Final Project - Credit default prediction report

By Lin Xiao & Miruthula Sivakumar

1. Introduction

The main goal of this project is to predict whether a loan applicant is likely to default using machine learning techniques. Accurately identifying potential defaulters allows financial institutions to make informed and safer lending decisions, minimizing financial risk.

This project involves:

- Understanding the structure and semantics of the dataset
- Cleaning, encoding, and transforming raw features
- Standardizing numeric inputs
- Building and evaluating classification models using supervised learning

Problem Type: Binary classification

Target Variable: default (0 = no default, 1 = default)

Business Value: Helps banks and lending institutions reduce losses by identifying high-risk applicants in advance

Approach:

We utilize multiple variants of logistic regression and support vector machines (SVM), with a focus on minimizing false negatives—i.e., reducing the number of defaulters incorrectly predicted as safe. Key performance metrics include:

- Recall (for minimizing missed defaulters)
- Precision
- Confusion matrix
- ROC-AUC score

2. Data Preprocessing

Effective data preprocessing is crucial for building robust and accurate machine learning models. In this project, we undertook several steps to prepare the dataset for modeling:

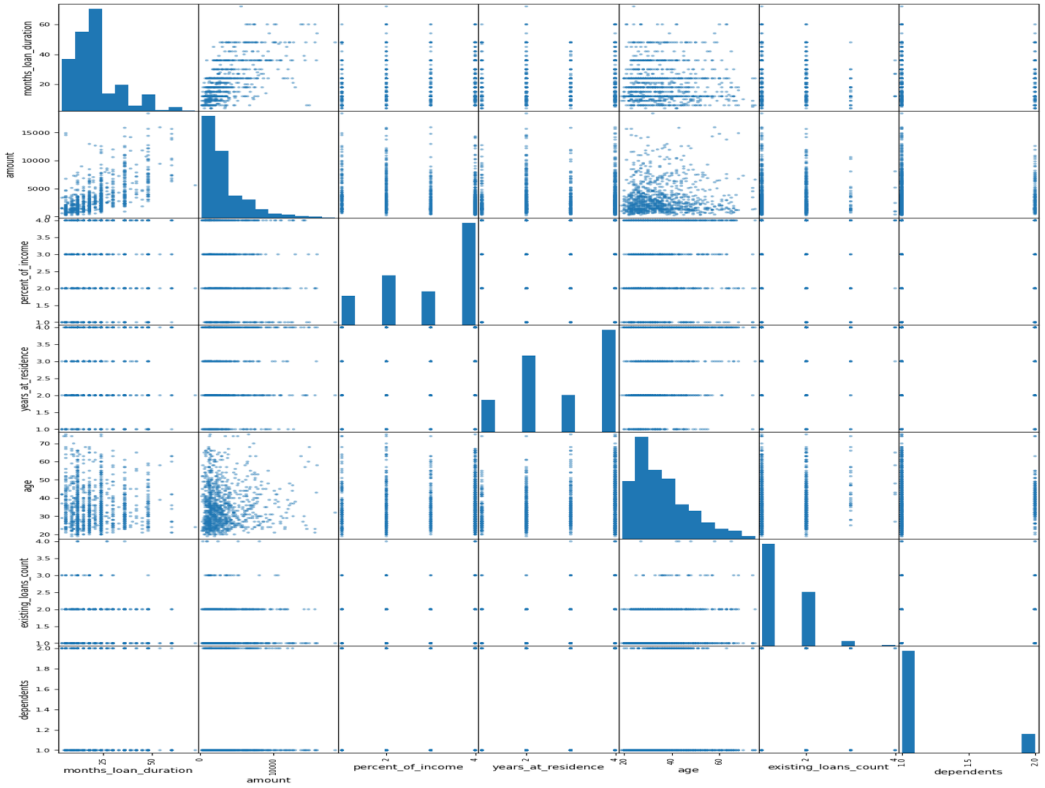
2.1. Data Loading and Initial Inspection

We began by importing the dataset from a CSV file into a panda DataFrame. This allowed us to leverage pandas' powerful data manipulation capabilities for subsequent preprocessing tasks.

2.2. Exploratory Data Analysis (EDA)

To understand the underlying structure and relationships within the data, we performed exploratory data analysis:

- Scatter Plots: We generated scatter plots for various feature pairs to visually inspect relationships and identify potential outliers or anomalies.
- Distribution Analysis: Histograms and box plots were used to assess the distribution of numerical features, helping us identify skewness and the need for transformations.

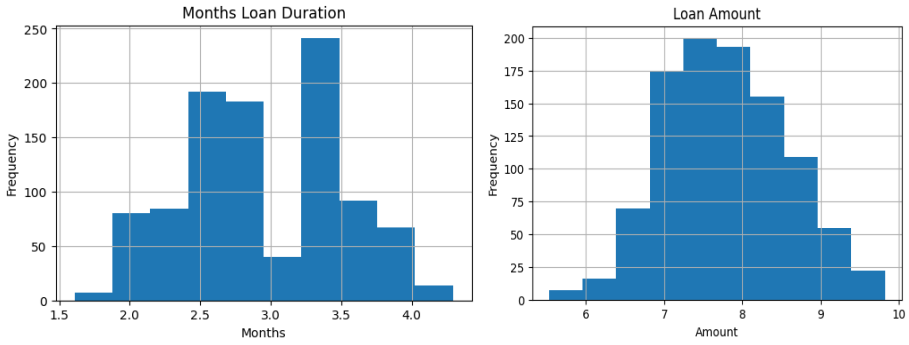


2.3. Feature Transformation

2.3.1. Log Transformation of Skewed Features

Some numerical features exhibited right-skewed distributions, which can adversely affect model performance. To address this, we applied logarithmic transformations to these features, normalizing their distributions

```
# feature to log
credit_data['months_loan_duration'] = np.log1p(credit_data['months_loan_duration'])
credit_data['amount'] = np.log1p(credit_data['amount'])
```



2.3.2. Binary Encoding

Binary categorical variables were converted to numerical format:

- 'phone': Originally containing 'yes' and 'no' values, this feature was mapped to 1 and 0, respectively.
- 'default': The target variable, indicating whether a client defaulted, was already in binary format.

```
# set Binary data
credit_data['default'] = credit_data['default'].map({'yes': 1, 'no': 0})
credit_data['phone'] = credit_data['phone'].map({'yes':1, 'no':0})
```

2.3.3. Ordinal Encoding

Ordinal categorical features, which have a meaningful order, were encoded accordingly:

- 'checking_balance': Categories like 'unknown', '<0 DM', '1 – 200DM', '>200 DM' were mapped to 0, 1, 2, and 3.
- 'credit_history': Categories like 'poor', 'critical', 'good', 'very good', 'perfect' were mapped to 0, 1, 2, 3, and 4.
- 'savings_balance': Categories like 'unknown', '<100DM', '100-500DM', '500-1000DM', '>1000DM' were mapped to 0, 1, 2, 3, and 4.
- 'employment_duration': Categories like 'unknown', '<1 year', '1-4 years', '4-7 years', '>7 years' were mapped to 0, 1, 2, 3, and 4.

2.3.4. Feature Engineering

We created new features to capture additional information:

- Age_group: According age to split as young, middle, and senior three groups
- Debt-to-Income Ratio (DTI): Calculated as the ratio of monthly loan payment to monthly income.
- Credit Utilization: Calculated as the ratio of loan amount to the sum of checking and savings balances.

```
# Creat new feature age_group, debt_income_ratio,credit_unilization
credit_data['age_group'] = credit_data['age'].apply(lambda x: 'young' if x <30 else 'middle' if x <50 else 'senior')

# debt to income ration: monthly payment/ monthly income
credit_data['debt_income_ratio'] = (
    credit_data['amount']/credit_data['months_loan_duration'])/(
        credit_data['amount']/credit_data['percent_of_income'] * 100)

#Credit Utilization, +1 for if saving balance is 0
credit_data['credit_utilization'] = credit_data['amount']/(credit_data['savings_balance_encoded']+1)
```

2.4. Final Dataset Preparation

2.4.1. After preprocessing, we finalized the dataset by:

- Dropping original features that were transformed or encoded.
- Ensuring no missing values remained.
- Splitting the dataset into training and testing sets for model evaluation.

```
from  sklearn.model_selection import train_test_split

# set X and y
y = credit_data['default']
X = credit_data.drop(columns=['default', 'savings_balance', 'checking_balance', 'employment_duration', 'credit_history'])

# Selecting 80% of the training data randomly
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

2.4.2. Feature Scaling

To ensure consistent scale across all numerical features, we applied z-score standardization using StandardScaler() from scikit-learn. This step helps improve the convergence and performance of many machine-learning models, including logistic regression and support vector machines.

Feature scaling was implemented as part of a pipeline using ColumnTransformer, which applies different preprocessing strategies to numerical and categorical features:

- Numerical features were scaled using StandardScaler()
- Categorical features were one-hot encoded using OneHotEncoder()

```
num_feature = [
    'checking_balance_encoded','months_loan_duration', 'credit_history_encoded', 'amount','savings_balance_encoded',
    'employment_duration_encoded', 'percent_of_income', 'years_at_residence', 'age', 'dependents',
    'phone', 'debt_income_ratio', 'credit_utilization','existing_loans_count'
]
cat_feature = [
    'age_group', 'purpose','other_credit','housing','job'
]


preprocessor = ColumnTransformer(transformers = [
    ('num', StandardScaler(), num_feature),
    ('cat',  OneHotEncoder(), cat_feature),
])

x_prepared = preprocessor.fit_transform(x_train)
test_x_prepare = preprocessor.transform(x_test)
```

This comprehensive preprocessing pipeline ensured that the data was clean, well-structured, and suitable for training effective machine learning models.

3. Modeling

In this section, we implement and compare several supervised learning models to predict credit default. The models were chosen based on their interpretability, performance, and ability to handle imbalanced data. Our primary evaluation goal is to minimize false negatives, which correspond to missed defaulters. To achieve this, we emphasize recall as the key performance metric.

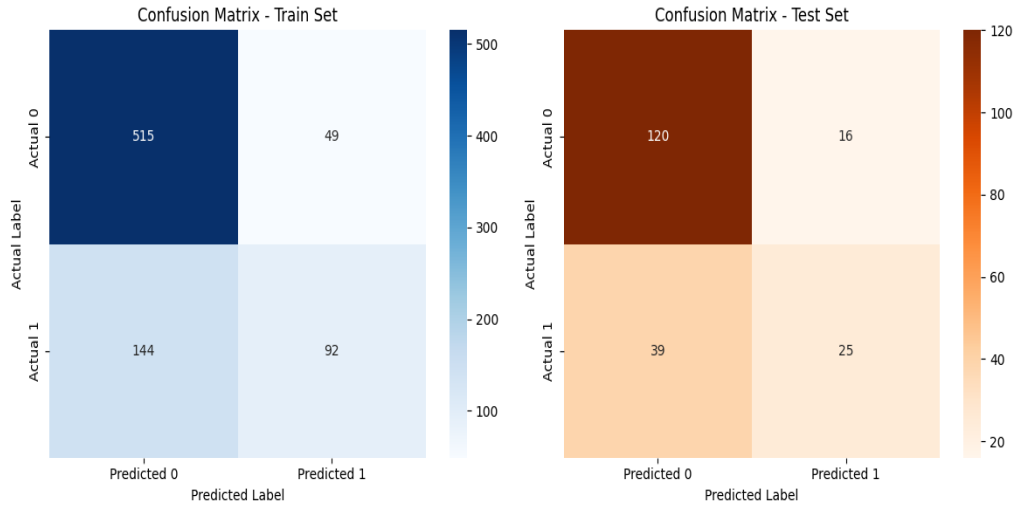
3.1. Model Overview

We evaluated the following models:

Model	Description	Goal
Model 1: Default Logistic Regression	No class weighting, default L2 regularization	Baseline
Model 2: Weighted Logistic Regression with GridSearch	Uses class_weight='balanced' and GridSearchCV to optimize C, focusing on maximizing recall	Minimize false negatives
Model 3: ElasticNet Logistic Regression	Combines L1 and L2 penalties via penalty='elasticnet', with GridSearchCV for both C and l1_ratio	Feature selection + generalization
Model 4: Support Vector Machine (SVM)	Kernel-based classifier using RBF kernel with GridSearch for C and gamma	Non-linear classification

3.2. Model 1: Default Logistic Regression

We trained a baseline logistic regression model without any class weighting or threshold adjustment. The resulting confusion matrices for both the training and testing sets are shown below:



- The model performs reasonably well in identifying non-defaulters (Actual 0), but has difficulty recognizing defaulters (Actual 1).
- The number of false negatives is high, which means the model is failing to flag many default-risk applicants.
- This makes the default logistic regression model unsuitable for credit risk applications where missing a defaulter is costly.

3.3. Model 2: Weighted Logistic Regression with GridSearch

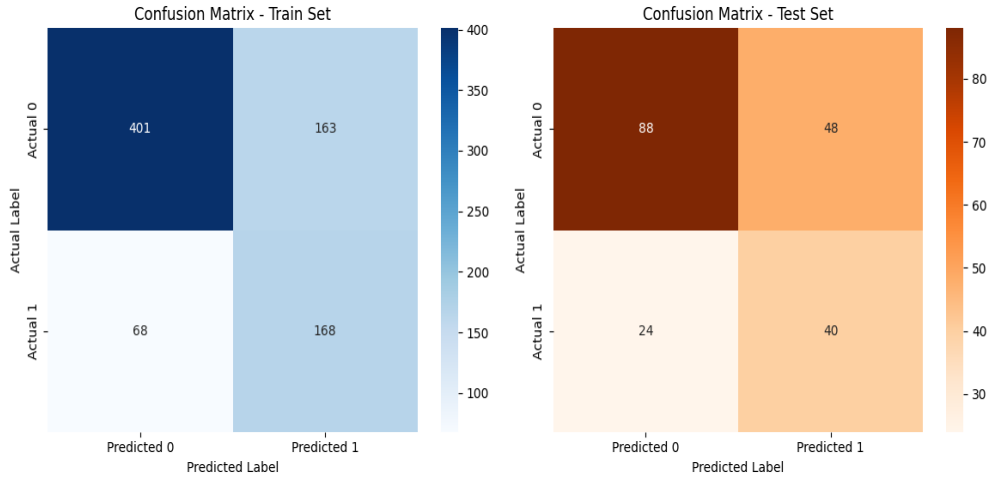
To reduce false negatives and better identify potential defaulters, we trained a logistic regression model with the following enhancements:

- Class imbalance handling via class_weight='balanced'
- Regularization strength (C) optimized using GridSearchCV
- Recall used as the scoring metric in cross-validation

```
from sklearn.model_selection import GridSearchCV

# params = {"C": np.logspace(-3, 1, 10)}  # 0.001 ~ 10
params = {"C": np.linspace(0.01, 2, 10)}

# put cv as integer no shuffle
grid_logregC = GridSearchCV(log_reg_c, params, cv =5, scoring= 'recall')

grid_logregC.fit(x_prepared, y_train)
```

Best Parameters Found via GridSearch:

- Best C: 1.77809
- Cross-validated Recall Score: 0.6480



- The weighted model drastically reduces false negatives, which aligns with our business objective of catching all potential defaulters.
- Although precision dropped slightly (more false positives), this trade-off is acceptable in credit risk modeling where missing a defaulter is more costly.
- This model strikes a better balance between recall and generalization, making it more practical for real-world deployment.

3.4. Model 3: ElasticNet Logistic Regression

This model combines both L1 and L2 regularization techniques using the elasticnet penalty in scikit-learn. L1 regularization encourages sparsity (feature selection), while L2 improves model generalization. This combination is well-suited for high-dimensional data or when we suspect some features may be redundant.

Tuning Parameters:

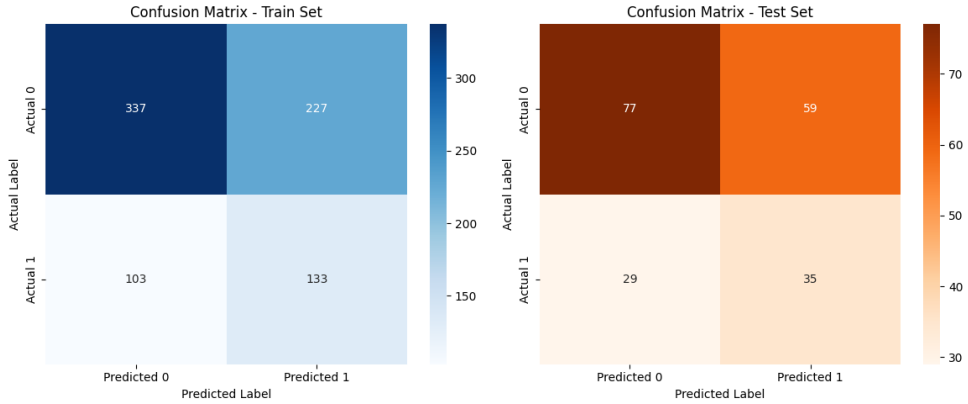
- C: Inverse of regularization strength
- l1_ratio: Controls the mix of L1 (sparsity) and L2 (stability)

We used GridSearchCV to tune both hyperparameters with recall as the scoring metric.

```
params_model3 = {
    'l1_ratio': np.linspace(0.01, 1, 30),
    'C': np.linspace(0.01, 0.99, 10)
}

log_reg_model3 = LogisticRegression(
    penalty='elasticnet', solver='saga', class_weight='balanced', max_iter=10000
)
grid_model3 = GridSearchCV(log_reg_model3, params_model3,scoring='recall', cv=5)

grid_model3.fit(x_prepared, y_train)

best_C_model3 = grid_model3.best_params_['C']
best_l1_ratio = grid_model3.best_params_['l1_ratio']
```

Best Parameters from GridSearch:

- C = 0.01000
- L1 Ratio = 0.51724
- Cross-validated Recall Score: 0.70842



- Compared to Model 1 (default logistic), recall improved substantially.
- Compared to Model 2 (weighted logistic), this model performed slightly worse in recall on the test set (Model 2 FN = 24 vs Model 3 FN = 29).
- However, ElasticNet can help in identifying important features by shrinking some coefficients towards zero.

While not the top-performing model in recall, it offers an interpretable balance of feature selection and model complexity.

3.5. Model 4: Support Vector Machine (SVM)

Support Vector Machines (SVMs) are powerful classifiers particularly well-suited for non-linear decision boundaries. In this model, we use an RBF (radial basis function) kernel to capture complex relationships between features.

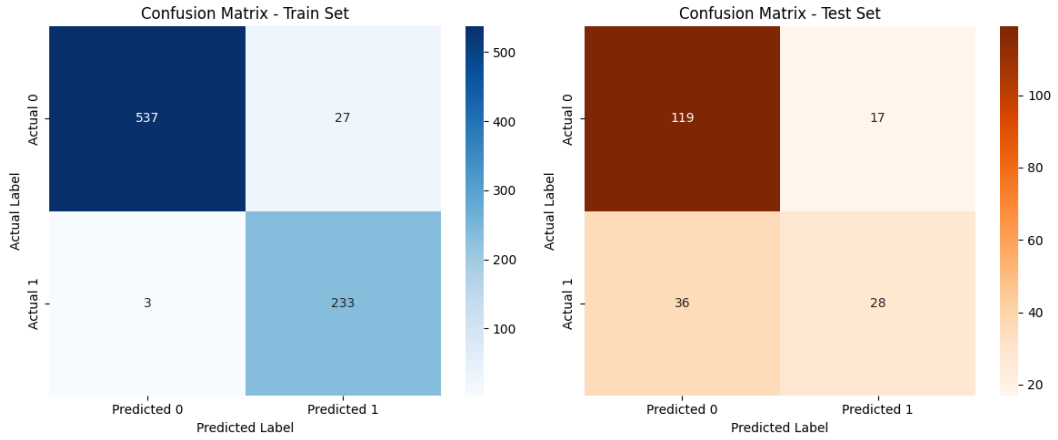
To address class imbalance, we set class_weight='balanced', and optimized the two most important hyperparameters using GridSearchCV:

- C: Controls the trade-off between margin size and classification error
- gamma: Defines the influence of a single training example

```
from sklearn.svm import SVC

svm_clf = SVC(kernel="rbf", class_weight='balanced')
svm_clf.fit(x_prepared, y_train)

params1 = {
    'gamma': np.linspace(0.01, 1, 30),
    'C': np.linspace(0.01, 0.99, 10)
}

grid_search_model4 = GridSearchCV(svm_clf, params1, cv=5, scoring="accuracy")
grid_search_model4.fit(x_prepared, y_train)
```

Best Parameters Found via GridSearch:

- C = 9.19
- Gamma = 0.19
- Cross-validated Accuracy Score: 0.72875



- The SVM model overfits the training set, achieving almost perfect recall on training data but performing worse than Models 2 and 3 on the test set.
- Despite tuning for accuracy, the model's test recall (0.437) is still higher than Model 1's baseline.
- This model captures complex boundaries well but is highly sensitive to hyperparameter settings and may require recalibration with recall-based optimization.

4. Model comparation

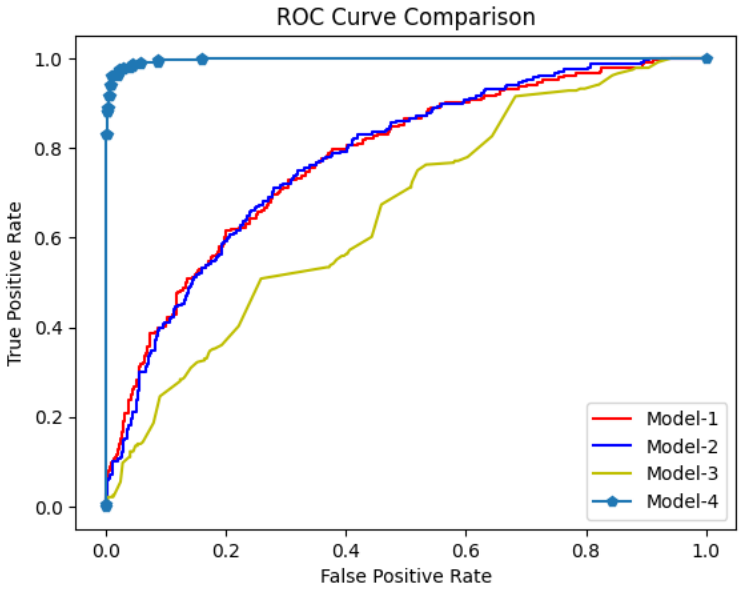
To evaluate and compare model performance, we focused on key metrics: Recall, Precision, F1-score, and the number of False Negatives and True Positives. Since our business goal is to reduce missed defaulters, we prioritized Recall.

The table below summarizes the test performance for each model:

Metric	Model 1	Model 2	Model 3	Model 4
Recall	0.390625	0.625	0.546875	0.4375
Precision	0.609756	0.454545	0.372340	0.622222
F1-score	0.47619	0.526316	0.443038	0.513761
False Negatives	39	24	29	36
True Positives	25	40	35	28

4.1. ROC Curve Comparison

We also plotted the ROC curves to visualize the trade-off between True Positive Rate (Recall) and False Positive Rate across all models.



- Models 2 and 4 have better ROC curves, especially model 4, indicating stronger discrimination power.
- Model 3's curve sits lower, indicating less effective separation of classes.
- Model 2 achieves the best balance between sensitivity (recall) and overall robustness.

4.2. Conclusion of Model Comparison

- Model 2 (Weighted Logistic Regression + GridSearch) achieved the best recall (0.625), lowest false negatives (24), and strong generalization, making it the most suitable for this business scenario.
- Model 4 (SVM) had the highest precision (0.62) but higher false negatives than Model 2.
- Model 3 (ElasticNet) was effective for feature selection but didn't outperform Model 2 in recall or F1.
- Model 1 (Default Logistic) served as a baseline and performed the worst on recall.