BT4014 Analytics Driven Design of Adaptive Systems
2020/2021 Semester 2
Group Project
Question 2

BUILDING A BEER RECOMMENDER

with the use of Bandit Algorithms

Prepared by: Group 7

Chan Cheah Cha	A0189006A
Chua Kai Bing	A0185606Y
Goh Jia Yi	A0185610J
Lim Jia Qi	A0189626M
Tan Zen Wei	A0188424X

Table of Contents

1. Introduction

1.1 Background

In 2017, the global beer market was valued at $0.593 billion, and this number is expected to rise to $0.685 billion by 2025 (Sinha, 2018). This increase in consumption of beer is largely due to growing disposable income and changes in consumer preferences towards beer over other alcoholic beverages. Moreover, the surge in female drinkers and the unprecedented rise in youth population further fuels the growth of the beer market. Coupled with the growing market, the number of brewing companies is also expanding. According to The Carling Partnership (2017), there are currently more than 19,000 brewing companies in the world. Besides, there are 300,000 beers that are listed on BeerAdvocate in 2017, a go-to beer resource for millions of consumers each month and the benchmark for beer reviews (Alström, 2017).

1.2 Motivation

According to a study conducted by Nielson, a brand name can be a company's most valuable asset. Brand names can lend credibility to product efficacy and provide an assurance of quality (Nyström, 2017). However, this means that consumers tend to stick with well-established brands and refuse to try out new or smaller brands that are highly rated but with low popularity. However, amidst the current COVID-19 pandemic, a shift in consumer's shopping behaviour has been observed. Around 40% of consumers have expanded their sampling of new brands, with the level of brand switching doubling in 2020 compared to the previous year. This change in shopping behaviour is mostly driven by the search for quality brands that can match consumer's values, which is especially prevalent in teenagers, who cover the majority of the beer consumer market as aforementioned. While growth in consumer goods was driven by larger companies before, this shift in shopping behaviour has started helping smaller companies to grow increasingly in late 2020 and early 2021 (Charm, 2021).

1.3 Solution

As a licensed intermediary specialised in selling beers, we operate an eCommerce business to host our business online. We wish to make use of this opportunity by utilising a robust recommender system in our platform that can recommend less commonly found beers for consumers. This can ease consumer's efforts in the exploration of new brands to try among the plethora of beer brands that are available in the market, and aid smaller beer brewing companies that are often overshadowed by big brands to stand out. At the same time, it can help our business to minimise inventory loss and increase consumer engagement, which can help to build stronger brand loyalty and entice more consumers to make more purchases with us. All these will lead to an increase in our revenue, benefitting the intermediary and differentiating ourselves from our competitors.

2. Data Exploration and Preprocessing

To aid in our testing, a Beer Reviews dataset[1] from BeerAdvocate was retrieved from data.world. It contains 1,586,614 reviews of beers collected over the span of more than 10 years, up to November 2011. It shows 33,387 users and their ratings for 66,055 beers from 5,840 different breweries. The users are able to rate as many beers as they want in the dataset, and the ratings are on a scale from 1 (worst) to 5 (best). If no rating has been recorded, it will be represented by 0. Each review includes their ratings for the beer in terms of 5 aspects — appearance, aroma, palate, taste, and an overall review.

This section will walk through the entire process of exploring and preprocessing the dataset used to build our recommender system.

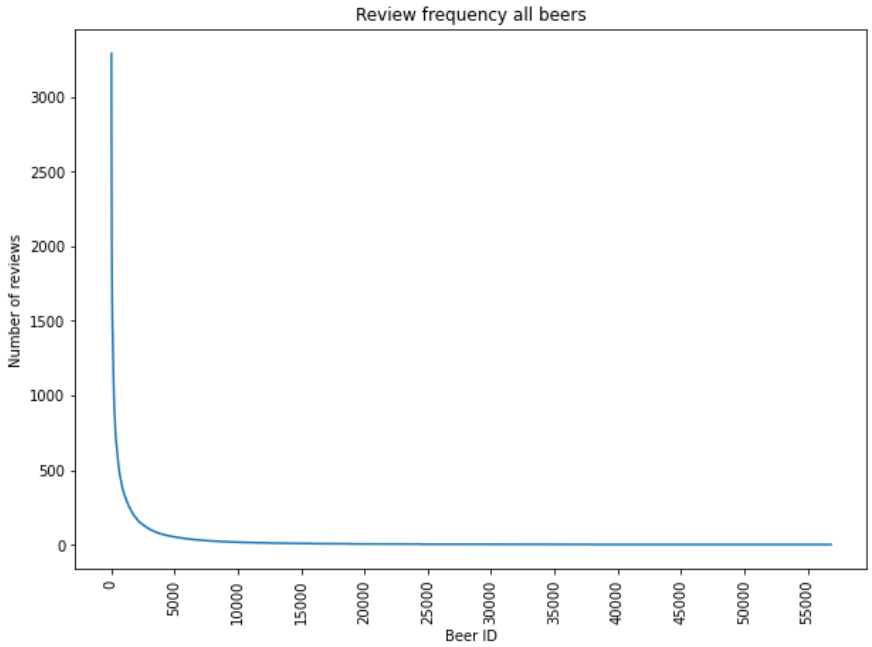

Figure 1: Review Frequency of all beers

Figure 1 above shows the frequency of reviews for each individual beer. It can be observed that only 3,104 out of 66,055 beers (approximately 5%) have received more than 100 ratings from unique users; while the rest has little or no user interactions. These sparse reviews makes a beer less generalizable to all users and becomes highly sensitive to specific individuals that have rated these obscure beers. To remove such undesirable and noisy patterns present, the dataset used will be narrowed down. In addition, this helps to reduce the computing power required for large datasets.

[1] https://data.world/socialmediadata/beeradvocate

2.1 Shortlisting Less Popular but High Quality Beers

Out of the 66,055 unique beers in the dataset, we wish to identify beers that are less popular yet are of high quality. Popularity is defined by the review count of the beer; the more the number of reviews for a particular beer, the more popular it is. Quality is defined by the average rating of the beer among all its reviews; the higher the average overall rating, the higher the quality of the beer. Out of the 5 aspects of the ratings that are available in the dataset as mentioned above, we will be using the overall review for this metric.

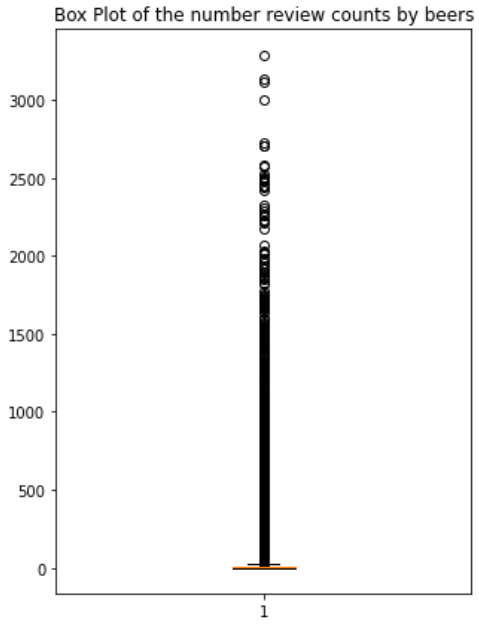

Figure 2: Box Plot of beer review counts

	count
count	56857.000000
mean	27.905341
std	122.198636
min	1.000000
25%	1.000000
50%	3.000000
75%	9.000000
max	3290.000000

Figure 3: Summary Statistics of beer review counts

From the summary statistics in Figure 3 above, it can be observed that the beer review counts ranges between 1 to 3,290, with its interquartile range at 1 to 9 reviews. While the median review count is 3, the mean is significantly higher at 28 review counts. The distribution is highly skewed to the right as we can see from the boxplot of the beer review counts in Figure 2 above. This means that out of the 66,055 beers, most of them are not as popular with very little or close to zero reviews. On the other hand, the popular beers with relatively higher review counts are considered outliers. To further analyze the disparity among the outliers, we look at the reviews counts of the top 50 beers.

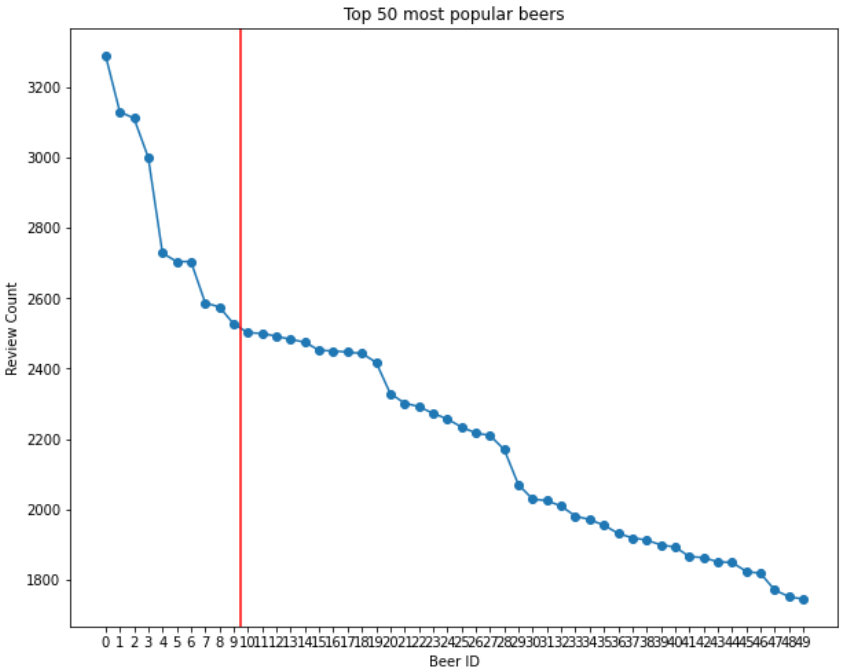

Figure 4: Review Count of the top 50 beers by popularity

By looking at the review count of the top 50 most popular beers in Figure 4 above, we can see a relatively steep gradient for the top 10 beers. This means that the top beers contribute greatly to the disparity in the distribution of the beer popularity.

Since our goal is to promote and recommend underrated beers that our consumers are unfamiliar with, we will be filtering out these top 10 most popular beers from our dataset. This is made under the assumption that the review count is an accurate representation of the actual popularity of the beers amongst the general population.

When narrowing down the data, we also have to ensure that sufficient reviews are present for the beers shortlisted to reduce data sparsity issues. Thus, only beers with more than 100 review counts were chosen. 3,094 beers were eventually shortlisted, with review count ranging from 101 to 2,502.

Now that we have shortlisted the beers that we want to recommend in our popularity range, we will next look at the average overall rating scores of the beers to further filter out lower quality beers from the list.

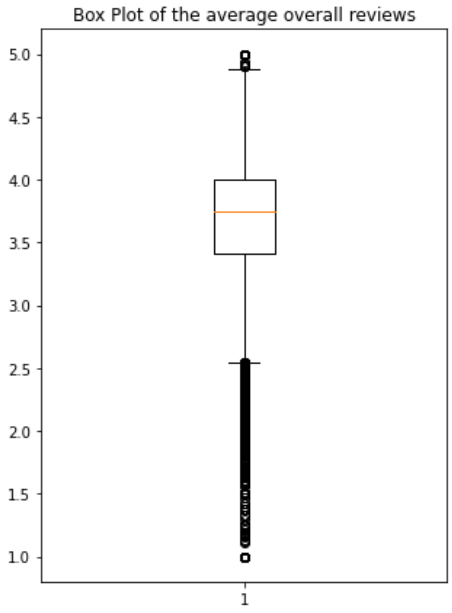

Figure 5: Box Plot of beer average overall ratings

	review_mean
count	56857.000000
mean	3.654631
std	0.617673
min	1.000000
25%	3.416667
50%	3.750000
75%	4.000000
max	5.000000

Figure 6: Summary Statistics of average overall ratings

From the summary statistics in Figure 6 above, we can see that the average overall ratings range from 1.0 to 5.0, having a mean average rating of 3.65 and a median average rating being 3.75. The distribution is skewed to the left as we can see in Figure 5, where most of the average ratings lie between the interquartile range of 3.5 to 4.0.

In the interests of having a denser dataset, we further shortlisted the 3,094 beers to the 100 beers based on the average overall rating.The lowest average overall rating was 4.29, which is above the 75th percentile of 4.0. Hence, we can safely conclude that our final 100 shortlisted beers are of high quality, while still staying within our targeted popularity range. This final 100 beers will allow us to build a test recommendation system, before actually deploying in production.

We conducted further data exploratory on the final 100 beers selected to be used for our algorithms.

2.2 Exploring Shortlisted Less Popular but Higher Quality Beers

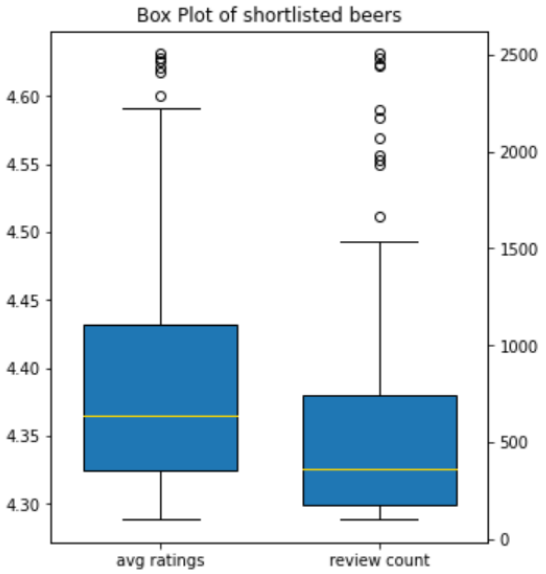

Figure 7: Histogram plot of average ratings and review count of final shortlisted 100 beers

	review_mean	count
count	100.000000	100.0000
mean	4.390753	622.6900
std	0.091847	650.5884
min	4.288674	103.0000
25%	4.323950	175.2500
50%	4.364329	362.5000
75%	4.432115	743.5000
max	4.630952	2502.0000

Figure 8: Summary statistics of average ratings and review count of final shortlisted 100 beers

The distributions of the final 100 shortlisted beers are less skewed compared to the original dataset, with lesser outliers as seen in Figure 7 above. The average ratings ranges from 4.29 to 4.63, with its interquartile range lying between 4.32 to 4.43. It has a mean of 4.39 and a median of 4.36 rating. Meanwhile, the review count ranges from 103 to 2,502, with its interquartile range lying between 175 to 744. It has a mean of 623 and a median of 363 reviews.

From this final list of shortlisted beers, we have a total of 62,257 reviews with 10,437 users that have reviewed at least 1 of the final 100 beers. We preprocessed our data into a utility matrix for the bandit algorithms, with the final 100 beers as the column and the users as the row.

	Citra DIPA	Cantillon Blåbær Lambik	Heady Topper	Deviation - Bottleworks 9th Anniversary	Trappist Westvleteren 12	Pliny The Younger	Founders CBS Imperial Stout	Live Oak HefeWeizen	Portsmouth Kate The Great	Rare Bourbon County Stout	Duck Duck Gooze	Reality Czeck	Weihenstephaner Hefeweissbier	Trappist Westvleteren 8	Zombie Dust
0	0.0	0.0	0.0	0.0	4.5	5.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	4.5	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.5	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 9: Utility Matrix of Final 100 Beers

Each row represents whether the user has rated the beer and if so, the respective rating score. For example, 0 indicates that the user has not rated the beer while a score from 1 to 5 indicates the score that the user has given for the beer.

This final preprocessed dataset will be used to test different algorithms that can be used to build a recommender system. For this project, we will be comparing multi-armed bandit algorithms such as Epsilon-Decay, Annealing Softmax, UCB1, Bayesian UCB and Thompson Sampling, against the use of naive algorithms such as Collaborating Filtering. A more detailed description of the algorithms can be found in the subsequent sections of this report.

3. Collaborative Filtering

The collaborative filtering method is the one of the most commonly used recommender systems. It makes predictions on user preference by utilizing historical data of users' preferences on a set of items. This is built on the assumption that people who have liked similar items, will tend to like the same things. User preference is usually expressed by implicit rating or explicit rating. Explicit rating is a rating score given by a user on a sliding scale while implicit rating indicates user preferences indirectly, for example, whether a user views a page (Luo, 2019). For our project, we have used explicit ratings on a scale of 1 to 5.

There are 2 classes of collaborative filtering: user-based and item-based.

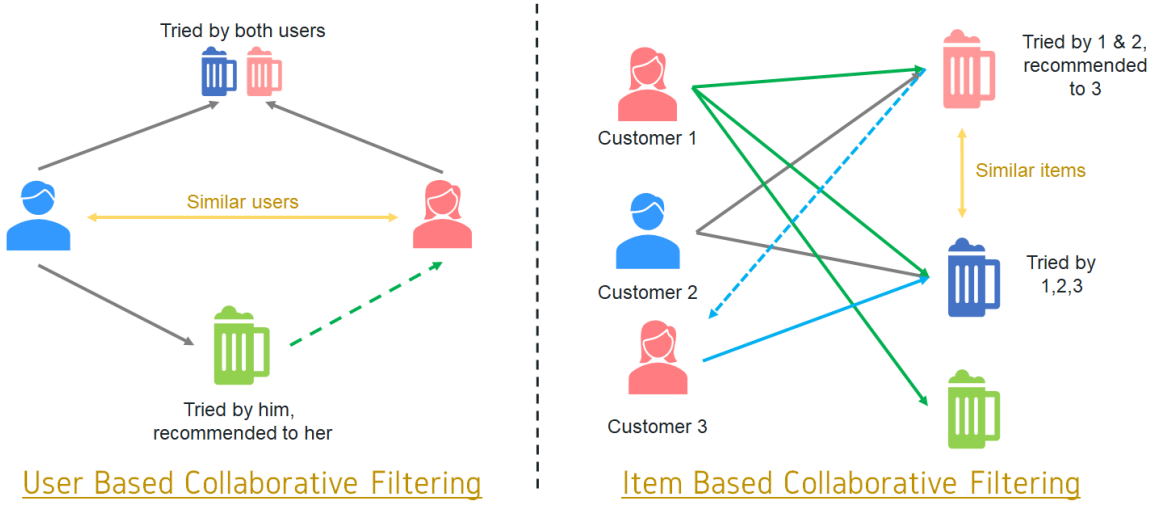

Figure 10: Collaborative Filtering Diagram

3.1 User Based Collaborative Filtering

User Based Collaborative Filtering predicts the items that a user might like on the basis of ratings given to that item by the other users who have similar taste with that of the target user. It calculates a similarity function between each pair of users and predicts the rating that a user would give to an item that is new to the user based on how other users have rated it.

3.2 Item Based Collaborative Filtering

Item Based Collaborative Filtering finds similar items based on items users have already liked. 2 items are deemed as similar when they receive similar ratings from the same user. It calculates the similarity between the ratings of each pair of items using a similarity metric and predicts the rating the user would give to a new item based on how the user has rated items similar to it (Qutbuddin, 2020).

3.3 Model Used

We aim to recommend less commonly found beers to users, hence **user based collaborative filtering** was chosen for our project. It is more suitable in recommending beers that are novel to the users, giving them an opportunity to discover new interest in other types of beers. This is in contrast to item based filtering, which is a more conservative approach as it recommends beers that are similar to the user's existing taste and preferences (Katipoglu, 2020).

The model used was the K-Nearest Neighbours (KNN) with Means. It calculates the similarities between the target user and other users and selects the top K similar users. Then, it will take the weighted average ratings from these K users, using their similarities as the weights. To avoid bias when computing the weighted average (since different people have different baselines when they give their ratings), we can subtract each user's average rating of all items and add it back for the target user (Sharma, 2021). The items will then be ranked according to the predicted scores and the item with the highest predicted rating will be recommended to the user.

Several methods can be used to compute similarity. Using the utility matrix in Figure 10, each row and column is described by a feature vector, and with the usage of similarity metrics, we can next determine the similarity of these vectors. Some of the common similarity metrics used are: Pearson correlation, Mean Squared Difference (MSD) and Cosine similarity. The Pearson correlation calculates the correlation between 2 vectors while MSD takes the mean of the squared difference of the 2 vectors. On the other hand, the Cosine similarity score measures the Cosine angle between the vectors (Pocs, 2020).

Grid Search was done to try out different parameters to choose the best for the current algorithm. The selected similarity metric is MSD, with k=50, representing the number of neighbours to consider for aggregation.

3.4 Performance Analysis

The 62,257 reviews were randomly split into train set and test set, with a test size of 30%. Since the split was done according to reviews, it meant that the same user could be assigned to both the test and train set. This was necessary for the team to evaluate the prediction of beer ratings made by the collaborative filtering method against the actual ratings made by the user.

The beer selected for each user is the beer with the highest predicted rating by the recommender. The table below shows the error metrics of the recommender.

Error Metric	Score
Mean Absolute Error (MAE)	0.3992
Root Mean Squared Error (RMSE)	0.2890
Mean Squared Error (MSE)	0.5375

Table 1: Performance metrics of Collaborative Filtering

The error metrics measure the recommender's accuracy of predicting the ratings. The MAE score of 0.3992 indicates that on the scale of 1 to 5, the estimated ratings are about 0.3992 above or below the actual ratings on average. The RMSE and MSE score of 0.2890 and 0.5375 respectively also implies that the difference between the predicted and actual rating is not large. This means that the recommender is able to predict the ratings rather accurately, with only slight differences.

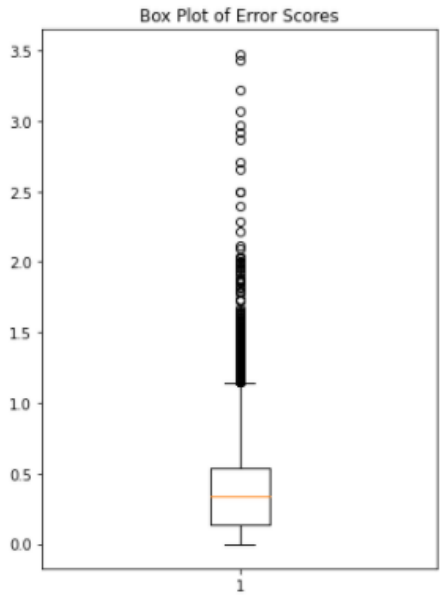

Figure 11: Box Plot Prediction Error Scores

```
count    4673.000000
mean        0.399239
std         0.359976
min         0.000000
25%         0.132861
50%         0.334657
75%         0.535547
max         3.476582
Name: error, dtype: float64
```

Figure 12: Summary Statistics of Prediction Error Score

The error score ranges from 0 to 3.477, with a mean of 0.399. Based on the box plot in Figure 11 above, the distribution of error scores is skewed to the right with a few outliers that have high error scores. This supports that the recommender provides a rather accurate recommendation and prediction rating. However, this only applies to users that have data of their past preferences exposed to the recommender. For the "new users", which the recommender does not have any information about, it was unable to make an accurate recommendation. This is known as the cold start problem (More details on this in Section 3.5).

Out of 6,121 users in the test set, 1,448 of them are "new" to the system. This poses a challenge to the collaborative filtering method as the lack of information prevents it from making an accurate prediction of their ratings. It resulted in the recommender providing a prediction

score of 4.379839 for users that it does not have historical data on, regardless of beer choice as shown in the 'predict' column in Figure 13 below.

	user	beer	actual	predict	User Data in Train	error
7	duke1258	Founders Breakfast Stout	5.0	4.379839	0	0.620161
21	sleestak4life	Saison - Brett	4.5	4.379839	0	0.120161
39	Sleestak	Weihenstephaner Hefeweissbier Dunkel	5.0	4.379839	0	0.620161
42	littleg	Tröegs Nugget Nectar	5.0	4.379839	0	0.620161
46	ztprez	Duvel	5.0	4.379839	0	0.620161
...	...	...	...	...	...	...
18666	FlyFisher2782	Temptation	4.5	4.379839	0	0.120161
18669	JayRey	Tröegs Nugget Nectar	5.0	4.379839	0	0.620161
18671	ChuggyMcBeer	Daisy Cutter Pale Ale	5.0	4.379839	0	0.620161
18675	beejayud	Bell's Hopslam Ale	4.5	4.379839	0	0.120161
18677	SpillyBeers	Sculpin India Pale Ale	4.5	4.379839	0	0.120161

Figure 13: Prediction of "New" Users

As we can see from Figure 13 above, most of the predicted ratings from the 'predict' column are significantly different from the actual ratings in the 'actual' column. As such, the recommender was only able to make an accurate recommendation for the users that the system had historical data on.

3.5 Limitations

3.5.1 Cold Start Problem

The cold start problem is a prominent problem in collaborative filtering. It happens when the system cannot draw any inferences or make predictions for users or items that the model has insufficient information on. This was evident in our results where the collaborative filtering method was unable to make an accurate prediction on the new users that did not have historical data in the training dataset.

The cold start problem can lead to nonoptimal recommendations being made to users, which is undesirable for our business. Being unable to cater to new users will hinder our business progression as getting an unsatisfactory recommendation will deter them from making purchases with us in the future. Therefore, it is important for the recommender system to be able to make a suitable recommendation that could cater to both existing and new users.

3.5.2 Dynamic Catalog and User Preferences

User preferences and item popularity are constantly changing in the highly dynamic business environment. Thus, the business would need to consistently obtain updated data of its users and items. Moreover, it is important that enough data points are captured as collaborative filtering requires a large dataset with active users and ratings to make accurate predictions.

However, collaborative filtering does not allow the embedding of new items not present in the feature vectors of the training dataset. This means that as reviews of new beers are added to the collection, they will not be recommended to users. The model will only be recommending beers from the original training dataset, which will become outdated over time. As the building of a collaborative filtering recommender system requires a dedicated training and testing phase, this issue can only be rectified if the model is retrained with the updated data. However, this is time consuming and counterproductive for the business. The model will need enough data and time steps before it can make an accurate recommendation, which can cause the business opportunity costs. Hence, collaborative filtering would not be a suitable recommender for our eCommerce platform as it is unable to dynamically update its data to accommodate new users and items.

3.5.3 ‘Similar tastes’ Ambiguity

Popular items tend to appeal to people with different tastes. This may affect the clustering of similar users, causing the recommendations given to the users to not align with their actual preferences since collaborative filtering makes the assumption that users who rate items similarly have similar tastes. The underlying tastes expressed by the latent features of an item are not interpretable in the absence of item properties (Luo, 2018). For example, a user can give a high rating to Carlsberg due to its aroma or taste. However, that user may not actually enjoy pilsners, and instead prefer ales. Hence, user-based collaborative filtering lacks the transparency and explainability for this level of information.

As we wish to recommend beers that are less popular, the collaborative filtering method may not be the most suited algorithm for our recommender system due to the limitations that come with it. Therefore, the need arises for a better recommendation system that is able to recommend beers to users even without prior information about the particular user and is able to dynamically update to cater to new users and beers.

4. Multi-Armed Bandit

The multi-armed bandit problem is used when there is a fixed limited set of resources that must be efficiently allocated between choices, in a manner where the expected gains are maximised eventually. In these situations, the properties of each choice are uncertain, hence, in an aim to ensure we get the maximum rewards, we can introduce the multi-armed bandit solution.

The multi-armed bandit solution uses machine learning algorithms to dynamically allocate resources to variations that are performing well, with the goal of determining the best or most profitable outcome through a series of trials. At each trial, the algorithm would have to face a tradeoff between exploitation or exploration - if exploration is selected, the algorithm would gain more information about the properties of the aforementioned choices, on the other hand, if exploitation is chosen, the algorithm will allocate resources to the choice with the current highest expected payoff. Thus, learning how to balance between the two is essential in this reinforcement learning process.

Bandit algorithms are commonly used for eCommerce websites to solve problems such as building a recommender system; they are able to overcome the limitations faced by collaborative filtering in Section 3.5 above.

As bandit algorithms do not require specific knowledge of users to make recommendations, beers will be recommended based on ratings at that point in time. Overtime, it will learn to choose beers that are well-liked by general users. This resolves the issue of the cold start problem and similar tastes ambiguity.

On the other hand, the bandit algorithm can also respond quickly to member feedback as it does not have a clear distinction between the training and testing phase. Instead, the bandit algorithm balances exploration and exploitation, and will be more flexible to pick up new insights, solving the dynamic catalog and user preference issue.

In general, bandit algorithms align more closely with our purpose of recommending good quality and underrated beers to new and existing users. Hence, it should be adopted in building our recommender system. This section will further explore and evaluate different bandit algorithm variants for use.

4.1 Approach

To execute a bandit algorithm, we will have to first define the arm, where each arm will represent a beer type. The arm has a Gaussian distribution and is initialised with the mean and standard deviation values derived from the preprocessed dataset. The rewards of each arm are drawn from the fixed Gaussian distribution which are independent across rounds and across actions. The rewards returned by the arms will fall between the range of 0 to 5 and adjustments were made for values that fall out of the aforementioned range.

This experiment serves as a pilot use case before the actual implementation. Strictly following the time steps of the historical data will not give us an accurate representation of the dynamic

real-world data. Hence, to curb this issue, we will define a random factor in our arm to better simulate actual real-world scenarios from the derived distribution from the historical data.

We have chosen to test the following bandit algorithms – Epsilon-Decay, Annealing Softmax, UCB1, Bayesian UCB and Thompson Sampling. 200 rounds of simulation were performed for 100 arms across 10,437 time steps. Each time step represents a user's feedback on different beers for the business.

4.2 Epsilon-Decay

One widely used algorithm for multi-bandit problem is the Epsilon-Greedy algorithm, where one balances the exploration and exploitation of an experiment through the definition of an epsilon (ε) value. For example, an ε value of 0.4 refers to a 40% probability of the algorithm exploring. Unfortunately, one main disadvantage of a Epsilon-Greedy algorithm is that random noise will continuously be introduced at the fixed epsilon rate, even after the best arm is identified. In order to mitigate this, a decay function can be introduced in the usage of an Epsilon-Decay algorithm to decrease the amount of noise introduced over time.

In an Epsilon-Decay algorithm, the epsilon value will decay over time, eventually diverting its resources to settle on an optimal solution and exploit it, rather than to constantly explore the sub-optimal options. We have defined a decay formula to adjust the rate of decay with reference from previous works by Natarajan (2020). The 3 hyperparameters A, B and C can be adjusted to control the following characteristics of the decay factor:

A. Extent of exploration and exploration at the start of time series. A larger value of A will result in more exploring, by lengthening the left tail of the decay graph.

B. Decides the slope of transition region between Exploration to Exploitation zone. The lower the value of B, the steeper the transition.

C. Controls the steepness of the left and right tail of the graph. The higher the value of C, the steeper the tails of the graph.

By defining a decay formula, we will be able to test if the decay formula which allows for more exploration or one which prioritises exploitation will perform better in our context.

Exploration vs. Exploitation

With the decaying function we have defined, we are able to control the rate of decay and the extent of exploration carried out in initial timesteps. Two simulations will be conducted with different hyperparameter A values of 0.1 and 0.3 to test different levels of exploration and exploitation. Conversely, the hyperparameters B and C will be fixed at 0.1 to emulate an exponential distribution. The performance between these two simulations will then be compared and the better performing hyperparameters will then be selected to represent Epsilon-Decay.

The following simulations were conducted:

1. **Exploitation**: the rate of decay is set to be faster, hence prioritising exploitation
    - A = 0.1; B = 0.1; C = 0.1
2. **Exploration**: there will be a slower rate of decay hence allowing for more exploration
    - A = 0.3; B = 0.1; C = 0.1

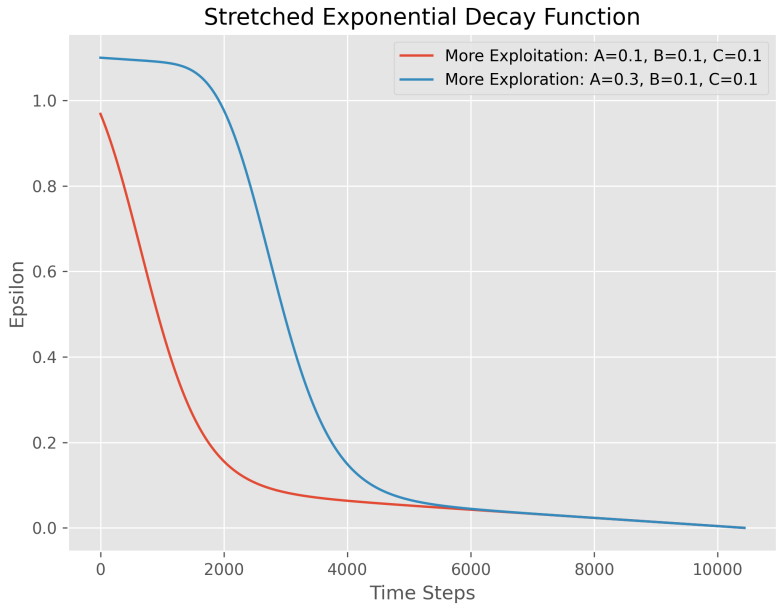

Figure 14: Rate of decay function

Simulation 1: Exploitation	**Simulation 2: Exploration**
A = 0.1	A = 0.3
Smaller value of A results in less time dedicated to exploring since epsilon value decays faster. Simulation 2 has a heavier emphasis on exploitation.	Larger value of A results in more exploration, as seen from the lengthening left tail of the decay graph. The left tail has epsilon values above 1, forcing the algorithm to explore more in earlier timesteps.

Table 2: Comparison of Decay Functions

Based on the plots above, the epsilon value drops below 0.2 at around t = 1800 for *Simulation 1 (Exploitation)* and before t = 4000 for *Simulation 2 (Exploration)*. *Simulation 1 (Exploitation)* exploits to a larger extent compared to *Simulation 2 (Exploration).* For both plots, the right tails have epsilon values close to zero, indicating more exploitation at later time points. This is consistent with our understanding of the Epsilon-Decay algorithm, which focuses on exploiting better options over time to reap higher rewards.

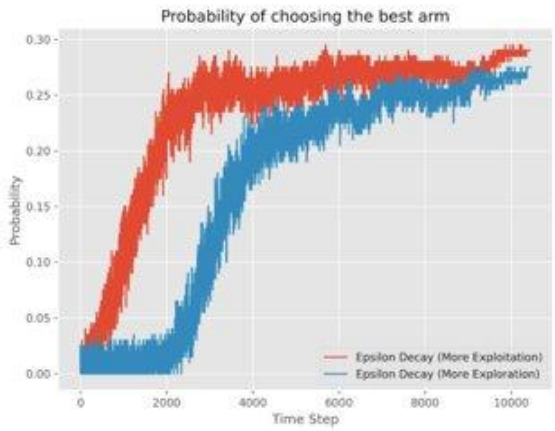

Figure 15: Epsilon-Decay - Probability of Choosing the Best Arm

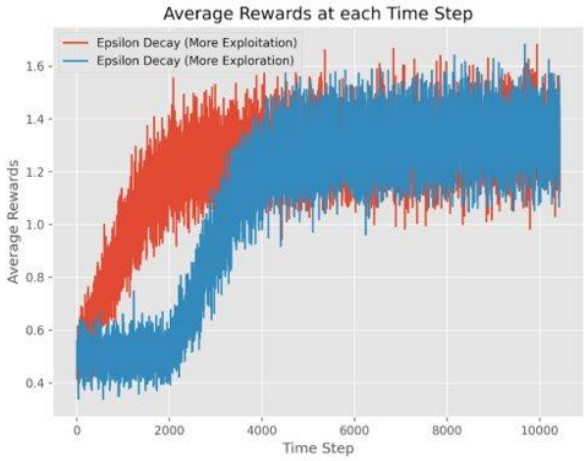

Figure 16: Epsilon-Decay - Average Rewards at each Time Step

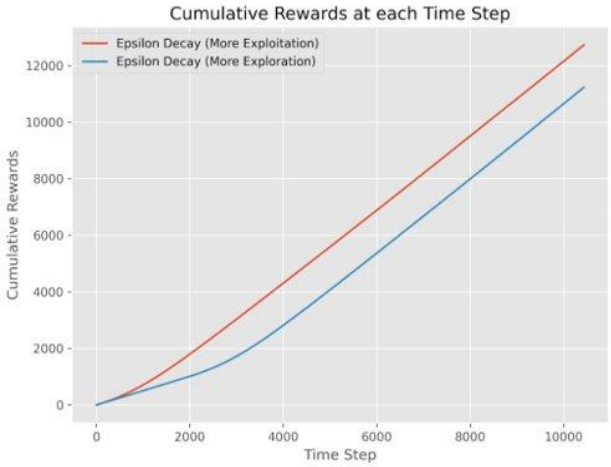

Figure 17: Epsilon-Decay - Cumulative Rewards at each Time Step

As seen in Figure 15 above, Simulation 1 *(Exploitation)* converges faster and reaches a higher probability of choosing the best arm at 0.29. Simulation 2 *(Exploration)* converges at a lower level of around 0.27. Hence, Simulation 1 *(Exploitation)* seems to perform better in terms of pulling the best arm in terms of rewards gained.

Based on the average rewards plot in Figure 16 above, *Simulation 1 (Exploitation)* again converges at a faster rate than *Simulation 2 (Exploration).* However, both simulations peak at around the same level of 1.40. This means that over time, the average rewards gained at each time step approaches 1.40.

As seen in the cumulative results plot in Figure 17 above, *Simulation 1 (Exploitation)* accumulates more reward over time since it spends less time exploring sub-optimal options, leading to higher rewards over time.

In general, *Simulation 2 (Exploration)* is expected to accumulate more knowledge about the reward distributions of the arms due to heavier exploration. Despite this, *Simulation 1 (Exploitation)* still showed a greater probability of choosing the best arm. This means that *Simulation 2 (Exploitation)* is more efficient and has achieved a greater balance of exploration and exploitation. The short-term performance of both simulations are comparable based on the average rewards obtained at each time step. The long term performance of *Simulation 1 (Exploitation)* was better. Hence, we can conclude that *Simulation 2 (Exploitation)* has better overall performance and will be used to compare against other bandit algorithms in later stages. In our context, we should spend more time exploiting the better options at later time points, rather than increasing our opportunity costs by exploring sub-optimal options.

4.2 Annealing Softmax

The Softmax algorithm uses the Boltzmann distribution to assign a probability to each arm, and the algorithm will randomly select an arm to exploit based on these probabilities. Unlike Epsilon algorithms which explore arms randomly without considering reward levels, the Softmax algorithm offers a more structured exploration (Foo, 2020) by assigning the probability of each arm in proportion to their estimated values. While the random exploration in the Epsilon-Decay algorithm allows us to try out options that we do not know much about, its randomness may contribute to inefficient exploration if a known bad action is selected.

The temperature parameter *(tau)* determines how random the selection is. (E, 2016). A tau value equivalent to 0 would mean that the algorithm will always select the arm with the highest expected value.

$$P(Explore\ arm\ i) = \frac{e^{r_i/\tau}}{\sum e^{r_k/\tau}}$$

Figure 18: Probability of exploring an arm

Similar to Epsilon-Decay, the Annealing Softmax algorithm introduces an annealing factor which aims to decrease the randomness of selection towards the end of the experiment in order to exploit the optimal arm more. This will minimise resources wasted and maximise rewards gained.

Annealing softmax is robust for arms with higher variance in means, meaning that it will perform better when applied on a set of arms that have larger differences in mean. In the context of our dataset, since the average ratings for the set of beers we will be testing on is relatively large, this algorithm is expected to perform well.

4.3 Upper Confidence Bound (UCB)

For Epsilon-Decay and Annealing Softmax algorithms, estimations of reward levels could potentially be coincidence since the algorithm designs encompasses a random factor. To tackle this potential pitfall, we can use the Upper Confidence Bound (UCB) algorithm.

The UCB algorithm constructs a confidence interval for the estimated rewards for each arm, while factoring in the uncertainty caused by data variance and the limited knowledge gained on each arm (LeDoux, 2020). With lesser knowledge on each arm, the estimated reward is less accurate which results in a huge confidence interval; this confidence interval shrinks with more information gained (Glowacka, 2019).

The algorithm is optimistic in the face of uncertainty and will select arms with the highest upper confidence bound (Kun, 2013). Favouring arms with lower confidence value estimation also implies that the algorithm is encouraged to explore all the arms that have yet to be explored.

There are multiple different variations of the UCB algorithm and we will be utilising both the UCB1 and Bayesian UCB algorithms.

4.3.1 UCB1

$$\mathbb{P}\left(|\overline{X}_n - p| > \epsilon\right) \leq 2e^{-2n\epsilon^2}$$

Figure 19: Hoeffding's Inequality

UCB1 algorithm uses Hoeffding's Inequality to assign an upper bound to an arm's mean reward, where there's high probability that the true mean will be below the UCB assigned by the algorithm.

$$U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}} \text{ and } a_t^{UCB1} = \arg\max_{a \in \mathcal{A}} Q(a) + \sqrt{\frac{2\log t}{N_t(a)}}$$

Figure 20: UCB1 Algorithm

When the probability (p) of the true mean being greater than the UCB is set to be less than or equal to $p=t^{-4}$, we get the UCB1 algorithm. This probability (p) converges to zero as the number of rounds (t) increases, allowing us to make a more confident bound estimation with more rewards observed.

At each step, we will play the action which maximises the expression in Figure 20 above. $N_t(a)$ represents the number of times the action a was played so far, while t represents the number of time steps.

4.3.2 Bayesian UCB

In the UCB1 algorithm, no assumptions were made on the reward distribution and very generalised estimations were made while relying on Hoeffding's Inequality (Weng, 2018). On the other hand, the Bayesian UCB algorithm takes the same principles of UCB1 but incorporates the knowledge of the arms rewards distribution, allowing better bound estimation to be made with more efficient exploration (LeDoux, 2020).

For each Gaussian arm generated, we assume that the mean μ is known and thus, the Inverse Gamma distribution is selected as the conjugate prior (Wikipedia, 2021).

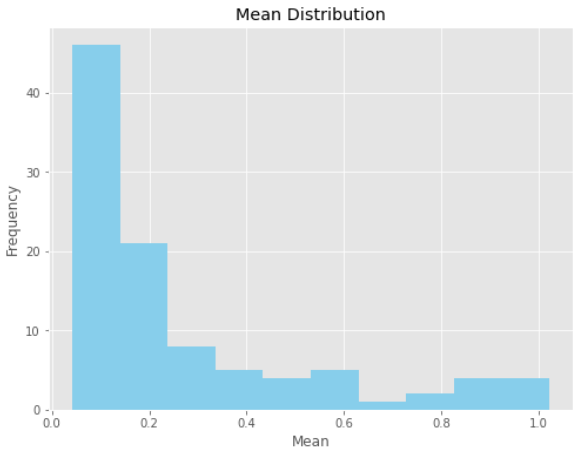

Figure 21: Distribution of the Mean of all arms

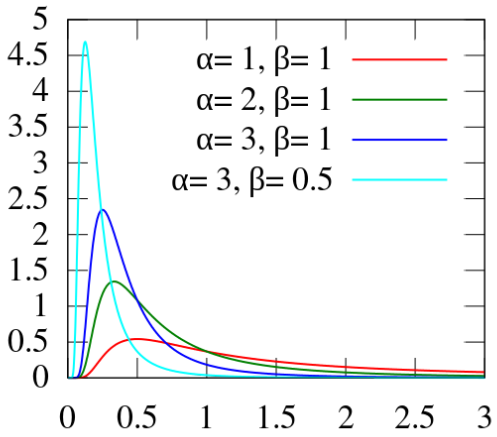

Figure 22: Inverse Gamma Probability Density Function

To select the posterior hyperparameters, we compared the distribution of the means of all the arms and the varying Inverse Gamma distributions. From the above, it can be observed that both graphs are skewed to the right. More specifically, the probability density function where α = 3, β = 0.5 is the most similar to the distribution of the means. Thus, based on this prior knowledge, we have set α = 3, β = 0.5 and σ = 3. The posterior hyperparameters were then updated with $\alpha + \frac{n}{2}, \beta + \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{2}$.

4.4 Thompson Sampling

The Thompson Sampling algorithm utilises probability matching (Wikipedia, 2021) as its decision strategy through the building of a probability model from the obtained rewards, and then sampling from this distribution to choose an action (Roberts, S., 2021). Similar to that of the Bayesian UCB algorithm, we also assume that the mean μ is known, and the Inverse Gamma distribution is used as its conjugate prior.

The main difference between the Thompson Sampling algorithm and the UCB algorithms is that the former is fully Bayesian; generating a vector of expected results from the selected, Inverse Gamma distribution, and choosing the arm based on the one with the highest expected reward. That is unlike the UCB algorithm which selects an arm based on the estimated expected rewards under optimistic assumptions (Hubbs, C., 2020)

In our Thompson Sampling model, we have set our prior hyperparameters to α= 3, β= 0.5, using the same selection method as the Bayesian UCB method. The posterior hyperparameters were also updated similarly with $\alpha + \frac{n}{2}$, $\beta + \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{2}$.

4.5 Performance Analysis

In this section, we will be comparing the performances across the bandit algorithms mentioned above — Epsilon-Decay, Annealing Softmax, UCB1, Bayesian UCB, and Thompson Sampling algorithms.

4.5.1 Probability of Choosing the Best Arm

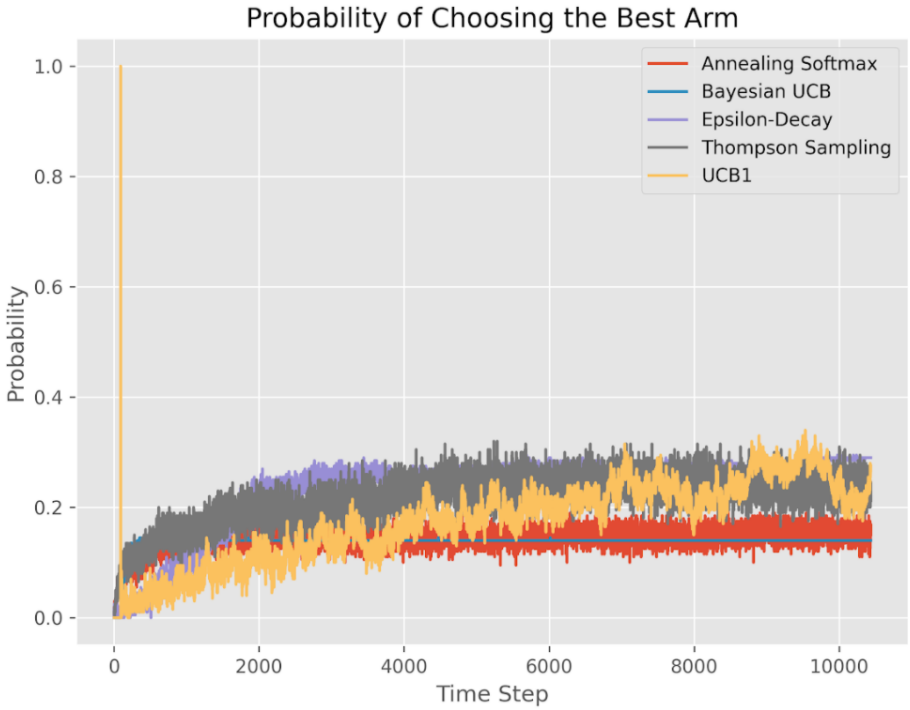

Figure 23: Probability of Choosing the Best Arm

Figure 23 above depicts the probability of choosing the best arm. In general, the best performing algorithms are that of the Epsilon-Decay, Thompson Sampling and UCB1 algorithms which all peaked at roughly 0.3. However, we can distinguish that the aforementioned 3, the rate of convergence was the highest for the Epsilon-Decay algorithm, followed by the Thompson Sampling algorithm and the UCB1 algorithm. At the same time, an initial spike for the UCB1 algorithm can be seen in the first few time steps. This can be attributed to the need of the algorithm to explore all the arms in the beginning.

The worst performing algorithms in this aspect are the Annealing Softmax and Bayesian UCB algorithms. Both of them had roughly the same convergence rate, and they both stabilised at approximately 0.16.

4.5.2 Average Rewards at each Time Step

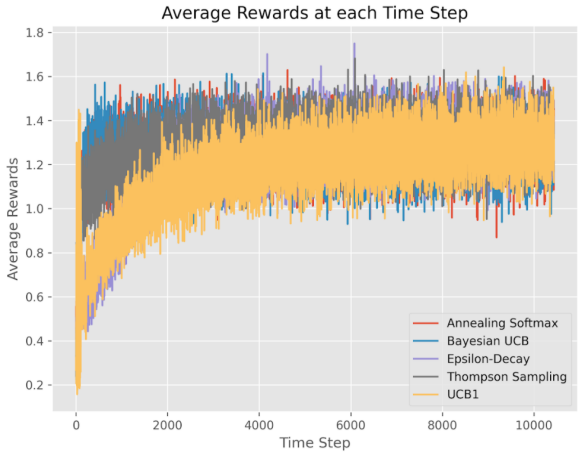

Figure 24: Average Rewards at each Time Step

Figure 25: Average Rewards at each Time Step (Smooth)

Figures 24 and 25 above both show the average rewards at each time step, with Figure 25 being more clear after a smoothing function was applied. From Figure 25, it can be observed that all 5 algorithms peaked at roughly the same average rewards value, hovering close around the value of 1.3. From the 5 algorithms, the Annealing Softmax and Bayesian UCB algorithms converged the quickest, followed closely by the Thompson Sampling algorithm. Next, the UCB1 and Epsilon Decay algorithms converged at about the same rate but are the slowest in comparison to the rest.

4.5.3 Cumulative Rewards at each Time Step

Figure 26: Cumulative Rewards at each Time Step

Figure 26 above illustrates the cumulative rewards at each time step. It can be seen that the best performing algorithm is the Thompson Sampling algorithm, followed extremely closely by the Annealing Softmax and Bayesian UCB algorithms. The cumulative rewards of the 3 algorithms reached approximately 13,000. Next, the cumulative rewards of the Epsilon-Decay algorithm achieved roughly lower than 12,500. The worst performing algorithm is the UCB1 algorithm, and the cumulative rewards achieved is slightly above 12,000.

The above findings are rather surprising as we have expected more sophisticated algorithms such as UCB1, Bayesian UCB and Thompson Sampling to consistently outperform the simpler algorithms (Epsilon-Decay and Annealing Softmax). However, it can be observed that the UCB1 algorithm has the poorest relative performance when compared to the others; this can be attributed to the UCB1 algorithm being overly optimistic, which can lead to selection of arms with lower actual rewards.

4.5.4 Overall Performance

In order to prevent making myopic decisions which are extremely risky for our business, we shifted our focus to analyzing the cumulative rewards graph since it is more telling of the true rewards analysis of our experiments in the long run.

Overall, we can conclude that Thompson Sampling was the best performing algorithm, seeing that it has achieved the highest cumulative rewards at the end of the experiment and that it has a relatively high probability of selecting the best arm. This is followed closely behind by the Bayesian UCB and Annealing Softmax algorithms.

5. Algorithm Selection

In recent years, there has been an increase in consumption for low and non-alcoholic beers with more health-conscious consumers. According to the IWSR, the consumption for such beers have increased by 33% in 2020 and is expected to grow by 31% by 2024 (Eads, 2021). We are expecting the beer ratings to fluctuate slowly over time, with beers gaining and losing popularity. According to Frosmo (2020), the optimal algorithm for such trend expectations will be UCB1 or Softmax.

From the performance analysis conducted in Section 4.5.4 above, the best performing algorithms were Thompson Sampling, Annealing Softmax and Bayesian UCB. Although the Thompson Sampling and Bayesian UCB algorithms have one of the best performances, the use case of the recommender system has to be taken into consideration. As both algorithms assume a certain distribution of the rewards, one of its cons is that it can take a long time for the algorithm to respond to changes in reward levels and select the new best-performing option.

On the other hand, although the UCB1 algorithm is recommended for our use case, it has a relatively poor performance when simulations were run over 100 arms. The algorithm explores all arms in the initial stages, which has led to slow convergence in average rewards observed in Figure 25. In the real world where there are more than 66,000 arms, a longer time period will be required for our system to calibrate before making good recommendations. In this case, bad recommendations from the system in early stages may cause a direct negative impact on our users which is undesirable.

It is essential to optimise our system based on the market expectations so that it has the ability to quickly adapt to new trends and maximise profits. Based on the algorithm tests we have conducted with historical data, Annealing Softmax has performed relatively well in this aspect. In addition, it is suited well for our use case based on the expected reward trends, as suggested by Frosmo. We can most likely observe good performance after the implementation of such recommenders and thus, we will be using the **Annealing Softmax algorithm** for our recommender system.

6. Future Improvements

Future work for this project includes using a more updated dataset, different metrics and testing unexplored algorithms, which may potentially yield better results for our business context.

6.1 Obtaining an Updated Robust Dataset

The dataset used to build the test recommender system has a few issues. First, the beer reviews were outdated as they were collected from 2001 to 2011, which was 10 years ago as mentioned in Section 2 above. This means that our test recommender system is unable to account for the latest beer trends and thus, may affect us in selecting the best algorithm for deployment in today's context. Second, the data has sparsity issues which can cause biases when used to train a recommender system. Having a better dataset can help to enhance accuracy and robustness of the recommender system built and facilitate better selection process.

6.2 Explore Different Metrics and Algorithms

Besides using beer ratings as the main metric to build a beer recommender system, we can also leverage on metrics such as conversion rate or purchase volume. This helps to add depth and dimension for the systems built. Moreover, we can leverage data such as user and item data to build a more robust and accurate recommender system with other untested algorithms.

Basic bandit algorithms that we have tried may be rather simple which is not suitable for the real world. For example, the UCB1 algorithm adopts an optimistic view on uncertainty. However, it can be arguably naïve as reward structures in the real world are extremely complex. Adversarial bandits are bandit algorithms which assumes that payoffs could conceivably operate in any manner despite being fixed in the beginning (Kun, 2013). Some examples will include Exp3 and its application will allow more accurate simulation of the real-world problem to obtain more representative results.

On the other hand, bandit algorithms are also unable to take into account the contextual knowledge of users to recommend the best option. To tackle this problem, we can make use of contextual bandits, an extension of the multi-armed bandit which makes actions conditional on the state of the environment. They take into account available context information to help guide their choice of action (Chumley, 2018). Under this algorithm, users can be grouped together by similar features via classification or clustering techniques, assuming that users who belong to the same cluster have similar behaviours. With that, we would be able to make more user-specific recommendations that have a higher chance of being more accurate.

7. Conclusion

In the era of big data analytics, it is essential to leverage on such valuable information to help a business discover insights, improve decision-making processes and increase efficiency. We have tested out both Collaborative Filtering and Bandit Algorithms to build a beer recommender system, and after taking the business context as well as the algorithms' performances into consideration, we have selected the **Annealing Softmax algorithm** as the final algorithm in our recommender system. This recommender system will then help our business to accurately recommend less commonly found, but high quality beers for our users. In doing so, our intermediary can successfully ride the wave of the growing beer industry and changing consumer behaviours. Eventually, with the usage of big data, we can successfully build our business and differentiate ourselves from our competitors.

References

Alström, T. (2017, July 11). How many different beers are there? Retrieved April 19, 2021, from https://www.beeradvocate.com/community/threads/how-many-different-beers-are-there.525578

Charm, T. (2021, March 24). Survey: US consumer sentiment during the coronavirus crisis. Retrieved April 18, 2021, from https://www.mckinsey.com/business-functions/marketing-and-sales/our-insights/survey-us-consumer-sentiment-during-the-coronavirus-crisis#

E. (2016, March 15). Bandit algorithms 5 - The Softmax Algorithm. Retrieved April 18, 2021, from http://duoduo2011.blogspot.com/2016/03/5-softmax-algorithm.html

Eads, L. (2021, March 01). The brands and trends shaping the low- and no-alcohol category. Retrieved April 19, 2021, from https://www.thedrinksbusiness.com/2021/03/the-brands-and-trends-shaping-the-low-and-no-alcohol-category/

Foo, K. (2020, February 21). Multi-Armed bandit analysis of Epsilon greedy algorithm. Retrieved April 19, 2021, from https://medium.com/analytics-vidhya/multi-armed-bandit-analysis-of-epsilon-greedy-algorithm-8057d7087423

Foo, K. (2020, February 21). Multi-Armed bandit analysis of SoftMax Algorithm. Retrieved April 18, 2021, from https://medium.com/analytics-vidhya/multi-armed-bandit-analysis-of-softmax-algorithm-e1fa4cb0c422

Frosmo. (2020, September 09). Multi-armed bandit optimization. Retrieved April 19, 2021, from https://docs.frosmo.com/display/ui/Multi-armed+bandit+optimization

Glowacka, D. (n.d.). *Bandits in Recommender Systems*. Lecture. Retrieved April 19, 2021, from https://glowacka.org/lectures/bandits/bandits-recsys.pdf

Honda, J., & Takemura, A. (2013, November 8). *Optimality of Thompson Sampling for Gaussian Bandits Depends on Priors* (Tech.). Retrieved April 19, 2021, from http://proceedings.mlr.press/v33/honda14.pdf

Hubbs, C. (2020, January 08). Multi-armed bandits: UCB Algorithm. Retrieved April 18, 2021, from https://towardsdatascience.com/multi-armed-bandits-ucb-algorithm-fa7861417d8c

Kun, J. (2013, November 8). Adversarial bandits and the exp3 algorithm. Retrieved April 18, 2021, from https://jeremykun.com/2013/11/08/adversarial-bandits-and-the-exp3-algorithm/

Kun, J. (2013, October 28). Optimism in the face of uncertainty: The ucb1 algorithm. Retrieved April 18, 2021, from https://jeremykun.com/2013/10/28/optimism-in-the-face-of-uncertainty-the-ucb1-algorithm/

LeDoux, J. (2020, March 24). Multi-Armed bandits in python: Epsilon Greedy, UCB1, Bayesian UCB, and EXP3. Retrieved April 18, 2021, from https://jamesrledoux.com/algorithms/bandit-algorithms-epsilon-ucb-exp-python/

Luo, S. (2019, February 06). Intro to Recommender SYSTEM: Collaborative filtering. Retrieved April 15, 2021, from https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

Natarajan, S. (2020, May 3). Stretched exponential decay function for Epsilon greedy algorithm. Retrieved April 19, 2021, from https://medium.com/analytics-vidhya/stretched-exponential-decay-function-for-epsilon-greedy-algorithm-98da6224c22f

Nyström, Å. (2017, August 16). The research behind how brand names impact customers and what name we've changed at buffer. Retrieved April 18, 2021, from https://buffer.com/resources/names-impact-customers/

Pocs, M. (2020, December 08). How to build a memory-based recommendation system using python surprise. Retrieved April 15, 2021, from https://towardsdatascience.com/how-to-build-a-memory-based-recommendation-system-using-python-surprise-55f3257b2cf4

Qutbuddin, M. (2020, October 02). Comprehensive guide on item based recommendation systems. Retrieved April 15, 2021, from https://towardsdatascience.com/comprehensive-guide-on-item-based-recommendation-systems-d67e40e2b75d

Roberts, S. (2020, November 2). Thompson sampling. Retrieved April 19, 2021, from https://towardsdatascience.com/thompson-sampling-fc28817eacb8

Roberts, S. (2020, October 26). The upper confidence Bound (ucb) bandit algorithm. Retrieved April 18, 2021, from https://towardsdatascience.com/the-upper-confidence-bound-ucb-bandit-algorithm-c05c2bf4c13f

Roberts, S. (2021, March 09). Thompson sampling. Retrieved April 18, 2021, from https://towardsdatascience.com/thompson-sampling-fc28817eacb8

Sharma, N. (2021, April 08). Recommender systems with Python- Part Ii: Collaborative Filtering (K-Nearest Neighbors Algorithm). Retrieved April 15, 2021, from

https://heartbeat.fritz.ai/recommender-systems-with-python-part-ii-collaborative-filtering-k-nearest-neighbors-algorithm-c8dcd5fd89b2

Sinha, B. (2018, March). Beer market Size, share & Growth Analysis: Research report - 2025. Retrieved April 18, 2021, from https://www.alliedmarketresearch.com/beer-market

The Carling Partnership. (2017). Who owns who in the world of beer. Retrieved April 18, 2021, from https://www.carlingpartnership.com/insights/who-owns-who-in-the-world-of-beer/

Weng, L. (2018, January 23). The multi-armed bandit problem and its solutions. Retrieved April 18, 2021, from https://lilianweng.github.io/lil-log/2018/01/23/the-multi-armed-bandit-problem-and-its-solutions.html#hoeffdings-inequality

Wikipedia. (n.d.). Conjugate prior. Retrieved April 18, 2021, from https://en.wikipedia.org/wiki/Conjugate_prior