情感分析大作业实验报告

组员：高鹏轩、邹佳峻、张露雨

一、简介

（1.1）背景介绍

微博，即微型博客的简称，是一个基于用户关系的信息分享、传播和获取的平台，用户可以通过 Web\WAP 等客户端组件，用 140 字左右的文字更新信息，并实施即时分享。微博给与了网络用户更快捷更自由的方式来沟通信息和表达观点，已经成为国内最为热门的互联网应用之一。本文研究的对象就是外网的一种微博 Twitter，对用户在其上输出的大量的语句进行人工标注后进行情感分析。

情感分析是指分析说话者在传递信息时所隐含的情绪状态，对说话者的态度、意见进行判断和评估。情感分类是对带有感情色彩的主观性文本进行分析、推理的过程，即分析对说话人的态度，倾向正面，还是反面。情感分析在海量数据上的应用，将有助于完善互联网的舆情监控系统；丰富和拓展企业的营销能力；通过波动分析，实现对物理世界异常或突发事件的检测；此外，还可以应用于心理学、社会学、金融预测等领域的研究，故而基于情感分析的研究有着非常重要的现实意义。此次任务本质是为二分类任务。

（1.2）任务简介

样本量：训练集 160 万条，收集的数据集带有六个维度的信息，包括标签、ID、日期、话题、用户及推文内容。正负样本比例为 1:1。

本次实验我们小组的研究方法主要是先综合使用逻辑回归、支持向量机、随机梯度下降和集成学习的技术对给定的数据集进行直接性的学习，考察使用不同的分类方法下的准确率并作比较分析，然后使用两个已经训练好的大数据模型 BERT 和 GPT2 进行对原数据集分类的方法进行微调。

二、模型/方法

（2.1）机器学习模型

机器学习使用 skleanr 中的 4 个模型：

1. 线性分类支持向量机 LinearSVC（）
2. 逻辑回归算法 LogisticRegression（）
3. 随机梯度下降分类器 SGDClassifier（）
4. 集成学习 VotingClassifier（）

（2.2）BERT 模型

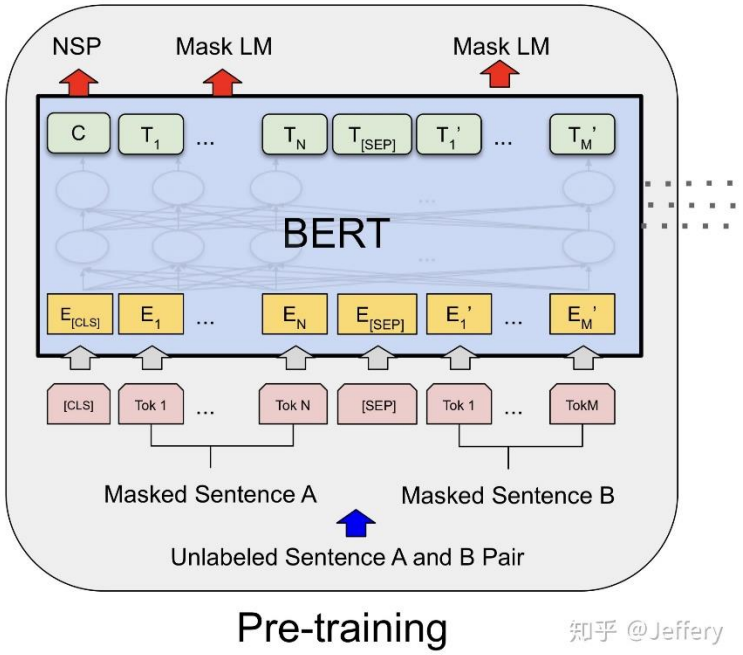
BERT（Bidirectional Encoder Representation from Transformers），是一个预训练的语言表征模型。它强调了不再像以往一样采用传统的单向语言模型或者把两个单向语言模型进行浅层拼接的方法进行预训练，而是采用新的 masked language model（MLM），以致能生成深度的双向语言表征。

该模型有以下主要优点：

- 采用 MLM 对双向的 Transformers 进行预训练，以生成深层的双向语言表征。
- 预训练后，只需要添加一个额外的输出层进行微调，就可以在各种各样的下游

任务中取得 state-of-the-art 的表现。在这过程中并不需要对 BERT 进行任务特定的结构修改。

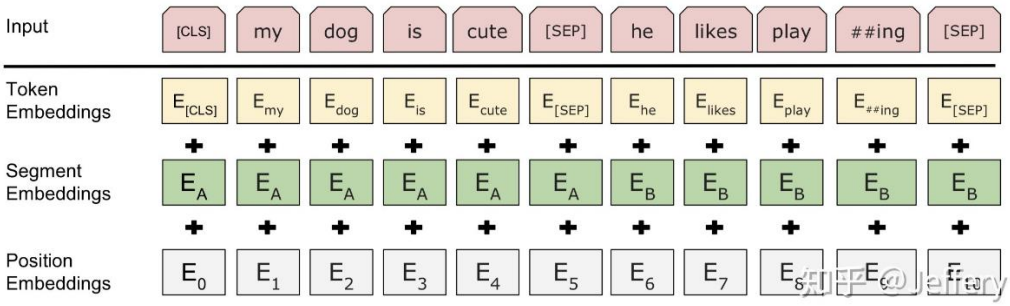
BERT 的架构基于原始实现的 multi-layer bidirectional Transformer 编码器，由多层 transfromer 堆叠实现主体结构。



BERT 目前有两种可用的变体：

- BERT Base：12 层，12 个注意力头，768 个隐藏和 110M 参数
- BERT Large：24 层，16 个注意力头，1024 隐藏和 340M 参数

BERT 的输入为每一个 token 对应的表征，该表征是由三部分组成的，分别是对应的 token，分割和位置 。



为了完成具体的分类任务，除了单词的 token 之外，在输入的每一个序列开头都插入特定的分类 token（[CLS]）。

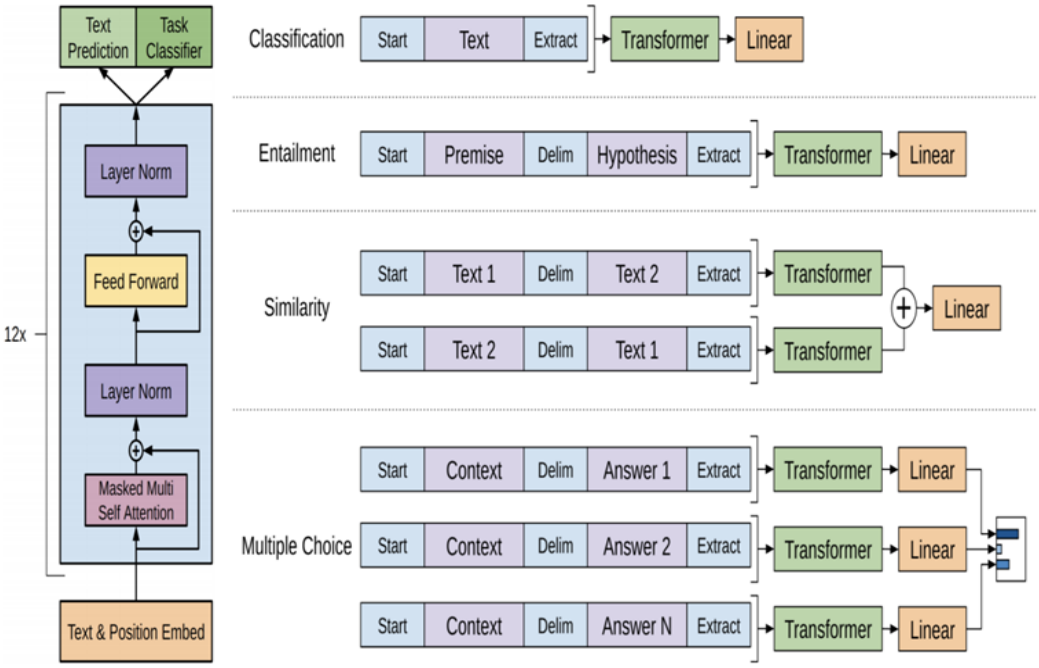
在序列 tokens 中把分割 token（[SEP]）插入到每个句子后，以分开不同的句子 tokens。

为每一个 token 表征都添加一个可学习的分割 embedding 来指示其属于句子 A 还是句子 B。

BRET 预训练包含两个任务：Next Sentence Prediction（NSP）与 Masked Language Model（MLM）。MLM 以 15%的概率用 mask token （[MASK]）随机地对每一个训练序列中的 token 进行替换，然后预测出[MASK]位置原有的单词，这使得 BERT 能够不受单向语言模型所限制，对所有的 token 都敏感，以致能抽取出任何 token 的表征信息。NSP 预测两个句子是否连在一起，来获取句子层次的表征。

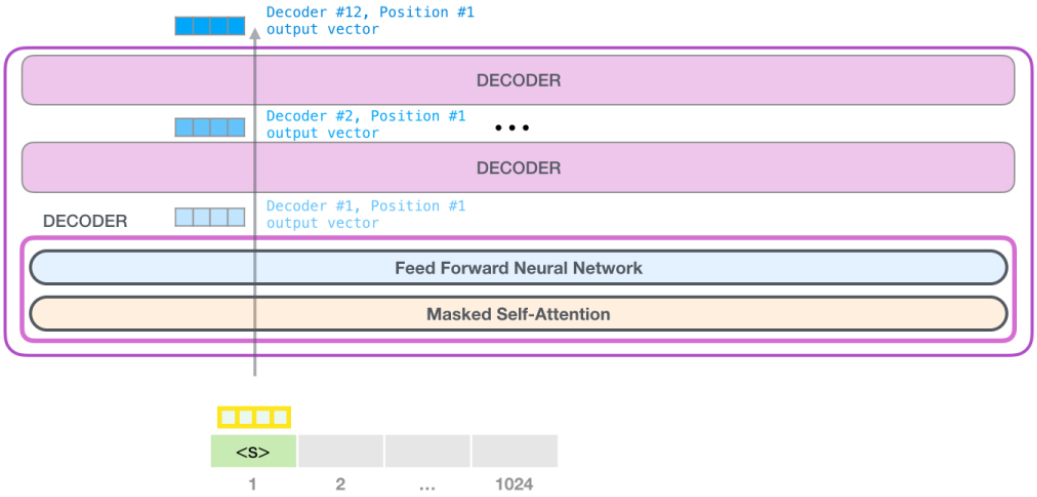
（2.3）GPT 模型

GPT2 是一个 OpenAI 组织在 2018 年在 GPT 模型的基础上发布的新的预训练模型。GPT2 模型的预训练语料库为超过 40G 的近 8000 万的网页文本数据，GPT2 的预训练语料库比 GPT 来说增大了将近 10 倍。

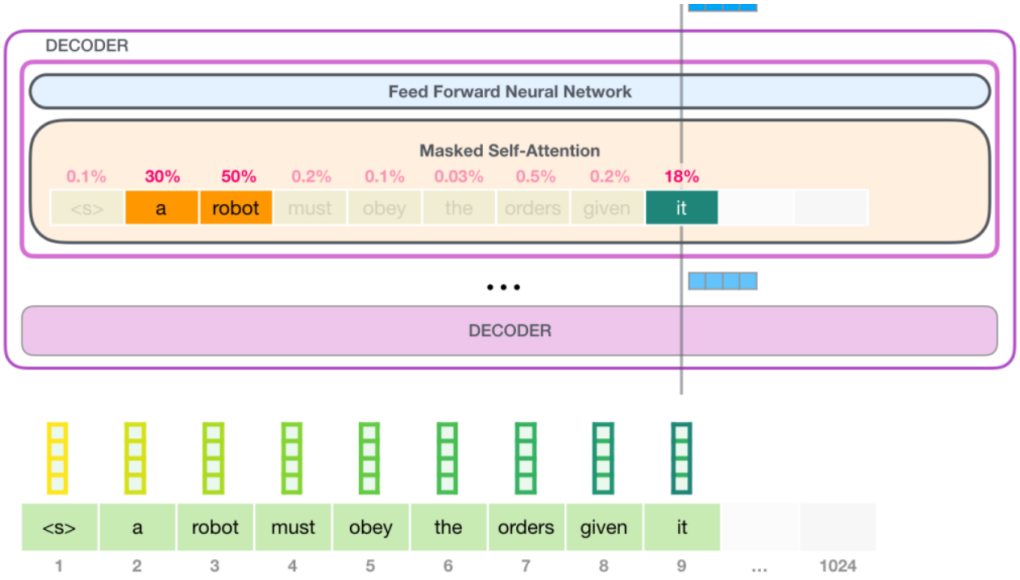


GPT 的模型由多层单向的 Transformer 的解码器部分构成，本质上是自回归模型。自回归的意思是每次产生新的单词预测以后，都会将新生成的单词添加到原输出句后面，作为新的输出句。

模型中的每个单词都要通过 Self-Attention 层，然后通过神经网络层。一旦 Transformer 的第一个模块处理了 token，会得到一个结果向量，这个结果向量会被发送到堆栈的下一个模块处理。每个模块的处理都是相同的，不过每个模块都有自己的 Self-Attention 和神经网络层。类似上面的图像的。



通过自注意层可以帮助 GPT2 在句子的理解中将代词的指代义更好的标注。它在处理某个词之前，将模型对这个词的相关词和关联词的理解融合起来（并输入到一个神经网络中）。它通过对句子片段中的每个词的关联性打分，并将这些词的表示向量加权求和。



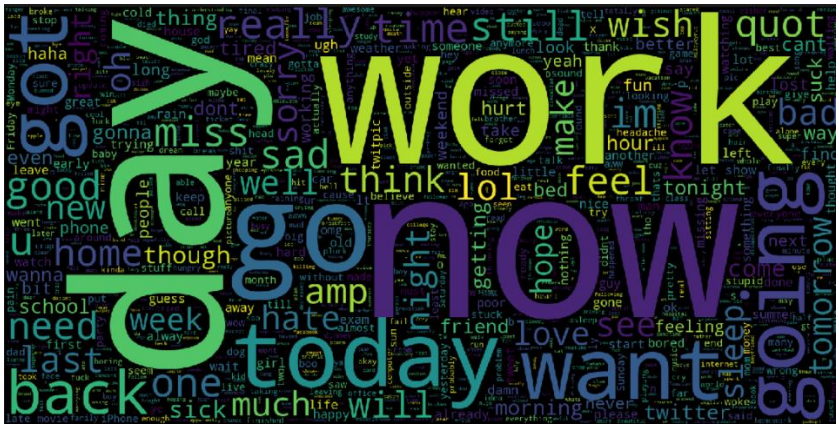
BERT 是基于双向 Transformer 结构构建，而 GPT-2 是基于单向 Transformer，这里的双向与单向，是指在进行注意力计算时，BERT 会同时考虑被遮蔽词左右的词对其的影响，而 GPT-2 只会考虑在待预测词位置左侧的词对待预测词的影响。

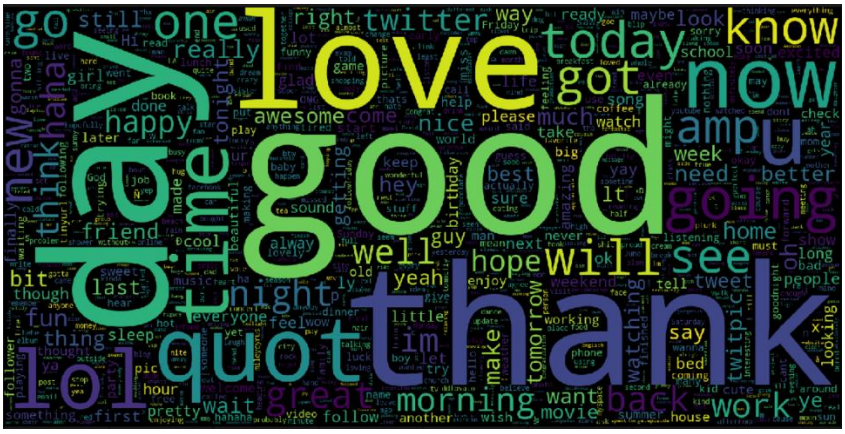
本实验中主要是使用GPT2在生成文本预测的时候会生成一个附带的Task Classifier来标示情感分类。

三、实验过程

（3.1） 机器学习

3.1.1、首先读入数据，生成正负词云





3.1.2、进行数据预处理

- 所有文本转小写
- 去掉停用词
- 去掉标点符号
- 去掉重复字符
- 删除链接和数字

3.1.3、划分数据集

训练集：测试集=8:2

3.1.4、向量化

使用 TfidfVectorizer 对训练集与测试集的单词向量化，并使用 TfidfVectorizer.transfrom 将训练集与测试集转为向量值

```
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
vectoriser.fit(x_test)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

```
No. of feature_words:  4393
```

```
C:\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarni
s deprecated in 1.0 and will be removed in 1.2. Please use get_feature_name
  warnings.warn(msg, category=FutureWarning)
```

```
X_train = vectoriser.transform(X_train)
X_val   = vectoriser.transform(X_val)
x_test  = vectoriser.transform(x_test)
```

3.1.5、训练

使用 L2 正则化、正则化系数λ的倒数 C=2、优化算法：solver=sag 训练逻辑回归模型；

使用 L2 正则化、损失函数：loss=squared_hinge、最大迭代次数：max_iter=1000 训练线性支持向量机模型；

使用 L2 正则化、损失函数：loss='log'、最大迭代次数：max_iter=1000 训练随机梯度下降模型。

3.1.6、训练集成学习模型

训练 VotingClassifier（）参数为 hard，使用第 5 步的三个模型遵从少数服从多数的规则集成学习。

```
#投票法


clf1 = LogisticRegression()
clf2 = SGDClassifier()
clf3 = LinearSVC()
# eclf1 = VotingClassifier(estimators=[('LR', clf1), ('SGDC', clf2),('LSVC', clf3)], voting='hard')
eclf1 = VotingClassifier(estimators=[('LR', clf1), ('SGDC', clf2)], voting='hard')
eclf1.fit(X_train, y_train)

# predictions_val = eclf1.predict(X_val)
# print(classification_report(y_val, predictions_val))

predictions_test = eclf1.predict(x_test)
print(classification_report(y_test, predictions_test))

```

（3.2）BRET 微调

3.2.1、数据预处理

首先读入数据并进行预处理，包括:

- 删除无用项，仅保留句子和 label 想 4 项
- 去停用词
- 标签替换，将 4 替换为 1
- 数据清洗，删除空白字符，替换链接话题用户名等
- 从中随机抽取的一定样本数，作为训练集

3.2.2、分词格式化

使用 BertTokenizer 加载预训练模型做分词，并使用 tokenizer.encode_plus 将原数据集转换为可以训练 BERT 的格式

tokenizer.encode_plus 的功能有

- 将句子拆分为标记。
- 添加[CLS]和[SEP]。
- 将令牌映射到其 ID。
- 将所有句子填充或截断为相同长度。
- 创建注意掩码，以明确区分真实令牌和[PAD]令牌

由此我们获得了训练 bert 的训练集，分为三部分：input_ids_train 输入，attention_mask_train 掩码，labels_train 标签

```
from transformers import BertTokenizer, BertForSequenceClassification, AdamW
tokenizer = BertTokenizer.from_pretrained('textattack/bert-base-uncased-SST-2', do_lower_case = True)
```

```
def three_part(text, labels):
    input_ids = []
    attention_mask = []
    for i in text:
        encoded_data = tokenizer.encode_plus(
                i,
                add_special_tokens=True,
                max_length=64,
                pad_to_max_length = True,
                return_attention_mask= True,
                return_tensors='pt')
        input_ids.append(encoded_data['input_ids'])
        attention_mask.append(encoded_data['attention_mask'])

    input_ids = torch.cat(input_ids, dim=0)
    attention_mask = torch.cat(attention_mask, dim=0)
    labels = torch.tensor(labels)
    return input_ids, attention_mask, labels

input_ids_train, attention_mask_train, labels_train=three_part(text_train, labels_train)
input_ids_test, attention_mask_test, labels_test=three_part(text_test, labels_test)
```

3.2.3、加载模型

定义 BertForSequenceClassification 模型，并加载预训练参数，该模型是在 BertModel 的基础上，添加了一个线性层 + 激活函数，用于分类。而 Huggingface 提供的预训练模型 bert-base-uncased 只包含 BertModel 的权重，不包括线性层 + 激活函数的权重。我们训练的目的，就是通过微调学习到线性层 + 激活函数的权重。

3.2.4、训练

使用 AdamW 损失函数与 get_linear_schedule_with_warmup 学习率预热方法，开始训练 10 代并记录。

```
for epoch in tqdm(range(1, epochs+1)):
    progress_bar = tqdm(train_dl, desc='Epoch {:1d}'.format(epoch), leave=False, disable=False)
    test_loss, predictions, true_tests = evaluate(test_dl)
    test_losses.append(test_loss)
    test_acc = accuracy(predictions, true_tests)
    test_acces.append(test_acc)
    tqdm.write(f'Testing loss: {test_loss}')
    tqdm.write(f'Accuracy: {test_acc}')
    model.train()
    loss_train_total = 0
    for batch in progress_bar:
        batch1=[]
        model.zero_grad()
        batch = tuple(b.to(device) for b in batch)
        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }
        outputs = model(**inputs)
        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()
        progress_bar.set_postfix({'training_loss': '{:.3f}'.format(loss.item()/len(batch))})
    batches.append(batch1)
    tqdm.write(f'\nEpoch {epoch}')
    loss_train_avg = loss_train_total/len(train_dl)
    train_losses.append(loss_train_avg)
    tqdm.write(f'Training loss: {loss_train_avg}')
```

（3.3）GPT 微调

```
# import
from torch.utils.data import Dataset, random_split
from transformers import GPT2Tokenizer, TrainingArguments, Trainer, GPT2LMHeadModel

# model
model_name = "gpt2"
# seed = 42

# # seed
# torch.manual_seed(seed)
```

3.3.1、获取 GPT2 的模型和相关训练所需函数

```
for trial_no in range(5):

    print("Loading model...")
    # load tokenizer and model
    tokenizer = GPT2Tokenizer.from_pretrained(model_name, bos_token='<|startoftext|>',
                                              eos_token='<|endoftext|>', pad_token='<|pad|>')
    model = GPT2LMHeadModel.from_pretrained(model_name).cuda()
    model.resize_token_embeddings(len(tokenizer))

    print("Loading dataset...")
    train_dataset, test_dataset = load_sentiment_dataset(tokenizer, trial_no)

    print("Start training...")
    training_args = TrainingArguments(output_dir='results', num_train_epochs=2,
                                      logging_steps=200, load_best_model_at_end=True,
                                        save_strategy="epoch", per_device_train_batch_size=2, per_device_eval_batch_size=2,
                                      warmup_steps=100, weight_decay=0.01, logging_dir='logs')

    Trainer(model=model, args=training_args, train_dataset=train_dataset,
            eval_dataset=test_dataset, data_collator=lambda data: {'input_ids': torch.stack([f[0] for f in data]),
                                                                   'attention_mask': torch.stack([f[1] for f in data]),
                                                                   'labels': torch.stack([f[0] for f in data])}).train()
```

3.3.2、训练模型

```
# test
print("Start testing...")
# eval mode on model
_ = model.eval()

# compute prediction on test data
original, predicted, all_text, predicted_text = [], [], [], []
map_label = {0:'negative', 4: 'positive'}
for text, label in tqdm(zip(test_dataset[0], test_dataset[1])):
    # predict sentiment on test data
    prompt = f'<|startoftext|>Review: {text}\nSentiment:'
    generated = tokenizer(f"<|startoftext|> {prompt}", return_tensors="pt").input_ids.cuda()
    sample_outputs = model.generate(generated, do_sample=False, top_k=50, max_length=512, top_p=0.90,
            temperature=0, num_return_sequences=0)
    pred_text = tokenizer.decode(sample_outputs[0], skip_special_tokens=True)
    # extract the predicted sentiment
    try:
        pred_sentiment = re.findall("\nSentiment: (.*)", pred_text)[-1]
    except:
        pred_sentiment = "None"
    original.append(map_label[label])
    predicted.append(pred_sentiment)
    all_text.append(text)
    predicted_text.append(pred_text)
#transform into dataframe
df = pd.DataFrame({'text': all_text, 'predicted': predicted, 'original': original, 'predicted_text': predicted_text})
df.to_csv(f"result_run_{trial_no}.csv", index=False)
# compute f1 score
acc=f1_score(original, predicted, average='macro')
test_acces.append(acc)
print(acc)
print('-------------------------------------------------',trial_no,'-------------------------------------------------------------------------------')
```

3.3.3、测试

对测试集进行一次测试，并计算统计正确率。

本实验中采用了 GPT2 的 Tokenizer 和模型，将输入划分为测试集和验证集导入模型，并在测试集上对模型进行测试。最后输出对应的成功率，并在 5 趟迭代的过程中看模型收敛的速度以及效果。整体代码见代码部分。

参数设置如下：

```
print("Loading model...")
# load tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained(model_name, bos_token='<|startoftext|>',
                                          eos_token='<|endoftext|>', pad_token='<|pad|>')
model = GPT2LMHeadModel.from_pretrained(model_name).cuda()
model.resize_token_embeddings(len(tokenizer))

print("Loading dataset...")
train_dataset, test_dataset = load_sentiment_dataset(tokenizer, trial_no)

print("Start training...")
training_args = TrainingArguments(output_dir='results', num_train_epochs=2,
                                  logging_steps=200, load_best_model_at_end=True,
                                    save_strategy="epoch", per_device_train_batch_size=2, per_device_eval_batch_size=2,
                                  warmup_steps=100, weight_decay=0.01, logging_dir='logs')

Trainer(model=model, args=training_args, train_dataset=train_dataset,
        eval_dataset=test_dataset, data_collator=lambda data: {'input_ids': torch.stack([f[0] for f in data]),
                                                               'attention_mask': torch.stack([f[1] for f in data]),
                                                               'labels': torch.stack([f[0] for f in data])}).train()
```

四、实验结果

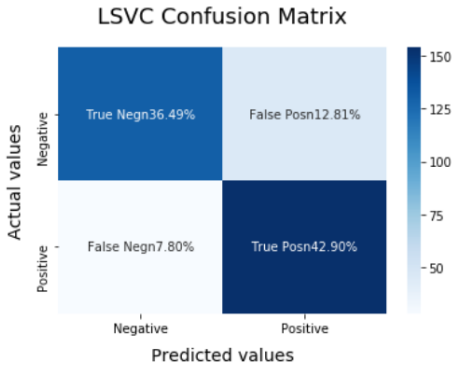
（4.1）机器学习

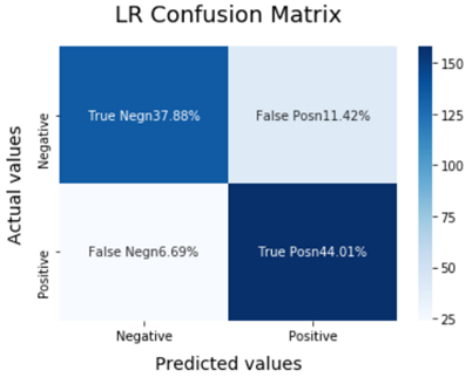
4.1.1、各混淆矩阵

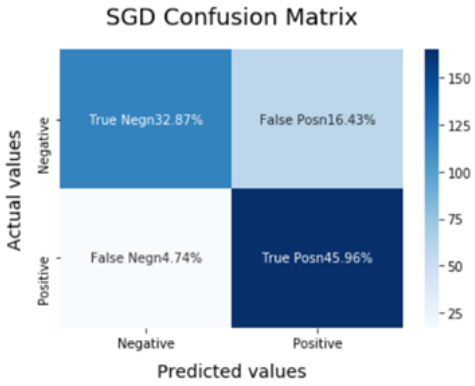
	precision	recall	f1-score	support
0	0.82	0.74	0.78	177
1	0.77	0.85	0.81	182
accuracy			0.79	359
macro avg	0.80	0.79	0.79	359
weighted avg	0.80	0.79	0.79	359

	precision	recall	f1-score	support
0	0.85	0.77	0.81	177
1	0.79	0.87	0.83	182
accuracy			0.82	359
macro avg	0.82	0.82	0.82	359
weighted avg	0.82	0.82	0.82	359

	precision	recall	f1-score	support
0	0.87	0.67	0.76	177
1	0.74	0.91	0.81	182
accuracy			0.79	359
macro avg	0.81	0.79	0.78	359
weighted avg	0.80	0.79	0.79	359



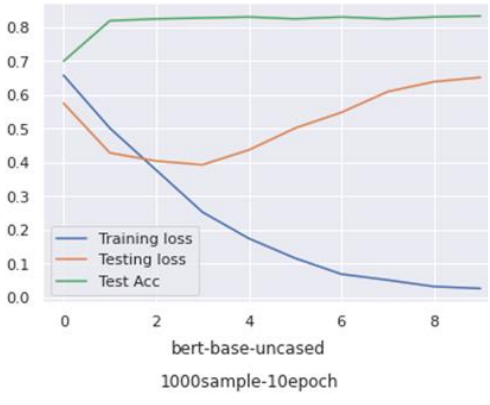


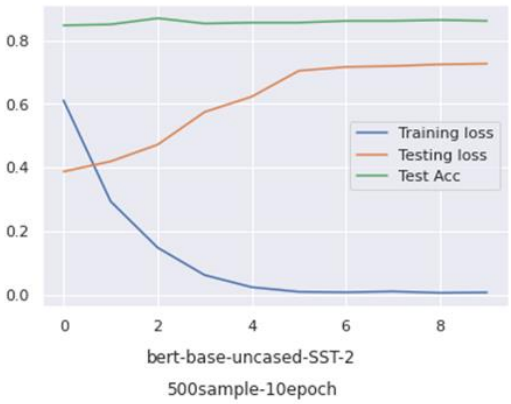


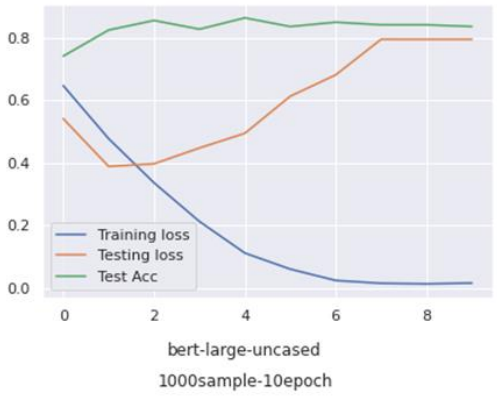
4.1.2、各项指标

	LSVC	LR	SGD	Voting
Precision	0.80	0.82	0.80	0.83
Recall	0.79	0.82	0.79	0.83
F1_score	0.79	0.82	0.79	0.83
Accuracy	0.79	0.82	0.79	0.83

（4.2）BERT 微调







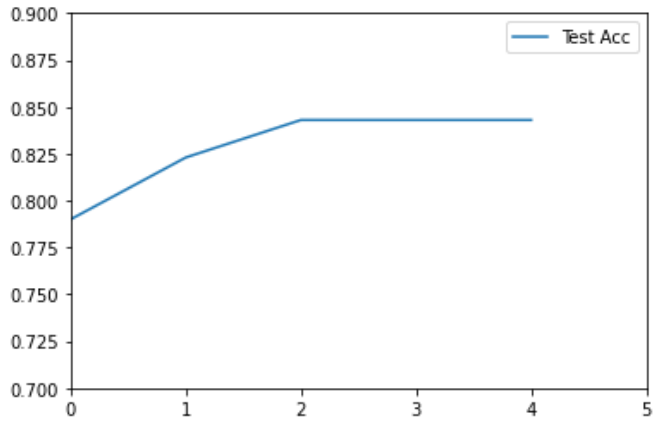
模型	Bert-base-uncased	Bert-base-uncased-sst-2	Bert-large-uncased
accuracy	0.832	0.8635	0.869

（4.3）GPT 微调

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
14it [00:00, 31.54it/s]Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
18it [00:00, 28.09it/s]Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
21it [00:00, 26.28it/s]Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
24it [00:00, 25.35it/s]Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
27it [00:01, 25.00it/s]Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
...
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
357it [00:13, 20.54it/s]Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
359it [00:14, 25.34it/s]

0.843980877879183
-------------------------------------------------- 3 --------------------------------------------------
```

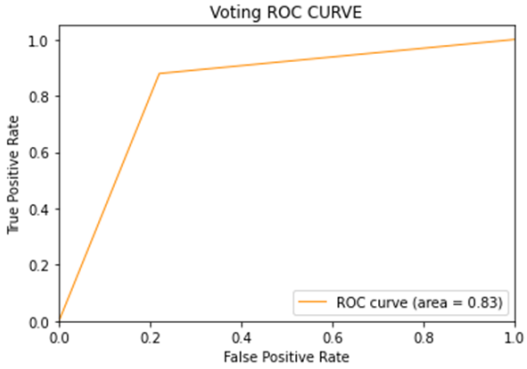
实验完美完成，经过几轮的迭代训练，最后的正确率稳定在了 84.3%。将多次训练的正确率画成一张表则为下图。



五、实验探究

（5.1）机器学习

我们发现集成学习的各项指标比其他三个模型的效果都要好，说明三个模型之间存在互补的关系，通过集成学习提高了准确率集，并且对于时间的影响不大

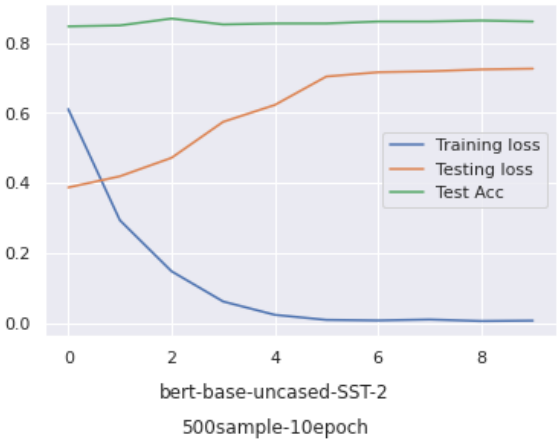


	precision	recall	f1-score	support
0	0.86	0.78	0.82	177
1	0.80	0.88	0.84	182
accuracy			0.83	359
macro avg	0.83	0.83	0.83	359
weighted avg	0.83	0.83	0.83	359

（5.2）BERT 微调

BERT 的基础建立在 transformer 之上，拥有强大的语言表征能力和特征提取能力，而且适配文本分类与情感分析任务，理论上只需要进行微调就可以有很好的效果。

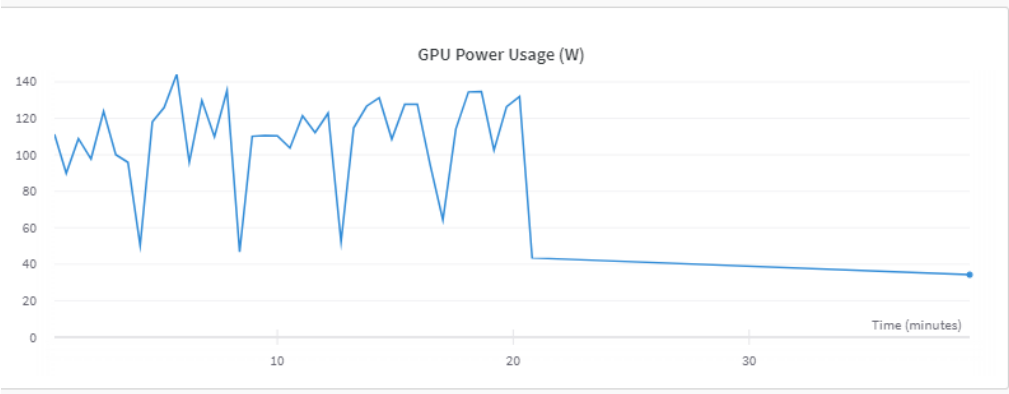
训练结果说明 BERT 微调的确具有很好的效果，在只使用 500 或 1000 条数据训练后，BERT 就可以达到最高 0.869 的准确率。

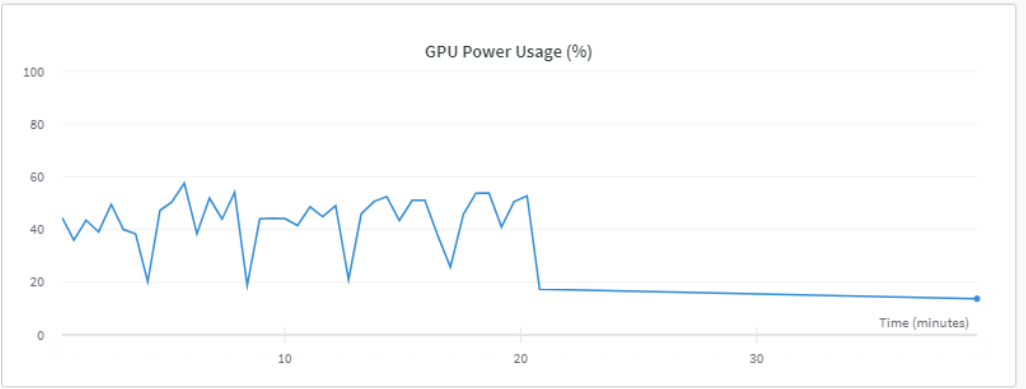


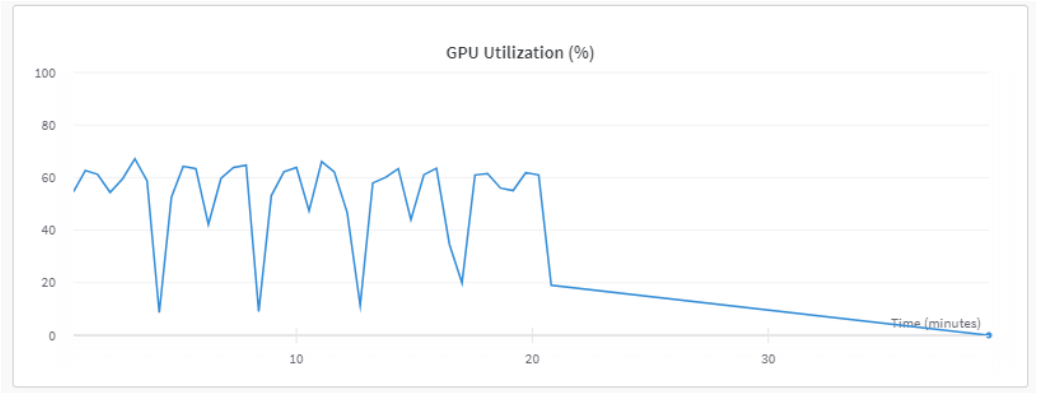
（5.3）GPT 微调

由于 GPT2 模型已经非常成熟，基础性的参数调整已经在传统模型上进行过调整试验，在 GPT2 上，小组进行了对性能的关注和查看，在训练的过程中记录了 GPU 的性能变换，并将其转化成了对应的图表形式。

小组成员探讨了图表的表征趋势为何会如此。得出的结论是几次使用率非常低的间隔都是一整次训练并测试完毕的时间，中间输出各类信息并计算准确率，并有准备和读取模型的过程。






GPU Utilization (%)
100
80
60
40
20
0
10
20
30
Time (minutes)