基于半监督学习方法的模型微调模块实现

在此系统中，微调模型需要用到两个数据集，分别是有标签数据集和无标签数据集，其中有标签数据集在读取数据时已经获得，对于无标签数据集而言，我们通过联合上述所有模块，最终可以得到精确构造的伪标签数据集用于微调模型，上述过程即为半监督学习方法。具体过程如下述算法 1 所示：

算法 1：用于小样本语言隐写分析的具有自训练功能的预训练语言模型

输入：少量有标签数据集 D_l ，大量无标签数据集 D_u ，基础模型 $B^W(\cdot)$ ，前向传播次数 T ，自训练次数 N ，学习率 η。

输出：模型 $B^{W_N}(\cdot)$。

1：使用少量有标签数据集 D_l 微调基础模型 $B^W(\cdot)$ 得$B^{W_0}(\cdot)$ ；

2：从第一次自训练开始，也就是 $n=0$ ；

3：进入共计 N 次的 while 循环；

4：使用 MC dropout 方法前向传播 T 次，每次的对象都是 D_u 中的全体样本，每个样本最终都得到 T 个预测标签 $\tilde{y}^t$ ；

5：使用公式 (14) 得每个样本的伪标签 $\tilde{y}$ ；

6：使用公式 (15) 估算每个样本伪标签的信息增益；

7：使用公式 (16) 对所有样本进行排序，之后从中选择具有明显且有代表性隐写分析特征的样本组成数据集 D_p ；

8：使用公式 (18) 向样本添加稳定性权重，得出新的损失函数 $Loss'$ ，用于后续微调；

9：使用数据集 D_p 微调基础模型 $B^{W_n}(\cdot)$ ，更新模型参数，得$B^{W_{n+1}}(\cdot)$，损失函数为 $Loss'$ ，也就是公式 (17) ；

10：n 自加，准备进行下一次自训练；

11：N 次自训练后，结束 while 循环。至此自训练结束。

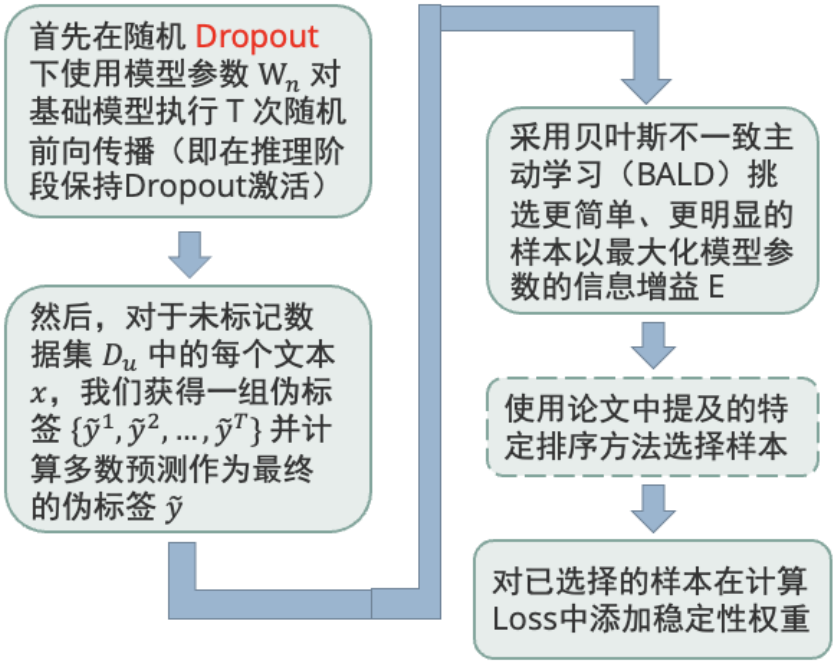

图 5 自训练框架流程图（不包含训练）

其中的自训练框架流程图（不包含训练）如图 5 所示。在上述算法中，小样本标记数据集 D_l 用作我们系统模型的初始训练集。通过对 D_l 进行训练得到参数为 W_0 的基础模型 $B^{W_0}(\cdot)$。然后将基本模型应用于 D_u 以获得伪标记数据集 D_p。利用 D_p 对基础模型进行微调得到 $B^{W_n}(\cdot)$。后续不断重复这一步骤，直到满足收敛标准。值得注意的是，如果在未标记的输入中没有注释，那么基础模型的预测标签可能不可靠且有噪声。为了提高模型区分普通文本和隐写文本的能力，我们重点关注表述清晰且具有被广泛认可的隐写分析特征的可信度高的样本。通过这样做，模型便可以更好地学习普通文本和隐写文本之间的分布差异，而不会受到各种噪声源的干扰。有必要衡量模型的预测有多少不确定性。正如一位学者所研究[46]的，softmax 之后的单个预测不包含与模型本身对此特定输出的确定性有关的任何信息，因此不适合估计模型的不确定性。模型的不确定性估计只能借助模型已经学习到的参数的近似后验分布 $p'(W|X,Y)$ 来完成。有了这样的后验分布，输出本身就变成了一个随机变量，并且可以根据样本中固有的噪声量来评估其自身的不确定性。就像执行近似变分推理一样，我们在一个易于处理的族中找到了一个简单的分布 $q_\Theta(W)$，它最大限度地减少了与真实模型后验 $p(W|X,Y)$ [47] 的 Kullback-Leibler (KL) 散度。

根据上述算法我们实现的具体模块如下所示：

一、有标签数据集微调子模块

- **模块整体框架**

```
for counter in range(N_base):
    with strategy.scope():
        print(pt_teacher_checkpoint)
        model = models.construct_teacher(TFModel, Config, pt_teacher_checkpoint, max_seq_length, len(labels), dense_dropout=dense_dropout, attention_probs_dropout_prob=attention_probs_dropout_prob, hidden_dropout_prob=hidden_dropout_prob)
        model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08), loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="acc")])
        if counter == 0:
            logger.info(model.summary())

    model_file = os.path.join(model_dir, "model.h5")
    logger.info("Model file is {}".format(model_file))
    model.fit(x=X_train, y=y_train, shuffle=True, epochs=sup_epochs, validation_data=(X_dev, y_dev), batch_size=sup_batch_size * gpus, callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)])
```

1. 循环创建和训练模型

```
for counter in range(N_base):
```

这里通过一个循环，创建并训练 N_base 个模型，在后续子训练过程中选用这些模型中性能最优秀的作为基础模型。

2. 设置分布式策略范围

```
with strategy.scope():
```

在 strategy.scope() 内创建和编译模型，使得训练可以在多个 GPU 上进行，通过调用 tf.distribute.MirroredStrategy() ，系统可以平行化模型训练，加速模型收敛。

3. 打印检查点路径

```
print(pt_teacher_checkpoint)
```

打印当前的检查点路径，用于调试或者记录信息，方便记录模型训练信息及后续 debug 过程。

4. 构造教师模型

```
model = models.construct_teacher(TFModel, Config, pt_teacher_checkpoint, max_seq_length, len(labels), dense_dropout=dense_dropout, attention_probs_dropout_prob=attention_probs_dropout_prob, hidden_dropout_prob=hidden_dropout_prob)
```

此处我们调用自定义函数 construct_teacher 来创建教师模型，模型参数包括模型结构 TFModel，配置 Config，预训练的检查点路径 pt_teacher_checkpoint，输入序列的最大长度 max_seq_length，标签数 len(labels) 以及各个 Dropout 概率（dense_dropout、attention_probs_dropout_prob、hidden_dropout_prob）。

5. 编译模型

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08),
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name="acc")])
```

- 使用 Adam 优化器进行模型编译，学习率为 3e-5。
- 损失函数为稀疏分类交叉熵（SparseCategoricalCrossentropy），用于该检测系统中类别标签是整数而不是独热编码的分类任务。
- 评估指标为稀疏分类准确率（SparseCategoricalAccuracy）。

6. 打印模型结构（仅第一次循环时）

```
if counter == 0:
    logger.info(model.summary())
```

仅在第一次循环时记录模型结构的摘要信息，以便检查模型配置，及时审查模型运行状态。

7. 定义模型文件路径

```
model_file = os.path.join(model_dir, "model.h5")
logger.info("Model file is {}".format(model_file))
```

为了将模型保存到指定位置，我们在模块中定义并记录模型文件路径，该路径用于后续的模型保存操作。

8. 训练模型

```
model.fit(
    x=X_train,
    y=y_train,
    shuffle=True,
    epochs=sup_epochs,
    validation_data=(X_dev, y_dev),
    batch_size=sup_batch_size * gpus,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)]
)
```

- x=X_train, y=y_train: 使用训练数据 X_train 和标签 y_train 进行模型训练。

- shuffle=True: 在每个 epoch 开始前对训练数据进行随机打乱，以避免模型学到数据的固定序列。
- epochs=sup_epochs: 设定训练的轮数 sup_epochs。
- validation_data=(X_dev, y_dev): 指定验证数据 (X_dev, y_dev)，用于在每个 epoch 结束后进行模型评估。
- batch_size=sup_batch_size * gpus: 指定批处理大小，乘以 gpus 的数目，以适应多 GPU 并行训练。
- callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)]: 使用 EarlyStopping 回调函数监控验证损失 val_loss，如果验证损失在 10 个 epoch 内没有改善，提前停止训练，并恢复到验证损失最小的权重。

该子模块定义了模型使用有标签数据集微调的过程，模型通过少量有标注的数据进行训练，优化自身的权重参数，初步在验证集上拥有了一定的分类性能。

二、无标签数据集半监督微调子模块

- **模块整体框架**

```
for epoch in range(25):
    X_batch, y_batch, X_conf = f_(
        tokenizer, X_unlabeled_sample, y_mean, y_var, y_pred, unsup_size, len(labels), y_T=y_T)
    if not conf:
        logger.info("Not using confidence learning.")
        X_conf = np.ones(len(X_batch['input_ids']))
        logger.info("Weights ".format(X_conf[:10]))
    else:
        logger.info("Using confidence learning ".format(X_conf[:10]))
        X_conf = -np.log(X_conf+1e-10)*alpha
        logger.info("Weights ".format(X_conf[:10]))
    model.fit(x=X_batch, y=y_batch, shuffle=True, epochs=unsup_epochs, validation_data=(X_dev, y_dev), batch_size=unsup_batch_size*gpus,sample_weight=X_conf, callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)])
```

1. 迭代训练轮次

```
for epoch in range(25):
```

我们设定模型自训练的轮次数为 25 次，这个次数为可调的超参数，在系统运行过程中不断地调整，优化模型训练效果。

2. 采样生成伪标签

```
X_batch, y_batch, X_conf = f_(
    tokenizer, X_unlabeled_sample, y_mean, y_var, y_pred, unsup_size, len(labels), y_T=y_T)
```

- 调用函数 f_ 从无标签数据集 X_unlabeled_sample 中采样，并使用通过 Dropout 技术生成的伪标签数据 y_mean、y_var、y_pred 和 y_T 来计算置信度 X_conf。
- 参数包括：
- tokenizer：Bert 模型中对输入数据进行分词和编码的工具。
- X_unlabeled_sample：无标签数据集。
- y_mean, y_var, y_pred, y_T：通过蒙特卡罗 Dropout 计算得到的伪标签信息。
- unsup_size：无标签数据的批次大小。
- len(labels)：类别数。
- y_T：蒙特卡罗 Dropout 生成的多次采样结果。

3. 处理置信学习 (Confidence Learning)

```
if not conf:
    logger.info("Not using confidence learning.")
    X_conf = np.ones(len(X_batch['input_ids']))
    logger.info("Weights ".format(X_conf[:10]))
else:
    logger.info("Using confidence learning ".format(X_conf[:10]))
    X_conf = -np.log(X_conf + 1e-10) * alpha
    logger.info("Weights ".format(X_conf[:10]))
```

- 检查是否使用置信学习：
- 如果 conf 为 False，表示不使用置信学习，为所有样本分配相同的权重，即均匀权重 X_conf = np.ones(len(X_batch['input_ids']))。
- 如果 conf 为 True，表示使用置信学习，将置信度 X_conf 转换为相应的权重。通过 -np.log(X_conf + 1e-10) * alpha 计算，alpha 是一个放大系数，用于调整权重的幅度。
- np.log 函数确保低置信度的样本将具有较高权重，从而在训练中对它们进行更多关注。

4. 微调模型

```
model.fit(
    x=X_batch,
    y=y_batch,
    shuffle=True,
    epochs=unsup_epochs,
    validation_data=(X_dev, y_dev),
    batch_size=unsup_batch_size * gpus,
    sample_weight=X_conf,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)]
)
```

其中，sample_weight=X_conf：设定每个样本的权重，以反映其置信度，其他部分与有标签数据集微调子模块中相应的部分保持一致。

通过运用上述自训练架构，模型可以从无标签数据中提取有用信息，以进一步提升其性能。这在我们恶意隐写载体检测系统的数据资源匮乏但无标签数据丰富的情况下，大大提高了数据利用率，改善了模型的泛化能力和精度。

通过上述两个子模块的联同配合，我们在系统中实现了基于半监督学习方法对模型的微调，其中包括一个重要的架构，即自训练架构，大大提高了整个系统对于目标载体的检测能力。