1 数据预处理

对于语料库中的文本，首先进行如下的预处理操作。

a). 取出文本中的所有标点符号，并用空格代替。

b). 用自然语言处理库 nltk 对文本进行切词。

c). 将所有单词的字母全部转变为小写字母。

d). 如果某个单词不是由字母组成，则将其过滤。

e). 将单词的形式进行标准化。

2 特征提取

采用 sklearn 中的 `CountVectorizer` 和 `TfidfTransformer` 对经过预处理过后的语料库进行特征提取，取语料库的出现频率最高的 20000 个 unigram 作为特征词，并将这些特征词的 Tf-IDF 作为文本的特征。

值得说明的是，sklearn 中的 `CountVectorizer` 会对语料库中的停用词进行过滤，所以在数据预处理中没有考虑停用词。

3 LogLinear 模型的实现

3.1 模型参数

模型的参数是一个大小为 $4 \times d$ 的 Numpy 矩阵 $\boldsymbol{P}$, 其中 $d = 20000$ 表示特征的维度。

3.2 前向传播

模型的输入：大小为 $b \times d$ 的矩阵 $\boldsymbol{D}$，其中 b 表示一个 batch 所包含的数据条数。

模型的输出：

$$\text{LogLinear}(\boldsymbol{D}) = \text{SoftMax}(\boldsymbol{D} \times \boldsymbol{P}^T)$$

3.3 反向传播

设矩阵

$$\boldsymbol{P} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix}$$

其中 λ_i 表示矩阵 $\boldsymbol{P}$ 的第 i 个行向量，对应着标签为第 i 类的特征参数。同时，设矩阵 $\boldsymbol{D}$ 的第 i 个行向量为 f_i，表示第 i 条数据，其真实的特征为 y_i。

则优化目标为

$$\min_{\lambda_{1\sim 4}} \Phi = \sum_{i=1}^{b} \left(-f_i \cdot \lambda_{y_i} + \log \left(\sum_{j=1}^{4} e^{f_i \cdot \lambda_j} \right) \right)$$

求梯度：

$$\frac{\partial \Phi}{\partial \lambda_j} = \sum_{i=1}^{n} \left(-f_i \cdot \delta_{[y_i=j]} + f_i \frac{e^{f_i \cdot \lambda_j}}{\sum_{k=1}^{4} e^{f_i \cdot \lambda_k}} \right)$$

在每次迭代中进行一次梯度下降：

$$\lambda_i \leftarrow \lambda_i - \frac{\partial \Phi}{\partial \lambda_i} \times \mathrm{lr}$$

其中，lr 表示学习率。

```
class Log_Linear:
    def __init__(self):
        self.params = np.random.rand(4, feature_size).astype(np.float64)
    def forward(self, data_batch : np.ndarray, labels_batch : np.ndarray, require_cache=
        self.product = data_batch @ np.transpose(self.params)
        result = np.argmax(self.product, axis=1)
        acc = float(np.sum(result == labels_batch)) / labels_batch.shape[0]
        if require_cache == False:
            return result, acc
        self.cache = data_batch
        self.labels_cache = labels_batch
```

```
        self.sum_exp = np.sum(np.exp(self.product), axis=1)
        acc_params = self.params[labels_batch]
        loss = float(np.mean(-np.sum(data_batch * acc_params, axis=1) + np.log(self.sum_
        return result, loss, acc
    def backward(self):
        self.prob = np.exp(self.product) / np.expand_dims(self.sum_exp, axis=1)
        self.grad = np.zeros_like(self.params, dtype=np.float64)
        for i in range(self.cache.shape[0]):
            self.grad += np.outer(self.prob[i], self.cache[i])
            self.grad[self.labels_cache[i]] -= self.cache[i]

        self.params -= self.grad * lr
```

3.4 训练策略

采用随机梯度下降法对模型进行训练。在一次 epoch 中，首先随机打乱训练集，每次取出 1024 条数据喂给模型进行训练，直至遍历完整个训练集。一共进行 4 次 epoch，随着 epoch 的增加，逐渐降低学习率。

数据迭代器实现如下：

```
class Data_Iterator:
    def __init__(self, data, labels, batch_size):
        self.data = data
        self.labels = labels
        self.btch_siz = batch_size
    def __iter__(self):
        self.order = np.random.permutation(data_size)
        self.idx = 0
        return self
    def __len__(self):
        return int(np.ceil(data_size / self.btch_siz))
    def __next__(self):
        if self.idx == data_size:
```

```
            raise StopIteration
        nxt = min(self.idx + self.btch_siz, data_size)
        slce = self.order[self.idx : nxt]
        self.idx = nxt
        return self.data[slce].astype(np.float64), self.labels[slce]
```

4 实验结果

一共进行 4 次实验，实验结果如下。

次数	最后一次迭代的正确率
1	0.896
2	0.873
3	0.880
4	0.927
统计	0.894±0.0208

表 1: 在训练集上的表现

次数	正确率	四个标签的平均 F1 分数
1	0.896	0.896
2	0.896	0.896
3	0.897	0.896
4	0.896	0.896
统计	0.896±0.0004	0.896±0

表 2: 在测试集上的表现