个性化新闻标题生成模型优化方法及代码修改建议

1. 用户兴趣建模优化

（1）引入更多行为特征

思路：在用户编码器输入中加入时间、设备等特征。

```
# filepath: pensmodule/Preprocess/preprocess.ipynb
# 假设原有用户行为数据有时间戳字段
df['time_feature'] = df['timestamp'].apply(lambda x: extract_time_feature(x))
# 在用户编码器的数据准备部分增加 time_feature
user_feature = [clicked_news_ids, time_feature]
```

（2）使用更复杂的用户建模结构

思路：将 NRMS 替换为 Transformer 编码器。

```
# filepath: pensmodule/UserEncoder/model.py
import torch.nn as nn

class UserTransformerEncoder(nn.Module):
    def __init__(self, embedding_matrix, n_layers=2, n_heads=4):
        super().__init__()
        self.embed = nn.Embedding.from_pretrained(torch.tensor(embedding_matrix,
dtype=torch.float))
        encoder_layer =
nn.TransformerEncoderLayer(d_model=embedding_matrix.shape[1], nhead=n_heads)
        self.transformer = nn.TransformerEncoder(encoder_layer,
num_layers=n_layers)
    def forward(self, user_clicked_news):
        x = self.embed(user_clicked_news)
        x = x.permute(1, 0, 2)  # (seq_len, batch, embed_dim)
        out = self.transformer(x)
        return out.mean(dim=0)
# 替换原 NRMS
model = UserTransformerEncoder(embedding_matrix)
```

---

2. 生成器优化

（1）使用预训练语言模型

思路：用 HuggingFace 的 T5/BERT 作为编码器或解码器。

```
# filepath: pensmodule/Generator/model.py
from transformers import T5ForConditionalGeneration, T5Tokenizer

tokenizer = T5Tokenizer.from_pretrained('t5-small')
model = T5ForConditionalGeneration.from_pretrained('t5-small')
# 输入格式需适配
input_ids = tokenizer(news_body, return_tensors='pt', padding=True,
truncation=True).input_ids
labels = tokenizer(news_title, return_tensors='pt', padding=True,
truncation=True).input_ids
outputs = model(input_ids=input_ids, labels=labels)
loss = outputs.loss
```

（2）多任务学习

思路：同时优化点击率预测和标题生成。

```
# filepath: pensmodule/Generator/train.py
# 生成损失
gen_loss = generation_loss_fn(outputs, targets)
# 点击率预测损失
click_pred = click_predictor(user_vec, news_vec)
click_loss = click_loss_fn(click_pred, click_labels)
# 总损失
loss = gen_loss + alpha * click_loss
```

---

3. 训练策略优化

（1）课程学习（Curriculum Learning）

思路：先训练基础生成，再逐步引入个性化目标。

```
# filepath: pensmodule/Generator/train.py
for epoch in range(total_epochs):
    if epoch < base_epochs:
        # 只优化生成损失
        loss = gen_loss
    else:
        # 加入个性化奖励
        loss = gen_loss + beta * personalization_reward
```

（2）奖励函数细化

思路：区分不同类型的个性化需求，奖励函数分层。

```
# filepath: pensmodule/Generator/train.py
def reward_function(generated, reference, user_profile):
    rouge_reward = compute_rouge(generated, reference)
    personalization_reward = compute_personalization(generated, user_profile)
    diversity_reward = compute_diversity(generated)
    return rouge_reward + 0.5 * personalization_reward + 0.2 * diversity_reward
```

---

4. 数据增强与负采样

（1）数据增强

思路：对新闻正文进行同义词替换、随机删除等。

```
# filepath: pensmodule/Preprocess/preprocess.ipynb
def synonym_replacement(text):
    # 简单同义词替换实现
    # ...实现细节...
    return new_text
news_body = [synonym_replacement(body) for body in news_body]
```

（2）负采样优化

思路：采样更难区分的负样本。

```
# filepath: pensmodule/Preprocess/preprocess.ipynb
def hard_negative_sampling(pos_list, all_news, news_index):
    # 选取与正样本相似度高的负样本
    # ...实现细节...
    return hard_neg_list
```

---

5. 推理阶段优化

（1）多样化解码

思路：使用 Top-k 或 Top-p 采样。

```
# filepath: pensmodule/Generator/eval.py
outputs = model.generate(
    input_ids,
    do_sample=True,
    top_k=50,
    top_p=0.95,
    num_return_sequences=3
)
```

（2）结合用户实时反馈

思路：根据用户反馈动态调整生成策略。

```python
# filepath: pensmodule/Generator/train.py
if user_feedback == 'like':
    reward += feedback_bonus
elif user_feedback == 'dislike':
    reward -= feedback_penalty
```

> 以上代码片段仅为思路示例，具体实现需结合你的项目结构和数据格式进行适配。