目录

W2_110常永炷

5 939 23

biorad_week2

Biorad LV2 1

Fork 点赞

内容 数据集 Fork记录 23 评论 4

内容

版本 3 2018-02-08 19:51

目录

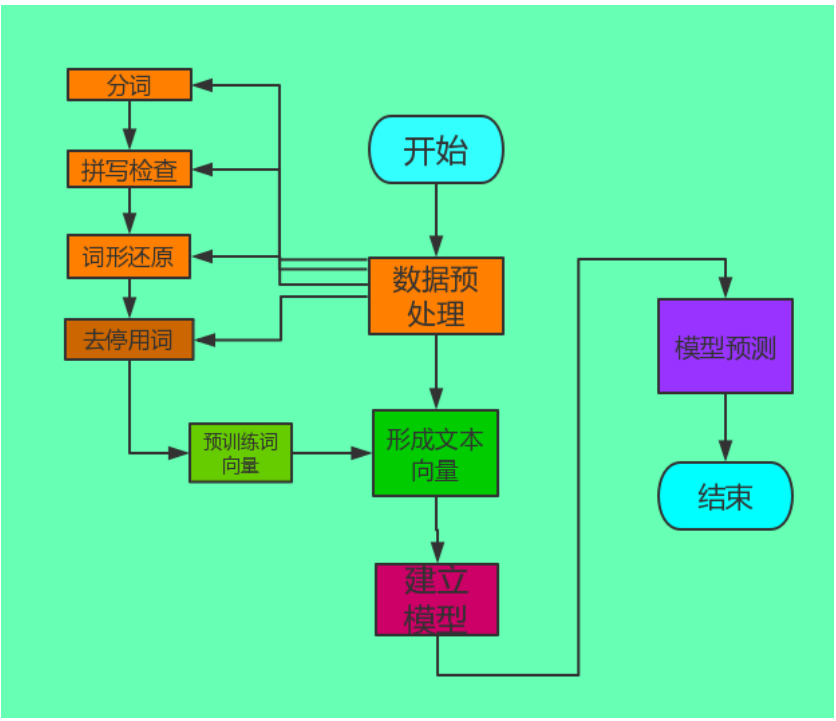
1. 总体想法
2. 探索数据
3. 数据处理
4. 构建深度网络模型
5. 利用深度模型抽取特征
6. 构建"胶囊"网络 + RCNN
7. 本周任务总结
8. 长远的思考~
9. 参考内容
10. 好未来AI Lab 训练营总结

W2_110常永炷

Fork 点赞

1. 总体想法

这是训练营的第二周的任务《自动作文打分》，首先，说明一下这个任务的数据集。在这个任务中的数据集主要内容是给一篇英语作文打分，数据集中包括两个老师的打分的情况以及总分情况；如果按照平常的NLP的任务来提供解决方案的话，这个任务算是比较简单的。通常的处理流程是:



如果按照上图所表述的来看，在这个任务中关键部分就是转化文本向量部分，这个可以使用预训练的词向量进行解决。
 有个大佬说过：**“赛题方发布赛题的主要目的是为了别人能更好的复现你的模型，而不是你的模型只能你自己复现”!!**
 因此在这个任务中，我打算试着使用容易复现的方式来完成这个解决方案，并且在基础上试着去创新一下，如果最后达到的效果是既容易复现又可以有不错的成绩；那就是本任务的成功的地方吧！接下来说一下我的解决方案的大体流程:

- 使用google预训练的词向量进行一个baseline的设计(仿照论文基于深度学习方法的评分系统 (http://www.aclweb.org/old_anthology/D/D16/D16-1193.pdf" 论文")中的实现)
- 试着修改其中的网络，以便得到较好的结果。
- 试着将上述网络的倒数第二层的权重放入机器学习模型中进行预测。
- 试着利用“胶囊”网络来进行探索。
 总上所述，在本任务中，我主要是进行深度模型的探索，对机器学习的解决方案不多涉及。

2. 探索数据

In [5]:

```
# 加载数据分析常用库
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
```

In [6]:

```
trains = pd.read_csv("/home/kesci/input/Tal_SenSco/Tal_SenSco_train_data.txt",sep="\t",names=["id","art","s1","s2"
tests = pd.read_csv("/home/kesci/input/Tal_SenSco/Tal_SenSco_test_data.txt",sep="\t",names=["id","art"])
```

目录

In [7]:

```
trains.head()
```

Out[7]:

	id	art	s1	s2	score
0	1	Dear Local Newspaper, I believe the computer d...	4	4	8
1	2	Dear @CAPS1 @CAPS2, I have heard the concern o...	6	4	10
2	3	@CAPS1 you know we all like computers and we a...	4	4	8
3	4	I Believe that computers have a positive effec...	6	5	11
4	5	Dear newstimes, I really think that computers ...	5	4	9

In [8]:

```
tests.head()
```

Out[8]:

	id	art
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 ...

In [9]:

```
print("the scores's that is equal to the add of two teachers scores is:",sum(trains["s1"] + trains["s2"] == trains
trains.info()
tests.info()
```

```
the scores's that is equal to the add of two teachers scores is: 1200
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 5 columns):
id       1200 non-null int64
art      1200 non-null object
s1       1200 non-null int64
s2       1200 non-null int64
score    1200 non-null int64
dtypes: int64(4), object(1)
memory usage: 47.0+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550 entries, 0 to 549
Data columns (total 2 columns):
id     550 non-null int64
art    550 non-null object
dtypes: int64(1), object(1)
memory usage: 8.7+ KB
```

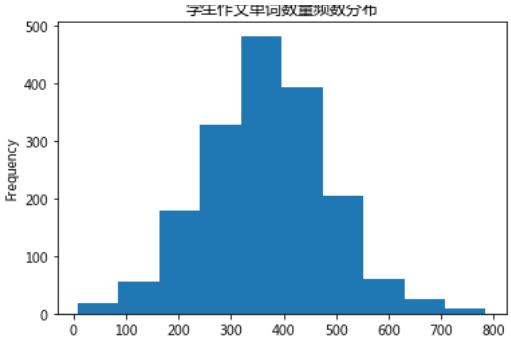
上面表示该数据没有空值，而且还能说明的是两个老师的打分的总和始终等于总分；因此就会有一个初步的想法，直接去掉两个老师打分的情况

In [10]:

```
all_data = pd.concat([trains,tests],axis=0,ignore_index=True)
all_data["len"] = all_data["art"].apply(lambda x: len(x.split()))
all_data["len"].plot(kind="hist",title = "学生作文单词数量频数分布")
all_data["len"].describe()
```

Out[10]:

```
count    1750.000000
mean      365.978857
std       119.366175
min         8.000000
25%       288.250000
50%       365.000000
75%       441.000000
max       785.000000
Name: len, dtype: float64
```

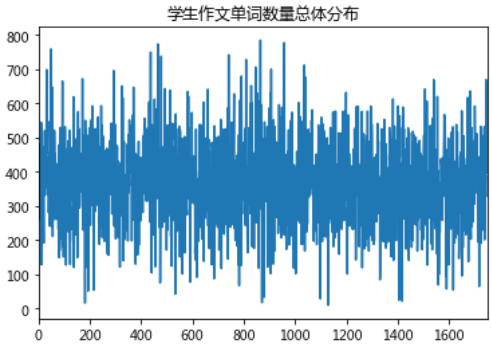


In [11]:

```
all_data["len"].plot(title = "学生作文单词数量总体分布")
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e26ac6c50>
```

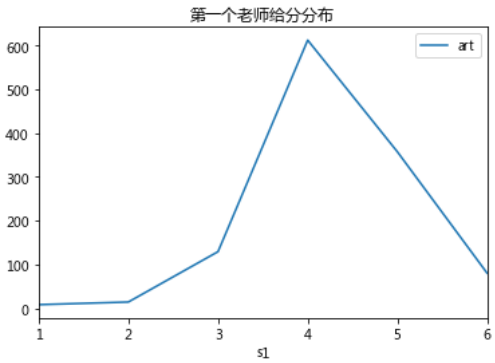


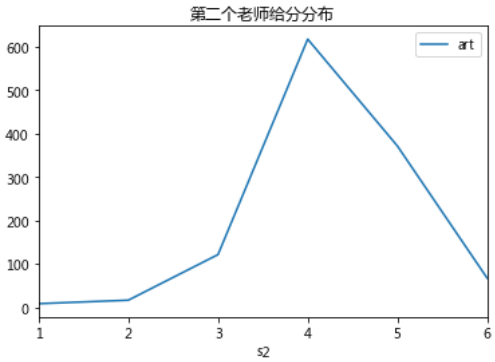
In [12]:

```
trains[["art",'s1']].groupby(["s1"]).count().plot(title="第一个老师给分分布")
trains[["art","s2"]].groupby(["s2"]).count().plot(title="第二个老师给分分布")
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6dfd3a6780>
```





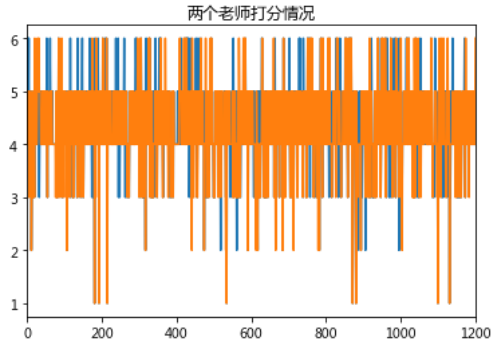
In [13]:

```
trains["s1"].plot()
trains["s2"].plot(title="两个老师打分情况")
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6dfd340dd8>
```

目录

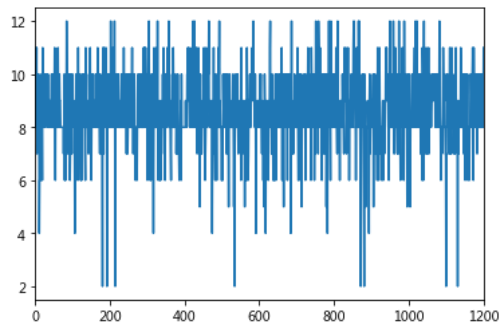


经过上面分析可以知道，两个老师在训练集中的打分数据分布是一样的，比如4分都是给的最多的，几乎在分值上的分布属于同分布；在下面的图中可以分析出，两个老师对于作文的要求是不一样的，在图中明显可以看到其中一个老师给了6分，但是另外一个老师却是小于4分的

In [14]:

```
trains["score"].plot()
print("the all score's describe:\n",trains["score"].describe())
print("the s1 describe:\n",trains["s1"].describe())
print("the s2 describe:\n",trains["s2"].describe())
```

```
the all score's describe:
 count    1200.000000
mean        8.553333
std         1.531593
min         2.000000
25%         8.000000
50%         8.000000
75%        10.000000
max        12.000000
Name: score, dtype: float64
the s1 describe:
 count    1200.000000
mean        4.280000
std         0.836969
min         1.000000
25%         4.000000
50%         4.000000
75%         5.000000
max         6.000000
Name: s1, dtype: float64
the s2 describe:
 count    1200.000000
mean        4.273333
std         0.820071
min         1.000000
25%         4.000000
50%         4.000000
75%         5.000000
max         6.000000
Name: s2, dtype: float64
```



In [15]:

```
trains["dif"] = trains.apply(lambda x: 1 if (x["s1"] > 4 and x["s2"] < 4) or (x["s1"] < 4 and x["s2"] > 4) else 0,
```

In [16]:

```
trains["dif"].sum()
```

Out[16]:

```
2
```

由上面分析可知，基本上可以断定，两个老师的水平差距不大，对学生作文的理解差不多；在上面作文描述中出现了许许多多的没有意义的单词，比如：@CAPS1、@PERCENT1等，因此在进行数据处理的时候需要考虑

目录

3. 数据处理

经过对数据的初步探索发现，两个老师的水平差距不大，打分分布比较类似，因此，在接下来的训练深度学习模型的过程中，对训练集只取总成绩当作label值，接下来对数据的处理采用以下几个步骤：

1. 对作文进行分词处理。
2. 去除非单词部分和停用词
3. 提取词干\形态还原
4. 使用google预训练词向量将文章采用拼接的方式转化为文档向量

In [17]:

```
all_data = pd.concat([trains[["id","art","score"]], tests[["id","art"]]],axis=0, ignore_index=True)
all_data.shape
```

Out[17]:

```
(1750, 3)
```

3.0 修改拼写错误的单词

In [18]:

```
# %%time
# from textblob import TextBlob
# def modify_error_words(sentence):
#     b = TextBlob(sentence)
#     return str(b.correct())
# from multiprocessing import Pool
# pool = Pool(processes=20)
# all_data["art"] = all_data["art"].str.lower()
# all_data["art_fu"] = pool.map(modify_error_words, all_data["art"])
```

3.1 分词以及去除非单词字符

In [19]:

```
all_data["art"] = all_data["art"].str.lower()
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'[a-zA-Z]+')
all_data["art_split"] = all_data["art"].apply(tokenizer.tokenize)
tokenizer = RegexpTokenizer(r'@[a-zA-Z]+')
all_data["non_words"] = all_data["art"].apply(tokenizer.tokenize)
non_list = []
for x in list(all_data["non_words"].values):
    non_list.extend(x)
non_list = list(set(non_list))
non_list = [x[1:] for x in non_list]
all_data["art_split"] = all_data["art_split"].apply(lambda x: [w for w in x if w not in non_list])
```

In [20]:

```
all_data.head()
```

Out[20]:

	art	id	score	art_split	non_words
0	dear local newspaper, i believe the computer d...	1	8.0	[dear, local, newspaper, i, believe, the, comp...	[@percent, @num, @caps, @num]
1	dear @caps1 @caps2, i have heard the concern o...	2	10.0	[dear, i, have, heard, the, concern, of, many,...	[@caps, @caps, @date, @organization, @person]
2	@caps1 you know we all like computers and we a...	3	8.0	[you, know, we, all, like, computers, and, we,...	[@caps, @caps]
3	i believe that computers have a positive effec...	4	11.0	[i, believe, that, computers, have, a, positiv...	[@caps, @organization, @organization, @caps, @...
4	dear newstimes, i really think that computers ...	5	9.0	[dear, newstimes, i, really, think, that, comp...	[@percent, @caps, @caps, @percent, @location, ...

3.2 去除停用词

3.2 去除停用词

In [21]:

```
%%time
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
all_data["art_split"] = all_data["art_split"].apply(lambda x: [w for w in x if w not in stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /home/kesci/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
CPU times: user 1min 28s, sys: 6.25 s, total: 1min 34s
Wall time: 1min 37s
```

In [22]:

```
all_data.head()
```

Out[22]:

	art	id	score	art_split	non_words
0	dear local newspaper, i believe the computer d...	1	8.0	[dear, local, newspaper, believe, computer, po...	[@percent, @num, @caps, @num]
1	dear @caps1 @caps2, i have heard the concern o...	2	10.0	[dear, heard, concern, many, scientists, compu...	[@caps, @caps, @date, @organization, @person]
2	@caps1 you know we all like computers and we a...	3	8.0	[know, like, computers, use, different, tyoes,...	[@caps, @caps]
3	i believe that computers have a positive effec...	4	11.0	[believe, computers, positive, effect, people,...	[@caps, @organization, @organization, @caps, @...
4	dear newstimes, i really think that computers ...	5	9.0	[dear, newstimes, really, think, computers, go...	[@percent, @caps, @caps, @percent, @location, ...

3.4 获取google的预训练词向量

在这个步骤中将为生成训练的数据集和测试的数据集

In [23]:

```
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
from tqdm import tqdm
def get_vec_from_file():
    google_vec = {}
    found = 0
    with open("/home/kesci/input/w2v2140/txt2vec.txt",mode="r",encoding="utf-8") as rf:
        lines = rf.readlines()
        for line in lines:
            k,v = line.split("_split_tag_")
            google_vec[k] = np.array(eval(v))
    return google_vec

def get_google_vec(word_vectors):
    """
    根据词向量模型产生word2index, 和词向量矩阵
    :return:
    """
    embeddings_index = {}
    word2index = {}
    print("get word2index...")
    i = 0
    for word,vocab_obj in tqdm(word_vectors.items()):
        word2index[word] = i + 2
        embeddings_index[word] = word_vectors[word]
        i += 1
    word2index["PAD"] = 0
    word2index["UNK"] = 1
    index2word = {v : k for k,v in word2index.items()}
    print("found %s word vectors." % len(embeddings_index.get(index2word.get(2))))
    print("found %s words. " % len(word2index))
    embedding_matrix = np.zeros((len(word2index),len(embeddings_index.get(index2word.get(5)))))
    print("begin generate word matrix\n")
    for word,i in tqdm(word2index.items()):
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            # 文本数据中的词在词向量字典中没有, 向量为取0; 如果有则取词向量中该词的向量
            embedding_matrix[i] = embedding_vector
```

```
        embedding_matrix[i] = embedding_vector
    return embedding_matrix, len(word2index), word2index, index2word

def load_train_data(train_data,word2index,max_len = 800):
    contents = train_data.art_split.values
    labels = train_data.score.values
    X = np.empty(train_data.shape[0],dtype=list)
    print("the train2index's shape is: ", X.shape)
    label = []
    print("\nbegin covert text to index\n")
    for i in tqdm(range(len(contents))):
        seg_word_list = contents[i]
        seqs = []
        for w in seg_word_list:
            if w in word2index:
                seqs.append(word2index[w])
            else:
                seqs.append(word2index["UNK"])
        X[i] = seqs
        if i < len(labels):
            label.append(labels[i])
    print("finish covert text to index..")
    train_labels = np.array(label)
    print("the label's shape: ",labels.shape)
    trains = sequence.pad_sequences(X,maxlen=max_len)
    print("split data")
    print("the train's shape is {0}".format(trains.shape))
    x_train,x_test,y_train,y_test = train_test_split(trains,labels,test_size=0.2,random_state=42)
    print('Shape of x_train tensor:',x_train.shape)
    print('Shape of y_train tensor:',y_train.shape)
    print('Shape of x_test tensor:',x_test.shape)
    print('Shape of y_test tensor:',y_test.shape)
    return x_train,x_test,y_train,y_test,trains,labels

def load_test_data(test_data,word2index,max_len=800):
    contents = test_data.art_split.values
    id_list = list(test_data["id"])
    X = np.empty(test_data.shape[0],dtype=list)
    print("the test2index's shape is: ", X.shape)
    print("\nbegin covert text to index\n")
    for i in tqdm(range(len(contents))):
        seg_word_list = contents[i]
        seqs = []
        for w in seg_word_list:
            if w in word2index:
                seqs.append(word2index[w])
            else:
                seqs.append(word2index["UNK"])
        X[i] = seqs
    print("finish covert text to index..")
    tests = sequence.pad_sequences(X,maxlen=max_len)
    return tests, id_list
```

```
Using TensorFlow backend.
```

4. 构建深度网络模型

In [24]:

```
# 导入相应的包
from keras.layers import Embedding,Input, Convolution1D, MaxPooling1D, Flatten, Dense, Bidirectional,concatenate,G
from keras.models import Model,load_model
from keras.layers import Conv1D
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.models import load_model
from keras.optimizers import Adam
```

In [25]:

```
def RCNN(vocab_size, embedding_matrix,ckpt_path,max_len,embedding_size):
    main_input = Input(shape=(max_len,))
    embed = Embedding(vocab_size,embedding_size,input_length=max_len, weights=[embedding_matrix],trainable=True)(m
    cnn = Convolution1D(256,3,padding='same',strides=1,activation='relu')(embed)
    cnn = MaxPooling1D(pool_size=4)(cnn)
    cnn = Flatten()(cnn)
    cnn = Dense(128)(cnn)
    rnn = LSTM(256)(embed)
    rnn = Dense(128)(rnn)
    con = concatenate([cnn,rnn],axis=-1)
    den = Dense(32)(con)
    den = Dropout(0.5)(den)
```

```
    main_output = Dense(1,activation='relu')(den)
    model = Model(inputs=main_input,outputs=main_output)
    adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
    print(model.summary())
    try:
        model.load_weights(ckpt_path)
        print("load weights finish....")
    except:
        print("no pre-weights...")
    model.compile(loss='mse', optimizer=adam, metrics=['mse'])
    return model
```

In [25]:

```
def train_model(vocab_size, x_train, y_train, x_test, y_test, embedding_matrix,
                batch_size,epoch,ckpt,model_file,max_len,embedding_size):
    """
    训练模型
    :param vocab_size:
    :param x_train:
    :param y_train:
    :param x_test:
    :param y_test:
    :param embedding_matrix:
    :return:
    """
    callback = ModelCheckpoint(ckpt, verbose=1,save_best_only=True,save_weights_only=True)
    early_stop = EarlyStopping(monitor='val_mse',patience=5,verbose=1)
    reduce_lr = ReduceLROnPlateau(monitor='val_mse',factor=0.2,
                                  patience=5,min_lr=0.001)
    callback_list = [callback,early_stop,reduce_lr]
    try:
        model = load_model(model_file)
        meritcs = model.evaluate(x_test,y_test)
        print('\ntest meritcs',meritcs)
        print("load model finish...")
    except:
        model = RCNN(vocab_size,embedding_matrix,ckpt, max_len,embedding_size)
        print("create model finish...")
    model.fit(x_train,y_train,
              batch_size=batch_size,
              callbacks=callback_list,
              shuffle=True,
              epochs=epoch,
              verbose=1,
              validation_data=(x_test,y_test))
    model.save(model_file)

def predict(tests, tests_id, vocab_size, embedding_matrix,
            batch_size,ckpt,max_len,embedding_size,sub_path):
    model = RCNN(vocab_size,embedding_matrix,ckpt,max_len,embedding_size)
    print("Created model and loaded weights from file")
    print("begin predict")
    pred = model.predict(tests,verbose=1,batch_size=batch_size)
    print("the pred's shape: ",pred.shape)
    test_pd = pd.DataFrame(tests_id,columns=["id"])
    test_pd["pred"] = pred
    print(list(pred)[0:100])
    print("test data's shape: ",test_pd.shape)
    print("predict finish...")
    test_pd["pred"].to_csv(sub_path, index=False,header=False)
    return pred
```

In [26]:

```
batch_size = 32
epoch = 5
max_len = 450
embedding_size = 300
google_vec = get_vec_from_file()
embedding_matrix, len_words, word2index, index2word = get_google_vec(google_vec)
train_data = all_data[~all_data["score"].isnull()]
test_data = all_data[all_data["score"].isnull()]
x_train,x_test,y_train,y_test,trains,labels = load_train_data(train_data,word2index,max_len = max_len)
tests, id_list = load_test_data(test_data,word2index,max_len=max_len)
train_model(len_words, x_train, y_train, x_test, y_test, embedding_matrix,
                          batch_size, epoch, "/home/kesci/work/rcnn.hdf5", "/home/kesci/work/model",max_len,embedd
```

```
100%|██████████| 10569/10569 [00:00<00:00, 814104.15it/s]
100%|██████████| 10571/10571 [00:00<00:00, 320000.49it/s]
100%|██████████| 1200/1200 [00:00<00:00, 17751.60it/s]
100%|██████████| 550/550 [00:00<00:00, 18781.13it/s]
```

目录

```
get word2index...
found 300 word vectors.
found 10571 words.
begin generate word matrix

the train2index's shape is:  (1200,)

begin covert text to index

finish covert text to index..
the label's shape:  (1200,)
split data
the train's shape is (1200, 450)
Shape of x_train tensor: (960, 450)
Shape of y_train tensor: (960,)
Shape of x_test tensor: (240, 450)
Shape of y_test tensor: (240,)
the test2index's shape is:  (550,)

begin covert text to index

finish covert text to index..


__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 450)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 450, 300)     3171300     input_1[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 450, 256)     230656      embedding_1[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 112, 256)     0           conv1d_1[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 28672)        0           max_pooling1d_1[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 256)          570368      embedding_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 128)          3670144     flatten_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 128)          32896       lstm_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 256)          0           dense_1[0][0]
                                                                 dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 32)           8224        concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 32)           0           dense_3[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1)            33          dropout_1[0][0]
==================================================================================================
Total params: 7,683,621
Trainable params: 7,683,621
Non-trainable params: 0
__________________________________________________________________________________________________
None
load weights finish....
create model finish...
Train on 960 samples, validate on 240 samples
Epoch 1/5
928/960 [============================>.] - ETA: 31s - loss: 4.2300 - mean_squared_error: 4.2300 Epoch 00001: val_los
960/960 [==============================] - 954s 994ms/step - loss: 4.1834 - mean_squared_error: 4.1834 - val_loss: 1

/opt/conda/lib/python3.5/site-packages/keras/callbacks.py:493: RuntimeWarning: Early stopping conditioned on metric
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning
/opt/conda/lib/python3.5/site-packages/keras/callbacks.py:889: RuntimeWarning: Reduce LR on plateau conditioned on m
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning

Epoch 2/5
928/960 [============================>.] - ETA: 31s - loss: 3.5936 - mean_squared_error: 3.5936 Epoch 00002: val_los
960/960 [==============================] - 951s 991ms/step - loss: 3.5737 - mean_squared_error: 3.5737 - val_loss: 1
Epoch 3/5
928/960 [============================>.] - ETA: 30s - loss: 3.4574 - mean_squared_error: 3.4574 Epoch 00003: val_los
960/960 [==============================] - 938s 977ms/step - loss: 3.4398 - mean_squared_error: 3.4398 - val_loss: 6
Epoch 4/5
928/960 [============================>.] - ETA: 30s - loss: 3.1854 - mean_squared_error: 3.1854 Epoch 00004: val_lo
960/960 [==============================] - 927s 965ms/step - loss: 3.2471 - mean_squared_error: 3.2471 - val_loss: 1
Epoch 5/5
928/960 [============================>.] - ETA: 30s - loss: 3.2661 - mean_squared_error: 3.2661 Epoch 00005: val_lo
960/960 [==============================] - 928s 967ms/step - loss: 3.2913 - mean_squared_error: 3.2913 - val_loss: 1
```

In [28]:

In [28]:

```
!ls /home/kesci/work/
```

```
kesci_submit  model  rcnn.hdf5  submission.csv  submission_sample
```

目录

In [52]:

```
from keras.models import Model,load_model
import scipy as sc
import numpy as np
model = RCNN(len_words,embedding_matrix,"/home/kesci/work/rcnn.hdf5",max_len,embedding_size)
meritcs = model.predict(x_test,verbose=1,batch_size=batch_size)
val_pd = pd.DataFrame(meritcs, columns=["pred"])
val_pd["pred"] = val_pd["pred"].apply(lambda x: np.around(x))
print("the test's pearsonr is: ", sc.stats.pearsonr(y_test, val_pd["pred"])[0])
print("load model finish...")
test_preds = model.predict(tests,verbose=1,batch_size=batch_size)
test_pd = pd.DataFrame(test_preds, columns=["pred"])
test_pd["id"] = id_list
test_pd["pred"] = test_pd["pred"].apply(lambda x: np.around(x))
test_pd[["id","pred"]].to_csv("submission_sample",index=False, header=False)
```

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_7 (InputLayer)            (None, 450)          0
__________________________________________________________________________________________________
embedding_7 (Embedding)         (None, 450, 300)     3171300     input_7[0][0]
__________________________________________________________________________________________________
conv1d_7 (Conv1D)               (None, 450, 256)     230656      embedding_7[0][0]
__________________________________________________________________________________________________
max_pooling1d_7 (MaxPooling1D)  (None, 112, 256)     0           conv1d_7[0][0]
__________________________________________________________________________________________________
flatten_7 (Flatten)             (None, 28672)        0           max_pooling1d_7[0][0]
__________________________________________________________________________________________________
lstm_7 (LSTM)                   (None, 256)          570368      embedding_7[0][0]
__________________________________________________________________________________________________
dense_25 (Dense)                (None, 128)          3670144     flatten_7[0][0]
__________________________________________________________________________________________________
dense_26 (Dense)                (None, 128)          32896       lstm_7[0][0]
__________________________________________________________________________________________________
concatenate_7 (Concatenate)     (None, 256)          0           dense_25[0][0]
                                                                 dense_26[0][0]
__________________________________________________________________________________________________
dense_27 (Dense)                (None, 32)           8224        concatenate_7[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 32)           0           dense_27[0][0]
__________________________________________________________________________________________________
dense_28 (Dense)                (None, 1)            33          dropout_7[0][0]
==================================================================================================
Total params: 7,683,621
Trainable params: 7,683,621
Non-trainable params: 0
__________________________________________________________________________________________________
None
load weights finish....
240/240 [==============================] - 9s 36ms/step
the test's pearsonr is:  0.788163666944
load model finish...
550/550 [==============================] - 19s 35ms/step
```

In [34]:

```
test_pd["pred"].describe()
test_pd["pred"].plot()
```

In [38]:

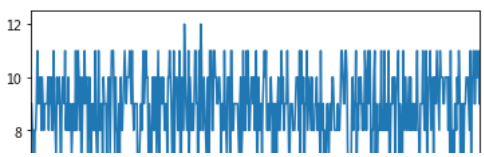
```
!pwd && ls && head -n 5 submission_sample
```

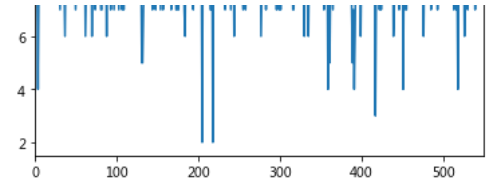
In [44]:

```
!wget -nv -O kesci_submit https://cdn.kesci.com/submit_tool/v1/kesci_submit&&chmod +x kesci_submit
!./kesci_submit -token e03990204bf0de93 -file submission_sample
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0cd6d54e0>
```





目录

In [45]:

```
/home/kesci/work
kesci_submit  model  rcnn.hdf5  submission.csv  submission_sample
1201,9.0
1202,9.0
1203,8.0
1204,8.0
1205,4.0
```

In [46]:

```
2018-02-07 02:18:13 URL:https://cdn.kesci.com/submit_tool/v1/kesci_submit [7840472/7840472] -> "kesci_submit" [1]
Kesci Submit Tool
Working...
Failed.
Serevr Response:
 400 - 比赛信息错误
```

5. 利用深度模型抽取特征

5.1 抽取模型特征

In [37]:

```
import numpy as np
from keras.models import load_model, Model
batch_size = 32
epoch = 5
max_len = 450
embedding_size = 300
def extract_nn_feature():
    google_vec = get_vec_from_file()
    embedding_matrix, len_words, word2index, index2word = get_google_vec(google_vec)
    model = RCNN(len_words,embedding_matrix,"/home/kesci/work/rcnn.hdf5",max_len,embedding_size)
    model = Model(model.layers[0].input, model.layers[-2].output)
    adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
    model.compile(loss='mse', optimizer=adam, metrics=['mse'])
    print(model.summary())
    train_data = all_data[~all_data["score"].isnull()]
    test_data = all_data[all_data["score"].isnull()]
    x_train,x_test,y_train,y_test,trains,labels = load_train_data(train_data,word2index,max_len = max_len)
    tests, id_list = load_test_data(test_data,word2index,max_len=max_len)
    shuffled_indices = np.random.permutation(trains.shape[0])
    trains = trains[shuffled_indices]
    labels = labels[shuffled_indices]
    predictions = model.predict(tests, batch_size=32, verbose=1)
    np.save('/home/kesci/work/test-feats.npy', predictions)
    np.savetxt('/home/kesci/work/test-id.txt', id_list)
    predictions = model.predict(trains, batch_size=32, verbose=1)
    np.save('/home/kesci/work/train-feats.npy', predictions)
    np.savetxt('/home/kesci/work/train-labels.txt', labels)
# extract_nn_feature()
```

```
100%|██████████████| 10569/10569 [00:00<00:00, 840212.26it/s]
100%|██████████████| 10571/10571 [00:00<00:00, 413219.02it/s]

get word2index...
found 300 word vectors.
found 10571 words.
begin generate word matrix


  0%|          | 0/1200 [00:00<?, ?it/s]

__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 450)          0
```

```
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 450, 300)     3171300     input_2[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 450, 256)     230656      embedding_2[0][0]
__________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 112, 256)     0           conv1d_2[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 28672)        0           max_pooling1d_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   (None, 256)          570368      embedding_2[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 128)          3670144     flatten_2[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 128)          32896       lstm_2[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 256)          0           dense_5[0][0]
                                                                 dense_6[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 32)           8224        concatenate_2[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 32)           0           dense_7[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 1)            33          dropout_2[0][0]
==================================================================================================
Total params: 7,683,621
Trainable params: 7,683,621
Non-trainable params: 0
__________________________________________________________________________________________________
None
load weights finish....
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, 450)          0
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 450, 300)     3171300     input_2[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 450, 256)     230656      embedding_2[0][0]
__________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 112, 256)     0           conv1d_2[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 28672)        0           max_pooling1d_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   (None, 256)          570368      embedding_2[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 128)          3670144     flatten_2[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 128)          32896       lstm_2[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 256)          0           dense_5[0][0]
                                                                 dense_6[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 32)           8224        concatenate_2[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 32)           0           dense_7[0][0]
==================================================================================================
Total params: 7,683,588
Trainable params: 7,683,588
Non-trainable params: 0
__________________________________________________________________________________________________
None
the train2index's shape is:  (1200,)

begin covert text to index


100%|██████████████████| 1200/1200 [00:00<00:00, 7691.75it/s]
100%|██████████████████| 550/550 [00:00<00:00, 19615.38it/s]

finish covert text to index..
the label's shape:  (1200,)
split data
the train's shape is (1200, 450)
Shape of x_train tensor: (960, 450)
Shape of y_train tensor: (960,)
Shape of x_test tensor: (240, 450)
Shape of y_test tensor: (240,)
the test2index's shape is:  (550,)

begin covert text to index

finish covert text to index..
```

```
550/550 [==============================] - 20s 37ms/step
1200/1200 [==============================] - 41s 34ms/step
```

5.2 深度网络特征+机器学习模型

In [86]:

```
class model:
    from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
    def __init__(self,X_train,X_test,y_train,y_test):
        self.X_train = X_train
        self.X_test = X_test
        self.y_train = y_train
        self.y_test = y_test
    def svr(self, c=5, g = 3):
        from sklearn.svm import SVR
        from sklearn.pipeline import make_pipeline
        from sklearn.preprocessing import RobustScaler
        import scipy as sc
#  索引最好的c = 5 gamma =1
# bag of words c = le3 gamma = 0.1
# tfidf c=2.8, g=2
        lasso = make_pipeline(SVR(C=2.1, gamma=0.005))
        lasso.fit(self.X_train,self.y_train)
        print("the model is SVR and the test's pearsonr is: ", sc.stats.pearsonr(self.y_test, lasso.predict(self.X
        return lasso
    def rf(self):
        from sklearn.pipeline import make_pipeline
        from sklearn.ensemble import RandomForestRegressor
        import scipy as sc
        rf = make_pipeline(RandomForestRegressor(random_state=590,n_estimators =6))
        rf.fit(self.X_train,self.y_train)
        print("the model is rf and the test's pearsonr is: ", sc.stats.pearsonr(self.y_test, rf.predict(self.X_tes
        return rf
    def gboost(self):
        from sklearn.ensemble import GradientBoostingRegressor
        import scipy as sc
        GBoost = GradientBoostingRegressor(n_estimators=500, learning_rate=0.01,
                                           max_depth=12, max_features='sqrt',
                                           min_samples_leaf=15, min_samples_split=97,
                                           loss='ls', random_state =200)
        GBoost.fit(self.X_train,self.y_train)
        print("the model is gboost and the test's pearsonr is: ", sc.stats.pearsonr(self.y_test, GBoost.predict(se
        return GBoost
    def xgboost(self):
        from xgboost import XGBRegressor
        import xgboost as xgb
        import scipy as sc
    #     for i in range(0,1000,10):
        model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=10,
                                     learning_rate=0.01, max_depth=11,
                                     min_child_weight=1.7817, n_estimators=500,
                                     reg_alpha=0.01, reg_lambda=5,
                                     subsample=0.5213, silent=1,
                                     seed =1024, nthread = -1)
        model_xgb.fit(self.X_train,self.y_train)
        print("the model is xgboost and the test's pearsonr is: ", sc.stats.pearsonr(self.y_test, model_xgb.predic
        return model_xgb
    def lgb(self):
        import lightgbm as lgb
        from lightgbm import LGBMRegressor
        import scipy as sc
        model_lgb = LGBMRegressor(objective='regression',num_leaves=5,
                                  learning_rate=0.05, n_estimators=550,
                                  max_bin = 25, bagging_fraction = 1,
                                  bagging_freq = 5, feature_fraction = 0.7,
                                  feature_fraction_seed=9, bagging_seed=9,
                                  min_data_in_leaf =42, min_sum_hessian_in_leaf = 40)
        model_lgb.fit(self.X_train,self.y_train)
        print("the model is lgb and the test's pearsonr is: ", sc.stats.pearsonr(self.y_test, model_lgb.predict(se
        return model_lgb
    def stacking(self):
        from sklearn.svm import SVR
        from sklearn.pipeline import make_pipeline
        from sklearn.preprocessing import RobustScaler,MinMaxScaler
        from sklearn.preprocessing import StandardScaler
        from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
        from xgboost import XGBRegressor
        import lightgbm as lgb
        from lightgbm import LGBMRegressor
        import xgboost as xgb
        from mlxtend.regressor import StackingRegressor
```

目录

```
        import scipy as sc
        lasso = make_pipeline(SVR(C=2.1, gamma=0.005))
        rf = make_pipeline(RandomForestRegressor(random_state=590,n_estimators =6))
        GBoost = GradientBoostingRegressor(n_estimators=500, learning_rate=0.01,
                                           max_depth=12, max_features='sqrt',
                                           min_samples_leaf=15, min_samples_split=97,
                                           loss='ls', random_state =200)
        model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=10,
                                     learning_rate=0.01, max_depth=11,
                                     min_child_weight=1.7817, n_estimators=500,
                                     reg_alpha=0.01, reg_lambda=5,
                                     subsample=0.5213, silent=1,
                                     seed =1024, nthread = -1)
        model_lgb = LGBMRegressor(objective='regression',num_leaves=5,
                                  learning_rate=0.05, n_estimators=550,
                                  max_bin = 25, bagging_fraction = 1,
                                  bagging_freq = 5, feature_fraction = 0.7,
                                  feature_fraction_seed=9, bagging_seed=9,
                                  min_data_in_leaf =42, min_sum_hessian_in_leaf = 40)
        regressors = [rf,lasso,GBoost, model_lgb,model_xgb]
        stregr = StackingRegressor(regressors=regressors, meta_regressor=model_xgb)
        stregr.fit(self.X_train,self.y_train)
        print("the model is staking and the test's pearsonr is: ", sc.stats.pearsonr(self.y_test, stregr.predict(s
        return stregr
    class _AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):

        def __init__(self, models):
            self.models = models
        # we define clones of the original models to fit the data in
        def fit(self, X, y):
            from sklearn.base import clone
            self.models_ = [clone(x) for x in self.models]
            # Train cloned base models
            for model in self.models_:
                model.fit(X, y)
            return self
        #Now we do the predictions for cloned models and average them
        def predict(self, X):
            predictions = np.column_stack([
                model.predict(X) for model in self.models_
            ])
            return np.mean(predictions, axis=1)
    def average(self):
        from sklearn.svm import SVR
        from sklearn.pipeline import make_pipeline
        from sklearn.preprocessing import RobustScaler,MinMaxScaler
        from sklearn.preprocessing import StandardScaler
        from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
        from xgboost import XGBRegressor
        import lightgbm as lgb
        from lightgbm import LGBMRegressor
        import xgboost as xgb
        from mlxtend.regressor import StackingRegressor
        from sklearn.kernel_ridge import KernelRidge
        import scipy as sc
#         self._load_package()
# c=7,g=0.075
        lasso = make_pipeline(SVR(C=2.1, gamma=0.005))
        rf = make_pipeline(RandomForestRegressor(random_state=590,n_estimators =6))
        GBoost = GradientBoostingRegressor(n_estimators=500, learning_rate=0.01,
                                           max_depth=12, max_features='sqrt',
                                           min_samples_leaf=15, min_samples_split=97,
                                           loss='ls', random_state =200)
        model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=10,
                                     learning_rate=0.01, max_depth=11,
                                     min_child_weight=1.7817, n_estimators=500,
                                     reg_alpha=0.01, reg_lambda=5,
                                     subsample=0.5213, silent=1,
                                     seed =1024, nthread = -1)
        model_lgb = LGBMRegressor(objective='regression',num_leaves=5,
                                  learning_rate=0.05, n_estimators=550,
                                  max_bin = 25, bagging_fraction = 1,
                                  bagging_freq = 5, feature_fraction = 0.7,
                                  feature_fraction_seed=9, bagging_seed=9,
                                  min_data_in_leaf =42, min_sum_hessian_in_leaf = 40)
        regressors = [rf, lasso,GBoost, model_lgb,model_xgb]
        stregr = StackingRegressor(regressors=regressors, meta_regressor=model_xgb)
        averaged_models =self._AveragingModels(models = (rf,stregr,model_xgb,lasso))
        averaged_models.fit(self.X_train,self.y_train)
        print("the model is average and the test's pearsonr is: ", sc.stats.pearsonr(self.y_test, averaged_models.
        return averaged_models
```

In [92]:

目录

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from numpy import loadtxt
import numpy as np
import pandas as pd
import pickle
import scipy as sc
import random
TRAIN_FEATURES_FILE = '/home/kesci/work/train-feats.npy'
TRAIN_LABELS_FILE = '/home/kesci/work/train-labels.txt'
TEST_FEATURES_FILE = '/home/kesci/work/test-feats.npy'
TEST_ID_FILE = "/home/kesci/work/test-id.txt"
CLASSIFIER = "svr"
MODEL_FILE = 'cnn-feats-%s.pkl' % CLASSIFIER
TRAIN = True
C = 2.1
MAX_ITER = 1000
if TRAIN:
    X_train = np.load(TRAIN_FEATURES_FILE)
    y_train = loadtxt(TRAIN_LABELS_FILE, dtype=float).astype(int)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1,random_state = 554)
    print("the train's shape is: ", X_train.shape, y_train.shape, X_val.shape, y_val.shape)
    print("the train's label's shape is:", y_train.shape)
    print("the val's shape is: ", X_val.shape)
    print("the val's label's shape is:", y_val.shape)
    m = model(X_train=X_train, y_train=y_train,X_test=X_val, y_test=y_val)
    m.gboost()
    m.lgb()
    m.rf()
    m.stacking()
    m.svr()
    m.xgboost()
    average = m.average()
    val_preds = average.predict(X_val)
    print("the model is svr and the test's pearsonr is: ", sc.stats.pearsonr(y_val, val_preds)[0], i)
    with open(MODEL_FILE, 'wb') as mf:
        pickle.dump(model, mf)
    X_test = np.load(TEST_FEATURES_FILE)
    print("the test's shape: ",X_test.shape)
    test_preds = average.predict(X_test)
    id_list = loadtxt(TEST_ID_FILE, dtype=float).astype(int)
    test_pd = pd.DataFrame(test_preds, columns=["pred"])
    test_pd["id"] = id_list
    test_pd["pred"] = test_pd["pred"].apply(lambda x: np.around(x))
    test_pd[["id","pred"]].to_csv("sub.csv",index=False, header=False)
else:
    with open(MODEL_FILE, 'rb') as mf:
        model = pickle.load(mf)
    X_test = np.load(TEST_FEATURES_FILE)
    print("the test's shape: ",X_test.shape)
    test_preds = model.predict(X_test)
    id_list = loadtxt(TEST_ID_FILE, dtype=float).astype(int)
    test_pd = pd.DataFrame(test_preds, columns=["pred"])
    test_pd["id"] = id_list
    test_pd["pred"] = test_pd["pred"].apply(lambda x: np.around(x))
    test_pd[["id","pred"]].to_csv("sub.csv",index=False, header=False)
```

```
the train's shape is:  (1080, 32) (1080,) (120, 32) (120,)
the train's label's shape is: (1080,)
the val's shape is:  (120, 32)
the val's label's shape is: (120,)
the model is gboost and the test's pearsonr is:  0.912647663056
the model is lgb and the test's pearsonr is:  0.895946637724
the model is rf and the test's pearsonr is:  0.885464841126
the model is staking and the test's pearsonr is:  0.895402460946
the model is SVR and the test's pearsonr is:  0.922354729043
the model is xgboost and the test's pearsonr is:  0.921309542697
the model is average and the test's pearsonr is:  0.914672483749
the model is svr and the test's pearsonr is:  0.914672483749 216
the test's shape:  (550, 32)
```

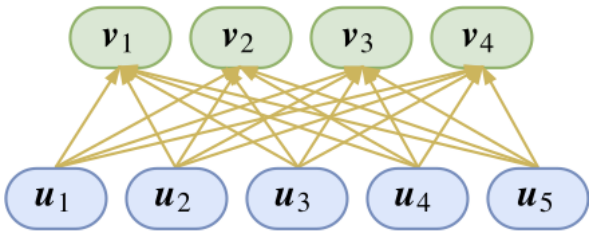
the train's shape is: (1080, 32) (1080,) (120, 32) (120,)
 the train's label's shape is: (1080,)
 the val's shape is: (120, 32)
 the val's label's shape is: (120,)
 [LibSVM]the model is svr and the test's pearsonr is: 0.873139876862
 线上0.8219

6. 构建"胶囊"网络+RCNN

6.1 Capsule的基本概念和流程

目录

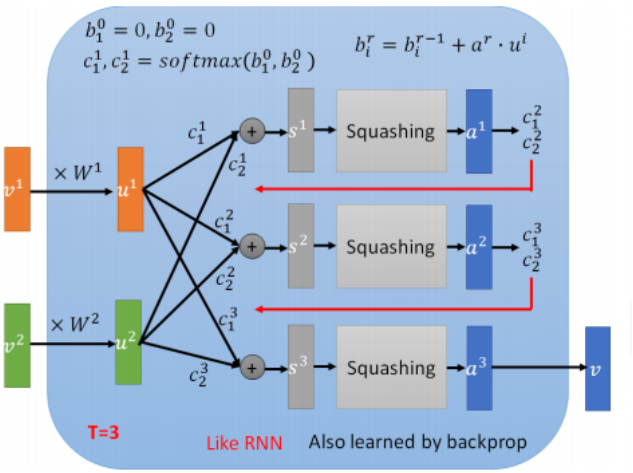
由深度学习先驱Hinton开源的Capsule论文《Dynamic Routing Between Capsules》，无疑是去年深度学习界最热点的消息之一。看过一些博客以及介绍，初步了解了这个网络。刚好趁着这个好未来AI lab训练营的机会试一下这个网络在解决具体任务中有什么独特的地方。Capsule我的理解是就像借鉴了seq2seq，相当于vector in vector out,只不过在Capsule网络中倾向于解释数据的传输，即为什么。
　　简单说一下Capsule,看这个图片:



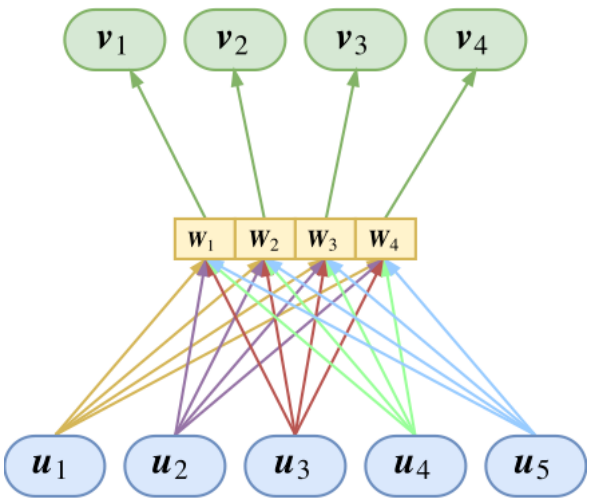
在这个图中底层的胶囊和高层的胶囊构成一些连接关系。（胶囊：引入“南苏林北陈云” 中苏大佬的说法：只要把一个向量当作一个整体来看，它就是一个“胶囊”，神经元就是标量，胶囊就是向量，就这么粗暴！）
　　Hinton希望Capsule能有的一个性质是：胶囊的模长能够代表这个特征的概率。因此在Capsule中有个squash的概念，squash在英语中有着浓缩果汁的意思。Hinton在论文中的squash是

$$squash(\boldsymbol{x}) = \frac{\|\boldsymbol{x}\|^2}{1+\|\boldsymbol{x}\|^2}\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$$

　　在论文中还出现了一个名词：动态路由，这个动态路由也是老技术新概念。就很类似于深度学习以前被称作感知机，现在叫神经网络一样。这个动态路由的机制跟RNN特别相似，看这个图就会非常的清楚了，下面这个图节选自国立台湾大学的自然语言处理课堂一个老师的PPT:(附上链接：https://www.csie.ntu.edu.tw/~yvchen/f106-adl/syllabus.html (https://www.csie.ntu.edu.tw/~yvchen/f106-adl/syllabus.html))



　　上述图详细描述了动态路由机制。Capsule的共享参数版的网络类似于：



　　以上就是Capsule的一些大体的概念！！！

In [2]:

```
from keras import backend as K
from keras.engine.topology import Layer
from keras.layers import *
```

```
from keras.layers import

def squash(x, axis=-1):
    s_squared_norm = K.sum(K.square(x), axis, keepdims=True) + K.epsilon()
    scale = K.sqrt(s_squared_norm)/ (0.5 + s_squared_norm)
    return scale * x


#define our own softmax function instead of K.softmax
def softmax(x, axis=-1):
    ex = K.exp(x - K.max(x, axis=axis, keepdims=True))
    return ex/K.sum(ex, axis=axis, keepdims=True)


#A Capsule Implement with Pure Keras
class Capsule(Layer):
    def __init__(self, num_capsule, dim_capsule, routings=3, share_weights=True, activation='default', **kwargs):
        super(Capsule, self).__init__(**kwargs)
        self.num_capsule = num_capsule
        self.dim_capsule = dim_capsule
        self.routings = routings
        self.share_weights = share_weights
        if activation == 'default':
            self.activation = squash
        else:
            self.activation = activation
        print(self.activation)

    def build(self, input_shape):
        super(Capsule, self).build(input_shape)
        input_dim_capsule = input_shape[-1]
        if self.share_weights:
            self.W = self.add_weight(name='capsule_kernel',
                                     shape=(1, input_dim_capsule,
                                            self.num_capsule * self.dim_capsule),
                                     initializer='glorot_uniform',
                                     trainable=True)
        else:
            input_num_capsule = input_shape[-2]
            self.W = self.add_weight(name='capsule_kernel',
                                     shape=(input_num_capsule,
                                            input_dim_capsule,
                                            self.num_capsule * self.dim_capsule),
                                     initializer='glorot_uniform',
                                     trainable=True)

    def call(self, u_vecs):
        if self.share_weights:
            u_hat_vecs = K.conv1d(u_vecs, self.W)
        else:
            u_hat_vecs = K.local_conv1d(u_vecs, self.W, [1], [1])

        batch_size = K.shape(u_vecs)[0]
        input_num_capsule = K.shape(u_vecs)[1]
        u_hat_vecs = K.reshape(u_hat_vecs, (batch_size, input_num_capsule,
                                            self.num_capsule, self.dim_capsule))
        u_hat_vecs = K.permute_dimensions(u_hat_vecs, (0, 2, 1, 3))
        #final u_hat_vecs.shape = [None, num_capsule, input_num_capsule, dim_capsule]

        b = K.zeros_like(u_hat_vecs[:,:,:,0]) #shape = [None, num_capsule, input_num_capsule]
        for i in range(self.routings):
            c = softmax(b, 1)
            o = K.batch_dot(c, u_hat_vecs, [2, 2])
            o = self.activation(o)
            if i < self.routings - 1:
                b = K.batch_dot(o, u_hat_vecs, [2, 3])
        return o

    def compute_output_shape(self, input_shape):
        return (None, self.num_capsule, self.dim_capsule)
```

```
Using TensorFlow backend.
```

In [29]:

```
from keras.layers import Embedding,Input, Convolution1D, MaxPooling1D, Flatten, Dense, Bidirectional,concatenate,G
from keras.models import Model,load_model
from keras.layers import Conv1D
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.models import load_model
from keras.optimizers import Adam
def capsule_RCNN(vocab_size, embedding_matrix,ckpt_path,max_len,embedding_size):
```

```
    main_input = Input(shape=(max_len,))
    embed = Embedding(vocab_size,embedding_size,input_length=max_len, weights=[embedding_matrix],trainable=True)(m
    cnn = Convolution1D(256,3,padding='same',strides=1,activation='relu')(embed)
    cnn = MaxPooling1D(pool_size=4)(cnn)
    cnn = Flatten()(cnn)
    cnn = Dense(128)(cnn)
    rnn = LSTM(256)(embed)
    rnn = Dense(128)(rnn)
    con = concatenate([cnn,rnn],axis=-1)
    cnn = Reshape((-1, 256))(con)
    capsule = Capsule(1, 16, 3, True)(cnn)
    output = Lambda(lambda x: K.sqrt(K.sum(K.square(x), 2)), output_shape=(1,))(capsule)
    model = Model(inputs=main_input,outputs=output)
    adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
    print(model.summary())
    try:
        model.load_weights(ckpt_path)
        print("load weights finish....")
    except:
        print("no pre-weights...")
    model.compile(loss='mse', optimizer=adam, metrics=['mse'])
    return model
def train_capsule_model(vocab_size, x_train, y_train, x_test, y_test, embedding_matrix,
                  batch_size,epoch,ckpt,model_file,max_len,embedding_size):
    """
    训练模型
    :param vocab_size:
    :param x_train:
    :param y_train:
    :param x_test:
    :param y_test:
    :param embedding_matrix:
    :return:
    """
    callback = ModelCheckpoint(ckpt, verbose=1,save_best_only=True,save_weights_only=True)
    early_stop = EarlyStopping(monitor='val_mse',patience=5,verbose=1)
    reduce_lr = ReduceLROnPlateau(monitor='val_mse',factor=0.2,
                                  patience=5,min_lr=0.001)
    callback_list = [callback,early_stop,reduce_lr]
    try:
        model = load_model(model_file)
        meritcs = model.evaluate(x_test,y_test)
        print('\ntest meritcs',meritcs)
        print("load model finish...")
    except:
        model = capsule_RCNN(vocab_size,embedding_matrix,ckpt, max_len,embedding_size)
        print("create model finish...")
    model.fit(x_train,y_train,
              batch_size=batch_size,
              callbacks=callback_list,
              shuffle=True,
              epochs=epoch,
              verbose=1,
              validation_data=(x_test,y_test))
    model.save(model_file)
```

In [ ]:

```
batch_size = 32
epoch = 5
max_len = 450
embedding_size = 300
google_vec = get_vec_from_file()
embedding_matrix, len_words, word2index, index2word = get_google_vec(google_vec)
train_data = all_data[~all_data["score"].isnull()]
test_data = all_data[all_data["score"].isnull()]
x_train,x_test,y_train,y_test,trains,labels = load_train_data(train_data,word2index,max_len = max_len)
tests, id_list = load_test_data(test_data,word2index,max_len=max_len)
train_capsule_model(len_words, x_train, y_train, x_test, y_test, embedding_matrix,
                          batch_size, epoch, "/home/kesci/work/capsule.hdf5", "/home/kesci/work/model_capsule",max
```

```
100%|██████████████| 10569/10569 [00:00<00:00, 884575.15it/s]
100%|██████████████| 10571/10571 [00:00<00:00, 362736.33it/s]
100%|██████████████| 1200/1200 [00:00<00:00, 22303.01it/s]
100%|██████████████| 550/550 [00:00<00:00, 24086.82it/s]

get word2index...
found 300 word vectors.
found 10571 words.
begin generate word matrix

the train2index's shape is:  (1200,)
```

目录

```
begin covert text to index

finish covert text to index..
the label's shape:  (1200,)
split data
the train's shape is (1200, 450)
Shape of x_train tensor: (960, 450)
Shape of y_train tensor: (960,)
Shape of x_test tensor: (240, 450)
Shape of y_test tensor: (240,)
the test2index's shape is:  (550,)

begin covert text to index

finish covert text to index..


<function squash at 0x7f5e57124e18>
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 450)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 450, 300)     3171300     input_1[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 450, 256)     230656      embedding_1[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 112, 256)     0           conv1d_1[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 28672)        0           max_pooling1d_1[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 256)          570368      embedding_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 128)          3670144     flatten_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 128)          32896       lstm_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 256)          0           dense_1[0][0]
                                                                 dense_2[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 1, 256)       0           concatenate_1[0][0]
__________________________________________________________________________________________________
capsule_1 (Capsule)             (None, 1, 16)        4096        reshape_1[0][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)               (None, 1)            0           capsule_1[0][0]
==================================================================================================
Total params: 7,679,460
Trainable params: 7,679,460
Non-trainable params: 0
__________________________________________________________________________________________________
None
no pre-weights...
create model finish...
Train on 960 samples, validate on 240 samples
Epoch 1/5
928/960 [============================>.] - ETA: 31s - loss: 59.7221 - mean_squared_error: 59.7221 Epoch 00001: val_l
960/960 [==============================] - 950s 990ms/step - loss: 59.4756 - mean_squared_error: 59.4756 - val_loss:
Epoch 2/5

/opt/conda/lib/python3.5/site-packages/keras/callbacks.py:493: RuntimeWarning: Early stopping conditioned on metric
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning
/opt/conda/lib/python3.5/site-packages/keras/callbacks.py:889: RuntimeWarning: Reduce LR on plateau conditioned on m
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning

928/960 [============================>.] - ETA: 32s - loss: 58.8930 - mean_squared_error: 58.8930 Epoch 00002: val_l
960/960 [==============================] - 988s 1s/step - loss: 59.0141 - mean_squared_error: 59.0141 - val_loss: 61
Epoch 3/5
928/960 [============================>.] - ETA: 33s - loss: 59.0027 - mean_squared_error: 59.0027 Epoch 00003: val_l
960/960 [==============================] - 998s 1s/step - loss: 59.0054 - mean_squared_error: 59.0054 - val_loss: 61
Epoch 4/5
928/960 [============================>.] - ETA: 33s - loss: 59.0411 - mean_squared_error: 59.0411 Epoch 00004: val_l
960/960 [==============================] - 1013s 1s/step - loss: 58.9997 - mean_squared_error: 58.9997 - val_loss: 6
Epoch 5/5
 32/960 [>.............................] - ETA: 16:27 - loss: 66.2361 - mean_squared_error: 66.2361
```

从上面的训练过程来看，capsule在文本上的应用还需要一些在网络上的改进，因为如果是一般的网络在3个epoch之后,mse都会有着明显下降，而capsule网络却没有这样。因此经过思考觉得，在这个网络中还需要进一步改进来应用到文本中。一家之言~

7. 本周任务总结

最终的成绩如下：

第二周K-Lab	sub1.csv	2018-02-07 11:57:25	0.821957

目录

在这个任务中，我主要是使用了深度学习进行探索，由于时间紧、强度大，因此在本周的任务中并没有提取什么特征，而是采用了深度学习这种端到端模式。这个解决方案主要包括以下几个步骤：

1. 数据预处理
2. 使用预训练词向量搭建深度网络
3. 训练神经网络
4. 利用训练好的神经网络模型进行特征抽取
5. 抽取深度特征将其与机器学习进行结合
6. 探索式的进行了Capsule在文本中的应用（由于水平有限，效果并不理想）
 总的来讲，在这周的任务中学到非常多的知识。其中有对以前的知识的一个梳理，还有一些新的内容学习探索，收获很大。由于时间有限，平台没有提供GPU服务，因此没有设计比较深的网络，都是一些浅显的网络。我的网络是以前设计的，并没有经过大的变动。在这里https://www.kesci.com/apps/home/competition/forum/5a7aa2f687f59f7b77dd03b8 (https://www.kesci.com/apps/home/competition/forum/5a7aa2f687f59f7b77dd03b8) 分享了我以前设计的有关NLP任务的神经网络。欢迎大家收藏~

8. 长远的思考~

在这个任务中依然有比较多的改进地方。在这里粗略的罗列几条，等之后闲下来再进行改进：

1. 进行统计特征工程，将其与深度模型抽取的特征合并，然后放入机器学习中进行学习。
2. 进行统计特征工程，将其和原始数据设计一个多模态输入的深度学习模型。
3. 更换更好的神经网络结构（尝试其他的神经网络结构）
4. Capsule网络在文本方面应用的改进（这是一个比较新而且难得地方）
5. 进行深度模型的调参工作。
 暂时想起来这么多，会有许许多多的改进地方，这仅属于不到一周给出的解决方案，比较粗糙！欢迎指正其他问题~

9. 参考内容

1. 以前自己所总结的一些神经网络：https://github.com/cyzLoveDream/text_dnn_classifier/blob/master/untils/NlpNetWork.py (https://github.com/cyzLoveDream/text_dnn_classifier/blob/master/untils/NlpNetWork.py)
2. 苏神的博文《揭开迷雾，来一顿美味的Capsule盛宴 (https://spaces.ac.cn/archives/4819/)》
3. 台湾国立大学 (https://www.csie.ntu.edu.tw/~yvchen/f106-adl/syllabus.html)的课堂PPT:https://www.csie.ntu.edu.tw/~yvchen/f106-adl/doc/171116+171120_Tip.pdf (https://www.csie.ntu.edu.tw/~yvchen/f106-adl/doc/171116+171120_Tip.pdf)

10. 好未来AILab训练营总结

为期两周的训练营生活结束了~，在这两周中结实了很多热爱NLP的小伙伴，很多实力很强的人。从他们身上学到很多，我们共同学习，一起进步！就像当初进营之前所希望的那样，结实小伙伴，专业技能的培养，项目能力的强化都得到了；这是我第一次参加训练营，收获真的超级大，特别是一周一个项目这种高强度的训练，真的让人成长很快，从没有接触过英文文本的处理到现在侃侃而谈，两周~一个人能改变的真的很多。在这个训练营中，由于小伙伴的谦让，也终于收获了比赛生涯第一个冠军。真的很感谢主办方！让我们有这么好的机会进行技能的训练。在这即将到来的新年之际，祝愿好未来越来越好！走向更加美好的未来。再次感谢！！！

目录

目录

目

录

和鲸社区公众号

工具 内容 实战

K-Lab 项目 比赛

数据集 众包

专栏

专区

heywhale和鲸

商务合作

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