KENDAXA ASSIGNMENT
MACHINE LEARNING POSITION

Matěj Ščerba

1 ALGORITHMIC TASK

1.1 Problem definition

Let $\mathcal{G} = (V, E)$ be a connected undirected graph without cycles (a tree graph), where each leaf is colored either using white, blue or red color. The task is to find the maximum number of different pairs of leaves, where one leaf is red and the other blue such that all pairs can connected by mutually disjunctive paths — these paths connects leaves of each pair and these path has no common node.

1.2 Idea

My idea of solving this problem is to reduce given tree's leaves. I take one of the leaves of given tree, push its color to its neighbor and delete the leaf. Once leaf and its neighbor have different colors, a path has been found and I increase the counter. Once a node of tree has some color, then there exists path from such node to some leaf of that color. That means that once I find neighbors of different colors, then there exists a path connecting some red and blue leaf using both neighbors. When these neighbors are found, I remove them both from the tree.

1.3 Algorithm

Each node v of tree has color $c(v)$. The input tree has some leaves colored red, some blue and rest of nodes have no color.

While there are some leaves, the algorithm takes one of them and reduces it. The reduction is done in the following way:

If the leaf has no color or its color is the same as its (only) neighbor's, then it is simply removed.

If its neighbor has no color, then the leaf's color is passed to its neighbor and the leaf is removed.

If the leaf's and its neighbor's color is different (one of them is red, the other is blue), then a path connecting one red and one blue leaf was found, the counter is incremented and the leaf and its neighbor is removed.

Once a node is removed, some nodes may become leaves. Because the given graph is a tree, all nodes eventually become leaves. Hence the algorithm reduces every node of given graph.

1.3.1 Pseudocode

Input: Graph $\mathcal{G} = (V, E)$ with function $c(v) \in \{R, B, N\} : \forall v \in V$.

Output: Maximum number of different pairs (u, v) of nodes, where $c(u) = R$ and $c(v) = B$, that all pairs can be connected by mutually disjunctive paths.

1. `p` $\leftarrow$ `0`
2. while there exists at least one leaf in graph $\mathcal{G}$
3. Take leaf $l \in V$ and its neighbor $u \in V$
4. Remove l from graph
5. if $c(l) \neq N$ and $c(u) \neq N$ and $c(u) \neq c(u)$:
6. Remove u from graph
7. `p` $\leftarrow$ `p + 1`
8. else if $c(u) = N$:
9. $c(u) \leftarrow c(l)$
10. return `p`

1.3.2 Correctness

Once two neighbors have been reduced so that color of one is red and the other's is blue, then there exists a path from a red leaf to blue leaf in given graph using edge connecting those neighbors. So each time a path is found, there exists a pair of leaves that are connected by a path.

This path is disjunctive with other paths because each node of the path has been removed from graph as soon as it is found.

Suppose the algorithm returned submaximal number of pairs. Then the algorithm must have removed a node from tree without finding a path among different-colored leaves.

The leaves that are a part of maximal set of pairs have different color and their color has been pushed to its neighbors. Because the algorithm didn't find a path among them, then at least one of them must have participated in a path with other different-colored leaf.

This means that the algorithm found different pair and thus the returned number of pairs is maximal.

1.3.3 Time complexity

The loop is performed $O(n)$ times, where n is the number of nodes of given tree.

Selecting a leaf is done in constant time, so is the selection of its neighbor (because leaf has only one neighbor).

Pushing leaf's color is done in constant time.

Removing a node is done in amortized constant time, because the complexity depends linearly on the node's degree. Removing a node removes at least one edge of tree, but since number of edges of a tree is $n - 1 = O(n)$, then the amortized time complexity of removing one node is constant.

Checking the condition of finding a path is done in constant time.

Each loop is done in constant time, so the time complexity of the whole algorithm is linear.

1.3.4 Space complexity

Space complexity of the algoritmh is also linear.

Color takes constant space for each node.

Number of list of neighbors is linear (there are $2n = O(n)$ neighbors in list.

Number of leaves is also linear, so they take up linear amount of memory.

1.3.5 Benchmarks

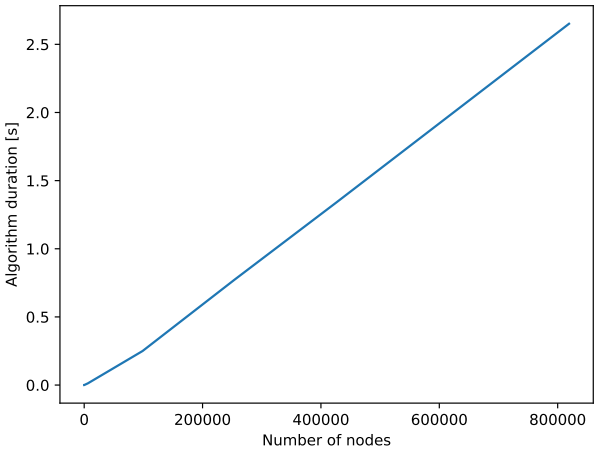

Figure 1.1: A plot showing dependency of algorithm's duration on input size. Each test case has been run 30 times, the plot shows average duration.

I have run the alogorithm on given test cases. Each case has been run 30 times and average duration has been considered. The plot depicted in figure 1.1 shows linear time complexity as described in section 1.3.3, although initialization affects smaller inputs so they seem to be more time-consuming.

2 CLASSIFICATION TASK

I would start with feature engineering: scaling values to interval $[0, 1]$ and creating one hot features for categorical features.

Afterwards, I would create several models, especially neural networks with different number of layers and hidden neurons, and possibly decision trees or random forests.

Then I would run k-fold cross validation using (I would chose k around 5), and possibly tried different parameters for tested models (MLP optimizer, learning rates, ...).

After getting some results, I would perform statistical tests using `scipy` framework to compute N-% confidence interval to show statistical significance.

3 REGRESSION TASK

I would handle the regression task in similar fashion to the classification one.

The main difference would be the usage of data outside the train set itself, mainly calendar datasets focused on economics and stocks.