Lazy FCA Report

Prikhno M.A.

12 December 2022

1. Dataset

The dataset that we are going to explore is Body signal of smoking. The target of our classification is to find smokers based on their vital signs and medical data.

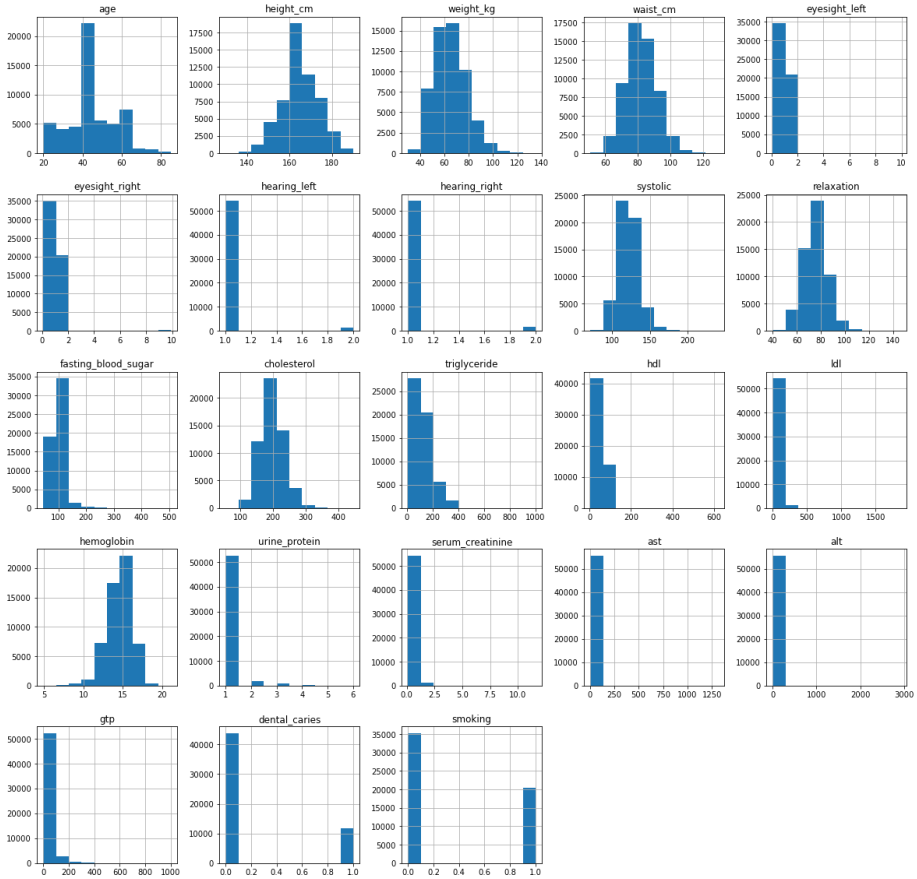
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55692 entries, 0 to 55691
Data columns (total 26 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   gender               55692 non-null  object 
 1   age                  55692 non-null  int64  
 2   height_cm            55692 non-null  int64  
 3   weight_kg            55692 non-null  int64  
 4   waist_cm             55692 non-null  float64
 5   eyesight_left        55692 non-null  float64
 6   eyesight_right       55692 non-null  float64
 7   hearing_left         55692 non-null  float64
 8   hearing_right        55692 non-null  float64
 9   systolic             55692 non-null  float64
 10  relaxation           55692 non-null  float64
 11  fasting_blood_sugar  55692 non-null  float64
 12  cholesterol          55692 non-null  float64
 13  triglyceride         55692 non-null  float64
 14  hdl                  55692 non-null  float64
 15  ldl                  55692 non-null  float64
 16  hemoglobin           55692 non-null  float64
 17  urine_protein        55692 non-null  float64
 18  serum_creatinine     55692 non-null  float64
 19  ast                  55692 non-null  float64
 20  alt                  55692 non-null  float64
 21  gtp                  55692 non-null  float64
 22  oral                 55692 non-null  object 
 23  dental_caries        55692 non-null  int64  
 24  tartar               55692 non-null  object 
 25  smoking              55692 non-null  int64  
dtypes: float64(18), int64(5), object(3)
memory usage: 11.0+ MB
```

The dataset consists of both categorical and numerical data. It has 55962 rows and 26 columns. The value that we need to predict, that is, y, is in column

"smoking".

We need to decide upon binarization strategy, because the algorithm we use requires binary data: each value in columns should be either True or False. The categorical data can be easily binarized using pandas method `pd.get_dummies()`. The numerical data should be explored first, the histogram of numerical values is below.



Some values (e.g. "gtp", "dental_caries", "ldl") are densely clustered around one point. We need an intelligent binarization strategy that will divide the numerical data to smart bins according to the percentage of values in the bin. For example, in each of five bins should be 20% of data. Such binarization method is `pd.qcut(q=3, duplicates='drop')`, which creates 3 bins and drops them if their edges intersect. After numerical data is distributed between bins and thus turned into categorical, we use `pd.get_dummies()` to binarize it. The resulting dataset has 55962 rows and 63 columns. Because lazy FCA algorithm takes quite some time, we clip the dataset and leave only 1000 rows.

2. Quality measure

Before we start predicting, we need to pick metrics to measure quality. We picked two metrics, accuracy score and F1 score.

Accuracy score is a basic metric, which formally is defines thus:

$$Accuracy = \frac{Correct_predictions}{Total_predictions}$$

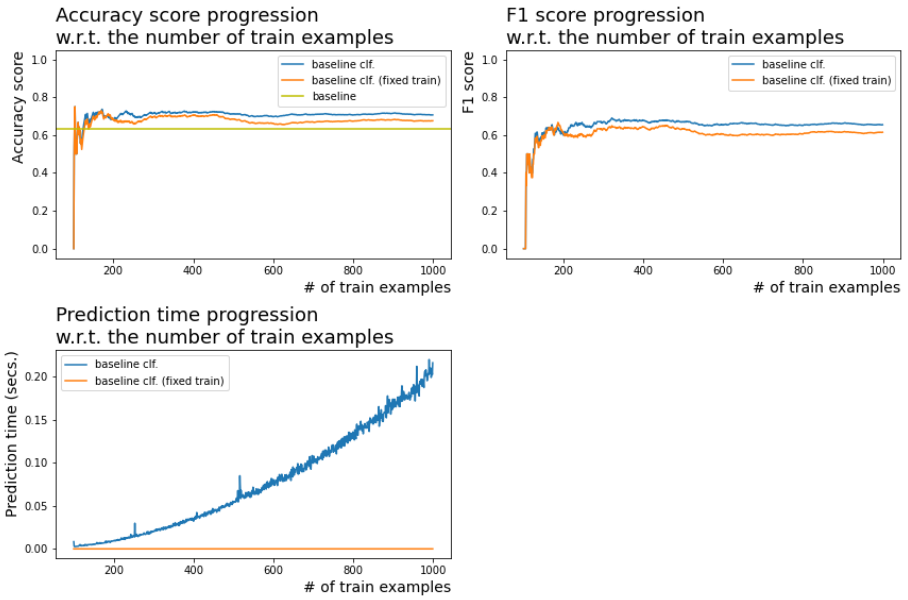
.

This metric is common and can be used for our dataset because there's no imbalance between classes: the amount of "smokers" is close to the amount of "non-smokers".

F1-score is the harmonic mean of precision ($\frac{TP}{TP+FP}$) and recall ($\frac{TP}{TP+FN}$). It minimizes FN, FalseNegative result, which is the most harmful for our task, when someone is a smoker but was classified as a non-smoker.

3. Prediction function optimization

First, let's use basic prediction function from lazy_pipe package. Results are displayed below.



Resulting scores are such: accuracy score: 0.7063403781979978, F1 score: 0.6544502617801047. We compare accuracy score to the baseline value, which is the percentage of "smoking=False" values in column "smoking". Prediction took 1min 9s with train updates and 2.28 s without train updates.

It took us quite some time to predict 1000 values. Let's try and speed up the process by using numpy package and converting lists of sets to `np.array`. The original part of function is displayed below.

```
X_pos = [x_train for x_train, y in zip(X_train, Y_train) if y]
X_neg = [x_train for x_train, y in zip(X_train, Y_train) if not y]

n_counters_pos = 0  # number of counter examples for positive intersections
for x_pos in X_pos:
    intersection_pos = x & x_pos
    if len(intersection_pos) < min_cardinality:  # the intersection is too small
        continue

    for x_neg in X_neg:  # count all negative examples that contain intersection_pos
        if (intersection_pos & x_neg) == intersection_pos:
            n_counters_pos += 1

n_counters_neg = 0  # number of counter examples for negative intersections
for x_neg in X_neg:
    intersection_neg = x & x_neg
    if len(intersection_neg) < min_cardinality:
        continue

    for x_pos in X_pos:  # count all positive examples that contain intersection_neg
        if (intersection_neg & x_pos) == intersection_neg:
            n_counters_neg += 1

perc_counters_pos = n_counters_pos / len(X_pos)
perc_counters_neg = n_counters_neg / len(X_neg)

prediction = perc_counters_pos < perc_counters_neg
return prediction
```

And the modified function is next.

```
X_pos = X_train[Y_train]
X_neg = X_train[~Y_train]

intersections_pos = x.reshape(1, -1) & X_pos
intersections_pos = intersections_pos[intersections_pos.sum(axis=1) >= min_cardinality]
intersections_pos = intersections_pos @ (~X_neg.T)
n_counters_pos    = (intersections_pos == 0).sum()

intersections_neg = x.reshape(1, -1) & X_neg
intersections_neg = intersections_neg[intersections_neg.sum(axis=1) >= min_cardinality]
intersections_neg = intersections_neg @ (~X_pos.T)
n_counters_neg    = (intersections_neg == 0).sum()

perc_counters_pos = n_counters_pos / len(X_pos)
perc_counters_neg = n_counters_neg / len(X_neg)

prediction = perc_counters_pos < perc_counters_neg
return prediction
```

Because we use `np.array`, we can use arrays as indices. Thus, `X_train[y_train]` is equivalent to `[x_train for x_train, y in zip(X_train, Y_train) if y]`, and the same for `~y_train` and `if not y`.

Next, the original algorithm loops through all rows in `X_pos` and collects intersections with `x`. We can use the same bitwise operator `&` to compare `x` to each row of `X_pos` via `x.reshape(1, -1) & X_pos`.

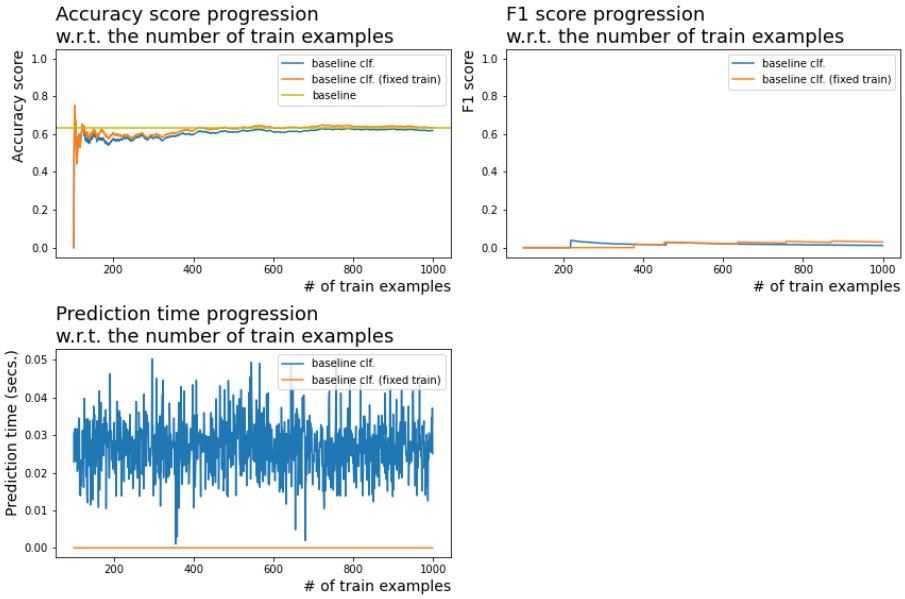
Then, only those intersections are left that are longer than `min_cardinality`. Because our arrays consist of True and False values, sum of values in such an

array is equal to amount of True values, which is the length of an intersection.

Next, the original algorithm loops through values of `X_neg` and counts all negative examples that have found that intersection inside. That's a bit tricky to translate. Suppose that the row a is contained in b, that is, all True values in a are also in b. Then the inverted row where True are changed False and the other way around has False everywhere row a has True. Therefore, their product is scalar zero. That's exactly what we calculate to get `n_counters_pos`.

The same is done for `n_counters_neg` and the `prediction` is calculated as in the original algorithm.

Let's check predictions using modified algorithm. Results are displayed below.



Resulting scores are such: accuracy score: 0.6184649610678532, F1 score: 0.011527377521613834. Prediction took 24.2 s with train updates and 529 ms without train updates. While the prediction rate is higher and algorithm spent less time on classification, the results are not so good. Accuracy score is approximate to the baseline, and F1-score is very low, which is a problem for our task. We have specified that we need good F1-score because we do not wish to make FalseNegative mistakes.

4. Model comparison

Last, let's compare our Lazy algorithm to some common rule-based models like Decision Tree, Random Forest and CatBoost classifier. The resulting accuracy scores are: DecisionTreeClassifier: 0.68, RandomForestClassifier: 0.71, CatBoostClassifier: 0.73. As we can see, original Lazy classifier is better than

the Decision Tree classifier and has slightly less scores than Random Forest and CatBoost classifiers.

5. Conclusion

We have prepared and binarized the chosen dataset for our task. We successfully used original algorithm to get decent scoring, enhanced it with numpy arrays to achieve better runtime but lost accuracy. We compared Lazy algorithm to some popular models and discovered that it's predictions accuracy is close to that of popular rule-based models.