武漢大學

《推特下新冠疫苗副作用提取》

——《自然语言处理》大作业

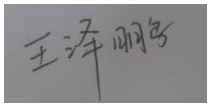
姓名：王泽鹏

班级：2021 级硕士 2 班

学号：2021202210073

专业：网络空间安全

指导教师：姬东鸿

签名： 

二〇二二年 二月

1 任务要求：

提取给定文本内疫苗种类、对疫苗的态度以及疫苗的副作用。

2 实验过程：

2.1 数据清洗及预处理

首先对数据集进行预处理及数据清洗工作。

对数据集初步读取发现，存在 attitude 和 side_effects 为空的无效数据：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2926 entries, 0 to 2925
Data columns (total 4 columns):
text            2926 non-null object
vaccine         2926 non-null object
attitude        2735 non-null object
side_effects    2759 non-null object
dtypes: object(4)
memory usage: 91.5+ KB
None
```

这可能是因为标注人员标注数据时操作不规范导致的，如查看数据集发现存在下面的情况：

```
2924  a. My booster wiped me out for 3 days. I never felt sick but I had zero energy.   Pfizer  Negative    Tiredness
2925  a. My booster wiped me out for 3 days. I never felt sick but I had zero energy.   Moderna
```

这里同一条信息被标注了两次，且第二次的 attitude 和 side_effectis 信息均为空，这种信息就需要提前进行过滤清洗处理。

利用 dropna 对含空信息的行进行去除，清洗后的数据信息如下：

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2655 entries, 0 to 2925
Data columns (total 4 columns):
text            2655 non-null object
vaccine         2655 non-null object
attitude        2655 non-null object
side_effects    2655 non-null object
dtypes: object(4)
memory usage: 103.7+ KB
None
```

再次观察数据集发现，仍存在同一条文本信息被标注多次的特殊情况：

```
30  I was fine. First AZ shot was way way worse. Just a sore arm with Moderna booster   Moderna Neutral Muscle Pain
31  I was fine. First AZ shot was way way worse. Just a sore arm with Moderna booster   AstraZeneca Negative    None
```

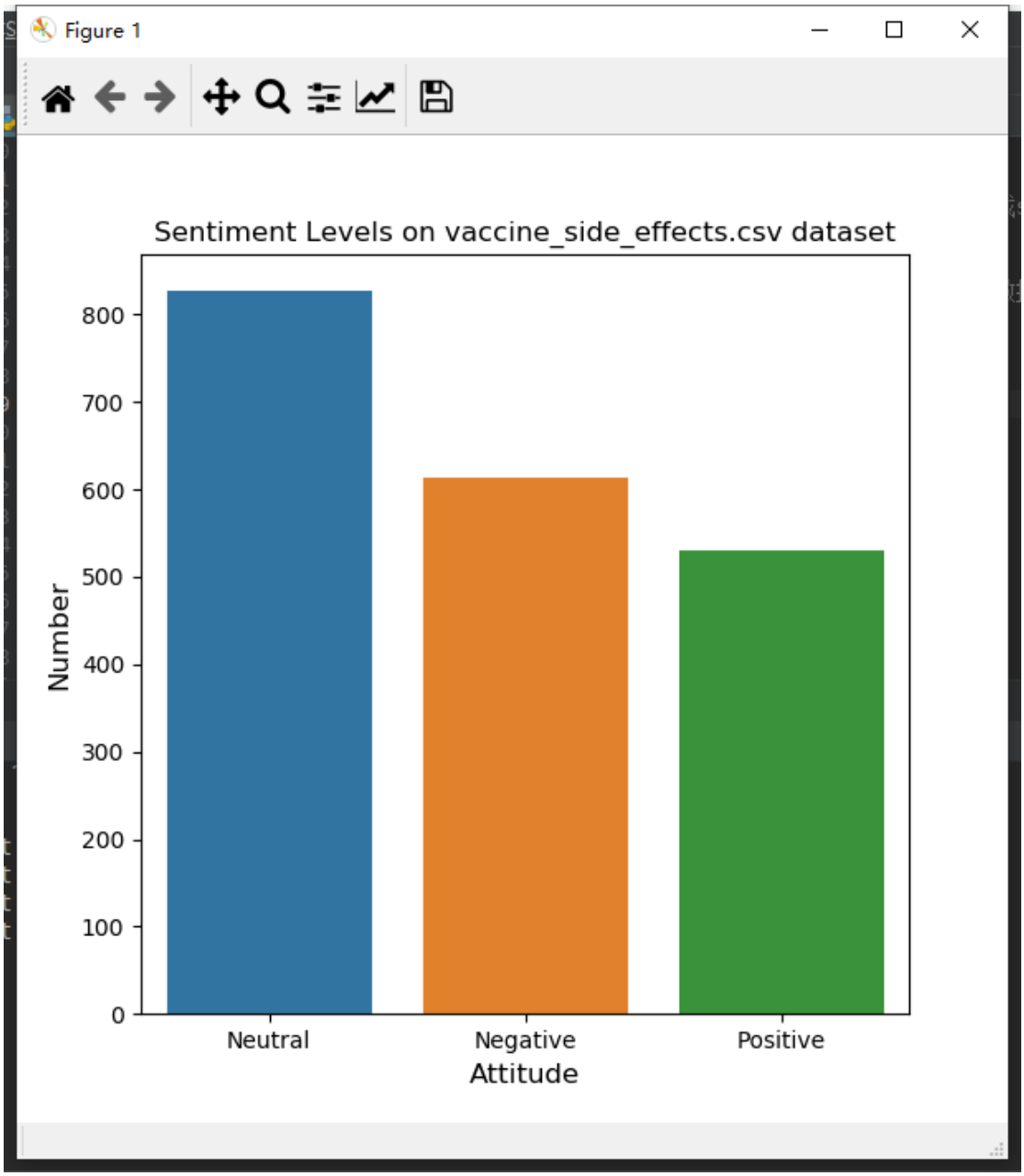
这是由于标注规则要求的对一条文本信息出现多种疫苗则进行多次标注，导致数据集中同一条文本信息对应多个标签，这不利于我们的训练和测试，故将此类数据也一并清洗去除。最终得到清洗后的数据信息如下：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1973 entries, 0 to 1972
Data columns (total 4 columns):
text            1973 non-null object
vaccine         1973 non-null object
attitude        1973 non-null object
side_effects    1973 non-null object
dtypes: object(4)
memory usage: 61.7+ KB
None
```

2.2 数据初步可视化

对清洗后的数据集的 attitude 信息进行初步统计：

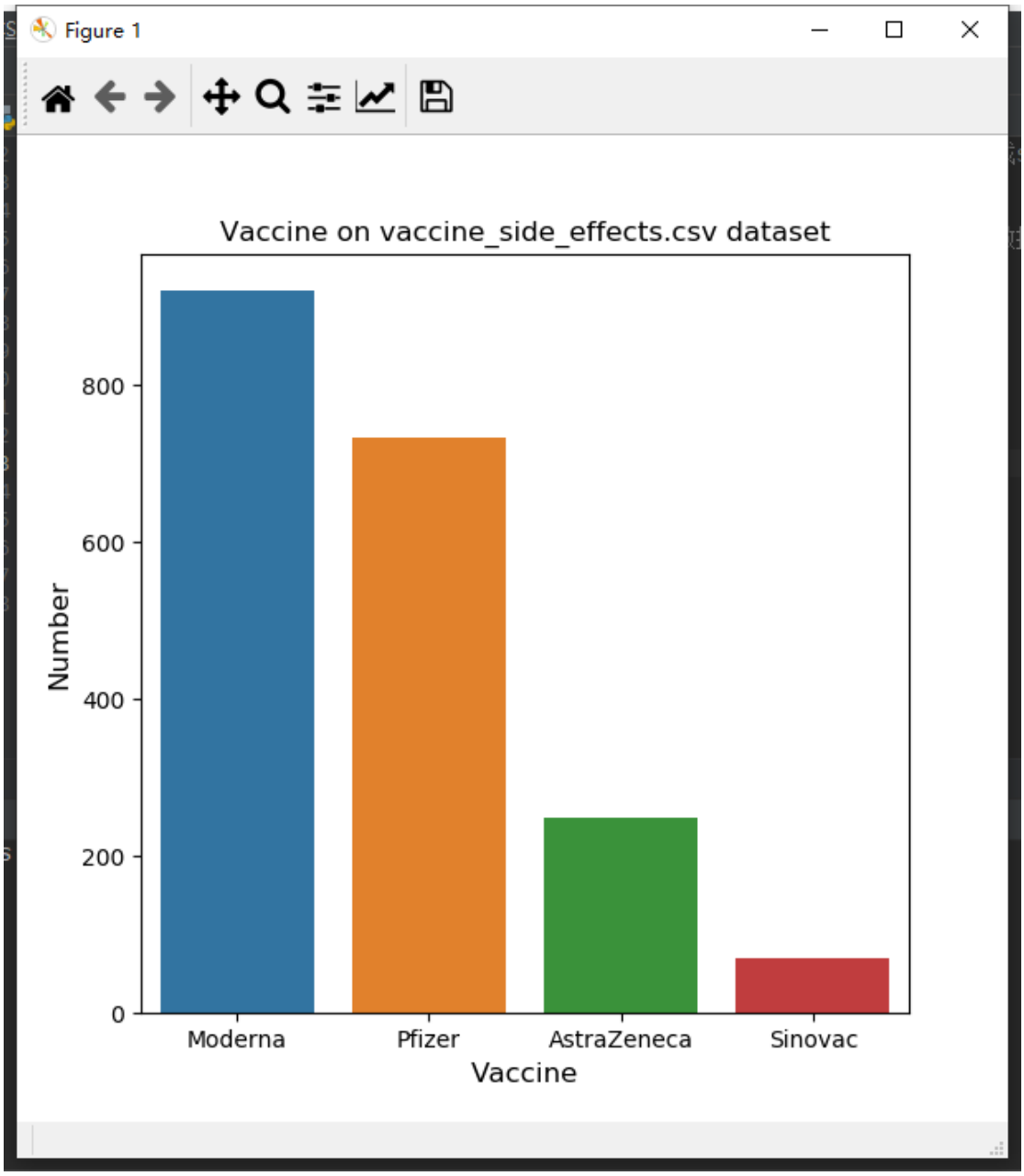
```
Neutral     828
Negative    614
Positive    531
Name: attitude, dtype: int64
```



可以初步看出，该数据集下的国外网民对于新冠疫苗的态度持中立态度的占大多数，而持积极和消极态度的比例相近。

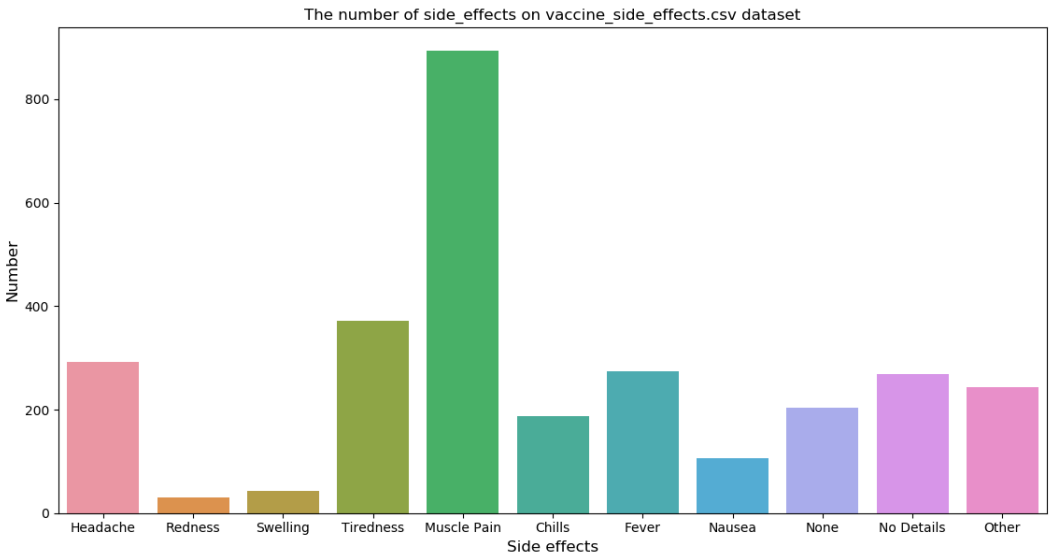
对 vaccine 信息进行初步统计：

```
Moderna        921
Pfizer         733
AstraZeneca    249
Sinovac         70
Name: vaccine, dtype: int64
```



可以看到，网民对 Moderna 和 Pfizer 的评论占大多数，可能是这类疫苗的市场占有率（流行度）在该地区更大。

对 side_effects 信息进行初步统计：



可以看出，Muscle Pain 是最常见的一种副作用，其他副作用也或多或少存在。

2.3 构建模型

2.3.1 对 attitude 和 vaccine 的预测模型

对 attitude 和 vaccine 预测的方法思路类似，下面以 attitude 的预测模型建模过程为例详细说明，vaccine 预测过程同理就不再详述，在下一部分将包含每个任务的实验结果对比。

首先对 text 文本部分作进一步的预处理，包括大小写转换、标点符号去除、特殊符号去除、停用词去除等部分：

```
'''
    ----------------------------
    对text文本部分作进一步的预处理
    ----------------------------
'''
txt_df = df4['text']
# remove hashtags
txt_df2 = txt_df.apply(lambda x: re.sub(r"#\S+", "", str(x)))
# convert to lowercase
txt_df3 = txt_df2.apply(lambda x: x.lower())
# remove punctuations
txt_df4 = txt_df3.apply(lambda x: x.translate(str.maketrans('', '', string.punctuation)))
# remove special characters
txt_df5 = txt_df4.apply(lambda x: re.sub('[^a-zA-Z0-9]', ' ', str(x)))
nltk.download("stopwords")
# removing stopwords
stop_words = set(stopwords.words('english'))
txt_df6 = txt_df5.apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))
print(txt_df6.head(10))
```

由于我们是一个多分类预测任务，故对标签进行数值化处理方便训练预测：

```
'''
    ------------------------------
        对label进行数值化处理
    ------------------------------
'''
df4['attitude'].replace(['Negative','Neutral','Positive'],[0,1,2],inplace=True)
# print(df4.head(10))
df4['vaccine'].replace(['Pfizer','Moderna','AstraZeneca','Sinovac'],[0,1,2,3],inplace=True)

'''
```

下面首先对文本特征进行提取，即文本特征向量化过程。文本特征提取有多种模型，如词集模型、BOW(Bag of Words)词袋模型、TF-IDF 模型等，本次实验我们采用了三种特征向量化模型，下面分别进行介绍。

我们采用的第一个文本特征向量化模型是 BOW 词袋模型。BOW 模型将文本中所有单词构成一个词典，每个单词对应唯一一个索引，单词的顺序与在句子中的顺序没有关系，这样每个文本都可以构成一个 N 维向量，作为特征向量送入分类器中进行训练。

```
# 1 bag-of-words feature matrix
print("----------CountVectorizer----------")
from sklearn.feature_extraction.text import CountVectorizer
bow_vectorizer = CountVectorizer(stop_words='english')
X = bow_vectorizer.fit_transform(txt_df6)
y = df4["attitude"]
print(X.shape, y.shape)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=21)
# print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# print(X_train[0], y_train[0])
```

第二个特征向量化模型是 TF-IDF 模型。TF-IDF 是一种统计方法，用以评估一个字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF 值越大说明这个词越重要，也可以说这个词是关键词。实验中我们直接使用 TF-IDF 完成向量化生成特征向量，为了对比不同特征提取方法的性能差异，实际上我们可以将 BOW 和 TF-IDF 模型结合起来以实现更好的性能。

```
# 2 TF-IDF feature matrix
from sklearn.feature_extraction.text import TfidfVectorizer
print("----------TfidfVectorizer----------")
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
X = tfidf_vectorizer.fit_transform(txt_df6)
y = df4["attitude"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=21)
```

第三个特征向量化模型是基于 word2vec 的向量化模型。我们采用的是 skip-gram 的语言模型，即用一个词语作为输入，来预测它周围的上下文的方式。同时采用负采样，利用训练文本构建一个词嵌入矩阵，其中作了一些细节处理，如给未知词单独分配一个 index，这样文本中的每个词都对应上了一个词向量，后续可以传入不同的分类器中进行训练，具体处理细节可参考代码及代码注释。

```
feature = txt_df6
y = df4["attitude"]
y2 = df4["vaccine"]
# 训练模型，词向量的长度设置为500，采用skip-gram模型，采用负采样，窗口选择6，最小词频是7，模型保存为pkl格式
w2v_model=Word2Vec(sentences=feature, vector_size=500, sg=1,hs=0,window=6, min_count=7)
w2v_model.wv.save_word2vec_format("./word2Vec" + ".pkl", binary=True)

NUM_CLASS = 3       # 态度数量
NUM_CLASS2 = 4      # 疫苗种类数量
NUM_CLASS3 = len(mlb.classes_)  # 副作用种类数量
INPUT_SIZE = 64     # 输入维度
# # 序列对齐文本数据
# Tokenizer是一个用于向量化文本，或将文本转换为序列
tokenizer = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n', lower=True, split=" ")
tokenizer.fit_on_texts(feature)
vocab = tokenizer.word_index
# print("vocab:", len(vocab)) # 5725
x_ids = tokenizer.texts_to_sequences(feature)
pad_s = pad_sequences(x_ids, maxlen=INPUT_SIZE)
target_u = to_categorical(y, NUM_CLASS)
X_train, X_test, y_train, y_test = train_test_split(pad_s, target_u, random_state=22, test_size=0.2)
# target_u2 = to_categorical(y2, NUM_CLASS2)
# X_train2, X_test2, y_train2, y_test2 = train_test_split(pad_s, target_u2, random_state=22, test_size=0.2)
# target_u3 = labels1
# X_train3, X_test3, y_train3, y_test3 = train_test_split(pad_s, target_u3, random_state=22, test_size=0.2)
```

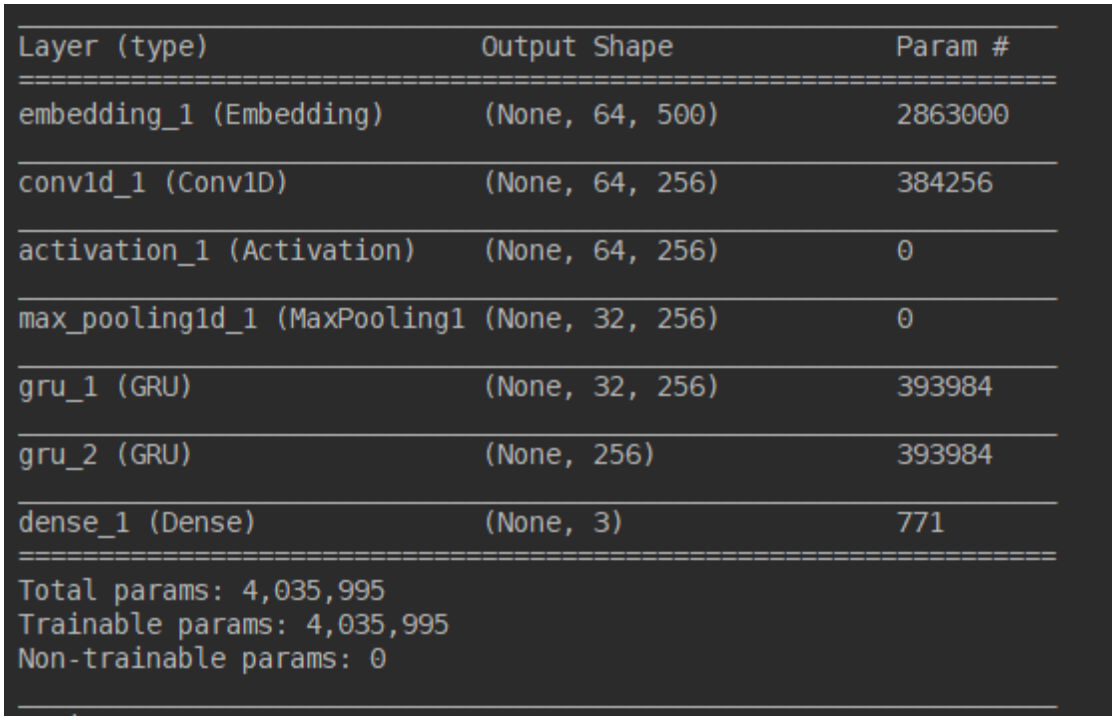
```
embedding_matrix = np.zeros((len(vocab)+1, 500))
for word, i in vocab.items():
    try:
        embedding_vector=w2v_model.wv[str(word)]
        embedding_matrix[i]=embedding_vector
    except:
        print("Word: [",word,"] not in wvmodel! Use random embedding instead.")

main_input = Input(shape=(INPUT_SIZE,), dtype='float64')
```

下一步便是构建文本分类器。我们采用了多种分类器进行对比实验，包括朴素贝叶斯、SVM、决策树、word2vec+RNN、word2vec+CNN+GRU、word2vec+Bi-GRU 等多种实现。其中，训练集与测试集的比例为 4:1，后三种基于神经网络的模型使用的优化器为 adam 优化器、损失函数采用 categorical_crossentropy，batchsize 取 32，训练轮数为 10 轮，相关超参数保持一致，具体网络结构如下：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 64, 500)           2863000
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               775168
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 771
=================================================================
Total params: 3,638,939
Trainable params: 3,638,939
Non-trainable params: 0
_________________________________________________________________
```

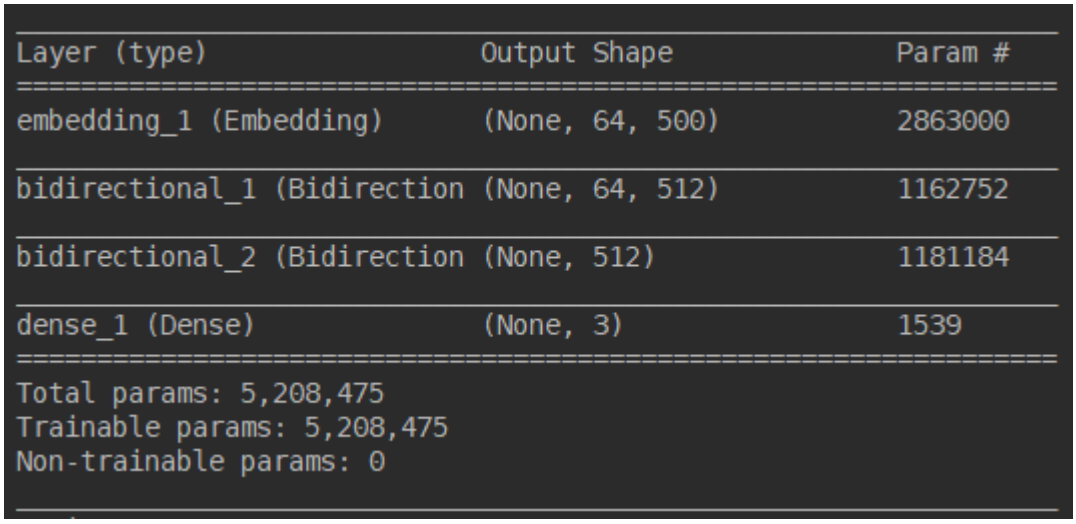
word2vec+RNN



Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 64, 500)	2863000
conv1d_1 (Conv1D)	(None, 64, 256)	384256
activation_1 (Activation)	(None, 64, 256)	0
max_pooling1d_1 (MaxPooling1	(None, 32, 256)	0
gru_1 (GRU)	(None, 32, 256)	393984
gru_2 (GRU)	(None, 256)	393984
dense_1 (Dense)	(None, 3)	771

Total params: 4,035,995
Trainable params: 4,035,995
Non-trainable params: 0

word2vec+CNN+GRU



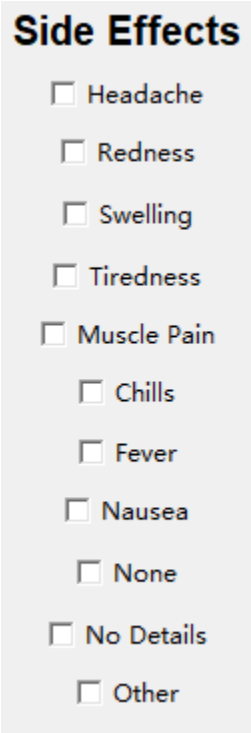
Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 64, 500)	2863000
bidirectional_1 (Bidirection	(None, 64, 512)	1162752
bidirectional_2 (Bidirection	(None, 512)	1181184
dense_1 (Dense)	(None, 3)	1539

Total params: 5,208,475
Trainable params: 5,208,475
Non-trainable params: 0

word2vec+Bi-GRU

2.3.2 对 side_effects 的预测模型

对于副作用的提取，这个任务难度比较大，因为这不是一个简单的单标签分类任务，而是多标签分类任务，且有些症状通过人工理解文本也比较难以确定具体副作用，如下图为对于副作用的所有分类名称。



对于多标签问题，业界还没有很成熟的解决方法，主要是因为标签之间可能会存在复杂的依赖关系，这种依赖关系现阶段还没有成熟的模型来解决。多标签分类与多分类问题的区别是前者每个类是独立的而不是互斥的，而后者每个类是互斥的。

我们的思路是把多标签分类转化为多个二分类问题，利用 sklearn 模块中的 MultiLabelBinarizer 进行多标签编码，再使用 sigmoid 激活函数来处理多标签分类问题，送入不同结构的神经网络中进行训练对比。

首先我们利用 sklearn 的 MultiLabelBinarizer 进行多标签编码，如果对应的副作用存在则对应位置的元素值为 1，否则为 0，这样每一条数据都对应一个 11 维的 0-1 向量。

```
from sklearn.preprocessing import MultiLabelBinarizer
# 获取训练集合、测试集的事件类型
labels = []
side_effects_list = []
for line in df4['side_effects']:
    genres = line.split(",")
    labels.append(genres)
    side_effects_list.extend(genres)
# 利用sklearn中的MultiLabelBinarizer进行多标签编码
mlb = MultiLabelBinarizer()
mlb.fit(labels)
# print("一共有%d种事件类型。" % len(mlb.classes_))
```

```
# 进行多标签编码
labels1 = []
for line in df4['side_effects']:
    genres = line.split(",")
    labels1.append(mlb.transform([genres])[0])
labels1 = np.array(labels1)
```

然后将其作为训练集和测试集标签，使用前面提到的多种神经网络进行训练，利用

sigmoid 激活函数和 binary_crossentropy 损失函数进行训练即可。

```
# 3.3 word2vec+Bi-GRU
model.add(Embedding(len(vocab)+1, 500, input_length=INPUT_SIZE, weights=[embedding_matrix], trainable=True))
model.add(Bidirectional(GRU(256, dropout=0.2, recurrent_dropout=0.1, return_sequences=True)))
model.add(Bidirectional(GRU(256, dropout=0.2, recurrent_dropout=0.1)))
# model.add(Dense(NUM_CLASS, activation='softmax'))
# model.add(Dense(NUM_CLASS2, activation='softmax'))
model.add(Dense(NUM_CLASS3, activation='sigmoid'))

model.summary()
# model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])   # 副作用模型
```

2.4 进行训练

2.4.1 传统机器学习模型

对 attitude 预测的模型：

```
----------CountVectorizer----------
(1973, 5551) (1973,)
------MultinomialNB------
Training accuracy:  0.8548795944233206
Test accuracy:  0.44556962025316454
------LinearSVC------
Training accuracy:  0.9860583016476553
Test accuracy:  0.4253164556962025
------DecisionTree------
Training accuracy:  1.0
Test accuracy:  0.41265822784810124
----------TfidfVectorizer----------
------MultinomialNB------
Training accuracy:  0.6939163498098859
Test accuracy:  0.43544303797468353
------LinearSVC------
Training accuracy:  0.9759188846641318
Test accuracy:  0.4506329113924051
------DecisionTree------
Training accuracy:  1.0
Test accuracy:  0.41265822784810124

Process finished with exit code 0
```

CountVectorizer

TfidfVectorizer

2.4.2 基于神经网络的模型

对 attitude 预测的模型：

```
Train on 1578 samples, validate on 395 samples
Epoch 1/10
1578/1578 [==============================] - 41s 26ms/step - loss: 1.0725 - accuracy: 0.4221 - val_loss: 1.0320 - val_accuracy: 0.4557
Epoch 2/10
1578/1578 [==============================] - 33s 21ms/step - loss: 0.8015 - accuracy: 0.6616 - val_loss: 1.0745 - val_accuracy: 0.4456
Epoch 3/10
1578/1578 [==============================] - 34s 21ms/step - loss: 0.4344 - accuracy: 0.8365 - val_loss: 1.4558 - val_accuracy: 0.3899
Epoch 4/10
1578/1578 [==============================] - 31s 20ms/step - loss: 0.2390 - accuracy: 0.9018 - val_loss: 1.5648 - val_accuracy: 0.4608
Epoch 5/10
1578/1578 [==============================] - 30s 19ms/step - loss: 0.1364 - accuracy: 0.9544 - val_loss: 1.9378 - val_accuracy: 0.4734
Epoch 6/10
1578/1578 [==============================] - 31s 20ms/step - loss: 0.0785 - accuracy: 0.9753 - val_loss: 2.1763 - val_accuracy: 0.4582
Epoch 7/10
1578/1578 [==============================] - 34s 21ms/step - loss: 0.0613 - accuracy: 0.9823 - val_loss: 2.1669 - val_accuracy: 0.4481
Epoch 8/10
1578/1578 [==============================] - 35s 22ms/step - loss: 0.0438 - accuracy: 0.9848 - val_loss: 2.6921 - val_accuracy: 0.4177
Epoch 9/10
1578/1578 [==============================] - 32s 20ms/step - loss: 0.0509 - accuracy: 0.9854 - val_loss: 2.8893 - val_accuracy: 0.4354
Epoch 10/10
1578/1578 [==============================] - 31s 19ms/step - loss: 0.0732 - accuracy: 0.9772 - val_loss: 2.3760 - val_accuracy: 0.4506
395/395 [==============================] - 2s 4ms/step

Process finished with exit code 0
```

word2vec+RNN

```
Train on 1578 samples, validate on 395 samples
Epoch 1/10
1578/1578 [==============================] - 31s 20ms/step - loss: 1.0779 - accuracy: 0.4176 - val_loss: 1.0343 - val_accuracy: 0.4835
Epoch 2/10
1578/1578 [==============================] - 27s 17ms/step - loss: 0.8599 - accuracy: 0.6065 - val_loss: 1.0576 - val_accuracy: 0.4734
Epoch 3/10
1578/1578 [==============================] - 27s 17ms/step - loss: 0.4186 - accuracy: 0.8302 - val_loss: 1.3859 - val_accuracy: 0.4506
Epoch 4/10
1578/1578 [==============================] - 26s 17ms/step - loss: 0.2063 - accuracy: 0.9221 - val_loss: 1.7459 - val_accuracy: 0.4456
Epoch 5/10
1578/1578 [==============================] - 26s 17ms/step - loss: 0.0996 - accuracy: 0.9626 - val_loss: 2.7631 - val_accuracy: 0.4430
Epoch 6/10
1578/1578 [==============================] - 26s 17ms/step - loss: 0.0612 - accuracy: 0.9778 - val_loss: 3.1229 - val_accuracy: 0.4532
Epoch 7/10
1578/1578 [==============================] - 26s 17ms/step - loss: 0.0608 - accuracy: 0.9797 - val_loss: 3.0504 - val_accuracy: 0.4532
Epoch 8/10
1578/1578 [==============================] - 27s 17ms/step - loss: 0.0640 - accuracy: 0.9778 - val_loss: 3.2622 - val_accuracy: 0.4532
Epoch 9/10
1578/1578 [==============================] - 26s 17ms/step - loss: 0.0765 - accuracy: 0.9721 - val_loss: 2.7935 - val_accuracy: 0.4532
Epoch 10/10
1578/1578 [==============================] - 26s 17ms/step - loss: 0.0513 - accuracy: 0.9816 - val_loss: 3.1418 - val_accuracy: 0.4203
395/395 [==============================] - 1s 3ms/step

Process finished with exit code 0
```

word2vec+CNN+RNN

```
Train on 1578 samples, validate on 395 samples
Epoch 1/10
1578/1578 [==============================] - 160s 102ms/step - loss: 1.0831 - accuracy: 0.3967 - val_loss: 1.0640 - val_accuracy: 0.4177
Epoch 2/10
1578/1578 [==============================] - 172s 109ms/step - loss: 0.8223 - accuracy: 0.6293 - val_loss: 1.1156 - val_accuracy: 0.4456
Epoch 3/10
1578/1578 [==============================] - 179s 114ms/step - loss: 0.4284 - accuracy: 0.8308 - val_loss: 1.2995 - val_accuracy: 0.4430
Epoch 4/10
1578/1578 [==============================] - 189s 120ms/step - loss: 0.2172 - accuracy: 0.9233 - val_loss: 1.4881 - val_accuracy: 0.4506
Epoch 5/10
1578/1578 [==============================] - 180s 114ms/step - loss: 0.1255 - accuracy: 0.9569 - val_loss: 1.7676 - val_accuracy: 0.4506
Epoch 6/10
1578/1578 [==============================] - 173s 110ms/step - loss: 0.0746 - accuracy: 0.9728 - val_loss: 2.3045 - val_accuracy: 0.4481
Epoch 7/10
1578/1578 [==============================] - 181s 115ms/step - loss: 0.0912 - accuracy: 0.9651 - val_loss: 2.0876 - val_accuracy: 0.4734
Epoch 8/10
1578/1578 [==============================] - 196s 124ms/step - loss: 0.0658 - accuracy: 0.9797 - val_loss: 2.3396 - val_accuracy: 0.4506
Epoch 9/10
1578/1578 [==============================] - 172s 109ms/step - loss: 0.0215 - accuracy: 0.9943 - val_loss: 2.8446 - val_accuracy: 0.4506
Epoch 10/10
1578/1578 [==============================] - 173s 109ms/step - loss: 0.0345 - accuracy: 0.9886 - val_loss: 2.7988 - val_accuracy: 0.4430
395/395 [==============================] - 6s 16ms/step

Process finished with exit code 0
```

word2vec+Bi-GRU

对 vaccine 预测的模型：

```
Train on 1578 samples, validate on 395 samples
Epoch 1/10
1578/1578 [==============================] - 56s 35ms/step - loss: 1.1138 - accuracy: 0.4677 - val_loss: 0.9567 - val_accuracy: 0.4759
Epoch 2/10
1578/1578 [==============================] - 47s 30ms/step - loss: 0.5659 - accuracy: 0.7852 - val_loss: 0.5308 - val_accuracy: 0.8203
Epoch 3/10
1578/1578 [==============================] - 37s 24ms/step - loss: 0.1883 - accuracy: 0.9430 - val_loss: 0.3746 - val_accuracy: 0.8684
Epoch 4/10
1578/1578 [==============================] - 29s 18ms/step - loss: 0.0816 - accuracy: 0.9785 - val_loss: 0.4580 - val_accuracy: 0.8785
Epoch 5/10
1578/1578 [==============================] - 30s 19ms/step - loss: 0.0625 - accuracy: 0.9816 - val_loss: 0.4908 - val_accuracy: 0.8532
Epoch 6/10
1578/1578 [==============================] - 32s 20ms/step - loss: 0.0693 - accuracy: 0.9772 - val_loss: 0.4319 - val_accuracy: 0.8734
Epoch 7/10
1578/1578 [==============================] - 32s 21ms/step - loss: 0.0195 - accuracy: 0.9943 - val_loss: 0.4962 - val_accuracy: 0.8633
Epoch 8/10
1578/1578 [==============================] - 33s 21ms/step - loss: 0.0094 - accuracy: 0.9981 - val_loss: 0.4913 - val_accuracy: 0.8608
Epoch 9/10
1578/1578 [==============================] - 34s 21ms/step - loss: 0.0044 - accuracy: 0.9994 - val_loss: 0.5501 - val_accuracy: 0.8557
Epoch 10/10
1578/1578 [==============================] - 32s 20ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.5387 - val_accuracy: 0.8734
395/395 [==============================] - 1s 3ms/step

Process finished with exit code 0
```

vaccine：word2vec+RNN

```
Train on 1578 samples, validate on 395 samples
Epoch 1/10
1578/1578 [==============================] - 30s 19ms/step - loss: 1.0086 - accuracy: 0.5507 - val_loss: 0.5014 - val_accuracy: 0.7848
Epoch 2/10
1578/1578 [==============================] - 27s 17ms/step - loss: 0.3411 - accuracy: 0.8790 - val_loss: 0.4783 - val_accuracy: 0.8203
Epoch 3/10
1578/1578 [==============================] - 29s 18ms/step - loss: 0.1480 - accuracy: 0.9493 - val_loss: 0.5386 - val_accuracy: 0.8203
Epoch 4/10
1578/1578 [==============================] - 29s 18ms/step - loss: 0.0600 - accuracy: 0.9804 - val_loss: 0.6338 - val_accuracy: 0.7747
Epoch 5/10
1578/1578 [==============================] - 29s 18ms/step - loss: 0.0269 - accuracy: 0.9911 - val_loss: 0.7415 - val_accuracy: 0.7772
Epoch 6/10
1578/1578 [==============================] - 30s 19ms/step - loss: 0.0185 - accuracy: 0.9949 - val_loss: 0.6993 - val_accuracy: 0.8000
Epoch 7/10
1578/1578 [==============================] - 29s 18ms/step - loss: 0.0130 - accuracy: 0.9956 - val_loss: 0.6339 - val_accuracy: 0.8253
Epoch 8/10
1578/1578 [==============================] - 27s 17ms/step - loss: 0.0114 - accuracy: 0.9975 - val_loss: 0.6318 - val_accuracy: 0.8380
Epoch 9/10
1578/1578 [==============================] - 27s 17ms/step - loss: 0.0106 - accuracy: 0.9968 - val_loss: 1.1072 - val_accuracy: 0.7367
Epoch 10/10
1578/1578 [==============================] - 28s 18ms/step - loss: 0.0048 - accuracy: 0.9987 - val_loss: 0.7836 - val_accuracy: 0.8177
395/395 [==============================] - 1s 4ms/step

Process finished with exit code 0
```

vaccine：word2vec+CNN+GRU

```
Train on 1578 samples, validate on 395 samples
Epoch 1/10
1578/1578 [==============================] - 166s 105ms/step - loss: 0.9697 - accuracy: 0.5703 - val_loss: 0.8272 - val_accuracy: 0.7190
Epoch 2/10
1578/1578 [==============================] - 180s 114ms/step - loss: 0.4188 - accuracy: 0.8612 - val_loss: 0.7832 - val_accuracy: 0.6962
Epoch 3/10
1578/1578 [==============================] - 162s 102ms/step - loss: 0.2199 - accuracy: 0.9265 - val_loss: 0.8208 - val_accuracy: 0.6886
Epoch 4/10
1578/1578 [==============================] - 171s 109ms/step - loss: 0.1472 - accuracy: 0.9575 - val_loss: 0.9992 - val_accuracy: 0.6278
Epoch 5/10
1578/1578 [==============================] - 175s 111ms/step - loss: 0.0522 - accuracy: 0.9867 - val_loss: 1.0418 - val_accuracy: 0.6354
Epoch 6/10
1578/1578 [==============================] - 180s 114ms/step - loss: 0.0151 - accuracy: 0.9956 - val_loss: 1.2072 - val_accuracy: 0.6076
Epoch 7/10
1578/1578 [==============================] - 173s 110ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 1.1544 - val_accuracy: 0.6633
Epoch 8/10
1578/1578 [==============================] - 178s 113ms/step - loss: 8.9062e-04 - accuracy: 1.0000 - val_loss: 1.2073 - val_accuracy: 0.6684
Epoch 9/10
1578/1578 [==============================] - 174s 110ms/step - loss: 5.1749e-04 - accuracy: 1.0000 - val_loss: 1.2212 - val_accuracy: 0.6759
Epoch 10/10
1578/1578 [==============================] - 175s 111ms/step - loss: 4.0677e-04 - accuracy: 1.0000 - val_loss: 1.2400 - val_accuracy: 0.6810
395/395 [==============================] - 7s 17ms/step

Process finished with exit code 0
```

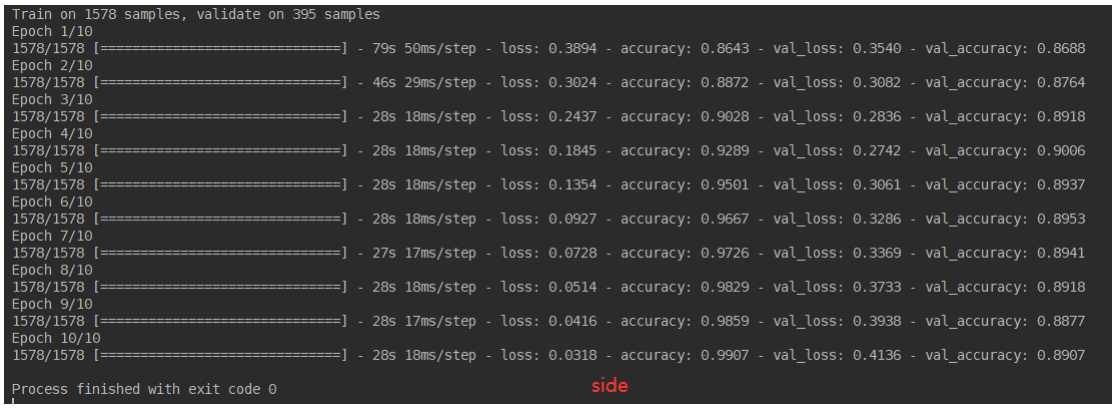
word2vec+Bi-GRU

对 side_effects 预测的模型：

```
Train on 1578 samples, validate on 395 samples
Epoch 1/10
1578/1578 [==============================] - 62s 39ms/step - loss: 0.3897 - accuracy: 0.8640 - val_loss: 0.3594 - val_accuracy: 0.8626
Epoch 2/10
1578/1578 [==============================] - 50s 32ms/step - loss: 0.3389 - accuracy: 0.8716 - val_loss: 0.3537 - val_accuracy: 0.8686
Epoch 3/10
1578/1578 [==============================] - 51s 32ms/step - loss: 0.3026 - accuracy: 0.8895 - val_loss: 0.3459 - val_accuracy: 0.8732
Epoch 4/10
1578/1578 [==============================] - 50s 32ms/step - loss: 0.2484 - accuracy: 0.9052 - val_loss: 0.3265 - val_accuracy: 0.8815
Epoch 5/10
1578/1578 [==============================] - 55s 35ms/step - loss: 0.1969 - accuracy: 0.9271 - val_loss: 0.3160 - val_accuracy: 0.8838
Epoch 6/10
1578/1578 [==============================] - 57s 36ms/step - loss: 0.1527 - accuracy: 0.9507 - val_loss: 0.3115 - val_accuracy: 0.8930
Epoch 7/10
1578/1578 [==============================] - 51s 32ms/step - loss: 0.1086 - accuracy: 0.9658 - val_loss: 0.3140 - val_accuracy: 0.8999
Epoch 8/10
1578/1578 [==============================] - 46s 29ms/step - loss: 0.0874 - accuracy: 0.9729 - val_loss: 0.3391 - val_accuracy: 0.9006
Epoch 9/10
1578/1578 [==============================] - 51s 32ms/step - loss: 0.0695 - accuracy: 0.9802 - val_loss: 0.3392 - val_accuracy: 0.8974
Epoch 10/10
1578/1578 [==============================] - 46s 29ms/step - loss: 0.0517 - accuracy: 0.9861 - val_loss: 0.3559 - val_accuracy: 0.9010

Process finished with exit code 0
```

side_effects：word2vec+RNN



2.5 实验结果对比

2.5.1 传统机器学习模型

特征向量化方式	模型	准确率
CountVectorizer	朴素贝叶斯	44.6%
	SVM	42.5%
	决策树	41.3%
TfidfVectorizer	朴素贝叶斯	43.5%
	SVM	45.0%
	决策树	41.3%

可以看出，使用传统机器学习模型对 attitude 进行预测时，当特征向量化方式采取 TfidfVectorizer，模型采取 SVM 时，预测准确率最高，为 45%（因为是多分类问题，预测难度较大，准确率较低）。

2.5.2 基于神经网络的模型

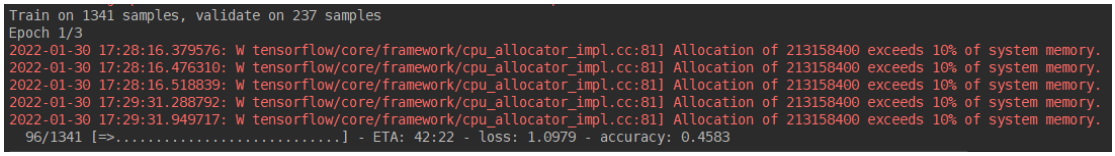
任务类型	神经网络模型	准确率
对 attitude 进行预测	word2vec+RNN	45.0%

	word2vec+CNN+GRU	42.0%
	word2vec+Bi-GRU	44.3%
对 vaccine 进行预测	word2vec+RNN	87.3%
	word2vec+CNN+GRU	81.7%
	word2vec+Bi-GRU	68.1%
对 side_effects 进行预测	word2vec+RNN	90.1%
	word2vec+CNN+GRU	89.0%
	word2vec+Bi-GRU	87.8%

可以看出，word2vec+RNN 的效果最好且训练用时最短，而 word2vec+Bi-GRU 效果最差且训练用时最长，从之前的网络结构图也可以看出，word2vec+RNN 模型的训练参数最少，因此更容易训练，而 word2vec+Bi-GRU 模型的训练参数最多，因此最难训练。对于后两个预测任务模型表现较好，但对于 attitude 预测的任务表现仍不尽如人意，从上一部分使用机器学习模型得到的结果也可以得出相同结论，即对该数据集来说很难训练出一个在 attitude 任务上表现很好的模型。

3 遇到的问题:

1.使用神经网络训练模型过程出现内存不足的报错:



一种可能是我的电脑内存空间确实不足，另一种可能是因为一开始我使用 BOW 分词导致输入维度过大（一条训练数据 5551 维的输入），若改为 word2vec 的固定 500 维后可以加快训练速度并解决该问题。

2.对 pandas 数据处理库的不熟练使用:

查阅官方文档及相关博客，多次尝试各种方法函数的用法，实现相应的功能，日后将加强相关 python 库的学习。

4 实验总结:

本次 NLP 的结课作业，一开始我是想在网上找一个简单的题目当做自选题目来完成，但后来觉得这样是给自己降低难度了，于是还是在老师给的三个题目中选择了一个有挑战性的且自己感兴趣的任务来做。在此次任务中，我参与了数据集的标注、实验与模型的设计实现前后两个工作，完整地感受了一个真实场景任务上的流程，对 NLP 技术在实际生活应用上有了更深的理解与掌握。同时，在实验过程中，对 pandas 库以及其他文本处理工具有了初步的学习，对各种机器学习模型和神经网络模型也有了更深的掌握，通过实验对比了不同模型之间的性能差异，为日后无论是 NLP 上的进一步学习还是人工智能其他领域的学习都有所帮助。总之，此次实验给我带来的收获很大，日后我也将继续努力学习相关知识完善自己。