SD201 TP02

1 Introduction

This lab follows on from the first lab which took place on October, 25th. You need to have completed the first lab before starting this one. The general instructions given in the text of the previous lab still apply to this lab.

1.1 Update the project for this lab

Before starting this lab, you need to replace the *main.py* and *check_results.py* files in the project on your computer with the ones available on ecampus. You also need to add all the files from the *input_data* archive on ecampus to your *input_data* folder, and the files from the *expected_results* archive on ecampus to your *results/expected* folder.

1.2 Code efficiency and cleanness

At the end of the last lab, you will have to submit your code. In addition to the points awarded for answering the questions (10 points for the first lab, 4 points for this lab and 3 points for the last lab), 2.5 points will be awarded for the efficiency of your code and 0.5 points will be awarded for its cleanliness, i.e. its overall readability including the use of clear variable names, the use of functions where appropriate and the sensible use of comments.

2 Questions

2.1 `get_best_gain` for continuous data

In this lab, we will work with continuous features (e.g. temperature, speed or salary whose range of values is potentially infinite while order matters). You are not expected to treat ordinal categorical features separately, they should be considered continuous features. You can distinguish continuous features from the other features by checking the `types` array, which denotes columns corresponding to continuous features with a `FeaturesTypes.REAL` element.

To improve the test error, when splitting continuous features we employ a more refined strategy than the one we saw in our course (slide 86). In particular, after having determined a feature and a threshold which maximize the Gini gain (e.g. price$<$ 100), we compute a new split value as follows. Let L be the set of points associated with the left child of a node v while let R be the set of points associated with the right child of v. Then, the split value is defined to be as $\frac{\max(L)-\min(R)}{2}$. For example, suppose that *price* and 100 are, respectively, the feature and the threshold with maximum Gini gain for the points $L \cup R$ associated with v, where $L = \{20, 45, 80\}$ and $R = \{100, 160, 200\}$. Then, the decision tree will use the value $90 = \frac{80+100}{2}$ as split value for v (instead of 100).

Question 7 (1.5/20) Modify the implementation of the `PointSet`'s `get_best_gain()` method so that it can also deal with continuous features. Also create a method of the PointSet class with the following signature

`get_best_threshold(self) -> float`

which, when called after the `get_best_gain()` method, returns `None` if the selected feature is boolean, the category ID if the selected feature is categorical, and the threshold value if the selected feature is continuous. Note that this method is not expected to call `get_best_gain()` itself, and that its result is not expected to be well-defined if `get_best_gain()` has not been called before. In this case, it might even raise an Exception.

When determining the best threshold, make sure to be able to deal with points with the same value on a given feature that cannot be split, when splitting on that feature.

The results for each test file will be the index of the feature with the best Gini gain, the related gain value and the best threshold that provides that gain.

Hint: For solving this question, you are advised to use either the `sorted()` core python function (link to the documentation) or the numpy `sort()` function (link to the documentation)

2.2 Binary Tree for continuous data

Question 8 (1/20) Update the `Tree` class so that it can build a tree while dealing with continuous features. The result for each test file will be the F1-score of the Tree built using the first 80% of the points of the file as training set, and tested using the remaining 20% points.

Question 9 (1.5/20) So far we considered only the maximum height as stopping condition when building the tree. Another stopping condition which is widely used is to make sure that the number of points associated to every node in the tree is not smaller than a given threshold specified in input. This ensures that our results are statistically significant. To this end, add to the `Tree` constructor a parameter called `min_split_points : int` with default value 1 and update your implementation with an early stopping rule, so that each node in the tree (including the leaves) is associated with at least `min_split_points` points. In particular, when processing a node v, if the split with the best Gini gain would violate such a constraint, then such a split should not be chosen. Instead, you should choose the split which maximizes the Gini gain among all feasible splits. If all splits violate such a constraint then v becomes a leaf.

The result for each test file is the F1-score of the Tree built using the first 80% of the points of the file as training set, and tested using the remaining 20% points.

Question 10 We discussed in our course that a naive implementation for selecting the best Gini gain for continuous features (Question 7) might not be efficient. In particular, its number of operations can be quadratic in the number of points. Part of the points for efficiency will be awarded if you manage to implement such an algorithm incrementally, so that the total running time for processing one feature is $O(n \log n)$, where n is the number of points. Overall, if your result for this question takes more than around 1 second to be calculated, it probably means that you do not have an efficient implementation. The computation for this question is the same as for question 9 but with much larger input files in order to test the efficiency.