《机器学习》

大作业

人工智能与计算机学院 学院 人工智能 专业

学号	姓名
6243112053	王玺

2024 年 11 月

一、实验目的

本次大作业的目的是利用所学机器学习相关知识实现垃圾邮件的分类任务。锻炼实际代码能力，加深课程内容了解，为之后本专业相关知识的学习打好基础。

二、实验内容

本次大作业数据集包括 5567 个垃圾邮件样本。其中 4567 个作为训练样本，1000 个作为测试样本，每个样本包括 2 个特征。样本类别不均衡。要求每位同学根据提供的数据，设计机器学习模型，自动检测垃圾邮件。

通过本次大作业，要求学生掌握数据预处理、数据建模、数据分析等完整的数据分析过程，并形成大作业报告，进行展示分享。设计者可以采用任何分类算法及数据预处理方法。

本次实验主要采用 Bert 模型，深度学习框架为 pytorch。

三、运行环境

操作系统：windows 11
CPU: 13th Gen Intel(R) Core(TM) i5-13600KF 3.50 GHz
GPU: NVIDIA GeForce RTX 4060 Ti
CUDA 版本: CUDA 12.6.65
编程语言：python3.8
代码编辑器：VS Code、Pycharm
相关库及版本：matplotlib 3.7.3 numpy 1.24.4
Pandas 2.0.3 pytorch 2.4.1
Seaborn 0.13.2 scikit-learn 1.3.2

四、数据预处理

垃圾邮件分类数据集

数据集包括以下数据文件:

(1) 训练集（mail_data_train.csv），包括 3 列，其中第 1 列 ID，即样本编号，第 2 列 Category 是垃圾邮件类别，0 表示不是垃圾邮件，1 表示垃圾邮件，第 3 列为邮件文本内容。

(2) 测试集（mail_data_test_nolable.csv），包括 2 列，其中第 1 列 ID，即样本编号，第 2 列为邮件文本内容。测试集内没有 Category。

数据读取和处理

```
# 加载训练和测试数据
def load_data():
    # 从 CSV 文件读取训练集和测试集数据
    train_data = pd.read_csv('mail_data_train.csv', encoding='ISO-8859-1')
    test_data = pd.read_csv('mail_data_test_nolable.csv',
```

```
encoding='ISO-8859-1')
        return train_data, test_data
```

训练集前 5 行

	id	Category	Message	Unnamed: 3
0	1	0	Hi Chikku, send some nice msgs	NaN
1	2	0	I met you as a stranger and choose you as my f...	NaN
2	3	1	Moby Pub Quiz.Win a Â£100 High Street prize if...	NaN
3	4	0	Nice line said by a broken heart- Plz don't cu...	NaN
4	5	0	Arms fine, how's Cardiff and uni?	NaN

测试集前 5 行

	id	Message
0	1	HMV BONUS SPECIAL 500 pounds of genuine HMV vo...
1	2	1Apple/Day=No Doctor. 1Tulsi Leaf/Day=No Cance...
2	3	He is impossible to argue with and he always t...
3	4	Send me the new number
4	5	Urgent! Please call 09061743811 from landline....

```
# 预处理邮件内容，去除非字母数字字符
    def preprocess_content(content):
    # 仅保留字母数字的单词并通过空格连接
        return ' '.join(word for word in content.split() if word.isalnum())
# 将训练数据拆分为训练集和验证集
        from sklearn.model_selection import train_test_split
        train_data, val_data = train_test_split(train_data, test_size=0.1)

    # 对训练、验证和测试数据的邮件内容进行预处理
train_data['Processed_Message'] = train_data['Message'].apply(preprocess_content)
test_data['Processed_Message'] = test_data['Message'].apply(preprocess_content)
val_data['Processed_Message'] =    val_data['Message'].apply(preprocess_content)
```

五、特征提取与数据建模

```
    # 自定义数据集类
    class EmailDataset(torch.utils.data.Dataset):
        # 初始化函数，接收编码后的数据并设置标签
        def __init__(self, encodings):
            self.encodings = encodings    # 输入特征
            self.labels = encodings.pop('labels', None)    # 标签（如果有的话）

        # 获取特定索引的数据项
        def __getitem__(self, idx):
            # 将编码后的特征按索引取出，组成一个字典
            item = {key: val[idx] for key, val in self.encodings.items()}
```

```
            if self.labels is not None:
                item['labels'] = self.labels[idx]  # 如果有标签，将其添加到数据项中
            return item

        # 返回数据集的长度
        def __len__(self):
            return len(self.encodings['input_ids'])  # 数据长度基于 input_ids 特征
```

六、模型训练与评估

使用 Bert 模型对数据进行训练，并评估其在验证集上的性能。

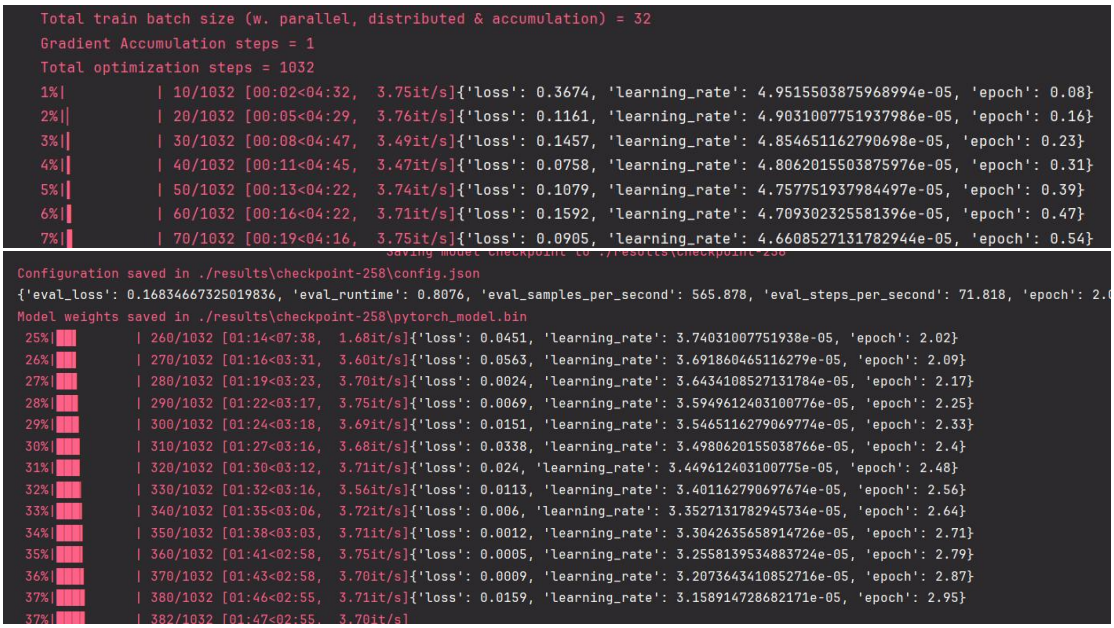
```
# 使用本地保存的 BERT 模型和分词器
    local_model_path = 'Bert_uncased'  # 本地模型文件夹路径
        tokenizer = BertTokenizer.from_pretrained(local_model_path)
        model = BertForSequenceClassification.from_pretrained(local_model_path, num_labels=len(train_data['Category'].unique()))

# 编码训练数据并创建 EmailDataset 数据集
    train_encodings = encode_data(tokenizer, train_data['Processed_Message'].tolist(), train_data['Category'].tolist())
    train_dataset = EmailDataset(train_encodings)

# 编码验证数据并创建 EmailDataset 数据集
    val_encodings = encode_data(tokenizer, val_data['Processed_Message'].tolist(), val_data['Category'].tolist())
    val_dataset = EmailDataset(val_encodings)

  # 定义训练参数
        training_args = TrainingArguments(
            output_dir='./results',  # 模型保存路径
            num_train_epochs=20,  # 设置训练周期数
            per_device_train_batch_size=64,  # 每个设备的批量大小
            learning_rate=1e-4,  # 设置学习率
            logging_dir='./logs',  # 日志保存路径
            logging_steps=10,  # 每隔 10 步记录一次日志
            evaluation_strategy='epoch',  # 在每个 epoch 结束时进行验证
            save_strategy='epoch',  # 在每个 epoch 结束时保存模型
            load_best_model_at_end=True,  # 在训练结束后加载最佳模型
```

```
        eval_steps=100,  # 每 100 步进行一次验证
        metric_for_best_model="eval_loss",  # 使用验证损失作为早停的评估标准
        greater_is_better=False,  # 评估标准为越小越好
    )
```

经过多次尝试得到的核心参数和训练策略：

Learning rate：0.0001
Batch_size: 64
logging_steps=10, # 每隔 10 步记录一次日志
num_train_epochs=20, # 设置训练周期数
metric_for_best_model="eval_loss", # 使用验证损失作为早停的评估标准
greater_is_better=False, # 评估标准为越小越好
callbacks=[EarlyStoppingCallback(early_stopping_patience=3)] # 早停策略：若 3 个周期内无提升则停止训练

训练过程记录



```
Training completed. Do not forget to share your model on huggingface.co/models =)


Loading best model from ./results\checkpoint-65 (score: 0.06988421827554703).
C:\Users\89556\.conda\envs\DL\lib\site-packages\transformers\trainer.py:1535: FutureWa
  state_dict = torch.load(best_model_path, map_location="cpu")
{'train_runtime': 438.5808, 'train_samples_per_second': 93.711, 'train_steps_per_secon
 40%|████      | 260/650 [07:18<10:57,  1.69s/it]
***** Running Prediction *****
  Num examples = 1000
  Batch size = 8
100%|██████████| 125/125 [00:02<00:00, 51.20it/s]***** Running Prediction *****
预测结果已保存到 sample_submission.csv
  Num examples = 4110
  Batch size = 8
639it [00:20, 30.52it/s]
```

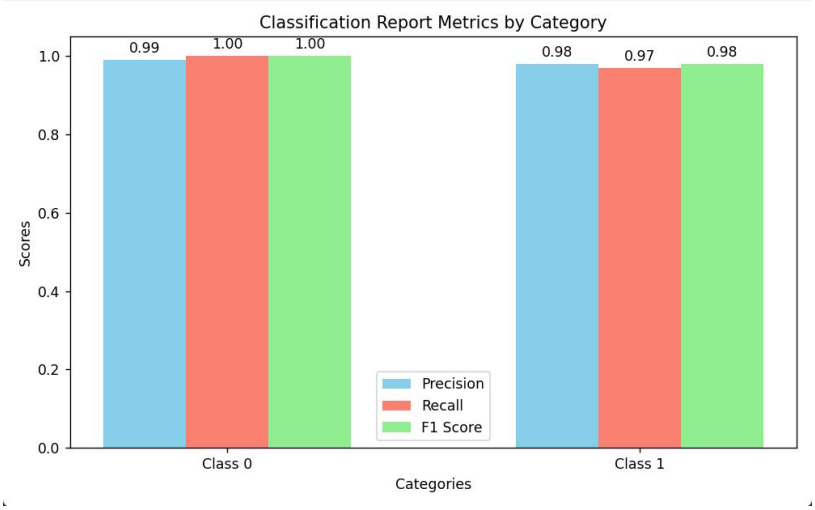
七、实验结果与分析

训练结束控制台输出结果

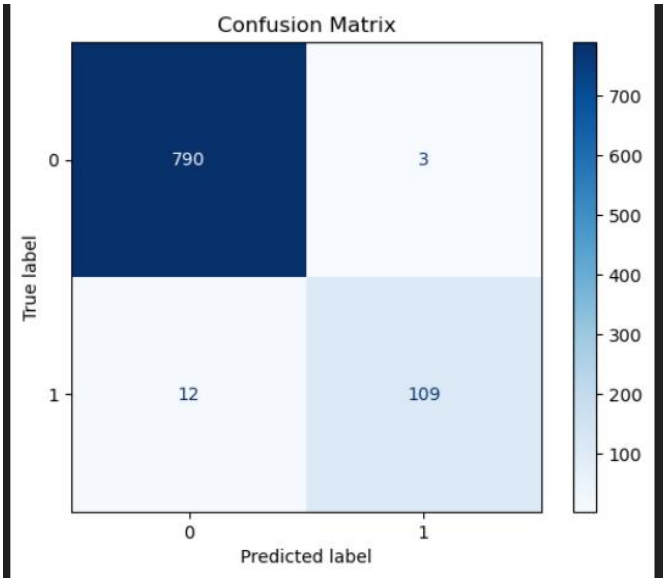
	precision	recall	f1-score	support
0	0.99	1.00	1.00	3561
1	0.98	0.96	0.97	549
accuracy			0.99	4110
macro avg	0.99	0.98	0.98	4110
weighted avg	0.99	0.99	0.99	4110

根据结果绘制图像

```
# 设置柱状图的位置
x = np.arange(len(categories))
width = 0.2  # 每个柱状图的宽度

fig, ax = plt.subplots(figsize=(8, 5))
bar1 = ax.bar(x - width, precision, width, label='Precision', color='skyblue')
bar2 = ax.bar(x, recall, width, label='Recall', color='salmon')
bar3 = ax.bar(x + width, f1_score, width, label='F1 Score', color='lightgreen')

# 添加标签和标题
ax.set_xlabel('Categories')
ax.set_ylabel('Scores')
ax.set_title('Classification Report Metrics by Category')
ax.set_xticks(x)
ax.set_xticklabels(categories)
```



混淆矩阵



kaggle 平台分数　　0.975609

八、心得体会

在完成这个大作业的过程中，我深入探索了自然语言处理（NLP）中的文本分类任务，使用了多种技术和工具来构建一个邮件分类模型。以下是我的一些心得体会：

在实验的开始阶段，数据预处理是至关重要的。原始数据往往包含噪声和无关信息，这可能会影响模型的性能。通过去除无效字符、进行分词和处理缺失值，我发现经过合理的预处理后，模型的训练效果显著提升。这让我深刻认识到，数据是机器学习的基础，数据质量直接影响到模型的表现。

选择适合任务的模型也是成功的关键。在本次实验中，我使用了 BERT 模型进行文本分类。BERT 在处理自然语言任务时表现出色，它的上下文理解能力使得模型能有效捕捉文本中的潜在含义。通过使用预训练的模型，我能更快地收敛并获得较高的准确率。

在模型训练过程中，我尝试了多种超参数组合，如学习率、批量大小和训练周期等。通过调整这些参数，我观察到不同设置对模型性能的影响。例如，增加训练周期有助于提高模型准确率，但同时也可能导致过拟合。因此，找到合适的超参数组合需要反复实验和仔细观察模型的表现。

引入早停策略是我在本次实验中的一个重要尝试。在经过数个训练周期后，发现验证集的损失不再显著降低，通过早停策略，我能够有效地防止过拟合，提高模型的泛化能力。这一策略让我意识到，训练的持续进行并不总是带来更好的结果，适时的停止是非常必要的。

在训练完成后，我使用分类报告对模型的表现进行了评估，重点关注了精确率、召回率和 F1-score。这些指标提供了对模型表现的全面理解，尤其是在类别不平衡的情况下，我意识到宏平均和加权平均的重要性。这些反馈使我对模型的性能有了更深入的认识，同时也为后续的改进指明了方向。

这次实验让我明白了机器学习的过程是不断迭代的。每一次的实验都是在前一次基础上的学习。随着对模型理解的加深和经验的积累，我相信在未来的项目中能够更加游刃有余。

总体而言，这次实验让我在实践中加深了对机器学习，特别是自然语言处理的理解。每一个环节都有其独特的重要性，合理的设计和调整将直接影响最终结果。未来我将继续探索更复杂的模型和更高效的数据处理技术，以应对更多的实际应用场景。