Vodopyanova Yulia,
Chekmareva Kristina

Lazy FCA Project

Repository:
https://github.com/kschekmareva/LazyFCA_Classifier_OSDA_project
Original dataset link: https://huggingface.co/datasets/ETdanR/adult_income

Dataset Description

This dataset is a binary classification dataset that predicts whether an individual's income is greater than $50,000 per year based on demographic and occupational features. Predict the income level (`income`) of an individual — a binary target variable with two possible values:

1) >50K
2) <=50K

Dataset size:

- Number of rows: 11,208
- Number of columns: 14

Columns and their types:

Column	Type	Description
age	int64	Age of the individual
workclass	object	Type of employment
education	object	Education level
educational-num	int64	Encoded education level
marital-status	object	Marital status
occupation	object	Occupation type
relationship	object	Relationship in family
race	object	Race
gender	object	Gender
capital-gain	int64	Capital gains
capital-loss	int64	Capital losses

hours-per-week	int64	Number of working hours per week
native-country	object	Native country
income	object	Target variable (income level)

Feature types:

- Numerical (5): age, educational-num, capital-gain, capital-loss, hours-per-week
- Categorical (9): workclass, education, marital-status, occupation, relationship, race, gender, native-country, income

Target variable statistics:

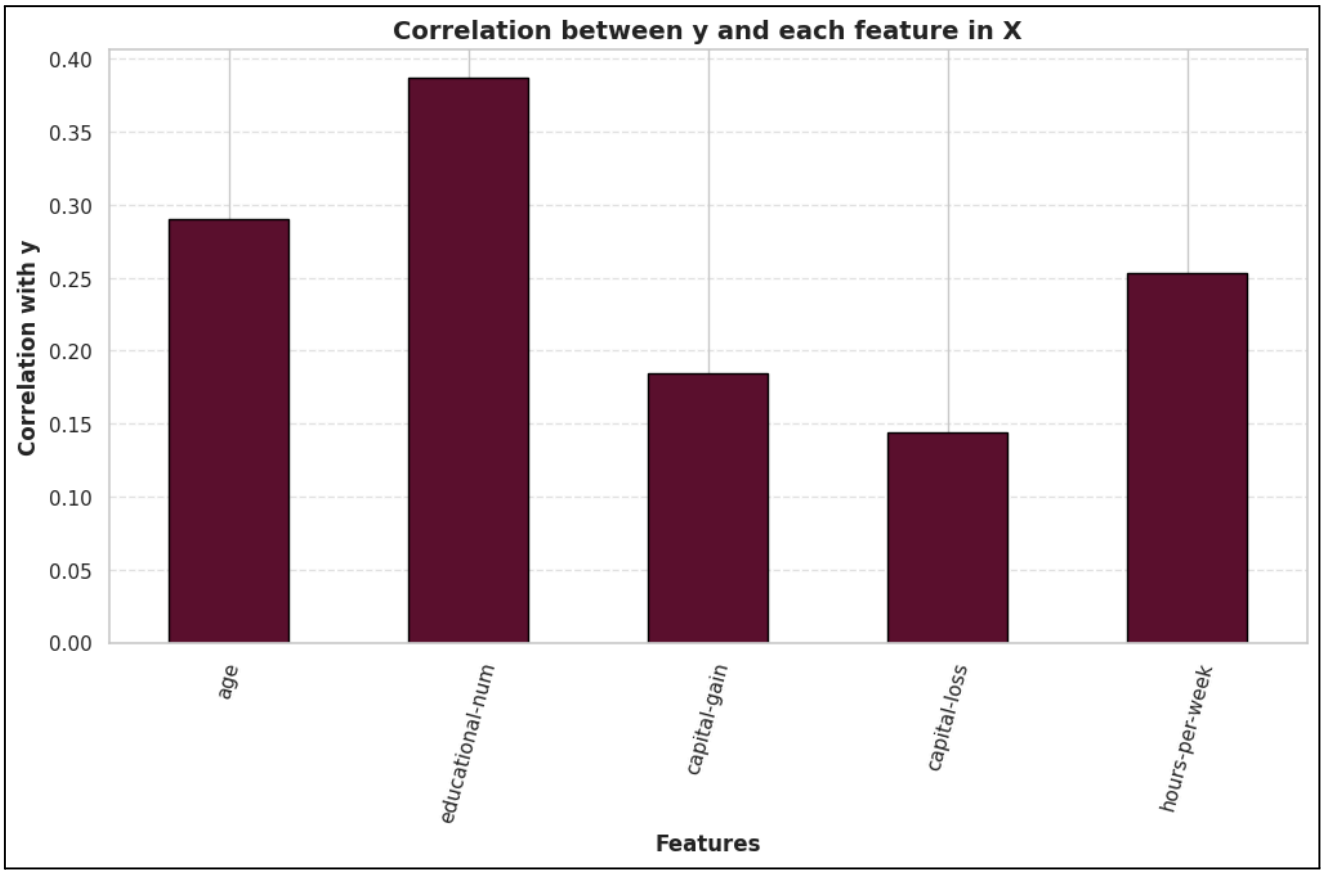

```
Class Value Counts:
income
>50K     5604
<=50K    5604
Name: count, dtype: int64

Class Proportions:
income
>50K     0.5
<=50K    0.5
Name: count, dtype: float64
```

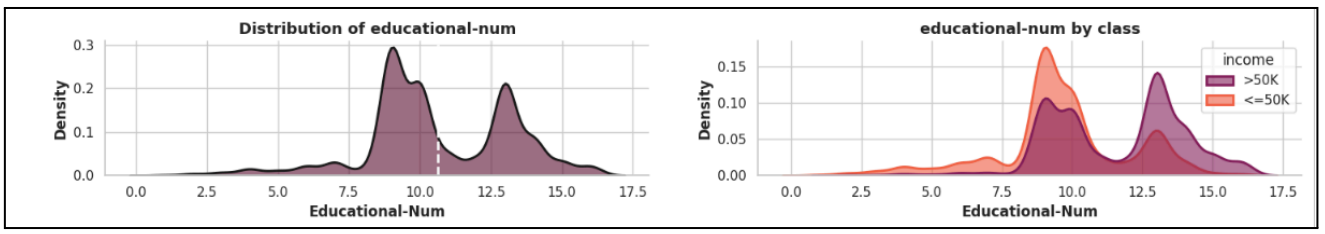
The target variable is well balanced.

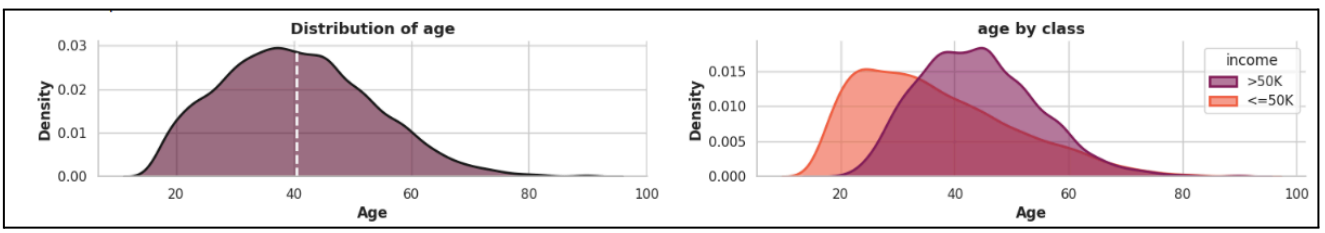
What steps were done

1. Explanatory Data Analysis:

- Finding missing values: None (all columns are complete)
- Unique values per feature: Ranges from 2 (gender) to 91 (capital-gain) for numerical features, and up to 39 for categorical features (native-country).
- Looking at basic feature distributions.
- Finding correlation between numerical features and target:
    1) It was found that there was a moderate positive correlation between the target and educational-num (around 0.40), target and age (around 0.29).
    2) No negative correlation was found between target and features.







2. Data Processing:

- Due to the computational complexity of LazyFCA, the dataset was reduced to smaller subsets:
    1) **Training:** 800 samples from the original 8,966 training rows
    2) **Testing:** 200 samples from the original 2,242 testing rows

Stratified sampling was used to preserve the class distribution in both subsets.

- **Pattern Structures**:
    1) Categorical features were manually binarized using **one-hot encoding.** Each unique category in a categorical feature was converted into a separate binary column.
    2) Numerical features were kept **unchanged.** But there are Interval-based descriptions for numerical features in class LazyClassifierFCA.
    3) After preprocessing, the training set had 101 features, combining numerical and one-hot encoded categorical variables.
    4) Thus, the overall pattern structure is defined as a product of intervals for all numerical attributes with binary sets for all categorical attributes. The meet operation acting independently on each component: interval component → interval intersection; boolean component → logical AND.
- Target variable: Converted to binary encoding: >50K → 1, <=50K → 0

3) **Lazy FCA – Vanilla Version**

The baseline Lazy FCA classifier follows the classical definition of lazy conceptual classification: for each test sample, the algorithm intersects it with **every training object** and checks whether the intersection forms a valid **positive** or **negative classifier**.

Performance and Limitations:

1) Prediction time for 200 rows: **~449 seconds**
2) F1 score: 0.8465, F1 score (weighted): 0.83406. Accuracy: 0.83

```
Classification report:
              precision    recall  f1-score   support

           0       0.89      0.76      0.82       100
           1       0.79      0.91      0.85       100

    accuracy                           0.83       200
   macro avg       0.84      0.83      0.83       200
weighted avg       0.84      0.83      0.83       200

F1 score (macro): 0.8340666247642992
F1 score (weighted): 0.8340666247642992
```

3) This algorithm is conceptually simple and interpretable, but extremely slow (requires checking all train objects). Moreover, there is no prioritization of informative classifiers and is susceptible to noisy numeric intervals.

4) **Improved Version 1 – Bit-Packed & Strength-Weighted Lazy FCA**

Key Improvements:

- **Bit Packing**: All binary columns are packed into uint64 bit-blocks, allowing fast bitwise operations. Numeric matrices are stored in float32 for speed and memory efficiency. Intersections and subset checks are performed with vectorized bitwise comparisons. The effect is a reduction of prediction time from ~449 s → ~15 s on the same test set.
- **Strength Scoring & Top-K Voting**: Instead of simply *counting* classifiers, each classifier receives a strength value: $\text{strength} = \frac{1}{1+\text{interval\_size}} \cdot (1+\log(1+\text{support}))$. This prioritizes: tight numeric intervals (more specific concepts) and high-support classifiers (more reliable). Only the top-K strongest classifiers (default K=400) participate in final voting.
- Performance:
  1. Prediction time: **15.4 seconds** (Much faster (≈30×)).
  2. F1 score: 0.8585, F1 score (weighted): 0.8495. Accuracy: 0.850

```
Classification report:
              precision    recall  f1-score   support

           0       0.90      0.79      0.84       100
           1       0.81      0.91      0.86       100

    accuracy                           0.85       200
   macro avg       0.86      0.85      0.85       200
weighted avg       0.86      0.85      0.85       200

F1 score (macro): 0.8494580489763148
F1 score (weighted): 0.8494580489763147
```

3. This algorithm is a more stable classification due to support-based weighting and is faster.

5) **Improved Version 2 – Neighbor-Filtered, Weighted Lazy FCA (Lazy FCA + kNN)**

Key improvements:

- **Neighborhood Reduction:** A weighted k-nearest neighbors (kNN) index is built using standardized numerical features, binary features, and mutual-information–based feature weights. For each test sample, only the nearest ~200 neighbors are considered as candidate classifiers. This drastically reduces the search space compared to intersecting with the full training set.
- **Distance-Weighted Voting:** Each neighbor contributes as a potential classifier, but its influence is weighted by distance. Positive and negative class scores are accumulated using these weights, and the class with the higher total weight is selected.
- **Local support (across neighbors):** Binary intersections, numeric interval computations, and positive/negative support counts are calculated only within the local neighbor subset, preserving core FCA reasoning.
- Performance:
    1. Prediction time: 309 seconds (slower than Version 1 due to nearest-neighbor search and local FCA computations).
    2. F1 score: 0.8667, F1 score (weighted): 0.8596. Accuracy: 0.860

```
Prediction time: 309.356
Classification report:
              precision    recall  f1-score   support

           0       0.90      0.81      0.85       100
           1       0.83      0.91      0.87       100

    accuracy                           0.86       200
   macro avg       0.86      0.86      0.86       200
weighted avg       0.86      0.86      0.86       200

F1 score (macro): 0.8596491228070176
F1 score (weighted): 0.8596491228070176
```

3. This variant achieves the highest predictive quality among all Lazy FCA versions. While slower than the bit-packed Version 1, it balances speed and accuracy effectively by reducing the candidate search space via neighborhood filtering, making it a robust and precise approach for classification.

Vanilla Lazy FCA is simple but very slow and treats all training objects equally, making it sensitive to noise. Improved Version 1 speeds up prediction using bit-packing and weights classifiers by strength, improving stability and efficiency. Improved Version 2 further focuses on local neighbors with distance-weighted voting, achieving higher accuracy by considering only relevant patterns. Overall, the improvements show how prioritizing strong and local patterns can balance speed and predictive quality.

6) **Results for all models**:

	Model	Accuracy	F1	ROC-AUC	TP	TN	FP	FN
1	LogReg	0.875	0.876847	0.93950	89	86	14	11
9	Lazy FCA Improved (NN)	0.860	0.866667	0.86000	91	81	19	9
8	Lazy FCA Improved	0.850	0.858491	0.85000	91	79	21	9
2	Naive Bayes	0.845	0.854460	0.94110	91	78	22	9
5	Random Forest	0.850	0.854369	0.92260	88	82	18	12
4	Decision Tree	0.845	0.853081	0.89805	90	79	21	10
3	SVM RBF	0.845	0.851675	0.92030	89	80	20	11
7	Lazy FCA	0.835	0.846512	0.83500	91	76	24	9
6	XGBoost	0.830	0.834951	0.92430	86	80	20	14
0	KNN	0.820	0.828571	0.89990	87	77	23	13

- **Top Performers:** Logistic Regression achieves the highest overall accuracy (0.875) and F1 score (0.8768), showing strong balance between precision and recall. Moreover, Lazy FCA Improved NN also showed good results in accuracy and F1 Score (the second place after LogReg).
- **Lazy FCA Variants:** Improved Lazy FCA with neighbor filtering reaches F1 0.8667 and accuracy 0.860, outperforming the standard Lazy FCA (F1 0.8465, accuracy 0.835), demonstrating the benefit of weighting and neighborhood-based filtering.
- **Ensemble Models:** Random Forest and XGBoost provide solid performance (F1 ~0.854 and 0.835, respectively), but XGBoost has slightly lower accuracy due to more false negatives.
- **Naive Bayes & SVM:** Both show moderate performance (accuracy ~0.845, F1 ~0.854–0.851), indicating good generalization but slightly lower robustness compared to top models.
- **KNN & Decision Tree:** These models perform the worst in this set (accuracy $\leq$0.845, F1 score $\leq$ 0.853), likely due to sensitivity to noise and limited ability to capture complex patterns.
- **Trade-offs:** Models like Lazy FCA Improved (NN) and Random Forest achieve a good balance between true positives and negatives, while top models like Logistic Regression maximize overall predictive quality.

7) **Interpretability analysis**

As for the interpretability of ready-made models, the results of which were compared with a lazy classifier, they can be divided into several arbitrary (unstrict) groups. These groups have different levels of interpretability.:

1) High interpretability (transparent rules):

- **Decision Tree**: one can view a decision tree with rules, get *feature_importances_*;
- **LogReg**: provides coefficients that show the influence how much and in which direction features influence the target;
- **Naive Bayes**: based on class probabilities; by providing us with *theta_* and *var_* parameters, we can easily understand the contribution of the feature.

2) Average interpretability (requires additional analysis):

- **Random Forest**: it is still possible to get *feature_importances_*, but, unlike Decision Tree, Random Forest doesn't provide specific rules (due to basing its decisions on several Trees);
- **XGBoost**: it is close to Random Forest as it also provides us with average feature importance for each feature, but still, since the model is based on trees (in our case, 200 estimators), we cannot know a specific algorithm for choosing a particular class as a prediction.

3) Low interpretability (require additional analysis):

- **SVM (RBF)**: complicated nonlinear transformation into a high-dimensional space (weights exist only in the feature space of the RBF-core, not in the initial one); hence, doesn't provide *feature_importances_* (we found them by applying permutation test) and a clear algorithm of its performance;
- **KNN**: based on the nearest neighbors in the feature space, hence, interpreting it directly is difficult; as SVM (RBF), KNN also provides no importance values or explicit coefficients and no classification rules.

*****Interpretability visualizations are available in the .ipynb file

As for our priority model (**Improved Version 1**), it has both advantages and disadvantages.

Advantages:

- Rules for classification are clear: "if the binary features are X and numeric ones are in the range [a,b], then the class is Y" (classifiers-based);
- The built-in *explain_sample()* method shows the specific rules for each prediction.

```
=== The strongest negative classifier: ===
Strength: 2.0986
Support: +0 / -2
Binary features:
  - workclass_Private: True
  - education_Some-college: True
  - marital-status_Never-married: True
  - race_White: True
  - gender_Male: True
Numerical intervals:
  - age: [24.00, 24.00]
  - educational-num: [10.00, 10.00]
  - capital-gain: [0.00, 0.00]
  - capital-loss: [0.00, 0.00]
  - hours-per-week: [40.00, 40.00]
```

So, we can understand why each class was chosen.

Disadvantages:

- There are a lot of rules (*top_k*), it is difficult to review everything.

Thus, the algorithm of the Lazy Classification is transparent and understandable, it is possible to deduce the triggered classifiers, however, due to their number, it is inefficient to consider and analyze each of them.