南開大學

Python 语言程序设计课程实验报告

基于数据集的虚假信息检测

学　　院　　网络空间安全学院

专　　业　　信息安全

学　　号　　2211044

姓　　名　　陆皓喆

任课教师　　陈晨

一、问题描述

数据集是中文微信消息，包括微信消息的 ***Official Account Name***（公众号名称）,***Title***（标题）,***News Url***（新闻的网址）,***Image Url***（图片的网址）,***Report Content***（新闻报告的评论）,***label***（符号、标签）。

其中，***Title*** 是微信消息的标题，***label*** 是消息的真假标签（0 是 *real* 消息，1 是 *fake* 消息）。

二、实验目的

通过各种机器学习与深度学习的方法来进行对于 ***train.news.csv*** 的模型建立与预测，然后用 ***test.news.csv*** 来进行判断，得出最后的 *AUC*、*ACC* 等值，再对模型进行不断的修改，不断提升预测的正确率。

我们需要测试与分析基于数据集的多种模型的结果差异，从中找出现阶段最好的模型来进行后续的预测；同时更进一步的，我们需要通过此次大作业来探究普适的虚假新闻检测模型与方法。

三、数据集说明

3.1 数据集名称

该数据集名称为 *WeFEND* 数据集。

3.2 数据集介绍

该数据集包含了一系列中文微信消息，比如说：*Official Account Name*（公众号名称）,*Title*（标题）,*News Url*（新闻的网址）,*Image Url*（图片的网址）,*Report Content*（新闻报告的评论）,*label*（符号、标签）。

3.4 数据集成分

如下表所示，数据集中所包含的内容有：

Columns	*Description*
Official Account Name	*The name of official account,news publisher*
Title	*News Title*
News Url	*The url of the news*
Image Url	*The url of the cover image*
Report Content	*The reports from reader,with the ## splited*
label	*Label of news,0 is real and 1 is fake*

3.5 部分数据集

下面的部分我截取了部分数据集中的信息，如：

train.news.csv

```
环球人物,中国反腐风刮到阿根廷，这个美到让人瘫痪的女总统，因为 8 个本子摊上大事了 ,http://mp.weixin.qq.com/s?__biz=MTAzNDI4MDc2MQ==&mid=2651677896&idx=1&sn=87f17336a5aad5eacf12dc1edfc1e7de&chksm=0e63ec9e39146588ba8187a5a45d7ae1aa9b4f4c47c06f9b5f23250937a214f5c9961a838691#rd,http://mmbiz.qpic.cn/mmbiz_jpg/hpcO6kWnPm6cX3MhPyCmgCMpvJ175oDIIQQ9I3wRkRvTnvuOBwz5ZzbZGpYyyyGun4BoAeXrLL9J9RLiaxkibxng/0?wx_fmt=jpeg,内容不符,0
```

test.news.csv

```
私家车第一广播,国务院宣布：生孩子有补助了！明年 1 月起实施，浙江属于这档！ ,http://mp.weixin.qq.com/s?__biz=MTA1NTc0MjE0MA==&mid=2652231657&idx=1&sn=eba7cb45aadbba9537ea3be42b37d117&chksm=0d33d83a3a44512c230ef152c94a3d4fd43ecf34d6ff7ed4e093d605166313525afe28ef6f08#rd,http://mmbiz.qpic.cn/mmbiz_jpg/j27ttKHs7TlFAL5JRURv3XKx5YIIbGrf7OZfibrVRIibXzu2DelTc15XiaY6ZjYloDR9KByia4YehavSxZtoJY2Wcw/0?wx_fmt=jpeg,国务院没有发布过类似信息,0
```

四、方法介绍

4.1 传统机器学习

首先我们利用一些常用的机器学习（*machine-learning*）方法来进行预测。在代码中，我们使用了多种机器学习库，比如说：

1.朴素贝叶斯（*Bayes*）

2.*K* 近邻（*k-NN*）

3.决策树（*Decision Tree*）

4.随机森林（*Random Forest*）

5.梯度上升（*Gradient Boosting*）

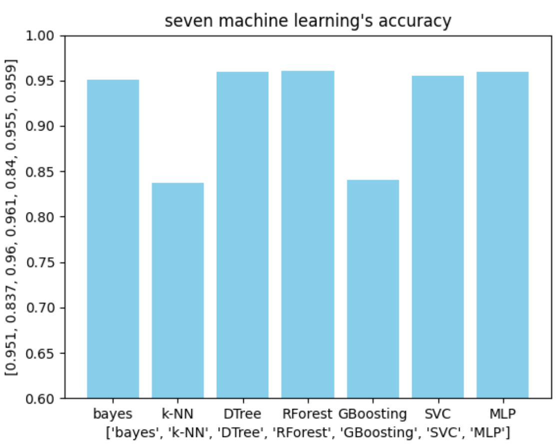
6.支持向量机（*SVM*）

7.神经网络（*neural network*）

最后，我们得出以下的正确率：

```
C:\Users\Lenovo\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Python_HomeWork_By_Luhaozhe
<class 'scipy.sparse._csr.csr_matrix'> <class 'pandas.core.series.Series'>
朴素贝叶斯方法在测试集的准确率为0.951
K近邻方法在测试集的准确率为0.837
决策树方法在测试集的准确率为0.96
随机森林方法在测试集的准确率为0.961
梯度上升方法在测试集的准确率为0.84
支持向量机方法在测试集的准确率为0.955
神经网络方法在测试集的准确率为0.959
```

将七种机器学习的 *ACC* 进行可视化，得到下面的图表：



我们可以发现，其中朴素贝叶斯、决策树、随机森林、神经网络法的正确率比较高，但是所对应的运算时间，贝叶斯的时间较短，另外三种的运算时间均较

长。

我们代入第一种方法的**朴素贝叶斯**的方法，将得出的 *result.csv* 载入系统，得出 *AUC*=0.6137。

可以看出该方法虽然机器学习的模型适合该数据集，但是在测算 *AUC* 的时候由于未对文本进行一些处理，所以导致 *AUC* 不是很高，我们进一步的做一些改进。

4.2 *Bayes*+去除停用词、标点符号

由于只是进行机器学习的预测的情况下，实际上经过测试，正确率和 *AUC* 均不是特别高，所以我们继续采用一些 *NLP* 中的文本处理方式——*jieba* 分词去除停用词和标点符号。

Jieba 分词是一种强大的中文分词组件，可以帮助我们对长的句子进行分块拆解，再进行导入。

我们使用该分词，对于一些常见的分词与停顿词“'是', '的', '了', '在', '和', '有', '更', '与', '对于', '并', '我', '他', '她', '它', '我们', '他们', '她们', '它们'”，我们进行去除，包括一些标点符号，因为这些词语与符号对于句子的正确与否没有任何作用。

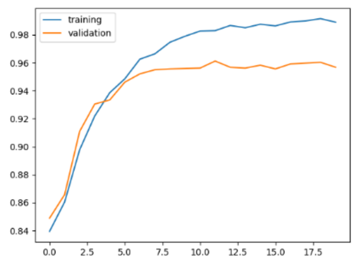
使用该方法得出的 *AUC* 在 0.6735 左右，可以看出也不是特别高。

4.3 进一步使用 *CNN* 模型进行构建

我们开始使用 *CNN* 模型来进行预测。

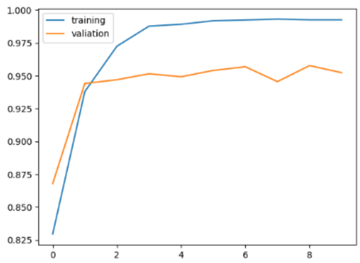
CNN 模型，全称 *Convolutional Neural Networks*，中文名为卷积神经网络，是一种深度学习模型或类似于人工神经网络的多层感知器。

首先我们还是使用最开始时的思路——利用 *jieba* 分词向量来进行对 *CNN* 的优化。经过测试，可以得出 *CNN*+*jieba* 法的 *AUC* 大概是在 0.7323。



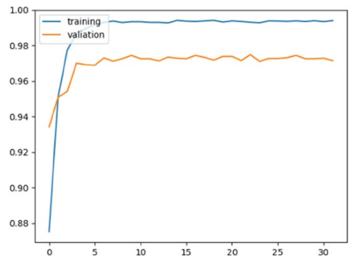
4.4 在 *CNN* 模型中加入大语言文本

为了提高我们的 *AUC*，我们继续在上一环节中的 *CNN* 模型中来进行优化。在网上我找到了一个基于自然语言处理的文件 *sgns.sogounews.bigram-char*，将其加入到我们的代码当中去，得出最后的 *AUC* 为 0.7627。



4.5 在 *CNN* 模型中加入情感色彩分析

进一步的，我们在 *CNN* 模型中继续使用一些网上的大语言模型，我们在此使用一个语言情感分析类的文件，*xmnlp-onnx-models* 文件，进行测试，最后得到 *AUC*=0.8237。

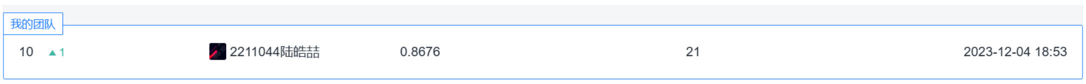


4.6 使用 *Bert* 模型

进一步，我们使用一种名为 *BERT* 的模型来进行训练。

BERT（*Bidirectional Encoder Representations from Transformers*）是一种基于变压器（*Transformer*）架构的预训练语言模型，由 *Google* 在 2018 年提出。*BERT* 的主要创新在于利用双向上下文来预训练模型，这使得模型更好地理解词语的含义和上下文之间的关系。

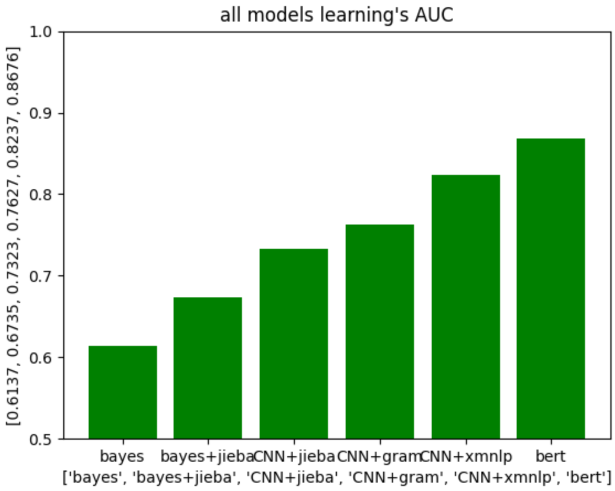
我们利用 *bert* 模型进行模型的测试，最后测试结果，得到 *AUC*=0.8676。



4.7 各种方法的总结

我们通过以上的一些方法，可以得出部分训练模型的 *AUC* 情况。

我们将我们测试的一些模型进行数据可视化，得到以下的柱状图：



五、关键代码细节

5.1 *CNN+jieba* 模型

（一）导入库

```python
import os
import jieba
import re
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Flatten
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import code
import matplotlib.pyplot as plt
```

我简单的介绍一下这些库的作用：

os 库是一个 *Python* 模块，它提供了一种与操作系统交互的方式，允许使用与操作系统相关的功能，例如读取或写入文件系统、操作路径等。

jieba 是一个中文文本分割库。它广泛用于中文中的自然语言处理 （*NLP*）任务，包括分词和词性标记。

re 是 *Python* 中的正则表达式模块。它提供了一组函数，允许您使用正则表达式，这些正则表达式是用于字符串模式匹配的强大工具。

Pandas 和 *numpy* 我就不再详细解释了，后面的 *sklearn* 和 *tensorflow* 是用于机器学习的数据挖掘与神经网络中的，用于定义模型（*Sequential*）、层（*Embedding*、 *Conv1D*、*GlobalMaxPooling1D*、*Dense*、*Flatten*）和文本处理实用程序（*Tokenizer*、*pad_sequences*）的模块。

（二）加载训练数据

```
def load_data(data_path):
    data = pd.read_csv(data_path)
    texts = data["Title"].tolist()
    labels = data["label"].tolist()
    return texts, labels
```

此段代码使用 *pandas* 库读取位于指定位置的 *CSV* 文件，并将数据存储在名为的 *Data* 中。从 *Data* 中提取“*Title*”列，并使用将其转换为 *Python* 列表 *texts*。提取“*label*”列，转化为 *Python* 列表 *labels*，最后返回 *texts* 与 *labels* 的值。

（三）加载测试数据

```
def load_test_data(data_path):
    data = pd.read_csv(data_path)
    texts = data["Title"].tolist()
    ids = data["id"].tolist()
    return texts, ids

```

该代码先从文件中加载数据存储在 *data* 中，然后分别读取 *data* 的 *title* 与 *id*，存储在列表当中，返回 *texts* 与 *ids* 的值。

（四）*jieba* 分词

```
def tokenize(texts):
    tokenized_texts = []
    for text in texts:
        words = jieba.cut(text)
        tokenized_text = ' '.join(words)
        tokenized_texts.append(tokenized_text)
    return tokenized_texts


```

通过 *jieba* 分词的使用对 *words* 进行拆分，存储到 *tokenized_texts* 中去。

（五）去除停用词与标点符号

```python
def remove_stopwords_punctuation(texts):
    stopwords = ['是', '的', '了', '在', '和', '有', '被', '这', '那', '之', '更', '与', '对于', '并', '我', '他', '她',
                 '它', '我们', '他们', '她们', '它们']

    processed_texts = []
    for text in texts:
        text = re.sub(r'[^\w\s]', '', text)
        words = text.split()
        words = [word for word in words if word not in stopwords]
        processed_texts.append(' '.join(words))
    return processed_texts


```

该部分实现了将 *text* 先分裂开来，然后对 *words* 进行遍历，如果遇到停用词（*stopwords*），则去除该词汇，如果没有遇到，那么就将该词汇写入 *processed_texts* 中，利用 *append* 函数即可。

（六）数据集与文本预处理

```python
data_path = 'train.news.csv'  # 数据集路径
texts, labels = load_data(data_path)
# 预处理数据集
tokenized_texts = tokenize(texts)
processed_texts = remove_stopwords_punctuation(tokenized_texts)
# 划分训练集和验证集
train_texts, val_texts, train_labels, val_labels = train_test_split(processed_texts, labels, test_size=0.2,
                                                                    random_state=42)
# 文本向量化
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_texts)
train_sequences = tokenizer.texts_to_sequences(train_texts)
val_sequences = tokenizer.texts_to_sequences(val_texts)
vocab_size = len(tokenizer.word_index) + 1
max_len = 100  # 设定文本的最大长度
train_sequences = pad_sequences(train_sequences, maxlen=max_len, padding='post')
val_sequences = pad_sequences(val_sequences, maxlen=max_len, padding='post')

```

该部分实现了对数据集与文本的一个预处理，就是将我们前面的一些函数进行使用，对文本进行简单的操作。并且设定文本的最大长度，防止文本长度过长而导致不能够很好的进行拆分。

（七）构建 *CNN* 模型

```python
embedding_dim = 100
num_filters = 128
filter_sizes = [3, 4, 5]

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(Conv1D(num_filters, filter_sizes[0], activation='relu'))
model.add(Conv1D(num_filters, filter_sizes[1], activation='relu'))
model.add(Conv1D(num_filters, filter_sizes[2], activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

这一部分是该代码的核心部分，即为构建 *CNN* 模型。

embedding_dim：此变量表示单词嵌入的维度。在这种情况下，输入文本中的每个单词都将表示为长度为 100 的向量。

num_filters：此变量表示每个卷积层中的过滤器数量。

filter_sizes：这是一个列表，其中包含每个卷积层的滤波器大小。

首先，我们创建一个顺序模型，该模型是层的线性堆栈。然后，我们利用 *embedding* 来实现单词向量的嵌入。之后，我们构建了三个卷积层，分别有 128 个过滤器，在前面也规定了滤波器的大小，分别是 3,4,5。其中这些卷积层的激活函数是“*relu*”。之后，此图层对时态数据执行最大池化操作，取时间维度上的最大值。它降低了输入的维度并捕获了最重要的特征。然后，我们对该卷积层进行激活，最后一句话我们指定了优化器（*Adam*）、损失函数（二元分类的 *binary_crossentropy*）和评估指标（准确性）。

（八）模型训练与输出

```
# 训练模型
train_labels = np.array(train_labels)
val_labels = np.array(val_labels)

model.fit(train_sequences, train_labels, validation_data=(val_sequences, val_labels), epochs=10, batch_size=32)

# 加载测试数据
test_path = "test.feature.csv"  # 测试集的路径
test_texts, test_ids = load_test_data(test_path)

# 预处理测试数据
tokenized_test_texts = tokenize(test_texts)
processed_test_texts = remove_stopwords_punctuation(tokenized_test_texts)
test_sequences = tokenizer.texts_to_sequences(processed_test_texts)
test_sequences = pad_sequences(test_sequences, maxlen=max_len, padding='post')

# 进行预测
y_pred = (model.predict(test_sequences) > 0.5).astype(int)

# 输出测试结果
results_df = pd.DataFrame({'id': test_ids, 'label': y_pred.flatten()})
results_df.to_csv('result_cnn_jieba.csv', index=False)
```

该代码实现了对训练数据的加载以及预处理，最后通过训练的模型来进行对 *test* 数据进行预测，最后输出一个 *csv* 文件“*result_cnn_jieba.csv*”。

5.2 *bert* 模型部分代码

（1）设置随机种子

```python
def set_seed(seed: int):
    """
    Helper function for reproducible behavior to set the seed in ``random``, ``numpy``, ``torch`` and/or ``tf`` (if
    installed).

    Args:
        seed (:obj:`int`): The seed to set.
    """
    random.seed(seed)
    np.random.seed(seed)
    if is_torch_available():
        torch.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        # ^^ safe to call this function even if cuda is not available
    if is_tf_available():
        import tensorflow as tf

        tf.random.set_seed(seed)

set_seed(123)
```

set_seed 函数的目的是确保当运行涉及随机性的代码。例如，在该部分代码的神经网络中初始化权重时，每次运行时都会获得相同的随机值，从而允许实验的可重复性。种子值是任意选择，可以替换为不同随机序列的任何整数。

（2）训练部分

```python
training_args = TrainingArguments(
    output_dir='/results',          # output directory
    num_train_epochs=2,              # total number of training epochs
    per_device_train_batch_size=10,  # batch size per device during training
    per_device_eval_batch_size=20,   # batch size for evaluation
    warmup_steps=100,                # number of warmup steps for learning rate scheduler
    logging_dir='/results',
    # directory for storing logs
    load_best_model_at_end=True,     # load the best model when finished training (default metric is loss)
    # but you can specify `metric_for_best_model` argument to change to accuracy or other metric
    logging_steps=200,               # log & save weights each logging_steps
    save_steps=200,
    evaluation_strategy="steps",     # evaluate each `logging_steps`
)

trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=computer_metrics,
)

trainer.train()
model.save_pretrained('./cache/model_bert1')
tokenizer.save_pretrained('./cache/tokenizer1')
```

该部分代码说明了训练的过程，一共训练两次，训练时每个设备训练的大小是 10，用于评估的大小是 20，学习率调度程序的预热步骤数是 100，我们为了模型的可跑性，选择了 200 的保存模型间隔，就是说每测试 200 个量，就进行一次检验，按照 200 个的数量进行评估。

（3）文本索引处理

```python
def get_prediction(text, convert_to_label=False):
    # prepare our text into tokenized sequence
    inputs = tokenizer(text, padding=True, truncation=True, max_length=max_length, return_tensors="pt").to("cuda")
    # perform inference to our model
    outputs = model(**inputs)
    # get output probabilities by doing softmax
    probs = outputs[0].softmax(1)
    # executing argmax function to get the candidate label
    d = {
        0: "reliable",
        1: "fake"
    }
    if convert_to_label:
        return d[int(probs.argmax())]
    else:
        return int(probs.argmax())
```

首先我们使用 *tokenizer* 进行对文本的截断，生成的标记化序列将转换为 *PyTorch* 张量并移动到 *GPU*。然后将获取的数据传输到模型中，*softmax* 表示模型的输出，获取到每个类的概率。然后我们使用 *argmax* 函数来查询出现概率最高的索引。最后，我们进行一个标签映射，返回我们所需要的索引值即可。

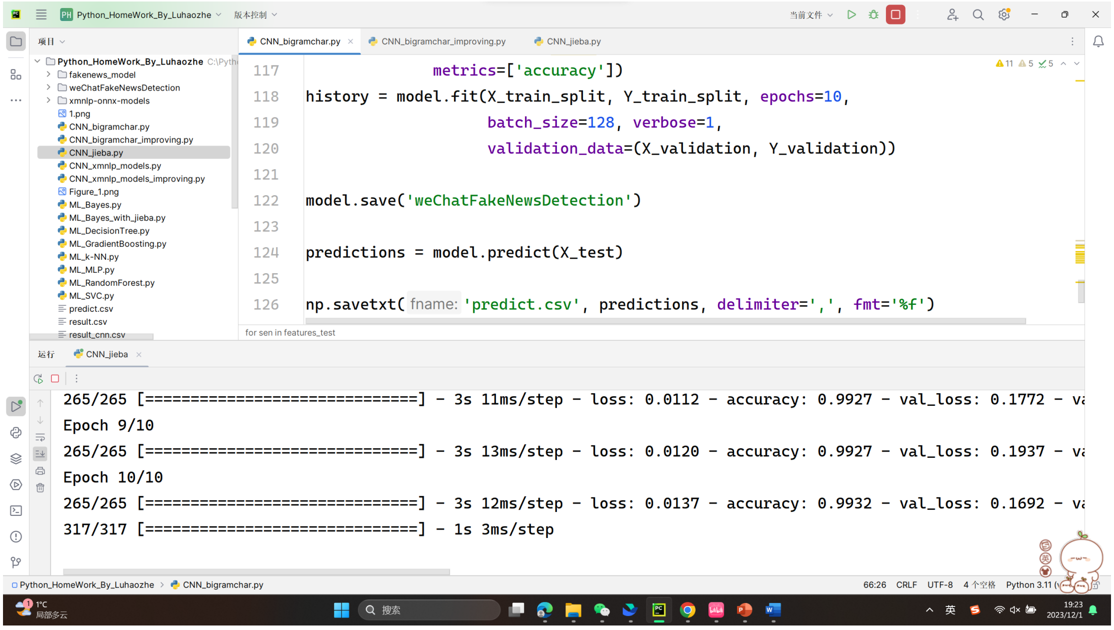
（4）数据处理与输出

```python
test_df = pd.read_csv("test.feature.csv")
# make a copy of the testing set
new_df = test_df.copy()
# add a new column that contains the author, title and article content
new_df["Report Content"] = new_df["Report Content"].apply(lambda x:x.split("##"))
t = pd.DataFrame(train_df.astype(str))
new_df['Title'] = t['Title'].apply(cleaning)
new_df['Report Content'] = t['Report Content'].apply(cleaning)
new_df['Ofiicial Account Name'] = t['Ofiicial Account Name']
new_df = new_df[columns]
new_df["new_text"] = new_df["Ofiicial Account Name"].astype(str) + " " + new_df["Title"].astype(str) + " " + new_df["Report Content"].astype(str)
new_df["label"] = new_df["new_text"].apply(get_prediction)
# make the submission file
final_df = new_df[["id", "label"]]
final_df.to_csv("result_bert.csv", index=False)
```

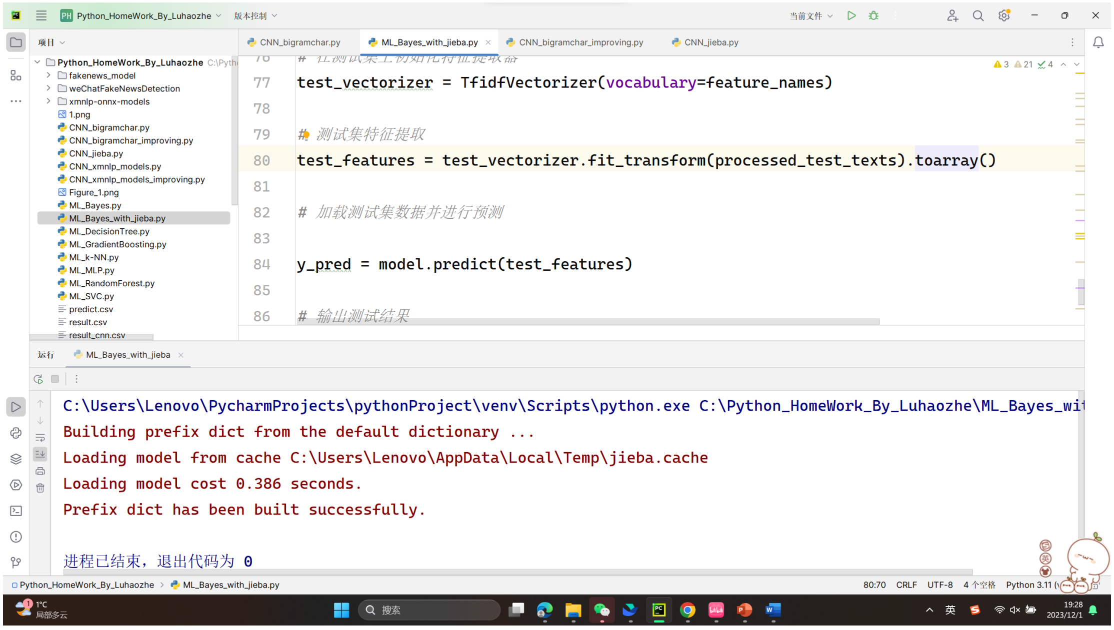
该部分与前面的程序几乎差不多，就是将训练集的数据进行导入，然后我们获取一个新的列表“*new_text*”，然后然后训练我们自己的一个模型，然后再对 *test* 数据集进行一个预测，最后输出我们的结果。

六、运行截图

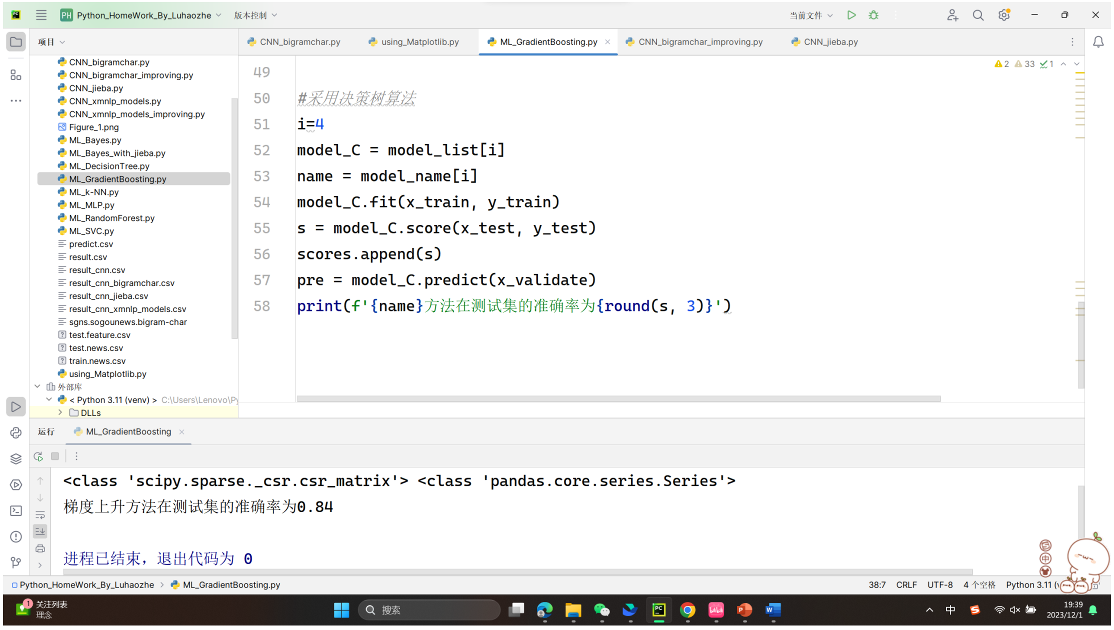
CNN_jieba 运行截图：



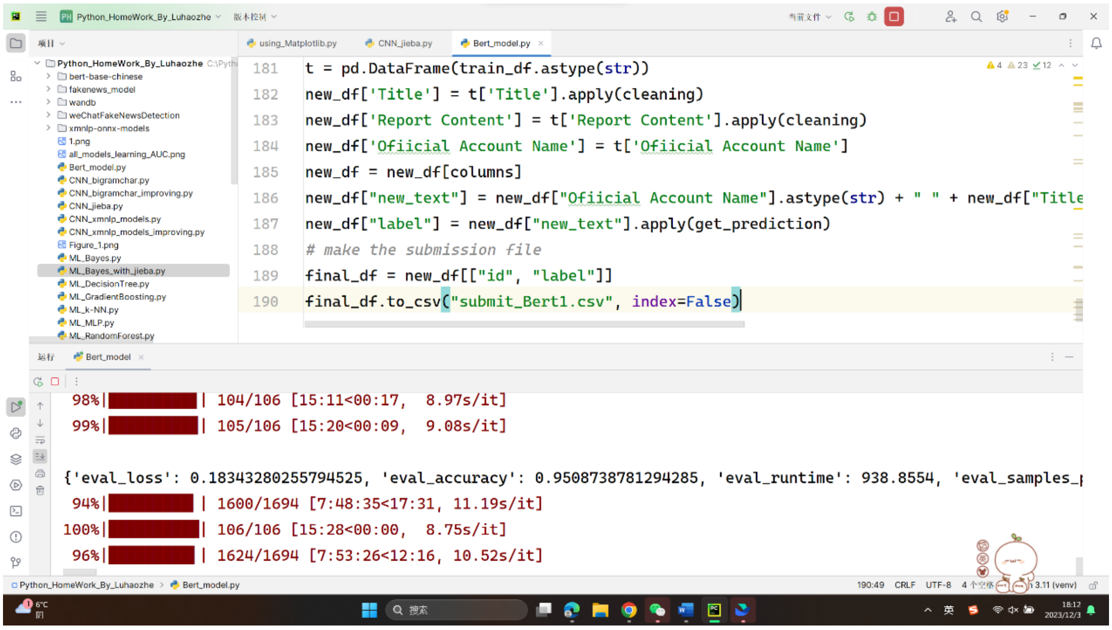
ML_bayes_with_jieba 运行截图：



ML_GradientBoosting 运行截图：

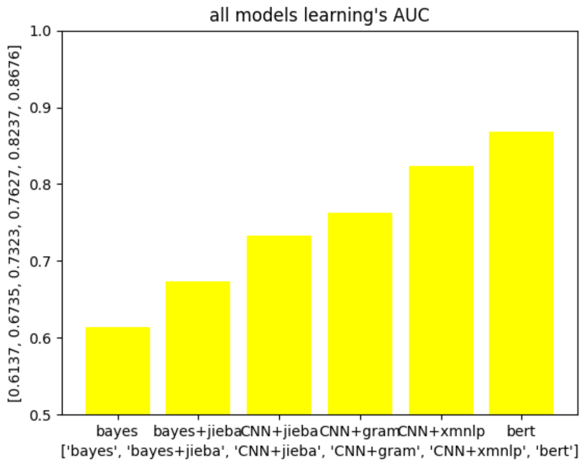


Bert 运行截图：



七、实验结果

经过各种 *models* 的训练与测试，我们得到了以下的结论：



我们通过 *matplotlib* 的可视化，得出 *bayes* 等六种方法测试出的 *AUC*，发现 *bert* 的 *AUC* 是最高的。

在运算时间的比较中，我们可以得出以下的结论：

虽然 *bert* 的正确率相比于别的几种都相对来说高一些，但是代码运行的时间相对与其他的模型来说会长很多（跑了 8 个多小时），但是相对应的正确率和 *AUC* 都是相对高的。

八、结果分析

如果追求 *ACC* 与 *AUC* 的话，我们可以选择使用 *bert* 模型。

如果对于时间的要求比较严格的话，在正确率与 *AUC* 都相对比较高的情况下，我们也可以选择使用相对来说较为快速的模型，比如说 *bayes+jieba* 或者 *CNN+xmnlp*。

九、总结与反思

从这次的虚假新闻的检测大作业中，我从对机器学习与深度学习的略懂皮毛到了现在的基本了解了各种模型的原理和如何使用各种模型。

我先是对一些简单的机器学习的模型进行了学习，发现这些模型的使用方法无非都是如出一辙，于是我先是完成了该部分的一些代码，但是发现正确率与

AUC 其实并不是特别的高。

然后我在网上看到了一种中文语言分块模型——*jieba* 分词。于是我把这个方法用在了机器学习的 *bayes* 当中，将 *bayes* 与 *jieba* 分词联系在一起，我得出了第二个 *AUC*，这一次的值要比之前的单纯的及其学习要高一些。

但是 *AUC* 还是只停留在 0.7 左右。于是我又开始想其他的方法——在网上我进行深度学习与机器学习的方法的搜索，发现了一种分层架构模型——*CNN*。于是我对照着网上的思路把模型进行了一个初步的构建。跑出来之后，我又想着能不能对该程序进行一个优化，于是我分别加入了 *jieba* 分词、大语言模型以及 xmnlp 的语言情感分析，正确率一次比一次高，*AUC* 稳步上升。

最后，我将 *CNN* 模型的 *AUC* 提升到了 0.82 左右。

在网上我还发现了一些大语言模型的处理操作方法，比如说常见的有 *bert*、*Longformer*、*ERNIE* 模型等等。我在此处选择了 *bert* 模型进行构建，在构建的过程中有一些困难，但是最后也是成功的把结果跑了出来。*bert* 的 *AUC* 大概是在 0.867 左右，达到了我的预期值。

通过这次的虚假新闻检测，我从中学到了许多的机器学习深度学习的方法，学会了如何去架构模型，使用模型，导入数据，筛选数据，处理信息……

因为在此项大作业中，我只是对新闻的 *title* 去进行了一个分析，所以还有许多的量没有去考虑，比如说一些 *url*，一些照片等等。可以在后期继续对此项目进行一个优化，提高正确率。

由于提交时间紧迫，在后面一阶段，我仅仅是把 *bert* 模型跑了出来，对于一些新出现的模型，比如说 *ERINE* 等，我会继续研究，包括也会对原先的模型继续优化调参，提升 *AUC*。

十、附录

此部分提供各部分的源代码。

1. ML_Bayes.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from    sklearn.svm import LinearSVC
```

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
data=pd.read_csv("train.news.csv")
data_validate=pd.read_csv("test.feature.csv")
x,y,x_validate=data['Title'],data['label'],data_validate['Title']
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform(x)
x_validate=vectorizer.transform(x_validate)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state = 0)
x_validate1,x_validate2=train_test_split(x_validate,test_size=0.5,random_state=0)
print(type(x_train),type(y_train))
# 朴素贝叶斯
model2 = MultinomialNB()
#K 近邻
model3 = KNeighborsClassifier(n_neighbors=50)
# 决策树
model4 = DecisionTreeClassifier(random_state=77)
# 随机森林
model5 = RandomForestClassifier(n_estimators=500, max_features='sqrt', random_state=10)
# 梯度上升
model6 = GradientBoostingClassifier(random_state=123)
# 支持向量机
model7 = SVC(kernel="rbf", random_state=77)
# 神经网络
model8 = MLPClassifier(hidden_layer_sizes=(16, 8), random_state=77, max_iter=10000)
model_list = [ model2, model3, model4, model5, model6, model7, model8]
model_name = [ '朴素贝叶斯', 'K 近邻', '决策树', '随机森林', '梯度上升', '支持向量机', '神经网络']
scores=[]
#采用 Bayes 模型
i=0
model_C = model_list[i]
name = model_name[i]
model_C.fit(x_train, y_train)
s = model_C.score(x_test, y_test)
scores.append(s)
pre = model_C.predict(x_validate)
print(f'{name}方法在测试集的准确率为{round(s, 3)}')
```

2. ML_DecisionTree.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from    sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
data=pd.read_csv("train.news.csv")
data_validate=pd.read_csv("test.feature.csv")
x,y,x_validate=data['Title'],data['label'],data_validate['Title']
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform(x)
x_validate=vectorizer.transform(x_validate)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state = 0)
x_validate1,x_validate2=train_test_split(x_validate,test_size=0.5,random_state=0)
print(type(x_train),type(y_train))
# 朴素贝叶斯
model2 = MultinomialNB()
#K 近邻
model3 = KNeighborsClassifier(n_neighbors=50)
# 决策树
model4 = DecisionTreeClassifier(random_state=77)
# 随机森林
model5 = RandomForestClassifier(n_estimators=500, max_features='sqrt', random_state=10)
# 梯度上升
model6 = GradientBoostingClassifier(random_state=123)
# 支持向量机
model7 = SVC(kernel="rbf", random_state=77)
# 神经网络
model8 = MLPClassifier(hidden_layer_sizes=(16, 8), random_state=77, max_iter=10000)
model_list = [ model2, model3, model4, model5, model6, model7, model8]
model_name = [ '朴素贝叶斯', 'K 近邻', '决策树', '随机森林', '梯度上升', '支持向量机', '神经网
络']
scores=[]
#采用决策树算法
i=2
model_C = model_list[i]
name = model_name[i]
```

```
model_C.fit(x_train, y_train)
s = model_C.score(x_test, y_test)
scores.append(s)
pre = model_C.predict(x_validate)
print(f'{name}方法在测试集的准确率为{round(s, 3)}')
```

3. ML_GradientBoosting.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from    sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
data=pd.read_csv("train.news.csv")
data_validate=pd.read_csv("test.feature.csv")
x,y,x_validate=data['Title'],data['label'],data_validate['Title']
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform(x)
x_validate=vectorizer.transform(x_validate)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state = 0)
x_validate1,x_validate2=train_test_split(x_validate,test_size=0.5,random_state=0)
print(type(x_train),type(y_train))
# 朴素贝叶斯
model2 = MultinomialNB()
#K 近邻
model3 = KNeighborsClassifier(n_neighbors=50)
# 决策树
model4 = DecisionTreeClassifier(random_state=77)
# 随机森林
model5 = RandomForestClassifier(n_estimators=500, max_features='sqrt', random_state=10)
# 梯度上升
model6 = GradientBoostingClassifier(random_state=123)
# 支持向量机
model7 = SVC(kernel="rbf", random_state=77)
# 神经网络
model8 = MLPClassifier(hidden_layer_sizes=(16, 8), random_state=77, max_iter=10000)
model_list = [ model2, model3, model4, model5, model6, model7, model8]
```

```
model_name = [ '朴素贝叶斯', 'K 近邻', '决策树', '随机森林', '梯度上升', '支持向量机', '神经网络']
scores=[]
#采用决策树算法
i=4
model_C = model_list[i]
name = model_name[i]
model_C.fit(x_train, y_train)
s = model_C.score(x_test, y_test)
scores.append(s)
pre = model_C.predict(x_validate)
print(f'{name}方法在测试集的准确率为{round(s, 3)}')
```

4. ML_k-NN.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from  sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
data=pd.read_csv("train.news.csv")
data_validate=pd.read_csv("test.feature.csv")
x,y,x_validate=data['Title'],data['label'],data_validate['Title']
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform(x)
x_validate=vectorizer.transform(x_validate)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state = 0)
x_validate1,x_validate2=train_test_split(x_validate,test_size=0.5,random_state=0)
print(type(x_train),type(y_train))
# 朴素贝叶斯
model2 = MultinomialNB()
#K 近邻
model3 = KNeighborsClassifier(n_neighbors=50)
# 决策树
model4 = DecisionTreeClassifier(random_state=77)
# 随机森林
model5 = RandomForestClassifier(n_estimators=500, max_features='sqrt', random_state=10)
# 梯度上升
```

```
model6 = GradientBoostingClassifier(random_state=123)
# 支持向量机
model7 = SVC(kernel="rbf", random_state=77)
# 神经网络
model8 = MLPClassifier(hidden_layer_sizes=(16, 8), random_state=77, max_iter=10000)
model_list = [ model2, model3, model4, model5, model6, model7, model8]
model_name = [ '朴素贝叶斯', 'K 近邻', '决策树', '随机森林', '梯度上升', '支持向量机', '神经网络']
scores=[]
#采用 k-NN 算法
i=5
model_C = model_list[i]
name = model_name[i]
model_C.fit(x_train, y_train)
s = model_C.score(x_test, y_test)
scores.append(s)
pre = model_C.predict(x_validate)
print(f'{name}方法在测试集的准确率为{round(s, 3)}')
```

5．ML_MLP.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from   sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
data=pd.read_csv("train.news.csv")
data_validate=pd.read_csv("test.feature.csv")
x,y,x_validate=data['Title'],data['label'],data_validate['Title']
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform(x)
x_validate=vectorizer.transform(x_validate)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state = 0)
x_validate1,x_validate2=train_test_split(x_validate,test_size=0.5,random_state=0)
print(type(x_train),type(y_train))
# 朴素贝叶斯
model2 = MultinomialNB()
# K 近邻
```

```
model3 = KNeighborsClassifier(n_neighbors=50)
# 决策树
model4 = DecisionTreeClassifier(random_state=77)
# 随机森林
model5 = RandomForestClassifier(n_estimators=500, max_features='sqrt', random_state=10)
# 梯度上升
model6 = GradientBoostingClassifier(random_state=123)
# 支持向量机
model7 = SVC(kernel="rbf", random_state=77)
# 神经网络
model8 = MLPClassifier(hidden_layer_sizes=(16, 8), random_state=77, max_iter=10000)
model_list = [ model2, model3, model4, model5, model6, model7, model8]
model_name = [ '朴素贝叶斯', 'K 近邻', '决策树', '随机森林', '梯度上升', '支持向量机', '神经网络']
scores=[]
#采用 MLP 算法
i=6
model_C = model_list[i]
name = model_name[i]
model_C.fit(x_train, y_train)
s = model_C.score(x_test, y_test)
scores.append(s)
pre = model_C.predict(x_validate)
print(f'{name}方法在测试集的准确率为{round(s, 3)}')
```

6. ML_RandomForest.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from    sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
data=pd.read_csv("train.news.csv")
data_validate=pd.read_csv("test.feature.csv")
x,y,x_validate=data['Title'],data['label'],data_validate['Title']
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform(x)
x_validate=vectorizer.transform(x_validate)
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state = 0)
x_validate1,x_validate2=train_test_split(x_validate,test_size=0.5,random_state=0)
print(type(x_train),type(y_train))
# 朴素贝叶斯
model2 = MultinomialNB()
#K 近邻
model3 = KNeighborsClassifier(n_neighbors=50)
# 决策树
model4 = DecisionTreeClassifier(random_state=77)
# 随机森林
model5 = RandomForestClassifier(n_estimators=500, max_features='sqrt', random_state=10)
# 梯度上升
model6 = GradientBoostingClassifier(random_state=123)
# 支持向量机
model7 = SVC(kernel="rbf", random_state=77)
# 神经网络
model8 = MLPClassifier(hidden_layer_sizes=(16, 8), random_state=77, max_iter=10000)
model_list = [ model2, model3, model4, model5, model6, model7, model8]
model_name = [ '朴素贝叶斯', 'K 近邻', '决策树', '随机森林', '梯度上升', '支持向量机', '神经网络']
scores=[]
#采用随机森林算法
i=3
model_C = model_list[i]
name = model_name[i]
model_C.fit(x_train, y_train)
s = model_C.score(x_test, y_test)
scores.append(s)
pre = model_C.predict(x_validate)
print(f'{name}方法在测试集的准确率为{round(s, 3)}')
```

7. ML_SVC.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from   sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

```
data=pd.read_csv("train.news.csv")
data_validate=pd.read_csv("test.feature.csv")
x,y,x_validate=data['Title'],data['label'],data_validate['Title']
vectorizer=TfidfVectorizer()
x=vectorizer.fit_transform(x)
x_validate=vectorizer.transform(x_validate)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,stratify=y,random_state = 0)
x_validate1,x_validate2=train_test_split(x_validate,test_size=0.5,random_state=0)
print(type(x_train),type(y_train))
# 朴素贝叶斯
model2 = MultinomialNB()
#K 近邻
model3 = KNeighborsClassifier(n_neighbors=50)
# 决策树
model4 = DecisionTreeClassifier(random_state=77)
# 随机森林
model5 = RandomForestClassifier(n_estimators=500, max_features='sqrt', random_state=10)
# 梯度上升
model6 = GradientBoostingClassifier(random_state=123)
# 支持向量机
model7 = SVC(kernel="rbf", random_state=77)
# 神经网络
model8 = MLPClassifier(hidden_layer_sizes=(16, 8), random_state=77, max_iter=10000)
model_list = [ model2, model3, model4, model5, model6, model7, model8]
model_name = [ '朴素贝叶斯', 'K 近邻', '决策树', '随机森林', '梯度上升', '支持向量机', '神经网
络']
scores=[]
#采用随机森林算法
i=5
model_C = model_list[i]
name = model_name[i]
model_C.fit(x_train, y_train)
s = model_C.score(x_test, y_test)
scores.append(s)
pre = model_C.predict(x_validate)
print(f'{name}方法在测试集的准确率为{round(s, 3)}')
```

8. ML_Bayes_with_jieba.py

```
import os
import jieba
import re
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

```python
# 加载训练数据
def load_data(data_path):
    data = pd.read_csv(data_path)
    texts = data["Title"].tolist()
    names = data["Ofiicial Account Name"].tolist()
    combined_texts = [str(x) + str(y) for x, y in zip(texts, names)]
    labels = data["label"].tolist()
    return combined_texts, labels

# 加载测试数据
def load_test_data(data_path):
    data = pd.read_csv(data_path)
    texts = data["Title"].tolist()
    ids = data["id"].tolist()
    return texts, ids

# 分词
def tokenize(texts):
    tokenized_texts = []
    for text in texts:
        words = jieba.cut(text)
        tokenized_text = ' '.join(words)
        tokenized_texts.append(tokenized_text)
    return tokenized_texts

# 去除停用词和标点符号
def remove_stopwords_punctuation(texts):
    stopwords = ['是', '的', '了', '在', '和', '有', '更', '与', '对于', '并', '我', '他', '她', '它', '我们', '他
们', '她们', '它们']
    processed_texts = []
    for text in texts:
        text = re.sub(r'[^\w\s]', '', text)
        words = text.split()
        words = [word for word in words if word not in stopwords]
        processed_texts.append(' '.join(words))
    return processed_texts

# 加载数据集
data_path = 'train.news.csv'
texts, labels = load_data(data_path)


# 预处理数据集
tokenized_texts = tokenize(texts)
```

```python
processed_texts = remove_stopwords_punctuation(tokenized_texts)    #进行停用词的查找与去除

# 训练集特征提取
vectorizer = TfidfVectorizer()
features = vectorizer.fit_transform(processed_texts).toarray()

# 保存特征提取器的配置
feature_names = vectorizer.get_feature_names_out()

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score


# 训练模型
model = MultinomialNB()
model.fit(features, labels)

# 加载测试数据
test_path = "test.feature.csv"    # 数据集的路径
test_texts, test_ids= load_test_data(test_path)

# 分词和去除停用词、标点符号
tokenized_test_texts = tokenize(test_texts)
processed_test_texts = remove_stopwords_punctuation(tokenized_test_texts)

# 在测试集上初始化特征提取器
test_vectorizer = TfidfVectorizer(vocabulary=feature_names)

# 测试集特征提取
test_features = test_vectorizer.fit_transform(processed_test_texts).toarray()

# 加载测试集数据并进行预测

y_pred = model.predict(test_features)

# 输出测试结果
results_df = pd.DataFrame({'id': test_ids, 'label': y_pred})
results_df.to_csv('result.csv', index=False)    #输出 result.csv 测试点集 p
```

9. CNN_jieba.py

```python
import os
import jieba
```

```python
import re
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Flatten
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import code
import matplotlib.pyplot as plt

# 加载训练数据
def load_data(data_path):
    data = pd.read_csv(data_path)
    texts = data["Title"].tolist()
    labels = data["label"].tolist()
    return texts, labels


# 加载测试数据
def load_test_data(data_path):
    data = pd.read_csv(data_path)
    texts = data["Title"].tolist()
    ids = data["id"].tolist()
    return texts, ids


# 分词
def tokenize(texts):
    tokenized_texts = []
    for text in texts:
        words = jieba.cut(text)
        tokenized_text = ' '.join(words)
        tokenized_texts.append(tokenized_text)
    return tokenized_texts


# 去除停用词和标点符号
def remove_stopwords_punctuation(texts):
    stopwords = ['是', '的', '了', '在', '和', '有', '被', '这', '那', '之', '更', '与', '对于', '并', '我', '他', '她',
                 '它', '我们', '他们', '她们', '它们']
```

```
    processed_texts = []
    for text in texts:
        text = re.sub(r'[^\w\s]', '', text)
        words = text.split()
        words = [word for word in words if word not in stopwords]
        processed_texts.append(' '.join(words))
    return processed_texts


# 加载数据集
data_path = 'train.news.csv' # 数据集路径
texts, labels = load_data(data_path)

# 预处理数据集
tokenized_texts = tokenize(texts)
processed_texts = remove_stopwords_punctuation(tokenized_texts)

# 划分训练集和验证集
train_texts, val_texts, train_labels, val_labels = train_test_split(processed_texts, labels,
test_size=0.2,

random_state=42)

# 文本向量化
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_texts)

train_sequences = tokenizer.texts_to_sequences(train_texts)
val_sequences = tokenizer.texts_to_sequences(val_texts)

vocab_size = len(tokenizer.word_index) + 1

max_len = 100   # 设定文本的最大长度
train_sequences = pad_sequences(train_sequences, maxlen=max_len, padding='post')
val_sequences = pad_sequences(val_sequences, maxlen=max_len, padding='post')

# 构建 CNN 模型
embedding_dim = 100
num_filters = 128
filter_sizes = [3, 4, 5]

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(Conv1D(num_filters, filter_sizes[0], activation='relu'))
```

```
model.add(Conv1D(num_filters, filter_sizes[1], activation='relu'))
model.add(Conv1D(num_filters, filter_sizes[2], activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


# 训练模型
train_labels = np.array(train_labels)
val_labels = np.array(val_labels)

model.fit(train_sequences, train_labels, validation_data=(val_sequences, val_labels), epochs=10,
batch_size=32)

# 加载测试数据
test_path = "test.feature.csv"   # 测试集的路径
test_texts, test_ids = load_test_data(test_path)

# 预处理测试数据
tokenized_test_texts = tokenize(test_texts)
processed_test_texts = remove_stopwords_punctuation(tokenized_test_texts)
test_sequences = tokenizer.texts_to_sequences(processed_test_texts)
test_sequences = pad_sequences(test_sequences, maxlen=max_len, padding='post')

# 进行预测
y_pred = (model.predict(test_sequences) > 0.5).astype(int)

# 输出测试结果
results_df = pd.DataFrame({'id': test_ids, 'label': y_pred.flatten()})
results_df.to_csv('result_cnn_jieba.csv', index=False)
```

10. CNN_bigramchar.py

```
import pandas as pd
import jieba
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import json

# 获取数据
```

```python
train_data = pd.read_csv('train.news.csv')
test_data = pd.read_csv('test.feature.csv')

print("finish read csv")

# 获取预训练 word2vec 并构建词表
word2vec = open("sgns.sogounews.bigram-char", "r", encoding='UTF-8')
t = word2vec.readline().split()
n, dimension = int(t[0]), int(t[1])
print(n)
print(dimension)
wordAndVec = word2vec.readlines()
wordAndVec = [i.split() for i in wordAndVec]
vectorsMap = []
word2index = {}
index2word = {}
for i in range(n):
    vectorsMap.append(list(map(float, wordAndVec[i][len(wordAndVec[i]) - dimension:])))
    word2index[wordAndVec[i][0]] = i
    index2word[i] = wordAndVec[i][0]

word2vec.close()
print("finish reading")

#jieba 分词与词向量构建
features_train = []
features_test = []
for text in train_data['Title']:
    word_feature = []
    for word in jieba.cut(text):
        if word in word2index:
            word_feature.append(vectorsMap[word2index[word]])
    features_train.append(word_feature)

for text in test_data['Title']:
    word_feature = []
    for word in jieba.cut(text):
        if word in word2index:
            word_feature.append(vectorsMap[word2index[word]])
    features_test.append(word_feature)


print("finish creating features")
```

```python
# 模型输入构建

max_len1 = max([len(i) for i in features_train])
max_len2 = max([len(i) for i in features_test])
max_len = max(max_len1, max_len2)
X_train = []
X_test = []
for sen in features_train:
    tl = sen
    tl += [[0] * 300] * (max_len - len(tl))
    X_train.append(tl)
for sen in features_test:
    tl = sen
    tl += [[0] * 300] * (max_len - len(tl))
    X_test.append(tl)

print("finish creating X_train X_test")

Y_train = train_data['label']

X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)

np.random.seed(1)
np.random.shuffle(X_train)
np.random.seed(1)
np.random.shuffle(Y_train)

split = len(X_train) // 3
X_validation = X_train[:split]
X_train_split = X_train[split:]
Y_validation = Y_train[:split]
Y_train_split = Y_train[split:]

print("finish creating X_train_split, X_validation, Y_split, Y_validation")



# 模型构建
def cnn(X_train):
    model = tf.keras.Sequential([
        tf.keras.layers.Convolution1D(input_shape=(X_train.shape[1], X_train.shape[2]),
                                      filters=128, kernel_size=3, activation='relu'),
```

```
            tf.keras.layers.MaxPool1D(),
            tf.keras.layers.Convolution1D(128, 4, activation='relu'),
            tf.keras.layers.MaxPool1D(),
            tf.keras.layers.Convolution1D(64, 5),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dropout(rate=0.5),
            tf.keras.layers.Dense(2, activation='softmax'),
    ])
    print(model.summary())
    return model


print("finish creating model")

# 模型训练
model = cnn(X_train)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
history = model.fit(X_train_split, Y_train_split, epochs=10,
                    batch_size=128, verbose=1,
                    validation_data=(X_validation, Y_validation))

model.save('weChatFakeNewsDetection')

predictions = model.predict(X_test)

np.savetxt('predict.csv', predictions, delimiter=',', fmt='%f')



plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'valiation'], loc='upper left')
plt.show()
```

11. CNN_bigramchar_improving.py

```
import pandas as pd
excel_file_path='predict.csv'
df=pd.read_csv(excel_file_path)

for i in range(10140):
    value_ij=df.iloc[i-1,1]
    if value_ij>0.5:
        df.iloc[i-1,1]=1
```

```
        else:
            df.iloc[i-1,1]=0

df.to_csv('result_cnn_bigramchar.csv',index=False)
```

12. CNN_xmnlp_models.py

```
import pandas as pd
import jieba
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import xmnlp

# 获取数据
train_data = pd.read_csv('train.news.csv')
test_data = pd.read_csv('test.feature.csv')
print("finish read csv")



# 获取预训练 word2vec 并构建词表
word2vec = open("sgns.sogounews.bigram-char", "r", encoding='UTF-8')
t = word2vec.readline().split()
n, dimension = int(t[0]), int(t[1])
print(n)
print(dimension)
wordAndVec = word2vec.readlines()
wordAndVec = [i.split() for i in wordAndVec]
vectorsMap = []
word2index = {}
index2word = {}
for i in range(n):
    vectorsMap.append(list(map(float, wordAndVec[i][len(wordAndVec[i]) - dimension:])))
    word2index[wordAndVec[i][0]] = i
    index2word[i] = wordAndVec[i][0]

word2vec.close()
print("finish reading")

# 情感判断
def SentimentAnalysis(text):
    xmnlp.set_model('./xmnlp-onnx-models')
    x = list(xmnlp.sentiment(text))
    x.extend([0.]*298)
```

```python
    return x

#jieba 分词与词向量构建
features_train = []
features_test = []
for text,comment in zip(train_data['Title'],train_data['Report Content']):
    #print(len(SentimentAnalysis(comment)),len([0]))
    word_feature = [SentimentAnalysis(comment)]
    for word in jieba.cut(text):
        if word in word2index:
            word_feature.append(vectorsMap[word2index[word]])
    features_train.append(word_feature)

for text,comment in zip(test_data['Title'],test_data['Report Content']):
    word_feature = [SentimentAnalysis(comment)]
    #word_feature=[]
    for word in jieba.cut(text):
        if word in word2index:
            word_feature.append(vectorsMap[word2index[word]])
    features_test.append(word_feature)


print("finish creating features")





# 模型输入构建

max_len1 = max([len(i) for i in features_train])
max_len2 = max([len(i) for i in features_test])
max_len = max(max_len1, max_len2)
X_train = []
X_test = []
for sen in features_train:
    tl = sen
    tl += [[0] * 300] * (max_len - len(tl))
    X_train.append(tl)
for sen in features_test:
    tl = sen
    tl += [[0] * 300] * (max_len - len(tl))
    X_test.append(tl)
```

```
print("finish creating X_train X_test")

Y_train = train_data['label']

X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)

np.random.seed(1)
np.random.shuffle(X_train)
np.random.seed(1)
np.random.shuffle(Y_train)

split = len(X_train) // 3
X_validation = X_train[:split]
X_train_split = X_train[split:]
Y_validation = Y_train[:split]
Y_train_split = Y_train[split:]

print("finish creating X_train_split, X_validation, Y_split, Y_validation")



# 模型构建
def cnn(X_train):
	model = tf.keras.Sequential([
		tf.keras.layers.Convolution1D(input_shape=(X_train.shape[1], X_train.shape[2]),
							filters=128, kernel_size=3, activation='relu'),
		tf.keras.layers.Dropout(rate=0.5),	# Added dropout layer after the first convolutional 
layer
		tf.keras.layers.MaxPool1D(),
		tf.keras.layers.Convolution1D(128, 4, activation='relu'),
		tf.keras.layers.Dropout(rate=0.5),	# Added dropout layer after the second 
convolutional layer
		tf.keras.layers.MaxPool1D(),
		tf.keras.layers.Convolution1D(64, 5),
		tf.keras.layers.Dropout(rate=0.5),	# Added dropout layer after the third convolutional 
layer
		tf.keras.layers.Flatten(),
		tf.keras.layers.Dropout(rate=0.5),	# Dropout layer before the dense layer (unchanged)
		tf.keras.layers.Dense(2, activation='softmax'),
	])
	# lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
	#	initial_learning_rate=1e-2,
```

```
    #       decay_steps=10000,
    #       decay_rate=0.9)
    # model.compile(loss='sparse_categorical_crossentropy',
    #               optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
    #               metrics=['accuracy'])
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=tf.keras.optimizers.Adam(),
                  metrics=['accuracy'])
    print(model.summary())
    return model


# 模型训练
model = cnn(X_train)

print("finish creating model")

history = model.fit(X_train_split, Y_train_split, epochs=20,
                    batch_size=128, verbose=1,
                    validation_data=(X_validation, Y_validation))

model.save('fakenews_model')

predictions = model.predict(X_test)

np.savetxt('predict.csv', predictions, delimiter=',', fmt='%f')



plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc='upper left')
plt.savefig('1.png')
```

13. CNN_xmnlp_models_improving.py

```
import pandas as pd
excel_file_path='predict.csv'
df=pd.read_csv(excel_file_path)

for i in range(10140):
    value_ij=df.iloc[i-1,1]
    if value_ij>0.5:
        df.iloc[i-1,1]=1
    else:
        df.iloc[i-1,1]=0
```

```
df.to_csv('result_cnn_xmnlp_models.csv',index=False)
```

14. Bert_model.py

```
import pandas as pd
import numpy as np
import jieba
import os
import csv
import jieba.analyse
from sklearn.utils import shuffle
from nltk.stem import PorterStemmer
import torch
from transformers.file_utils import is_tf_available, is_torch_available, is_torch_tpu_available
from transformers import BertTokenizerFast, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
import random
import wandb
wandb.login(key='737f2bbdefed89aeeea9e69073995c88b7da8336')


train_df = pd.read_csv("train.news.csv")
train_df = train_df.dropna()
train_df = shuffle(train_df)

stem = PorterStemmer()
punc=r'~`!#$%^&*()_+-=|\';":/.,?><~·！@#￥%……&*（）——+-=“：’；、。，？》《{}'
def stop_words_list(filepath):
    stop_words = [line.strip() for line in open(filepath,'r',encoding='utf-8').readlines()]
    return stop_words
stopwords = ['是', '的', '了', '在', '和', '有', '被', '这', '那', '之', '更', '与', '对于', '并', '我', '他', '她',
             '它', '我们', '他们', '她们', '它们']
def cleaning(text):
    cutwords = list(jieba.lcut_for_search(text))
    final_cutwords = ''
    for word in cutwords:
        if word not in stopwords and punc:
            final_cutwords += word + ' '
    return final_cutwords


train_df["Report Content"] = train_df["Report Content"].apply(lambda x:x.split("##"))
columns = ['Title', 'Report Content', 'label', 'Ofiicial Account Name']
t = pd.DataFrame(train_df.astype(str))
```

```
train_df['Title'] = t['Title'].apply(cleaning)
train_df['Report Content'] = t['Report Content'].apply(cleaning)
train_df['Ofiicial Account Name'] = t['Ofiicial Account Name']
train_df = train_df[columns]
data = train_df
print(data.head())
def set_seed(seed: int):
    """
    Helper function for reproducible behavior to set the seed in ``random``, ``numpy``, ``torch``
and/or ``tf`` (if
    installed).

    Args:
        seed (:obj:`int`): The seed to set.
    """
    random.seed(seed)
    np.random.seed(seed)
    if is_torch_available():
        torch.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        # ^^ safe to call this function even if cuda is not available
    if is_tf_available():
        import tensorflow as tf

        tf.random.set_seed(seed)

set_seed(123)

model_name = "bert-base-chinese"
max_length= 512

tokenizer = BertTokenizerFast.from_pretrained(model_name, do_lower_case=True)

data = data[data['Title'].notna()]
data = data[data['Ofiicial Account Name'].notna()]
data = data[data['Report Content'].notna()]


def prepare_data(df, test_size=0.2, include_title=True, include_author=True):
    texts = []
    labels = []

    for i in range(len(df)):
        text = df['Report Content'].iloc[i]
```

```
            label = df['label'].iloc[i]

            if include_title:
                text = df['Title'].iloc[i] + " - " + text
            if include_author:
                text = df['Ofiicial Account Name'].iloc[i] + " - " + text

            if text and label in [0, 1]:
                texts.append(text)
                labels.append(label)

    return train_test_split(texts, labels, test_size=test_size)


train_texts, valid_texts, train_labels, valid_labels = prepare_data(data)
train_encodings = tokenizer(train_texts, truncation=True, padding=True,
max_length=max_length)
valid_encodings = tokenizer(valid_texts, truncation=True, padding=True,
max_length=max_length)


class NewsGroupsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx]) for k, v in self.encodings.items()}
        item['labels'] = torch.tensor([self.labels[idx]], dtype=torch.long)    # 强制转换为
torch.long 类型
        return item

    def __len__(self):
        return len(self.labels)


# convert tokenize data into torch dataset
train_dataset = NewsGroupsDataset(train_encodings, train_labels)
valid_dataset = NewsGroupsDataset(valid_encodings, valid_labels)

model = BertForSequenceClassification.from_pretrained(model_name, num_labels=2)

from sklearn.metrics import accuracy_score
```

```
def computer_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    acc = accuracy_score(labels, preds)

    return {'accuracy': acc, }

training_args = TrainingArguments(
    output_dir='/results',              # output directory
    num_train_epochs=2,                 # total number of training epochs
    per_device_train_batch_size=10,  # batch size per device during training
    per_device_eval_batch_size=20,   # batch size for evaluation
    warmup_steps=100,                   # number of warmup steps for learning rate
scheduler
    logging_dir='/results',
    # directory for storing logs
    load_best_model_at_end=True,      # load the best model when finished training (default
metric is loss)
    # but you can specify `metric_for_best_model` argument to change to accuracy or other
metric
    logging_steps=200,                  # log & save weights each logging_steps
    save_steps=200,
    evaluation_strategy="steps",       # evaluate each `logging_steps`
)

trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=computer_metrics,
)

trainer.train()
model.save_pretrained('./cache/model_bert1')
tokenizer.save_pretrained('./cache/tokenizer1')
def get_prediction(text, convert_to_label=False):
    # prepare our text into tokenized sequence
    inputs = tokenizer(text, padding=True, truncation=True, max_length=max_length,
return_tensors="pt").to("cuda")
    # perform inference to our model
    outputs = model(**inputs)
    # get output probabilities by doing softmax
    probs = outputs[0].softmax(1)
```

```
    # executing argmax function to get the candidate label
    d = {
        0: "reliable",
        1: "fake"
    }
    if convert_to_label:
        return d[int(probs.argmax())]
    else:
        return int(probs.argmax())

test_df = pd.read_csv("test.feature.csv")
# make a copy of the testing set
new_df = test_df.copy()
# add a new column that contains the author, title and article content
new_df["Report Content"] = new_df["Report Content"].apply(lambda x:x.split("##"))
t = pd.DataFrame(train_df.astype(str))
new_df['Title'] = t['Title'].apply(cleaning)
new_df['Report Content'] = t['Report Content'].apply(cleaning)
new_df['Ofiicial Account Name'] = t['Ofiicial Account Name']
new_df = new_df[columns]
new_df["new_text"] = new_df["Ofiicial Account Name"].astype(str) + " " +
new_df["Title"].astype(str) + " " + new_df["Report Content"].astype(str)
new_df["label"] = new_df["new_text"].apply(get_prediction)
# make the submission file
final_df = new_df[["id", "label"]]
final_df.to_csv("result_bert.csv", index=False)
```

15. using_Matplotlib.py

```
#此程序用于实验报告中的一些图表的描绘
#1.七种机器学习方法的 ACC 绘图
import numpy as np
import matplotlib.pyplot as plt

A=['bayes','k-NN','DTree','RForest','GBoosting','SVC','MLP']
B=[0.951,0.837,0.96,0.961,0.84,0.955,0.959]
plt.bar(A,B,color='skyblue')
plt.title("seven machine learning's accuracy")
plt.xlabel(A)
plt.ylabel(B)
plt.ylim(0.6,1)
plt.show()

#2.所有的模型的 AUC 比较
import numpy as np
```

```python
import matplotlib.pyplot as plt

A=['bayes','bayes+jieba','CNN+jieba','CNN+gram','CNN+xmnlp','bert']
B=[0.6137,0.6735,0.7323,0.7627,0.8237,0.8676]
plt.bar(A,B,color='yellow')

plt.title("all models learning's AUC")
plt.xlabel(A)
plt.ylabel(B)
plt.ylim(0.5,1)
plt.show()
```