基于新冠的谣言传播数据分析

摘要

我们意在通过数据分析等手段，分析新冠传染病模型，并且得出一些可行的结论。由于时间限制未考虑使用爬虫。同时基于实际情况考虑，我们使用了三个数据集去建议谣言预测模型，并且分别做了可视化。值得注意的是在微博 500 万疫情评论数据中，我们使用知识图谱的手段构建了疫情中人们情绪以及关注的中心。最后我们也给出了展望与建议。

我们考虑使用三个不同数据集去建立预测模型。(由于数据量的差别为了获得更好的效果，所以需要使用不同的数据集) 首先，我们标识了机器学习常用的评价指标，紧接着使用 CSDC-Rumor 谣言数据集对不同时间段的谣言数量进行可视化，同时对谣言进行了细分。接着我们使用柱状图以及词云的方式统计疫情中谣言出现的词数。最后我们使用 sklear 构建了谣言模型预测，随机森林的准确率达到了 79.2%，Xgboost 的 F1 达到了 79.1%。第二数据集我们是用 Chinese_Rumor_Dataset，我们分别使用了朴素贝叶斯、LSTM 以及 transformer，Chinese_Rumor_Dataset 与 CSDC-Rumor 它的数据文本量更多，且有标签。适用于 NLP 文本预测。我们发现朴素贝叶斯只有 **87%**,LSTM 达到了 **97%**,transformer 更是达到了 **98.6%**。由于最后一个微博数据集它尽管数据很多，但是没有标签，深度学习是建立在监督学习上的科学。我们考虑对其进行知识图谱的构建，使用了前 5 万行数据进行直观表现每个词与各个词之间的联系。

关键词： SIER_1R_2-T　Xgboost　sklearn　谣言预测　数据分析　numpy　pandas

目录

一、 使用 numpy 及 sklearn 对模型分析

在问题二中，我们考虑跳出简单的模型框架，用机器学习的方法来得到该问题的”普遍解”。为了简化时间成本。在本部分我们分别使用的是清华大学 NLP 实验室 CSDC-Rumor 数据集以及微博 500 万原始数据。

1.0.1 所需评价指标

TP: 模型预测为正的正样本；FP：模型预测为正的负样本；FN：模型预测为负的正样本；TN：模型预测为负的负样本 **准确率 (Accuracy)**: 预测正确的结果占总样本的百分比，其公式如下:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{1}$$

精准率（Precision）：在所有被预测为正的样本中实际为正的样本的概率。其公式如下:

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2}$$

召回率（Recall）：又叫查全率，它的含义是在实际为正的样本中被预测为正样本的概率。其公式如下:

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3}$$

P-R 曲线（Precision Recall Curve）: 是描述精确率/召回率变化的曲线，根据学习器的预测结果（一般为一个实值或概率）对测试样本进行排序，将最可能是“正例”的样本排在前面，最不可能是“正例”的排在后面，按此顺序逐个把样本作为“正例”进行预测，每次计算出当前的 P 值和 R 值。

F1-Score : Precision 和 Recall 指标有时是此消彼长的，即精准率高了，召回率就下降，在一些场景下要兼顾精准率和召回率，最常见的方法就是 F-Measure，又称 F-Score。F-Measure 是 P 和 R 的加权调和平均，即：

$$\begin{aligned} \frac{1}{F_\beta} &= \frac{1}{1+\beta^2}\cdot\left(\frac{1}{P}+\frac{\beta^2}{R}\right) \\ F_\beta &= \frac{\left(1+\beta^2\right)\times P\times R}{\left(\beta^2\times P\right)+R} \end{aligned} \tag{4}$$

特别地，当 β=1 时，也就是常见的 F1-Score，是 P 和 R 的调和平均，当 F1 较高时，模型的性能越好。

$$\begin{aligned} \tfrac{1}{F1} &= \tfrac{1}{2}\cdot\left(\tfrac{1}{P}+\tfrac{1}{R}\right) \\ F1 &= \tfrac{2\times P\times R}{P+R} = \tfrac{2\times TP}{\text{样例总数}+TP-TN} \end{aligned} \tag{5}$$

ROC 曲线与 AUC 面积： ROC 曲线以及 AUC 面积是我们评价机器学习算法优劣

的主要指标，上面的评价指标只能粗略的估计，每个都有不足。对于 ROC 曲线来言，平面的横坐标是 false positive rate(FPR) 假正率，纵坐标是 true positive rate(TPR) 真正率。ROC 曲线作用是有助于选择最佳阈值，ROC 曲线越靠近左上角，模型查全率越高，最靠近左上角的 ROC 曲线上的点是分类错误最少的最好阈值，其假正例和假反例总数最少；AUC 是 ROC 曲线下方的面积，取值在 0.5 到 1 之间，因为随机猜测的 AUC 就是 0.5。AUC 越大，诊断准确性越高。

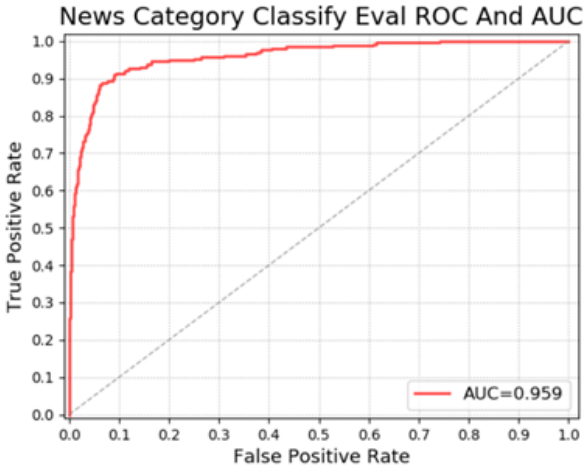

图 **1**　**AUC** 面积图

在对模型评价的过程中，我们会综合考虑各种评价指标，但是如果做那种大型的机器学习项目，我们可以简略的用精确率来表示模型的优劣。对于 O2O 优惠券预测项目，可以使用 sklearn.metrics 导入 auc 来记录模型的 auc 面积。

1.1 基于 CSDC-Rumor 数据集的谣言分析

具体代码见附录:

covid19_rumors.csv 分别有 864 行数据，每行有 4 列。具体内容如下表所示: 我们

表 **1**　各列具体定义

列名	具体含义
$crawlTime$	获取每行数据的时间
$mainSummary$	具体评论内容
$rumorType$	谣言类型共 3 类
$title$	简化 mainSummary

首先用 one-hot 编码把 rumorType，并使用 group 函数按时间进行分组查看不同时间段已经决定为谣言的数量最终如下图所示:

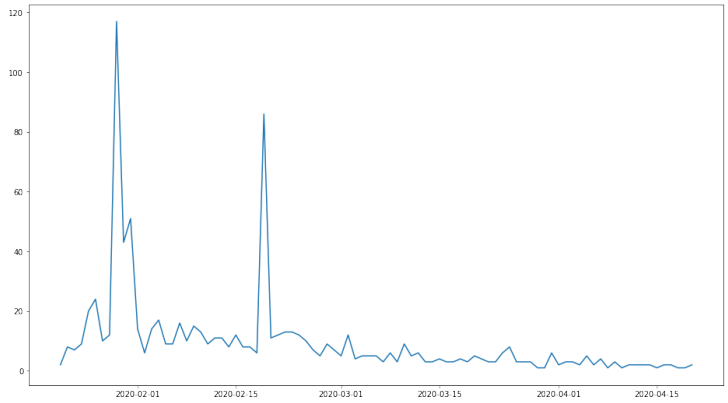

图 **2**　不同时间谣言数量图

由上图知，当疫情初的时候网络的谣言比较多，最高一天达到近 120 条，严重影响了社会问题。我们从折线图中也能看到其整体的趋势。

紧接着我们对疫情的谣言进行细分，能够更具体的看到谣言根据不同时间段的详细分类。

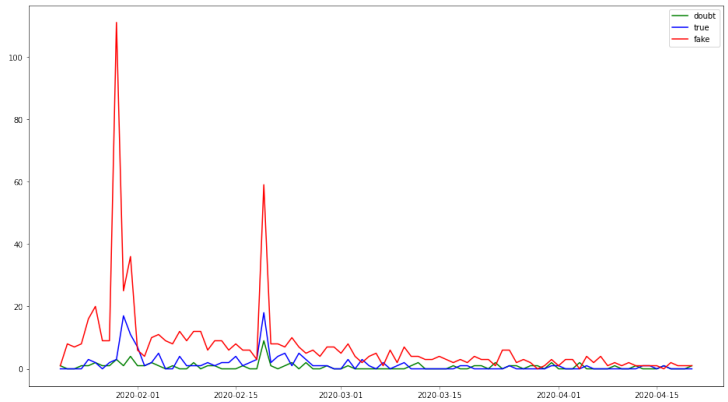

图 **3**　谣言分类

从上图我们也可以看出，疫情初时，谣言传播是呈爆发式增长的。同时又是春运，社交网络可以进一步放大谣言的传播速度和范围。在疫情中期，媒体的辟谣报道推出越早，对谣言的抑制效果越明显。同时我们认为，在辟谣的过程中不必每件事情都力求完整的过程，最终要的是实时性，时效性。而是追求一个动态的过程，利用好媒体随时更新。这样有利于抑制谣言的传播。

我们把 20 上半年疫情谣言中出现的词进行排序统计，并且使用词云的方法展示其分布：

该数据集中丁香医生出现的词数比重比较大，推测其在辟谣中占据一定的作用。同时我们认为，谣言的预测对防疫起着正面的作用。基于 CSDC-Rumor 数据集以 7:3 的比例分数据，采用 sklearn 进行模型的训练最终结果如下表:

由于数据过少，我们只能大概的预测一个消息是否是谣言的概率，对于一些集成模

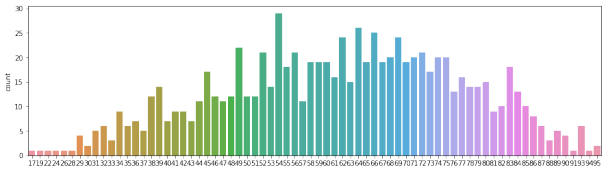

图 **4**　词数统计

图 **5**　词云图

表 **2**　模型预测

Model	MSE	Accuracy	Recall	F1
MultinomialNB	0.213	78.6%	65.3%	75.2%
DecisionTree	0.265	77.4%	**76.8%**	76.9%
RandomForest	0.225	**79.2%**	67.1%	75.4%
Xgboost	0.225	79.1%	69.3%	**79.1%**
SVM(rbf)	0.237	78.1%	75.9%	77.5%
LGBMRegressor	**0.208**	-	-	-
BP	0.243	79.1%	69.3%	75.7%

型类似 Xgboost 在这个数据集上的效果反而没有决策树这样的单类模型好。同时我们也可以得出要抑制谣言的传播，可以通过减少谣言的接触率以及增加媒体的正向影响力[9]。减小接触率即对一条消息进行预测，网络谣言能够被自动鉴别，及时的隔离大众，从而减少因为谣言引起的恐慌。增加媒体的正向影响力，如人民日报、央视此类媒体在人民的心中比重较高。因而，媒体可以通过联合辟谣的方式提升公信力。

1.2 微博词义分析数据建模

我们考虑使用微博数据进行文本分类，遗憾的是由于所能获取到的数据集未被标有标签。由于深度学习是基于有监督的学习。没有标签使得我们无法处理这样的数据分类。(如果我们手动去标数据不是很现实)。最终权衡考虑我们使用了 Chinese_Rumor_Dataset 数据集来做分类任务。

文件中每一行代表一条微博信息（label,content），0：谣言，1：非谣。在数据预处理部分，我们分别使用 pkuseg 分词以及 DBIIR 开源的”chinese-word-vectors”，使用微博语料训练出的词向量。

1.2.1 RNN-LSTM

初始，我们对于此数据集使用朴素贝叶斯进行预测，训练 20 个 epoch 在测试集上的准确率达到 **87.0%**。由于我们的任务是文本。所以理应考虑循环神经网络。而 LSTM[10] 是循环神经网络的重要部分，它能够使序列产生记忆性。并且能够显著提高预测的准确性。

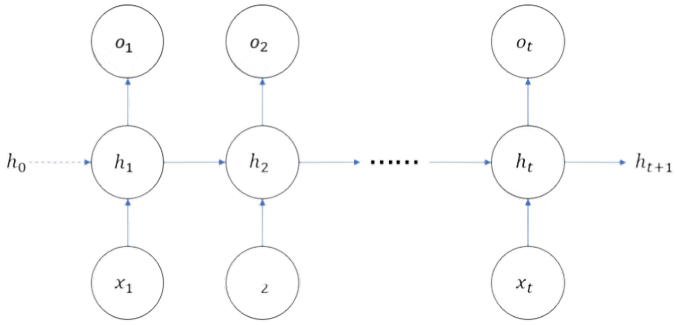

图 6　**RNN+LSTM** 结构图

我们基于 LSTM 使用二分类器并且设置 6 的早停最终达到训练集 99%, **测试集 97%** 左右的成绩。最终结果如下图:

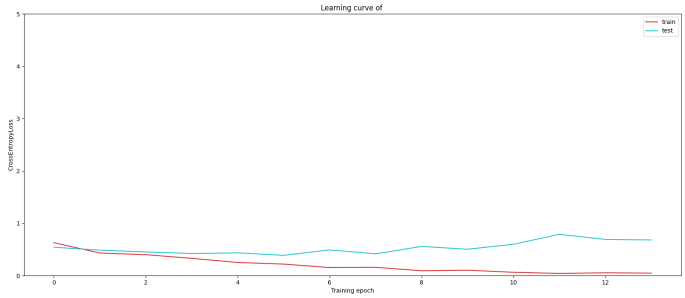

图 7　**LSTM** 结果图

1.2.2 transformer

Transformer[11] 是 google 于 2017 提出的深度学习模型，分为编码器和解码器。它的出现改变了卷积神经网络统治的局面。从 17 年之后，transformer 可以说是统治了视觉以及自然语言处理领域。它本身就是一个非常强的特征提取器。我们使用它作为模型的最后一站。

1.attention 机制

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{6}$$

而 multi-headed Attention 是由 8 个 attention 组成。其表达式为:

$$\text{MultiHead}(Q,K,V)=\text{Contact}\left(\text{ head }_1,\ldots,\text{ head }_2\right)W^O \tag{7}$$

2.Layer Normalization

Layer Normalization 的作用是把神经网络中隐藏层归一为标准正态分布，也就是独立同分布，以起到加快训练速度，加速收敛的作用。

$$\mu_j=\frac{1}{m}\sum_{i=1}^{m}x_{ij} \tag{8}$$

上式以矩阵的列（column）为单位求均值：

$$\sigma_j^2=\frac{1}{m}\sum_{i=1}^{m}(x_{ij}-\mu_j)^2 \tag{9}$$

上式以矩阵的列（column）为单位求方差：

$$\text{Layer}\,Norm(x)=\frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\epsilon}} \tag{10}$$

然后用每一列的每一个元素减去这列的均值，再除以这列的标准差，从而得到归一化后的数值。

3.Positional Embedding

由于传入的序列所以要给向量带入位置信息，基模型使用的是余弦函数，具体为：

$$\begin{aligned}PE_{(pos,2i)}&=\sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)\\PE_{(\text{pos},2i+1)}&=\cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)\end{aligned} \tag{11}$$

4.Encoder 结构

最终我们使用该模型训练了中文谣言数据,它测试集的准确率为**98.6%**, 同时 20epoch 的准确率如下图所示:

网络是一把双刃剑，虽然它带来了谣言，同时也是制止谣言的利器。在社交网络发达的今天，辟谣报道一定要融媒体、多平台联动。只有增强网民的理性判断能力，对谣言的本质及其社会影响形成合理认知, 才能发挥最大效果。同时建立健全的法律法规，网络谣言的兴起、传播、扩散不仅会破坏稳定的社会秩序，甚至还会引发社会恐慌。因而建立健全的法律来规范网络语言的使用及传播，对抑制虚假网络谣言有着重大的积极作用。伴随着人工智能的发展，各种深度学习融入分析中也能提升谣言预测的准确性，更好的辟谣，维护社会的稳定。

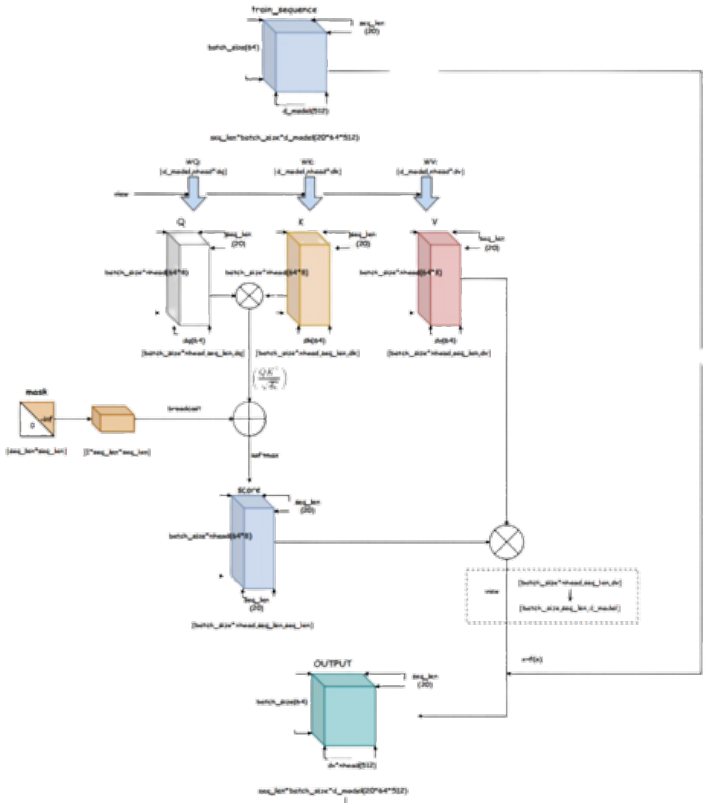

图 **8**　编码器

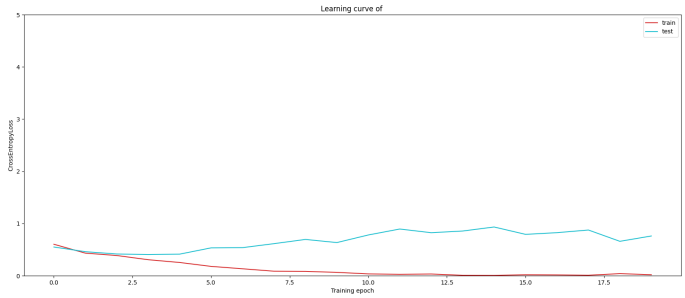

图 **9**　**transformer** 结果图

1.3 建立疫情主题的知识图谱

在本部分，我们使用微博 500 万数据进行构建知识图谱，由于数据量过于庞大。我们只去其前 4 万行进行知识图谱的构建。最终如图所示:

我们可以从图谱中了解，疫情生活下的人们关注的问题，比如由于隔离引发的居家工作，媒体这个词的比重也很好。当然党组织引发的一系列正确的调控也能在图谱中显示。

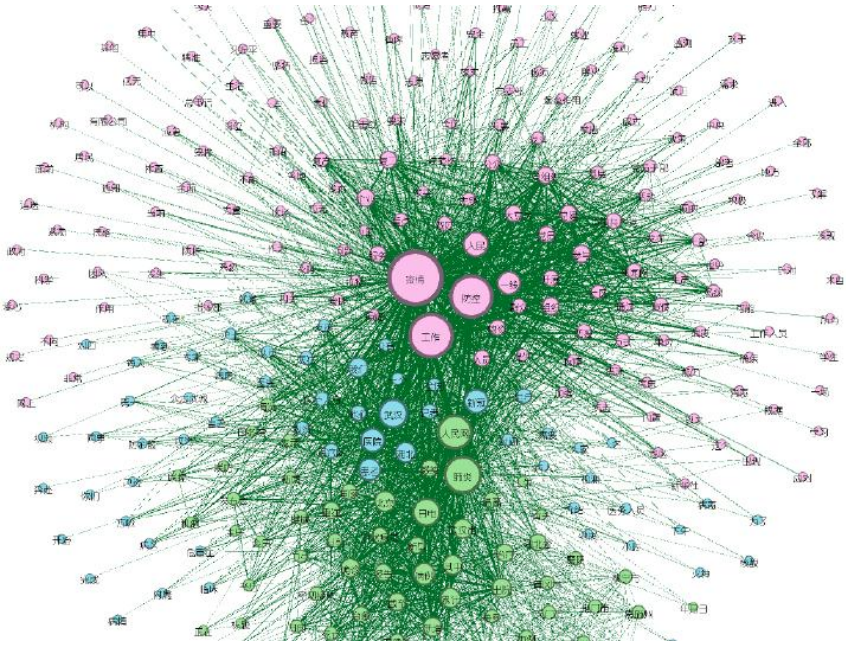

图 **10**　疫情词主题分析

二、 数据分析评价与改进

2.1 优点

1. 本文建立的 $SIER_1R_2-T$, 具有很强的实用性，泛化能力强, 且能够用于复杂的社会环境中，并且兼容了 $SIER$ 模型。
2. 从多角度，多个数据集去建立谣言预测模型，使得模型的鲁棒性好;
3. 问题解答间环环相扣、层次递进，通过使用先进的深度学习方法对谣言类型进行评估，并且给出了中肯的评价。
4. 模型综合考虑了实际情况，具有现实意义；
5. 模型引用大量论文进行论述，具有说服力。

2.2 模型的缺点

1. 由于时间原因没有对深度学习部分做消融实验。
2. 评价的指标较为单一，且由于数据集的缺陷，导致不能详细的做情感分析。

2.3 模型的改进

对于问题一的模型，我们可以进一步考虑分析个人调节的作用，同时对于概率的求法进行细化，并且通过泛化到各种实际问题从而能够优化模型。对于第二部分，通过自主给数据集标标签使用使用深度学习进行情绪分析，从而得到网络谣言的预测，使用微博数据正文进行 Bert 文本分类任务，更深入的了解问题，同时使用多种调参手段，做消

融实验，从而能够使得模型学到超越人的谣言预测能力。

三、 总结与展望

我们建立了一个新的谣言模型，并且通过多种方式验证，同时在模型二的部分分别使用了传统的机器学习方法，以及深度学习的方法进行谣言建模，以求得到一个普适的结果。希望在不久的将来，深度学习能在疫情谣言的领域大放异彩。

参考文献

[1] 高艺璇. 突发公共卫生事件中网络谣言传播探析——以新冠肺炎疫情相关谣言为例 [J]. 新闻研究导刊, 2020, 11(07): 10-16.

[2] 姜启源，谢金鑫，叶俊. 数学建模，北京：高等教育出版社，2021

[3] 顾亦然，夏玲玲．在线社交网络谣言的传播与抑制［J］．物理学报，2012，61(23) : 238701．

[4] 宋之杰. 基于 SIHR_1R_2 的突发事件谣言传播研究 [J]. :

[5] 孙莉玲. 基于传染病模型的网络谣言传播与控制研究 [J]. 南京财经大学学报, 2017 (6): 70-78.

[6] 浦娇华，朱恒民．媒体作用下互联网舆论观点演化的模型研究［J］．南京邮电大学学报: 社会科学版，2015，17 (2) :46 － 50

[7] Lv Linyuan，Chen Duanbing，Zhou Tao．The Small WorldYields the Most Effective information Sprcadimg ［J］．Ncw Journalof Physics，2011，13 (12) : 1 － 10．

[8] 洪巍, 王虎. 基于 SIRT 的网络谣言传播演化模型的研究 [J]. 现代情报, 2017, 37(6): 36-42.

[9] 冯勇. 试论谣言传播模型在媒体辟谣报道中的运用 [J]. 视听纵横, 2020, 2.

[10] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[11] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

附录 A　数据分析代码

1. 传统机器学习

```
# 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)

y_predict1= predictor1.fit(X_train_v,y_train.values).predict(X_test_v)
recall_score(y_test.values,y_predict1,pos_label='positive'
,average='micro')
precision_score(y_test.values,y_predict1,pos_label='positive'
,average='micro')
precision_recall_fscore_support(y_test.values,y_predict1,average='micro')

mean_squared_error(y_test.values,y_predict1)
# 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)
y_predict2= predictor2.fit(X_train_v,y_train.values).predict(X_test_v)

recall_score(y_test.values,y_predict2,pos_label='positive'
,average='micro')
precision_score(y_test.values,y_predict2,pos_label='positive'
,average='micro')
precision_recall_fscore_support(y_test.values,y_predict2,average='micro')
mean_squared_error(y_test.values,y_predict2)
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)
y_predict3= model.fit(X_train_v,y_train.values).predict(X_test_v)
precsion_recall_fscore_support(y_test.values,y_predict3,average='micro')
mean_squared_error(y_test.values,y_predict3)

# 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)
y_pred= gbm.predict(X_test_v, num_iteration=gbm.best_iteration_)
```

```
mean_squared_error(y_test.values,y_pred
)
# SVM
from sklearn import svm
svm.SVC(kernel='rbf').fit(X_train_v,y_train.values).score(X_test_v,y_test)
y_predict4= svm.SVC(kernel='rbf').fit(X_train_v,y_train.values).predict(X_test_v)
precision_recall_fscore_support(y_test.values,y_predict4,average='micro')
mean_squared_error(y_test.values,y_predict4)
```

2.LSTM

```
class LSTM_Model(nn.Module):
def __init__(self):
super(LSTM_Model, self).__init__()
self.embedding=nn.Embedding.from_pretrained(utils.embedding,freeze=True)

self.lstm = nn.LSTM(input_size=config.embedding_dim, # 300
hidden_size=config.lstm_hidden_size, # 128
num_layers=config.lstm_num_layers, # 2
batch_first=True,
bidirectional=config.lstm_bidirectional,
dropout=config.dropout) # 0.2
self.fc1 = nn.Linear(config.lstm_hidden_size * 2, config.lstm_hidden_size) # [128*2,128]
self.fc2 = nn.Linear(config.lstm_hidden_size, 2) # [128,2]

def forward(self, input):
"""
:param input: [batch_size,max_len],其中max_len表示每个句子有多少单词
:return:
"""

x = self.embedding(input) # [batch_size,max_len,embedding_dim]
# 经过lstm层，x:[batch_size,max_len,2*hidden_size]
# h_n,c_n:[2*num_layers,batch_size,hidden_size]
out, (h_n, c_n) = self.lstm(x)

# 获取两个方向最后一次的h，进行concat
output_fw = h_n[-2, :, :] # [batch_size,hidden_size]
output_bw = h_n[-1, :, :] # [batch_size,hidden_size]
out_put = torch.cat([output_fw, output_bw], dim=-1) # [batch_size,hidden_size*2]

out_fc1 = F.relu(self.fc1(out_put)) # []
out_put = self.fc2(out_fc1)
return out_put
```