Contextual Bandit News Recommender System

Angad U20220011

Overview

This project implements a **contextual bandit–based news recommendation** pipeline combined with **user classification**. Specifically:

1. We **classify** users into one of three categories: **User1**, **User2**, and **User3**.
2. We run **three** contextual bandit algorithms:
   - **Epsilon-Greedy**
   - **UCB**
   - **SoftMax**
3. We recommend the **best news category** (one of `Entertainment`, `Education`, `Tech`, or `Crime`) based on the **estimated expected reward** for that user's context.
4. We show **hyperparameter experiments** to analyze how different ϵ\epsilonϵ, CCC, and TTT values affect performance.

Data and Preprocessing

- **User Datasets**:
  - `train_users.csv`
  - `test_users.csv`

    Each has columns:
    `['user_id','age','income','clicks','purchase_amount','label']`
- **News Articles**:
  - `news_articles.csv`

    Columns:
    `['link','headline','category','short_description','authors','date']`

User Labels

- Original user labels: `User1`, `User2`, `User3`.
- We convert them to numeric {0,1,2}to serve as **contexts** for the bandit.

News Categories

- The dataset had many categories (Politics, Wellness, Entertainment, etc.).
- We **filter** down to four: **Entertainment (0), Education (1), Tech (2), Crime (3)**.

Implementation

Classification Model

We train a **DecisionTreeClassifier** (`max_depth=5`, `random_state=42`) to predict `User1`, `User2`, `User3` from features: `[age, income, clicks, purchase_amount]`.

Classification Accuracy on `test_users.csv`:

```
[Classification] Decision Tree Accuracy: 33.05%
```

The **classification report**:

```
              precision    recall  f1-score   support

       User1       0.32      0.26      0.29       672
       User2       0.34      0.51      0.41       679
       User3       0.33      0.21      0.26       649

    accuracy                           0.33      2000
   macro avg       0.33      0.33      0.32      2000
weighted avg       0.33      0.33      0.32      2000
```

Observation: The model does relatively better on `User2` than on `User1` or `User3`. This suggests we may need more features or more data to distinguish user classes effectively. Overall accuracy ~33%, near random for 3-class classification.

Contextual Bandits

We define **3 contexts** (0=User1,1=User2,2=User3) and **4 arms** (0=Entertainment,1=Education,2=Tech,3=Crime). We have a **sampler** that yields rewards for each (context, arm)

Algorithms

1. **Epsilon-Greedy**
   - We pick a random arm with probability ϵ\epsilonϵ, else pick the max Q-value arm.
   - Trained for 10,000 steps per context.
2. **UCB**
   - We pick the arm with the highest UCB index
   - Also trained for 10,000 steps per context.
3. **SoftMax**
   - We pick arms according to a softmax distribution based on Q-values
   - Trained for 10,000 steps per context.

Results & Discussion

Single Hyperparameter Setting

We first tested:

- **Epsilon-Greedy** (ϵ=0.1)
- **UCB** (C=1)
- **SoftMax** (T=1.0)

Final Average Rewards:

Algorithm	Context=0 (User1)	Context=1 (User2)	Context=2 (User3)	Overall Avg
Epsilon-Greedy (ε=0.1)	7.10	4.43	5.51	5.68
UCB (C=1)	7.98	4.99	5.99	6.32
SoftMax (T=1.0)	7.99	4.83	5.83	6.22

Observations:

- **UCB** and **SoftMax** yield slightly higher overall average rewards (~6.3 vs. 6.2).
- **Context=0 (User1)** consistently gets the highest average reward among the three contexts. **User2** gets the lowest.

Hyperparameter Comparisons

We further tested:

- **Epsilon-Greedy** with $\epsilon \in \{0.1,0.5,0.7\}$
- **UCB** with $C \in \{1,2,3\}$
- **SoftMax** with $T \in \{0.5,1.0,2.0\}$

Below are some **sample final average rewards** (Overall context average):

Epsilon-Greedy	ϵ=0.1	ϵ=0.5	ϵ=0.7
Overall Reward	5.697	3.184	1.934

UCB	C=1	C=2	C=3
Overall	6.333	6.328	6.327

SoftMax	T=0.5	T=1.0	T=2.0
Overall	6.189	6.231	5.729

Observations:

- **Epsilon-Greedy**: As ϵ **increases** from 0.1 → 0.5 → 0.7, the overall reward **decreases**. High random exploration drastically lowers performance.
- **UCB**: The overall reward hovers ~6.33 for $C \in \{1,2,3\}$ Little difference among these.
- **SoftMax**: T=1.0 yields the best result (~6.23). A **very high** temperature (2.0) causes more exploration, leading to a drop in reward (~5.73).

Learned Q-Values

At the end of training, we also printed the **final Q-values** for each context and arm (the bandit's estimate of expected reward). For instance, with Epsilon-Greedy ϵ=0.1:

```
[Final Q-Values (Expected Rewards) for Epsilon-Greedy, Hyperparam=0.1]
Context=0:
   Arm=0 (Entertainment) ~ Q=0.995
   Arm=1 (Education)     ~ Q=8.000
   ...
Context=1:
   Arm=0 (Entertainment) ~ Q=-7.040
   Arm=1 (Education)     ~ Q=4.996
   ...
Context=2:
   ...
```

We see that, for **Context=0**, `Education` is assigned a Q-value near 8.0, suggesting the bandit believes that category yields the highest expected reward for **User1**.

Example Recommendation

We tested a **new user** with features `[age=29, income=29862, clicks=91, purchase_amount=270.91]`.

The system recommended:

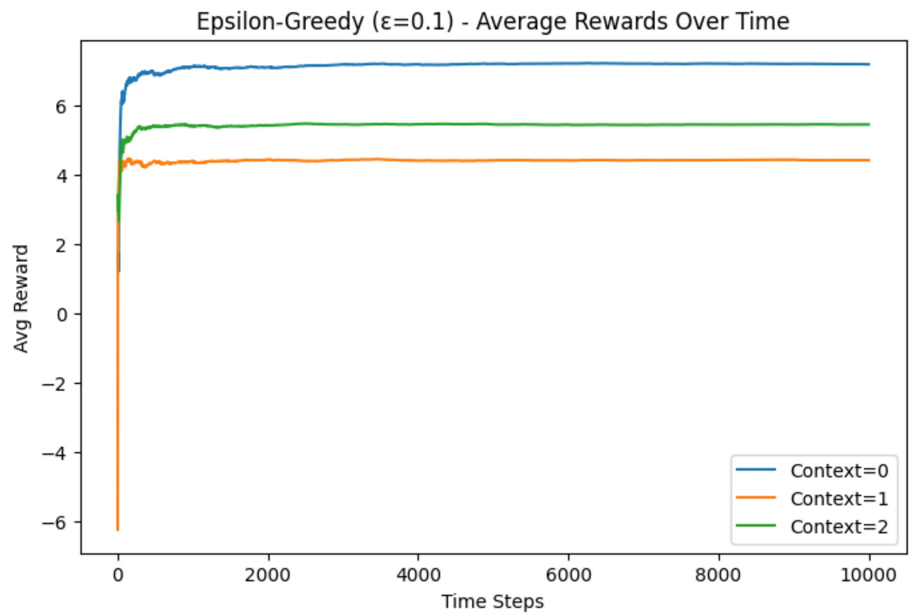
```
[Recommendations for new user]:
  - egreedy: Category=Education, Headline=Talk, Read and Sing to Kids to
Close the Word Gap
  - ucb: Category=Education,    Headline=The End of Reading in America and
Other Related Matters
  - softmax: Category=Education, Headline=Education and Philanthropy
```

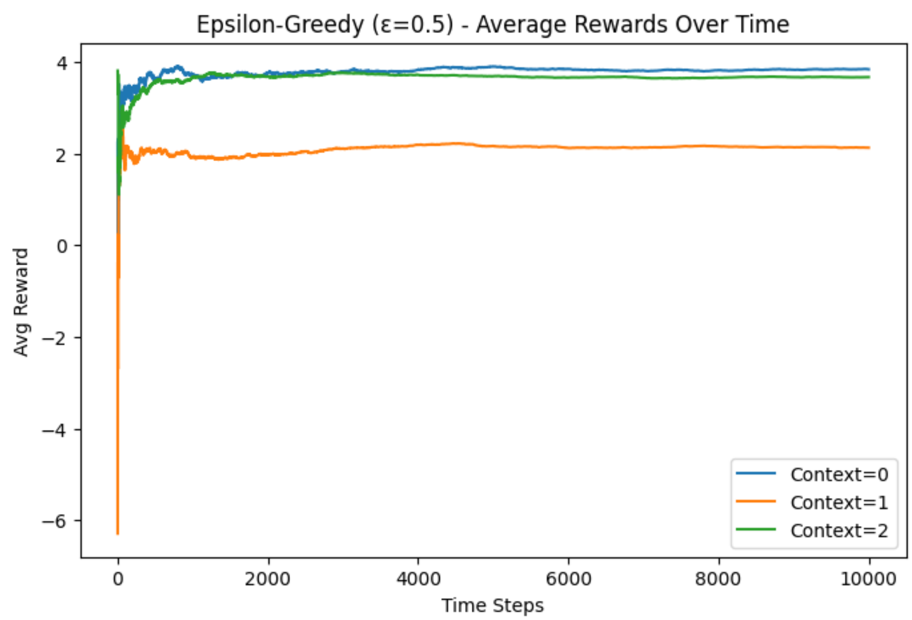
So all three policies (with their final Q-values) suggest **Education** for this user.

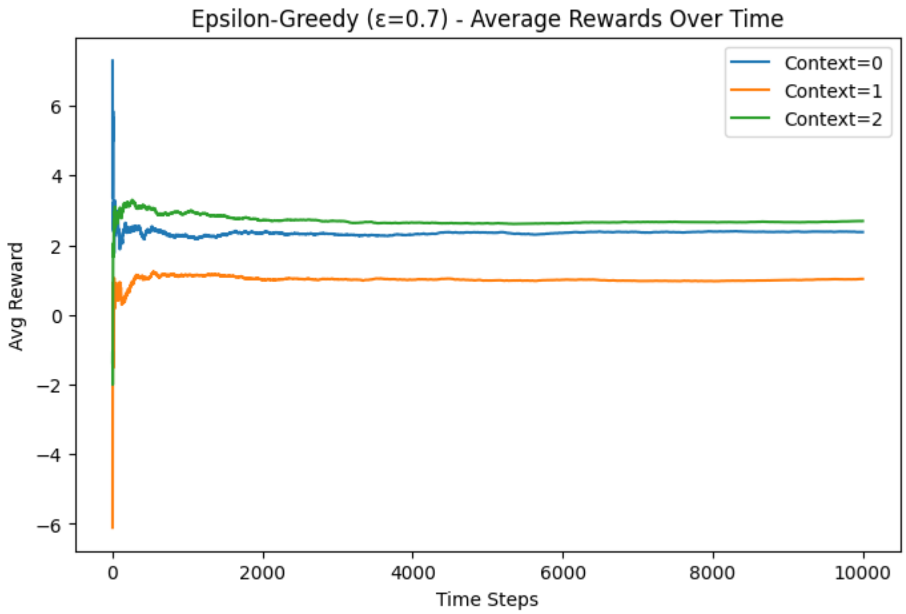
Conclusions

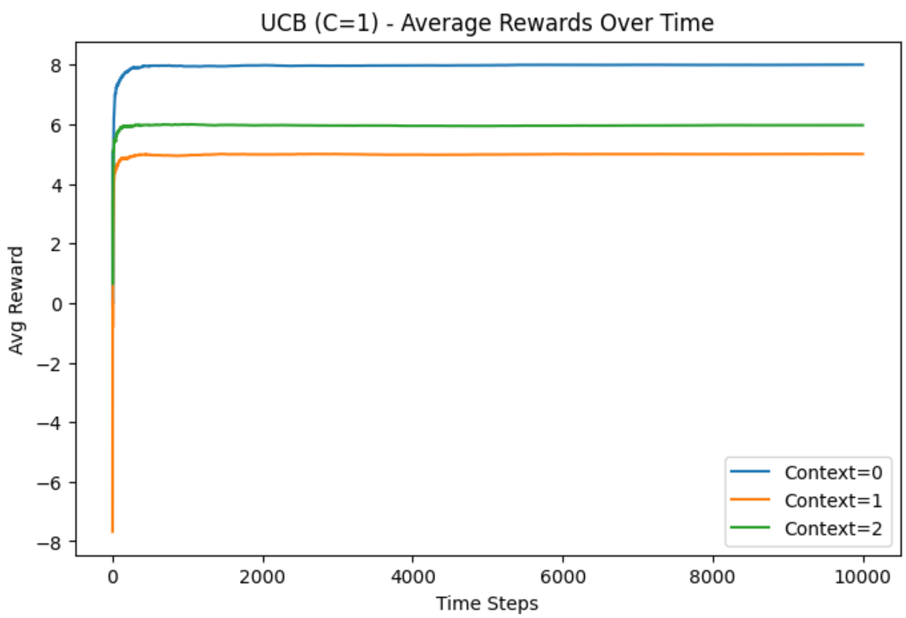
- With ~33% accuracy across 3 labels, the user classification is almost random. There is certain misclassification taking place across all categories
- **UCB** and **SoftMax** typically yield the highest overall average rewards (~6.2–6.3).
- **Epsilon-Greedy** is best at ϵ=0.1 (overall ~5.7) but significantly worse at higher ϵ. Higher exploration rates were giving better results initially but went down in the long term.
- This matches expectations: too much random exploration leads to suboptimal performance.
- **Context=0 (User1)** yields the highest rewards, indicating the reward sampler's distribution is more favorable for that user type.
- **User2** often ends up with the lowest average reward.
- We see that certain ϵ, C, and T values matter. For instance, ϵ=0.1is better than 0.5 or 0.7 for Epsilon-Greedy; T=1.0 is best overall for SoftMax.

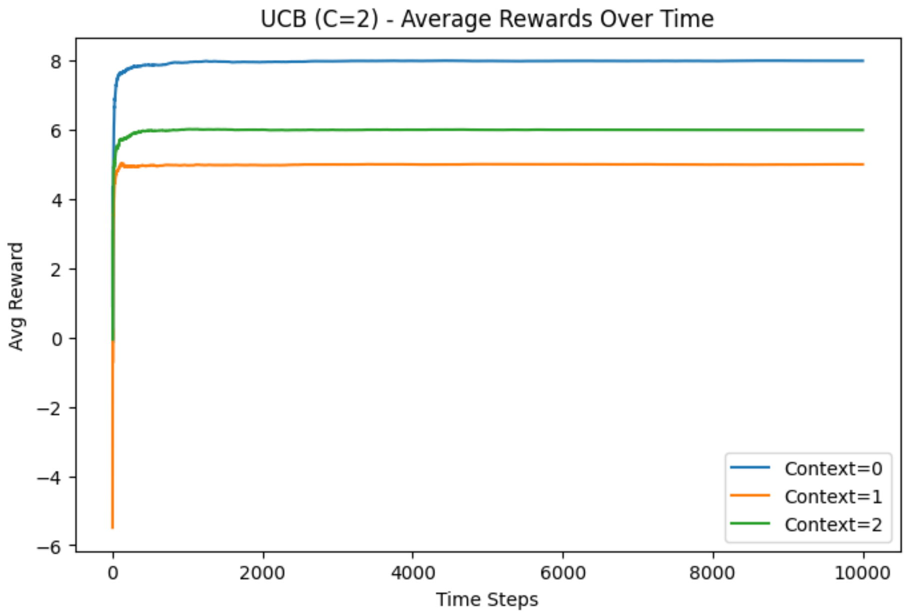
PLOTS-

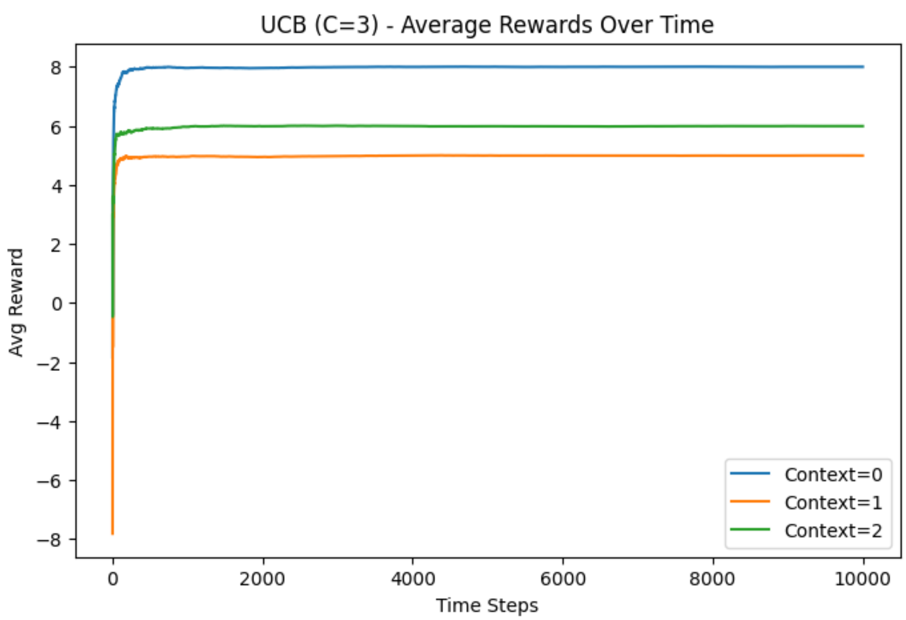

Epsilon-Greedy (ε=0.1) - Average Rewards Over Time
Avg Reward
Time Steps
Context=0
Context=1
Context=2

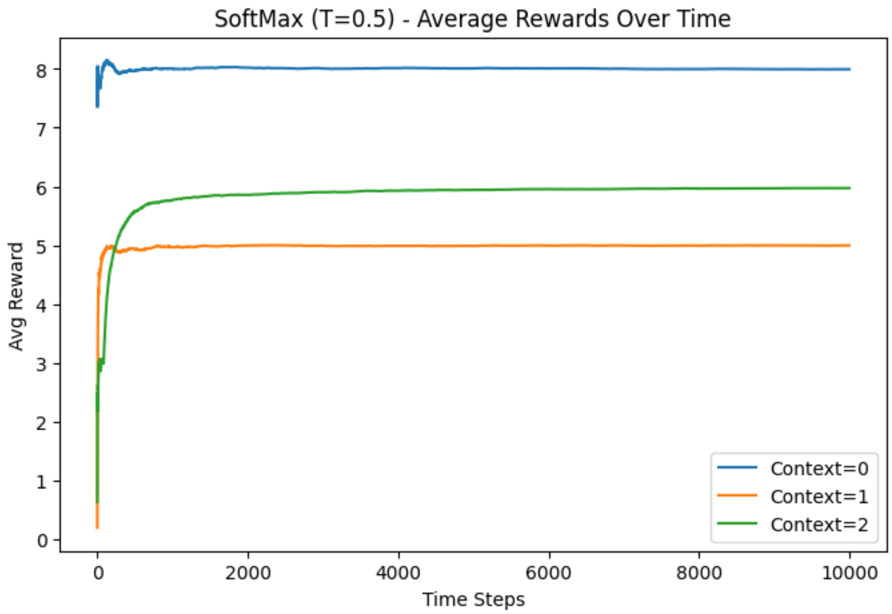

Epsilon-Greedy (ε=0.5) - Average Rewards Over Time
Avg Reward
Time Steps
Context=0
Context=1
Context=2

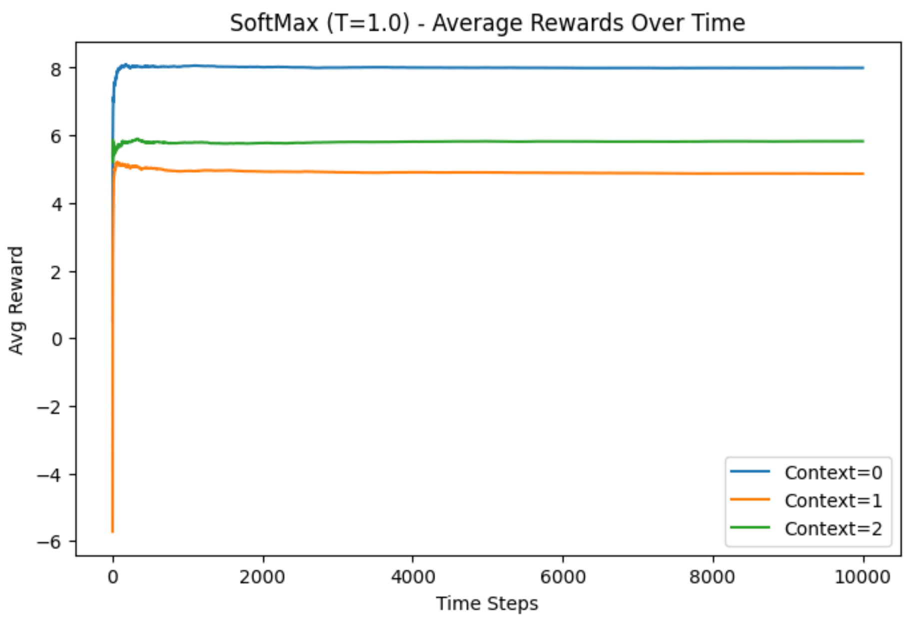

Epsilon-Greedy (ε=0.7) - Average Rewards Over Time
Context=0
Context=1
Context=2
Avg Reward
Time Steps
6
4
2
0
−2
−4
−6
0
2000
4000
6000
8000
10000

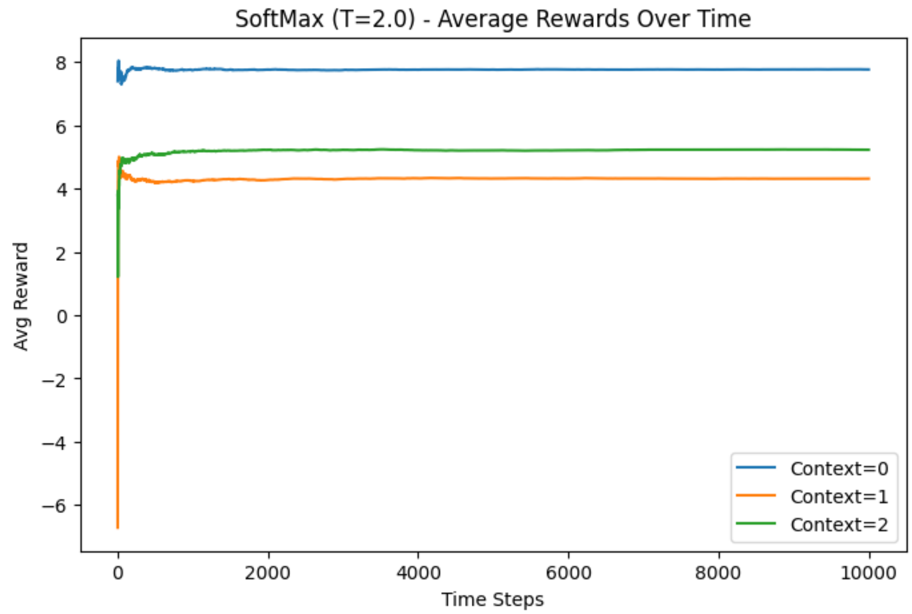

UCB (C=1) - Average Rewards Over Time
Context=0
Context=1
Context=2
Avg Reward
Time Steps
8
6
4
2
0
−2
−4
−6
−8
0
2000
4000
6000
8000
10000


UCB (C=2) - Average Rewards Over Time
Avg Reward
Time Steps
Context=0
Context=1
Context=2
8
6
4
2
0
−2
−4
−6
0
2000
4000
6000
8000
10000


UCB (C=3) - Average Rewards Over Time
Avg Reward
Time Steps
Context=0
Context=1
Context=2
8
6
4
2
0
−2
−4
−6
−8
0
2000
4000
6000
8000
10000


SoftMax (T=0.5) - Average Rewards Over Time
Avg Reward
Time Steps
0
1
2
3
4
5
6
7
8
0
2000
4000
6000
8000
10000
Context=0
Context=1
Context=2


SoftMax (T=1.0) - Average Rewards Over Time
Avg Reward
Time Steps
−6
−4
−2
0
2
4
6
8
0
2000
4000
6000
8000
10000
Context=0
Context=1
Context=2



Instructions to Run & Replicate

1. **Clone** or **Download** the code from this repository.
2. **Install** required packages, including the `sampler` wheel:

```
pip install -r requirements.txt
# If needed:
pip install /path/to/sampler-1.0-py3-none-any.whl
```

3. **Place Data**
   - Make sure `train_users.csv`, `test_users.csv`, and `news_articles.csv` are in the correct folder
4. **Run** - open the notebook (`Bandit_Assignment.ipynb`) and **run all cells**.