数据集：Sentiment Analysis

kaagle 链接：https://www.kaggle.com/datasets/suchintikasarkar/sentiment-analysis-for-mental-health

该数据集为情感分析相关，用于多分类问题，每行有三条数据：索引，文本(statement)和状态(status)。

```
# 读取csv文件
df = pd.read_csv("./Sentiment_Analysis.csv")
print(df.head())
```

```
   Unnamed: 0                                          statement   status
0           0                                        oh my gosh  Anxiety
1           1  trouble sleeping, confused mind, restless hear...  Anxiety
2           2  All wrong, back off dear, forward doubt. Stay ...  Anxiety
3           3  I've shifted my focus to something else but I'...  Anxiety
4           4  I'm restless and restless, it's been a month n...  Anxiety
```

共有 53043 个样本，注意这个数据集不是平衡的，不同标签的样本数是不同的。

```
# 不同标签的个数
print(df["status"].value_counts())
```

```
status
Normal                  16351
Depression              15404
Suicidal                10653
Anxiety                  3888
Bipolar                  2877
Stress                   2669
Personality disorder     1201
Name: count, dtype: int64
```

并且存在部分样本有缺失值，所以需要对数据集做初步筛选。

```
# 处理包含缺失值的行，删除开始的索引列
df.drop("Unnamed: 0",axis=1,inplace=True)
df.dropna(axis=0, inplace=True)
```

标签值编码：使用 LabelEncoder

```
le = LabelEncoder() # sklearn.preprocessing
df["status"] = le.fit_transform(df["status"])
print(df["status"].value_counts())
```

```
status
3    16343
2    15404
6    10652
0     3841
1     2777
5     2587
4     1077
Name: count, dtype: int64
```

计算权重：使用注意力机制来解决类不平衡问题

不同类别的样本数不同，所以为不同类别分配不同的权重，样本数少的类别权重更高，这样通过注意力机制可以在一定程度上解决类不平衡带来的问题。

前面提到的 SMOTE 算法，在二分类问题上有不错的效果，在多分类问题上还没有做测试，后续可能会尝试使用 SMOTE 算法优化。

```
# 计算每个类别的权重
# 对于标签的不同类别，占总样本的比例越小，其权重就会越大
class_weights = compute_class_weight( # sklearn.utils.class_weight
    class_weight="balanced",
    classes=np.unique(df["status"]),
    y=df["status"]
)
print(class_weights)
```

```
[1.95934838 2.71006739 0.48856512 0.46049423 6.98779679 2.90910597
 0.70652057]
```

其余预处理过程不做详细介绍。

数据集类的定义：

这里和原先的不同。

原先是传入特征值和标签值，并根据索引返回就行了，但是在这个例子中我们需要预处理特征值（将单词映射为索引），并且由于特征向量长度不一，较短的需要使用填充，较长的进行截断，所以特征中存在无意义的填充字符，这些需要进行标记。

```
class SentimentDataset(Dataset):
    def __init__(self, labels, tokens):
        super().__init__()
        self.labels = labels
        self.tokens = tokens
    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return {
            "input_ids": self.tokens["input_ids"][idx],
            "attention_mask": self.tokens["attention_mask"][idx],
            "label": torch.tensor(self.labels[idx], dtype = torch.long)
        }
```

这次的数据集类会返回三个数据：input_ids 为原特征的各个单词映射后的向量，attention_mask 为标记有效位置（有效为 1，填充部分为 0），label 为原先的标签值。

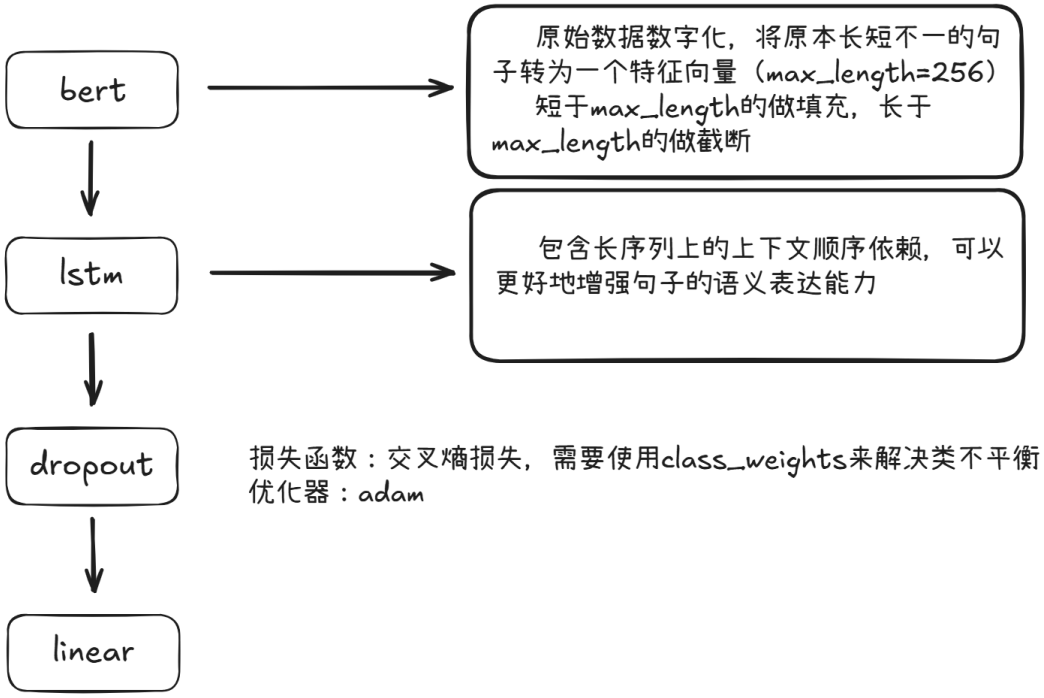
传入的 token 不是原特征值，而是使用预训练的模型 bert-base-uncased，这个模型可用于预处理英文数据，使用 transformer 中的 AutoTokenizer 加载。

```
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

training_tokens = tokenizer(
    X_train.to_list(), # 需要转换的数据
    truncation=True, # 句子过长时截断到max_length
    padding="max_length", # 所有输入填充到max_length长度
    max_length=256, # 设置max_length
    return_tensors="pt" # 返回pytorch的格式
)
print(type(training_tokens))
```

```
<class 'transformers.tokenization_utils_base.BatchEncoding'>
```

定义神经网络：



注意最后一层 linear 输出时不要加 Softmax 层，因为在损失函数 CrossEntropyLoss 中已经有 Softmax 了。

```
# 定义神经网络
class Net(nn.Module):
    def __init__(self, bert_model_name="bert-base-uncased",num_classes=7, dropout=0.3, hidden_dims=128):
        super().__init__()

        self.bert = BertModel.from_pretrained(bert_model_name)

        for param in self.bert.parameters():
            param.requires_grad = False

        self.lstm = nn.LSTM(
            input_size = self.bert.config.hidden_size,
            hidden_size=hidden_dims,
            bidirectional=True,
            batch_first=True
        )

        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(hidden_dims*2, num_classes)

    def forward(self, input_ids, attention_mask):
        with torch.no_grad():
            outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
            embedding = outputs.last_hidden_state

        intermediate_hidden_outs, (final_hissen_state, call_state) = self.lstm(embedding)
        hidden = torch.cat((final_hissen_state[-2], final_hissen_state[-1]), dim=1)

        out = self.dropout(hidden)

        logits = self.linear(out)
        return logits
```

模型训练：

这个神经网络训练需要花很多时间，所以我 epoch 只有 5 轮，训练方法和之前类似，

这中间涉及一个 pytorch 上的问题：交叉熵损失要求传入的两个参数，第一个为 float32，第二个为 long，但是在使用 lstm 时会自动把类型变为 float64，导致出现问题，这个问题卡了很长时间。

```
def train():
    model.train()
    for epoch in range(epochs):
        total_loss = 0.0
        for batch in train_loader:

            features = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["label"].long().to(device)

            optimizer.zero_grad()
            outputs = model(features, attention_mask)
            loss = criterion(outputs.float(), labels)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)
        print(f"epoch: [{epoch}/{epochs}], loss: {avg_loss}")
```

模型输出结果：

```
epoch: [0/5], loss: 0.9416846592861794
epoch: [1/5], loss: 0.6652360024834413
epoch: [2/5], loss: 0.5811502724302892
epoch: [3/5], loss: 0.5330520166102203
epoch: [4/5], loss: 0.4934483202603549
model saved.
```

模型评估：

使用测试集对训练好的模型做测试，可以得到最后的准确率在 79.6%

虽然模型训练和测试使用了两个 py 脚本，数据集分割了两次，但是由于使用了相同的 random_state = 42，所以不存在测试集分割不同。

```
avg_val_loss : 0.5313371097861912, accuracy : 0.7961469108854513
```

	precision	recall	f1-score	support
Normal	0.81	0.85	0.83	768
Depression	0.84	0.83	0.83	556
Suicidal	0.83	0.63	0.72	3081
Anxiety	0.94	0.93	0.93	3269
Bipolar	0.53	0.83	0.65	215
Stress	0.65	0.76	0.70	517
Personality Disorder	0.65	0.81	0.72	2131
accuracy			0.80	10537
macro avg	0.75	0.81	0.77	10537
weighted avg	0.81	0.80	0.80	10537