embedding_arry=np.random.randn(len(dict_word)+1,embedding_size)#句子embedding矩阵
embedding_arry[0]=0
```

In [827]:

```
for index,word in enumerate(dict_word):
    if word in model.wv.vocab:
        embedding_arry[index]=model.wv.word_vec(word)
# embedding_arry[0]
```

In [831]:

```
# del model
def get_id(sent):
    words = [i.strip('.').lower() for i in sent.split(' ')]
    return pad_sequences([[dict_word[word] for word in words if word in model]], maxlen=13)
data['id1'] = data['s1'].apply(lambda x:get_id(x))
data['id2'] = data['s2'].apply(lambda x:get_id(x))
# data.head()
# data.to_pickle('data-2012200.pic')
# !要不要lemmatization
```

In [236]:

```
# data = pd.read_pickle('data2012019')
```

In [837]:

```
tain_data_l_n = np.asarray(data.loc[train.id,'id1'])
tain_data_r_n = np.asarray(data.loc[train.id,'id2'])
tain_data_l_n = np.asarray([i[0] for i in tain_data_l_n])
tain_data_r_n = np.asarray([i[0] for i in tain_data_r_n])
# data
tain_data_r_n.shape
# max_length
```

Out[837]:

In [910]:

```
def get_model():
    embedding_layer = Embedding(len(embedding_arry),
                                256,
```

```
                                    weights=[embedding_arry],
                                    input_length=13,
                                    trainable=False)
    lstm_layer = LSTM(128,
#                       dropout=0.1,
                      recurrent_dropout=0.1)
#     lstm_layer = LSTM(num_lstm, dropout=rate_drop_lstm, recurrent_dropout=rate_drop_lstm)

    sequence_1_input = Input(shape=(13,), dtype='int32')
    embedded_sequences_1 = embedding_layer(sequence_1_input)
    y1 = lstm_layer(embedded_sequences_1)

    sequence_2_input = Input(shape=(13,), dtype='int32')
    embedded_sequences_2 = embedding_layer(sequence_2_input)
    y2 = lstm_layer(embedded_sequences_2)

    merged = keras.layers.concatenate([y1, y2])
#     merged = Dropout(rate_drop_dense)(merged)
    merged = BatchNormalization()(merged)

    merged = Dense(128, activation='relu')(merged)
#     merged = Dropout(rate_drop_dense)(merged)
    merged = BatchNormalization()(merged)
    preds = Dense(1, activation='sigmoid')(merged)

    model = Model(inputs=[sequence_1_input, sequence_2_input], \
                  outputs=preds)
    model.compile(loss='mse',
                  optimizer='nadam',
                  metrics=['acc'])
    return model
```

In [911]:

```
y_data_n = np.array(data['p_n'])
```

In [912]:

```
model_lstm = get_model()
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
STAMP = "rnn_mode_bs64"
bst_model_path = STAMP + '.h5'
model_checkpoint = ModelCheckpoint(bst_model_path, save_best_only=True, save_weights_only=True)

hist = model_lstm.fit([np.asarray([tain_data_l_n[i] for i in train1_id]), np.asarray([tain_data_r_n[i] for i in tr
                  validation_data=([np.asarray([tain_data_l_n[i] for i in train2_id]), np.asarray([tain_data_r_n[i]
                  epochs=200, batch_size=10, shuffle=True,
                   callbacks=[early_stopping, model_checkpoint])
```

```
Train on 500 samples, validate on 500 samples
Epoch 1/200
500/500 [==============================] - 5s - loss: 0.3170 - acc: 0.5440 - val_loss: 0.2580 - val_acc: 0.5340
Epoch 2/200
500/500 [==============================] - 2s - loss: 0.1856 - acc: 0.7460 - val_loss: 0.2424 - val_acc: 0.5800
Epoch 3/200
500/500 [==============================] - 2s - loss: 0.1277 - acc: 0.8160 - val_loss: 0.2543 - val_acc: 0.5900
Epoch 4/200
500/500 [==============================] - 2s - loss: 0.1000 - acc: 0.8620 - val_loss: 0.2736 - val_acc: 0.5820
Epoch 5/200
500/500 [==============================] - 2s - loss: 0.0859 - acc: 0.8940 - val_loss: 0.2867 - val_acc: 0.5760
Epoch 6/200
500/500 [==============================] - 3s - loss: 0.0625 - acc: 0.9220 - val_loss: 0.3072 - val_acc: 0.5720
Epoch 7/200
500/500 [==============================] - 3s - loss: 0.0747 - acc: 0.8960 - val_loss: 0.3371 - val_acc: 0.5460
Epoch 8/200
500/500 [==============================] - 3s - loss: 0.0505 - acc: 0.9320 - val_loss: 0.3329 - val_acc: 0.5420
```

In [914]:

```
preds = []
for i in data.iterrows():
    preds.append(model_lstm.predict([i[1]['id1'],i[1]['id2']]))
```

In [915]:

```
(data['p_n']-[i[0][0] for i in preds]).describe()
```

Out[915]:

队友发现的在知乎上找到的一种 improved bow & w2v（姑且这么理解吧）

队友神助攻（虽然最后这个特征可能我没怎么会用），发现了一个花式的特征构建方式： 取一个适当大小n的W2V模型，对于模型中的每一个单词w，对应句子中的每一个单词产生句子单词数m个相似度，取这些相似度的最大值，作为这个w和这个句子的相似度。也就意味着，每一个句子，可以构建一个n维的向

量。知乎上的意思是10W维左右，但是我的电脑实在是扛不住这么大的计算量。而且训练集并不大所以我觉着这个维度太大也没啥用（不知道对不对，没经验）。

接下来就直接去尝试着写这个代码了。然后发现，跑了一个小时完全没动静，是一列句子都没跑完。我就考虑了一下是不是模型太大了。。经过了N个小时的努力（说实话感觉努力这N个小时可能都跑完了），我决定换个方式来： 不用整个模型的内容，只关注每一个给定的数据集中的单词在模型中的15个最相似的单词，构造一个模型单词集合。我给起了名字叫做映射集。

目录

In [1162]:

```
vocab_given = set() #用于装平台给定的数据集单词的set
for i in data['bag1']:
    for j in i:
        vocab_given.add(j)
for i in data['bag2']:
    for j in i:
        vocab_given.add(j)

# 对于平台给的每一个单词，获得W2V模型中和它最相近的15个词作为映射单词的基准
project_word_base = [j[0] for i in vocab_given for j in model.most_similar(i,topn=15)]
project_word_base = set(project_word_base)
# 将原本词袋中的单词也装进去
for i in data['bag1']:
    for j in i:
        project_word_base.add(j)
for i in data['bag2']:
    for j in i:
        project_word_base.add(j)
len(project_word_base)
```

Out[1162]:

接着构建一个大矩阵，len(数据集)行， len(映射集)列，因为我不关注模型里面的某一个词对给定的数据集之外的单词的相似度。 因为numpy.ndarray只支持整数索引，所以必须要对数据集和映射集进行编号。 然后根据标号（比如(4,6)）就能找到数据集第4个字对应映射集第6个字的相似度。

In [730]:

```
# 构建一个数据集中的词-编号的字典
dict_data_word={}
# 给vocab_given里面的每一个词编号
for i, word in enumerate(vocab_given):
    if word not in dict_data_word:
        dict_data_word[word] = len(dict_data_word)

# 根据编号反查
id_data_word = {v:k for k,v in dict_data_word.items()}


# 构建一个映射集中的词-编号的字典
dict_project_word={}
# 给project_word_base里面的每一个词编号
for i,word in enumerate(project_word_base):
    if word not in dict_project_word:
        dict_project_word[word]=len(dict_project_word)

# 根据编号反查
id_project_word = {v:k for k,v in dict_project_word.items()}


# 一个空矩阵，len(数据集)行， len(映射集)列
project_matrix = np.zeros((len(dict_data_word),len(dict_project_word)))
project_matrix.shape


# 对于数据集中的每一个单词，构建它所对应的 len(映射集) 维向量
for i in range(len(dict_data_word)):
    for j in range(len(dict_project_word)):
        project_matrix[i][j] = model.similarity(id_data_word[i],id_project_word[j])


# 至此，数据集和映射集的相似度对应矩阵完成
```

In [1198]:

```
project_matrix.tofile('project_matrix')
```

In [1199]:

```
del model
```

删除模型是因为模型太占内存了.....接下来编写函数，对每一个词袋应用函数。

In [1213]:

目录

```
# 对每一个映射集中的单词，返回一个与句子中词相关性最大的similarity
length = len(project_word_base)
project_word_base = list(project_word_base)
def get_projection_vec(bag):
    vec = np.zeros(length)
    if not bag:
        return vec
    for index, project_word in enumerate(project_word_base):
        vec[index] = max([project_matrix[dict_data_word[word]][dict_project_word[project_word]] for word in bag])
    return vec
```

In [1214]:

```
# 对每一个句子的词袋应用该函数
data['pro_vec1'] = data['bag1'].apply(get_projection_vec)
data['pro_vec2'] = data['bag2'].apply(get_projection_vec)
```

得到了这个向量海之后，其实我们可以做很多事情了。但是因为我只会这几种相似度的计算。。所以就只用了这些相似度。 最后其实余弦相似度在随机森林里面所占的比重非常小，一度把它抛弃掉了，而皮尔逊相关系数所占的比重就会相对大一些。

In [1216]:

```
import numpy as np
from numpy import linalg as la

#欧式距离
def euclidSimilar(inA,inB):
    return 1.0/(1.0+la.norm(inA-inB))
#皮尔逊相关系数
def pearsonSimilar(inA,inB):
    if len(inA)<3:
        return 1.0
    return 0.5+0.5*np.corrcoef(inA,inB,rowvar=0)[0][1]
#余弦相似度
def cosSimilar(inA,inB):
    inA=np.mat(inA)
    inB=np.mat(inB)
    num=float(inA*inB.T)
    denom=la.norm(inA)*la.norm(inB)
    return 0.5+0.5*(num/denom)
```

In [1218]:

```
data['pro_euc_sim'] = data.apply(lambda x:euclidSimilar(x['pro_vec1'],x['pro_vec2']), axis=1)
```

In [1220]:

```
data['pro_cos_sim'] = data.apply(lambda x:cosSimilar(x['pro_vec1'],x['pro_vec2']), axis=1)
```

```
/Users/a/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:18: RuntimeWarning: invalid value encountered
```

顺便也把这个函数应用在了词袋所对应的平均向量上。

In [1237]:

```
data['vec_pear_sim'] = data.apply(lambda x:pearsonSimilar(x['bag1_vec'],x['bag2_vec']), axis=1)
```

In [1223]:

```
# data.to_pickle('data-02021406.pic')
```

In [1219]:

```
data['pro_pear_sim'] = data.apply(lambda x:pearsonSimilar(x['pro_vec1'],x['pro_vec2']), axis=1)
```

```
/Users/a/anaconda3/lib/python3.6/site-packages/numpy/lib/function_base.py:3162: RuntimeWarning: invalid value encour
  c /= stddev[:, None]
/Users/a/anaconda3/lib/python3.6/site-packages/numpy/lib/function_base.py:3163: RuntimeWarning: invalid value encour
  c /= stddev[None, :]
```

最长相同词性子序列

很惭愧，今天才开始看论文，大部分都看不明白……但是SemEval025这篇文章中提到的POS Tags Longest Common Subsequence我尝试了一下。 正好上个月在刷数据结构和算法的Mooc，跟着C++写了一个Python版的LCS在这正好用上了。 We measure the length of the longest common subsequence of POS tags between sentence pairs. Additionally, we also average this length by dividing it by the total number of tokens in each sentence separately. 最后这个特征占了2.8%的权重。

In [282]:

```
from functools import wraps
```

目录

```
def memo(func):
    cache = {}
    @wraps(func)
    def wrapper(*args):
        if args not in cache:
            cache[args] = func(*args)
        return cache[args]
    return wrapper

@memo
def LCS(l1, l2):
    """
    寻找l1(0,n) l2(0,m)的最长子序列。


    """
    if len(l1) == 0 or len(l2) == 0:
#        print('递归机')
        return ''
    elif l1[0] == l2[0]:
#        print('找到一致', l1[-1])
        return l1[0] + LCS(l1[1:], l2[1:])
    else:
#        print('找到不一致')
        left = LCS(l1[1:], l2)
        right = LCS(l1, l2[1:])
        if len(left) >= len(right):
#            这里会出问题:
#            如果最后左和右一样长，最后只会输出一个结果
            return left
        else:
            return right
```

In [731]:

```
def get_pos(pos_pair):
    pos = []
    for i in pos_pair:
        pos.append(i[1][0])
    return pos
```

In [273]:

```
data['cixing1'] = data['pos1'].apply(get_pos)
```

In [274]:

```
data['cixing2'] = data['pos2'].apply(get_pos)
```

In [287]:

```
data['LCS'] = data.apply(lambda x:LCS(tuple(x['cixing1']),tuple(x['cixing2'])), axis=1)
```

In [289]:

```
def get_lcs_ratio(lcs, s1,s2):
    len_s1 = len(s1.split(' '))
    len_s2 = len(s2.split(' '))
    return len(lcs)/(len_s1+len_s2)
```

In [291]:

```
data['lcs_ratio'] = data.apply(lambda x:get_lcs_ratio(x['LCS'],x['s1'],x['s2']), axis=1)
```

In [293]:

```
data.to_pickle('data-02021837.pic')
```

一些零散的特征

构造了一些零散的特征。其中两个句子相同的的词（函数写的不太好.....）这个特征在最后的权重还有一些的。其他的并没有特别高的权重，

In [855]:

```
data['bag_vec_add'] = data['bag1_vec'] + data['bag2_vec']
data['dist'] = data['bag_diff'].apply(lambda x:np.linalg.norm(x))
data['dist'] = data['dist']/data['dist'].max()
data['dist']
data['add_norm'] = data['bag_vec_add'].apply(lambda x:np.linalg.norm(x))
data['add_norm'] = data['add_norm']/data['add_norm'].max()
data['add_norm']
data['seq_sim'] = data.apply(lambda x:1-(x['dist']/x['add_norm']), axis=1)
data['seq_sim'].min()
data['add_div_dist'] = data.apply(lambda x:(x['add_norm']/(x['dist']+1)), axis=1)
data['cos'] = data.apply(lambda x:np.dot(x['bag1_vec'], x['bag2_vec'])/(np.linalg.norm(x['bag1_vec'])*np.linalg.no
```

```
data['bag1_norm']=data['bag1_vec'].apply(lambda x:np.linalg.norm(x))
data['bag2_norm']=data['bag2_vec'].apply(lambda x:np.linalg.norm(x))
data['bag1_norm'] = data['bag1_norm']/data['bag1_norm'].max()
data['bag2_norm'] = data['bag2_norm']/data['bag2_norm'].max()
data.to_pickle('data-02020418.pic')
def get_same_ratio(a,b):
    count = 0

    for i in a:
        for j in b:
            if i == j:
                count += 1
    return count/(min(len(a),len(b))+1)

def get_word_same_ratio(a,b):
    count = 0
    len_add = len(a)+len(b)
    for i in a:
        for j in b:
            if get_same_ratio(i,j) >= 0.3:
                count += 1
    return count/len_add

data['word_same_ratio'] = data.apply(lambda x:get_same_ratio(x['bag1'],x['bag2']), axis=1)
# for i in data.columns:
#     print(i)
# data[['bag1','bag2']]

data['lem_word_same_ratio'] = data.apply(lambda x:get_same_ratio(x['bag1_lem'],x['bag2_lem']), axis=1)
for i in data.columns:
    print(i,end='/')
def get_noun(sent):
    nouns = []
    for i in sent:
        if i[1].startswith('N'):
            if lemmatizer.lemmatize(i[0]) in model:
                nouns.append(lemmatizer.lemmatize(i[0]))
            elif i[0] in model:
                nouns.append(i[0])
    return nouns
def get_similariy(bag1, bag2):
    if not bag1 or not bag2:
        print(1)
        return np.NaN
    else:
        return model.n_similarity(bag1, bag2)
data['s1_noun'] = data['pos1'].apply(get_noun)
data['s2_noun'] = data['pos2'].apply(get_noun)
data['noun_sim'] = data.apply(lambda x:get_similariy(x['s1_noun'],x['s2_noun']), axis=1)
data['noun_sim']
```

把数据送到模型里

终于到了激动人心的时刻了。这我就没啥说的了。 从NLP_newbee队伍的助教大佬常永炷大佬的分享中，学到了好多库的使用。然后就一点一点搞就可以了。 调参实在是不太懂，从来自三缺一队伍的张庆恒大佬的分享中学到的GridSearchCV（这些库看文档还蛮友好的）也用上了，但是调参真的不太会。

不知道为什么，有些特征有Nan，所以首先用均值填充了一下。

In [683]:

```
data_fill = data.fillna(data.mean())
```

In [684]:

```
# data_fill.describe()
```

In [742]:

```
A = data_fill[[i for i in data.columns if data[i].dtype.str.startswith('<f')]]
X = pd.DataFrame()
# X = data_normalization(X,[i for i in X.columns])
all_columns = list(A.columns.values)
#the first two columns don't need to be normalized
for column in all_columns:
    X[column] = A.loc[:,column] / A.loc[:,column].max()
```

In [743]:

```
# for i in X.columns:
#     print(i)
```

In [744]:

```
y = data_fill['sim']
```

```
y = data_fit['sim']
```

这里尝试着drop了很多特征。最高的那次留下谁了我忘掉了.....

目录

In [745]:

```
X = X.drop('sim', axis=1)
# X = X.drop('o_sim', axis=1)
# X = X.drop('n_sim', axis=1)
X = X.drop('seq_sim', axis=1)
X = X.drop('add_norm',axis=1)
# X=X.drop('cos',axis=1)
# X=X.drop('word_same_ratio',axis=1)
# X = X.drop('s_sim', axis=1) # 68
# X = X.drop('pro_euc_sim', axis=1)#68

X = X.drop([i for i in range(256)],axis=1)
# X = X.drop('lstm_pn',axis=1)
# X.describe()
# X = X.drop('lem_word_same_ratio', axis=1)
# X = X.drop('noun_sim', axis=1)
```

In [746]:

```
X_train_raw = np.asarray(X.loc[train.id,:])
y_train_raw = np.asarray(y[train.id]).ravel()
```

In [747]:

```
X_result = np.asarray(X.loc[test.id])
```

In [748]:

```
import random
train_id = random.sample(range(1500),1000)
train1_id = random.sample(train_id,500)
train2_id = [i for i in train_id if i not in train1_id]
test_id = [i for i in range(1500) if i not in train_id]
# print(train1_id)
# test_id
```

In [749]:

```
# # from sklearn.model_selection import train_test_split
# # X_train,X_test,y_train,y_test = train_test_split(X_train_raw,y_train_raw, test_size=0.3)

# # X_train.shape
# # def get_value(lst, index_lst):

X_train = np.array([X_train_raw[i] for i in train_id])
y_train = np.array([y_train_raw[i] for i in train_id])
X_train1 = np.array([X_train_raw[i] for i in train1_id])
X_train2 = np.array([X_train_raw[i] for i in train2_id])
y_train1 = np.array([y_train_raw[i] for i in train1_id])
y_train2 = np.array([y_train_raw[i] for i in train2_id])
X_test = np.array([X_train_raw[i] for i in test_id])
y_test = np.array([y_train_raw[i] for i in test_id])
```

In [750]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.cross_validation import cross_val_score
```

In [751]:

```
from sklearn.neural_network import MLPRegressor
mlpr = MLPRegressor(
                (300,200,100,10),
                    early_stopping=True,
#                     learning_rate="adaptive",
                    batch_size=20,
#                     solver='lbfgs',
                    max_iter=400)
mlpr.fit(X_train,y_train)
print(mlpr.score(X_test,y_test))
# MLPRegressor?
```

```
0.605152221864
```

In [752]:

```
# from sklearn.kernel_ridge import KernelRidge
# parameters = {'alpha':[0.05,0.06,0.07,0.1,0.2]}
# kr = GridSearchCV(KernelRidge(), parameters, verbose=1, cv=5)
# kr.fit(X_train,y_train)
# print(np.mean(cross_val_score(kr,X_test,y_test)))
```

```
# print(kr.best_params_)
knr = KernelRidge(alpha=0.06)
knr.fit(X_train,y_train)
print(knr.score(X_test,y_test))
```

目录

```
0.625004646359
```

In [753]:

```
# parameters = {'kernel':('linear', 'rbf'), 'C':[2.5,3,3.5], 'gamma':[1.1,1.2]}
# clf = GridSearchCV(SVR(), parameters, verbose=1, cv=5)
# clf.fit(X_train,y_train)
# # from sklearn.svm import SVR
# clf.best_params_
svr1 = SVR(C=2.5,
           gamma=1.2,
           kernel='rbf')
svr1.fit(X_train,y_train)
print(svr1.score(X_test,y_test))

# svr2 = SVR(kernel='linear')
# svr2.fit(X_train,y_train)
# print(svr2.score(X_test,y_test))
```

```
0.639977385717
```

In [754]:

```
from xgboost import XGBRegressor
model_xgb1 = XGBRegressor(
#                          colsample_bytree=0.9,
                         gamma=3,
                         learning_rate=0.01,
                         max_depth=9,
                         min_child_weight=3,
                         n_estimators=896,
                         reg_alpha=0.011,
                         reg_lambda=2.1,
                         subsample=0.545,
#                          silent=0,
                         seed =1024,
                         nthread = -1)
model_xgb1.fit(X_train,y_train)
print(model_xgb1.score(X_test,y_test))

from xgboost import XGBRegressor
model_xgb2 = XGBRegressor(
#                          colsample_bytree=0.9,
                         gamma=3,
                         learning_rate=0.01,
                         max_depth=9,
                         min_child_weight=3,
                         n_estimators=896,
                         reg_alpha=0.011,
                         reg_lambda=2.1,
                         subsample=0.545,
#                          silent=0,
                         seed =512,
                         nthread = -1)
model_xgb2.fit(X_train,y_train)
print(model_xgb2.score(X_test,y_test))
```

```
0.658121938448
0.657026514966
```

In [755]:

```
# from lightgbm import LGBMRegressor
lgb = LGBMRegressor(objective='regression',
                    num_leaves=7,
                          learning_rate=0.02,
                    n_estimators=500,
                          max_bin = 70,
                          bagging_freq = 3,
                    feature_fraction = 0.7,
                          feature_fraction_seed=9,
                    bagging_seed=9,
                          min_data_in_leaf =42,
                    min_sum_hessian_in_leaf = 40
                   )
lgb.fit(X_train,y_train)
print(lgb.score(X_test,y_test))
```

```
0.649552150546
```

In [756]:

```
# test = LGBMRegressor()
# test.fit(X_train,y_train)
# print(test.score(X_test,y_test))
```

In [757]:

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
GBoost = GradientBoostingRegressor(
        learning_rate=0.06,
            n_estimators=115,

)
GBoost.fit(X_train,y_train)
GBoost.score(X_test,y_test)
```

Out[757]:

In [758]:

```
# GradientBoostingRegressor?
```

In [ ]:

In [759]:

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
rf = RandomForestRegressor(random_state=590,n_estimators =800)

rf.fit(X_train2,y_train2)
print(rf.score(X_test,y_test))
for (i,j) in zip(X.columns,rf.feature_importances_):
    print(i,'\t',j)
```

```
0.628457772076
s_sim    0.0328436916039
o_sim    0.388272667482
v_sim    0.0586436305671
n_sim    0.0141257815967
lstm_pn          0.0643799208894
dist     0.0361463594716
add_div_dist     0.0325179244386
cos      0.0134677487819
bag1_norm        0.0348014902092
bag2_norm        0.0360521254774
word_same_ratio          0.035282168035
lem_word_same_ratio      0.0293881401164
noun_sim         0.068346830424
pro_euc_sim      0.0253655738661
pro_pear_sim     0.053780182805
pro_cos_sim      0.0408561746783
lcs_ratio        0.0357295895575
```

In [644]:

```
# model.most_similar('cat')
```

In [760]:

```
Regressors = [svr1, lgb, model_xgb1, model_xgb2,rf]
for i in Regressors:
    i.fit(X_train,y_train)
    print(i.score(X_test,y_test))
rf.feature_importances_
```

```
0.639977385717
0.649552150546
0.658121938448
0.657026514966
0.659587021055
```

Out[760]:

这个Stacking的方法是和常永炷大佬学的……

In [761]:

```
from mlxtend.regressor import StackingRegressor
stregr = StackingRegressor(regressors=Regressors, meta_regressor=model_xgb1)
stregr.fit(X_train,y_train)
stregr.score(X_test,y_test)
```

Out[761]:

目录

In [762]:

```
regs = Regressors + [stregr]
for i in regs:
    i.fit(X_train_raw,y_train_raw)
```

In [763]:

```
final_pd = data_fill[[i for i in data.columns if data[i].dtype.str.startswith('<f')]]

final_pd = final_pd.drop('sim', axis=1).drop([i for i in range(256)],axis=1).drop('seq_sim',axis=1).drop('add_norm
```

In [764]:

```
result_pd = pd.DataFrame()
for index, reg in enumerate(regs):
    result_pd[index] = reg.predict(np.array(final_pd))
```

In [765]:

```
# result_pd.drop('result',axis=1)
result_pd['result'] = result_pd.apply(lambda x:np.mean(x),axis=1)
```

In [766]:

```
preds = result_pd.loc[1500:,'result']
result_pd
```

Out[766]:

In [767]:

```
preds
```

Out[767]:

In [768]:

```
with open("submission_sample", "w") as f:
    for idx, pred in zip(test.id, preds):
        f.write(str(idx) + "," + str(pred) + "\n")
```

In [ ]:

目录

W1_138王泽群

Fork 点赞

目录

目

录

和鲸社区公众号

工具

K-Lab

内容

项目

数据集

专栏

专区

实战

比赛

众包

商务合作

© 和鲸HeyWhale 沪ICP备14038218号-1 沪公网安备 31010402003637号