目录

W2_120李碧涵

4 746 9

TAIL CAMP——AI实战训练营 自然语言处理 Week2_作文自动评分

Yummy蛋卷 LV2 3

Fork 点赞

内容 数据集 Fork记录 9 评论

内容

版本 3 2018-02-09 23:44

TAIL CAMP AI实战训练营 NLP 第二周项目报告——手写作文自动评分

目录

> 组名：要没时间了——小组成员李碧涵和陈文帆

Contents

总述

1. 项目简介

- 本次项目针对学生所写的英文作文进行机器自动评分，所写作文围绕一个主题；
- 作文人工评分标准，共6个等级打分，用1——6之间的分值来表征
- 共有两个老师进行打分，最终得分为两个老师得分之和

> 方法一：特征工程 + 传统算法

2. 特征选择

> 参照学习大纲中给定的论文以及相关的参考论文，此处共采用32个特征+稀疏特征posngrams，特征解释如下:

1） Length Features

> - 单词数word_count
> - 句子数sentence_count
> - 每个句子的平均单词数avg_sentence_len
> - 每个单词的平均长度avg_word_len
> - 长单词数long_word (这里选定长度≥7的为长单词)
> - 停用词个数stopwords_count

2） Occurrence Features

> - 感叹号出现的数目exc_count
> - 问号出现的数目que_count
> - 逗号出现的数目comma_count

3） Error Features

> - 拼写错误的单词数spelling_errors

目录

4） ngrams_counts Features

此特征可以说明作者的词汇丰富程度

- unigrams_count：将文章分词后-->采用1-gram-->统计非重复gram的个数
- bigrams_count：将文章分词后-->采用2-grams-->统计非重复grams的个数
- trigrams_count：将文章分词后-->采用3-grams-->统计非重复grams的个数

5） POS counts Features

此特征用于统计文章中不同词性的个数

- 名词noun_count
- 形容词adj_count
- 副词adv_count
- 动词verb_count
- 外来词fw_count

6） Readability Features

可读性分析，可参见textstat模块 (http://textblob.readthedocs.io/en/dev/quickstart.html#quickstart)

- 音节检测syllable_count
- flesch_reading_ease,smog_index,flesch_kincaid_grade,coleman_liau_index, automated_readability_index,dale_chall_readability_score,linsear_write_formula, gunning_fog
- 困难单词数difficult_words

7） Personality Features

分析文中每句话的语气:positive，negative or neutrual

- 消极语气neg_sentiment
- 中立语气neu_sentiment
- 积极语气pos_sentiment

8） Cohesion Features

文章的连词的使用，可以反映文章结构 --> 再用文章的token总数来normalize

- cohesion

9） POS ngrams稀疏特征

对文章词性做ngrams --> 过滤出所有文章中频数超过30的ngrams特征 --> 统计每篇文章中ngrams特征出现的个数

- posngs

* 论文中表明此特征很重要，但是从实际意义上暂未获得合理解释。

3. 模型选择

采用LassoCV,XGBRegressor,RandomForestRegressor和GradientBoostingRegressor四个模型；

用4个模型分别预测score1和score2,最终预测采取score = score1+score2求和的形式；

对四个模型预测到的score取平均。

1） 刚开始考虑了以下7个模型，分别测试了score = score1+score2和直接预测score两种方式，得到以下结果：

	直接预测 score	score = score1+score2
LogisticRegression	0.6458680291851369	0.6644192462674752
Lasso	0.8140479312552429	0.8082751969235769
LassoCV	0.817965416989607	0.8181317310494515
XGBRegressor	0.8192058190153856	0.8210317669184185
RandomForestRegressor	0.8017463503474 67	0.8031391977172094

RandomForestRegressor	0.8017463553547467	0.8053591977172094
GradientBoostingRegressor	0.8176579049616708	0.818029212009945
SVR	0.09704862677364856	0.09626108405035987

目录

所以最终选用了上述表现较好的4个模型，预测方式采用score = score1+score2

2） 线下测试了一下四个模型取平均和Stacking方式，发现取平均略好。

正文

In [ ]:

```
!pip install wordcloud textstat seaborn wordcloud
```

In [4]:

```
# 加载库
import nltk
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
% matplotlib inline

from textblob import Word
from scipy.stats import pearsonr
from nltk.corpus import stopwords
from textstat.textstat import textstat
from xgboost.sklearn import XGBRegressor
from sklearn.linear_model import LassoCV
from nltk import ngrams,word_tokenize,pos_tag
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
```

1. 数据集预览

In [6]:

```
train_path = '/home/kesci/input/Tal_SenSco/Tal_SenSco_train_data.txt'
test_path = '/home/kesci/input/Tal_SenSco/Tal_SenSco_test_data.txt'
df_train = pd.read_table(train_path,header = None,names=["id", "text", "score1", "score2","score"])
print(df_train.head())
df_test = pd.read_table(test_path,header = None,names=["id", "text"])
print(df_test.head())
```

```
   id                                               text  score1  score2  \
0   1  Dear Local Newspaper, I believe the computer d...       4       4
1   2  Dear @CAPS1 @CAPS2, I have heard the concern o...       6       4
2   3  @CAPS1 you know we all like computers and we a...       4       4
3   4  I Believe that computers have a positive effec...       6       5
4   5  Dear newstimes, I really think that computers ...       5       4

   score
0      8
1     10
2      8
3     11
4      9
     id                                               text
0  1201  Dear Local Newspaper @CAPS1, I am against that...
1  1202  Dear local newspaper, @PERCENT1 of @CAPS1 own ...
2  1203  Computers, computers, computers. Who doesn't o...
3  1204  Dear Local Newspaper, Computers are a vey uniq...
4  1205  I think having computers are good because you ...
```

2. 数据分析

1) 训练集和测试集词云

In [11]:

```
from wordcloud import WordCloud

plt.figure(figsize=(20, 15))
```

```
plt.subplot(1, 2, 1)
cloud1 = WordCloud(width=1440, height=1080).generate(' '.join(df_train["text"]))
plt.title('Wordcloud of training data', fontsize=15)
plt.imshow(cloud1)
plt.axis('off')

plt.subplot(1, 2, 2)
cloud1 = WordCloud(width=1440, height=1080).generate(' '.join(df_test["text"]))
plt.title('Wordcloud of test data', fontsize=15)
plt.imshow(cloud1)
plt.axis('off')
plt.show()
```

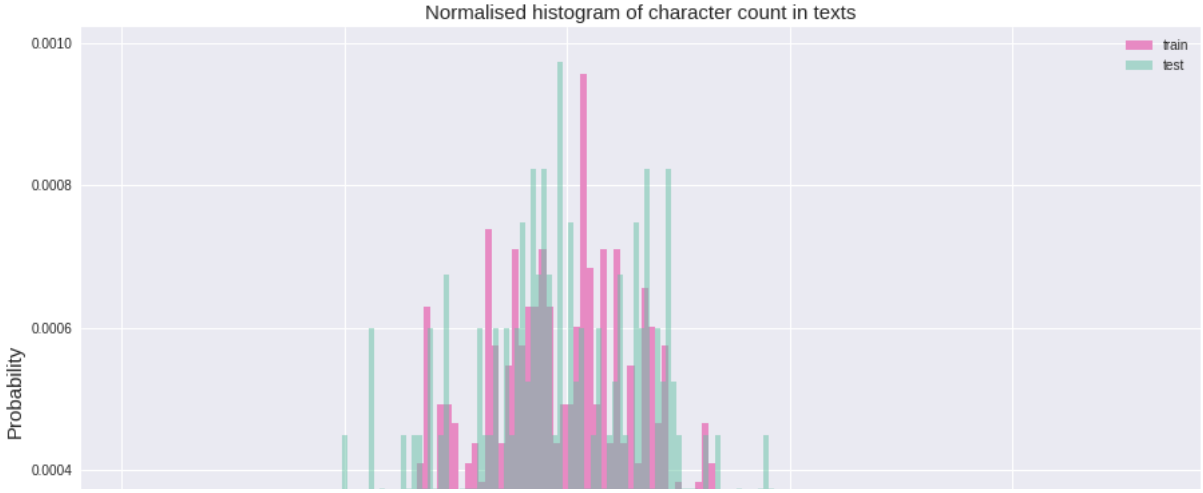
从上图可以看出,训练集和测试集的高频词汇差不多。

2) 文本长度

In [13]:

```
import seaborn as sns
pal = sns.color_palette("Set2", 10)

dist_train = df_train["text"].apply(len)
dist_test = df_test["text"].apply(len)

plt.figure(figsize=(15, 10))
plt.hist(dist_train, bins=150, color=pal[3], normed=True, label='train')
plt.hist(dist_test, bins=150, color=pal[8], normed=True, alpha=0.5, label='test')
plt.title('Normalised histogram of character count in texts', fontsize=15)
plt.legend()
plt.xlabel('Number of characters', fontsize=15)
plt.ylabel('Probability', fontsize=15)

print('均值-训练集:{:.2f},标准差-训练集:{:.2f},最大值-训练集:{:.2f}\n均值-测试集:{:.2f},标准差-测试集:{:.2f},最大值-测试集:{:.
                        dist_train.std(), dist_train.max(), dist_test.mean(), dist_test.std(), dist_test.max()))
```

```
均值-训练集:2053.13,标准差-训练集:699.68,最大值-训练集:4616.00
均值-测试集:1981.88,标准差-测试集:661.98,最大值-测试集:3754.00
```



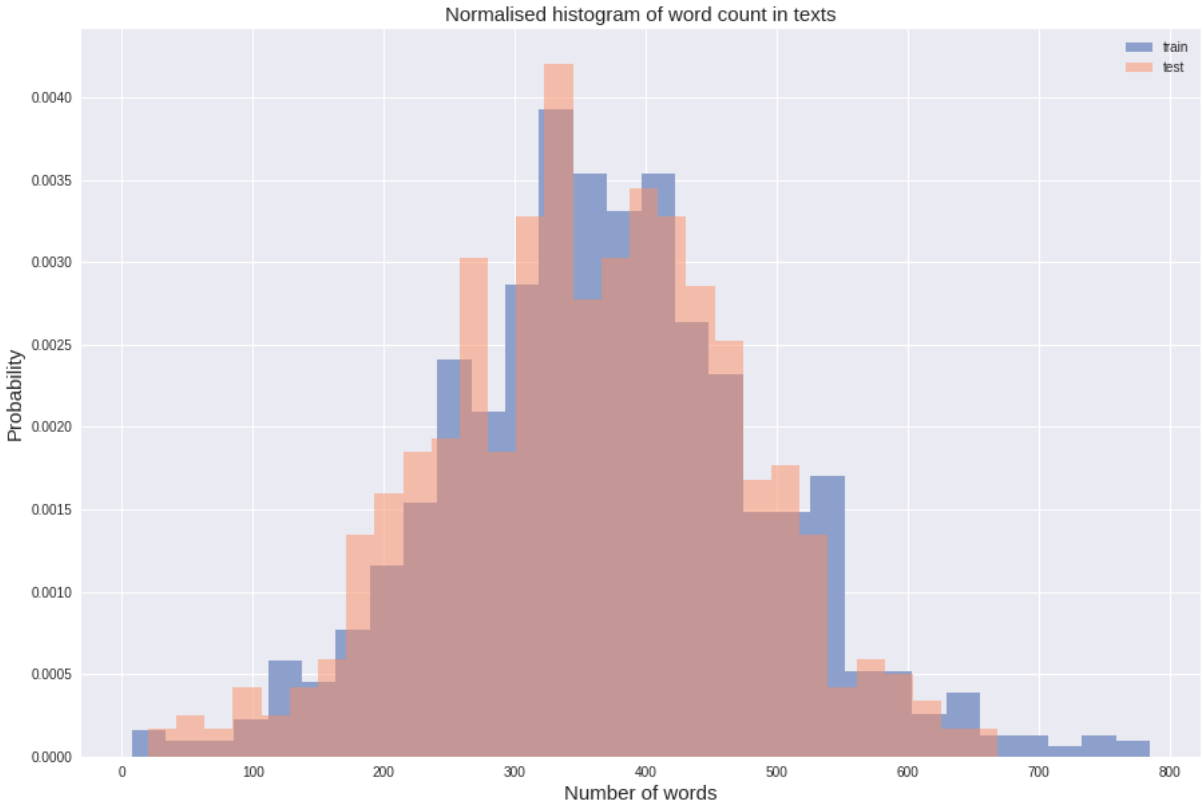
目录

3) 单词数

In [14]:

```
dist_train = df_train["text"].apply(lambda x: len(x.split(' ')))
dist_test = df_test["text"].apply(lambda x: len(x.split(' ')))

plt.figure(figsize=(15, 10))
plt.hist(dist_train, bins=30, color=pal[2], normed=True, label='train')
plt.hist(dist_test, bins=30, color=pal[1], normed=True, alpha=0.5, label='test')
plt.title('Normalised histogram of word count in texts', fontsize=15)
plt.legend()
plt.xlabel('Number of words', fontsize=15)
plt.ylabel('Probability', fontsize=15)

print('均值-训练集:{:.2f},标准差-训练集:{:.2f},最大值-训练集:{:.2f}\n均值-测试集:{:.2f},标准差-测试集:{:.2f},最大值-测试集:{:.
                      dist_train.std(), dist_train.max(), dist_test.mean(), dist_test.std(), dist_test.max()))
```

```
均值-训练集:370.06,标准差-训练集:121.00,最大值-训练集:785.00
均值-测试集:357.36,标准差-测试集:115.40,最大值-测试集:669.00
```



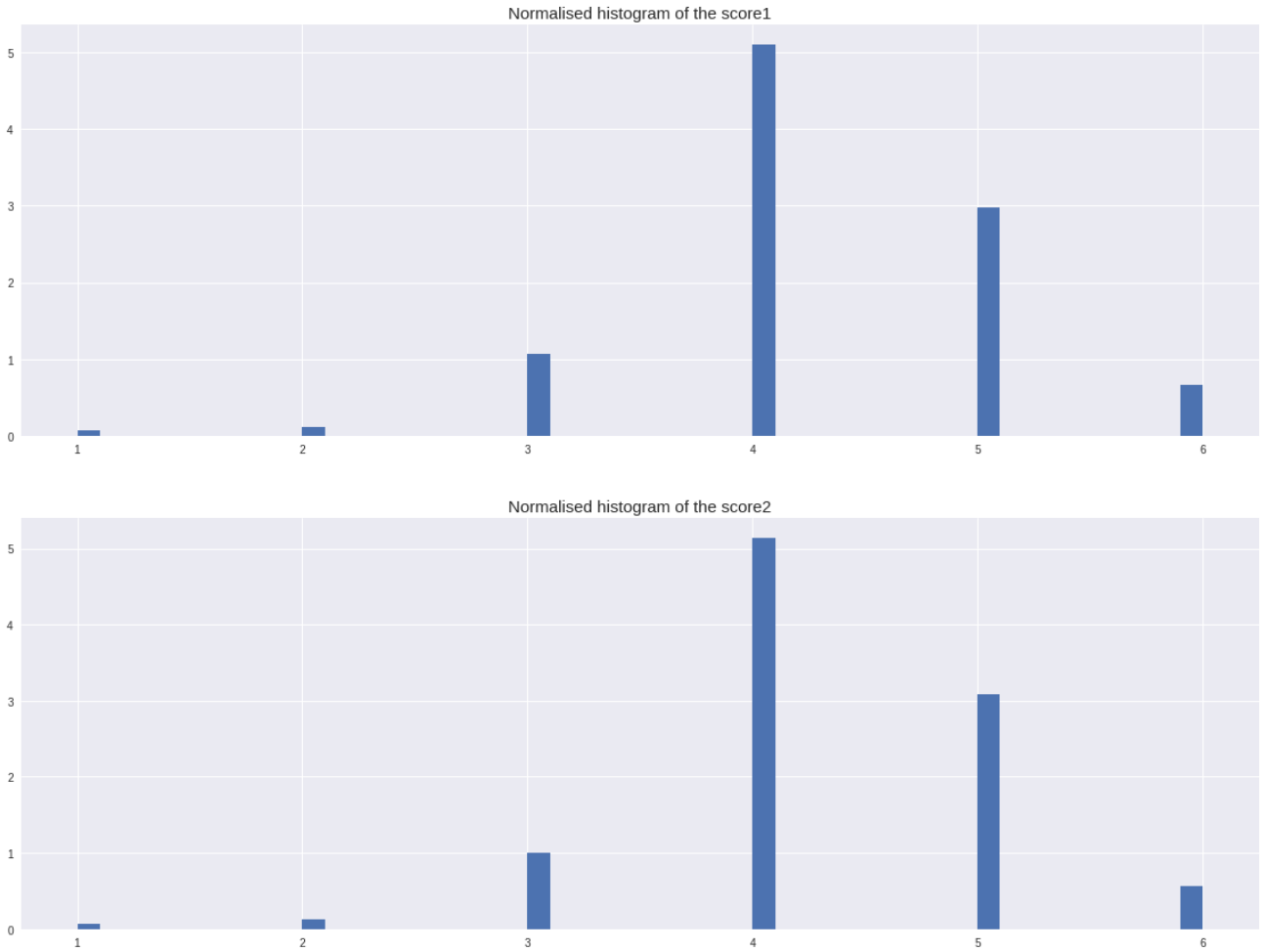
4) 评分分析

从下图可以看出，score1和score2的取值范围是{2,3,4,5,6}

In [15]:

```
plt.figure(figsize=(20, 15))
plt.subplot(2, 1, 1)
plt.hist(df_train.score1,bins=50,normed=True)
plt.title('Normalised histogram of the score1', fontsize=15)
```

```
plt.subplot(2, 1, 2)
plt.hist(df_train.score2,bins=50,normed=True)
plt.title('Normalised histogram of the score2', fontsize=15)
plt.show()
```



3. 特征提取

In [16]:

```
train_features = pd.DataFrame()
test_features = pd.DataFrame()
```

In [22]:

```
# 分词,不会自动滤除标点符号
df_train['words'] = df_train.text.apply(word_tokenize)
df_test['words'] = df_test.text.apply(word_tokenize)

# 分句
sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
df_train['sents'] = df_train.text.apply(sent_detector.tokenize)
df_test['sents'] = df_test.text.apply(sent_detector.tokenize)

# 分字母,求得长度
df_train['character_count'] = df_train['words'].apply(lambda x: len(''.join(x)) )
df_test['character_count'] = df_test['words'].apply(lambda x: len(''.join(x)) )

#停用词
stopword = stopwords.words("english")

# 分词的tag(tag)
df_train['tags'] = df_train['words'].apply(pos_tag)
df_test['tags'] = df_test['words'].apply(pos_tag)
```

1. Length Features

> 6个

In [23]:

目录

```
# 单词数
train_features['word_count'] = df_train['words'].apply(len)
test_features['word_count'] = df_test['words'].apply(len)

# 句子数
train_features['sentence_count'] = df_train['sents'].apply(len)
test_features['sentence_count'] = df_test['sents'].apply(len)

# 每个句子的平均单词数
train_features['avg_sentence_len'] = train_features['word_count'].values / train_features['sentence_count'].values
test_features['avg_sentence_len'] = test_features['word_count'].values / test_features['sentence_count'].values

# 每个单词的平均长度
train_features['avg_word_len'] = df_train['character_count'].values / train_features['word_count'].values
test_features['avg_word_len'] = df_test['character_count'].values / test_features['word_count'].values

# 长单词数
train_features['long_word'] = df_train['words'].apply(lambda x: sum([len(word) >=7 for word in x]))
test_features['long_word'] = df_test['words'].apply(lambda x: sum([len(word) >=7 for word in x]))

# 停用词个数
train_features['stopwords_count'] = df_train['words'].apply(lambda x: len([word for word in x if word in stopword]
test_features['stopwords_count'] = df_test['words'].apply(lambda x: len([word for word in x if word in stopword]))
```

2. Occurrence Features

3个

In [24]:

```
train_features['exc_count'] = df_train['text'].apply(lambda x: x.count('!'))
test_features['exc_count'] = df_test['text'].apply(lambda x: x.count('!'))

train_features['que_count'] = df_train['text'].apply(lambda x: x.count('?'))
test_features['que_count'] = df_test['text'].apply(lambda x: x.count('?'))

train_features['comma_count'] = df_train['text'].apply(lambda x: x.count(','))
test_features['comma_count'] = df_test['text'].apply(lambda x: x.count(','))
```

3. Error Features

1个

In [25]:

```
#拼写错误的单词数
train_features['spelling_errors'] = df_train['words'].apply(lambda x: sum([Word(word).spellcheck()[0][0] != word f
test_features['spelling_errors'] = df_test['words'].apply(lambda x: sum([Word(word).spellcheck()[0][0] != word for
```

4.ngrams_counts Features

3个

In [26]:

```
train_features['unigrams_count'] = df_train['words'].apply(lambda x:len(set([grams for grams in ngrams(x,1)])))
train_features['bigrams_count'] = df_train['words'].apply(lambda x:len(set([grams for grams in ngrams(x,2)])))
train_features['trigrams_count'] = df_train['words'].apply(lambda x:len(set([grams for grams in ngrams(x,3)])))
```

5. POS counts Features

5个

In [27]:

```
train_features['noun_count'] = df_train['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("NN")])
test_features['noun_count'] = df_test['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("NN")]))

train_features['adj_count'] = df_train['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("JJ")]))
test_features['adj_count'] = df_test['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("JJ")]))

train_features['adv_count'] = df_train['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("RB")]))
test_features['adv_count'] = df_test['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("RB")]))

train_features['verb_count'] = df_train['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("VB")])
test_features['verb_count'] = df_test['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("VB")]))
```

```
train_features['fw_count'] = df_train['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("FW")]))
test_features['fw_count'] = df_test['tags'].apply(lambda x: len([tag for tag in x if tag[1].startswith("FW")]))
```

目录

6.Readability Features

10个

In [28]:

```
train_features['syllable_count'] = df_train.text.apply(textstat.syllable_count)
train_features['flesch_reading_ease'] = df_train.text.apply(textstat.flesch_reading_ease)
train_features['smog_index'] = df_train.text.apply(textstat.smog_index)
train_features['flesch_kincaid_grade'] = df_train.text.apply(textstat.flesch_kincaid_grade)
train_features['coleman_liau_index'] = df_train.text.apply(textstat.coleman_liau_index)
train_features['automated_readability_index'] = df_train.text.apply(textstat.automated_readability_index)
train_features['dale_chall_readability_score'] = df_train.text.apply(textstat.dale_chall_readability_score)
train_features['linsear_write_formula'] = df_train.text.apply(textstat.linsear_write_formula)
train_features['gunning_fog'] = df_train.text.apply(textstat.gunning_fog)
train_features['difficult_words'] = df_train.text.apply(textstat.difficult_words)  # 困难单词数

test_features['syllable_count'] = df_test.text.apply(textstat.syllable_count)
test_features['flesch_reading_ease'] = df_test.text.apply(textstat.flesch_reading_ease)
test_features['smog_index'] = df_test.text.apply(textstat.smog_index)
test_features['flesch_kincaid_grade'] = df_test.text.apply(textstat.flesch_kincaid_grade)
test_features['coleman_liau_index'] = df_test.text.apply(textstat.coleman_liau_index)
test_features['automated_readability_index'] = df_test.text.apply(textstat.automated_readability_index)
test_features['dale_chall_readability_score'] = df_test.text.apply(textstat.dale_chall_readability_score)
test_features['linsear_write_formula'] = df_test.text.apply(textstat.linsear_write_formula)
test_features['gunning_fog'] = df_test.text.apply(textstat.gunning_fog)
test_features['difficult_words'] = df_test.text.apply(textstat.difficult_words)  # 困难单词数
```

7. Personality Features

3个 分析文中每句话的语气:positive，negative or neutrual

In [29]:

```
sid = SentimentIntensityAnalyzer()
df_train['sents_polar'] = df_train['sents'].apply(lambda x: [sid.polarity_scores(sent) for sent in x])
df_test['sents_polar'] = df_test['sents'].apply(lambda x: [sid.polarity_scores(sent) for sent in x])
```

In [30]:

```
train_features['neg_sentiment'] = df_train['sents_polar'].apply(lambda x: sum([p for item in x for polar,p in item
train_features['neu_sentiment'] = df_train['sents_polar'].apply(lambda x: sum([p for item in x for polar,p in item
train_features['pos_sentiment'] = df_train['sents_polar'].apply(lambda x: sum([p for item in x for polar,p in item

test_features['neg_sentiment'] = df_test['sents_polar'].apply(lambda x: sum([p for item in x for polar,p in item.i
test_features['neu_sentiment'] = df_test['sents_polar'].apply(lambda x: sum([p for item in x for polar,p in item.i
test_features['pos_sentiment'] = df_test['sents_polar'].apply(lambda x: sum([p for item in x for polar,p in item.i
```

8.Cohesion Features

1个

In [31]:

```
connectives = {
        "also", "besides", "futhermore", "too", "moreover", "in addition",
        "finally", "of equal importance", "equally important", "from here on", "from now on",
        "hence", "next","then", "to begin with", "last of all", "after", "before", "as soon as",
        "in the end", "gradually", "yet", "however", "but", "rather", "on the other hand",
        "on the contrary", "nevertheless", "in spite of", "in contrast to this", "although",
        "therefore", "consequently", "in summary", "to sum up", "briefly", "in short",
        "in conclusion", "as you can see", "in fact", "of course", "to be sure", "naturally",
        "obviously", "no doubt", "at least", "anyhow", "for example", "for instance",
        "indeed", "specially", "in particular", "unfortunately", "afterward", "later",
        "at last", "now", "subsequently", "in the meantime", "meanwhile", "otherwise",
        "accordding to", "so that", "so then", "unless", "on condition", "in case", "but that",
        "suppose", "in order", "first of all", "at the same time", "wherever", "let",
        "wherefore", "ultimately", "presently", "at length", "on the following day",
        "the next", "after a short time", "soon", "lastly", "in especial", "probably", "it is certoin",
        "to be sure", "perhaps", "on the whole", ""
    }

def calculate_cohesion(text):
```

```
test_features["cohesion"] = df_test["text"].apply(str.lower).apply(calculate_cohesion) / test_features['word_count
```

目录

上述特征存储到本地,方便读取

In [32]:

```
train_features.to_csv('/home/kesci/input/features6663/train_data.csv',index=False)
train_features.to_csv('/home/kesci/input/features6663/test_data.csv',index=False)
```

In [33]:

```
### 验证下
train_data = pd.read_csv('/home/kesci/input/features6663/train_data.csv')
test_features = pd.read_csv('/home/kesci/input/features6663/test_data.csv')
```

In [34]:

```
print("Number of training features:",train_data.shape[1]-3, "\nNumber of test features:", test_features.shape[1])
```

```
Number of training features: 32
Number of test features: 32
```

9.POS ngrams

In [35]:

```
# 将词转换成tag的列表
# df_train['tags']
def poslist(essay):
    tokens = word_tokenize(essay.lower().strip())
    poslist = []
    p = pos_tag(tokens)
    for (token,tag) in p:
        poslist.append(tag)

    return ' '.join(poslist)
```

In [36]:

```
cv = CountVectorizer(lowercase=False, ngram_range=(1, 3), binary=True)
# getting binarized pos-ngrams
pos_train = df_train['text'].apply(poslist)  #将每句话转换成tag的列表
ngs = cv.fit_transform(pos_train).toarray()  #将不易操作的tags转换成词向量  #Train:1200 rows x 10404 columns,共有10404
train_posngs = ngs[:, np.where(ngs.sum(axis=0) > 30)[0]]

pos_test = df_test['text'].apply(poslist)
ngs_test = cv.transform(pos_test).toarray()
test_posngs = ngs_test[:, np.where(ngs.sum(axis=0) > 30)[0]]
```

此特征存储到本地,方便读取

In [37]:

```
pd.DataFrame(train_posngs).to_csv('/home/kesci/input/features6663/train_posngs.csv',index=None)
pd.DataFrame(test_posngs).to_csv('/home/kesci/input/features6663/test_posngs.csv', index=None)
```

☆ 载入train和test数据集(已提取过特征)

In [7]:

```
train_data = pd.read_csv('/home/kesci/input/features6663/train_data.csv')
test_features = pd.read_csv('/home/kesci/input/features6663/test_data.csv')
test_ids = df_test['id']    #存一下测试集的id
```

In [8]:

```
print("Number of training features:",train_data.shape[1]-3, "\nNumber of test features:", test_features.shape[1])
```

```
Number of training features: 32
Number of test features: 32
```

In [40]:

```
train_posngs = pd.read_csv('/home/kesci/input/features6663/train_posngs.csv')
test_posngs = pd.read_csv('/home/kesci/input/features6663/test_posngs.csv')

train_data = pd.concat([train_posngs, train_data], axis=1)
test_features = pd.concat([test_posngs, test_features], axis=1)
```

In [41]:

```
print("Number of training features:",train_data.shape[1]-3, "\nNumber of test features:", test_features.shape[1])
```

```
Number of training features: 2360
Number of test features: 2360
```

目录

归一化特征

In [42]:

```
std = StandardScaler()

dataSet = std.fit_transform(train_data.iloc[:, :-3])
test_features = std.transform(test_features)
labelSet = train_data.iloc[:,-3:]
train_text, test_text, train_labels, test_labels = train_test_split(dataSet, labelSet, test_size=0.01,random_state
```

4. 模型训练

线下测了一下以下模型：LogisticRegression,Lasso,LassoCV,XGBRegressor,RandomForestRegressor,GradientBoostingRegressor,SVR 挑选了四个性能最优的。

In [26]:

```
# ① LassoCV模型
model_lscv = LassoCV(alphas = [0.01]) #调参alphas
# 拟合score1
model_lscv.fit(train_text, train_labels.iloc[:, -3])
lscv_pred1 = model_lscv.predict(test_text)
# 拟合score2
model_lscv.fit(train_text, train_labels.iloc[:, -2])
lscv_pred2 = model_lscv.predict(test_text)
lscv_pred = lscv_pred1 + lscv_pred2

# ② XGBRegressor模型
model_xgb1 = XGBRegressor(learning_rate =0.1, n_estimators=50, max_depth=3,min_child_weight=3, gamma=0.02,
                          colsample_bytree=0.8,subsample=0.5, seed=27)
model_xgb2 = XGBRegressor(learning_rate =0.1, n_estimators=50, max_depth=2,min_child_weight=3,gamma=0.05,
                           colsample_bytree=0.6,subsample=0.7, seed=27)
# 拟合score1
model_xgb1.fit(train_text, train_labels.iloc[:, -3])
xgb_pred1 = model_xgb1.predict(test_text)
# 拟合score2
model_xgb2.fit(train_text, train_labels.iloc[:, -2])
xgb_pred2 = model_xgb2.predict(test_text)
xgb_pred = xgb_pred1 + xgb_pred2

# ③ RandomForestRegressor模型
model_rfg1 = RandomForestRegressor(n_estimators=30, min_samples_split=90, min_samples_leaf=10,max_depth=8, random_
model_rfg2 = RandomForestRegressor(n_estimators=30, min_samples_split=90, min_samples_leaf=10,max_depth=9, random_
# 拟合score1
model_rfg1.fit(train_text, train_labels.iloc[:, -3])
rfg_pred1 = model_rfg1.predict(test_text)
# 拟合score2
model_rfg2.fit(train_text, train_labels.iloc[:, -2])
rfg_pred2 = model_rfg2.predict(test_text)
rfg_pred = rfg_pred1 + rfg_pred2

# ④ GradientBoostingRegressor模型
model_gb1 = GradientBoostingRegressor(n_estimators=40,min_samples_split=100,min_samples_leaf=20,max_depth=3, subsa
model_gb2 = GradientBoostingRegressor(n_estimators=40,min_samples_split=450,min_samples_leaf=5,max_depth=4, subsam
# 拟合score1
model_gb1.fit(train_text, train_labels.iloc[:, -3])
gb_pred1 = model_gb1.predict(test_text)
# 拟合score2
model_gb2.fit(train_text, train_labels.iloc[:, -2])
gb_pred2 = model_gb2.predict(test_text)
gb_pred = gb_pred1 + gb_pred2
```

下面的preds_test1表示分别fit--score1,score2，然后求和；preds_test2表示直接fit--score。 对于单个模型,stacking模型以及模型求得结果后取平均这三种情况，分别测试了preds_test1 和 preds_test2；结果表明下述选择较好：

- 分别fit--score1,score2，然后求和
- 模型求得结果后取平均

In [27]:

```
pred = (lscv_pred + xgb_pred + rfg_pred + gb_pred)/4

print('Fit score1 + score2: the pearsonr of test set is {}'.format(pearsonr(list(test_labels.iloc[:, -1]), list(pr
```

```
Fit score1 + score2: the pearsonr of test set is 0.909544295025824
```

目录

对于给定的无label测试集,预测其结果。

In [28]:

```
# ① LassoCV模型
lscv_pred1 = model_lscv.predict(test_features)
lscv_pred2 = model_lscv.predict(test_features)
lscv_pred = lscv_pred1 + lscv_pred2

# ② XGBRegressor模型
xgb_pred1 = model_xgb1.predict(test_features)
xgb_pred2 = model_xgb2.predict(test_features)
xgb_pred = xgb_pred1 + xgb_pred2

# ③ RandomForestRegressor模型
rfg_pred1 = model_rfg1.predict(test_features)
rfg_pred2 = model_rfg2.predict(test_features)
rfg_pred = rfg_pred1 + rfg_pred2

# ④ GradientBoostingRegressor模型
gb_pred1 = model_gb1.predict(test_features)
gb_pred2 = model_gb2.predict(test_features)
gb_pred = gb_pred1 + gb_pred2
```

In [31]:

```
preds = (lscv_pred + xgb_pred + rfg_pred + gb_pred)/4
```

In [95]:

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

In [96]:

```
!pwd && ls && head -n 5 submission_sample
```

```
/home/kesci/work
kesci_submit  submission_sample
1201,8.30945762304
1202,8.50035484178
1203,8.35197087695
1204,7.93432920771
1205,4.6458501977
```

In [97]:

```
!wget -nv -O kesci_submit https://cdn.kesci.com/submit_tool/v1/kesci_submit&&chmod +x kesci_submit
```

```
2018-02-09 13:53:28 URL:https://cdn.kesci.com/submit_tool/v1/kesci_submit [7840472/7840472] -> "kesci_submit" [1]
```

In [98]:

```
!./kesci_submit -token 03dffe053acbe945 -file submission_sample
```

```
Kesci Submit Tool
Working...
Success.
OK
```

3. 回顾与反思

【不足】

- 特征提取方面：主题相关性特征没有提取
- 结果分析方面：！！错例反思！！直播时老师有提到，因为时间原因，没来得及做。

【问题】

[有关特征选取]在特征选取方面,通过附录-特征分析，考虑

- 单个特征与score相关系数
- 单特征应用于模型的pearsonr

然后滤除掉pearsonr较差的特征,发现测试结果并不好。 那么有没有特征

目录

4. 附录

1. 特征分析

1）特征与label的相关系数

In [9]:

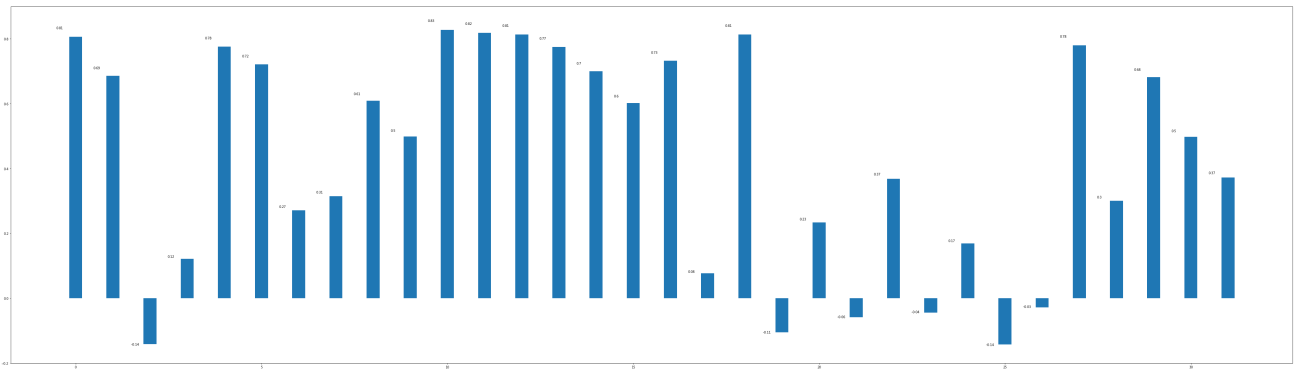
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
```

In [10]:

```
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        plt.text(rect.get_x()-rect.get_width(), 1.03*height, '%s' % round(height,2))
```

In [17]:

```
# 读取数据
train_data = pd.read_csv('/home/kesci/input/features6663/train_data.csv')

# 载入train和test数据集
dataSet = train_data.iloc[:, :-3]
labelSet = train_data.iloc[:,-3:]

y_label = []
for i in range(dataSet.shape[1]):
    y_label.append(pearsonr(list(labelSet.iloc[:, -1]), list(dataSet.iloc[:, i]))[0])

plt.figure(figsize=(70, 20))
# plt.xticks(range(len(y_label)),dataSet.columns,size='small',rotation=90)
rect = plt.bar(range(len(y_label)),y_label,width = 0.35)
plt.ylim(-0.2, 0.9)

autolabel(rect)
plt.show()
```



2）单个特征应用于模型的效果

In [21]:

```
def model_select(model):
    model.fit(train_text, train_labels.iloc[:, -3])
    preds1_test = model.predict(test_text)
    model.fit(train_text, train_labels.iloc[:, -2])
    preds2_test = model.predict(test_text)
    preds_test1 = preds1_test + preds2_test

    model.fit(train_text, train_labels.iloc[:, -1])
    preds_test2 = model.predict(test_text)
    return preds_test1,preds_test2
```

In [23]:

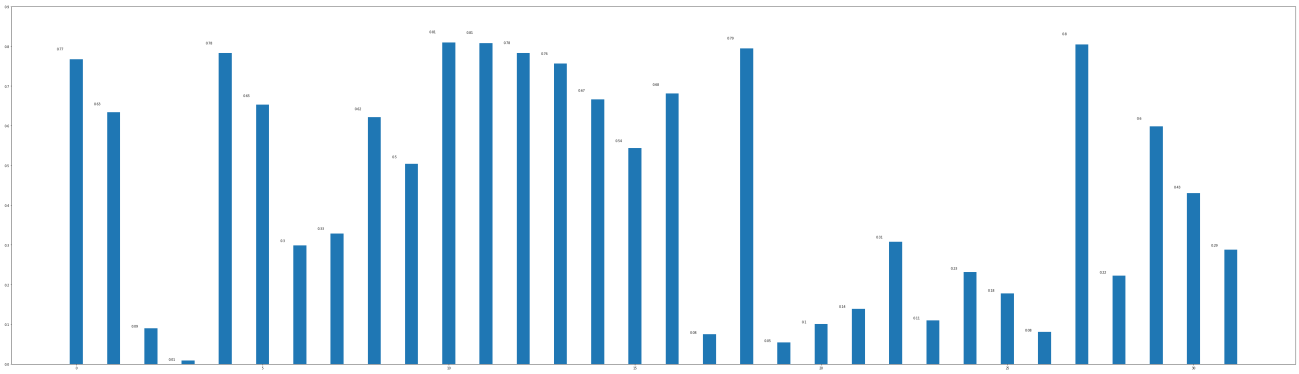
```
y_label = []
for i in range(test_features.shape[1]):
    train_text, test_text, train_labels, test_labels = train_test_split(dataSet.iloc[:,i], labelSet, test_size=0.3
```

目录

```
    train_text = np.mat(train_text).T
    test_text = np.mat(test_text).T
    # ========================================================================
    # 模型选择
    # ========================================================================
    model_lscv = LassoCV()
    model_xgb = XGBRegressor()
    model_rfg = RandomForestRegressor()
    model_gb = GradientBoostingRegressor()

    preds1_test1,preds1_test2 = model_select(model_lscv)
    preds2_test1,preds2_test2 = model_select(model_xgb)
    preds3_test1,preds3_test2 = model_select(model_rfg)
    preds4_test1,preds4_test2 = model_select(model_gb)

    pred1 = (preds1_test1+preds2_test1+preds3_test1+preds4_test1)/4
    pred2 = (preds1_test2+preds2_test2+preds3_test2+preds4_test2)/4
#     print('-----第%s个特征-------'%i)
#     print('Fit score1 + score2: The pearsonr of test set is {}'.format(pearsonr(list(test_labels.iloc[:, -1]), l
#     print('Only fit score: the pearsonr of test set is {}'.format(pearsonr(list(test_labels.iloc[:, -1]), list(p

    y_label.append(pearsonr(list(test_labels.iloc[:, -1]), list(pred1))[0])
```

In [25]:

```
plt.figure(figsize=(70, 20))
# plt.xticks(range(len(y_label)),dataSet.columns,size='small',rotation=30)
rect = plt.bar(range(len(y_label)),y_label,width = 0.35)
plt.ylim(0, 0.9)

autolabel(rect)
plt.show()
```



从上面两图可以看出,所求的两种相关性总体趋势是相同的，所以去除pearsonr<0.15的特征,测试了一下 --> 结果并没有变好。

2. 参考资料

【论文】

http://www.cs.cmu.edu/~norii/pub/aes.pdf (http://www.cs.cmu.edu/~norii/pub/aes.pdf)

http://cs229.stanford.edu/proj2012/MahanaJohnsApte-AutomatedEssayGradingUsingMachineLearning.pdf (http://cs229.stanford.edu/proj2012/MahanaJohnsApte-AutomatedEssayGradingUsingMachineLearning.pdf)

http://www.aclweb.org/anthology/D/D13/D13-1180.pdf (http://www.aclweb.org/anthology/D/D13/D13-1180.pdf)

【Github】

https://help.github.com/articles/basic-writing-and-formatting-syntax/ (https://help.github.com/articles/basic-writing-and-formatting-syntax/)

https://github.com/SahilC/AutomaticEssayGrading/tree/master/src (https://github.com/SahilC/AutomaticEssayGrading/tree/master/src)

https://github.com/adamcsvarga/essay_scoring (https://github.com/adamcsvarga/essay_scoring)

https://github.com/RDulepet19/AES (https://github.com/RDulepet19/AES)

【其他(基础知识学习)】

利用NLTK进行分句分词 (http://blog.csdn.net/baidu_27438681/article/details/60468848)

拼写检查 (http://www.jb51.net/article/64895.htm)

拼写检查 (http://norvig.com/spell-correct.html)

目
录

目
录

和鲸社区公众号

目

录