基于LSTM（RNN）在NLP领域的情绪分析预测

一、研究背景

在当今信息时代，社交媒体、在线评论和其他大规模的文本数据源不断涌现，用户在这些平台上表达了丰富的情感和观点。对于企业、政府和学术界而言，深入理解这些情感背后的含义对于决策制定和社会反馈至关重要。情感分析，作为自然语言处理（NLP）领域的一个关键任务，致力于自动识别和理解文本中的情感倾向。情感分析预测在商业、社交和政治等领域具有广泛的应用。通过深入分析用户在社交媒体上的评论、产品评论或新闻文章中的情感，企业可以更好地了解消费者对产品和服务的感受。政府可以通过监测公众的情感倾向来更好地了解社会舆论，以更有针对性地制定政策。在学术研究中，情感分析还可以用于分析文本数据集中的情感变化趋势，揭示人类社会的情感体验。

二、研究目标

学习LSTM、NLP相关基础知识，能够训练出对应情绪分析的预测模型，并且模型具有一定的准确性；同时对于输入的情绪文本能够进行识别情绪类别，即能够对对应问题得到相应答案。简而言之抱着学习的态度，获得相应的模型预测成果。

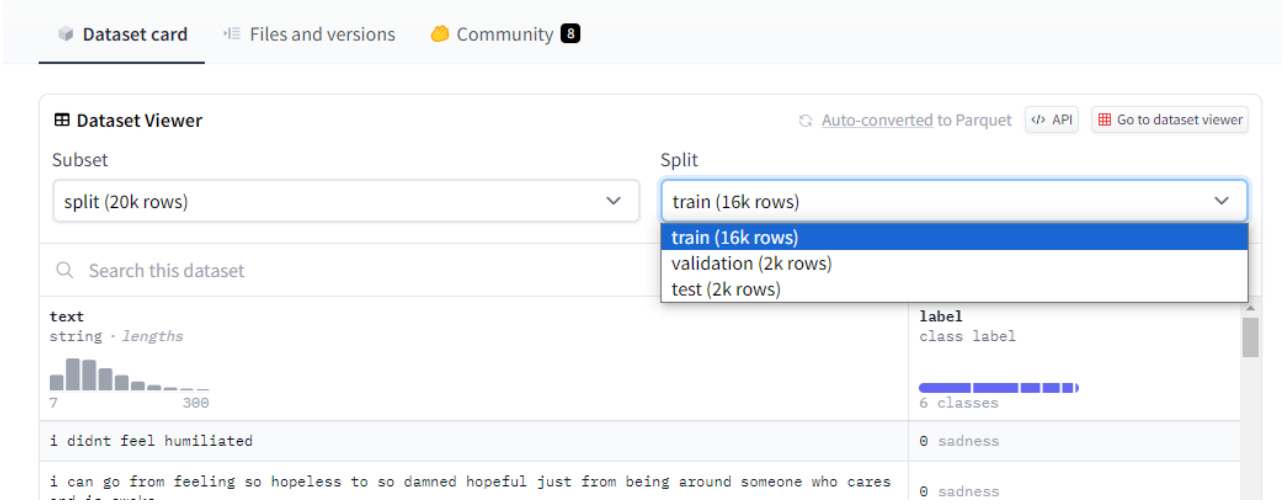
三、研究工具

- Google Colab Notebook，它内部已经集成了相应CPU、GPU或TPU能力且支持在线Jupyter Notebook;
- HuggingFace数据集，其中有相应的情绪数据集，可直接使用；
- Python绘画图库，便于进行情绪文本分析，以及模型预测结果分析；

四、收集数据集

上文已提到数据集直接使用的HuggingFace情绪数据集，地址为：https://huggingface.co/datasets/dair-ai/emotion/，其中：

- 训练数据集为：16k
- 验证数据集为：2k
- 测试数据集为：2k



数据格式为：

```
{"text":"im feeling rather rotten so im not very ambitious right now","label":0}
{"text":"im updating my blog because i feel shitty","label":0}
{"text":"i never make her separate from me because i don t ever want her to feel like i m ashamed with her","label":0}
{"text":"i left with my bouquet of red and yellow tulips under my arm feeling slightly more optimistic than when i arrived","label":1}
```

五、文本预处理

由于数据列非常整齐，且只有两列文本列（text）和标签列（label）预处理无需考虑数据列的问题，只需进行相应的NLP的处理：

5.1、用NLTK标记化文本

5.2、对训练数据集和验证数据集进行文本分词

```
def tokenization(inputs):
        return word_tokenize(inputs) #REFERENCE[1]

# 对 训练数据集 和 验证数据集 进行文本分词
train['text_tokenized'] = train['text'].apply(tokenization)
validation['text_tokenized'] = validation['text'].apply(tokenization)
```

```
[51] train.head()
```

	text	label	length_of_text	text_tokenized
0	i didnt feel humiliated	0	4	[i, didnt, feel, humiliated
1	i can go from feeling so hopeless to so damned...	0	21	[i, can, go, from, feeling, so, hopeless, to, ..
2	im grabbing a minute to post i feel greedy wrong	3	10	[im, grabbing, a, minute, to, post, i, feel, g

5.3、对文本进行停止词删除

```
[53] stop_words = set(stopwords.words('english'))

     def stopwords_remove(inputs):
         return [item for item in inputs if item not in stop_words]

     # 对 训练数据集 和 验证数据集 进行停用词去除
     train['text_stop'] = train['text_tokenized'].apply(stopwords_remove)
     validation['text_stop'] = validation['text_tokenized'].apply(stopwords_remove)

     train.head()
```

	text	label	length_of_text	text_tokenized	text_stop
0	i didnt feel humiliated	0	4	[i, didnt, feel, humiliated]	[didnt, feel, humiliated]
1	i can go from feeling so hopeless to so damned...	0	21	[i, can, go, from, feeling, so, hopeless, to, ...	[go, feeling, hopeless, damned, hopeful, aroun...

5.4、对文本数据词元化

```
[55] # 进行词形还原（lemmatization）：词形还原是将词语转换为其基本形式（称为词元或词根）的过程，以减少词汇的复杂性。在这段代码中，词形还原的目标是将单词还原为其原始的动词形式。
     lemmatizer = WordNetLemmatizer()

     def lemmatization(inputs):
         return [lemmatizer.lemmatize(word=x, pos='v') for x in inputs]

     # 对 训练数据集 和 验证数据集 进行
     train['text_lemmatized'] = train['text_stop'].apply(lemmatization)
     validation['text_lemmatized'] = validation['text_stop'].apply(lemmatization)

     train.head()
```

	text	label	length_of_text	text_tokenized	text_stop	text_lemmatized
0	i didnt feel humiliated	0	4	[i, didnt, feel, humiliated]	[didnt, feel, humiliated]	[didnt, feel, humiliate]
1	i can go from feeling so hopeless to so damned...	0	21	[i, can, go, from, feeling, so, hopeless, to, ...	[go, feeling, hopeless, damned, hopeful, aroun...	[go, feel, hopeless, damn, hopeful, around, so...
2	im grabbing a minute to post i feel greedy wrong	3	10	[im, grabbing, a, minute, to, post, i, feel, g...	[im, grabbing, minute, post, feel, greedy, wrong]	[im, grab, minute, post, feel, greedy, wrong]
3	i am ever feeling nostalgic about the fireplac...	2	18	[i, am, ever, feeling, nostalgic, about, the, ...	[ever, feeling, nostalgic, fireplace, know, st...	[ever, feel, nostalgic, fireplace, know, still...

5.5、还原文本

```
+ 代码  + 文本

[56] # 经过词形还原后的文本数据（存储在text_lemmatized列中的列表形式）转换为单个字符串，并将结果存储在一个新的列text_cleaned中
     train['text_cleaned'] = train['text_lemmatized'].str.join(' ')
     validation['text_cleaned'] = validation['text_lemmatized'].str.join(' ')

     train.head() # Final form of the dataset
```

	text	label	length_of_text	text_tokenized	text_stop	text_lemmatized	text_cleaned
0	i didnt feel humiliated	0	4	[i, didnt, feel, humiliated]	[didnt, feel, humiliated]	[didnt, feel, humiliate]	didnt feel humiliate
1	i can go from feeling so hopeless to so damned...	0	21	[i, can, go, from, feeling, so, hopeless, to, ...	[go, feeling, hopeless, damned, hopeful, aroun...	[go, feel, hopeless, damn, hopeful, around, so...	go feel hopeless damn hopeful around someone c...
2	im grabbing a minute to post i feel greedy wrong	3	10	[im, grabbing, a, minute, to, post, i, feel, g...	[im, grabbing, minute, post, feel, greedy, wrong]	[im, grab, minute, post, feel, greedy, wrong]	im grab minute post feel greedy wrong

六、特征提取

使用TensorFlow对训练数据集和验证数据集文本的词元进行标记和填充数字序列，以便提取词元与情绪标识的特征关联关系。

七、模型开发

7.1、创建模型

```
# 创建 Sequential 模型，是 Keras 中最简单的一种模型类型，按顺序堆叠层
model = Sequential()

# 这是一个嵌入层（Embedding Layer）；它将整数编码的单词映射到具有固定大小的稠密向量（16维）中。num_words 是词汇表的大小，maxlen 是输入序列的长度
model.add(Embedding(num_words, 16, input_length=maxlen))
# 这是一个全局平均池化层，对输入的所有时间步骤计算平均值，将输入序列的长度降至1。这有助于减小序列长度，提取关键信息
model.add(GlobalAvgPool1D())

# 这是三个双向 LSTM 层，每个都带有一个激活函数为 'relu' 的激活函数。每个 LSTM 层的返回序列设置为 True，表示输出完整的序列而不仅仅是最后一个时间步的输出
# 在每个 LSTM 层之后，都添加了一个 Dropout 层，用于随机断开一定比例的神经元，以防止过拟合。
tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(50, return_sequences=True, activation='relu'))
model.add(Dropout(0.3))

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(40, activation='relu', return_sequences=True))
model.add(Dropout(0.3))

tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(40, activation='relu'))
model.add(Dropout(0.3))

# 这是一个密集连接层（Dense Layer），具有6个输出单元，使用 softmax 激活函数。这是因为模型的任务是对6个类别进行分类
model.add(Dense(6, activation='softmax'))

# 这一行编译了模型，指定了损失函数为稀疏分类交叉熵（sparse categorical crossentropy），优化器为 'adam'，并使用准确度作为评估指标
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# 显示每个层的参数数量和总参数数量等信息，助于了解模型的整体结构
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 40, 16)            160000
```

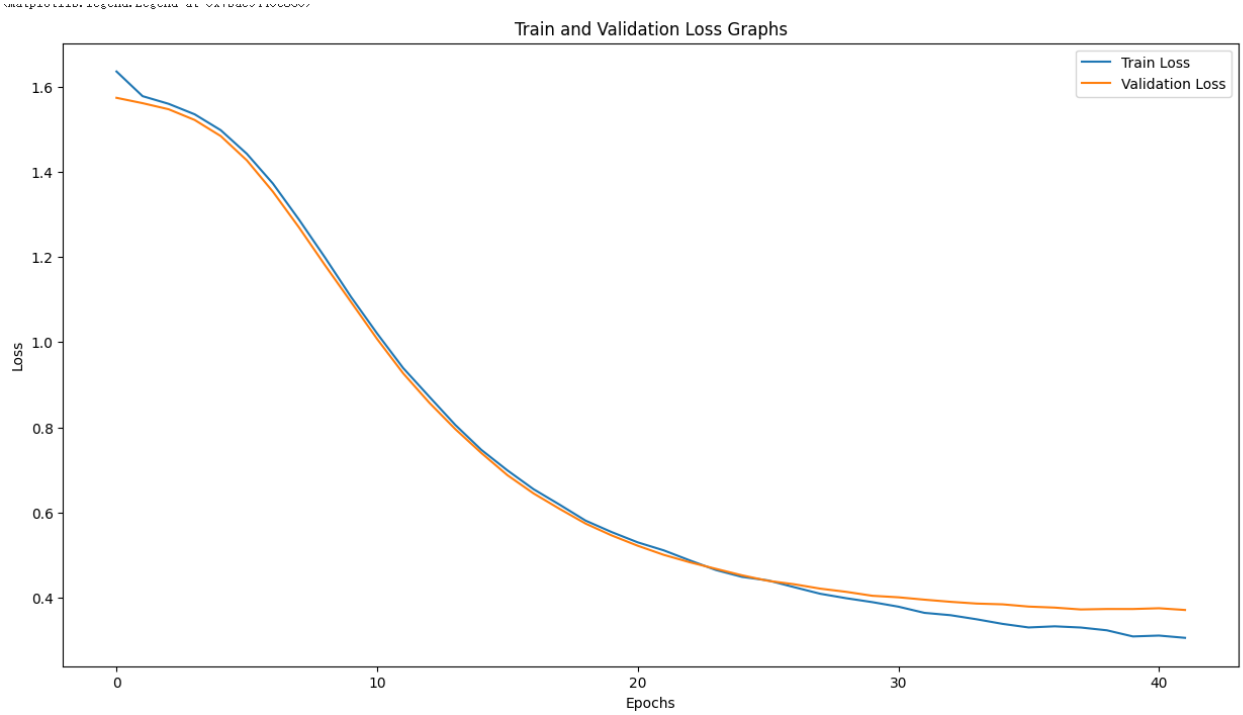
7.2、训练模型

```
# restore_best_weights=True：在停止训练后，回滚到在验证集上性能最好的那一轮的权重。
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', mode='auto', patience=5, restore_best_weights=True)

# 模型的训练
# epochs=80 表示训练的最大轮数为80
# 模型使用 Padded_train 和 train['label'] 进行训练，使用验证集 (Padded_val, validation['label']) 进行验证
# 同时，通过 callbacks 参数传递 Early Stopping 回调函数，以便在训练期间启用 Early Stopping
# 训练的历史信息存储在 hist 变量中
epochs = 80
hist = model.fit(Padded_train,
                 train['label'],
                 epochs=epochs,
                 validation_data=(Padded_val, validation['label']),
                 callbacks=[early_stopping])

# 损失（Loss）：衡量模型在训练数据上的拟合程度。损失越低越好。
# 准确度（Accuracy）：模型在训练集或验证集上的分类准确度。它表示模型正确预测的样本比例。
# 训练过程中可能出现的一些观察：
    #随着训练的进行，训练集和验证集上的损失和准确度可能会有所改善。
    #"Early Stopping" 回调函数可能会在验证集上的准确度不再提高时停止训练，以防止过拟合。
    #可以根据验证集的性能调整模型架构、超参数等，以提高模型的泛化性能。
```

7.3、模型训练信息

```
Epoch 35/80
500/500 [==============================] - 5s 10ms/step - loss: 0.3381 - accuracy: 0.8993 - val_loss: 0.3840 - val_accuracy: 0.8730
Epoch 36/80
500/500 [==============================] - 2s 4ms/step - loss: 0.3297 - accuracy: 0.8994 - val_loss: 0.3787 - val_accuracy: 0.8790
Epoch 37/80
500/500 [==============================] - 2s 4ms/step - loss: 0.3323 - accuracy: 0.8963 - val_loss: 0.3763 - val_accuracy: 0.8805
Epoch 38/80
500/500 [==============================] - 2s 4ms/step - loss: 0.3295 - accuracy: 0.8981 - val_loss: 0.3720 - val_accuracy: 0.8785
Epoch 39/80
500/500 [==============================] - 2s 5ms/step - loss: 0.3230 - accuracy: 0.9003 - val_loss: 0.3731 - val_accuracy: 0.8730
Epoch 40/80
500/500 [==============================] - 3s 7ms/step - loss: 0.3086 - accuracy: 0.9053 - val_loss: 0.3730 - val_accuracy: 0.8760
Epoch 41/80
500/500 [==============================] - 2s 4ms/step - loss: 0.3106 - accuracy: 0.9026 - val_loss: 0.3748 - val_accuracy: 0.8755
Epoch 42/80
500/500 [==============================] - 2s 4ms/step - loss: 0.3053 - accuracy: 0.9066 - val_loss: 0.3707 - val_accuracy: 0.8775
```

7.4、训练和验证损失



八、模型评估

8.1、准备测试数据集

```
[67] test['text_tokenized'] = test['text'].apply(tokenization)
     test['text_stop'] = test['text_tokenized'].apply(stopwords_remove)
     test['text_lemmatized'] = test['text_stop'].apply(lemmatization)
     test['text_cleaned'] = test['text_lemmatized'].str.join(' ')

     Tokenized_test = tokenizer.texts_to_sequences(test['text_cleaned'])
     Padded_test = pad_sequences(Tokenized_test, maxlen=maxlen, padding='pre')

     test['label'] = test['label'].replace(label_)

     test_evaluate = model.evaluate(Padded_test, test['label'])
```

8.2、测试精度结果

```
63/63 [==============================] - 0s 5ms/step - loss: 0.3716 - accuracy: 0.8765
```

可见**0.8765**的精准度还是算高的！

九、模型预测

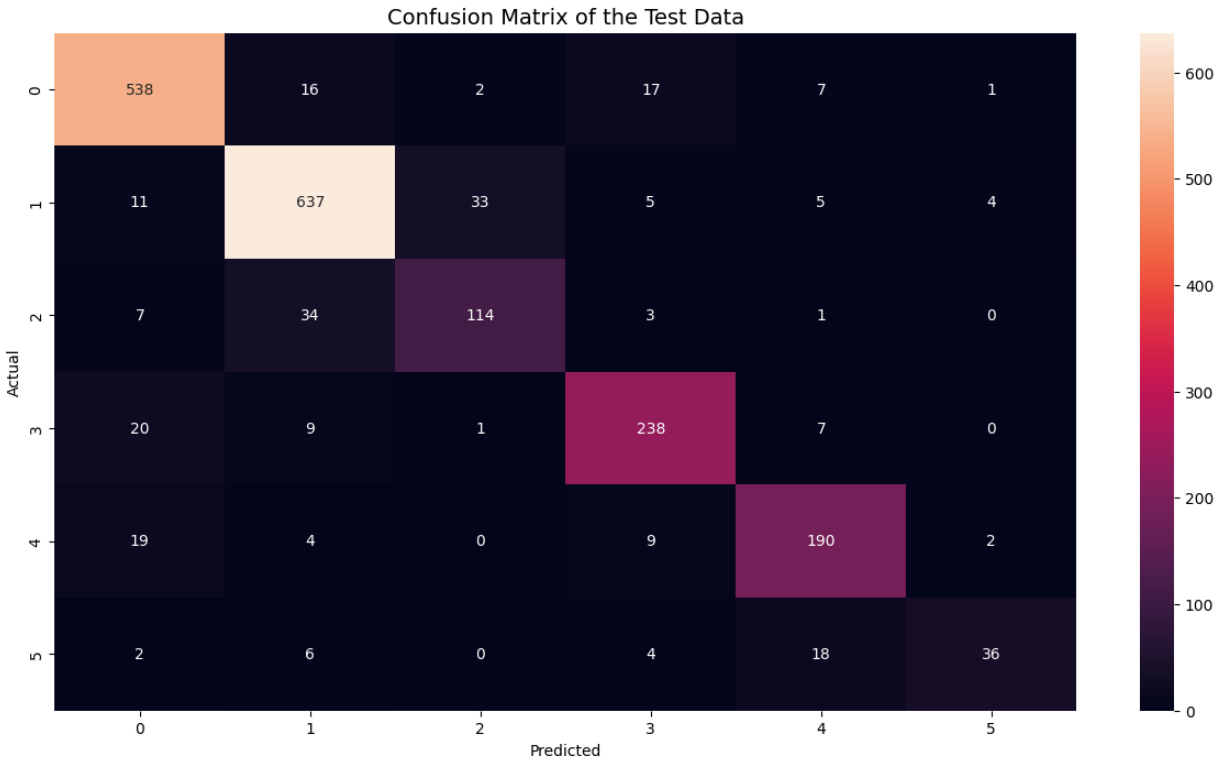
9.1、在数据集中进行预测

```python
def make_predictions(text_input):
        text_input = str(text_input)
        text_input = tokenization(text_input)
        text_input = stopwords_remove(text_input)
        text_input = lemmatization(text_input)
        text_input = ' '.join(text_input)
        text_input = tokenizer.texts_to_sequences([text_input])
        text_input = pad_sequences(text_input, maxlen=maxlen, padding='pre')
        text_input = np.argmax(model.predict(text_input))

        if text_input == 0:
                print('预测情感: 悲伤')
        elif text_input == 1:
                print('预测情感: 高兴')
        elif text_input == 2:
                print('预测情感: 喜爱')
        elif text_input == 3:
                print('预测情感: 愤怒')
        elif text_input == 4:
                print('预测情感: 恐惧')
        else:
                print('预测情感: 惊讶')
        return text_input

label_ = {0: "悲伤", 1: "高兴", 2: "喜爱", 3: "愤怒", 4: "恐惧", 5: "惊讶"}
test['label'] = test['label'].replace(label_)

i = random.randint(0, len(test) - 1)

print('测试预测内容:', test['text'][i])
print(' ')
print('实际情感:', test['label'][i])
make_predictions(test['text'][i])
print('-'*50)
print('测试预测内容:', test['text'][i+1])
print(' ')
print('实际情感:', test['label'][i+1])
make_predictions(test['text'][i+1])
```

9.2、预测结果

```
测试预测内容: i don t know what to feel as in i am not sure should i feel sad cause it is ending or should i feel glad that it is over and i can move on

实际情感: 高兴
1/1 [==============================] - 0s 185ms/step
预测情感: 高兴
--------------------------------------------------
测试预测内容: i feel so discontent so guilty so pathetic so lonley and i hate myself for it

实际情感: 悲伤
1/1 [==============================] - 0s 37ms/step
预测情感: 悲伤
0
```

9.3、预测混淆矩阵



9.4、数据预测

```
# 我喜欢人民大学
make_predictions("i love RUC!!!")
```

```
1/1 [==============================] - 0s 41ms/step
预测情感: 喜爱
2
```

```
[81] # 胡鹤老师的线上课程马上结束了，我会非常想念它
make_predictions("Teacher Hu He's online course will be over soon, I will miss it very much")
```

```
1/1 [==============================] - 0s 46ms/step
预测情感: 悲伤
0
```

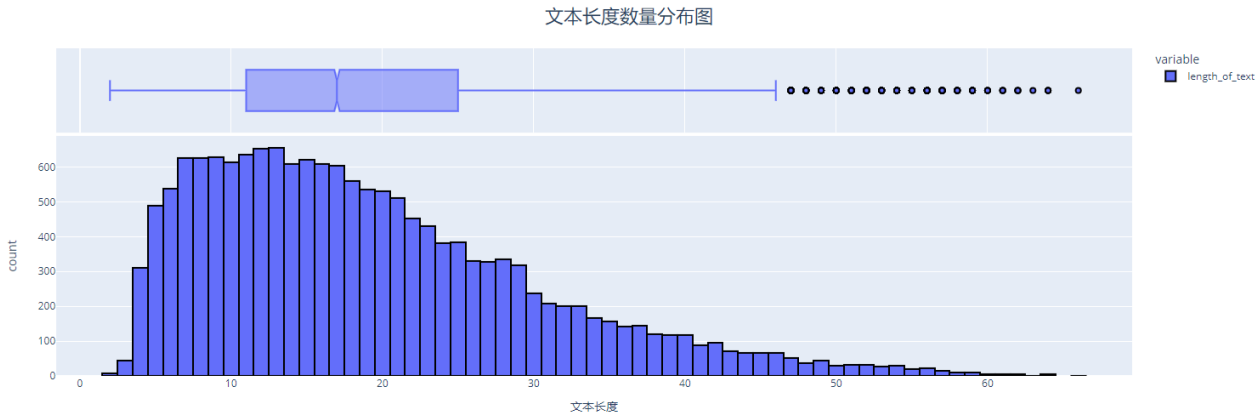
```
[82] # 五月天乐队竟然会假唱，简直让人难以置信
make_predictions("It's hard to believe that Mayday can lip-sync")
```

```
1/1 [==============================] - 0s 29ms/step
预测情感: 愤怒
3
```

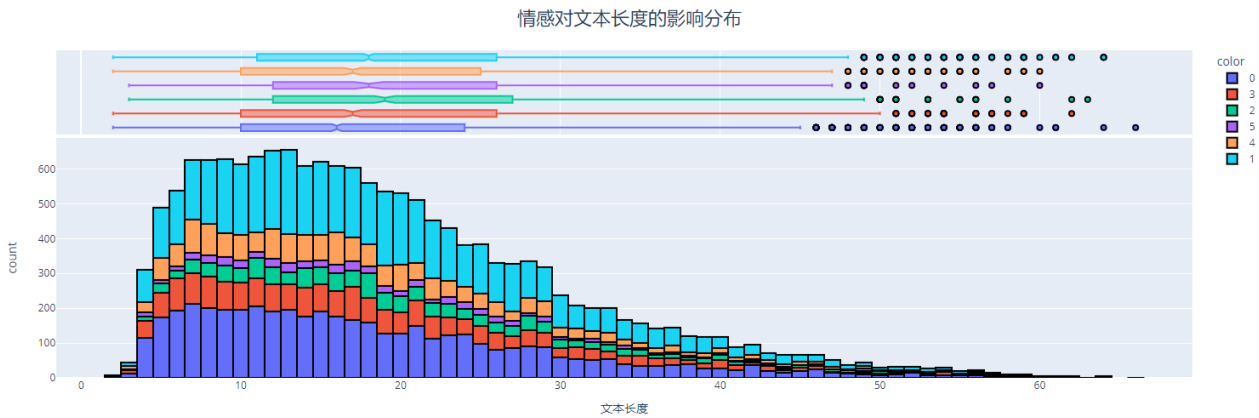
可见以上的语言预测都通过了测试，准确率确实跟混淆矩阵中显示的情况一致。

十、附图

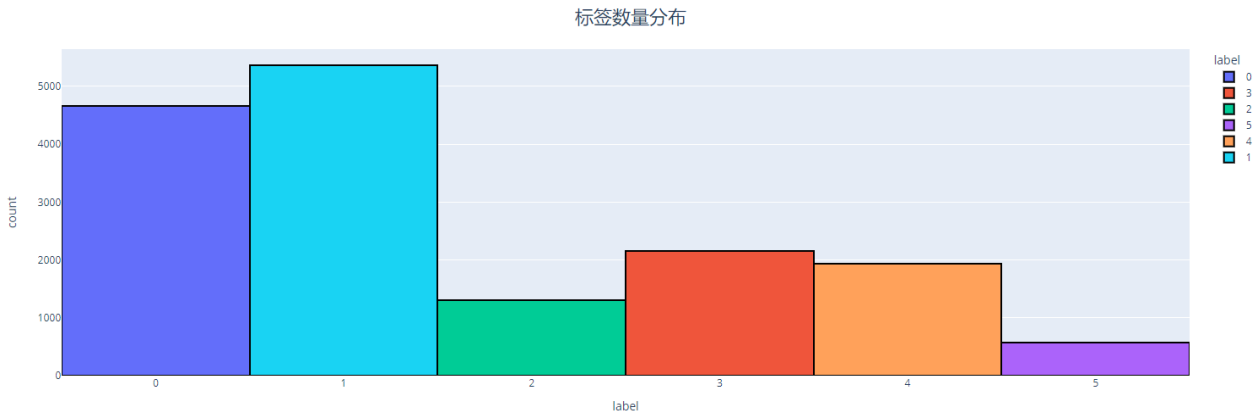
10.1、数据文本长度分布



10.2、情绪对文本长度的影响

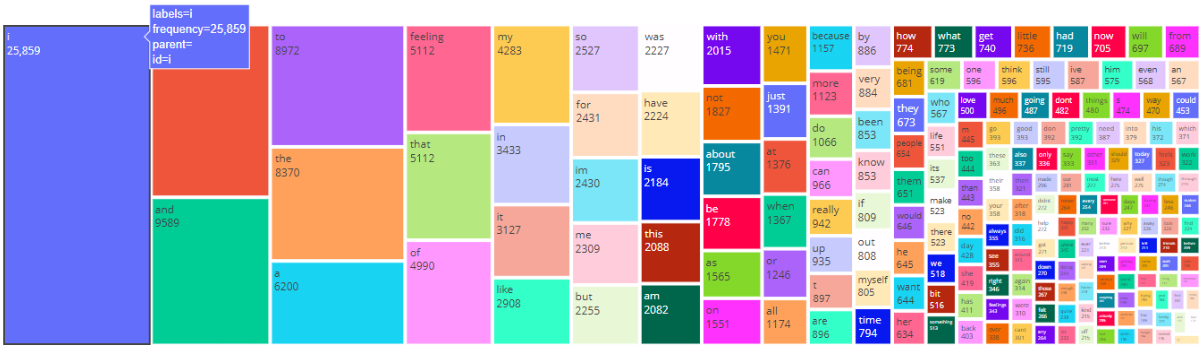


10.3、标签数量分布



10.4、词汇出现数量

词汇出现的次数


labels=i
frequency=25,859
parent=
id=i
i
25,859
and
9589
to
8972
the
8370
a
6200
feeling
5112
that
5112
of
4990
my
4283
in
3433
it
3127
like
2908
so
2527
for
2431
im
2430
me
2309
but
2255
was
2227
have
2224
is
2184
this
2088
am
2082
with
2015
not
1827
about
1795
be
1778
as
1565
on
1551
you
1471
just
1391
at
1376
when
1367
or
1246
all
1174
because
1157
more
1123
do
1066
can
966
really
942
up
935
t
897
are
896
by
886
very
884
been
853
know
853
if
809
out
808
myself
805
time
794
how
774
what
773
get
740
little
736
had
719
now
705
will
697
from
689