项目报告

项目背景

随着互联网内容的快速增长，新闻文本的数量呈现爆炸式上升。如何对海量新闻内容进行自动化分类，成为信息检索、内容推荐和舆情分析等领域中的重要问题。

新闻主题分类任务旨在根据新闻文本内容，自动判断其所属的主题类别，例如财经、体育、科技、娱乐等，是自然语言处理（NLP）中的经典文本分类问题。

本项目基于真实新闻数据集，围绕“新闻主题自动分类”这一实际应用场景，设计并实现了一个基于 **深度学习的中文文本分类系统**。通过对新闻文本进行分词、数值化表示，并利用循环神经网络（LSTM）对文本序列进行建模，最终实现对新闻主题的自动预测。

在项目实现过程中，重点关注了以下几个问题：

1. **中文文本的预处理方式**，包括分词、词表构建与文本长度控制；
2. **训练集、验证集和测试集的合理划分与使用**，避免数据泄漏；
3. **模型结构与超参数的选择**，在训练效率与分类性能之间取得平衡；
4. **模型训练、验证与测试流程的工程化实现**，保证实验结果可复现。

通过本项目的实现，加深了对自然语言处理基本流程和深度学习文本分类模型的理解，同时也提升了将算法模型落地为完整工程项目的能力。

项目结构

项目文件结构

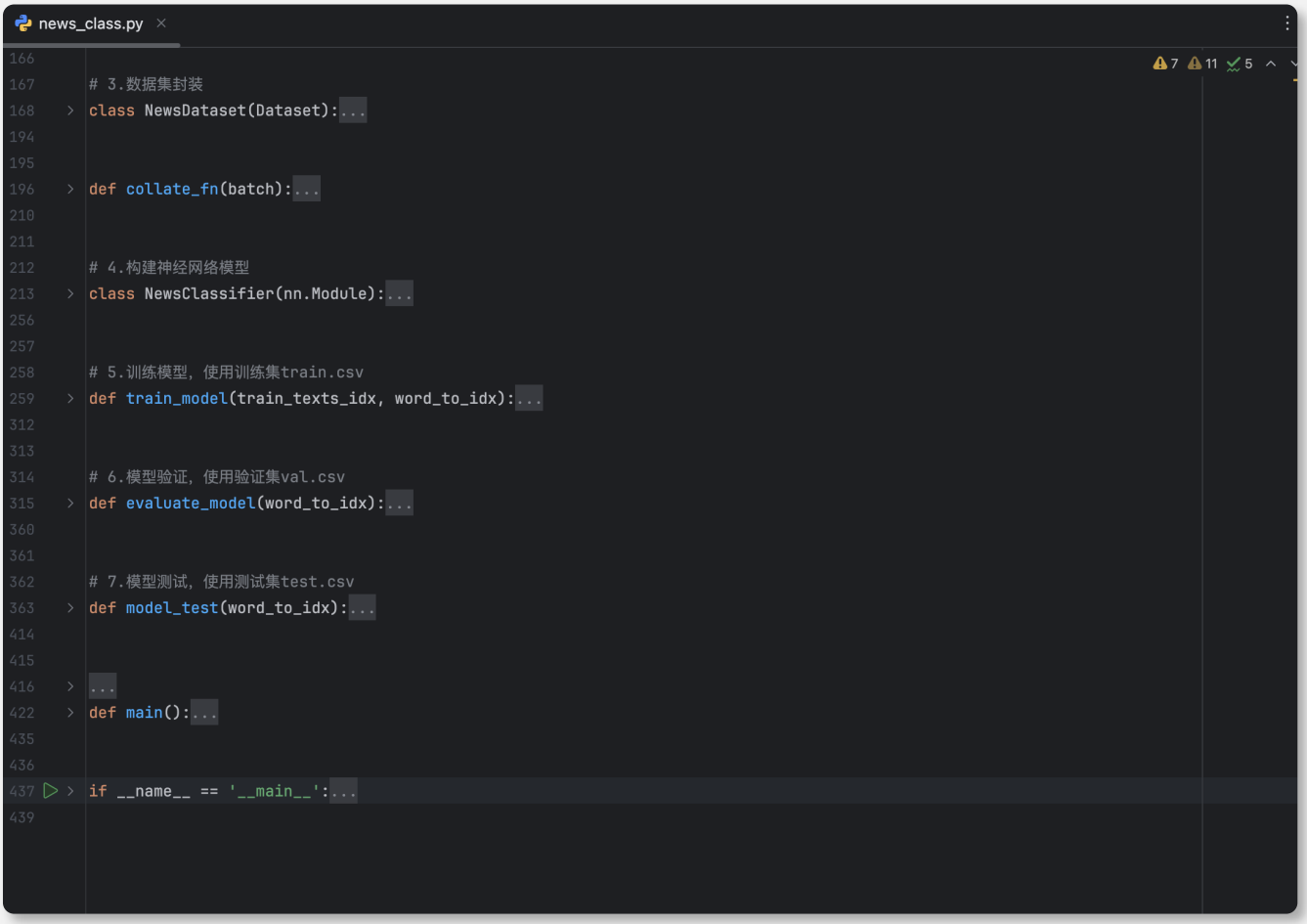
```
NewsTextClassifier
├── code                                # 核心代码目录
│   ├── data_visualize.ipynb            # 数据分布与训练结果可视化分析
│   ├── news_class.py                   # 主程序，包含数据处理、模型定义、训练、验证与测试流程
```

```
5  │   └── t01.py                          # 辅助测试脚本
6  ├── data                                # 数据集与预测结果
7  │   ├── class.txt                       # 类别编号与中文类别名称映射
8  │   ├── images                          # 数据分析相关图片
9  │   ├── test.csv                        # 测试集（无真实标签）
10 │   ├── train.csv                       # 训练集
11 │   ├── val.csv                         # 验证集
12 │   └── 李贺童2201140218.csv            # 测试集预测结果文件
13 ├── doc                                 # 项目文档
14 │   ├── images                          # Markdown图片目录
15 │   │   └── ProjectReport               # Markdown图片字目录，对应具体的文件
16 │   ├── ProjectReport.md                # 项目实验报告（Markdown）
17 │   ├── ProjectReport.pdf               # 项目实验报告（PDF）
18 │   └── 新闻分类分类任务书.pdf           # 项目任务书
19 ├── model                               # 模型存放目录
20 │   └── news_lstm_i128_h512_e20_0p001.pth   # 模型名称，规范参数命名
21 └── README.md                           # 项目说明
```

核心代码

news_class.py

```python
import ...

# 设置最大文本长度
MAX_LEN = 200

# 设备选择：优先使用 Apple GPU (MPS)
device = torch.device("mps" if torch.backends.mps.is_available() else "cpu")
print("Using device:", device)


# 工具函数，用于读取类别描述文件
def load_label_map(path='../data/class.txt'):...


# 0.数据验证
def data_validation():...


# 1.数据读取和清洗
def data_clean():...


# 2.分词，构建词表
def build_vocab():...


# 验证 / 测试文本数值化
def texts_to_indices(texts, word_to_idx):...


# 3.数据集封装
class NewsDataset(Dataset):...
```



思考过程

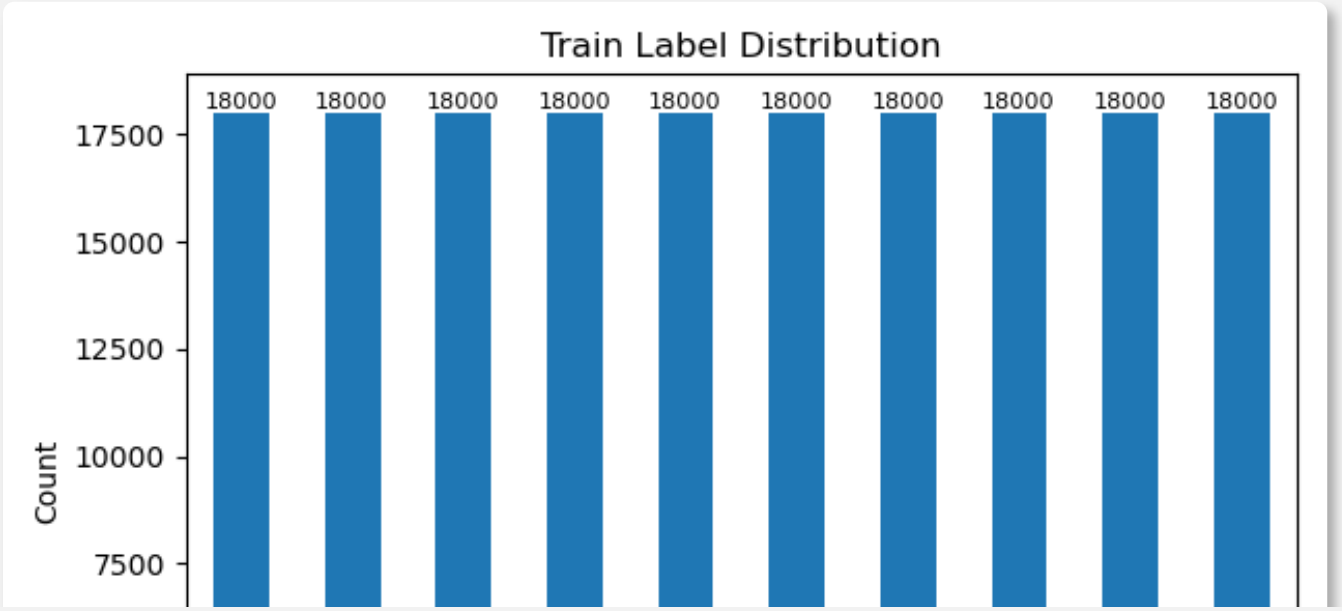
1.数据验证和清洗

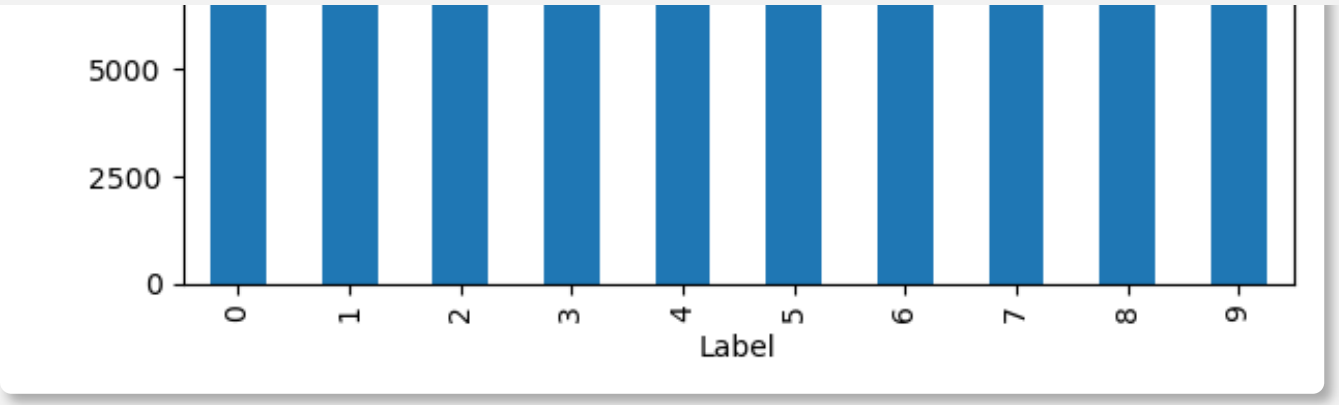
1. 在模型训练前，首先对数据集进行完整性验证，确保后续实验结果可信
2. 项目分别加载 train / val / test 三个数据文件，检查字段结构、样本数量及缺失值情况
3. 同时对训练集标签分布进行可视化分析，用于判断是否存在明显的类别不平衡问题，为后续模型设计提供依据

```python
data_validation()
```

```python
# 输出结果

[train_data] (180000, 2)
[val_data] (10000, 2)
[test_data] (10000, 2)
Index(['text', 'label'], dtype='object')
Index(['text', 'label'], dtype='object')
Index(['text', 'label'], dtype='object')
[train_text_nan] 0
[val_text_nan] 0
[test_text_nan] 0
[train_label_nan] 0
[val_label_nan] 0
```

```python
def data_clean():
    """
    数据没有缺失值，可以不动，尽量不破坏原数据
    :return: 返回三个读取的数据文件
    """
    train_data = pd.read_csv('../data/train.csv')
    val_data = pd.read_csv('../data/val.csv')
    test_data = pd.read_csv('../data/test.csv')
    return train_data, val_data, test_data
```





2.分词，构建词表

1. 本项目采用 jieba 对中文文本进行分词，并仅基于训练集构建词表
2. 这样可以避免验证集或测试集中的词信息在训练阶段被模型“提前看到”，从而防止数据泄漏
3. 对文本长度设置最大阈值（MAX_LEN），既保证了主要语义信息，又控制了训练时间和显存消耗
4. 在构建词表时，我将 `<PAD>` 映射为 0，作为填充符，用于 batch 内对齐长度；将 `<UNK>` 映射为 1，用于表示训练集中未出现的词。这种设计符合 PyTorch Embedding 的常见约定，也有利于模型区分无效填充与未知词，从而提升训练稳定性
5. 在验证和测试阶段，我不会重新构建词表，而是使用训练阶段得到的 `word_to_idx`。对于未在训练集中出现的词，统一映射为 `<UNK>`，从而保证模型结构的一致性，并避免数据泄漏问题

```python
def build_vocab():
    # 0.数据验证,获取数据
    train_data, val_data, test_data = data_clean()
    # 1.对train文本进行分词
    train_texts = []
    all_words = []
    # 遍历每行文本并分词，一行为一个分词列表
    for text in train_data['text']:
        participle = jieba.lcut(text)[:MAX_LEN]
        train_texts.append(participle)
        all_words.extend(participle)
    # 2.去重，集合去重转列表
    train_texts_unique = list(set(all_words))
    # 3.词频，统计每个词出现的次数
    word_counter = Counter(all_words)
    # 4.构建词表，全保留
    word_to_idx = {'<PAD>': 0, '<UNK>': 1}
    # 索引从2开始,追加元素
    id_idx = 2
```

```python
    for word in train_texts_unique:
        word_to_idx[word] = id_idx
        id_idx += 1
    train_texts_idx = [
        [word_to_idx[word] for word in text]
        for text in train_texts
    ]
    return train_texts_idx, word_to_idx
```

```python
# 验证 / 测试文本数值化
def texts_to_indices(texts, word_to_idx):
    # 最终结果容器，二维列表
    texts_idx = []
    # 遍历每行文本
    for text in texts:
        # 中文分词，最大长度截断
        words = jieba.lcut(text)[:MAX_LEN]
        texts_idx.append(
            [word_to_idx.get(word, word_to_idx['<UNK>']) for word in words]
        )
    return texts_idx
```

3.数据集封装

1. 为适配 PyTorch 的训练流程，项目将文本与标签封装为 Dataset 类
2. `collate_fn` 用于定义 DataLoader 在生成一个 batch 时，如何将多条样本进行组合、补齐和张量化

```python
class NewsDataset(Dataset):
    def __init__(self, texts, labels):
        self.texts = texts
        self.labels = labels

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

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
```

```python
        return text, label


def collate_fn(batch):
    texts, labels = zip(*batch)
    # 找 batch 中最长句子
    max_len = max(len(text) for text in texts)
    padded_texts = []
    for text in texts:
        padded = text + [0] * (max_len - len(text))
        padded_texts.append(padded)
    texts_tensor = torch.tensor(padded_texts, dtype=torch.long)
    labels_tensor = torch.tensor(labels, dtype=torch.long)
    return texts_tensor, labels_tensor
```

4.构建神经网络模型

1. 模型采用 **Embedding + LSTM + 全连接层** 的经典文本分类结构

2. Embedding 层将离散词索引映射为连续向量表示，一般使用128维，太小表达能力不够，太大参数膨胀，容易过拟合

3. LSTM 用于建模文本序列中的上下文依赖关系，并取最后时刻的隐藏状态作为整句语义表示，hidden_size为256是因为在表达能力与计算成本之间进行权衡，通过实验发现该配置在验证集上表现稳定

4. 最后通过全连接层输出各类别的预测结果

```python
class NewsClassifier(nn.Module):
    def __init__(self, vocab_size, num_class):
        super().__init__()
        # 1.词嵌入层
        # 参数: 词表大小，词嵌入维度
        self.embedding = nn.Embedding(vocab_size, 128)
        # 2.循环网络层
        self.lstm = nn.LSTM(
            input_size=128,
            hidden_size=256,
            num_layers=1,
            batch_first=True
        )
        # 3.输出层
        self.fc = nn.Linear(256, num_class)

    def forward(self, x):
        x = self.embedding(x)
        _, (hidden, _) = self.lstm(x)
        hidden = hidden.squeeze(0)
        out = self.fc(hidden)
```

```python
        out = self.fc(hidden)
        return out
```

5.训练模型

1. 训练过程中采用 Adam 优化器，结合动量与自适应学习率机制，加快模型收敛并提升训练稳定性
2. 通过训练损失曲线可以观察到 loss 在前期快速下降，后期趋于平稳，模型收敛良好
3. build_vocab函数由main函数统一调用，返回值统一保存
4. 在 DataLoader 中通过 collate_fn 定义 batch 内样本的组织方式，该函数会在 DataLoader 每次生成 batch 时自动调用，用于实现动态 padding 和张量化

```python
# 5.训练模型，使用训练集train.csv
def train_model(train_texts_idx, word_to_idx):
    # 1.获取数据
    train_data, val_data, test_data = data_clean()
    # 2.构建词表
    # train_texts_idx, word_to_idx = build_vocab()
    # 取出所有的行的标签存为列表
    labels = train_data['label'].tolist()
    # 3.数据集和数据加载器
    dataset = NewsDataset(train_texts_idx, labels)
    dataloader = DataLoader(
        dataset,
        batch_size=32,
        shuffle=True,
        collate_fn=collate_fn
    )
    # 4.初始化模型
    vocab_size = len(word_to_idx)
    # 分类的种类数量
    num_class = len(set(labels))
    # 模型，使用GPU
    model = NewsClassifier(vocab_size, num_class).to(device)
    # 5.损失函数和优化器
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    # 6.循环训练模型
    epochs = 10
    train_losses = []
    for epoch in range(epochs):
        # 开始时间
        start = time.time()
        # 迭代次数
        iter_num = 0
        # 总损失
```

```python
        total_loss = 0
        for texts, labels in dataloader:
            texts = texts.to(device)
            labels = labels.to(device)
            outputs = model(texts)
            loss = criterion(outputs, labels)
            # 梯度清零，反向传播，更新参数
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            # 累加损失
            total_loss += loss.item()
            iter_num += 1
        # 打印本轮训练信息
        print(
            f'epoch [{epoch + 1}/{epochs}] '
            f'loss: {total_loss / iter_num:.4f} '
            f'time: {time.time() - start:.2f}s'
        )
        # 保存训练损失
        train_losses.append(total_loss / iter_num)
    # 绘制训练损失曲线
    plt.plot(train_losses)
    plt.title('Training Loss Curve')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.savefig('../data/images/train_loss.png')
    plt.close()
    # 保存模型
    torch.save(model.state_dict(),
'../model/news_lstm_i128_h256_e10_0p001.pth')
```

python

!

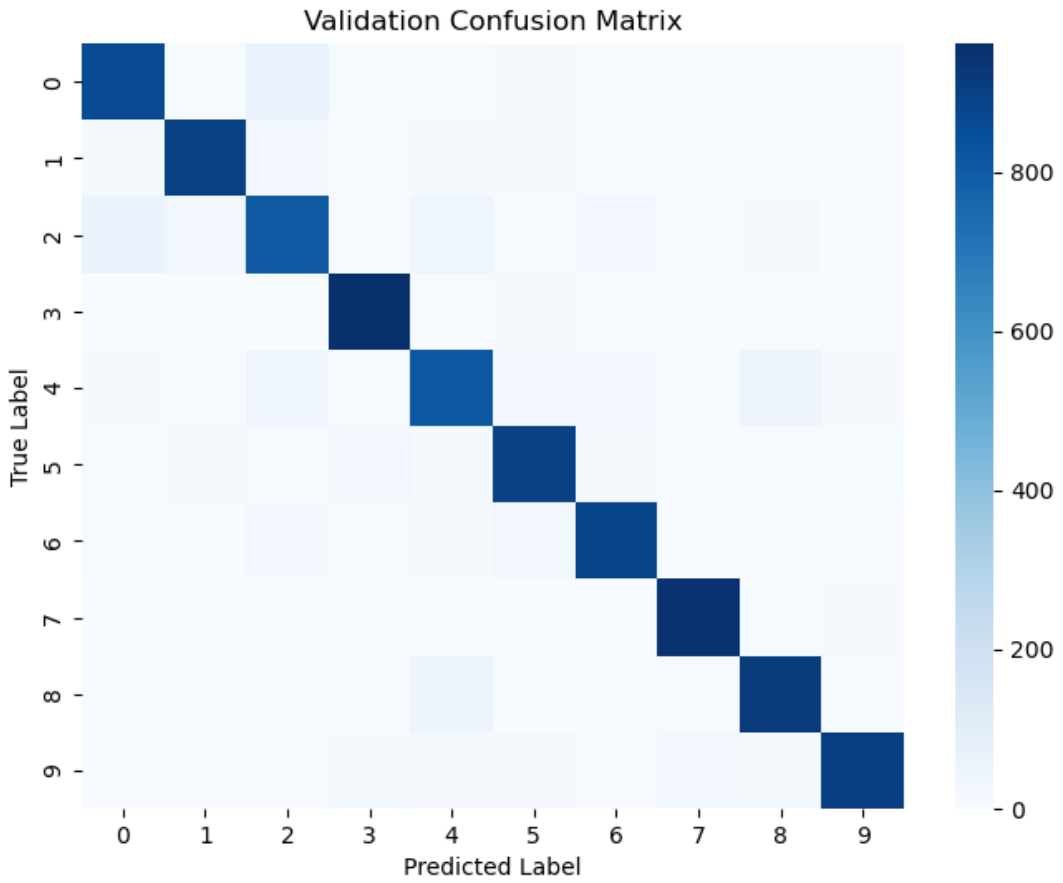
6.模型验证

1. 在模型验证阶段，验证集仅用于评估模型性能
2. 在验证阶段，特别注意模型初始化时的类别数量必须与训练阶段保持一致，避免由于类别数不匹配导致的模型加载错误
3. 混淆矩阵中，对角线表·示预测正确的样本数量，可以看到大部分类别集中在对角线上，说明模型整体分类效果较好
4. 少量非对角元素反映了语义相近类别之间的混淆现象

```python
def evaluate_model(word_to_idx):
    print(f'{'-' * 30}开始模型验证{'-' * 30}')
    # 1.加载数据
```

```
    train_data, val_data, test_data = data_clean()
    # 2.构建词表
    # train_texts_idx, word_to_idx = build_vocab()
    # 3.处理验证集文本和标签
    val_texts = val_data['text'].tolist()
    val_labels = val_data['label'].tolist()
    # 文本转为词索引
    val_texts_idx = texts_to_indices(val_texts, word_to_idx)
    # 4.构建验证集 Dataset 和 DataLoader
    val_dataset = NewsDataset(val_texts_idx, val_labels)
    val_dataloader = DataLoader(
        val_dataset,
        batch_size=32,
        shuffle=False,
        collate_fn=collate_fn
    )
    # 5.加载模型
    vocab_size = len(word_to_idx)
    # 验证集的标签用自己的，但是标签种类数量需要使用train的，种类数量需要对齐
    # num_class = len(set(val_labels))
    num_class = len(set(train_data['label']))
    model = NewsClassifier(vocab_size, num_class).to(device)
    model_path = '../model/news_lstm_i128_h256_e10_0p001.pth'
    model.load_state_dict(torch.load(model_path, map_location=device))
    # 7.切换为评估模式
    model.eval()
    # 8.开始验证
    all_preds = []
    all_labels = []
    # 验证阶段不需要梯度
    with torch.no_grad():

        for texts, labels in val_dataloader:
            texts = texts.to(device)
            labels = labels.to(device)
            outputs = model(texts)
            preds = torch.argmax(outputs, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    # 计算准确率
    acc = accuracy_score(all_labels, all_preds)
    print(f'[val_acc] {acc:.4f}\t[model_path]{model_path}')
    print(f'{'-' * 30}结束模型验证{'-' * 30}')
    # 混淆矩阵
    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(8, 6))
    sns.heatmap(
        cm,
        annot=False,
        fmt='d',
        cmap='Blues'
    )
    plt.xlabel('Predicted Label')
```

```python
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Validation Confusion Matrix')
    # 保存图片
    os.makedirs('../data', exist_ok=True)
    plt.savefig('../data/images/confusion_matrix.png')
    plt.close()
```



7.模型测试

1. 在测试阶段，由于无真实标签，采用占位标签构建 Dataset，仅用于模型推理
2. 最终将预测的类别编号及其对应的中文类别名称保存为 CSV 文件，作为最终提交结果

```python
def model_test(word_to_idx):
    """
    模型测试（无真实标签）：
    1. 使用训练好的模型对 test.csv 进行预测
    2. 将预测的类别 id 与中文类别名保存为 CSV
    """
    print(f'{'-' * 30}开始模型测试{'-' * 30}')

    # 1. 加载数据
    train_data, _, test_data = data_clean()
```

```python
    # lable全0占位，只取test列
    test_texts = test_data['text'].tolist()
    # 2. 文本数值化
    test_texts_idx = texts_to_indices(test_texts, word_to_idx)
    # 3. 构建 Dataset & DataLoader
    dummy_labels = [0] * len(test_texts)
    test_dataset = NewsDataset(test_texts_idx, dummy_labels)
    test_dataloader = DataLoader(
        test_dataset,
        batch_size=32,
        shuffle=False,
        collate_fn=collate_fn
    )
    # 4. 加载模型
    vocab_size = len(word_to_idx)
    num_class = len(set(train_data['label'].tolist()))
    model = NewsClassifier(vocab_size, num_class).to(device)
    model_path = '../model/news_lstm_i128_h256_e10_0p001.pth'
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.eval()
    # 5. 加载类别映射
    label_map = load_label_map()
    # 6. 推理
    all_preds = []
    with torch.no_grad():
        for texts, _ in test_dataloader:
            texts = texts.to(device)
            outputs = model(texts)
            preds = torch.argmax(outputs, dim=1)
            all_preds.extend(preds.cpu().numpy())
    # 7. 保存结果
    result_df = pd.DataFrame({
        'text': test_texts,
        'label': all_preds
    })
    save_path = '../data/李贺童2201140218.csv'
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    result_df.to_csv(save_path, index=False, encoding='utf-8-sig')
    print(f'预测结果已保存至: {save_path}')
    print(f'{'-' * 30}结束模型测试{'-' * 30}')
```

python

8.优化

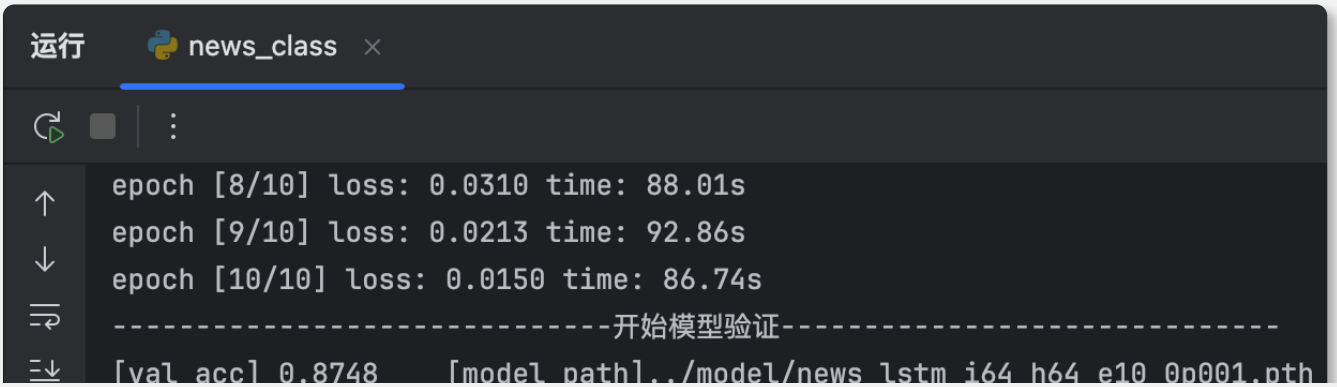
1. 模型结构优化
   - 使用双向 LSTM 提升上下文建模能力
   - 引入 Transformer 结构以捕捉长距离依赖
2. 文本表示优化

- 使用预训练词向量（Word2Vec / fastText）
- 使用 BERT 等预训练语言模型进行特征抽取

3. 训练策略优化
   - 使用学习率衰减或早停策略防止过拟合
   - 引入 Dropout 提升泛化能力
4. 工程优化
   - 模型参数配置化
   - 训练日志与实验结果记录

训练成果

1. news_lstm_i64_h64_e10_0p001.pth



```
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至：../data/test_prediction_result.csv
------------------------------结束模型测试------------------------------
```

2 news_lstm_i128_h64_e10_0p001.pth

运行 news_class

```
epoch [8/10] loss: 0.0176 time: 119.39s
epoch [9/10] loss: 0.0124 time: 121.67s
epoch [10/10] loss: 0.0103 time: 125.11s
------------------------------开始模型验证------------------------------
[val_acc] 0.8775    [model_path]../model/news_lstm_i128_h64_e10_0p001.pth
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至：../data/test_prediction_result.csv
------------------------------结束模型测试------------------------------
```

3 news_lstm_i256_h64_e10_0p001.pth

运行 news_class

```
epoch [8/10] loss: 0.0137 time: 263.82s
epoch [9/10] loss: 0.0100 time: 320.43s
epoch [10/10] loss: 0.0090 time: 326.93s
------------------------------开始模型验证------------------------------
[val_acc] 0.8754    [model_path]../model/news_lstm_i256_h64_e10_0p001.pth
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至：../data/test_prediction_result.csv
------------------------------结束模型测试------------------------------
```

4 news_lstm_i64_h128_e10_0p001.pth

运行 news_class

```
epoch [8/10] loss: 0.0213 time: 72.94s
epoch [9/10] loss: 0.0155 time: 71.34s
```

```
epoch [10/10] loss: 0.0121 time: 76.65s
------------------------------开始模型验证------------------------------
[val_acc] 0.8814    [model_path]../model/news_lstm_i64_h128_e10_0p001.pth
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至: ../data/李贺童2201140218.csv
------------------------------结束模型测试------------------------------
```

5 news_lstm_i64_h256_e10_0p001.pth

运行 news_class

```
epoch [8/10] loss: 0.0174 time: 89.15s
epoch [9/10] loss: 0.0136 time: 91.68s
epoch [10/10] loss: 0.0112 time: 90.67s
------------------------------开始模型验证------------------------------
[val_acc] 0.8766    [model_path]../model/news_lstm_i64_h256_e10_0p001.pth
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至: ../data/李贺童2201140218.csv
------------------------------结束模型测试------------------------------
```

6 news_lstm_i128_h256_e10_0p001.pth

运行 news_class

```
epoch [8/10] loss: 0.0137 time: 259.89s
epoch [9/10] loss: 0.0121 time: 221.90s
epoch [10/10] loss: 0.0094 time: 258.54s
------------------------------开始模型验证------------------------------
[val_acc] 0.8873    [model_path]../model/news_lstm_i128_h256_e10_0p001.pth
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至: ../data/李贺童2201140218.csv
------------------------------结束模型测试------------------------------
```

7 news_lstm_i128_h256_e20_0p001.pth

运行 news_class

```
epoch [18/20] loss: 0.0048 time: 219.03s
epoch [19/20] loss: 0.0049 time: 184.03s
epoch [20/20] loss: 0.0044 time: 215.68s
------------------------------开始模型验证------------------------------
[val_acc] 0.8848    [model_path]../model/news_lstm_i128_h256_e20_0p001.pth
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至：../data/李贺童2201140218.csv
------------------------------结束模型测试------------------------------
```

8 news_lstm_i128_h512_e20_0p001.pth

运行 news_class

```
epoch [18/20] loss: 0.0063 time: 297.55s
epoch [19/20] loss: 0.0057 time: 297.56s
epoch [20/20] loss: 0.0064 time: 312.91s
------------------------------开始模型验证------------------------------
[val_acc] 0.8915    [model_path]../model/news_lstm_i128_h512_e20_0p001.pth
------------------------------结束模型验证------------------------------
------------------------------开始模型测试------------------------------
预测结果已保存至：../data/李贺童2201140218.csv
------------------------------结束模型测试------------------------------
```

项目后续扩展

Git管理项目

远程仓库GitHub地址： https://github.com/Hikari0x/NewsTextClassifier.git