Shodh-AI ML Assignment Final Report

Policy Optimization for Financial Decision-Making: DL vs. Offline RL

Author: Hammad Shaikh | Manipal University Jaipur

GitHub Link: https://github.com/Hammad-1105/Loan-Approval-Shodh-AI.git

1. Executive Summary

This project explores the transition from predicting credit risk to *optimizing* financial returns in a distressed lending portfolio. Using a substantial **subset of 200,000 data-points** from the LendingClub dataset (2007-2018), I developed two distinct AI systems to automate loan approval decisions:

1. **A Supervised Deep Learning Classifier (MLP) :** Predicts the probability of default.
2. **An Offline Reinforcement Learning (RL) Agent:** Learns an optimal approval policy directly from historical outcomes using *Conservative Q-Learning* (CQL).

The Core Finding: While both models outperformed the historical human-baseline, the **Optimized Deep Learning Classifier** emerged as a superior policy. By rigorously tuning the decision threshold using various experimental values, the DL model created a “strict” policy. In contrast, the RL agent learned a more "lenient" strategy, while this successfully improved upon human returns

This counterintuitive result highlights that a well-calibrated supervised model can sometimes outperform complex reinforcement learning when the "safe" action (Deny) is strictly defined.

1.1 Key Results (Task 4.1 from Project Statement)

The table below summarizes the financial performance of each policy on the **held-out test set**.

Model Strategy	Primary Metric	Performance	Est. Policy Value (EPV)	Net Improvement
Human Baseline	N/A	N/A	-$1,805.50 per loan	-
RL Agent (CQL)	EPV	95.2% Approval	-$1,339.02 per loan	+$466 / loan (+26%)
DL Model (Tuned MLP)	AUC / F1	0.741 / 0.456	-$273.19 per loan	**+$1,532 / loan (+85%)***

**DL Model performed the best amongst all the policies with a net improvement of 85% in loss reduction.*

The portfolio is financially toxic (Average Human Loss: -$1,805). The RL agent learned to be "safer" than humans, rejecting the worst ~5% of loans. However, the DL model, when tuned to a strict threshold of 0.25, rejected a significantly larger portion of risky loans, minimizing the catastrophic losses.

2. Methodology & Model Development

2.1 Phase 1: EDA & “The Financial Imbalance” [Code link: *01_data_preprocessing.ipynb*]

The dataset contained 150+ features, many of which (e.g. total_pymnt’, ‘recoveries’) constituted *data leakage* – information not available at the time of real-world banking applications.

- **Data Preprocessing**: I removed 58 columns with >30% missing data (Fig. 1) and strictly purged leakage features; used **one-hot** for cat. features & **standard scaler** for num. scaling.
- **Financial Imbalance**: EDA revealed a critical asymmetry – A successful loan yields ~$2,000 in interest, while a default costs up to $35,000 in principle. This explicitly framed the problem as *"Loss Avoidance"* rather than *"Profit Maximization"*.
- **Feature Selection**: Correlation analysis (Fig. 2) identified redundant pairs (e.g., 'loan_amnt' vs. 'installment', correlation >0.95), which were removed; Selected **61 final features**.

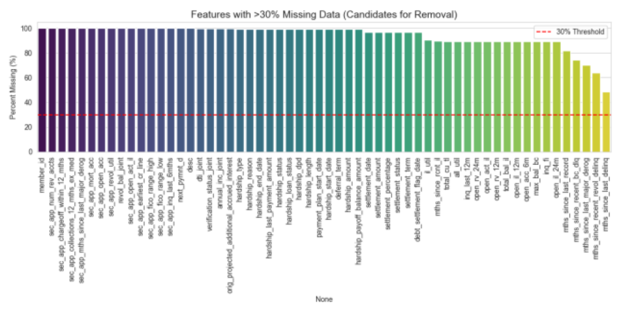

Fig. 1 Features with >30% missing data

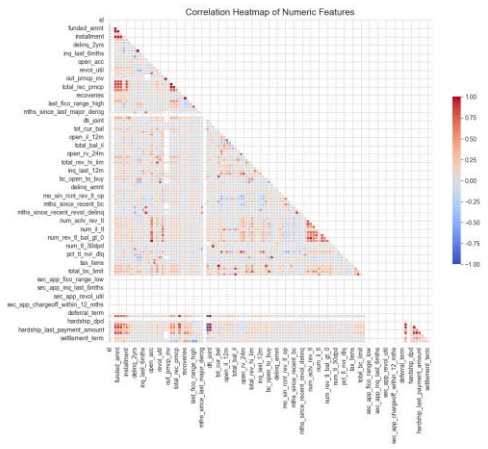

Fig. 2 Correlation Heatmap

2.2 Phase 2: The Deep Learning Classifier [Code link: *02_deep_learning.ipynb*]

- **Architecture**: I implemented a *MLP* using *PyTorch* with a funnel architecture (256 → 128 → 64 neurons) and 30% *Dropout* to prevent overfitting, ran for 50 epochs (Fig. 3).
- **Optimization Strategy**: Standard training yielded an F1-score of 0.00 because the model defaulted to approving everyone (due to the 80/20 class imbalance).
- **The Fix**: By shifting the decision boundary from 0.50 to **0.25**, the F1-score increased to **0.46** (Fig. 4). This created a conservative policy that rejects any applicant with >25% predicted risk.

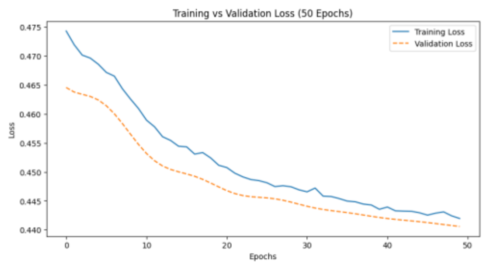

Fig. 3 Training vs Validation Loss

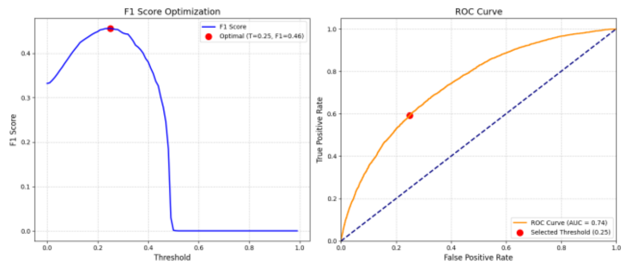

Fig 4. ROC Curve showing AUC 0.74

2.3 Phase 3: The Offline RL Agent [Code link: *03_offline_rl.ipynb*]

- **The problem**: The targeted dataset (*accepted_2007_to_2018.csv*) only contained "Approved" loans (Action=1). The agent had no concept of "Denial" (Action=0).
- **Counterfactual Augmentation (Solution)**: I doubled the dataset size by creating a synthetic "Deny" action for every applicant with a Reward of $0. I defined Reward for "Approve" by: R(Approved | Paid) = +(loan_amnt * int_rate), R(Approved | Defaulted) = -loan_amnt.
- **Algorithm:** I utilized **Discrete Conservative Q-Learning** using *d3rlpy*. CQL was chosen for its *pessimistic nature*—it suppresses the value of risky, unknown actions, which is essential.

3. Critical Analysis & Metrics (Task 4.2 from Project Statement)

3.1 Why AUC & F1 for Deep Learning?

- **AUC (0.741)** measures the model's ability to *rank* applicants. It tells us: "If we pick a random defaulter and a random payer, the model correctly identifies the defaulter 74% of the time."
- **F1 Score (0.456)** measures the balance between Precision and Recall. It was the optimization target for our threshold tuning, ensuring we didn't just default to the "Approve All" strategy.

3.2 Why EPV for Reinforcement Learning?

- **Estimated Policy Value (EPV)** translates "Accuracy" into "Dollars." It represents the expected financial return per loan under the agent's policy.
- In business context, **EPV is the superior metric**. Model with high accuracy could still bankrupt the bank if it misses just a few large $35k defaults. EPV captures this magnitude directly.

4. Policy Comparison: "The Divergence" (Task 4.3 from Project Statement)

The two models disagreed on approximately **30%** of the test cases. Analysing this *divergence* reveals why the DL model won. [Code link: *04_analysis_and_comparision.ipynb*]

- **RL Behaviour**: The RL Agent was **Too Lenient**. It had an approval rate of **95.2%**, effectively acting as a "Filter" that only removed the absolute worst applicants.
- **DL Behaviour**: The DL model (at threshold 0.25) had an approval rate of **65.6%** and was strict. It aggressively rejected 15,000+ applicants that the RL agent would have approved.

While DL incurred "False Negatives" (rejecting good loans), this cost was negligible compared to the massive savings from avoiding "False Positives" (approving defaults).

4.1 Divergent Case Study: Applicant #45

I explicitly coded a case to highlight 1 example of an applicant, in which DL denied, but RL approved:

- **Profile:** High-interest rate loan.
- **DL Decision: Deny** (Predicted Risk 30.29% > 25%).
- **RL Decision: Approve** (Action 1).
- **Actual Outcome:** Defaulted (Loss: -$6,000).

The RL agent's approval resulted in a **Net Loss of $6,000**. The DL model avoided this loss ($0 impact).

4.2 Limitations: Our model assumed a "Total Loss" (-100% Principal) on default. Banks recover ~30% through collections. This assumption likely made the models overly risk averse.

4.3 Future Scope & Recommendations (Task 4.4 from Project Statement)

Immediate Deployment: I recommend deploying the **Deep Learning Model (Threshold 0.25)** as the primary gatekeeper. It offers an 85% reduction in losses and is computationally cheaper to run.

Future Development: Train a **TD3+BC** agent. CQL acts as a lower-bound constraint; a different offline algorithm might better balance risk and reward without being as passive as our current agent.

Data Collection: Collect data on "Rejected" applications using a small subset to validate the counterfactual assumption that "Deny = $0 Reward".