Lazy FCA Report

Sifei Meng

December 2, 2024

1 Dataset Description

The dataset is taken from Kaggle, the link is here. I've also uploaded my code and the dataset to my public github repository for this project. It's here.

The dataset consists of 45,000 records and 14 variables. These include:

- 5 categorical features
- 8 numeric features
- 1 target feature (`loan_status`)

The target feature, `loan_status`, indicates whether the loan application was approved (1) or rejected (0).

The following table details the features:

Feature Name	Description	Data Type
person_age	Age of the person	Float
person_gender	Gender of the person	Categorical
person_education	Highest education level	Categorical
person_income	Annual income	Float
person_emp_exp	Years of employment experience	Integer
person_home_ownership	Home ownership status (e.g., rent, own, mortgage)	Categorical
loan_amnt	Loan amount requested	Float
loan_intent	Purpose of the loan	Categorical
loan_int_rate	Loan interest rate	Float
loan_percent_income	Loan amount as a percentage of annual income	Float
cb_person_cred_hist_length	Length of credit history in years	Float
credit_score	Credit score of the person	Integer
previous_loan_defaults_on_file	Indicator of previous loan defaults	Categorical

Table 1: Feature details of the dataset.

2 EDA

First of all, I ensured that there were no empty cells in the dataset. Next, I divided the features into numeric and categorical groups for better analysis and preprocessing. Upon inspection, I identified that the dataset was imbalanced: the number of negative classes (`loan_status = 0`) significantly exceeded the number of positive classes (`loan_status = 1`).

I then investigated the distribution of each numeric feature. The initial distribution histograms for these features are presented in Figure 1. As observed, the `person_age` feature contains outliers, with data

indicating ages exceeding 100 years. Logically, banks are unlikely to grant credit to individuals over 100 years old, as they may not have the capacity to repay loans during their lifetime. Consequently, I removed the outliers by deleting values above the 90% quantile for the `person_age` feature, effectively reducing the long tail.

Following this, I observed that other numeric features also exhibited long tails. However, for these features, the abnormal data points might not necessarily qualify as outliers. In such cases, instead of removing data, I applied the natural logarithm transformation (`np.log`) to these features. The logarithmic transformation is particularly effective for compressing the range of values while preserving their relative proportions. The features to which I applied logarithmic transformation were `person_age`, `person_income`, `person_emp_exp`, and `cb_person_cred_hist_length`.

After applying these transformations, the resulting distributions closely resembled a normal distribution. The updated histograms for the transformed features are shown in Figure 2. These transformations not only addressed the issue of long tails but also helped standardize the data, facilitating better performance during the model training phase.

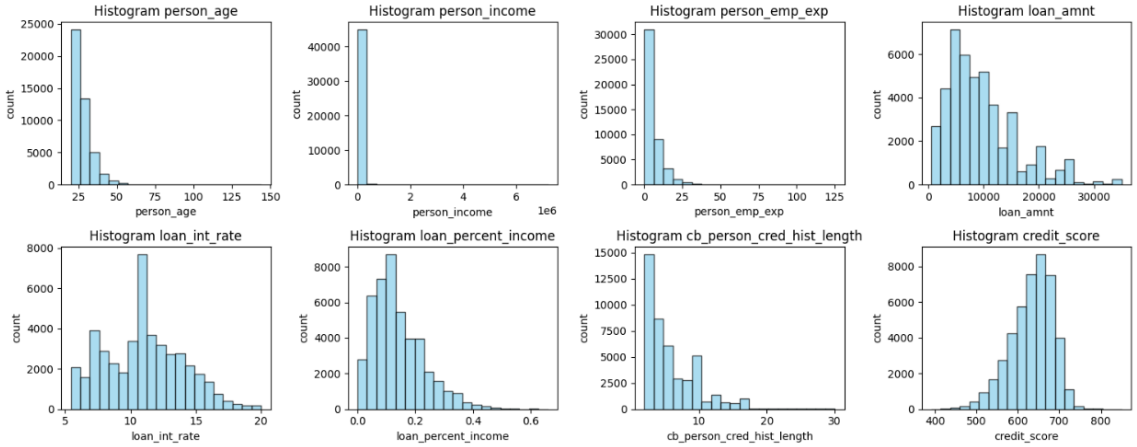

Figure 1: Initial distribution of numeric features

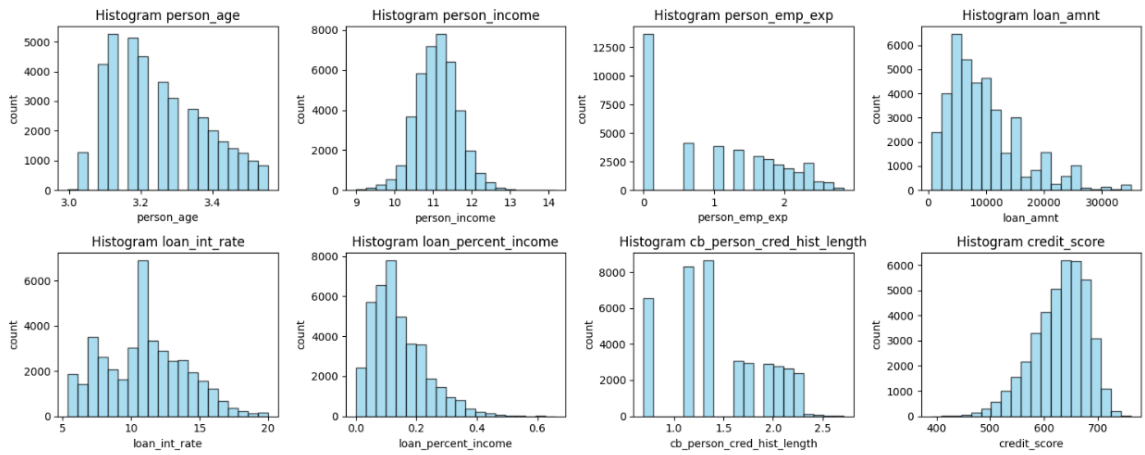

Figure 2: New distribution of numeric features

After cleaning the dataset, the next step was to investigate the relationship between each numeric feature and the target variable (`loan_status`). The initial analysis involved examining the correlation matrix, as shown in Figure 3. This correlation map provides a quick overview of the linear relationships between features and the target variable.

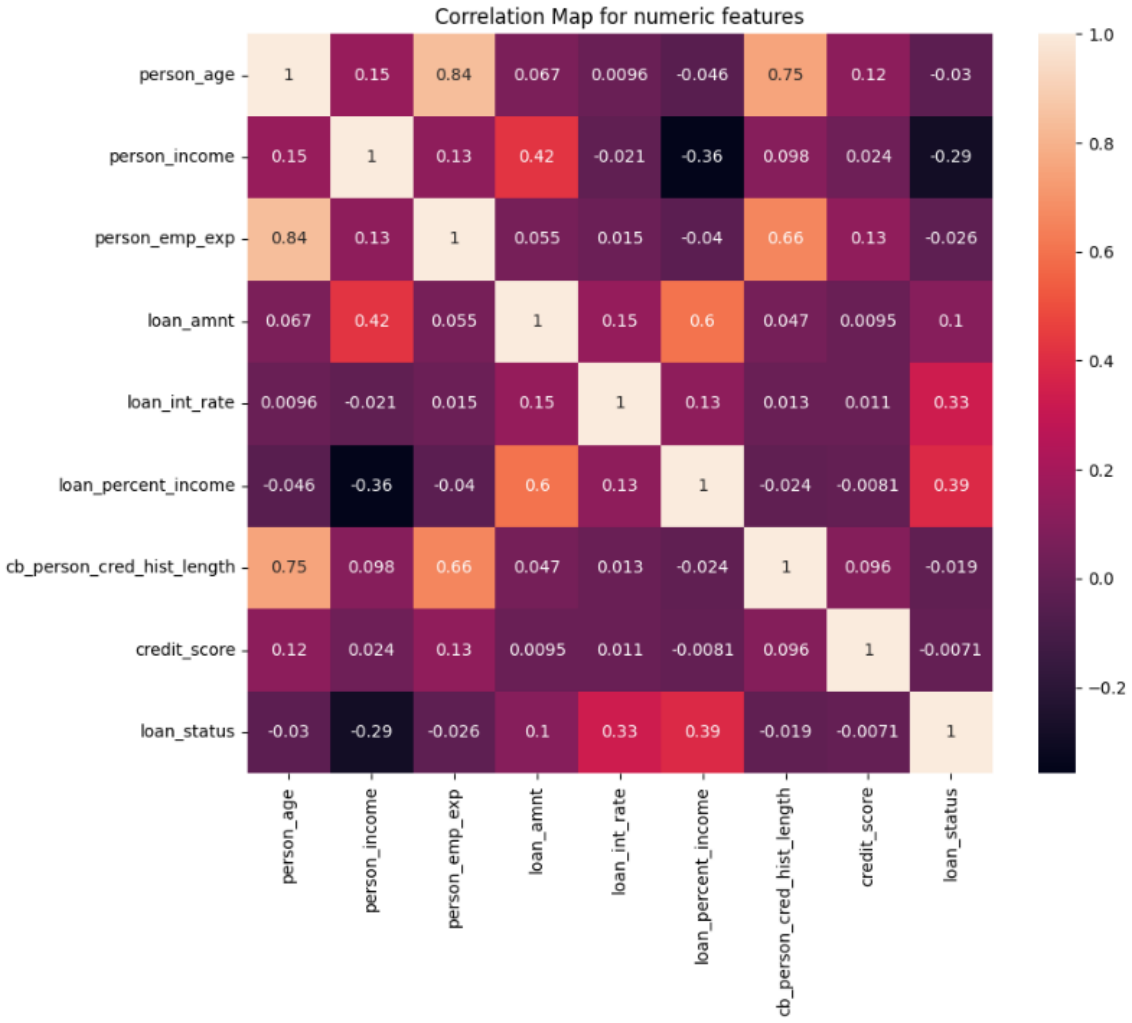

Figure 3: Correlation map of numeric features

From the correlation matrix, we observe that the features `person_income`, `loan_int_rate`, and `loan_percent_income` exhibit relatively strong correlations with the target variable. On the other hand, features such as `credit_score` and `cb_person_cred_hist_length` show almost no linear correlation with the target.

However, it is important to note that correlation does not necessarily imply a causal or meaningful relationship. To confirm whether they are indeed unrelated to the target variable, I visualized the data directly by plotting the distributions and scatter plots of these features against the target variable. As illustrated in Figure 4, the data confirms that `cb_person_cred_hist_length` and `credit_score` lack any discernible patterns or relationships with the target classes. This reinforces the conclusion that these features are unlikely to contribute meaningfully to predicting `loan_status`. Consequently, these features will be deleted.

I conducted an analysis of the box plots and density plots for all numeric features to understand their relationship with the target variable (`loan_status`). Based on these visualizations, a conclusion can be drawn that most numeric features have a significant impact on the target variable. This insight highlights the importance of these features in predicting loan approval outcomes.

Next, I shifted my focus to the categorical features. Beyond examining the distribution of each categorical feature, I analyzed the proportions of positive (`loan_status = 1`) and negative (`loan_status = 0`) classes for each unique value within the categorical features. This analysis was performed by constructing contingency tables, which were then normalized to display the proportions of each class. The normalized

contingency tables provide a clearer understanding of how each categorical feature value influences the target variable.

The results of this analysis are illustrated in Figure 5. These visualizations provide valuable insight into the relationship: the proportion of negative and positive values for each unique value for the features `person_gender, person_education` are almost equal, so I decided to delete them.

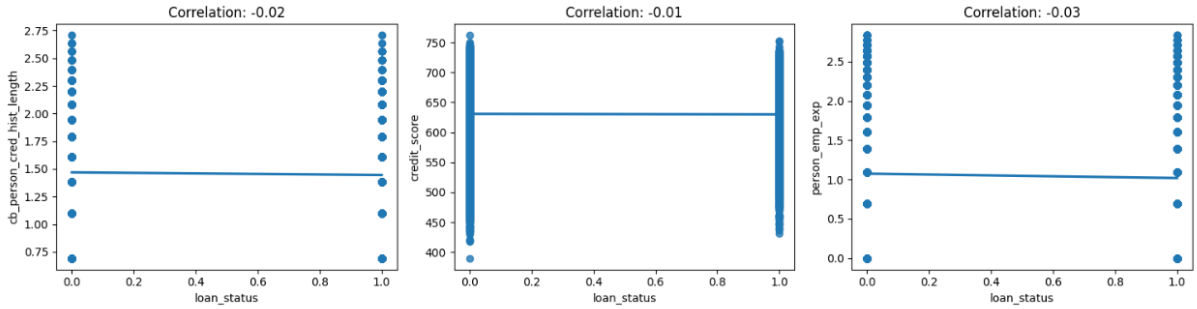

Figure 4: Relationship between our target and weakly correlated features

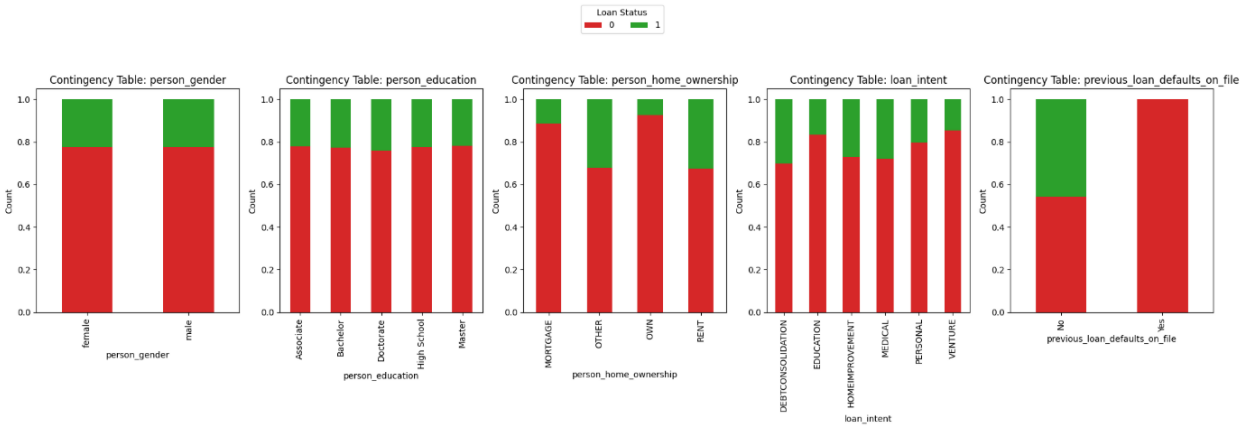

Figure 5: Relationship between our target and weakly correlated features

In conclusion, I addressed data preprocessing by removing outliers, normalizing long tails in distributions using the natural logarithm transformation, and deleting columns that were deemed unrelated to the target variable based on correlation analysis, relation plots, and contingency tables.

3 Binarization

The next step in the project involved selecting appropriate scaling methods to binarize the dataset. The primary idea behind my approach was to use inter-ordinal scaling for numeric features. This method involves splitting values into intervals, as categorizing individuals into bins provides more meaningful information for the classifier compared to simple thresholds like "less than or equal to" or "greater than or equal to." To ensure fairness, the thresholds for each feature were determined manually, aiming to distribute values approximately equally among bins. These thresholds were derived exclusively from the training subset, mimicking a real-world scenario where the distribution of the test dataset is unknown.

Let's take the feature `person_age` as an example. For this feature, I divided individuals into nearly equally distributed categories and created a new column called `AgeCategory`. Each category was used as a threshold to generate binary features using inter-ordinal scaling. After this transformation, the original

numeric feature was removed. A more detailed explanation of the thresholds and the resulting distribution of categories can be found in my Jupyter notebook.

Categorical features were scaled using OneHot encoding, as the unique values within a categorical feature do not have ordinal relationships and should be treated independently. This encoding ensures that each category is represented as a separate binary feature.

After the binarization process, the dataset consisted of 40,514 rows and 65 columns, providing a comprehensive and structured representation of the data, ready for model training.

Given the size of the dataset, training it directly using the Lazy FCA algorithm would be excessively time-consuming. To address this challenge, I decided to resample the dataset to achieve a balanced representation of positive (`loan_status = 1`) and negative (`loan_status = 0`) classes. By taking approximately equal amounts of samples from each class (1450 positive elements and 1006 negative elements) and combining them, I created a new balanced dataset consisting of 3,070 rows and 65 features.

This balanced dataset was then split into training (80%) and test (20%) subsets for subsequent use in model training and evaluation. This approach ensures that the model can be trained efficiently while minimizing the effects of class imbalance, thereby enhancing the reliability of performance metrics on the test set.

4 Metrics

I've decided to use the following metrics:

1. True Positive (TP)

Definition: Cases where the model correctly predicts the positive class.
Usefulness: Indicates how well the model identifies positive cases.

2. True Negative (TN)

Definition: Cases where the model correctly predicts the negative class.
Usefulness: Reflects the model's ability to recognize non-events or normal cases.

3. False Positive (FP)

Definition: Cases where the model incorrectly predicts the positive class for a negative sample.
Usefulness: Represents false alarms or over-predictions.

4. False Negative (FN)

Definition: Cases where the model incorrectly predicts the negative class for a positive sample.
Usefulness: Represents missed detections.

5. True Negative Rate (Specificity)

Definition: The proportion of actual negatives correctly predicted as negative:

$$Specificity = \frac{TN}{TN + FP}$$

Usefulness: Measures the model's ability to avoid false alarms.

6. Negative Predictive Value (NPV)

Definition: The proportion of predicted negatives that are truly negative:

$$NPV = \frac{TN}{TN + FN}$$

Usefulness: Measures the reliability of negative predictions.

7. False Positive Rate (FPR)

Definition: The proportion of actual negatives incorrectly predicted as positive:

$$FPR = \frac{FP}{FP + TN}$$

Usefulness: Indicates the likelihood of generating false alarms.

8. False Discovery Rate (FDR)

Definition: The proportion of predicted positives that are actually negative:

$$FDR = \frac{FP}{FP + TP}$$

Usefulness: Measures how often positive predictions are incorrect.

9. Accuracy

Definition: The proportion of all correct predictions (TP + TN) to the total number of samples:

$$Accuracy = \frac{TP + TN}{TotalSamples}$$

Usefulness: Provides an overall measure of model performance. Best used for balanced datasets, as it can be misleading for imbalanced datasets.

10. Precision

Definition: The proportion of predicted positives that are actually positive:

$$Precision = \frac{TP}{TP + FP}$$

Usefulness: Measures the quality of positive predictions.

11. Recall (True Positive Rate, Sensitivity)

Definition: The proportion of actual positives correctly predicted as positive:

$$Recall = \frac{TP}{TP + FN}$$

Usefulness: Indicates the model's ability to detect positive cases.

12. F1 Score

Definition: The harmonic mean of Precision and Recall:

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Usefulness: Balances Precision and Recall, making it suitable for imbalanced datasets. Useful when both false positives and false negatives need to be minimized.

As our dataset is not perfectly balanced, F1 score is more valued in our project, but every metric listed above will be assessed.

5 LazyFCA Baseline

The Lazy FCA algorithm is based on Formal Concept Analysis (FCA), which uses a mathematical approach to classify data by finding relationships between objects (data points) and their attributes (features). The "lazy" aspect of the algorithm means that classification is performed only when needed, without building a global model in advance. Instead, it uses the training data directly during classification.

First of all, it splits the training data into two groups based on the target labels, and each group contains only the features of the respective class as boolean arrays.

After that, each sample is converted to a boolean array (sample_array), where True indicates the presence of a feature. Then we find overlaps with the training data by computing intersections between the sample and the positive and negative class arrays. If the sample matches any pattern from the positive class and does not match any contradictory pattern from the negative class, then it counts as a positive classifier. The same logic goes for negative classifiers. Finally, if there are no corresponding positive or negative classifiers, we give the sample a default class (1), because there are more positive elements in the dataset. Otherwise, if positive classifiers outnumber negative classifiers, predict the positive class (1). Otherwise, predict the negative class (0).

The baseline algorithm is derived from Alan's example. I've changed the code a bit to vectorize the algorithm to improve efficiency.

The result is depicted in Table 2

Metric	**Value**
True Positives	360
True Negatives	12
False Positives	2
False Negatives	240
Accuracy	0.6059
Precision	0.9945
Recall (Sensitivity)	0.6000
Specificity	0.8571
Negative Predictive Value	0.0476
False Positive Rate	0.1429
False Discovery Rate	0.0055
F1 Score	0.7484

Table 2: Performance Metrics of LazyClassifierFCA

6 State-of-the-art methods

In this section, I applied KNN, Naive Bayes (NB), Logistic Regression (LR), SVM, Decision Tree (DT), Random Forest (RF), and XGBoost (XGB) to both the binarized dataset, as shown in Table 3, and the unbinarized dataset, as shown in Table 4. Detailed hyperparameter fine-tuning processes and results are provided in the accompanying notebook for reference.

As observed, classical methods generally perform better on the unbinarized dataset. This is likely because the application of the Standard Scaler, which normalizes numeric features into continuous values, preserves more detailed information. In contrast, dividing data into bins during binarization introduces generalization, which can lead to a loss of granularity in the dataset.

Among all methods, XGBoost and Random Forest consistently achieve the best performance across both datasets. Additionally, it is worth noting that most algorithms exhibit a tendency toward false negative errors. This behavior may be attributed to the higher prevalence of positive class elements in the dataset, which makes it more challenging for models to capture all positive cases accurately.

Metric	KNN	NB	LR	SVM	DT	RF	XGB
True Positives	333	309	338	343	334	342	338
True Negatives	176	199	189	184	182	183	192
False Positives	29	53	24	19	28	20	24
False Negatives	76	53	63	68	70	69	60
Accuracy	0.8290	0.8274	0.8583	0.8583	0.8404	0.8550	0.8632
Precision	0.9199	0.8536	0.9337	0.9475	0.9227	0.9448	0.9337
Recall (Sensitivity)	0.8142	0.8536	0.8429	0.8345	0.8267	0.8321	0.8492
Specificity	0.8585	0.7897	0.8873	0.9064	0.8667	0.9015	0.8889
Negative Predictive Value	0.6984	0.7897	0.7500	0.7302	0.7222	0.7262	0.7619
False Positive Rate	0.1415	0.2103	0.1127	0.0936	0.1333	0.0985	0.1111
False Discovery Rate	0.0801	0.1464	0.0663	0.0525	0.0773	0.0552	0.0663
F1 Score	0.8638	0.8536	0.8860	0.8875	0.8721	0.8849	0.8895

Table 3: Performance Metrics for Binarized dataset

Metric	KNN	NB	LR	SVM	DT	RF	XGB
TP	343	346	342	346	351	345	337
TN	188	177	182	190	176	186	206
FP	19	16	20	16	11	17	25
FN	64	75	70	62	76	66	46
Accuracy	0.8648	0.8518	0.8534	0.8730	0.8583	0.8648	0.8844
Precision	0.9475	0.9558	0.9448	0.9558	0.9696	0.9530	0.9309
Recall	0.8428	0.8219	0.8301	0.8480	0.8220	0.8394	0.8799
Specificity	0.9082	0.9171	0.9010	0.9223	0.9412	0.9163	0.8918
NPV	0.7460	0.7024	0.7222	0.7540	0.6984	0.7381	0.8175
FPR	0.0918	0.0829	0.0990	0.0777	0.0588	0.0837	0.1082
FDR	0.0525	0.0442	0.0552	0.0442	0.0304	0.0470	0.0691
F1 Score	0.8921	0.8838	0.8837	0.8987	0.8897	0.8926	0.9047

Table 4: Performance Metrics for Unbinarized dataset

7 LazyFCA optimization

To optimize the algorithm, it is crucial to identify the key issues in the baseline Lazy FCA approach. The primary challenge lies in its reliance on exact matches, which introduces several limitations:

1. Limited Dataset Size Due to High Time Complexity

The time complexity of the Lazy FCA algorithm restricts its scalability to larger datasets. As a result, we are constrained to using relatively small datasets, which may lack sufficient diversity to effectively represent all potential patterns in the data.

2. Dependency on Perfect Intersections

The Lazy FCA algorithm relies heavily on finding exact intersections between the features of test samples and training samples. However, in real-world scenarios, perfect matches are rare.

3. Undefined Classes and Default Classification

When the algorithm fails to find an exact match or intersection for a test sample, the class remains undefined. In the vanilla implementation, such undefined samples are classified as positive by default. This behavior can skew predictions, especially when there is a significant class imbalance in the dataset.

Optimization Directions

Addressing these issues requires enhancing the algorithm's flexibility. The solutions proposed include:

- Allowing for approximate matches by introducing similarity thresholds.
- Manually setting minimum cardinality and maximum counter-examples.
- Applying additional weights for classes to handle imbalanced dataset.

Therefore, I've added several new hyper-parameters:

- xxx (Maximum Counter-Examples): The maximum number of counterexamples allowed for a positive or negative classifier to be considered valid.
- min_cardinality (Minimum Cardinality): The minimum number of matched features required in an intersection to qualify as a valid classifier.
- sample_threshold (Proportion Threshold for Matching Samples in the Current Class): The minimum proportion of matching features between a sample and current class training data for it to be considered a valid classifier for the current class.
- counter_threshold (Proportion Threshold for Matching Counter Samples): The minimum proportion of matching features between a sample and training data of the alternative class for it to be considered a counterexample.
- class_weight (Weight for Balancing Positive vs. Negative Classifiers): The factor by which the number of negative classifiers is multiplied when comparing with positive classifiers to decide the final prediction.

The optimal solution's hyperparameter is fine-tuned, and the final result reached a f1 score of 0.893506, which is better than random forest and XGBoost for the binarized dataset and comparable with them for the unbinarized dataset. More detailed solution can be found in my notebook. Therefore, by addressing these challenges, the Lazy FCA algorithm can become more robust, scalable, and effective in real-world applications.

8 Results comparison

In Table 5, a significant improvement can be observed between the old version of the LazyClassifierFCA and the optimized algorithm. The enhanced performance metrics, such as accuracy, precision, and F1 score, demonstrate the effectiveness of the optimizations.

Moreover, as shown in Table 6, our optimized classifier performs better than state-of-the-art methods on the binarized dataset.

Finally, we can see in Table 7 , our optimized classifier is also comparable with state-of-the-art methods on the unbinarized dataset. These results underline the robustness and competitiveness of the improved LazyClassifierFCA.

9 Conclusion

In this project, I selected a binary dataset on loan approval, consisting of both numeric and categorical features. The workflow began with data cleaning, including the removal of outliers, followed by preprocessing steps. Relationships between each feature and the target variable were thoroughly investigated, and all features were transformed into binary representations.

Subsequently, I implemented the Lazy FCA baseline algorithm and evaluated its performance. Additionally, I applied several state-of-the-art methods, such as Random Forest and XGBoost, to the same dataset in both binarized and unbinarized versions, incorporating hyperparameter fine-tuning. The results indicated that Random Forest and XGBoost consistently achieved the best performance across both versions of the dataset.

Metric	Previous Value	Updated Value
True Positives	360	**344**
True Negatives	12	**188**
False Positives	2	**18**
False Negatives	240	**64**
Accuracy	0.6059	**0.8664**
Precision	0.9945	**0.9503**
Recall (Sensitivity)	0.6000	**0.8431**
Specificity	0.8571	**0.9126**
Negative Predictive Value	0.0476	**0.7460**
False Positive Rate	0.1429	**0.0874**
False Discovery Rate	0.0055	**0.0497**
F1 Score	0.7484	**0.8935**

Table 5: Comparison of Previous and Updated Performance Metrics of LazyClassifierFCA

Metric	KNN	NB	LR	SVM	DT	RF	XGB	LazyClassifierFCA
True Positives	333	309	338	343	334	342	338	**344**
True Negatives	176	199	189	184	182	183	192	**188**
False Positives	29	53	24	19	28	20	24	**18**
False Negatives	76	53	63	68	70	69	60	**64**
Accuracy	0.8290	0.8274	0.8583	0.8583	0.8404	0.8550	0.8632	**0.8664**
Precision	0.9199	0.8536	0.9337	0.9475	0.9227	0.9448	0.9337	**0.9503**
Recall (Sensitivity)	0.8142	0.8536	0.8429	0.8345	0.8267	0.8321	0.8492	**0.8431**
Specificity	0.8585	0.7897	0.8873	0.9064	0.8667	0.9015	0.8889	**0.9126**
Negative Predictive Value	0.6984	0.7897	0.7500	0.7302	0.7222	0.7262	0.7619	**0.7460**
False Positive Rate	0.1415	0.2103	0.1127	0.0936	0.1333	0.0985	0.1111	**0.0874**
False Discovery Rate	0.0801	0.1464	0.0663	0.0525	0.0773	0.0552	0.0663	**0.0497**
F1 Score	0.8638	0.8536	0.8860	0.8875	0.8721	0.8849	0.8895	**0.8935**

Table 6: Performance Metrics for Binarized Dataset Including LazyClassifierFCA

Metric	KNN	NB	LR	SVM	DT	RF	XGB	LazyClassifierFCA
TP	343	346	342	346	351	345	337	**344**
TN	188	177	182	190	176	186	206	**188**
FP	19	16	20	16	11	17	25	**18**
FN	64	75	70	62	76	66	46	**64**
Accuracy	0.8648	0.8518	0.8534	0.8730	0.8583	0.8648	0.8844	**0.8664**
Precision	0.9475	0.9558	0.9448	0.9558	0.9696	0.9530	0.9309	**0.9503**
Recall	0.8428	0.8219	0.8301	0.8480	0.8220	0.8394	0.8799	**0.8431**
Specificity	0.9082	0.9171	0.9010	0.9223	0.9412	0.9163	0.8918	**0.9126**
NPV	0.7460	0.7024	0.7222	0.7540	0.6984	0.7381	0.8175	**0.7460**
FPR	0.0918	0.0829	0.0990	0.0777	0.0588	0.0837	0.1082	**0.0874**
FDR	0.0525	0.0442	0.0552	0.0442	0.0304	0.0470	0.0691	**0.0497**
F1 Score	0.8921	0.8838	0.8837	0.8987	0.8897	0.8926	0.9047	**0.8935**

Table 7: Performance Metrics for Unbinarized Dataset Including LazyClassifierFCA

Further, I improved the baseline Lazy FCA algorithm by introducing various thresholds and class weights. After fine-tuning the hyperparameters, the optimized algorithm achieved results comparable to those of Random Forest and XGBoost.

Looking ahead, I believe there is potential to further enhance the algorithm's efficiency and performance through additional optimizations and refinements.