基于贝叶斯不一致主动学习的局部采样器方法实现

- 方法整体框架

```
def get_BALD_acquisition(y_T):

    expected_entropy = - np.mean(np.sum(y_T * np.log(y_T + 1e-10), axis=-1), axis=0)
    expected_p = np.mean(y_T, axis=0)
    entropy_expected_p = - np.sum(expected_p * np.log(expected_p + 1e-10), axis=-1)
    return (entropy_expected_p - expected_entropy)

def sample_by_bald_easiness(tokenizer, X, y_mean, y_var, y, num_samples, num_classes, y_T):

    logger.info ("Sampling by easy BALD acquisition function")
    BALD_acq = get_BALD_acquisition(y_T)
    p_norm = np.maximum(np.zeros(len(BALD_acq)), (1. - BALD_acq)/np.sum(1. - BALD_acq))
    p_norm = p_norm / np.sum(p_norm)
    logger.info (p_norm[:10])
    indices = np.random.choice(len(X['input_ids']), num_samples, p=p_norm, replace=False)
    X_s = {"input_ids": X["input_ids"][indices], "token_type_ids": X["token_type_ids"][indices], "attention_mask": X["attention_mask"][indices]}
    y_s = y[indices]
    w_s = y_var[indices][:,0]
    return X_s, y_s, w_s

def sample_by_bald_class_easiness(tokenizer, X, y_mean, y_var, y, num_samples, num_classes, y_T):

    logger.info ("Sampling by easy BALD acquisition function per class")
    BALD_acq = get_BALD_acquisition(y_T)
    BALD_acq = (1. - BALD_acq)/np.sum(1. - BALD_acq)
    logger.info (BALD_acq)
    samples_per_class = num_samples // num_classes
    X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s = [], [], [], [], []
    for label in range(num_classes):
        X_input_ids, X_token_type_ids, X_attention_mask = X['input_ids'][y == label], X['token_type_ids'][y == label], X['attention_mask'][y == label]
        y_ = y[y==label]
```

```
        y_var_ = y_var[y == label]
        p_norm = BALD_acq[y==label]
        p_norm = np.maximum(np.zeros(len(p_norm)), p_norm)
        p_norm = p_norm/np.sum(p_norm)
        if len(X_input_ids) < samples_per_class:
            logger.info ("Sampling with replacement.")
            replace = True
        else:
            replace = False
        indices = np.random.choice(len(X_input_ids), samples_per_class, p=p_norm, replace=replace)
        X_s_input_ids.extend(X_input_ids[indices])
        X_s_token_type_ids.extend(X_token_type_ids[indices])
        X_s_attention_mask.extend(X_attention_mask[indices])
        y_s.extend(y_[indices])
        w_s.extend(y_var_[indices][:,0])
    X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s = shuffle(X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s)
    return {'input_ids': np.array(X_s_input_ids), 'token_type_ids': np.array(X_s_token_type_ids), 'attention_mask': np.array(X_s_attention_mask)}, np.array(y_s), np.array(w_s)
```

该模块下的这些函数主要用于贝叶斯不确定性下的主动学习，通过计算信息增益（BALD，Bayesian Active Learning by Disagreement）来进行样本的选择和构造新的训练数据集，具体而言有三个函数：

- **get_BALD_acquisition 函数**

基于蒙特卡罗 (MC) Dropout 结果进行计算信息增益（BALD）值，该值用于衡量模型对每个样本不确定性的程度。

- **sample_by_bald_easiness 函数**

基于 BALD 值和置信度进行全局样本选择。选择一些样本，这些样本具有较高的不确定性，从而增加训练集的多样性和改进模型的性能。

- **sample_by_bald_class_easiness 函数**

与 sample_by_bald_easiness 函数类似，但进一步按照类别进行平衡采样。这确保每个类别中选取一定比例的样本，有效避免类别不平衡导致的模型偏差。

1.1.1. 信息增益计算模块

- **模块整体框架**

```
def get_BALD_acquisition(y_T):
    expected_entropy = - np.mean(np.sum(y_T * np.log(y_T + 1e-10), axis=1), axis=0)
```

```
    expected_p = np.mean(y_T, axis=0)
    entropy_expected_p = - np.sum(expected_p * np.log(expected_p + 1e-10), axis=-1)
    return (entropy_expected_p - expected_entropy)
```

1. 计算期望熵

   - **y_T**：形状为 (T, N, C)，其中 T 是 MC Dropout 样本数，N 是样本数，C 是类别数。
   - **np.sum(y_T * np.log(y_T + 1e-10), axis=-1)**：计算每个样本在每次 MC Dropout 中的预测分布的熵。
   - **np.mean(..., axis=0)**：对 T 次采样结果取平均，得到每个样本的期望熵。

2. 计算期望预测

```
expected_p = np.mean(y_T, axis=0)
```

   - **np.mean(y_T, axis=0)**：计算 T 次采样的平均预测分布。

3. 计算期望预测的熵

```
entropy_expected_p = - np.sum(expected_p * np.log(expected_p + 1e-10), axis=-1)
```

   - **np.sum(expected_p * np.log(expected_p + 1e-10), axis=-1)**：计算 N 个样本的平均预测分布的熵。

4. 计算并返回信息增益

```
return (entropy_expected_p - expected_entropy)
```

   - **entropy_expected_p - expected_entropy**：计算并返回信息增益（BALD），这表示模型对每个样本的预测分布之间的不一致性。

我们设计这一模块使得我们的系统能够根据不同的样本预测结果做到真正的模型不确定性量化，并为后续带权排序提供依据。

1.1.2. 带权采样及伪标签数据集构造模块

一、sample_by_bald_easiness 子模块

- **模块整体框架**

```
def sample_by_bald_easiness(tokenizer, X, y_mean, y_var, y, num_samples, num_classes, y_T):

    logger.info ("Sampling by easy BALD acquisition function")
    BALD_acq = get_BALD_acquisition(y_T)
    p_norm = np.maximum(np.zeros(len(BALD_acq)), (1. - BALD_acq)/np.sum(1. - BALD_acq))
    p_norm = p_norm / np.sum(p_norm)
    logger.info (p_norm[:10])
```

```
    indices = np.random.choice(len(X['input_ids']), num_samples, p=p_norm, replace=False)
    X_s = {"input_ids": X["input_ids"][indices], "token_type_ids": X["token_type_ids"][indices], "attention_mask": X["attention_mask"][indices]}
    y_s = y[indices]
    w_s = y_var[indices][:,0]
    return X_s, y_s, w_s
```

1. 计算 BALD 值

```
logger.info("Sampling by easy BALD acquisition function")
BALD_acq = get_BALD_acquisition(y_T)
```

调用上述已经设计好的 get_BALD_acquisition 模块计算每个样本的信息增益（BALD）。

2. 归一化采样概率

```
p_norm = np.maximum(np.zeros(len(BALD_acq)), (1. - BALD_acq) / np.sum(1. - BALD_acq))
p_norm = p_norm / np.sum(p_norm)
logger.info(p_norm[:10])
```

- 计算 (1 - BALD_acq)，即信息增益的补集。
- 对该补集进行归一化，得到采样概率 p_norm。

3. 随机采样

```
indices = np.random.choice(len(X['input_ids']), num_samples, p=p_norm, replace=False)
```

- 根据 p_norm 进行带权随机采样 num_samples 个样本。

4. 构造伪标签数据集

```
X_s = {
    "input_ids": X["input_ids"][indices],
    "token_type_ids": X["token_type_ids"][indices],
    "attention_mask": X["attention_mask"][indices]
}
y_s = y[indices]
w_s = y_var[indices][:, 0]
return X_s, y_s, w_s
```

我们根据采样得到的 indices 提取相应的样本，构造伪标签数据集 X_s，标签集 y_s 和用于辅助后续 loss 函数收敛的权重集 w_s。

二、sample_by_bald_class_easiness 子模块

- **模块整体框架**

```
def sample_by_bald_class_easiness(tokenizer, X, y_mean, y_var, y, num_samples, num_classes, y_T):

    logger.info ("Sampling by easy BALD acquisition function per class")
```

```
    BALD_acq = get_BALD_acquisition(y_T)
    BALD_acq = (1. - BALD_acq)/np.sum(1. - BALD_acq)
    logger.info (BALD_acq)
    samples_per_class = num_samples // num_classes
    X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s = [], [], [], [], []
    for label in range(num_classes):
        X_input_ids, X_token_type_ids, X_attention_mask = X['input_ids'][y == label], X['token_type_ids'][y == label], X['attention_mask'][y == label]
        y_ = y[y==label]
        y_var_ = y_var[y == label]
        p_norm = BALD_acq[y==label]
        p_norm = np.maximum(np.zeros(len(p_norm)), p_norm)
        p_norm = p_norm/np.sum(p_norm)
        if len(X_input_ids) < samples_per_class:
            logger.info ("Sampling with replacement.")
            replace = True
        else:
            replace = False
        indices = np.random.choice(len(X_input_ids), samples_per_class, p=p_norm, replace=replace)
        X_s_input_ids.extend(X_input_ids[indices])
        X_s_token_type_ids.extend(X_token_type_ids[indices])
        X_s_attention_mask.extend(X_attention_mask[indices])
        y_s.extend(y_[indices])
        w_s.extend(y_var_[indices][:,0])
    X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s = shuffle(X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s)
    return {'input_ids': np.array(X_s_input_ids), 'token_type_ids': np.array(X_s_token_type_ids), 'attention_mask': np.array(X_s_attention_mask)}, np.array(y_s), np.array(w_s)
```

在这一子模块中，我们按类别进行采样，以此可以有效减缓正负样本不平衡带来的模型倾斜问题。

1. 计算 BALD 值并完成权重归一化

```
logger.info("Sampling by easy BALD acquisition function per class")
BALD_acq = get_BALD_acquisition(y_T)
BALD_acq = (1. - BALD_acq) / np.sum(1. - BALD_acq)
logger.info(BALD_acq)
```

- 调用 get_BALD_acquisition 计算每个样本的信息增益（BALD）。
- 将信息增益转变为归一化的采样权重。

2. 初始化变量

```
samples_per_class = num_samples // num_classes
X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s = [], [], [], [], []
```

在这一过程中，我们分别进行两项处理为采样做准备：

- 计算每个类别需要的样本数 samples_per_class。
- 初始化空列表 X_s_input_ids、X_s_token_type_ids、X_s_attention_mask、y_s 和 w_s，用于存储采样的结果。

3. 按类别抽样

```
for label in range(num_classes):
    X_input_ids, X_token_type_ids, X_attention_mask = X['input_ids'][y == label], X['token_type_ids'][y == label], X['attention_mask'][y == label]
    y_ = y[y == label]
    y_var_ = y_var[y == label]

    p_norm = BALD_acq[y == label]
    p_norm = np.maximum(np.zeros(len(p_norm)), p_norm)
    p_norm = p_norm / np.sum(p_norm)

    if len(X_input_ids) < samples_per_class:
        logger.info("Sampling with replacement.")
        replace = True
    else:
        replace = False

    indices = np.random.choice(len(X_input_ids), samples_per_class, p=p_norm, replace=replace)

    X_s_input_ids.extend(X_input_ids[indices])
    X_s_token_type_ids.extend(X_token_type_ids[indices])
    X_s_attention_mask.extend(X_attention_mask[indices])
    y_s.extend(y_[indices])
    w_s.extend(y_var_[indices][:, 0])
```

接下来进入到我们这一子模块的核心部分，即分类采样，我们对每个类别分别进行如下操作：

- 提取该类别的所有样本 X_input_ids、X_token_type_ids、X_attention_mask 和 y、y_var。
- 根据 BALD 值计算归一化的概率 p_norm。
- 如果某类别样本数目不足 samples_per_class，使用有放回采样（replace=True），否则使用无放回采样。

- 根据计算得到的概率 p_norm 进行采样，并将选中的样本和相应的标签、置信度添加到结果列表中。

4. 对采样结果进行打乱洗牌

```
X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s = shuffle(X_s_input_ids, X_s_token_type_ids, X_s_attention_mask, y_s, w_s)
return {
    "input_ids": np.array(X_s_input_ids),
    "token_type_ids": np.array(X_s_token_type_ids),
    "attention_mask": np.array(X_s_attention_mask)
}, np.array(y_s), np.array(w_s)
```

- 首先打乱数据顺序，确保采样的多样性。
- 之后返回包含输入 ID、token 类型 ID 和注意力掩码的新样本字典 X_s，相应的标签 y_s 和置信度 w_s。

经过以上三个子模块，我们最终实现了一套完整的基于贝叶斯不一致主动学习的部分采样器，通过信息增益计算、全局和类别平衡采样，使得模型能够高效地学习并提高其在恶意隐写载体检测这一场景下的性能。总的来说，这三个子模块在系统中各司其职但又相互配合：

在不确定性计算与采样方面：

- get_BALD_acquisition 模块帮助计算样本的信息增益，通过这些信息增益可以确定哪些样本在当前模型下不确定性最高。
- sample_by_bald_easiness 子模块通过这些不确定性值选择样本，确保新采样的数据能够帮助模型改进。
- sample_by_bald_class_easiness 子模块则进一步确保每个类别样本的平衡，减缓了因类别不平衡导致的模型偏差。

在模型改进与多样性采样方面：

- 通过主动地选择不确定样本，最大限度地利用有限的标注资源，增强模型的泛化能力。
- 在恶意隐写载体检测任务中，挑选不确定性高的样本进行人工标注，或者选择这些样本进行重点监测使得我们的系统能够更早发现隐蔽的攻击行为并以此迭代式地优化检测模型。