# Lazy binary classifier based on Formal Concept Analysis

Vitaliy Pozdnyakov

December 2019

**Abstract**

This report was prepared as part of the individual project of the course "Ordered sets in data analysis". The project is dedicated to the development of a lazy binary classifier based on methods of Formal Concept Analysis (FCA). Usually, the work of the classifier can be separated into two steps: the selection of patterns in the training sample (training) and, in fact, classification. Since not all the information from the training sample is used during the classification, but only an aggregated part of it, the classification occurs quite quickly. The lazy classification method differs in that the classifier uses the entire training sample without advance training, which takes much longer, but can increase the accuracy of the classification. The first section will describe the data representation model and classification method. The second — considers the implementation of the method in Python and its algorithmic complexity. The third — describes the data sets on which the classification quality was tested. The fourth — describes the metrics for testing the quality of the classifier, presents the results of testing and compares it with other classifiers — decision tree and logistic regression. Finally, the fifth section will present conclusions about the work of the classifier.

## 1 Model and classification method

Represent the training data set as ordered sets (context) $K = (G, M, I)$, where $G$ — the set of objects, $M$ — the set of attributes, $I$ — the set of relations between $G$ and $M$. We can define $I$ as a set $\{(g, m)|g \in G, m \in M, gIm\}$. One of all attributes is called the target attribute $m_t$. Then the initial set of objects are divided into two sets according to the rule:

$$G_+ = \{g \in G|(g, m_t) \in I\}$$

$$G_- = \{g \in G|(g, m_t) \notin I\}$$

Now consider the set of test objects $G_\tau$, which consist information about relations of all attributes except the target. Objective of the classifier — define $(g_\tau, m_t) \in I$ or $(g_\tau, m_t) \notin I$.

In the next sections we will use the "prime" operator $(\cdot)'$ from FCA theory, which define as:

$$A' = \{m \in M | (g, m) \in I, \forall g \in A\}$$
$$B' = \{g \in G | (g, m) \in I, \forall m \in B\}$$

Call the set $g'$ as the description of the object $g$. Then the algorithm can be represented as:

1. An arbitrary object $g_\tau$ from the test sample $G_\tau$ is fed to the classifier input.

2. We select the $i$-th element $g_i^+$ from the set $G_+$ and calculate the intersection of its description with the description $g_\tau$, that is

$$\Delta_i^+ = (g_i^+)' \cap (g_\tau)'$$

3. The proportion of objects $S_i^+$ from the set $G_-$ for which the resulting intersection is included in the descriptions of these objects is calculated. That is

$$S_i^+ = \frac{|\{g^- \in G_- | \Delta_i^+ \subseteq (g^-)'\}|}{|G_-|}$$

4. If the share of $S_i^+$ does not exceed the pre-selected threshold value $p \in [0, 1]$, then the power of $i$-th intersection $\Delta_i^+$ is taken into account in the support for a positive decision:

$$Support^+ = \frac{\sum_i |\Delta_i^+| \text{ if } S_i^+ < p}{|G_+|}$$

   This sums up the support for all elements from the positive set.

5. Similarly, support for a negative decision is calculated

$$Support^- = \frac{\sum_i |\Delta_i^-| \text{ if } S_i^- < p}{|G_-|}$$

6. The classifier evaluates the rule

$$Prediction = \begin{cases} g_\tau \in G_+ \text{ if } Support^+ > Support^- \\ g_\tau \in G_- \text{ if } Support^+ < Support^- \end{cases}$$

We should also consider the situation when $Support^+ = Support^-$. In this case, one (pre-selected) of the three strategies is used:

1. The class is chosen randomly with a probability of each class 0.5

2. Always choose a positive class

3. Always choose a negative class

In addition, consider a modification of the method in which the classification is not used for the entire training sample, but only a random fraction of it. Then at all classification steps, instead of $G_+$ and $G_-$ will be used respectively:

$$G_+^{rand} = \{g_i | g_i \in G_+, i \in X_+\}$$
$$G_-^{rand} = \{g_i | g_i \in G_-, i \in X_-\}$$

where $X_+$ — a random set of indexes of elements from $G_+$ and $k < |G_+|$, each of which is equally likely to fall into this set. Similarly, the set $X_-$ for $l < |G_-|$ indexes of elements from $G_-$ is defined. The values $k$ and $l$ are selected in advance before the classification begins. This modification allows to significantly reduce the calculation time. Although the *Prediction* estimate loses its accuracy, it remains unbiased by using random equally probable selection of elements.

# 2 Python implementation and complexity estimation

Development was based on the open library scikit-learn, using inheritance of base classes:

1. BaseEstimator — base class for objects that implement statistical estimates
2. ClassifierMixin — base class for classifiers

This allowed to focus on implementing classification rules, while methods for quality assessment and interfaces for loading training samples and unloading predictions were already ready on the library. Let's consider the main interfaces of the classifier of the scikit-learn library:

- fit — the method is used to train the model. A training sample is sent to the input, on the basis of which the classification rules are formed. Since no training is required in our case, this method is used to store the entire training sample in the classifier object. In addition, some optimization procedures are performed, which will be mentioned later.
- predict — the method is used for classification. The input is a sample, for each element of which you want to predict the value of the target attribute.
- score — the method is used to calculate the accuracy metric. More information about metrics will be described later. The input is a sample for prediction, as well as the true values of the target attribute. By comparing the predicted and true values, the accuracy metric is calculated.

Now consider the algorithmic complexity of the classification algorithm. Let us denote the number of objects and attributes respectively

$$m = |G|$$

$$n = |M|$$

Since the number of objects in a positive and negative sample can not differ from the total number of elements by more than some constant, that is, $\Theta(|G_+|) = \Theta(|G_-|) = \Theta(|G|)$, then we will consider only the complexity of the calculation of $support^+$, bearing in mind that the calculation of $support^-$ has the same asymptotic complexity.

First, consider calculating the intersection of $\Delta_i^+$. In the worst case, when crossing the description of two identical objects, we get all the set of attributes, that is, the complexity of such an operation — $\Theta(n)$.

Next, to calculate the parameter $S_i^+$, we need to check whether the intersection is included in the description of each object of the negative selection, that is, the complexity of the operation taking into account the previous one — $\Theta(n \cdot n \cdot m)$.

Next, to calculate support for $Support^+$ , we need to calculate the values of $S_i^+$ and $\Delta_i^+$ for each object in $G_+$. The calculation of power for each $\Delta_i^+$ can be performed directly during the calculation of this intersection, so this operation does not affect the overall complexity. That is, the complexity of the operation taking into account the previous — $\Theta(m \cdot n \cdot n \cdot m)$.

Finally, the overall complexity of the algorithm (in time):

$$Complexity = \Theta(m^2 \cdot n^2)$$

However, given the fact that the number of attributes is usually significantly less than the number of objects, we can perform the following optimization. Before starting the classification for each attribute $m_i$ we get "index" — a set of objects for which $(g, m_i) \in I$, that is

$$Index_i = \{g | (g, m_i) \in I\}$$

This operation must be performed for a positive and negative class and thus for each attribute $m_i$ we get a pair of indexes $Index_i^+$ and $Index_i^-$.

Now we can use the formula to calculate $S_i^+$:

$$S_i^+ = \frac{\left|\left\{g | g \in \left(\bigcap_{j \in \Delta_i^+} Index_j^-\right)\right\}\right|}{|G_-|}$$

Thus, the overall complexity of the algorithm after optimization:

$$Complexity = \Theta(m \cdot n^3)$$

Thanks to this, the classification time was reduced by about 3 times (on the test data set: 31 sec. before optimization, 9 sec. after).

# 3 Description of the selected data sets

Two data sets were selected for the project:

1. Tic-Tac-Toe Endgame Data Set from the repository UCI Machine Learning Repository. The data set shows the States of the game field of the Tic-Tac-Toe game after the end of the game.

2. Heart Disease Data Set from the UCI Machine Learning Repository. The data set presents data on the presence or absence of heart disease in 303 patients.

A data set is a table where the rows correspond to observations and the columns correspond to feature values. Note that the formal context $K$ can be represented as a similar table, where the rows correspond to objects (observations), and the columns — attributes (features). In this case, each $i$-th row and $j$-th column will have a value of 1 if $(g_i, m_j) \in I$ and 0 if $(g_i, m_j) \notin I$. Such a set of data can be called binary, and features that can take only two values — binary. Since the classifier works only with binary features, it is necessary to transform the selected data sets to the binary form (scaling). Different scaling strategies were chosen for (1) and (2).

First, consider (1). Here we use the identifiers of the field cells of the form v1, v2,..., v9, where v1 is the upper-left cell, and v9 is the lower-right cell. Each attribute can take 3 values: x — cross, o — toe, b — empty cell. The target attribute v10 contains information about the victory (Positive) or loss (Negative) of the player who plays with crosses. Since the location of the crosses on the field is most important for the player to win, the values in each feature were replaced with 1 if there was a cross and 0 otherwise. A replacement was performed for the target attribute: 1 if "Positive" was specified and 0 otherwise. Thus, a binary data set was obtained that the classifier can work with.

Now consider (2). Different information is provided for each patient, such as gender, age, and blood pressure. As a target feature, a binary one was selected: the presence or absence of heart disease. All these features can be divided into three groups, each of which has been applied its own way of scaling:

1. Binary (for example, gender, presence of disease). For such features, the values were replaced with 0 and 1. The specific value for which we need to specify 0 (or 1) does not play a role, so it was chosen randomly.

2. Categorical (for example, the type of pain). For each such feature, new ones were added, one for each value. The value 1 was set if the given attribute had this value. After that, the original categorical features were removed.

3. Numeric (for example, blood pressure). For each such feature, an interval of acceptable values of the form $[\min(m_i), \max(m_i)]$ was calculated. Further, each interval was proportionally divided into 5 intervals, each of which was added as a new feature. The value 1 was set if the condition "greater than or equal to" value from the original attribute was met for the object. After that, the original numeric characters were removed.

Thus, a binary data set was obtained that the classifier can work with.

# 4 The results of testing the classifier

The testing used a cross-validation approach — each data set is randomly divided into a training and test sample 10 times, then for each partition, the test sample is classified, compared with the true values, and quality metrics are calculated.

The model parameters were selected as:

- **Bias** — the decision to make if $Support^+ = Support^-$. There are three options: Positive (always set a positive class), Negative (always set a negative class), and Random (set a random class).
- **Threshold** — threshold value for the parameter $p \in [0, 1]$, which is used in the formula
$$Support^+ = \frac{\sum_i |\Delta_i^+| \text{ if } S_i^+ < p}{|G_+|}$$
- **Random** — True to enable a mode that uses only a randomly selected portion of the training sample, False — to disable the mode.
- **Sample share** — if **Random** mode is used, this parameter sets the percentage of entries from the positive and negative set. Valid values in the range $[0, 1]$.

Widely known metrics were used to evaluate the quality of the model:

$$Recall = \frac{tp}{tp + fn}$$

$$Precision = \frac{tp}{tp + fp}$$

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

The classification time in seconds was also estimated. The test results are given below. The "#" column contains the number of the cross-validation partition.

### 4.1 Tic-Tac-Toe

| # | **Accuracy** | **Presicion** | **Recall** | **Time (sec)** |
|---|---|---|---|---|
| 1 | 1.0000 | 1.0000 | 1.0000 | 64 |
| 2 | 1.0000 | 1.0000 | 1.0000 | 59 |
| 3 | 1.0000 | 1.0000 | 1.0000 | 62 |
| 4 | 1.0000 | 1.0000 | 1.0000 | 54 |
| 5 | 1.0000 | 1.0000 | 1.0000 | 52 |
| 6 | 1.0000 | 1.0000 | 1.0000 | 53 |
| 7 | 1.0000 | 1.0000 | 1.0000 | 72 |
| 8 | 1.0000 | 1.0000 | 1.0000 | 65 |
| 9 | 1.0000 | 1.0000 | 1.0000 | 61 |
| 10 | 1.0000 | 1.0000 | 1.0000 | 56 |
| **Average** | **1.0000** | **1.0000** | **1.0000** | **60** |

Tab. 1. Lazy classifier quality metrics for parameters
Threshold = 0.000001, Random = False, Bias = Negative

| # | **Accuracy** | **Presicion** | **Recall** | **Time (sec)** |
|---|---|---|---|---|
| 1 | 0.9677 | 0.9531 | 1.0000 | 22 |
| 2 | 0.9540 | 0.9273 | 1.0000 | 21 |
| 3 | 0.9400 | 0.9155 | 1.0000 | 21 |
| 4 | 0.9888 | 0.9888 | 1.0000 | 20 |
| 5 | 0.9663 | 0.9538 | 1.0000 | 18 |
| 6 | 1.0000 | 1.0000 | 1.0000 | 18 |
| 7 | 1.0000 | 1.0000 | 1.0000 | 25 |
| 8 | 0.9907 | 0.9865 | 1.0000 | 25 |
| 9 | 0.9903 | 0.9859 | 1.0000 | 23 |
| 10 | 0.9780 | 0.9672 | 1.0000 | 21 |
| **Average** | **0.9776** | **0.9678** | **1.0000** | **21** |

Tab. 2. Lazy classifier quality metrics for parameters
Threshold = 0.000001, Random = True, Sample share = 0.3, Bias = Negative

| # | **Accuracy** | **Presicion** | **Recall** | **Time (sec)** |
|---|---|---|---|---|
| 1 | 0.9677 | 0.9531 | 1.0000 | 0 |
| 2 | 0.9540 | 0.9273 | 1.0000 | 0 |
| 3 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 4 | 0.9775 | 0.9831 | 0.9831 | 0 |
| 5 | 0.9888 | 1.0000 | 0.9839 | 0 |
| 6 | 0.9882 | 0.9825 | 1.0000 | 0 |
| 7 | 1.0000 | 1.0000 | 1.0000 | 0 |
| 8 | 0.9720 | 0.9730 | 0.9863 | 0 |
| 9 | 0.9903 | 1.0000 | 0.9857 | 0 |
| 10 | 0.9780 | 0.9831 | 0.9831 | 0 |
| **Average** | **0.9817** | **0.9802** | **0.9922** | **0** |

Tab. 3. Decision tree quality metrics

## 4.2 Heart Disease

| # | **Accuracy** | **Presicion** | **Recall** | **Time (sec)** |
|---|---|---|---|---|
| 1 | 0.8400 | 0.9091 | 0.7692 | 5 |
| 2 | 0.7100 | 0.8205 | 0.5926 | 5 |
| 3 | 0.8800 | 0.9245 | 0.8596 | 5 |
| 4 | 0.8700 | 0.9074 | 0.8596 | 4 |
| 5 | 0.8400 | 0.8621 | 0.8621 | 6 |
| 6 | 0.8700 | 0.9020 | 0.8519 | 6 |
| 7 | 0.8400 | 0.8333 | 0.8333 | 6 |
| 8 | 0.8400 | 0.8600 | 0.8269 | 6 |
| 9 | 0.8000 | 0.9348 | 0.7167 | 5 |
| 10 | 0.7200 | 0.8864 | 0.6290 | 5 |
| **Average** | **0.8210** | **0.8840** | **0.7801** | **5** |

Tab. 4. Lazy classifier quality metrics for parameters
Threshold = 0.9, Random = True, Sample share = 0.3, Bias = Negative

| # | **Accuracy** | **Presicion** | **Recall** | **Time (sec)** |
|---|---|---|---|---|
| 1 | 0.8300 | 0.9070 | 0.7500 | 16 |
| 2 | 0.7500 | 0.8537 | 0.6481 | 18 |
| 3 | 0.8500 | 0.9200 | 0.8070 | 19 |
| 4 | 0.8300 | 0.8846 | 0.8070 | 20 |
| 5 | 0.8000 | 0.8958 | 0.7414 | 19 |
| 6 | 0.8300 | 0.9302 | 0.7407 | 17 |
| 7 | 0.8400 | 0.8636 | 0.7917 | 18 |
| 8 | 0.8500 | 0.8491 | 0.8654 | 18 |
| 9 | 0.7900 | 0.9333 | 0.7000 | 21 |
| 10 | 0.7200 | 0.8269 | 0.6935 | 20 |
| **Average** | **0.8090** | **0.8864** | **0.7545** | **19** |

Tab. 5. Lazy classifier quality metrics for parameters
Threshold = 0.9, Random = False, Bias = Negative

| # | **Accuracy** | **Presicion** | **Recall** | **Time (sec)** |
|---|---|---|---|---|
| 1 | 0.8100 | 0.8000 | 0.8462 | 0 |
| 2 | 0.7900 | 0.7797 | 0.8519 | 0 |
| 3 | 0.9000 | 0.8615 | 0.9825 | 0 |
| 4 | 0.8600 | 0.8308 | 0.9474 | 0 |
| 5 | 0.8300 | 0.8154 | 0.9138 | 0 |
| 6 | 0.9100 | 0.9091 | 0.9259 | 0 |
| 7 | 0.8300 | 0.7719 | 0.9167 | 0 |
| 8 | 0.8000 | 0.7581 | 0.9038 | 0 |
| 9 | 0.8200 | 0.8889 | 0.8000 | 0 |
| 10 | 0.8300 | 0.8462 | 0.8871 | 0 |
| **Average** | **0.8380** | **0.8261** | **0.8975** | **0** |

Tab. 6. Logistic regression quality metrics

# 5 Conclusion

From the results obtained, it can be seen that on the Tic-Tac-Toe data set, the lazy classifier managed to achieve perfect accuracy (1.0) with the parameters Threshold = 0.000001, Random = False, Bias = Negative. The prediction time for each set averaged 60 seconds. If we use Random mode, the time is significantly reduced (up to 21 seconds), although the accuracy is still quite high — 0.98. The decision tree also achieves a high accuracy of 0.98, but the classification is almost instantaneous — on average 0 seconds, which is much better than the lazy classification in any mode.

For the Heart Disease data set, the lazy classifier achieves the best accuracy (0.82) with the parameters Threshold = 0.9, Random = True, Sample share = 0.3, Bias = Negative, and the average running time is 5 seconds. Disabling Random mode does not increase accuracy, and it also increases the calculation time. The logistic regression model achieves slightly higher accuracy (0.83), but it works much faster — the average running time is 0 seconds.

We can conclude that the methods of lazy classification are guaranteed to work longer than the usual classification models — decision tree and logistic regression. Although the calculation time can be reduced using Random mode, it still remains quite large. Perhaps the speed and accuracy of the lazy classification model can be improved by using probabilistic approaches, such as boosting. It is worth noting that the Tic-Tac-Toe data set was able to achieve perfect accuracy, which was not possible for the decision tree, but there is no certainty that a similar result will be reproducible on other data sets.