Machine Learning Task #2: Text_Classification 实验报告

徐浩淞**20226446**
1593120349@qq.com

摘要

大作业二的任务的主要内容是进行意见挖掘（在本任务中是情感标签分类），运用到斯坦福情感树库数据集，将其中标签为1的句子作为训练集，标签为2的句子作为测试集，对于训练集中的每个句子划分其情感分类标签，并运用cnn、rnn、lstm、transfomer等模型进行训练，预测测试集，并评估正确率。

关键词： Opinion Mining ； Sentiment Classification ； Model Evaluation

XuHaosong
1593120349@qq.com

Abstract

The main task of Assignment 2 is to conduct opinion mining (in this case, sentiment label classification) using the Stanford Sentiment Tree dataset. Sentences with label 1 are used as the training set, while sentences with label 2 are used as the testing set. Each sentence in the training set is assigned an sentiment classification label, and models such as CNN, RNN, LSTM, and Transformer are used for training, predicting the testing set, and evaluating accuracy.

Keywords: Opinion Mining , Sentiment Classification , Model Evaluation

1 实验内容与具体要求

本任务的主要内容是进行意见挖掘（ 在本任务中是情感标签分类） ， 运用到斯坦福情感树库数据集， 将数据集中标签为1的句子作为训练集， 标签为2的句子作为测试集， 对于训练集中的每个句子划分其情感分类标签。 其中情感标签具体是5分制， 用1、 2、 3、 4、 5分别表示从不积极到非常积极的情感。 对于每个训练集与测试集中的语句， 需要将其作为一个整体获取索引，并根据语句索引在给出的词典中查找其对应的情感标签索引， 再根据情感标签索引， 在情感标签词典中获取语句的情感标签。 并运用cnn、 rnn、 lstm、 transfomer等模型进行训练， 预测测试集中的语句的情感标签，根据正确率评估性能。

2 实验过程

2.1 数据处理

使用pandas库获取数据，包括句子内容、编号，词典序号，情感标签序号等，将这些内容使用pandas.merge的方法把有效内容合并，便于处理句子和它对应的情感标签并根据split 中的序号划分训练集、 测试集、 验证集。

```python
main_path = os.path.dirname(os.path.abspath(__file__))
output_path = os.path.join(main_path, 'output')
os.makedirs(output_path, exist_ok=True)

def fetch_data():
    #加载数据集
    sentences = pd.read_csv(os.path.join(main_path, 'datasetSentences.txt'), sep='\t', encoding='utf-8', names=['index', 'text'], skiprows=1)
    splits = pd.read_csv(os.path.join(main_path, 'datasetSplit.txt'), encoding='utf-8', names=['index', 'set_label'], skiprows=1)
    dictionary = pd.read_csv(os.path.join(main_path, 'dictionary.txt'), sep='|', encoding='utf-8', names=['phrase', 'id'])
    sentiments = pd.read_csv(os.path.join(main_path, 'sentiment_labels.txt'), sep='|', encoding='utf-8', names=['id', 'value'], skiprows=1)

    sentiments['class'] = sentiments['value'].apply(lambda x: min(int(x * 5), 4))

    merged_data = pd.merge(sentences, splits, on='index')#处理四个txt文本的信息，合并有效内容
    merged_data = pd.merge(merged_data, dictionary, left_on='text', right_on='phrase')
    merged_data = pd.merge(merged_data, sentiments[['id', 'class']], on='id')

    train_data = merged_data[merged_data['set_label'] == 1]#针对句子编号划分数据集
    val_data = merged_data[merged_data['set_label'] == 3]
    test_data = merged_data[merged_data['set_label'] == 2]

    return train_data, val_data, test_data
```

Figure 1: 数据处理

设计一个数据类，用于对划分好的训练集、测试集、验证集的数据进行处理，获取句子id和对应的情感标签，使用berttokenizer把句子转换为输入模型的格式，便于使用dataloader加载数据。

```python
class TextDataset(Dataset):
    def __init__(self, dataset, tokenizer, max_length=128):
        self.dataset = dataset
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, index):#获取文本编号以及情感标签
        text = self.dataset.iloc[index]['text']
        label = self.dataset.iloc[index]['class']
        encoded = self.tokenizer(text, truncation=True, padding='max_length', max_length=self.max_length, return_tensors='pt')#对文本进行编码

        return {
            'input_ids': encoded['input_ids'].flatten(),
            'attention_mask': encoded['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }
```

Figure 2: pytorch数据接口

2.2 CNN模型

包含词嵌入层、 卷积层、 池化层、 全连接层和用于防止过拟合的dropout层。

```python
class CNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_filters, filter_sizes, num_classes, dropout_rate, padding_idx):
        # 结构：词嵌入层、卷积层、全连接层、池化层和dropout层
        super(CNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=padding_idx)
        self.convolutions = nn.ModuleList([nn.Conv2d(1, num_filters, (fs, embed_dim)) for fs in filter_sizes])
        self.fc = nn.Linear(len(filter_sizes) * num_filters, num_classes)
        self.dropout = nn.Dropout(dropout_rate)

    def forward(self, input_ids, attention_mask):
        embedded = self.embedding(input_ids).unsqueeze(1)
        conv_results = [F.relu(conv(embedded)).squeeze(3) for conv in self.convolutions]
        pooled_results = [F.max_pool1d(conv, conv.shape[2]).squeeze(2) for conv in conv_results]
        concatenated = self.dropout(torch.cat(pooled_results, dim=1))
        return self.fc(concatenated)
```

Figure 3: CNN模型

2.3 RNN模型

包含词嵌入层、 lstm层、 全连接层和dropout层。

```python
class RNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim, num_classes, dropout_rate, padding_idx):
        #结构：词嵌入层、LSTM层、全连接层和dropout层
        super(RNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=padding_idx)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.fc = nn.Linear(hidden_dim * 2, num_classes)
        self.dropout = nn.Dropout(dropout_rate)

    def forward(self, input_ids, attention_mask):
        embedded = self.embedding(input_ids)
        packed_output, (hidden, cell) = self.lstm(embedded)
        hidden = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)
        hidden = self.dropout(hidden)
        return self.fc(hidden)
```

Figure 4: RNN模型

2.4 Transfomer模型

包含词嵌入层、 transformer编码器、 全连接层和dropout层。

```python
class TransformerModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_classes, nhead, num_layers, dropout_rate, padding_idx):
        #结构：词嵌入层、transformer编码器、全连接层和dropout层
        super(Transforme  __init__()
        self.embedding =  (module) nn  vocab_size, embed_dim, padding_idx=padding_idx)
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=embed_dim,
            nhead=nhead,
            dim_feedforward=embed_dim * 4,
            dropout=dropout_rate,
            batch_first=True
        )
        self.transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.fc = nn.Linear(embed_dim, num_classes)
        self.dropout = nn.Dropout(dropout_rate)

    def forward(self, input_ids, attention_mask):
        mask = attention_mask.float().masked_fill(attention_mask == 0, float('-inf')).masked_fill(attention_mask == 1, float(0.0))

        embedded = self.embedding(input_ids)
        transformer_out = self.transformer_encoder(embedded, src_key_padding_mask=~attention_mask.bool())
        pooled_output = transformer_out[:, 0]
        pooled_output = self.dropout(pooled_output)
        return self.fc(pooled_output)
```

Figure 5: Transfomer模型

2.5 训练模型并验证

训练模型，计算损失，并反向传播更新参数，然后在不计算梯度的情况下预测验证集的情感标签分类结果，计算验证损失。

```python
def train_model(model, train_loader, val_loader, loss_fn, optimizer, epochs, device):
    best_loss = float('inf')#初始化最佳损失

    for epoch in range(epochs):
        model.train()#训练
        total_train_loss = 0
        for batch in train_loader:
            optimizer.zero_grad()

            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            predictions = model(input_ids, attention_mask)
            loss = loss_fn(predictions, labels)

            loss.backward()
            optimizer.step()
            total_train_loss += loss.item()

        model.eval()#验证
        total_val_loss = 0
        with torch.no_grad():
            for batch in val_loader:
                input_ids = batch['input_ids'].to(device)
                attention_mask = batch['attention_mask'].to(device)
                labels = batch['labels'].to(device)

                predictions = model(input_ids, attention_mask)
                loss = loss_fn(predictions, labels)
                total_val_loss += loss.item()

        print(f'轮数: {epoch+1}: 训练损失: {total_train_loss/len(train_loader):.3f}, 验证损失: {total_val_loss/len(val_loader):.3f}')

        if total_val_loss < best_loss:
            best_loss = total_val_loss
            torch.save(model.state_dict(), os.path.join(output_path, f'model_{model.__class__.__name__}.pt'))
```

Figure 6: 训练模型并验证

2.6 测试模型并评估正确率的函数

禁用梯度计算，加速测试，使用dataloader加载数据，对比预测结果和真实标签，使用accuracy评估正确率。

```python
def test_model(model, test_loader, device):
    model.eval()
    preds = []
    true_labels = []

    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask)#预测
            _, predicted = torch.max(outputs, 1)#获取预测结果

            preds.extend(predicted.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())#收集预测结果和真实标签

    return accuracy_score(true_labels, preds)

def test():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    train_data, val_data, test_data = fetch_data()
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')#分词器

    test_dataset = TextDataset(test_data, tokenizer)#处理数据
    test_loader = DataLoader(test_dataset, batch_size=8)#加载数据
```

Figure 7: 测试模型并评估正确率的函数

2.7 测试模型并输出正确率

设置模型的超参数， 初始化模型到字典里， 用来一次性评估三个模型。 使用torchload加载训练好的模型， 然后调用testmodel进行测试并输出正确率。

```python
vocab_size = tokenizer.vocab_size
embed_dim = 256
num_filters = 100
filter_sizes = [2, 3, 4]
num_classes = 5
dropout_rate = 0.2
padding_idx = tokenizer.pad_token_id
hidden_dim = 256
nhead = 8
num_layers = 2

models = {#初始化模型
    'CNN': CNN(vocab_size, embed_dim, num_filters, filter_sizes, num_classes, dropout_rate, padding_idx),
    'RNN': RNN(vocab_size, embed_dim, hidden_dim, num_classes, dropout_rate, padding_idx),
    'Transformer': TransformerModel(vocab_size, embed_dim, num_classes, nhead, num_layers, dropout_rate, padding_idx)
}

for model_name, model in models.items():
    model_path = os.path.join(output_path, f'model_{model.__class__.__name__}.pt')
    if os.path.exists(model_path):
        model = model.to(device)
        model.load_state_dict(torch.load(model_path))
        accuracy = test_model(model, test_loader, device)
        print(f"\n{model_name}模型准确率: {accuracy:.4f}")
```

Figure 8: 测试模型并输出正确率

3 实验结果

3.1 训练结果

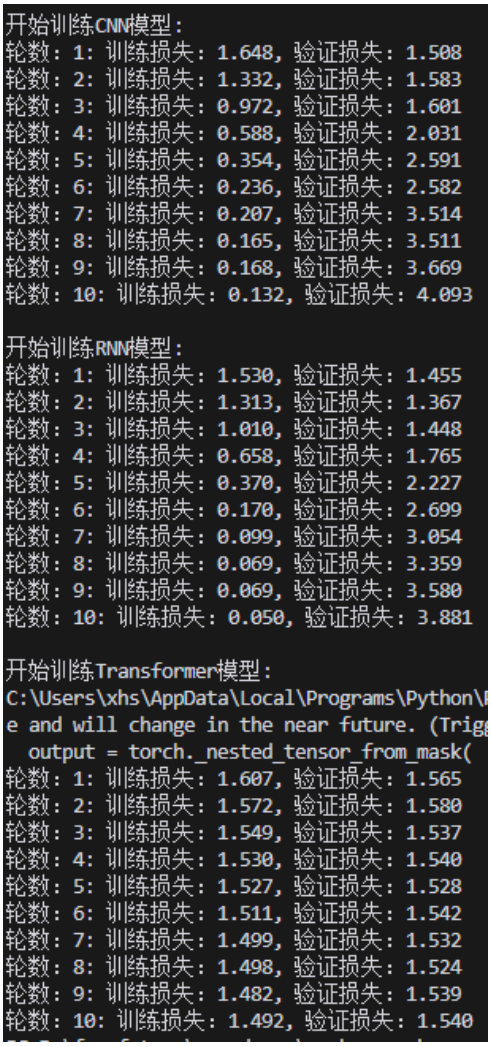

Figure 9: 训练结果

3.2 测试结果

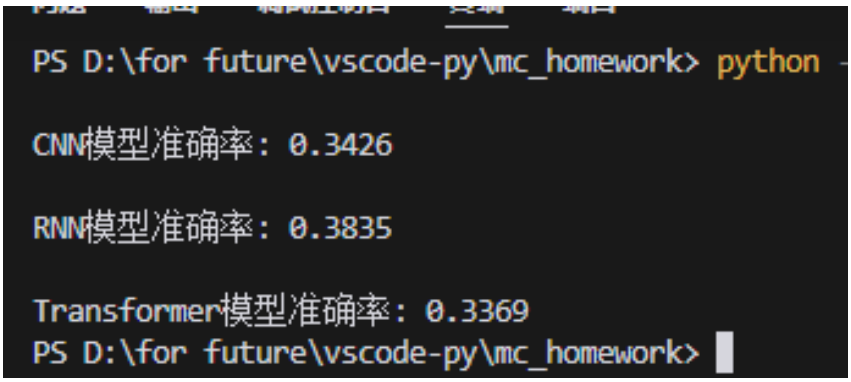

Figure 10: 测试结果

4 实验心得

通过本次作业，我学习了使用...的方法训练cnn、rnn、transfomer模型来实现对文本处理，将文本的情感标签用于训练模型，并使用这三个模型预测语句的情感标签。在这几个模型中，rnn的正确率最好但也低于百分之四十，可能是训练轮数太少，一些关键的参数等不会设置调优，尤其是模型调整后的训练时间成本也很高，有时候还会导致过拟合。同时我还对三个模型的区别有了更细致的理解，其中transfomer更适合处理复杂的情况，但是参数调整不好导致还没有rnn准确率高。