Higher School of Economics
(National Research University)
Faculty of Computer Science

Home Assignment
Course: "Ordered sets for Data Analysis"

Lazy Formal Concept Analysis

Student: Ryabykin Alexey
Professor: Kuznetsov Sergey Olegovich

Grade: _

Moscow, 2022

Table of Contents

1. Task description

This homework serves as an entry point for students into the data science world. So it introduces students to three main topics:

1. Typical machine learning project:
   The pipeline of loading a dataset, feature engineering, designing a new predictive algorithm, results comparison;
2. Lazy learning:
   Predicting labels for small or rapidly changing data;
3. Rule-learning (on part with FCA):
   Viewing data as binary descriptions of objects (instead of points in a space of real numbers).

ToDo list:

- ☒ Find a dataset for binary classification;
- ☒ Describe scaling (binarization) strategy for the dataset features;
- ☒ Describe quality measure best suited for the dataset'
- ☒ Adapt the pipeline from lazyfca.ipynb to your task;
- ☒ Improve the baseline algorithm:
  - ☒ Achieve better asymptotic complexity;
  - ☒ Improve the rruntime of examples comparison;
  - ☒ Rewrite the intersections of sets into the intersection of the corresponding bitmasks;
  - ☒ Modify the algorithm to achieve better qualtiy of predictions.
- ☒ Compare the proposed algorithm with at least 3 popular rule-based models;
- ☒ Test the proposed algorithm on more datasets (at least 3 others);
- ☐ Incorporate pattern structure into the pipeline to avoid scaling (binarization).

2. The chosen datasets

I have chosen three datasets to check the initial algorithm and my proposed one perfomances. They all belongs to UCI Machine Learning Repository. The first one is Car dataset (1728 instances). You can watch the description of variables below:

- target: car acceptabiltiy;

  It is not a binary feature (possible values: "unacc", "acc", "good", "v-good"). But two classes cover 92% of the dataset. I have merely dropped the least classes and obtained binary classification task. But it is still very bad balanced dataset. That is why the preffered metric to estimate the algorithm perfomances had been chosen f1 score.

- buying: buying price;

  Possible values: "v-high", "high", "med", "low". Applying the one-hot encoding is the best strategy to make binarization to this feature;

- maint: price of the maintenance;

  Possible values: “v-high”, “high”, “med”, “low”. The same strategy for binarization as for the previous feature.

- doors: number of doors;

  Possible values: “2”, “3”, “4”, “5more”. The same strategy.

- persons: capacity in terms of persons to carry;

  Possible values: “2”, “3”, “4more”. One-hot encoding.

- lug_boot: the size of luggage boot;

  Possible values: “small”, “med”, “big”.

- safety: estimated safety of the car;

  Possible values: “low”, “med”, “high”.

The second dataset is the Mushrooms dataset. The description of the variables below:

- cap-shape:

  bell = b, conical = c, convex = x, flat = f, knobbed = k, sunken = s;

- cap-surface:

  fibrous = f, grooves = g, scaly = y, smooth = s;

- cap-color:

  brown = n, buff = b, cinnamon = c, gray = g, green = r, pink = p, purple = u, red = e, white = w, yellow = y;

- bruises?:

  bruises = t, no = f;

- odor:

  almond = a, anise = l, creosote = c,fishy = y, foul = f, musty = m, none = n, pungent = p, spicy = s;

- gill-attachment:

  attached = a, descending = d, free = f, notched = n;

- gill-spacing:

  close = c, crowded = w, distant = d;itemize

- gill-size:

  broad = b, narrow = n;

- gill-color:

  black = k, brown = n, buff = b, chocolate = h, gray = g, green = r, orange = o, pink = p, purple = u, red = e, white = w, yellow = y;

- stalk-shape:

  enlarging = e, tapering = t;

- stalk-root:
  bulbous = b, club = c, cup = u, equal = e, rhizomorphs = z, rooted = r, missing = ?;
- stalk-surface-above-ring:
  fibrous = f, scaly = y, silky = k, smooth = s;
- stalk-surface-below-ring:
  fibrous = f, scaly = y, silky = k, smooth = s;
- stalk-color-above-ring:
  brown = n, buff = b, cinnamon = c, gray = g, orange = o, pink = p, red = e, white = w, yellow = y;
- stalk-color-below-ring:
  brown = n, buff = b, cinnamon = c, gray = g, orange = o, pink = p, red = e, white = w, yellow = y;
- veil-type:
  partial = p, universal = u;
- veil-color:
  brown = n, orange = o, white = w, yellow = y;
- ring-number:
  none = n, one = o, two = t;
- ring-type:
  cobwebby = c, evanescent = e, flaring = f, large = l, none = n, pendant = p, sheathing = s, zone = z;
- spore-print-color:
  black = k, brown = n, buff = b, chocolate = h, green = r, orange = o, purple = u, white = w, yellow = y;
- population:
  abundant = a, clustered = c, numerous = n, scattered = s, several = v, solitary = y;
- habitat:
  grasses = g, leaves = l, meadows = m, paths = p, urban = u, waste = w, woods = d.

The last dataset is the United States Congressional Voting Records Database. The values description is presented below:

- target: 2 (democrat, republican)
- handicapped-infants: 2 (y,n)
- water-project-cost-sharing: 2 (y,n)
- adoption-of-the-budget-resolution: 2 (y,n)
- physician-fee-freeze: 2 (y,n)
- el-salvador-aid: 2 (y,n)
- religious-groups-in-schools: 2 (y,n)
- anti-satellite-test-ban: 2 (y,n)
- aid-to-nicaraguan-contras: 2 (y,n)
- mx-missile: 2 (y,n)
- immigration: 2 (y,n)
- synfuels-corporation-cutback: 2 (y,n)
- education-spending: 2 (y,n)
- superfund-right-to-sue: 2 (y,n)

- crime: 2 (y,n)
- duty-free-exports: 2 (y,n)
- export-administration-act-south-africa: 2 (y,n)

3. Initial algorithm

The following algorithm is used in the homework as a baseline algorithm for lazy FCA-based classification. And it is called "Generators framework".

Description

Assume that we want to make a prediction for description $x \subseteq M$ given the set of training examples $X_{\text{train}} \subseteq 2^M$ and the labels $y_x \in \{\text{False}, \text{True}\}$, corresponding to each $x \in X_{\text{train}}$.

First, we split all examples X_{train} to positive X_{pos} and negative X_{neg} examples:

$$X_{\text{pos}} = \{x \in X_{\text{train}} \mid y_x \text{ is True}\}, \quad X_{\text{neg}} = X \backslash X_{\text{pos}}.$$

To classify the description x we follow the procedure:

1. Count the number of counterexamples for positive examples:

   For each positive example $x_{\text{pos}} \in X_{\text{pos}}$ we compute the intersection $x \cap x_{\text{pos}}$. Then, we count the counterexamples for this intersection, that is the number of negative examples $x_{\text{neg}} \in X_{\text{neg}}$ containing intersection $x \cap x_{\text{pos}}$;

2. Dually, count the number of counterexamples for negative examples.

Finally, we compare the average number of counterexamples for positive and negative examples. We classify as being positive if the number of counterexamples for positive examples is smaller the one for negative examples.

Results

The table with results and graphics of scores and time information are presented below:

Result / Dataset	accuracy	f1 score	time	time (fixed train)
Car	0.78	0.178	37.3 s	839 ms
Mushrooms	0.93	0.87	2h 23min 44s	3 min 13 s
Congress voting	0.6	0.46	303 ms	202 ms

Table 1 – Main results for the initial algorithm

Results for the Car dataset are presented on the Figure 1

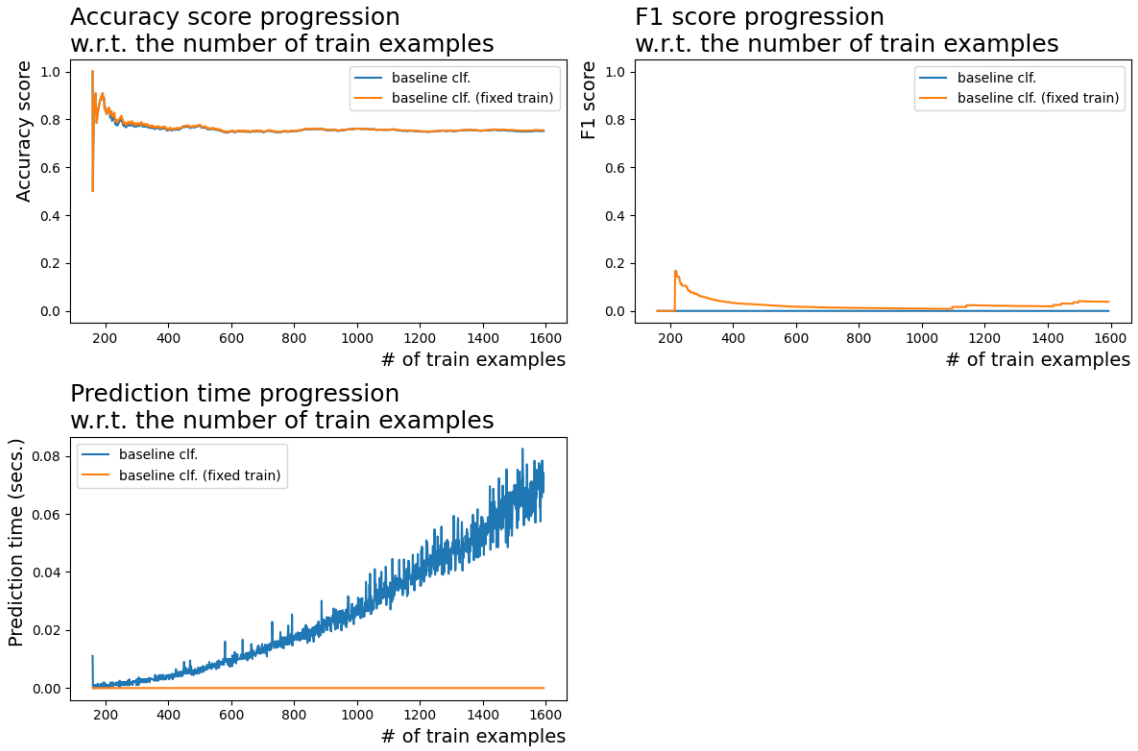

Figure 1 – Results for Car dataset (initial algorithm)

You can see on the Figure 2 the results of the initial algorithm applied to the Mushrooms dataset.

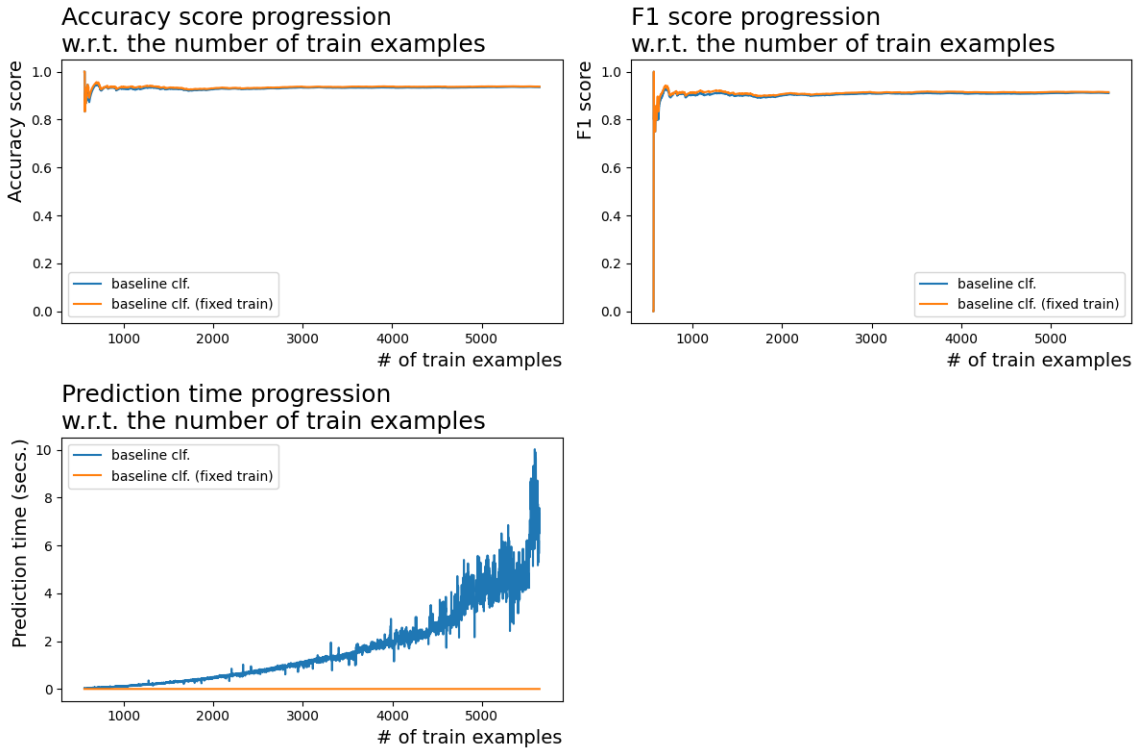

Figure 2 – Results for Mushrooms dataset (initial algorithm)

The Figure 3 contains the results of the initial algorithm appication to the Congress Voting dataset.

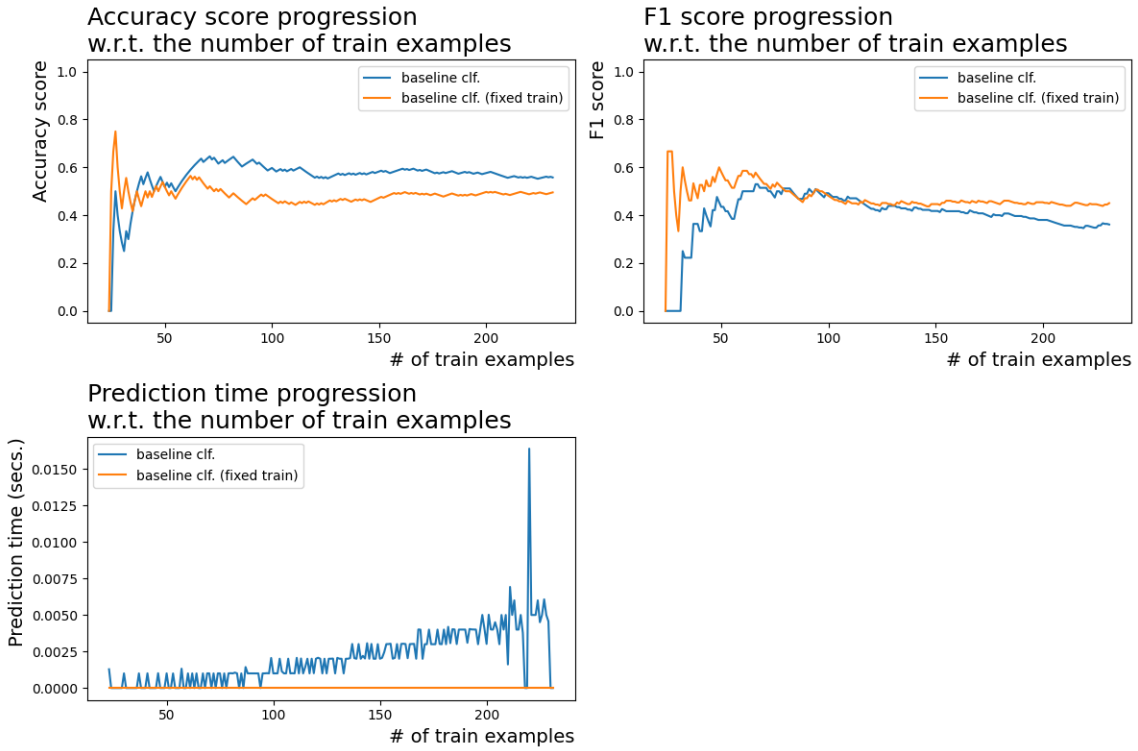

Figure 3 – Results for Congress Voting dataset (initial algorithm)

4. Proposed improvements and comparisons

The core idea is the same as for the initial algorithm. But the chosen path to calculate the intersections is different with the previous one. The procedure can be expressed in terms of matrix multiplication of bitmasks that had been obtained after binarization. The proposed code you can see below:

```python
def predict_with_generators(
  x: set, X_train: List[set], Y_train: List[bool],
  min_cardinality: int = 1
) -> bool:
  X_pos = np.array(X_train[np.array(Y_train, dtype=bool)], dtype=int)
  X_neg = np.array(X_train[~np.array(Y_train, dtype=bool)], dtype=int)

  idxs = X_pos @ x >= min_cardinality
  n_counters_pos = ((X_neg @ (X_pos & x)[idxs, :].T) == (X_pos @ x)[idxs]).sum()

  idxs = X_neg @ x >= min_cardinality
  n_counters_neg = ((X_pos @ (X_neg & x)[idxs, :].T) == (X_neg @ x)[idxs]).sum()

  perc_counters_pos = n_counters_pos / (n_counters_neg + n_counters_pos)
  perc_counters_neg = n_counters_neg / (n_counters_pos + n_counters_neg)
  prediction = perc_counters_pos < perc_counters_neg
  return prediction
```

Listing 1 – Python code

Also, there is some bad influence of proposed in the initial algorithm normalization of counts of counterexamples. In case of bad balanced target values the prediction of the algorithm can be exclusively in the direction of the class, which is larger. I removed the normalization; it improved the quality of the selected datasets. However, on the tic-tac-toe dataset from the baseline, the quality dropped slightly. It can be concluded that normalization is useful, but it should be different. The logic here is that, for example, with 20% of

negative objects in the training sample, finding a negative counterexample is more valuable than a positive one. This must be taken into account.

Result / Dataset	accuracy	f1 score	time	time (fixed train)
Car Dataset	0.78	0.78	8.97 s	563 ms
Mushrooms	0.93	0.91	29 min	43 s
Congress voting	0.92	0.9	120 ms	58 ms

Table 2 – Main results for the proposed algorithm

Results for the Car dataset are presented on the Figure 4

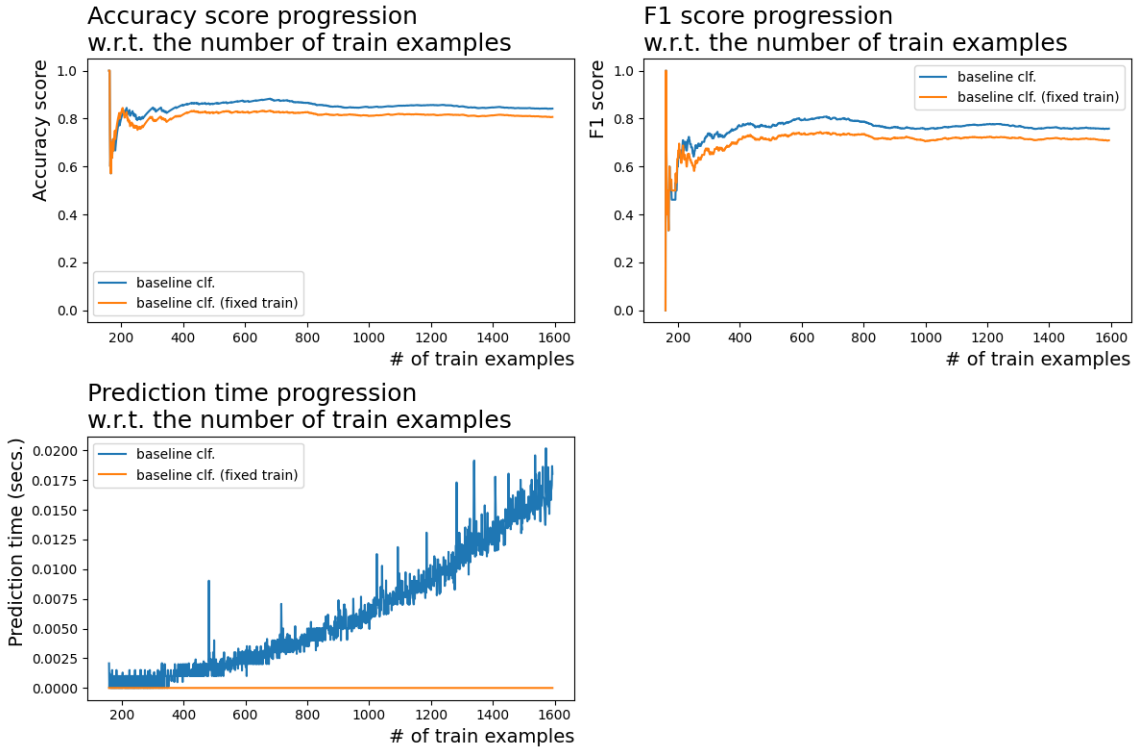

Figure 4 – Results for Car dataset (improved algorithm)

You can see on the Figure 5 the results of the improved algorithm applied to the Mushrooms dataset. Let's compare it with the initial algorithm. So, for the Car dataset you can see difference in the Table 3

Result / CLF	accuracy	f1 score	time	time (fixed train)
Initial algorithm	0.78	0.178	37.3s	839ms
Proposed algorithm	0.78	0.78	8.97s	563ms

Table 3 – Comparison of two algorithms on the Car Dataset

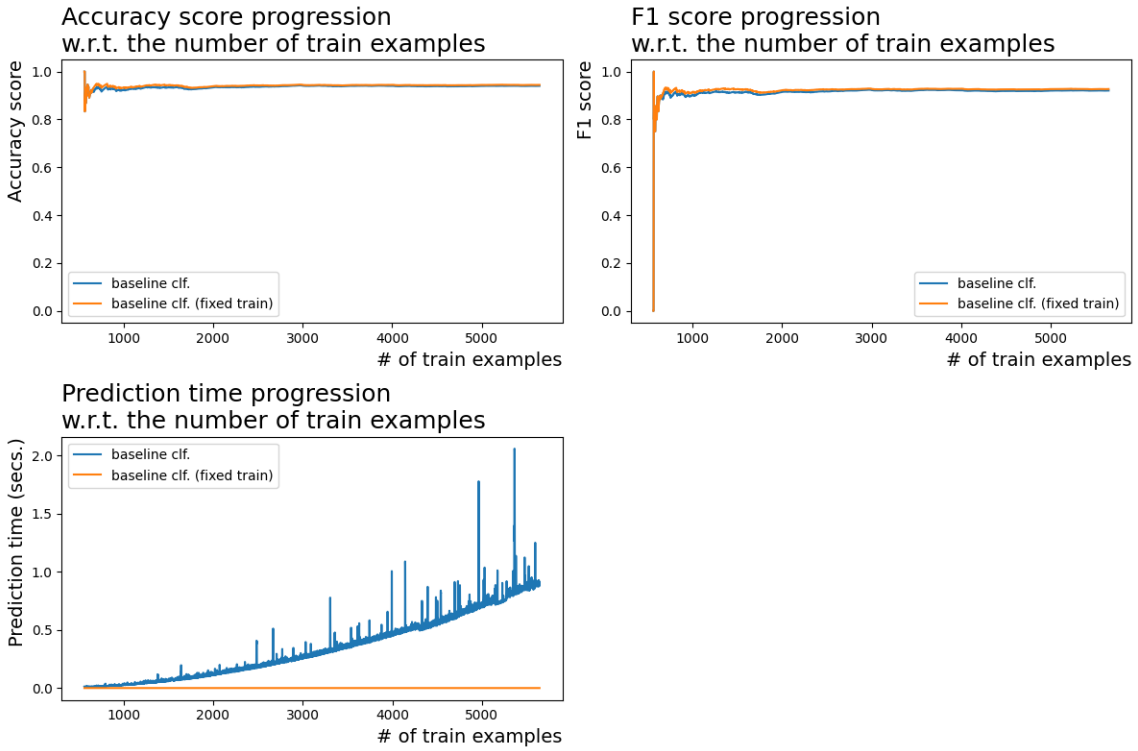

Figure 5 – Results for Mushrooms dataset (improved algorithm)

Comparison with the initial algorithm is presented below:

Result / CLF	accuracy	f1 score	time	time (fixed train)
Initial algorithm	0.93	0.9	2h 23min 44s	3min 13s
Proposed algorithm	0.93	0.91	29min 6s	43.5s

The Figure 6 contains the results of the improved algorithm application to the Congress Voting dataset. Compartison with the initial algorithm is presented below:

Result a / CLF	accuracy	f1 score	time	time (fixed train)
Initial algorithm	0.93	0.9	443ms	17ms
Proposed algorithm	0.93	0.91	120ms	15.6ms

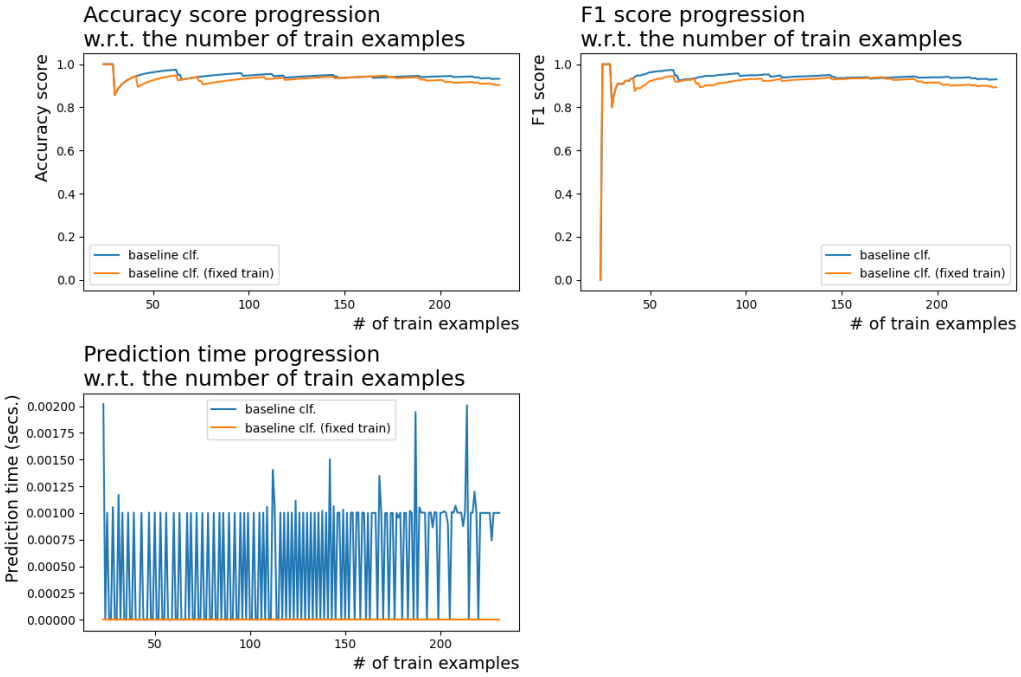

Figure 6 – Results for Congress Voting dataset (improved algorithm)

5. Comparison with other rule-based models

The chosen models to compare initial algorithm and algorithm with proposed improvements:

1. **CatBoostClassifier;**
2. **XGBRFClassifier;**
3. **LGBMClassifier;**
4. **DecisionTreeClassifier;**
5. **RandomForestClassifier.**

CLF / Data	CatBoost Classifier	DecisionTree Classifier	RandomForest Classifier	XGBRF Classifier	LGBM Classifier	Initial algorithm	Improved algorithm
Dataset Car	0.958	0.816	0.743	0.796	0.813	0.78	0.78
Dataset Mushrooms	1	0.926	0.917	0.932	0.927	0.93	0.93
Dataset Congress Voting	0.952	0.966	0.961	0.971	0.957	0.7	0.92

Table 4 – Average accuracy scores through cross-validation for the chosen models

CLF / Data	CatBoost Classifier	DecisionTree Classifier	RandomForest Classifier	XGBRF Classifier	LGBM Classifier	Initial algorithm	Improved algorithm
Dataset Car	0.921	0.838	0.829	0.844	0.865	0.18	0.78
Dataset Mushrooms	1	0.937	0.930	0.944	0.941	0.9	0.91
Dataset Congress Voting	0.949	0.965	0.961	0.970	0.957	0.66	0.9

Table 5 – Average f1 scores through cross-validation for the chosen models

CLF / Data	CatBoost Classifier	DecisionTree Classifier	RandomForest Classifier	XGBRF Classifier	LGBM Classifier	Initial algorithm	Improved algorithm
Dataset Car	16s	90ms	480ms	220ms	75ms	839ms	560ms
Dataset Mushrooms	17s	440ms	850ms	670ms	156ms	3min 13s	43.5s
Dataset Congress Voting	1.1s	70ms	360ms	90ms	65ms	303ms	58ms

Table 6 – Average elapsed time for the chosen models