Lazy binary classifier based on Formal Concepts

Glazkov Maxim

December 8, 2024

1 Introduction

This report was created as a project for the course "Ordered Sets in Data Analysis" at the HSE university. It provides a detailed exploration of a lazy binary classification algorithm based on formal concept analysis, which was applied to the Telco Customer Churn dataset.

The code with classification is here

The report describes the key steps involved in the classification process, including:

1. **Preliminary Dataset Analysis and Data Preprocessing**. This step involves examining the dataset to understand its structure, identifying missing values, outliers, or other irregularities, and performing necessary data cleaning and transformation. Special emphasis is placed on encoding categorical variables, scaling numerical features, and handling class imbalances to ensure the dataset is suitable for subsequent analysis.

2. **Description and Implementation of the Lazy Classification Algorithm**. The core of the report is dedicated to explaining the underlying principles of the lazy binary classification algorithm, which leverages formal concept analysis to evaluate the similarity between data points. The algorithm operates without constructing an explicit decision boundary during training, instead relying on local neighborhoods of test instances during prediction. Various modifications to the base algorithm are also discussed, including techniques to improve classification perfomance.

3. **Evaluation Metrics for Lazy Classification**. A comprehensive evaluation of the algorithm's performance is conducted using several quality metrics, including precision, recall, and F1-score. The unique characteristics of lazy classification are highlighted, emphasizing its advantages in specific scenarios such as imbalanced datasets or cases where interpretability is critical.

4. **Comparison with Popular Classification Algorithms**. The lazy binary classification algorithm is benchmarked against other widely used methods, such as logistic regression, random forests, and support vector machines. The comparison highlights the strengths and weaknesses of lazy classification in terms of accuracy, computational cost, and interpretability.

The report concludes with a summary of the results and recommendations for the practical application of lazy binary classification, highlighting its advantages and disadvantages.

2 Dataset description and Preprocessing

The Telco Customer Churn dataset is designed for exploring customer retention in the telecommunications industry. It contains information about customer demographics, account details, and services, alongside whether the customer has churned (stopped using the service)

2.1 Dataset overview

The dataset consists of **20 features** and **7043 instances (rows)**. Here is a description of each feature in the dataset:

- **customerID** (object): Unique identifier of each customer.

- **gender** (object): Customer gender (Male, Female).
- **SeniorCitizen** (int): Whether the customer is a senior citizen (1, 0).
- **Partner** (object): Does the customer have a partner (Yes, No).
- **Dependents** (object): Does the customer have dependents (Yes, No).
- **tenure** (int): Number of months the customer has been with the company.
- **PhoneService** (object): Does the customer have phone service (Yes, No).
- **MultipleLines** (object): Does the customer have multiple lines (Yes, No, No phone service).
- **InternetService** (object): Customer's Internet Service Provider (DSL, Fiber optic, No).
- **OnlineSecurity** (object): Does the customer have online security (Yes, No, No internet service).
- **OnlineBackup** (object): Does the customer have online backup (Yes, No, No internet service).
- **DeviceProtection** (object): Does the customer have device protection (Yes, No, No internet service).
- **TechSupport** (object): Does the customer have tech support (Yes, No, No internet service).
- **StreamingTV** (object): Does the customer have streaming TV (Yes, No, No internet service).
- **StreamingMovies** (object): Does the customer have streaming movies (Yes, No, No internet service).
- **Contract** (object): Customer's contract term (Month-to-month, One year, Two year).
- **PaperlessBilling** (object): Does the customer have paperless billing (Yes, No).
- **PaymentMethod** (object): Customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic)).
- **MonthlyCharges** (float): Customer's monthly charges.
- **TotalCharges** (float): Total charges for the customer.
- **Churn** (object): Has the customer churned (Yes, No).

Obviously **customerID** is dropped, because it doesn't give us any valuable information

There are only 11 missing values in the **TotalCharges** column. I decided to discard the rows that contain them, since the LazyFCA algorithm is very slow, and later I will take only a part of the full dataset.

After that, the features were encoded (encoding will be discussed later) and features with very low correlation with the target feature **Churn** were discarded. List of discarded features:

- **gender**
- **PhoneService**
- **MultipleLines**
- **DeviceProtection**
- **StreamingTV**
- **StreamingMovies**

Also I decided to check features with very high correlation with each other. Here is the pairs

- (**InternetService**, **MonthlyCharges**),
- (**tenure**, **TotalCharges**)

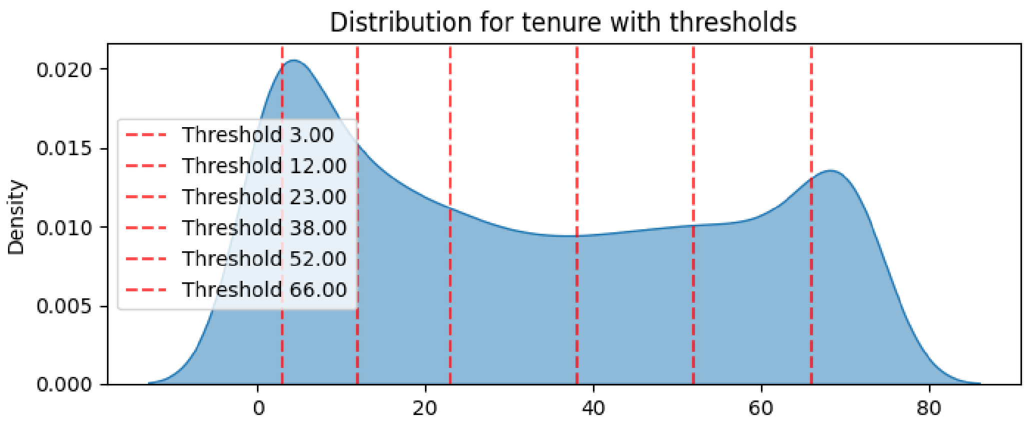
I decided not to remove one column from each pair, since the high correlation with each other should not affect the classification much.

2.2 Dataset encoding

Rules for encoding each feature:

- **customerID**: Dropped, because id doesn't give us any information
- **SeniorCitizen**, **Partner**, **Dependents**, **PaperlessBilling**: Dichotomic scale is used.
- **InternetService**: The combination of nominal scale (for values "DSL" and "Fiber optic") and dichotomic scale (for InternetService = "No" and InternetService = "Yes", when InternetService is "DSL" or "Fiber optic") is used. Information about availability of the Internet might be useful regardless of the type of Internet.
- **OnlineSecurity**, **OnlineBackup**, **TechSupport**: "No internet service" values replaced with "No" (because the "InternetService" column gives us information about Internet service, so that feature is redundant). After replacement, a dichotomic scale is used.
- **Contract**, **PaymentMethod**,
- **PaymentMethod**, **MonthlyCharges**, **tenure**: I found 6 evenly spaced quantiles (relative to the probability), the values of which were taken for Inter-Ordinal scaling. (that type of scaling has chosen, because dependecies between target and current features aren't clear)

Below there is some visualization of the threshold selection for the numerical feature **tenure**



Thresholds for tenure

- **Churn**: Target feature, encoded with "Yes" = True and "No" = False.

3 Description of the Lazy Classification Algorithm

For convenience, let the set of features be called M, and the set of objects G. The training set will be denoted as G_{train}, and the test set as G_{test}. The algorithm is lazy, meaning that when the model's `fit` method is called, it simply stores the training set.

When the `predict` method is called, the algorithm performs the following steps:

- Iterate over each sample $g' \in G_{\text{test}}$.
  - Divide the training set into two subsets:
    - Those classified positively ($\text{Churn} = 1$), denoted as G_+,
    - Those classified negatively ($\text{Churn} = 0$), denoted as G_-.
  - Initialize the counts of positive and negative classifiers, pos_class and neg_class, starting with 0.

- Iterate over each $g_+ \in G_+$:
  - Find intersection, the set of features whose values match between g_+ and g' (including their values).
  - Check if there exists at least one $g_- \in G_-$ that contains intersection.
  - If no such g_- is found, increment pos_class by 1.
- Iterate over each $g_- \in G_-$:
  - Find intersection, the set of features whose values match between g_- and g' (including their values).
  - Check if there exists at least one $g_+ \in G_+$ that contains intersection.
  - If no such g_+ is found, increment neg_class by 1.
- Classify g' based on the following rules:
  1. If pos_class > neg_class, classify g' as $+$.
  2. If pos_class < neg_class, classify g' as $-$.
  3. If pos_class = neg_class, classify g' with a default value (which can be set; in this case, it is $-$).

That's approach shows 0.74 accuracy score, but because of class imbalance this algorithm shows bad recall score for positive class. Thus, there is some modification, which tries to improve the algorithm performance

3.1 Class weights

First idea that came to mind is to create the class weights, which is connected with $|G_+|$ (number of all positive classifiers) and $|G_-|$ (number of all negative classifiers)

The formula for the balancing coefficient used in the algorithm is:

$$\text{balance\_coef} = \frac{|G_-|}{|G_+|}$$

This coefficient indicates how much smaller or larger G_+ is relative to G_-. It is used to compare the number of positive and negative classifiers. Instead of directly comparing pos_class and neg_class, the comparison is made between pos_class $\cdot$ balance_coef and neg_class. Example:

- We have $|G_+| = 3$ and $|G_-| = 7$
- We have instance g' with pos_class = 3 and neg_class = 4
- In original algorithm g' will be classified as negative, but with balance coefficient classification will be positive

This approach slightly improved the classification metrics, but the improvement was very minor and the recall for the positive class remained very low.

3.2 Acceptable probability of error

Another way to improve the metrics is to introduce an error probability with which we can decide whether a classifier is positive or negative. In the original algorithm, we have a fairly strict condition: **Check if there exists at least one $g_- \in G_-$ that contains intersection. If no such g_- is found, increment pos_class by 1.**

We modify this rule slightly by introducing error probabilities for each class:

- pos_err: the error probability we accept when saying the classifier is positive.
- neg_err: the error probability we accept when saying the classifier is negative.

Here is the updated description of the steps to determine if the classifier is positive:

- Find intersection, the set of features whose values match between g_+ and g' (including their values).
- Calculate num_pos, the number of all $g_+ \in G_+$ that contain intersection.
- Calculate num_neg, the number of all $g_- \in G_-$ that contain intersection.
- If num_neg $= 0$ and num_pos ≥ 1, declare the classifier as positive (strict condition from the original algorithm).
- If num_neg > 0 and $\frac{\text{num\_neg}}{\text{num\_pos}} \leq$ pos_err, also declare the classifier as positive.

For negative classifiers, the algorithm is modified similarly.

This approach, combined with balancing, improves the metrics to an acceptable level. However, it worsens the explainability of the model, as we can no longer say with certainty that a specific subset of feature values definitively indicates a positive (or negative) class, but only with a probability of $1 -$ pos_err (or $1 -$ neg_err).

4 Evaluated metrics

To begin, the main quantitative metrics were calculated: True Positive (tp), True Negative (tn), False Positive (fp), and False Negative (fn).

Next, quality metrics were used to evaluate the classification results. Below are the main formulas:

$$\text{Positive Predicted Value} = \text{Precision}_+ = \frac{tp}{tp + fp}$$

$$\text{True Positive Rate} = \text{Recall}_+ = \frac{tp}{tp + fn}$$

$$\text{Negative Predicted Value} = \text{Precision}_- = \frac{tn}{tn + fn}$$

$$\text{True Negative Rate} = \text{Recall}_- = \frac{tn}{tn + fp}$$

- **Precision** measures the proportion of correctly predicted positive samples among all samples predicted as positive.

- **Recall** measures the proportion of correctly predicted positive samples among all actual positive samples.

F1 score combines precision and recall into a single metric. It is the harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy measures the proportion of correctly classified samples among all samples:

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

For multiclass or imbalanced datasets, it is useful to calculate aggregated F1 scores:

F1 Macro: The average F1 score across all classes, treating each class equally:

$$F1_{\text{macro}} = \frac{F1_+ + F1_-}{2}$$

F1 Weighted: The weighted average of F1 scores across all classes, where the weight is proportional to the number of samples in each class:

$$F1_{\text{weighted}} = \frac{|G_+| \cdot F1_+ + |G_-| \cdot F1_-}{|G_+| + |G_-|}$$

Model	NPV	Precision	Specificity	Recall	Accuracy	$F1_{macro}$	$F1_{weighted}$
Original model	0.75	0.62	0.96	0.16	0.74	0.55	0.68
Balanced model	0.76	0.59	0.94	0.22	0.74	0.58	0.70
Errors model	0.84	0.56	0.82	0.60	0.76	0.71	0.76

Detailed Metrics for Different Lazy Algorithm Variants

The introduction of the balance coefficient and error probabilities (such as pos_err and neg_err) leads to slight improvements in overall performance (F1 scores), particularly for the positive class.

The accuracy stays relatively stable, suggesting that changes in classification behavior do not drastically affect the overall correctness of predictions, but rather the balance between positive and negative classes.

However, the recall for the positive class remains low across first two models, indicating that additional improvements may be needed to better identify positive instances in them. The last model somehow improve performance, but recall still remains lower that specificity.

5 Comparison with popular algorithms and results

Model	NPV	Precision	Specificity	Recall	Accuracy	$F1_{macro}$	$F1_{weighted}$
Lazy Algorithm	0.84	0.56	0.82	0.60	0.76	0.71	0.76
KNN	0.85	0.61	0.87	0.57	0.79	0.72	0.79
Logistic Regression	0.83	0.67	0.90	0.51	0.80	0.72	0.79
SVM	0.81	0.48	0.79	0.51	0.71	0.65	0.72
Random Forest	0.84	0.60	0.86	0.58	0.78	0.72	0.78
XGBoost	0.82	0.65	0.90	0.48	0.79	0.71	0.77

Comparison of Metrics for Various Classifiers

These are the metrics I obtained as a result of a GridSearch. In various notebooks, for example here, the Logistic Regression, Random Forest, and XGBoost models perform better than in my research (all F1 scores for these three models are above **0.81**). The results in them can be considered the best overall, since the Precision and Recall for each class are nearly balanced. It is evident that the results of lazy classification are inferior to popular models. However, in theory, they could be improved.

Another significant drawback of this model is its computational complexity. Since the algorithm is lazy, classifying a single instance takes considerable time. For example, in my notebook, predictions for the lazy classification models took longer than the GridSearch process for other models.

The advantage of lazy classification lies in its interpretability. In general, we can preserve intersections and explain that the classification decision was based on them. If this property is important, then this approach surpasses other models in terms of decision transparency.

5.1 Conclusion

To sum up, lazy classification has demonstrated decent results, slightly lagging behind other algorithms in classification quality and significantly lagging in computational efficiency. This algorithm can be useful when we need to classify a small test set and when the explainability of decisions is critical.