实验报告 - Project2 H

第一部分 HMM

---

1. 基本原理

隐马尔可夫模型（HMM）是一个统计模型，它假设系统以一种隐藏的状态序列存在，这些状态只能间接地通过观测到的数据序列来观察。在命名实体识别（NER）任务中，HMM被用来模拟文字和其对应的标签之间的关系。

HMM模型的三个基本元素包括：

- **状态转移概率矩阵** A：a_{ij} 表示从状态 i 转移到状态 j 的概率。
- **观测概率矩阵** B：$b_j(k)$ 表示在状态 j 观测到观测 k 的概率。
- **初始状态概率向量** π：π_i 表示系统在时间 $t = 1$ 处于状态 i 的概率。

给定观测序列 $O = o_1, o_2, \ldots, o_T$，HMM使用如下公式进行解码以找到最可能的状态序列 $S = s_1, s_2, \ldots, s_T$：
$P(S|O) = \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t}(o_t)$

维特比算法（Viterbi Algorithm）是一种动态规划算法，用于在已知观测序列和模型参数的情况下，计算最可能的状态序列。

---

2. 实验设置与数据处理

- **数据预处理**（`data_process.py`）：预处理阶段包括读取数据文件，生成 `word2id` 和 `tag2id` 映射，这些映射帮助将文本数据转换为数值数据，以便模型处理。

```
# data_process.py
def build_vocab(files):
    word2id = {'<UNK>': 0}  # 未知词处理
    for file in files:
        with open(file, 'r') as f:
            for line in f:
                word, tag = line.strip().split()
                if word not in word2id:
                    word2id[word] = len(word2id)
    return word2id

def build_tag2id(tags):
    tag2id = {tag: idx for idx, tag in enumerate(tags)}
    return tag2id
```

- **模型文件**（`HMM_model.py`）：实现HMM的基本结构，包括参数的初始化、模型的训练方法以及使用Viterbi算法的解码方法。

```
# HMM_model.py
import numpy as np
```

```
class HMM:
    def __init__(self, num_states, num_observations):
        self.A = np.zeros((num_states, num_states))
        self.B = np.zeros((num_states, num_observations))
        self.pi = np.zeros(num_states)

    def train(self, data):
        # 训练代码，包括估计A, B, pi
        # 伪代码示例
        pass

    def viterbi(self, observations):
        # Viterbi 算法的实现
        # 伪代码示例
        pass
```

- **训练和测试代码**（`train.ipynb` 和 `test.ipynb`）：使用Jupyter Notebook进行模型的训练和测试，有效地记录和展示实验过程和结果。

---

3. 使用的优化

为了防止在计算概率时出现数据下溢，我们采用对数概率的方法。在概率的乘法运算中，通过转换成对数空间，将乘法运算转换为加法运算，从而避免了连乘导致的数值下溢问题。以下是实现该技巧的关键代码片段：

```
def viterbi(self, observation):
    N, T = len(self.Pi), len(observation)
    delta = np.zeros((N, T))
    psi = np.zeros((N, T), dtype=int)

    delta[:, 0] = self.Pi + self.B[:, self.vocab.get(observation[0], 0)]
    for t in range(1, T):
        O_t = self.vocab.get(observation[t], 0)
        for j in range(N):
            delta[j, t] = np.max(delta[:, t - 1] + self.A[:, j]) + self.B[j, O_t]
            psi[j, t] = np.argmax(delta[:, t - 1] + self.A[:, j])

    best_path = np.zeros(T, dtype=int)
    best_path[-1] = np.argmax(delta[:, -1])
    for t in range(T - 2, -1, -1):
        best_path[t] = psi[best_path[t + 1], t + 1]

    return [self.idx2tag[id] for id in best_path]
```

---

4. 实验结果

- **中文数据集** 的 micro avg f1-score 为 **0.8734**

- **英文数据集** 的 micro avg f1-score 为 **0.8173**

第二部分 HMM+CRF

1. 基本原理

条件随机场（CRF）是一种用于标注和分割序列数据的统计建模方法。它是一种无向图模型，用于编码观测序列和标签序列之间的条件概率分布。与HMM不同的是，CRF能够在整个序列中同时考虑前后文信息，从而进行全局最优的标签预测。

CRF模型的核心是定义在状态转移和观测之上的潜在特征函数，并通过这些特征函数对标签序列进行概率建模。数学表示为：

$$P(Y|X) = \frac{1}{Z(X)} \exp\left(\sum_{t=1}^{T} \sum_{k} \lambda_k f_k(y_{t-1}, y_t, X, t)\right)$$

其中，f_k 是特征函数，λ_k 是对应的权重，$Z(X)$ 是规范化因子确保概率和为1。

---

2. 实验设置与数据处理

- **数据预处理**：与HMM部分共享，使用相同的 `word2id` 和 `tag2id` 映射，以及转换函数。
- **模型文件（`sklearn_crf.py`）**：使用 `sklearn-crfsuite` 库，实现CRF模型的训练和预测。库自动处理特征函数的设置和权重的学习。

```
# sklearn_crf.py
from sklearn_crfsuite import CRF

def train_crf(X_train, y_train):
    crf = CRF(
        algorithm='lbfgs',
        c1=0.1,  # L1正则化
        c2=0.1,  # L2正则化
        max_iterations=100,
        all_possible_transitions=True
    )
    crf.fit(X_train, y_train)
    return crf

def predict_crf(model, X_test):
    return model.predict(X_test)
```

- **训练和测试代码（`train.ipynb` 和 `test.ipynb`）**：训练和评估CRF模型，使用真实和预测的标签序列来计算性能指标，如F1-score。

---

3. 使用的优化

CRF模型通过特征函数对依赖关系进行编码，减少了手动特征工程的需要。此外，通过正则化参数（`c1` 和 `c2`）控制模型的复杂度，防止过拟合。`sklearn-crfsuite` 还提供了 `all_possible_transitions` 参数，这可以自动学习观测序列中未出现的状态转移的潜在特征，增强了模型的泛化能力。

```
# 特征提取示例代码
def word2features(sent, i):
    word = sent[i][0]
    features = {
        'bias': 1.0,
        'word.lower()': word.lower(),
        'word[-3:]': word[-3:],
        'word[-2:]': word[-2:],
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit(),
    }
    if i > 0:
        word1 = sent[i-1][0]
        features.update({
            '-1:word.lower()': word1.lower(),
            '-1:word.istitle()': word1.istitle(),
            '-1:word.isupper()': word1.isupper(),
        })
    else:
        features['BOS'] = True

    if i < len(sent)-1:
        word1 = sent[i+1][0]
        features.update({
            '+1:word.lower()': word1.lower(),
            '+1:word.istitle()': word1.istitle(),
            '+1:word.isupper()': word1.isupper(),
        })
    else:
        features['EOS'] = True

    return features
```

---

4. 实验结果

在validation数据集上，

- **中文数据集**的 micro avg f1-score 为**0.9497**
- **英文数据集**的 micro avg f1-score 为**0.8921**

这些结果表明CRF模型在两个数据集上均显示出较高的性能，特别是在处理中文数据集时。

第三部分 Transformer+CRF

1. 基本原理

Transformer+CRF 模型结合了 Transformer 的高效编码能力和 CRF 的序列解码能力，提供了一个在命名实体识别（NER）任务中具有强大表现的框架。Transformer 层通过自注意力机制捕捉序列的深层依赖关系，而 CRF 层则确保整个序列的输出在全局上是最优的。

数学模型：

Transformer 层输出的特征序列 H 被用作 CRF 层的输入，其中 CRF 利用转移概率和特征来计算标签序列的概率：

$$P(y|x) = \frac{\exp(\sum_{i=1}^{n} \psi(y_{i-1}, y_i, H_i))}{\sum_{y' \in Y} \exp(\sum_{i=1}^{n} \psi(y'_{i-1}, y'_i, H_i))}$$

这里，ψ 表示由转移概率和特征向量 H_i 组合定义的得分函数，Y 代表所有可能的标签序列。

---

2. 实验设置与数据处理

数据处理：

- **数据预处理**（`data_process.py`）：负责将原始文本转换为模型可以处理的格式，包括词汇和标签的索引化。
- **数据集构建**（`dataset.py`）：定义了一个 PyTorch 的 Dataset 类，用于加载和批量处理数据。

```
# dataset.py
from torch.utils.data import Dataset, DataLoader

class NERDataset(Dataset):
    def __init__(self, texts, tags, word2id, tag2id):
        self.texts = [[word2id.get(word, word2id['<UNK>']) for word in text] for text in
texts]
        self.tags = [[tag2id[tag] for tag in tag_seq] for tag_seq in tags]

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

    def __getitem__(self, idx):
        return torch.tensor(self.texts[idx]), torch.tensor(self.tags[idx])
```

模型定义（`model.py`）：

- 实现了包含 Transformer 编码器和 CRF 层的模型，用于学习序列的依赖关系并优化标签的解码过程。

```
# model.py
import torch
import torch.nn as nn
from torchcrf import CRF

class TransformerCRF(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab, label_map, device="cpu"):
        super(TransformerCRF, self).__init__()
        self.embedding_dim = embedding_dim
        self.hidden_dim = hidden_dim
        self.vocab_size = len(vocab)
        self.tagset_size = len(label_map)
        self.device = device
```

```
        self.word_embeds = nn.Embedding(self.vocab_size, embedding_dim)
        self.dropout = nn.Dropout(p=0.5)

        self.encoder_layer = TransformerEncoderLayer(d_model=embedding_dim, nhead=8)
        self.transformer_encoder = TransformerEncoder(self.encoder_layer, num_layers=2)

        self.hidden2tag = nn.Linear(embedding_dim, self.tagset_size, bias=True)
        self.crf = CRF(label_map, device)

    def forward(self, sentence, seq_len, tags=None, mode="train"):
        feats = self._get_transformer_features(sentence, seq_len)
        if mode == "train":
            if tags is None:
                raise ValueError("In training mode, tags must be provided")
            loss = self.crf.neg_log_likelihood(feats, tags, seq_len)
            return loss
        elif mode == "eval":
            all_tag = []
            for i, feat in enumerate(feats):
                all_tag.append(self.crf._viterbi_decode(feat[:seq_len[i]])[1])
            return all_tag
        elif mode == "pred":
            return self.crf._viterbi_decode(feats[0])[1]
```

- **嵌入层**（`self.embedding`）：将词汇索引转换为向量。
- **Transformer 编码器**（`self.transformer_encoder`）：对嵌入向量进行编码，捕捉词汇之间的依赖关系。
- **全连接层**（`self.fc`）：将 Transformer 的输出映射到标签空间的尺寸。
- **CRF 层**（`self.crf`）：接受来自 Transformer 的特征，进行最终的序列解码。

训练和评估（`runner.py`）：

- 定义了训练循环和评估过程，监控模型的性能并调整参数。

```
# runner.py
def train(model, train_loader, optimizer):
    model.train()
    total_loss = 0
    for sentences, tags in train_loader:
        optimizer.zero_grad()
        loss = model(sentences, tags)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(train_loader)

def evaluate(model, valid_loader):
    model.eval()
    with torch.no_grad():
        for sentences, tags in valid_loader:
```

```
            predictions = model(sentences)
            # Evaluate predictions
```

---

3. 实验结果

使用 Transformer+CRF 模型，在中文和英文数据集上进行训练和验证，得到以下 F1-score：

- **中文数据集** 的 micro avg f1-score 为 **0.9256**
- **英文数据集** 的 micro avg f1-score 为 **0.7248**

这些成绩表明了 Transformer+CRF 模型在处理具有复杂结构依赖的 NER 任务时的有效性。

---

4. 讨论

本实验突出了 Transformer 和 CRF 结合使用在序列标注任务中的优势。特别是在中文数据集上，该模型展现了出色的性能。这证明了 Transformer 在捕捉长距离依赖关系方面的能力，以及 CRF 在确保序列标签全局一致性方面的效率。未来的研究可以探讨不同的 Transformer 架构变体，或进一步优化 CRF 层的实现。