Gardent B Report 1

a) What has been done

1. Literature Review

1.1 LinUCB

- Li, Lihong, et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation.", (2010)
- Introduce a foundational application of contextual bandits in the domain of personalized news recommendations. This paper presents the LinUCB algorithm, a linear UCB-based method that uses user-specific contextual information to adaptively select news articles that align with user interests.
- **Methodology**: LinUCB uses a linear model for reward estimation, combining contextual information with a UCB exploration strategy. The algorithm adapts to user preferences by updating estimates as new data is observed, which allows it to balance exploration of new content and exploitation of known preferences.
- **Contributions**: By validating LinUCB on a large-scale news recommendation platform, the authors show its ability to handle dynamic and diverse user preferences. The linear model is computationally efficient and straightforward to implement.
- **Limitations**: Reliance on linear assumptions, limits its capability to capture complex, non-linear relationships between context and rewards. This can reduce its effectiveness in settings with high-dimensional features or intricate user behavior patterns.

1.2 Thompson Sampling (Linear)

- Shipra Agrawal, Navin Goyal, “Thompson Sampling for Contextual Bandits with Linear Payoffs” (2012)
- Introduce a Bayesian approach to the contextual bandit problem. This paper adapts Thompson Sampling, a popular Bayesian exploration strategy, for scenarios where the payoff is linearly dependent on the context.
- **Methodology**: Thompson Sampling uses probabilistic sampling, unlike UCB-based methods that rely on deterministic upper bounds. This allows it to balance exploration and exploitation naturally. The method updates the posterior distribution of reward estimates and selects actions based on sampled payoffs.
- **Contributions**: It outperforms UCB-based methods, especially when reward distributions are uncertain. It is widely used in applications where probabilistic modeling of rewards offers advantages.
- **Limitations**: Hard to extend to non-linear and high-dimensional contexts. The algorithm's reliance on accurate posterior distributions can be problematic with complex, non-linear data.

1.3 NeuralUCB

- Zhou, Dongruo, et al. "Neural Contextual Bandits with UCB-based Exploration.", (2020)
- Introduce an advance contextual bandits by incorporating neural networks to capture non-linear relationships between context and reward. NeuralUCB uses neural networks to approximate reward functions, enabling it to handle complex, high-dimensional contextual information that linear models struggle with.
- **Methodology**: NeuralUCB combines neural networks with UCB-based exploration. By estimating the reward function through neural networks, NeuralUCB can adapt to non-linear contexts, which improves performance in complex scenarios like personalized recommendations and real-time bidding.
- **Contributions**: NNs allow for a more expressive representation of context, addressing the limitations of linear models in capturing complex interactions. NeuralUCB shows strong results in simulations, especially in environments with non-linear and high-dimensional reward structures.
- **Limitations**: The computational complexity and training time of neural networks can hinder real-time use. Additionally, tuning neural network architectures to achieve optimal performance adds further complexity.

2. Background Research

For this project, the goal is to understand the core concepts, challenges, and recent advancements in **contextual bandits**, particularly in offline settings and their application to loan recovery strategies.

2.1 Contextual Bandits:

1. Contextual bandits are a type of reinforcement learning that selects actions (e.g., recovery strategies) based on context (e.g., loan data). The goal is to maximize rewards by balancing exploration (trying new strategies) and exploitation (using known successful ones).
2. **Challenge**: Only the reward for the chosen action is observed, making it hard to infer the effectiveness of unselected actions.

2.2 Exploration-Exploitation Trade-off:

Exploration refers to trying different strategies to discover which one works best, while exploitation involves using the strategy that is currently considered optimal. The trade-off is crucial, especially with **limited data** and **uncertainty** about the best recovery action.

2.3 Offline Contextual Bandits:

1. Offline bandits work with a **fixed dataset** (historical loan data) to evaluate and optimize policies. This approach is critical for UTP loan recovery, where real-time deployment isn't possible.
2. **Challenges**:
   - Distribution Shift: The data may not represent future loan behaviors accurately.
   - Historical Policy Bias: The dataset might have biases due to past decision-making policies.

2.4 Reward Estimation and Evaluation:

1. In contextual bandits, **rewards must be well-defined**. For UTP loans, rewards could include:
   - Success in returning a loan to performing status：a borrower resuming regular loan payments after being delinquent, meaning the loan is no longer at risk of default.
   - Partial recovery through a DPO：The lender settles for a reduced payment instead of the full loan amount.
   - Financial results from Liquidation：The lender recovers funds by selling the borrower's assets.
2. Offline Policy Evaluation methods, like Inverse Propensity Scores (IPS) and Doubly Robust Estimation, assess policies **using historical data without real-time feedback**.

2.5 Model Selection

2.5.1 Criteria for Model Selection

Given the task of UTP loan recovery, the chosen models must address the following:

- **Exploration vs. Exploitation**: Balance trying different strategies (RTP, DPO, Liquidation) with relying on proven successful strategies.
- **Uncertainty in Rewards**: Handle the uncertainty and bias present in simulated historical rewards.
- **Scalability**: Ensure the model can handle large datasets and non-linear relationships in the data.

2.5.2 Model Options

- **LinUCB**:

1. Advantage: Simple, efficient for linear reward estimation.
2. Limitation: Limited in handling complex, non-linear relationships.

- **Gaussian Process (GP) Thompson Sampling**:

1. Advantage: Captures reward uncertainty, good for non-linear environments.
2. Limitation: Computationally expensive, less scalable for large datasets.

- **Neural LinUCB**:

1. Advantage: Uses neural networks for reward estimation, handling non-linearities.
2. Limitation: More complex to implement and tune.

- **BNN Thompson Sampling**:

1. Advantage: Bayesian Neural Networks (BNNs) provide uncertainty estimation, useful for sparse or noisy data.
2. Limitation: Computationally intensive, requires Bayesian expertise.

3. Code Reading

3.1 Contextual Bandits 1 (LinUCB, Regret Minimization and Exploration vs Exploitation)

1. **Main Algorithm:** linUCB implements LinUCB, which selects arms based on confidence bounds.

   Algorithm Steps:

   a. Initialization: Arrays arm_choice, r_payoff, theta, and p are initialized to store the chosen arms, payoffs, and parameter estimates for each trial. The matrix A and vector b for each arm are initialized for storing context data.

   b. Arm Selection: For each trial, the function calculates the estimated reward (theta) and the UCB score (p) for each arm. The arm with the highest score is chosen.

   c. Reward Generation: The reward for the chosen arm is generated by the generate_reward function, and A and b are updated based on the observed reward.

   d. Return Values: The function returns a dictionary containing:

      theta: Parameter estimates for each arm and trial.

      p: The UCB scores for each arm.

      arm_choice: The selected arms across trials.

      r_payoff: The rewards received.

2. **Supporting Functions:**

- make_theta: Generates the true reward coefficients (theta) for each arm using a normal distribution.
- generate_reward: Simulates rewards with dynamic noise and oscillations based on the arm chosen and the context.
- optimal_arm: Identifies the optimal arm for each trial by calculating the reward for all arms using true_theta.
- make_regret: Computes the cumulative regret by comparing the chosen arms' rewards with those of the oracle (optimal choices).
- plot_regrets, plot_estimates, plot_selected_arms: These functions are used to visualize the algorithm's performance over time by plotting regrets, convergence of parameters, and the frequency of selected arms.

3. Evaluation: The model’s performance is evaluated with accuracy, F1 score, and cumulative regret.

3.2 Contextual Bandits 2 (Decision Boundaries and Batch Learning)

1. **NeuralUCB Algorithm**

The NeuralUCB algorithm extends LinUCB using a neural network to model more complex reward functions.

Key Components:

- Model Architecture: A simple feed-forward neural network is used with two hidden layers (64 neurons each), ReLU activations, and a final output layer.
- Training: The model is trained using the Adam optimizer and mean squared error (MSE) as the loss function. Contexts are standardized using StandardScaler.
- Prediction: Predicts reward probabilities for each arm based on the neural network's output.

2. **GP Thompson Sampling**

The GP Thompson Sampling algorithm leverages Gaussian Processes to model the reward distribution and applies Thompson Sampling for exploration:

Core Steps:

- Model Setup: Gaussian Processes are used to model the reward function with a kernel (RBF) that captures relationships between contexts and rewards.
- Model Update: For each arm, the GP model is trained on historical actions and rewards. The GP is updated after each batch of data.
- Prediction: The algorithm predicts the expected reward for each action. Thompson Sampling is used to generate reward samples, helping in exploring uncertain regions.

3. **Bayesian Neural Networks (BNN)**

The Bayesian Neural Network (BNN) model uses TensorFlow Probability to model uncertainties in the reward function.

Key Components:

- Bayesian Layers: Uses DenseVariational layers to account for uncertainties in the weights. The prior and posterior distributions are estimated for each layer.
- Training: The model is trained with the Adam optimizer, and log-likelihood is maximized to update the weights.
- Thompson Sampling: A single sample from the posterior is used for action selection (Thompson Sampling).

4. **Ridge Regression and Random Strategy**

Ridge Regression is employed as a baseline linear model that fits the relationship between contexts and rewards for each arm using regularization to avoid overfitting.

Random Strategy simply selects actions randomly, serving as a lower bound comparison.

5. **Backtesting Framework**

- A backtest is performed to compare the performance of multiple algorithms (LinUCB, NeuralUCB, GP Thompson Sampling, Ridge Regression, and Random Strategy).
- Cumulative Rewards: The cumulative sum of rewards across all samples is plotted to compare the algorithms' performance.
- Uncertainty Visualizations: Uncertainty in predictions (from Bayesian models) is visualized using confidence intervals.
- Accuracy and F1 Score: Each algorithm's accuracy and F1 score are computed based on how well the actions align with the optimal policy.

3.3 Contextual Bandits 3 (Reward Functions applied to Simulated and Statlog German credit data)

Compared to 2, this code focuses on applying reward functions to both simulated data and real-world datasets like the Statlog German Credit dataset. This transition allows testing how contextual bandits and reward function strategies perform in more realistic, domain-specific scenarios involving financial data and decision-making problems (e.g., credit scoring).

3.4 Contextual Bandits 4 (Unbiased Offline Policy Evaluation with Inverse Propensity Scores)

In this code, it performs backtesting for two contextual bandit algorithms: LinUCB and Thompson Sampling with Gaussian Processes (GP). Both algorithms are evaluated using Inverse Propensity Scores (IPS), which measure the expected performance of the model against a historical policy based on observed data.

1. **Inverse Propensity Score (IPS) Evaluation**

For both algorithms, IPS is used to evaluate the performance of the model:

- IPS Historical: Measures the performance of the historical policy based on the actual action propensities observed during data collection.
- IPS Evaluation: Measures the performance of the model-based policy (LinUCB or Thompson Sampling) by comparing the actions selected by the model to the historical propensities.

The IPS scores for both the training and test sets are computed, and a higher IPS evaluation score indicates that the model-based policy (LinUCB or Thompson Sampling) is expected to perform better than the historical policy.

2. **Regret Calculation and Comparison**

For both LinUCB and Thompson Sampling, regret is calculated as the difference between the rewards from the optimal action (the best action in hindsight) and the rewards from the model's selected actions.

- Cumulative Regret: The total regret accumulated over time by not selecting the optimal actions.
- Average Regret: The average regret per sample over time.

The code tracks both cumulative rewards and cumulative regret for the historical policy and the model-based policy (LinUCB or Thompson Sampling).

3. **Results and Visualization**

For both algorithms:

- Cumulative Rewards, Cumulative Regret, and Average Regret are plotted over time, comparing the historical policy to the model-based policy.
- IPS Scores for both the historical and model-based policies are printed for both the training and test sets. These scores help to determine whether the LinUCB or Thompson Sampling model is expected to outperform the historical policy.

b) Any questions or issues

1. **RL typically involves delayed rewards, and in the context of debt recovery, the recovery outcomes often have delayed feedback. However, LinUCB and Thompson Sampling assume immediate rewards by default,**

meaning that the reward is received right after an arm is selected. How can this situation be addressed?

Answer by ChatGPT：

- Standard Contextual Bandit algorithms assume immediate rewards, which can be problematic in real-world scenarios with delayed feedback. Here are some strategies to handle delayed rewards effectively:
  - **“Ignore Delay” Strategy**: Continue selecting arms as usual without waiting for delayed rewards. Once rewards arrive, update the model. While simple, this approach can lead to higher regret as the model can't promptly adjust based on immediate feedback.
  - **Batch Updates**: Accumulate choices and update the model in batches when rewards eventually arrive. This approach helps manage delays but may miss timely adjustments, especially if delays are significant.
  - **“Ghost” Rewards**: Use estimated rewards based on historical data to update the model before actual rewards are received. This approximation can guide decision-making in environments with predictable patterns, bridging the gap until real rewards are available.
  - **Compensatory Adjustments**: Modify the frequency of exploration to counterbalance the delay. For instance, increase the probability of choosing suboptimal arms to offset the lag, preventing premature convergence to potentially suboptimal solutions.
  - **Specialized Algorithms**: Use delayed versions of UCB and Thompson Sampling that account for the waiting time in their exploration strategies, or extend the framework to Reinforcement Learning (RL) models like Q-learning, which naturally handle delayed feedback.
- By applying these strategies, Contextual Bandit algorithms can be adapted to environments with delayed rewards, maintaining a balance between exploration and exploitation despite the feedback lag.

2. How can Contextual Bandit algorithms be deployed in an online setting?

Answer by ChatGPT：

- Contextual Bandit algorithms, such as LinUCB and Thompson Sampling, are highly suitable for online deployment and can be effectively used in real-time environments. Here’s how they operate in online settings:
  - **Designed for Online Use**: Most Contextual Bandit algorithms are incremental by nature, meaning they update their models continuously with each new data point. This makes them well-suited for online scenarios where decisions need to be made in real-time based on immediate feedback.
  - **Real-Time Learning and Adaptation**: These algorithms can process live data streams, like user clicks or engagement metrics, and adjust their models on-the-fly. This allows them to dynamically balance exploration (trying new options) and exploitation (using known information) as new data comes in.
  - **Implementing Online Deployment**:
    - Contextual Bandit algorithms perform well with real-time data inputs, updating incrementally with each decision. This efficiency makes them suitable for environments with limited computational resources.

        - In cases with delayed rewards, techniques like ghost rewards or batch updates can be applied, enabling the algorithms to handle delays while still updating the model effectively.
    - **Use in Offline Scenarios**: While Contextual Bandit algorithms excel in online settings, they can also be employed offline for model training and evaluation using historical data. Offline testing is valuable for tuning parameters or experimenting with different exploration-exploitation balances
- **In summary**, Contextual Bandit algorithms are versatile and adaptable, capable of both online and offline deployment. They excel in online environments due to their real-time learning capabilities, but can also be leveraged offline for strategic planning and testing.

3. **Currently, our understanding of the goals that need to be achieved for this project is not very clear. After gaining a deeper understanding of the project's details, are we expected to innovate in terms of methodology, achieve higher predictive accuracy on the target data, or deploy the relevant methods to other types of datasets? Alternatively, is it up to us to determine the direction in which we should dive deeper?**

c) The plan until the next meeting

Research Plan

1. Deep Dive into Neural Bandits

- **Zhou et al. (2020)**: Study the architecture and training of **Neural LinUCB** for non-linear reward estimation.
- **Riquelme et al. (2018)**: Understand the application of **Bayesian Neural Networks (BNNs)** for estimating reward uncertainty.

2. Understand Offline Policy Evaluation

Learn to evaluate models using historical loan data.

- **Dudik et al. (2011)**: Study **Doubly Robust Estimation** to reduce bias in policy evaluation.
- **Nguyen-Tang et al. (2022)**: Explore **Inverse Propensity Scores (IPS)** for better model generalization.

3. Learn from Related Applications

Investigate practical applications of contextual bandits in finance and marketing. Research applications in **credit risk assessment**, **investment strategies**, and **customer retention** to identify best practices.

4. Write Custom Scripts for Selected Models

1. Choose one or more models
2. Simulate different recovery strategies for loans using hypothetical data.

3. Create scripts to test different reward functions based on the recovery outcomes (e.g., Return to Performing (RTP), Debt Payoff Offer (DPO), and Liquidation).

5. Investigate Additional Models

1. Generalized Linear Bandits: Handle non-linear rewards in loan recovery.
2. Thompson Sampling with Dirichlet Priors: Useful for selecting among multiple recovery strategies.
3. Neural Thompson Sampling: Combines neural networks with Thompson Sampling for complex rewards.
4. Collaborative Filtering Bandits: Leverages relationships between loans to improve recovery strategy selection.