谣言预测

2022 年 5 月 31 日

[1]:
```
from collections import Counter
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
import jieba
import numpy as np
import random
import pandas as pd
random.seed(2022)
```

1 1. 数据分析

数据集载入

[2]:
```
data = pd.read_csv('covid19_rumors.csv')
data.shape
```

[2]:
```
(864, 4)
```

[3]:
```
data.head()
```

[3]:
```
    crawlTime                                        mainSummary rumorType  \
0  2020-02-24                                          2 月 23 日…       fake
1  2020-02-19  世界卫生组织辟谣 ：不能，肺炎球菌疫苗和乙型流感嗜血杆菌疫苗等肺炎疫苗不能预防新冠肺炎       fake
2  2020-01-29                  在法国，无论是“黄马甲”运动还是大罢工，目前都并未结束…       fake
3  2020-02-21                                经查证，这是一条电脑合成的焰火视频…       ␣
↪fake
```

```
4   2020-02-08                网传消息并没有罗列出将在上海哪些主干道、用什么药物进行消毒作业…        fake
```

```
                        title
0       国家体育总局下发 4 月 30 日前禁止办赛通知
1             普通肺炎的疫苗能预防新冠病毒
2  一个武汉女人终结了法国大罢工，法国人都去抢口罩了
3               武汉长江大桥燃放烟花驱疫
4  上海主干道今晚 12 点大面积消毒，10 点后不要出门
```

将 rumorType 转化成数字

[4]:
```
truth_le = LabelEncoder()
new_rumor = truth_le.fit_transform(data['rumorType'])
data['rumorType'] = new_rumor
```

[5]:
```
new_rumor[:5]
```

[5]:
```
array([1, 1, 1, 1, 1])
```

2　2. 可视化分析

2.1 看一下谣言数量随时间的变化

[6]:
```
data['crawlTime'] = pd.to_datetime(data['crawlTime'])
data = data.sort_values(by='crawlTime')
data.head(10)
# data.shape
```

[6]:
```
     crawlTime                                mainSummary  rumorType  \
838 2020-01-21              农业上确实有在大棚中燃烧硫磺杀灭害虫细菌的做法…          ␣
↪ 1
680 2020-01-21  降低感染新型冠状病毒风险的方法包括：用肥皂和清水或含有酒精的洗手液…           2
532 2020-01-22   这个装置学名叫隔离担架，主要用途是转运传染性疾病患者或者疑似患者…␣
↪          1
137 2020-01-22     京天成这次通报成功研制的抗体叫“2019 冠状病毒 N 蛋白抗体”…          ␣
↪      1
```

```
355 2020-01-22                  就医用口罩而言，只要正确佩戴合格产品…          1
570 2020-01-22                       口罩正确的戴法是…          1
210 2020-01-22        由于病毒主要通过飞沫微粒被吸入人体，后经呼吸道粘膜侵入…  ␣
↪    1
862 2020-01-22        支付宝官方微博公开辟谣：“这是网传‘维智科技’的制图…  ␣
↪  1
120 2020-01-22        青岛市公安局即墨分局通报称，即墨警方迅速组织力量展开调查…  ␣
↪     1
426 2020-01-22                  新京报记者致电北京协和医院，工作人员表示…          1

                          title
838              放烟花爆竹可以消毒，预防瘟疫
680      勤洗手和戴口罩可以降低新型冠状病毒感染风险
532     这个玻璃装置是“隔离舱”，用它进行患者的隔离
137           新型冠状病毒抗体能治疗和预防肺炎
355          戴多层口罩，才能有效预防新型冠状病毒
570    口罩正确戴法：感冒时有颜色的朝外，没感冒反过来
210   普通人出门要佩戴护目镜，防止新型冠状病毒侵入体内
862          支付宝提供了在武汉海鲜市场的支付数据
120     山东即墨区发现疑似新型冠状病毒感染的肺炎病例
426              北京协和医院有武汉肺炎患者出逃
```

[7]:
```
rumor_gp = data.groupby(['crawlTime'])['title'].agg('count').reset_index()
rumor_gp
```

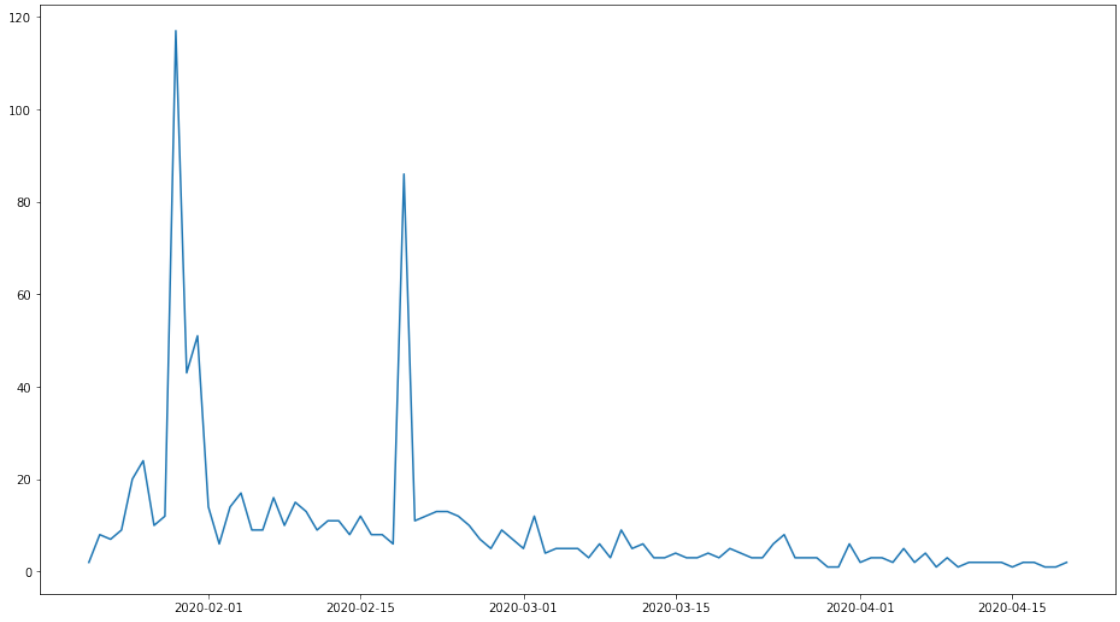
[7]:
```
    crawlTime  title
0  2020-01-21      2
1  2020-01-22      8
2  2020-01-23      7
3  2020-01-24      9
4  2020-01-25     20
..        …    …
86 2020-04-16      2
87 2020-04-17      2
88 2020-04-18      1
89 2020-04-19      1
```

```
90 2020-04-20      2

[91 rows x 2 columns]
```

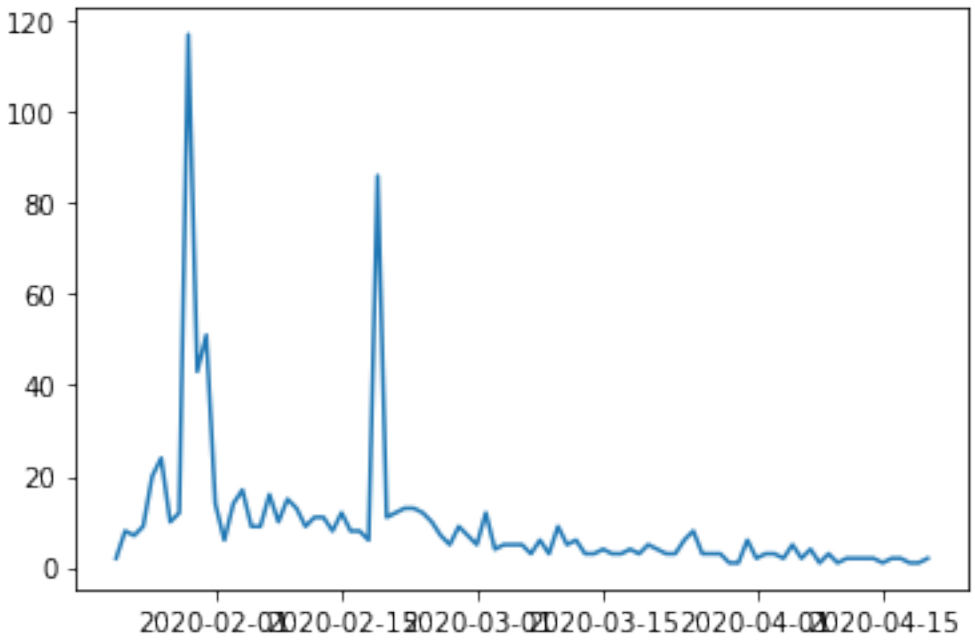
[8]:
```
fig, ax = plt.subplots(figsize=(16, 9))
ax.plot_date(rumor_gp['crawlTime'], rumor_gp['title'], fmt='-')
```

[8]: [<matplotlib.lines.Line2D at 0x1a38d98eec8>]



[9]:
```
plt.plot(rumor_gp['crawlTime'], rumor_gp['title'])
```

[9]: [<matplotlib.lines.Line2D at 0x1a38daacec8>]



2. 可视化不同类型的谣言（被证实，未被证实，实际为真）

[10]:
```
rumor_ByType = data.groupby(['crawlTime','rumorType'])['title'].agg('count').
↪reset_index()
# rumor_ByType
```

[11]:
```
rumor_ByType = rumor_ByType.
↪pivot(index='crawlTime',columns='rumorType',values='title')
rumor_ByType = rumor_ByType.rename_axis(None, axis=1)
# rumor_ByType
```

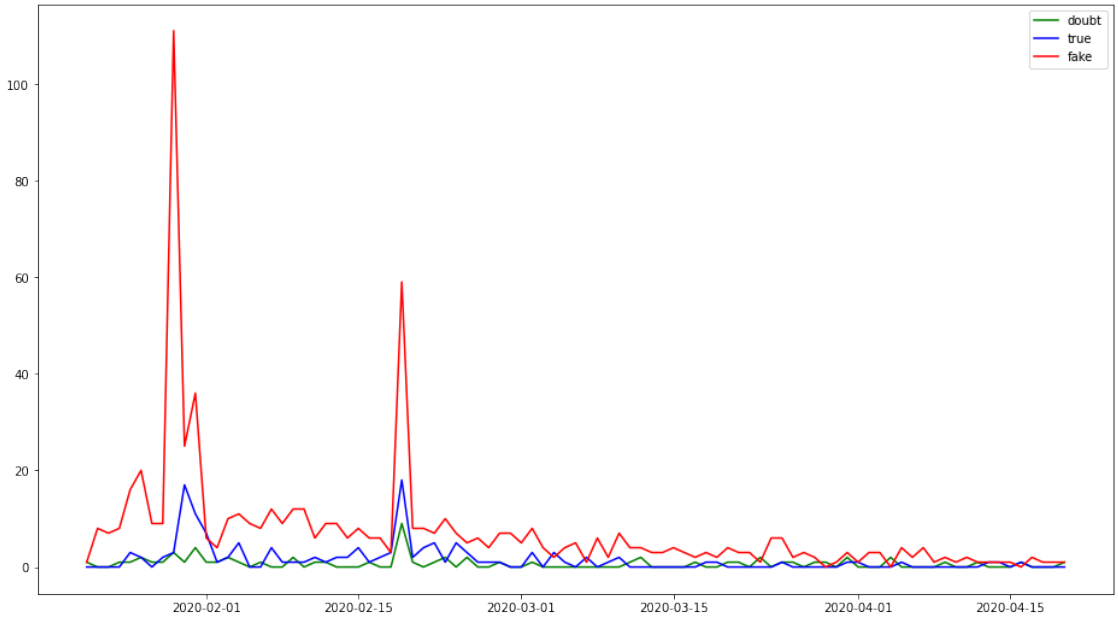
[12]:
```
rumor_ByType = rumor_ByType.rename(columns = {1:'fake',2:'doubt',0:'true'})
rumor_ByType = rumor_ByType.fillna(0)
```

[13]:
```
# rumor_ByType.head()
```

[14]:
```
fig,ax = plt.subplots(figsize=(16,9))
ax.plot_date(rumor_ByType.index,rumor_ByType['doubt'],fmt='g-')
ax.plot_date(rumor_ByType.index,rumor_ByType['true'],fmt='b-')
ax.plot_date(rumor_ByType.index,rumor_ByType['fake'],fmt='r-')
```

```
plt.legend(['doubt','true','fake'])
```

[14]: <matplotlib.legend.Legend at 0x1a38db4dd48>



具体看一下谣言出现最多的时候，都是哪些谣言

[15]:
```
cond =␣
↪(data['rumorType']==1)&(data['crawlTime']>'2020-01-28')&(data['crawlTime']<'2020-01-31')
data[cond].sort_values(by='crawlTime').head()
```

[15]:

	crawlTime	mainSummary	rumorType	title
597	2020-01-29	丁香医生团队辟谣：无论何时，接触了钱币都要洗手	1	不要收现金，会感染新冠病毒
403	2020-01-29	丁香医生团队辟谣：没有预防作用	1	服用 VC 可以预防感染
533	2020-01-29	丁香医生团队辟谣：浓度为 60%-80% 的乙醇才有效	1	酒精浓度越高，消毒效果越好
71	2020-01-29	丁香医生团队辟谣：暂无证据证明会传染猫狗	1	家里的宠物会传播病毒
86	2020-01-29	其实是误解	1	日本派出千名医疗人员

3 3. 构筑模型进行谣言预测

首先看下数据集当中被证实的谣言比例是多少

[82]:
```
from sklearn.metrics import *
y_dum = np.ones(len(data['rumorType'].values))
accuracy_score(y_dum, data['rumorType'])
```

[82]: 0.7662037037037037

基本模型有着 **76.6%** 的准确率，我们要需要做的比这个更好一些

我们在这里用最简单的 **Multinomial Naive Bayes** 模型首先，因为数据是文本形式，所以我们需要将文本转化成字段这里我们用了 **TfIdfVectorizer**，将文本向量化

[83]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
zhTokenizer = jieba.cut
v = TfidfVectorizer(token_pattern=r'(?u)\b\w+\b',
tokenizer = zhTokenizer,
lowercase = False,
stop_words = ['是','的'],
max_features = 250)
```

构建对应的数据集因为是 **Baseline**，所以我们这里只采用了 **title** 字段的信息

[84]:
```
##Lets ignore the timestamp and mainSummary for the moment
y = data['rumorType']
X_txt = data.drop(['rumorType','crawlTime','mainSummary'],axis=1)


# y, X_txt
```

[85]:
```
from sklearn.model_selection import␣
↪train_test_split,cross_val_score,cross_validate
X_train,X_test,y_train,y_test = train_test_split(X_txt,y,test_size=0.
↪2,stratify=y)
```

将训练集的文本向量化，并且进行对应的训练，这样我们就可以完成一个初步的实验

[86]:
```
#Convert X_train
X_train_v = v.fit_transform(X_train['title'])
```

[87]:
```
# X_train['title']
# X_train_v
```

[88]:
```
from sklearn.naive_bayes import MultinomialNB as mnb
model_bl = mnb()
model_bl.fit(X_train_v,y_train.values)
```

[88]:
```
MultinomialNB()
```

[89]:
```
X_test_v = v.transform(X_test['title'])
X_test_v
```

[89]:
```
<173x250 sparse matrix of type '<class 'numpy.float64'>'
        with 790 stored elements in Compressed Sparse Row format>
```

[90]:
```
y_pred = model_bl.predict(X_test_v)
y_pred
```

[90]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[91]:
```
accuracy_score(y_pred,y_test) # 准确率
```

[91]:
```
0.7861271676300579
```

[109]:
```
mean_squared_error(y_test.values,y_pred)
```

[109]:
```
0.2138728323699422
```

[92]:
```
precision_recall_fscore_support(y_test.values,y_pred,average='micro')
```

[92]:
```
(0.7861271676300579, 0.7861271676300579, 0.7861271676300579, None)
```

[93]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

[94]:
```
# 决策树
from sklearn.tree import DecisionTreeClassifier
predictor1=DecisionTreeClassifier(max_depth=5)
predictor1.fit(X_train_v,y_train.values).score(X_test_v,y_test)
```

[94]: 0.7745664739884393

[95]:
```
y_predict1= predictor1.fit(X_train_v,y_train.values).predict(X_test_v)
```

[96]:
```
recall_score(y_test.values,y_predict1,pos_label='positive'
                                            ,average='micro')
precision_score(y_test.values,y_predict1,pos_label='positive'
                                            ,average='micro')
```

```
D:\Anaconda\envs\pytorch1.0_gpu\lib\site-
packages\sklearn\metrics\_classification.py:1301: UserWarning: Note that
pos_label (set to 'positive') is ignored when average != 'binary' (got 'micro').
You may use labels=[pos_label] to specify a single positive class.
  % (pos_label, average), UserWarning)
D:\Anaconda\envs\pytorch1.0_gpu\lib\site-
packages\sklearn\metrics\_classification.py:1301: UserWarning: Note that
pos_label (set to 'positive') is ignored when average != 'binary' (got 'micro').
You may use labels=[pos_label] to specify a single positive class.
  % (pos_label, average), UserWarning)
```

[96]: 0.7687861271676301

[97]:
```
precision_recall_fscore_support(y_test.values,y_predict1,average='micro')
```

[97]: (0.7687861271676301, 0.7687861271676301, 0.7687861271676302, None)

[108]:
```
mean_squared_error(y_test.values,y_predict1)
```

[108]: 0.2658959537572254

[101]:
```
# 随机森林
from sklearn.ensemble import RandomForestClassifier
predictor2 = RandomForestClassifier(max_depth=20, random_state=1234)
predictor2.fit(X_train_v,y_train.values).score(X_test_v,y_test)
```

[101]: 0.791907514450867

[102]:
```
y_predict2= predictor2.fit(X_train_v,y_train.values).predict(X_test_v)
```

[103]:
```
recall_score(y_test.values,y_predict2,pos_label='positive'
                                              ,average='micro')
```

```
D:\Anaconda\envs\pytorch1.0_gpu\lib\site-
packages\sklearn\metrics\_classification.py:1301: UserWarning: Note that
pos_label (set to 'positive') is ignored when average != 'binary' (got 'micro').
You may use labels=[pos_label] to specify a single positive class.
  % (pos_label, average), UserWarning)
```

[103]: 0.791907514450867

[104]:
```
precision_score(y_test.values,y_predict2,pos_label='positive'
                                              ,average='micro')
```

```
D:\Anaconda\envs\pytorch1.0_gpu\lib\site-
packages\sklearn\metrics\_classification.py:1301: UserWarning: Note that
pos_label (set to 'positive') is ignored when average != 'binary' (got 'micro').
You may use labels=[pos_label] to specify a single positive class.
  % (pos_label, average), UserWarning)
```

[104]: 0.791907514450867

[105]:
```
precision_recall_fscore_support(y_test.values,y_predict2,average='micro')
```

[105]: (0.791907514450867, 0.791907514450867, 0.791907514450867, None)

[107]:
```
mean_squared_error(y_test.values,y_predict2)
```

[107]: 0.2254335260115607

[128]:
```
#xgboost


from xgboost import XGBClassifier
from xgboost import plot_importance



model = XGBClassifier(learning_rate=0.01,
```

```
                    n_estimators=10,           # 树的个数-10 棵树建立 xgboost
                    max_depth=4,               # 树的深度
                    min_child_weight = 1,      # 叶子节点最小权重
                    gamma=0.,                  # 惩罚项中叶子结点个数前的参数
                    subsample=1,               # 所有样本建立决策树
                    colsample_btree=1,         # 所有特征建立决策树
                    scale_pos_weight=1,        # 解决样本个数不平衡的问题
                    random_state=27,           # 随机数
                    slient = 0
                    )
model.fit(X_train_v,y_train.values).score(X_test_v,y_test)
```

```
[08:22:30] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.6.0/src/learner.cc:627:
Parameters: { "colsample_btree", "scale_pos_weight", "slient" } might not be
used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.
```

[128]:
```
0.791907514450867
```

[129]:
```
y_predict3= model.fit(X_train_v,y_train.values).predict(X_test_v)
```

```
[08:22:43] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.6.0/src/learner.cc:627:
Parameters: { "colsample_btree", "scale_pos_weight", "slient" } might not be
used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
```

being used
but getting flagged wrongly here. Please open an issue if you find any such cases.

[130]:
```
precision_recall_fscore_support(y_test.values,y_predict3,average='micro')
```

[130]:
```
(0.791907514450867, 0.791907514450867, 0.791907514450867, None)
```

[131]:
```
mean_squared_error(y_test.values,y_predict3)
```

[131]:
```
0.2254335260115607
```

[136]:
```
# lightGBM
from lightgbm import LGBMRegressor
gbm = LGBMRegressor(objective='regression', num_leaves=31, learning_rate=0.05,␣
 ↪n_estimators=20)
gbm.fit(X_train_v, y_train.values, eval_set=[(X_test_v,y_test)],␣
 ↪eval_metric='l1', early_stopping_rounds=5)
```

```
[1]     valid_0's l1: 0.295664  valid_0's l2: 0.227125
[2]     valid_0's l1: 0.294782  valid_0's l2: 0.224997
[3]     valid_0's l1: 0.293801  valid_0's l2: 0.223194
[4]     valid_0's l1: 0.293273  valid_0's l2: 0.221619
[5]     valid_0's l1: 0.292394  valid_0's l2: 0.220213
[6]     valid_0's l1: 0.291712  valid_0's l2: 0.218295
[7]     valid_0's l1: 0.291268  valid_0's l2: 0.217004
[8]     valid_0's l1: 0.290662  valid_0's l2: 0.215343
[9]     valid_0's l1: 0.290409  valid_0's l2: 0.214213
[10]    valid_0's l1: 0.290192  valid_0's l2: 0.213243
[11]    valid_0's l1: 0.289679  valid_0's l2: 0.212484
[12]    valid_0's l1: 0.289389  valid_0's l2: 0.212019
[13]    valid_0's l1: 0.289609  valid_0's l2: 0.211999
[14]    valid_0's l1: 0.28983   valid_0's l2: 0.211309
[15]    valid_0's l1: 0.289447  valid_0's l2: 0.210764
[16]    valid_0's l1: 0.28924   valid_0's l2: 0.210291
[17]    valid_0's l1: 0.288135  valid_0's l2: 0.209163
[18]    valid_0's l1: 0.287399  valid_0's l2: 0.208169
```

```
[19]    valid_0's l1: 0.287749  valid_0's l2: 0.208002
[20]    valid_0's l1: 0.288366  valid_0's l2: 0.208053

D:\Anaconda\envs\pytorch1.0_gpu\lib\site-packages\lightgbm\sklearn.py:726:
UserWarning: 'early_stopping_rounds' argument is deprecated and will be removed
in a future release of LightGBM. Pass 'early_stopping()' callback via
'callbacks' argument instead.
  _log_warning("'early_stopping_rounds' argument is deprecated and will be
removed in a future release of LightGBM. "
```

[136]:
```
LGBMRegressor(learning_rate=0.05, n_estimators=20, objective='regression')
```

[138]:
```
y_pred= gbm.predict(X_test_v, num_iteration=gbm.best_iteration_)
```

[139]:
```
mean_squared_error(y_test.values,y_pred
                   )
```

[139]:
```
0.2081689636192498
```

[143]:
```
# SVM
from sklearn import svm
svm.SVC(kernel='rbf').fit(X_train_v,y_train.values).score(X_test_v,y_test)
```

[143]:
```
0.7803468208092486
```

[147]:
```
y_predict4= svm.SVC(kernel='rbf').fit(X_train_v,y_train.values).
 ↪predict(X_test_v)
```

[148]:
```
precision_recall_fscore_support(y_test.values,y_predict4,average='micro')
```

[148]:
```
(0.7803468208092486, 0.7803468208092486, 0.7803468208092486, None)
```

[149]:
```
mean_squared_error(y_test.values,y_predict4)
```

[149]:
```
0.23699421965317918
```

改变模型

尝试使用神经网络解决问题。(样本量太少了，不太合适)

[151]:
```
from sklearn.neural_network import MLPClassifier as nn
```

[152]:
```
nn_base = nn(hidden_layer_sizes=(100,50,20,5),learning_rate='constant',
random_state=2020,
warm_start=True,
max_iter=500)
```

[153]:
```
nn_base
```

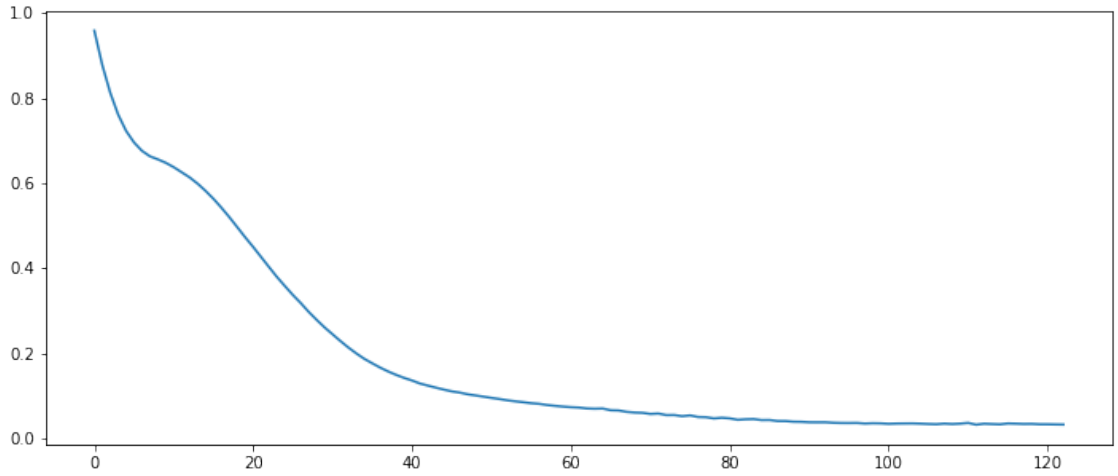
[153]:
```
MLPClassifier(hidden_layer_sizes=(100, 50, 20, 5), max_iter=500,
              random_state=2020, warm_start=True)
```

[155]:
```
nn_base.fit(X_train_v,y_train)
```

[155]:
```
MLPClassifier(hidden_layer_sizes=(100, 50, 20, 5), max_iter=500,
              random_state=2020, warm_start=True)
```

[156]:
```
_ = plt.figure(figsize=(12,5))
plt.plot(nn_base.loss_curve_)
```

[156]:
```
[<matplotlib.lines.Line2D at 0x1a394e4c148>]
```



[158]:
```
y_pred = nn_base.predict(X_test_v)
accuracy_score(y_pred,y_test)
```

[158]:
```
0.791907514450867
```

[159]:
```
mean_squared_error(y_test.values,y_pred)
```

[159]: 0.24277456647398843

[160]:
```
precision_recall_fscore_support(y_test.values,y_pred,average='micro')
```

[160]: (0.791907514450867, 0.791907514450867, 0.791907514450867, None)

[ ]:
```

```