# Policy Optimization for Financial Decision-Making: A Comparative Analysis

Project Report

October 26, 2025

**Abstract**

This report details the end-to-end process of developing a loan approval system using the LendingClub dataset. We frame the business problem in two distinct ways: first, as a supervised classification task to predict default risk (Task 2), and second, as an offline reinforcement learning (RL) task to directly optimize for financial return (Task 3). We build, train, and tune deep learning models for both tasks. The analysis reveals a significant divergence in the resulting policies: the predictive model approves 62.1% of loans, while the profit-maximizing RL model learns an extremely conservative policy, approving only 6.78%. This divergence highlights the critical difference between a model that predicts *risk* and one that understands *cost*.

## 1 Task 1: EDA and Preprocessing

The initial dataset (`accepted_2007_to_2018.csv`) was sampled to 200,000 rows for iterative development. We filtered for terminal loan statuses ("Fully Paid" and "Charged Off"), establishing a binary classification target where "Charged Off" was mapped to 1 (Default).

### 1.1 Feature Engineering and Selection

A core challenge was "data leakage." Features like `total_pymnt` or `recoveries` were discarded as they contain information about the loan's outcome. We selected 13 non-leaky features for their high predictive potential and low missing-value rates.

- **Numeric:** `loan_amnt`, `int_rate`, `installment`, `annual_inc`, `dti`, `fico_range_low`, `revol_bal`, `revol_util`, `total_acc`, `mort_acc`.
- **Categorical:** `term`, `home_ownership`, `verification_status`.

### 1.2 Reward Engineering (for Task 3)

The `reward` signal was engineered to represent the business objective:

- **If Fully Paid (Target=0):** `reward = loan_amnt * int_rate` (Profit)
- **If Defaulted (Target=1):** `reward = -loan_amnt` (Loss of Principal)

### 1.3 Data Cleaning

A `ColumnTransformer` pipeline was built to handle missing values (median for numeric, most frequent for categorical), scale all numeric features (StandardScaler), and one-hot encode categorical features. This pipeline was saved to `preprocessor.joblib` to ensure consistent transformation across all tasks.

# 2 Task 2: Model 1 - Predictive DL Classifier

The goal of this model was to predict the probability of default.

## 2.1 Methodology

We built a Multi-Layer Perceptron (MLP) in PyTorch. The key challenge was the severe class imbalance (Defaults are rare). We addressed this by:

1. Using a **weighted loss function** (`BCEWithLogitsLoss` with `pos_weight`) to penalize the model more for misclassifying the rare ”Default” class.
2. Performing hyperparameter tuning on epochs, learning rate, and network depth.
3. Evaluating the model using **AUC** (for overall discriminative power) and **F1-Score** (to balance precision and recall).
4. Finding the optimal **decision threshold** that maximized the F1-Score, as the default 0.5 threshold is meaningless for an imbalanced problem.

## 2.2 Results

The final optimized model achieved an AUC of 0.7311. By setting the optimal decision threshold to 0.5160, we achieved a strong F1-Score of 0.4470, driven by a high Recall of 0.6485. This demonstrates the model’s ability to successfully identify 65% of all actual defaults.

# 3 Task 3: Model 2 - Offline RL Agent

The goal of this model was to learn a policy that directly maximizes the engineered `reward` (profit).

## 3.1 Methodology

We pivoted from a complex `d3rlpy` implementation (which suffered from API ”data misalignment”) to a ”simple method with a powerful model.” We built a PyTorch MLP that functioned as a Q-network.

- **Input:** Applicant features (the ”State”).
- **Output:** Two Q-values: the expected profit for ”Deny” (Action 0) and the expected profit for ”Approve” (Action 1).

The model was trained like a regression task. We taught it to predict the scaled profit of “Deny” (a known value) and the observed scaled profit of “Approve” (from the historical data).

## 3.2 Results

The model’s performance was measured by simulating its decisions on the test set. The agent learned an extremely conservative policy, choosing to **approve only 6.77% of loans**. This policy yielded an **Average Scaled Profit of 0.0543**, confirming that the model successfully found a small, profitable niche of applicants while avoiding the high risk associated with the general population.

# 4 Task 4: Analysis and Comparison

## 4.1 Key Metrics Summary

The final results for both champion models highlight their different objectives.

Table 1: Key Model Performance Metrics on Test Set

| Model | Metric | Score |
|---|---|---|
| **Task 2: DL Classifier** | **AUC** | **0.7311** |
| | **Best F1-Score** | **0.4470** |
| | Optimal Threshold | 0.5160 |
| | Recall (at Best F1) | 0.6485 |
| | Precision (at Best F1) | 0.3410 |
| **Task 3: RL Q-Network** | **Avg. Scaled Profit** | **0.0543** |
| | Approval Rate | 6.77% |

## 4.2 Explanation of Metrics

- **AUC & F1-Score (for Task 2):** These are the correct metrics for a *classifier* on an imbalanced dataset. **AUC (0.7311)** tells us the model is significantly better than random chance at distinguishing a ”good” loan from a ”bad” one. **F1-Score (0.4470)** is crucial as it measures the *balance* between Precision and Recall. In our case, we prioritized **Recall (0.6485)**, accepting a lower Precision to ensure we caught as many costly defaults as possible.
- **Average Scaled Profit (for Task 3):** This is the key metric for the *RL agent* because it directly measures the business objective. It represents the model's ability to maximize financial return. Our positive score of **0.0543** (on a [0, 1] scale) proves the policy is profitable, on average, for the decisions it makes.

## 4.3 Policy Comparison: Prediction vs. Profit

The most significant finding of this project is the stark difference between the two policies.

- The **DL Classifier (Task 2)**, optimizing for prediction, approves **62.1%** of loans. It identifies that most applicants have a < 51.6% chance of default and deems them ”safe enough.”
- The **RL Agent (Task 3)**, optimizing for profit, approves only **6.77%** of loans.

Our analysis of the 19,485 loans where the models disagreed was revealing. The RL model would **DENY** applicants that the DL model would **APPROVE**.

A key example was an applicant who *Fully Paid* but had a very low reward (True Reward: $0.05).

- **DL Model:** Predicted a very low 8.86% default risk. Decision: **APPROVE**.
- **RL Model:** Predicted Q(Deny) = 0.7893 vs. Q(Approve) = 0.7883. Decision: **DENY**.

**Analysis:** The RL model learned what the DL model could not: even for an extremely safe applicant, the tiny potential profit was *not worth the risk*. The risk-adjusted return was negative, so the model correctly chose the guaranteed $0 profit of denying the loan. The DL model, blind to cost, only saw a ”safe” bet and approved it.

### 4.4 Future Steps and Limitations

- **Deploy a Hybrid Model:** Neither model is perfect. The DL model is too liberal, and the RL model is too conservative. A future strategy would be to use the DL model as a ”screener” to filter out high-risk applicants (e.g., P(Default) > 0.6) and then feed the remaining applicants to the RL agent for a final, profit-based decision.
- **Limitations:** Our models only make a binary ”Approve/Deny” decision. A more advanced system would also set the *interest rate* and *loan amount*. The RL framework is a perfect fit for this, where the action space would become continuous (e.g., ‘action = [amount, rate]‘).
- **Future Data:** The clear next step is to collect data on the *applicants we denied*. Our dataset is biased, as it only contains approved loans. We have no information on the ”what if” scenario of approving a loan we historically denied. Collecting this (e.g., through small, randomized A/B tests) would allow for true online RL and a far more robust policy.