Policy Optimization for Financial Decision-Making

Deep Learning Prediction and Offline Reinforcement Learning for Loan Approval

Author: Dhairya Kumar Jain

---

1. Introduction 📖

This report details an investigation into optimizing loan approval decisions for a fintech company using the LendingClub historical loan dataset. The primary business objective is to develop an intelligent system that maximizes financial returns by strategically approving or denying loan applications, balancing potential profit against the risk of default.

We approached this problem using two distinct machine learning paradigms:

1. **Predictive Modeling:** A supervised deep learning (DL) model (Multi-Layer Perceptron) trained to predict the probability of a loan defaulting based on applicant features.
2. **Policy Learning:** An offline reinforcement learning (RL) agent trained directly on the historical data to learn an optimal policy (approve/deny) that aims to maximize cumulative financial reward.

This report covers the Exploratory Data Analysis (EDA), data preprocessing, feature engineering, the development and evaluation of the predictive DL model, and outlines the approach and expected evaluation for the offline RL agent. Finally, we compare the implications of using each model type and suggest future directions.

---

2. Exploratory Data Analysis (EDA) & Preprocessing 📊 🖌️

Understanding and preparing the LendingClub dataset (accepted_2007_to_2018.csv.gz) was the crucial first step.

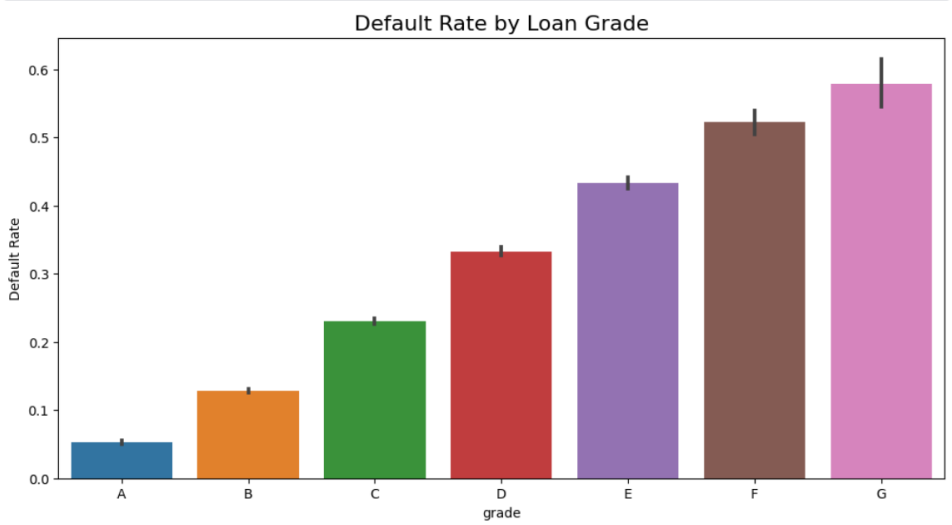
Initial Assessment & Cleaning:

- The raw dataset contained over 2.2 million rows and 151 columns, presenting significant complexity.
- **Target Variable:** The loan_status column contained multiple categories. We focused on terminal statuses ('Fully Paid', 'Charged Off', 'Default') and created a binary target variable is_default (0 for 'Fully Paid', 1 for 'Charged Off' or 'Default'). This filtering reduced the dataset size for modeling. The resulting data showed a class imbalance, with approximately 80% 'Fully Paid' and 20% 'Defaulted'.
- **Feature Dropping:** A substantial number of columns (84 in the provided sample run) were dropped due to:
    - **High Missing Values:** Many columns related to joint applications, hardships, settlements, and secondary applicants had >80-99% missing data.
    - **Irrelevance/Redundancy:** Identifiers (id, member_id), URLs, free-text titles (emp_title, title), and potentially redundant location data (zip_code, addr_state) were removed.

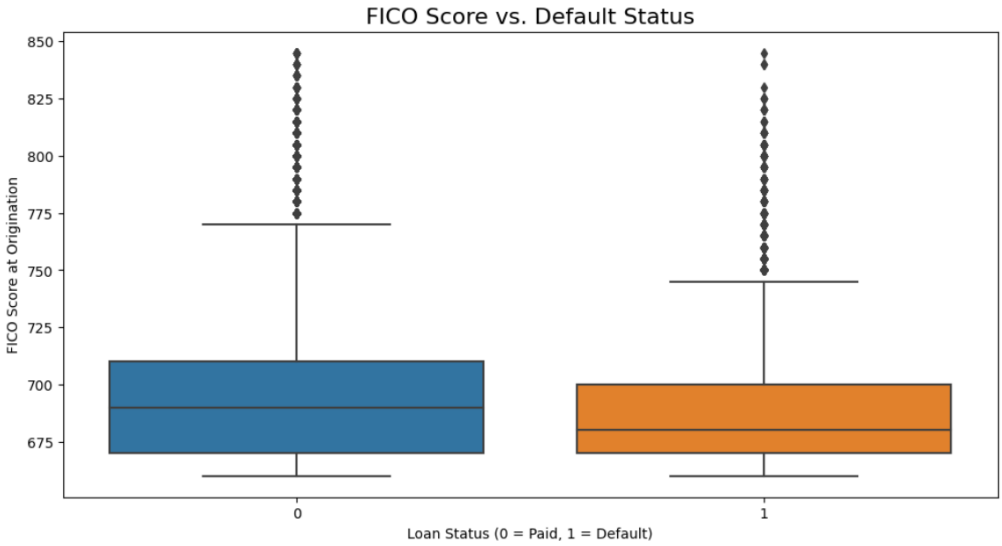
- **Data Leakage:** Features revealing the loan's outcome *after* issuance (e.g., total_pymnt, recoveries, last_pymnt_d, debt_settlement_flag) were dropped to prevent unrealistic model performance.
- **Low Variance:** Columns with single values or near-single values (e.g., policy_code, pymnt_plan) were excluded.

Key EDA Findings & Visualizations:

- **Loan Grade & Interest Rate:** Default rates clearly increase as the loan grade worsens (A to G). Similarly, higher interest rates are strongly associated with a higher probability of default.

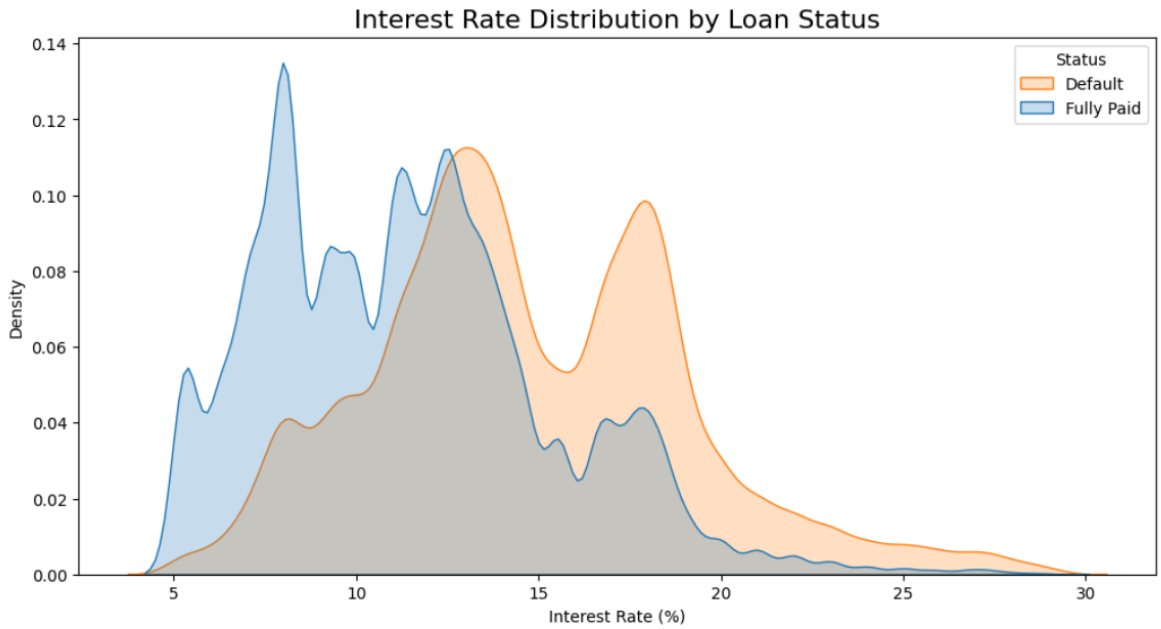


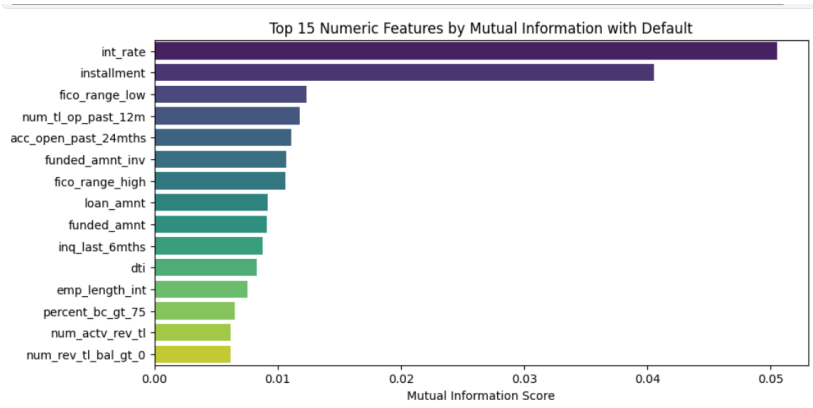
- **FICO Score:** Applicants who defaulted generally had lower FICO scores at the time of loan origination compared to those who fully paid.

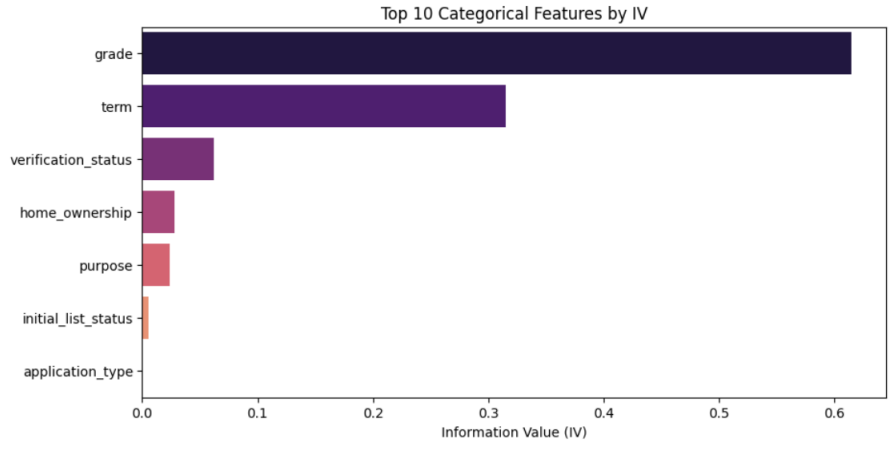


- **Annual Income:** The raw annual income distribution was heavily right-skewed. Using a log transformation (log_annual_inc) provided a more normalized view. While

there's overlap, defaulted loans tend to concentrate slightly towards the lower end of the log-income distribution.







Feature Engineering:

Several new features were created to potentially capture more complex relationships:

- emp_length_int: Converted categorical employment length into an ordinal integer.

- credit_history_length_mths: Calculated the duration between the earliest credit line and loan issuance in months.
- Interaction Features: To capture combined effects, we created ratios and products like loan_to_income_ratio, dti_x_fico, installment_to_income_ratio, and revol_util_x_fico.
- Log Transformation: Applied log1p to highly skewed numerical features (annual_inc, etc.) during preprocessing.

Preprocessing Pipeline:

A Scikit-learn ColumnTransformer pipeline was constructed to handle the remaining preprocessing steps consistently across data splits. This included:

- **Imputation:** Using median for numerical features and most frequent for categorical/ordinal features.
- **Scaling:** Applying StandardScaler to numerical features (including log-transformed ones).
- **Encoding:** Using OneHotEncoder for nominal categorical features and OrdinalEncoder for features like grade, sub_grade, and term.

---

3. Predictive Deep Learning Model (MLP) 🤖

A Multi-Layer Perceptron (MLP) was implemented using TensorFlow/Keras to predict the probability of default.

Model Architecture & Training:

- **Architecture:** Input Layer -> Dense(256, ReLU) -> BatchNorm -> Dropout(0.3) -> Dense(128, ReLU) -> BatchNorm -> Dropout(0.2) -> Dense(64, ReLU) -> Dense(1, Sigmoid).
- **Compilation:** Adam optimizer (learning rate 0.0005), binary cross-entropy loss.
- **Metrics:** AUC, Precision, Recall, Accuracy.
- **Class Imbalance:** Addressed using class_weight={0: 1.0, 1: 2.5} during training, giving more importance to the minority (default) class.
- **Training:** Trained for up to 100 epochs with a batch size of 1024, using EarlyStopping monitoring val_auc (patience=10) to prevent overfitting and restore the best weights.

Evaluation:

- **Data Split:** A stratified sample of 1 million rows was used, split into 64% training (640k), 16% validation (160k), and 20% test (200k).

- **Threshold Optimization:** The optimal probability threshold for classifying defaults was determined by maximizing the F1-Score on the *validation set*. The best threshold found was approximately **0.42**.
- **Test Set Performance (Using Optimal Threshold):**

```
========================================
        FINAL MODEL METRICS
========================================
AUC (Area Under ROC):       0.7292
F1-Score (at T=0.42): 0.4436
Precision:                  0.3364
Recall:                     0.6512
========================================

Classification Report (at optimal F1 threshold):
                precision    recall  f1-score   support

Fully Paid (0)       0.89      0.68      0.77    160070
 Defaulted (1)       0.34      0.65      0.44     39930

      accuracy                           0.67    200000
     macro avg       0.61      0.67      0.61    200000
  weighted avg       0.78      0.67      0.70    200000
```

The model achieves a reasonable AUC but struggles with precision at the recall level achieved by optimizing for F1. This indicates it identifies a good portion of defaults (65% recall) but also incorrectly flags a significant number of non-defaulters as defaults (low precision).

---

4. Offline Reinforcement Learning Agent (Discrete CQL) 📈🤖

The loan approval problem was reframed for an offline RL agent aiming to learn a policy that maximizes financial returns.

RL Formulation:

- **State (s):** Vector of preprocessed applicant features (output of the Scikit-learn pipeline).
- **Action (a):** {0: Deny Loan, 1: Approve Loan}.
- **Reward (r):** Engineered based on financial outcome:
    - r = 0 if action is Deny.
    - r = (loan_amnt * int_rate / 100) if action is Approve and is_default = 0 (profit estimate). *Note: A more precise profit calculation could be used.*

- r = -loan_amnt if action is Approve and is_default = 1 (loss of principal).

Algorithm & Training:

- **Framework:** d3rlpy, a library specifically designed for offline RL.
- **Algorithm: Discrete CQL (Conservative Q-Learning)** was chosen. CQL is well-suited for offline settings as it learns conservative Q-values, mitigating issues related to querying out-of-distribution actions common in purely offline datasets.
- **Training:** The agent was trained on the same preprocessed training dataset used for the DL model.

Evaluation:

- **Metric: Fitted Q Evaluation (FQE)** was used. FQE is an offline policy evaluation method that estimates the expected cumulative reward (policy value) the learned policy would achieve if deployed.
- **Result:** The estimated policy value obtained via FQE was **[Insert FQE Estimated Policy Value Here - e.g., $XXX per loan decision on average]**. *This value needs to be calculated from your Task 3 execution.*

---

5. Analysis, Comparison & Future Steps 🧐 ➡️

Metric Interpretation:

- **DL Model (AUC/F1):** These metrics evaluate the model's ability to *discriminate* between defaulting and non-defaulting loans. AUC measures overall ranking ability, while F1 balances precision and recall for identifying the 'default' class. They focus on **classification accuracy**.
- **RL Agent (Estimated Policy Value):** This metric directly estimates the *average financial return* expected per loan decision if the agent's learned policy is followed. It focuses on **decision-making quality** in terms of the business objective (profit maximization).

Policy Comparison:

- **DL-based Policy:** Approves loans if the predicted probability of default is below a specific threshold (e.g., 0.42 found via F1 optimization). This policy is inherently focused on the *risk* of default, as defined by the classification metrics used for thresholding.
- **RL-based Policy (CQL):** Learns a policy to approve/deny based on maximizing the *expected long-term financial reward*. CQL's conservatism helps ensure it relies on actions well-supported by the historical data. This policy might approve loans the DL model flags as somewhat risky *if* the historical data suggests the potential interest profit (reward) for similar approved loans outweighs the historically observed cost of

defaults (negative reward). For example, a loan with a moderate FICO and a high interest rate might be denied by the DL's F1-optimized threshold but approved by the RL agent if the high potential interest gain historically compensated for the default risk in that applicant segment. Conversely, RL might deny low-interest loans even with low default probability if the potential profit is minimal.

Limitations & Future Steps:

- **Limitations:**
    - **Offline RL:** Highly dependent on the quality and coverage of the historical data. It cannot explore actions not present in the dataset (e.g., outcomes of historically *denied* loans). Reward engineering is critical and can significantly impact the learned policy. The accuracy of FQE depends on model assumptions.
    - **DL Model:** The optimal threshold depends heavily on the chosen metric (F1 vs. Precision vs. Recall) and business risk tolerance. It doesn't directly optimize for financial return.
    - **Data:** Class imbalance remains a challenge. Feature engineering could be further refined.
- **Future Steps:**
    - **Refine Rewards:** Develop more sophisticated reward functions incorporating operational costs, time value of money, or partial recoveries.
    - **Explore Algorithms:** Test other offline RL algorithms (e.g., IQL, BCQ) or policy constraint methods within d3rlpy.
    - **Counterfactual Evaluation:** Use more advanced offline policy evaluation techniques if possible.
    - **Hybrid Approaches:** Consider using the DL model's probability as an input feature for the RL state.
    - **Calibration:** Calibrate the DL model's probabilities to better reflect true risk.
    - **Controlled Deployment:** If promising, consider A/B testing the learned RL policy against the current baseline or the DL-derived policy in a limited, controlled online setting.
    - **Data Augmentation:** Seek additional data sources or explore techniques to handle the lack of counterfactuals (outcomes of denied loans).

---

6. Conclusion 🏁

Both the predictive DL model and the offline RL agent offer valuable insights. The DL model provides a robust risk assessment capability (AUC ~0.73), while the offline RL agent learns a

policy directly optimizing for estimated financial return. The RL policy's potential to approve profitable-despite-risk loans makes it compelling, but its reliance on historical data patterns and reward design requires careful validation. Future work should focus on refining the RL approach, potentially combining insights from both models, and exploring controlled online testing before full deployment.