Gardant B Report 2

summary

1. Statlog: We try LinUCB and GP Thompson model on Statlog dataset, and try different reward functions on it.
2. OPE: We further explored the principles and applications of Off-Policy Evaluation (OPE) and attempted to use the Open Bandit Pipeline (OBP) package.

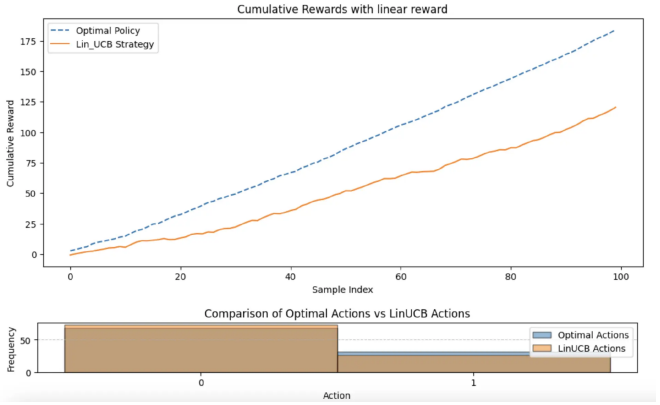
1. Statlog

Different reward functions

In conclusion, the **Tanh** function is the most effective choice, followed by the **Logarithmic and Sigmoid functions**. ReLU also performs well but is slightly less effective than the top three. The Polynomial function fails to provide useful results and may need adjustments or replacement for better performance.

- Linear

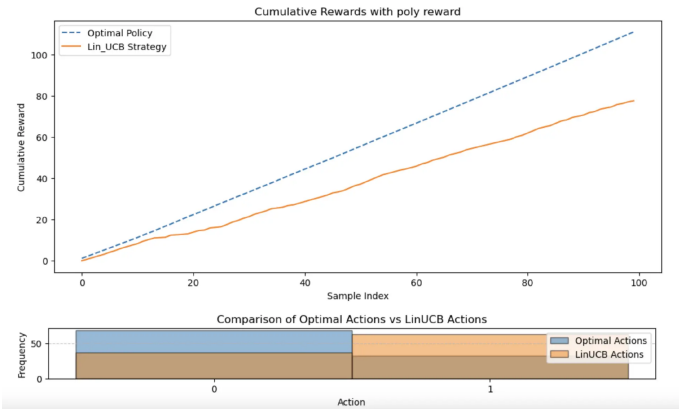
The linear function shows moderate performance, with both accuracy and F1 score around 60%. It is suitable for handling simple linear relationships but fails to capture more complex feature interactions.




```
LinUCB Accuracy: 0.6100
LinUCB F1 Score: 0.6004
```

- Polynomial

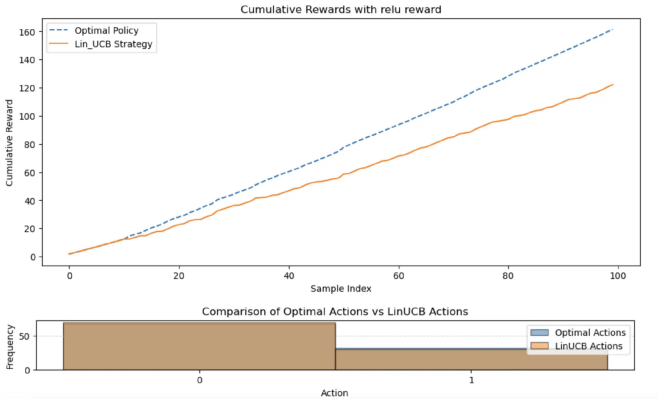
The polynomial function performed the worst. It may struggle with fitting the data properly, potentially suffering from overfitting or being overly sensitive to noise.




```
LinUCB Accuracy: 0.3100
LinUCB F1 Score: 0.3207
```

- Relu

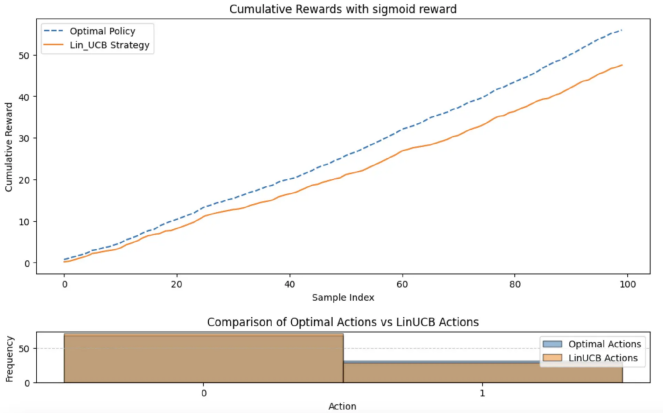
The ReLU function performs better than the linear function. It effectively captures important patterns, particularly in non-negative rewards and features.




```
LinUCB Accuracy: 0.6400
LinUCB F1 Score: 0.6368
```

- Sigmoid

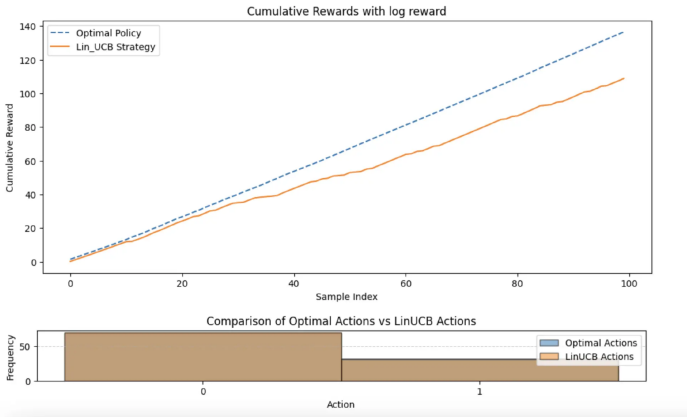
The sigmoid function stands out with excellent performance, achieving over 71% in both accuracy and F1 score. This suggests its effectiveness in capturing nonlinear relationships and handling binary classification tasks.




```
LinUCB Accuracy: 0.7100
LinUCB F1 Score: 0.7060
```

- Log

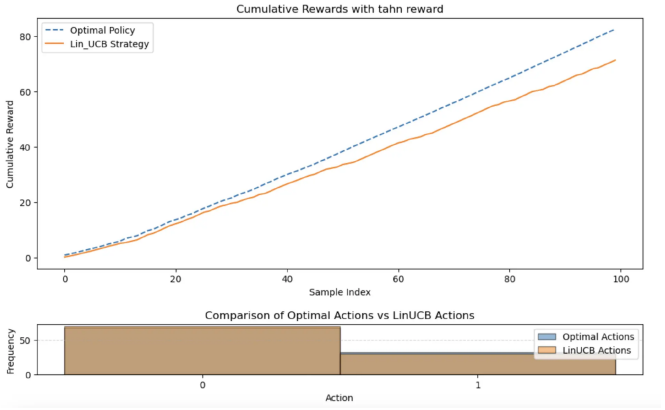
The logarithmic function also performs well. This indicates its ability to smooth rewards effectively when dealing with features that have a wide range of values.




```
LinUCB Accuracy: 0.7300
LinUCB F1 Score: 0.7288
```

- Tahn

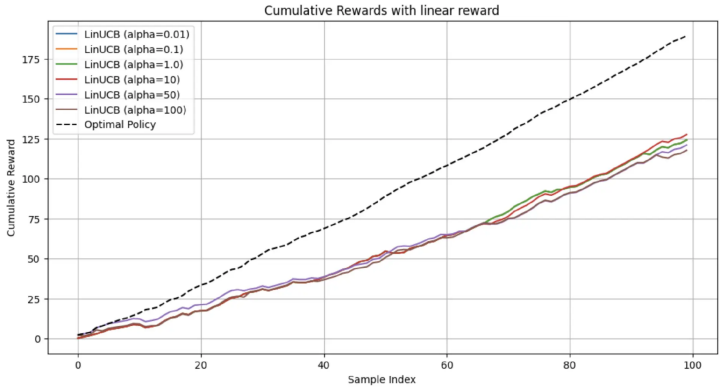
The tanh function delivers the best results, surpassing 76% in both accuracy and F1 score. This indicates its superior adaptability in handling contextual features and its ability to capture complex patterns and relationships.




```
LinUCB Accuracy: 0.7600
LinUCB F1 Score: 0.7579
```

Different alpha

- As alpha increases, the LinUCB cumulative reward curves get closer to the Optimal Policy (dashed line), indicating better exploration.
- **Smaller alpha values** (0.01, 0.1, and 1.0) **produce stable results**, with an accuracy of 0.6100 and an F1 score of 0.6167.**Larger alpha value**s (50 and 100) show a drop in accuracy and F1 score, **reflecting the negative impact of over-exploration**.




```
Alpha: 0.01
LinUCB Accuracy: 0.6100
LinUCB F1 Score: 0.6167

Alpha: 0.1
LinUCB Accuracy: 0.6100
LinUCB F1 Score: 0.6167

Alpha: 1.0
LinUCB Accuracy: 0.6100
LinUCB F1 Score: 0.6167

Alpha: 10
LinUCB Accuracy: 0.6300
LinUCB F1 Score: 0.6363

Alpha: 50
LinUCB Accuracy: 0.5500
LinUCB F1 Score: 0.5648

Alpha: 100
LinUCB Accuracy: 0.5200
LinUCB F1 Score: 0.5351
```

2. OPE

2.1 IPS

IPS is particularly useful in scenarios like contextual bandits or recommendation systems where you want to evaluate a new decision-making policy (which might have different actions) using data collected from a previous policy.

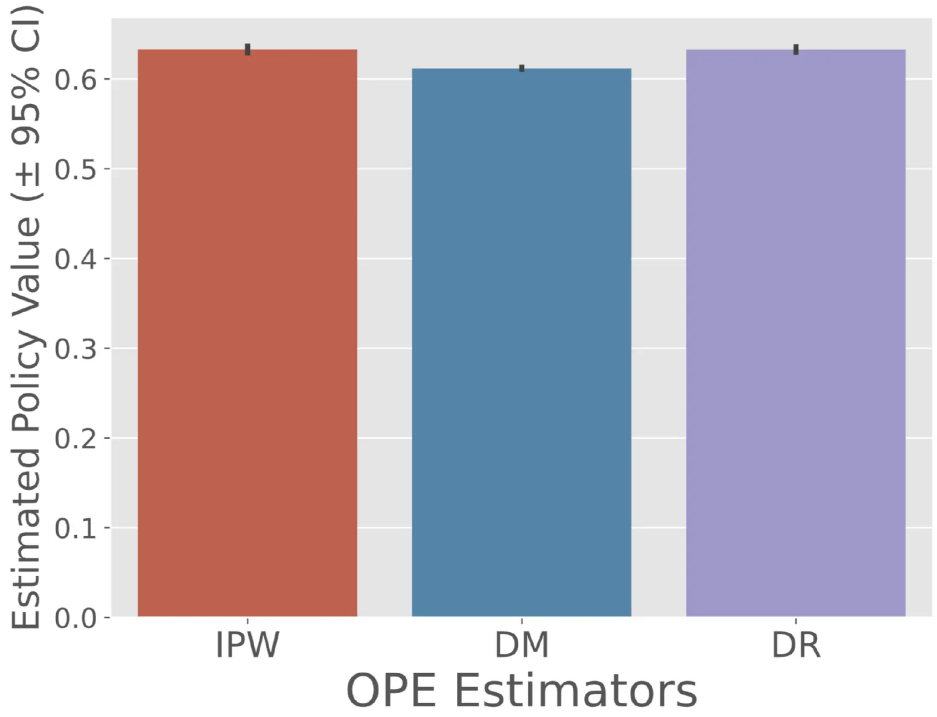
2.2 OBP: Open Bandit Pipeline

OBP provides various **OPE estimators**, such as:

- **Inverse Propensity Scoring (IPS)**: Estimates the performance of a new policy by reweighting the rewards based on the likelihood (propensity) of actions taken by the historical policy.
- **Self-Normalized IPS (SNIPS)**: A variant of IPS that normalizes the scores to reduce variance.
- **Doubly Robust (DR)**: Combines IPS with model-based estimates to improve robustness.
- **More advanced estimators**, such as **Switch-DR** and **Weighting Doubly Robust (WDR)**, are also available for more accurate or stable evaluation.

sample code:

```
ope = OffPolicyEvaluation(
    bandit_feedback=bandit_feedback_test,
    ope_estimators=[IPW(), DM(), DR()]
)
ope.visualize_off_policy_estimates(
    action_dist=action_dist,
    estimated_rewards_by_reg_model=estimated_rewards_by_reg_model,
)
```



- I failed when trying to align the Statlog data and LinUCB model with this package because the package requires `action_dist` to represent the probability distribution over the predicted actions, but the LinUCB model returns the probabilities for each arm. To fully align with this package, the model's code needs to be adjusted. I will align the package before the next meeting.

3. Some questions

a. During our coding process, we look into the generate_data_and_split function, where we find this line of code:

historical_actions = generate_actions(split_index)

And since the generate_actions function is sampling from a normal distribution thus random, we actually don't use the y of the optimal training set when we initialize historical_action. In our common sense as we do machine learning before, we need to utilize historical both x and y to train, while in the code we don't see the usage of y in the training phase, why is it this case?