Notes-splitleaf

In the algorithm, we pop out a tree from the queue and then iterate through all the leaves:

```
# enumerate through all the leaves
for i in range(len(leaves)):
```

For each leaf i, if it is not dead and if $\text{tree.splitleaf}[0, i] == 1$, then we split it, otherwise we continue:

```
# enumerate through all the leaves
for i in range(len(leaves)):
    # print("d!!!",d)
    # if the leaf is dead, then continue
    if tree.leaves[i].is_dead == 1:
        continue

    # 0 for not split; 1 for split

    if spl[i] == 0:
        continue
```

We split leaf i, and the lower bound is calculated using all leaves except leaf i. We iterate through all features, and for each feature j, if it is not in the path from root to leaf i, then we split leaf i using feature j, and then push the new tree into the queue.

```
lb = tree.lbound[i]  # the lower bound
b0 = tree.leaves[i].B0  # the b0 defined in (28) of the paper


d0 = removed_leaf.rules

# split the leaf d0 with feature j
for j in range(1, nrule + 1):
    if j not in d0 and -j not in d0:
        # split leaf d0 with feature j, and get 2 leaves l1 and l2
        l1 = d0 + (-j,)
        l2 = d0 + (j,)
```

Notes - the generate_new_splitleaf function

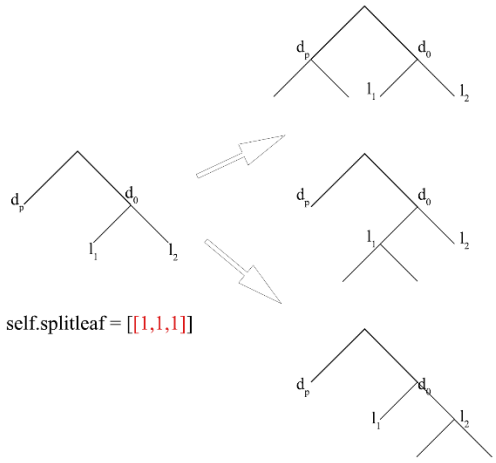
```
def generate_new_splitleaf(splitleaf_list, cap_l, incorr_l, ndata, nleaves, lb, b0, lamb, R_c):
```

Equivalent Support Bound combined with lookahead: bound $+ b_0 + lamb < R^c$,
Accurate Support Bound: a $< lamb$,

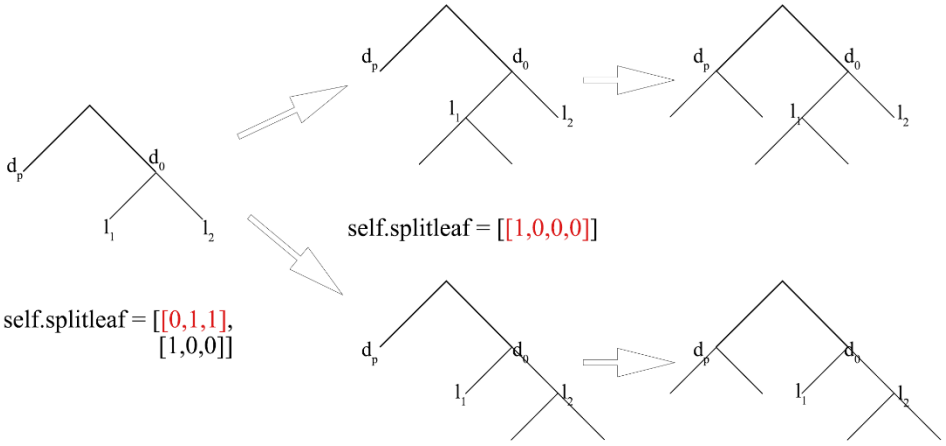
(1) If Equivalent Support Bound does NOT hold, we can prune all trees that contain d_p, which means we need to split at least one leaf in d_p.
(2) If Accurate Support Bound does NOT hold, we can prune all trees that contain (l_1, l_2), which means we need to split at least one of (l_1, l_2).

There are 4 cases, here is how it goes:

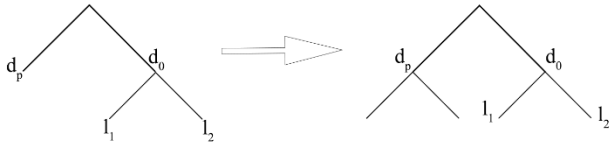
A. If not(1) and not(2), we don't prune any tree, and we push one row [1,1,1,1 …,1,1] into *self.splitleaf*, which marks every leaf to be split.



B. If (1) and (2), we push two rows into *self.splitleaf*, after the two rows are popped out and those leaves are split, we will only get trees which split one leaf in d_p and one leaf in (l_1, l_2).

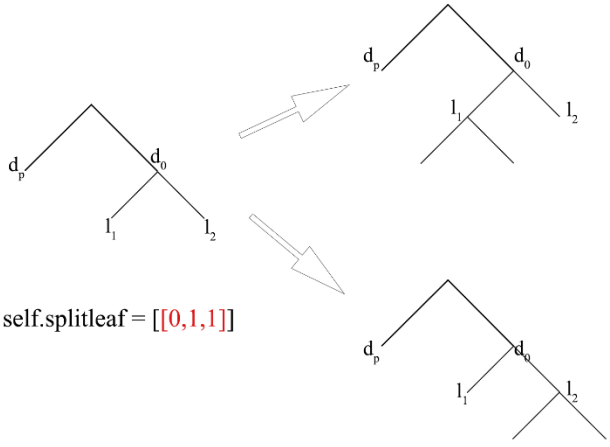


C. If (1) and not(2), we push one row into *self.splitleaf*, which marks all leafs in d_p to be split, but l_1 and l_2 to be not split.



self.splitleaf = [[1,0,0]]

D. If not(1) and (2), we push one row into *self.splitleaf*, which marks all leafs in (l_1, l_2) to be split, but d_p to be not split.



self.splitleaf = [[0,1,1]]