Ordered Sets in Data Analysis Project Report

Pugachev Alexander

15.12.2019

Contents

1 Statement of the lazy classification task

Within the project of Ordered Sets in Data Analysis it was required to develop and algorithm of lazy binary classification. The lazy classification algorithm works with the data which has only binary features. The task is for the test objects to predict which of the two classes each of them belongs to based on the objects from training set. The classification is done by finding intent for each test object, intersecting the found intent with all the objects from training set and calculating the positive and negative supports. Based on the value of aggregation function, we classify a test object as positive or negative. The whole variability of the lazy classification algorithm consists in choosing the method of numerical features binarization and choosing the aggregation function. In the report there will be discussed different approaches of binarization and aggregation functions in details.

2 Dataset information

For the project on Ordered Sets in Data Analysis I decided to choose Mobile Price Classification dataset[1]. This dataset represents different characteristics of various mobile phones and their price category (4 categories). It is used for classification task: to predict price category of mobile phone based on its characteristics. The initial dataset consists of 2000 entities with 21 features. For the project I have chosen all objects from 2 closest classes (because we have binary classification problem) and 14 features. These features are:

- battery_power (numerical): Total energy a battery can store in one time measured in mAh
- blue (binary): Has bluetooth or not
- clock_speed (numerical): Speed at which microprocessor executes instructions
- dual_sim (binary): Has dual sim support or not
- fc (numerical): Front Camera mega pixels
- four_g (binary): Has 4G or not
- int_memory (numerical): Internal memory in gigabytes
- n_cores (numerical): Number of cores of processor
- pc (numerical): Primary camera mega pixels
- ram (numerical): Random access memory in Megabytes
- talk_time (numerical): Longest time that a single battery charge will last when you are
- touch_screen (binary): Has touch screen or not
- wifi (binary): Has Wi-Fi or not
- price_range (binary): Price categories, target feature

Consider class ”0” as negative class and class ”1” as positive one. Thus, in our dataset there are 1000 objects (500 positive and 500 negative) with 13 features and 1 binary target feature. The scatter plot projection of the dataset onto 2–dimensional space which was computed using PCA algorithm is presented on Figure 1. The whole dataset was splitted into the training and testing parts in proportion 80% / 20% (each of the parts contains equal number of positive and negative objects).

3 Binarization strategies

There are no categorical values in the dataset, so I provide binarization strategy only for numerical features.

[1]Mobile Price Classification https://www.kaggle.com/iabhishekofficial/mobile-price-classification

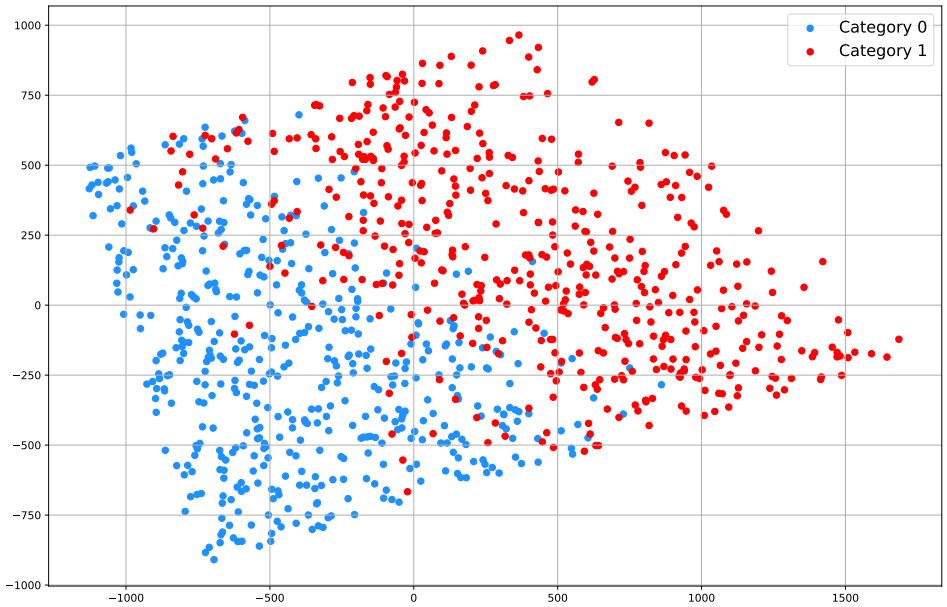

Figure 1: Visualization of the dataset using PCA.

3.1 Simple strategy

The first approach of binarization numerical features consists in the following. For each numerical feature I created 5 equally sized disjoint intervals. Each interval corresponds to a group of values of the feature. In this case the values that separate one group from the other (pivots) are:

$$P = \left\{ f_{\min} + \frac{i \cdot (f_{\max} - f_{\min})}{5} \mid i \in \{1, 2, 3, 4\} \right\}$$

where:

$f_{\min}$ – minimum value of the numerical feature
$f_{\max}$ – maximum value of the nuerical feature

It follows that for one numerical feature there are created 5 binary features. Each binary feature means entry of the feature value in the appropriate interval. For example, let's consider a numerical feature which has values from 0 to 10. For this feature there will be created pivots $\{2, 4, 6, 8\}$ which divide one group from the other. So, the intervals would be: $[0; 2), [2; 4), [4; 6), [6; 8), [8; 10]$.

3.2 Advanced strategy

In order to create more effective binarization approach I decided to look at the values of numerical features which I have in the dataset. I plotted the histogram of numerical features, all of them are presented in Figure 2.

The advanced approach consists in the following. We look at the histogram plots of our numerical features and find values of the feature which has low frequency. These values are considered as pivots which divide one group of feature values from the other, how it was described earlier. If there are no such values with low frequency, we divide feature by equally sized intervals. Also, for some of the features the number of intervals was revised. For example, n_cores feature values are integers from 1 to 8 inclusively, so I created 8 disjoint intervals for this feature – one interval per feature value. Finally we have the following pivots and intervals for numerical features:

battery_power – pivots: {1000, 1300, 1750}
clock_speed – 5 equally sized disjoint intervals
fc – 5 equally sized disjoint intervals
int_memory – pivots: {15, 25, 37, 47}
n_cores – 8 equally sized disjoint intervals

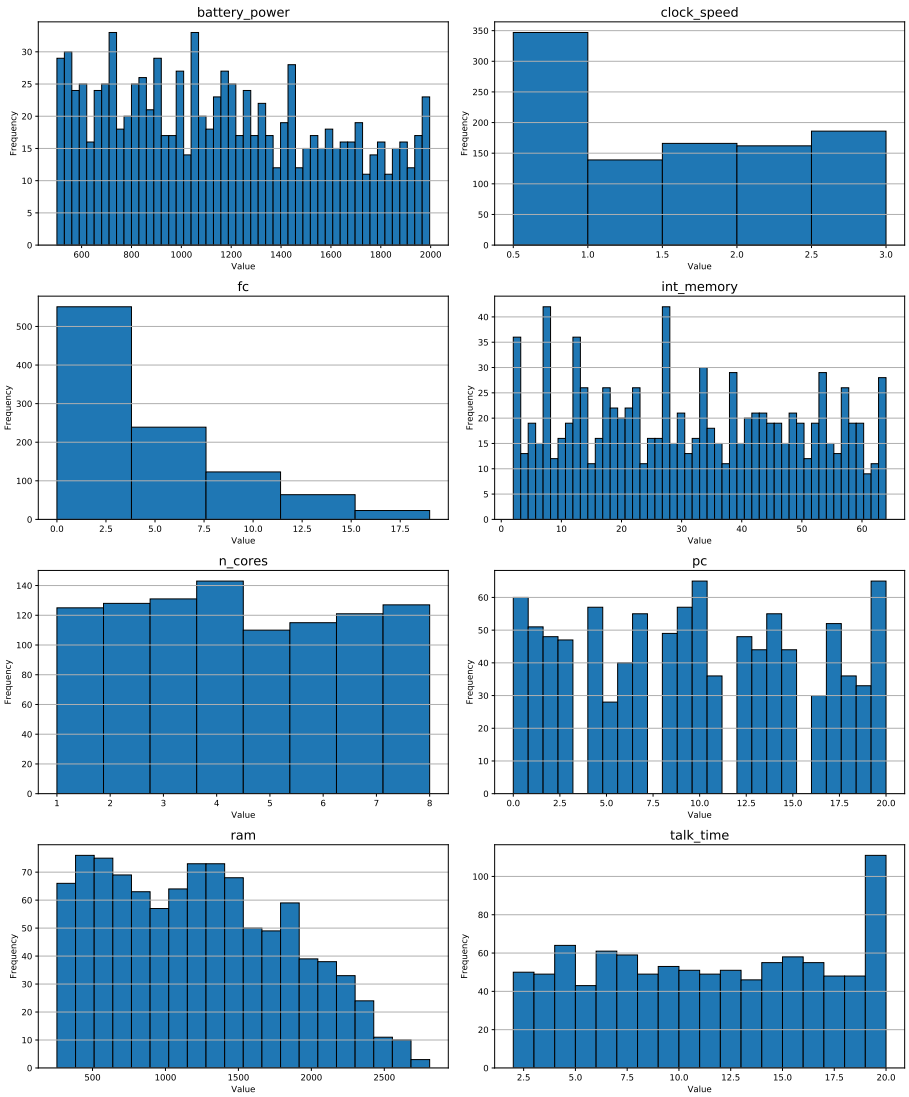

Figure 2: Histograms of the numerical features.

pc – pivots: {3.5, 7.5, 11.5, 15.5}
ram – 5 equally sized disjoint intervals
talk_time – pivots: {7, 11, 16}

For each of the versions of aggregation functions I used both simple and advanced binarization strategy.

4 Classification algorithm

4.1 Description

After binarization procedure is done, we start classification process. Each test object is intersected with each of the train objects. For each pair of test and train objects we find subset of attributes which are common for both of them. After that we find what proportion of objects among the positive and negative train objects do have the same attributes. In other words, we compute positive and negative supports for each pair of test and train objects. Supports for an arbitrary pair of test and train objects are defined by the following formula:

$$Sup_+ = \frac{|g'_{test} \cap g^+_{train}|^+}{|G_{pos}|}$$

$$Sup_- = \frac{|g'_{test} \cap g^-_{train}|^-}{|G_{neg}|}$$

where:

g'_{test} – attributes of a test object,
$+$ – prime operator for positive objects or their attributes,
$-$ – prime operator for negative objects or their attributes,
G_{pos} – positive train objects,
G_{pos} – negative train objects.

Finally, for each test object we have maximum 800 positive and 800 negative supports (there could be some pairs of test and train objects that do not have anything in common).

4.2 Aggregation functions

When we counted supports we need to define aggregation rules that would help us classify new object as positive or negative.

4.2.1 Simple approaches

As the basis for the first aggregation function I used the average function: for one test object we compute mean positive support and mean negative support. If positive mean support is greater than negative mean support, we classify test object as positive, otherwise as negative. Here is the formal description of the aggregation function:

$$Aggr^+_{mean}(g_{test}) = \frac{\sum_{i=1}^{K_+} Sup^i_+}{K_+}$$

$$Aggr^-_{mean}(g_{test}) = \frac{\sum_{i=1}^{K_-} Sup^i_-}{K_-}$$

$$Pred_{mean}(g_{test}) = \begin{cases} \text{Positive}, & Aggr^+_{mean}(g_{test}) > Aggr^-_{mean}(g_{test}) \\ \text{Negative}, & \text{otherwise} \end{cases}$$

where:

Sup^i_+ – i-th positive support in the list of positive supports,
Sup^i_- – i-th negative support in the list of negative supports,
K_+ – the amount of positive supports,
K_- – the amount of negative supports.

Besides the mean function I also used median and maximum functions. It means that if median (or maximum) value of positive supports is greater than the median (or maximum) value of negative supports then we classify object as positive, otherwise as negative. The formal description is provided below.

$$Aggr^+_{median}(g_{test}) = median(Sup_+)$$

$$Aggr^-_{median}(g_{test}) = median(Sup_-)$$

$$Pred_{median}(g_{test}) = \begin{cases} \text{Positive}, & Aggr^+_{median}(g_{test}) > Aggr^-_{median}(g_{test}) \\ \text{Negative}, & \text{otherwise} \end{cases}$$

where:

median – median function.

$$Aggr^{+}_{max}(g_{test}) = max(Sup_{+})$$

$$Aggr^{-}_{max}(g_{test}) = max(Sup_{-})$$

$$Pred_{max}(g_{test}) = \begin{cases} \text{Positive}, & Aggr^{+}_{max}(g_{test}) > Aggr^{-}_{max}(g_{test}) \\ \text{Negative}, & \text{otherwise} \end{cases}$$

where:

max – maximum function.

For the next aggregation function I used thresholds. When calculating aggregation function value we take into consideration only supports which are greater or equal some predefined threshold value. The mean and median aggregation functions were also computed with thresholds. For the experiments there were defined two thresholds: 0.3 and 0.5. For example, here is formal definition of median aggregation function with threshold ε:

$$Aggr^{+}_{median[\varepsilon]}(g_{test}) = median(Sup_{+}[\varepsilon])$$

$$Aggr^{-}_{median[\varepsilon]}(g_{test}) = median(Sup_{-}[\varepsilon])$$

$$Pred_{median[\varepsilon]}(g_{test}) = \begin{cases} \text{Positive}, & Aggr^{+}_{median[\varepsilon]}(g_{test}) > Aggr^{-}_{median[\varepsilon]}(g_{test}) \\ \text{Negative}, & \text{otherwise} \end{cases}$$

where:

median – median function.
$Sup_{+}[\varepsilon]$ – list of positive supports whose value is higher than ε
$Sup_{-}[\varepsilon]$ – list of negative supports whose value is higher than ε

4.2.2 Advanced approach

The last one approach that may be called advanced has the following idea. The values which we got from the aggregation functions described earlier can be weighted in order to achieve better results. For this we calculate positive and negative supports for the training objects, calculate mean positive support and mean negative support. Using calculated mean positive and negative supports and object labels from the training set, we train logistic regression that based on the supports predicts the category of the object. The trained logistic regression weights then are used to weigh positive and negative supports of the test objects. So, the final prediction of test object category is made by the logistic regression which was trained on trained supports. Formal description of this idea is written below.

$$Sup_{+} = \frac{|g'_{test} \cap g^{+}_{train}|^{+}}{|G_{pos}|}$$

$$Sup_{-} = \frac{|g'_{test} \cap g^{-}_{train}|^{-}}{|G_{neg}|}$$

$$Pred_{w\_mean}(g_{test}) = \begin{cases} \text{Positive}, & Logistic_{predict}(Sup_{+}, Sup_{-}) = 1 \\ \text{Negative}, & \text{otherwise} \end{cases}$$

where:

g'_{test} – attributes of a test object,
$+$ – prime operator for positive objects or their attributes,
$-$ – prime operator for negative objects or their attributes,
G_{pos} – positive train objects,
G_{pos} – negative train objects.
$Logistic_{predict}$ – logistic regression model

5 Results

All the results from experiments are presented in Table 1. Besides the lazy classification algorithms there were used some classical machine learning algorithms such as logistic regression and random forest. They were trained on non-binarized data and made predictions also on non-binarized data.. For each of them were found best set of hyperparameters which achieve highest scores. I used the following metrics: ROC AUC, accuracy, precision, F1 score.

Aggregation function	Binarization strategy	ROC AUC	Accuracy	Precision	F1 score
$Pred_{mean}$	Simple	0.921	0.925	0.873	0.932
$Pred_{mean}$	Custom	0.942	0.945	0.904	0.949
$Pred_{median}$	Simple	0.960	0.960	0.925	0.961
$Pred_{median}$	Custom	0.955	0.955	0.917	0.956
$Pred_{max}$	Simple	0.909	0.910	0.848	0.918
$Pred_{max}$	Custom	0.904	0.905	0.841	0.914
$Pred_{mean[0.5]}$	Simple	0.773	0.780	0.720	0.816
$Pred_{mean[0.5]}$	Custom	0.773	0.780	0.723	0.815
$Pred_{median[0.5]}$	Simple	0.550	0.550	0.554	0.531
$Pred_{median[0.5]}$	Custom	0.585	0.585	0.587	0.578
$Pred_{mean[0.3]}$	Simple	0.751	0.755	0.727	0.782
$Pred_{mean[0.3]}$	Custom	0.713	0.715	0.711	0.734
$Pred_{median[0.3]}$	Simple	0.520	0.520	0.516	0.567
$Pred_{median[0.3]}$	Custom	0.505	0.505	0.504	0.560
$Pred_{w\_mean}$	Simple	0.994	0.995	0.990	0.995
$Pred_{w\_mean}$	Custom	0.994	0.995	0.990	0.995
Logistic regression	–	0.900	0.900	0.911	0.902
Random forest	–	0.900	0.900	0.920	0.901

Table 1: Classification results.

6 Conclusion

According to the results obtained I can conclude that the lazy classification approach may be considered acceptable for solving binary classification tasks. Even with the simplest aggregation function it achieves better results than the classical machine learning algorithms such as logistic regression and random forest. In spite of this, the accuracy of a lazy classification algorithm highly depends on the binarization method and aggregation functions

that are chosen. Thus, according to F1 scores, in majority of cases the binarization strategy with custom pivots achieved better or equal results than simple binarization method. Also, the aggregation function is very important, for example, the use of tresholds significantly reduces the quality of the algorithm. This may happen because we make predictions based not on the full set of supports, but just on the best ones. It could influence much on the final score. As a result, the best accuracy was achieved using the aggregation function with weighted means.

The lazy classification algorithm is also easy to implement. The inference speed is acceptable (approximately 12 seconds for 200 object on a laptop). Moreover, the variability of choosing binarization method and aggregation function makes this algorithm applicable to any kind of binary classification task.

7 Technical details

The code for all experiments was written on Python language version 3.6. I used the following Python libraries:

- Numpy 1.14.5
- Scikit-learn 0.19.1
- Pandas 0.23.1
- Matplotlib 2.2.2

My code is freely available on Github[2].

8 References

[1] Kuznetsov S. Ordered Sets in Data Analysis Lectures, 2019.

[2] Strok F. Ordered Sets in Data Analysis Seminars, 2019.

[2]Alexander Pugachev Github https://github.com/apugachev/lazy-learning