CASE STUDY OF REINFORCEMENT LEARNING ON YELP'S RESTAURANT RECOMMENDATION

SRI HARSHA JANA

Thesis Report

FEBRUARY 2020

ACKNOWLEDGMENT

I would like to thank all the people, UpGrad & LJMU online platform and discussion forums for the information and support in perusing my thesis. Next, I want to thank my family for extending their support throughout the time, and I want to thank my Mentor Abhishek and Student Mentor Rashmi for their quick responses and helping in balancing my work and study. Also special thanks to my friends and colleagues Kedar, Balaji, Ishfaq and Chanpreet.

ABSTRACT

Recommendation systems play a very important role in restaurant suggestions, where there are so many options for the user to choose from and provide a very appropriate and tailored recommendation to match the likes of the user. The current recommender systems provide recommendations based on user, item or hybrid techniques. The aforementioned techniques are static methods, which try to model only current rewards and do not consider the user explicit feedbacks, which are ratings for restaurants, reviews text, and images of food items based on likes and dislikes. One of the prominent techniques, which considers explicit feedback, is reinforcement learning (RL). Which provides a framework to model the recommendation and feedback as Markov Decision Process (MDP) over discrete time steps. Besides, the RL agent will also capture dynamic changes of the user as well as provided with the context of the restaurants for that user. With RL, there are challenges of choosing appropriate actions that are recommendations for users from large discrete action space of all restaurants in data. In this study, to perform exploration and exploitation of actions for the recommendation process, users and restaurants contextual information modeled as CBF technique through the nearest neighbor. Another challenge with restaurant recommendation is capturing the user feedback and defining that as a reward. Continuing the case study, popular methods such as collaborative filtering is used to simulate the rating for restaurant recommended processes. To capture user behavior, temporal difference and q-learning RL techniques are explored that cater to personalized recommendations as learning policy. There will be a comparison of different deep reinforcement techniques to learn optimal recommendation policies given the user's historic rated restaurant information. This enables us to model the recommendation as an offline-online learning process through the RL framework.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1. RL – Reinforcement Learning
2. RA – Reinforcement Learning Agent
3. DL – Deep learning
4. DQN – Deep-Q Network
5. SARSA – State action reward state action
6. CF – Collaborative Filtering
7. CBF – Content-Based Filtering
8. MF – Matrix Factorization
9. MDP – Markov Decision Process
10. RS – Recommender Systems
11. PCA – Principal Component analysis
12. K-NN – K –Nearest Neighbours
13. SGD – Stochastic Gradient Descent
14. ALS – Alternating Least Squares
15. TD – Temporal Difference (policy evaluation in RL)
16. RBM - Restricted Boltzmann machine
17. CNN – Convolutional Neural Networks
18. RNN – Recurrent Neural Networks
19. RMSE – Root mean square error
20. MAP – Mean average precision
21. nDCG – Normalized discounted Cumulative Gain

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 Introduction to Recommender Systems

Recommender Systems are very essential in the current world, from the fact that there is information and data overload in all domains. The sheer volume of albums and songs for online music, different videos in the online channels, online content in blogs, news articles are few of the examples where there is an increase in amount data in recent years. Recommender systems capture the user and product/item interactions and suggest what is user interested in or predict the rating of an item based on user preferences. A variety of flavors and cuisines are available in the context of restaurants and each user will have a personalized liking for taste. In this context, the user will provide a "rating" for a restaurant that would define a user's preference. This will be explicit feedback on the restaurant and helps in capturing the likeness relation between user and item. Recommender systems aim to provide the edge in the market for enterprises and organizations to increase their revenue and customer loyalty. With personalized recommenders, the benefit is by attaining the state of providing the right product or services to the customer. Recommender systems have attracted the attention of a significant number of popular Internet sites, such as Amazon.com, YouTube, Netflix, Spotify, LinkedIn, Facebook and IMDB

1.2 Background of the Study

There are several methods and techniques to solve the problem of recommendations, given users and user rated items and contextual information of items. The broad classification of the recommender systems is context-based filtering (CBF) and Collaborative Filtering (CF). Besides, there are hybrid approaches in the recommendation process. (Beel et al., 2016; Seyednezhad et al., 2018) The core belief of CBF is to recommend items based on the comparison between items and user rated items. However, CF recommendation is upon item-item and user-user similarities. The recommendation process happens as a two-step process where the first step is to learn the user providing a rating for items. Secondly, use these pseudo ratings from the first step to rank a set of items based on the explicit ranking model or CBF, provide recommendations to the user. Usually, the ratings provided by a user to items are sparse. Due to the sparseness of data, a supervised regression algorithm will learn the ratings

by using CF techniques such as Matrix Factorization (MF) or even non-linear parameterized Deep Neural Networks (DNN). (Zhang et al., 2019) The aforementioned techniques are static methods as they try to model the rating without considering the explicit feedback from the user for a recommendation and do not capture the dynamic changes in user preferences.

In the case of restaurant recommendations, given an area or geolocation, the number of restaurants to recommend will be large in number. Solutions for such cases of problems are modeled using multi-arm bandit (MAB) techniques. Provided large actions, the problem of exploration and exploitation arises and to tackle them, Upper confidence bounds (UCB) were discovered (Sutton and Barto, 2015) Given user contextual information and oracle truth about the ratings of a user to each restaurant, UCB solution extended to contextual bandit setting which is nomenclature as Linear Upper confidence bounds (LinUCB) (Li et al., 2010). Bandit algorithms are the prequel to the reinforcement learning setting and the optimization happens for current reward only for the arm pulled. The learning policy in MAB will try to pull the arm with maximum reward. Like supervised (CBF and CF) models, for MAB there is a need for explicit ranking in the recommendation framework.

There is also much work done on reinforcement learning (RL) algorithms where there is explicit feedback from the environment. Some of the greatest examples of RL are playing Go game, Atari games, Robotics and in recent times applied for the problem of recommendation (Mnih et al., 2013). The general setting for reinforcement learning happens between agents and the Environment. From an initial state/observation from the environment, an agent will take action and will receive a reward for the action. Due to action taken by the Agent, the environment's state will transfer to a new state with the changes. [**Error! Reference source not found.**]. The objective of the agent will be to maximize the rewards over a set of episodes rallied between Agent and Environment. In the case of recommendation, the environment will be the user or group of users with similar interests. The agent will be recommending items as actions; the state space will be contextual information of the user and the user-rated items. Selecting an action from the Agent recommended list, the state will transfer to the next state with some reward. The reward will be explicit feedback from the user. In case of news recommendation a click/no click or case of restaurant recommendation the rating provided by the user. The aforementioned is a form of Markov Decision Process (MDP) where

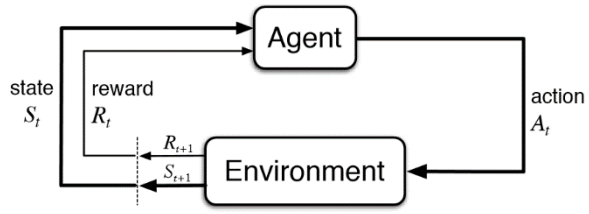


Figure 1.0.1 – Reinforcement learning framework

the reward for current action is dependent only on the reward for the next state. Based on the RL framework the recommendation process can capture the explicit reward from the user on the restaurant and model the dynamic changes of the user interests over time. This enables the current study to have an offline and online learning process that not only considers the current reward but the future rewards as well.

1.3 Problem Statement

The recommendation process in a classical framework consists of using a supervised model for predicting the rating of the items given user and item contextual details. This is a static process and will always try to model only the current preferences of the user. These systems will fail to capture the dynamic changes of the user interests over a period. Due to the drawback, user interests in the recommendation process will decrease. Besides, due to the passive learning process of supervised techniques, the models do not capture the explicit feedback from the user in real-time and in some cases, do not incorporate the historic browsing patterns from user data. To tackle this, supervised techniques will need frequent retraining of models and will cause cost and maintenance overhead in case of enterprise-level application or product life cycles. Therefore, there has to be an online/offline learning process that considers feedback and provides recommendations with a proper policy. The policy should consider current rewards along with future rewards to capture the dynamic changes happening in the user preferences. To capture the changes in preferences the algorithm needs to consider the contextual information from the restaurants such as type of food cuisine, average ratings of the across users, etc.

Given a set of restaurants, recommendation happens after ranking them. This requires additional efforts from Information retrieval and ranking techniques to coexist with CF techniques. This is nothing but the policy for the recommendation process, where ranking performs the task of exploring new restaurants. The policy should be smart enough to perform exploration and exploitation to avoid the deterministic nature of recommendations. Handling the large set of restaurants is another case of large source data to search, rank or explore. Restaurant data combined with its contextual information and user preferences will add to a large amount of data processing. Adding to the aforementioned challenges, data usually from the ratings will be sparse and that makes the above tasks hard to tackle.

Considering the problem setting to reinforcement learning, there are techniques such as model-free RL, epsilon greedy approaches and policy learning in RL. Which can learn the optimal recommendation policies? Also, provide a generalized framework for personalized recommendation considering the explicit feedback as rewards from the environment.

1.4 Aims and Objectives

The current study aims to formulate the recommendation process a reinforcement-learning framework. With state, action, reward, and next state formulated as MDP process. **Error! Reference source not found.** represents the recommendation as the formal setting of an episode in reinforcement learning will be a tuple of (S(t), A, S(t+1), R) where each variable in the tuple is explained below Table 1.0.1.

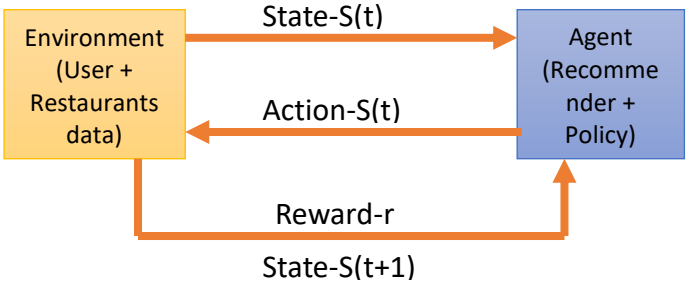


Figure 1.0.2 – Reinforcement learning for recommendation

State S(t)	{Rest1, Rest2, Res3}
Action A(t)	Rest4, Rest5, Rest6
Reward r	{1, 2, 3, 4, 5}
Next State S(t+1)	{Rest5, Rest1, Rest2}

Table 1.1 – State action and reward sample representations with restaurants data

State S(t) – Given historical rating/reviews data of the user to restaurants, the goal of creating a state is to get an overlook of capturing what the user was interested in or did not like. The contextual features of the restaurants represent user likes and dislikes. A 'k' window of previous reviews can form the basis of the state. The state will be a matrix with columns as time steps in the window and rows the features of a given restaurant.

Action A(t) – At time t when the Agent receives the state S(t), from the given set of restaurants using exploration and exploitation techniques such as epsilon-greedy with learning policy select a restaurant. This will form the action that the agent will send back to the environment for the reward.

Reward r – the reward for the recommendation will come from the environment. Based on the historic review data of users the rewards for the actions A(t) will be from real-valued numbers. If the recommended actions A(t) are closer to user historic rated restaurants then, state S(t) will move to S(t+1).

Some of the characteristics of the recommendation through RL are as follows, these will be the objectives that are focused to solve in this case study and the detailed out in Chapter 3.0. The first one being the large action space, items in case of restaurant recommendations are restaurants itself. Since they are very large in numbers if the RL agent has to choose an action using the brute-force technique the responsiveness of the RL Agent system will take hit. Hence, we will study the usage of CBF techniques to reduce the number of items to choose from in conjunction with RL Agent policy. Usage of CBF leads to a deterministic policy in RL Agents; hence, we will incorporate the usage of Gaussian Noise with mean zero and hyper-parameter controlled variance, which will lead to quasi-deterministic policy for the recommendation. Secondly, the Environment of the user should be very dynamic and should simulate the rewards as per user preferences. Since the state is represented as contextual information from the user rated restaurants, in this current study aims to simulate the rewards by using classical Collaborative filtering techniques will be explored on the assumption that user with similar interests will provide similar ratings to the restaurants. The rewards from the environment can be either binary in nature or a real value number, hence to capture these rewards and maximize them Q-Learning techniques will be explored in this case study.

1.5 Scope of the Study

In this case study, the following points cover the scope of the study:

1) Yelp's business data used for the aims and research objectives of this case study. Data consists of users and ratings provided by these users for the business data. The focus will be on the subset of Yelp's restaurant data filtered using keywords such as 'food' and 'restaurants'.
2) Pre-processing of the data and identifying all the contextual information of a restaurant. Choosing proper techniques for prepossessing of the data, missing value interpretation, conversion of nominal and ordinal variables to one-hot encoding or use them as is. As the feature-space is large, techniques of dimensionality reduction techniques such as Principle Component Analysis (PCA). Alternatively, build an Auto-Encoder model and identify each restaurant with an embedding of its latent space. The reduction of feature space will help with minimizing the state space.
3) As there is no specific model for the environment, which can tell about state transitions. The aim is to model the dynamically changing user preferences as a learning policy in

RL Agent using Model-free approaches. Initiate with states and then recommend a list of restaurants as action.

4) Moreover, to tackle large discrete action space for a recommendation, using unsupervised Nearest Neighbour CBF to perform relevant items recommendation to the users based on a user‘s history of rated restaurants.
5) Finding out the optimal reward function using collaborative principles, which states that users with similar interests will provide a similar rating to the recommended items. Hence trying out base rewards function. Defining the reward function as a CF model or technique in the field of the user environment simulation of rewards for restaurants, whereas most of the RL algorithms revolve around using Click through Rate (CTR) as rewards.

1.6 Significance of the study

The problem setting and solution of recommendation using RL for different domains such as Music, Movies, Products and new articles (Zhang et al., 2019) are successfully deployed for production environments at an enterprise level. For each of the recommendation process modeled as MDP over different time steps but, it has applied for one user session. For example, in the case of music recommendation, the session time of the user is a direct evaluation of the state and actions. However, in our case study at any given point in time, the user can order or visit only one restaurant. The significant contribution of this study will be to add knowledge on how to use the recent trends with the RL for the restaurant recommendation process. Additionally, the study will also focus on a comparative case study of different reward functions derived from CF models with contextual information. Since the rewards in most of the recommender-system are click events and these are discrete values, the proposed approach uses tries to minimize the cost between recommended ratings from reward function to the ground truth. Thereby maximizing the cumulative rewards from the long-term recommendation process. The approach will add a new way of using hybrid methods of recommendation systems. Using CF for the prediction of the ratings and CBF for providing recommendations, the setting applied to the RL problem. In this study, the CF using similarity approaches will be representing the user simulation environment and the CBF method combined with Deep Reinforcement learning techniques will provide the recommendations. Using CBF techniques will cause deterministic recommendations based on the user current state. Combining a mean zero and controlled variance noise with the epsilon greedy policy for

the recommendation is another contribution in terms of increasing the training time for RL Framework and avoiding deterministic recommendations.

1.7 Structure of the study

In the following chapters, we will have a detailed literature review of the existing recommendation systems. Adding to the literature review, there will be a discussion section on the comparative study of hybrid recommendation systems and how do they compare to the features of the recommendation using RL. In Chapter 3.0, the report will focus on the proposed methodology and try to provide solutions to the objectives mentioned earlier in section 1.4. In Chapter 4.0, discussion on the methodology followed for implementing the RL solution for the recommendation. This section details out how these solutions work for the given restaurant dataset. Chapter 5.0, discussed the implementation of the solution and results obtained from the experimentation. This will also cover what are the hyper-parameters used for experimentation and the reason behind it. Finally, in Chapter 6.0 the limitations and future work discussed and concluded.

CHAPTER 2

LITERATURE REVIEW

2.1 Recommender System

The main data dimensions in review data are users, items and user-provided ratings to the items, in some cases date-time as well. As per literation review (Beel et al., 2016), the main components that make up a complete framework in recommender systems are the review and rating corpus data of user items and ratings, recommender algorithm, items data repository and their contextual information, finally evaluation metrics. The recommender algorithm is then subdivided into different types (Seyednezhad et al., 2018)based on the key ideas that content-based filtering is that users are interested in items that are similar to items the users previously liked. Where are in collaborative filtering, the idea is that users like items that the users' peers liked. There are also hybrid mechanisms in the recommendation process classifications. Recent developments in Machine Learning techniques such as Supervised Learning, Deep Learning, and Reinforcement Learning are studied. For recommendations using Reinforcement learning, classical multi-arm-bandit techniques are also used.

2.1.1 Content-based Filtering (CBF)

The content-based filtering method works with the description of the item and/or the contextual information that represents the item. In the case of restaurants, the contextual data that can describe a restaurant are categories of food that it specializes, geo attributes, features, seating capacity, etc. Contextual information also attributes of the restaurant such as parking are, is Wi-Fi available, etc. In content-based we will have a content analyzer in which we will pre-process the data and perform analysis of the features. Based upon the user profile, like past ratings and likes, recommend similar items.

2.1.2 Collaborative Filtering (CF)

The classic CF approaches try to model the user-user and item-item interactions through similarity. Usually, the data considered here will be sparse, as most of the items will have less count of ratings from the user. Pearson co-relation gives a score for the similarity of the items and users. To handle the sparseness of the data, Matrix Factorization (MF) is one such

technique, which is an extension of dimensionality reduction techniques such as SVD (Singular vector decomposition). MF method will learn the item-to-item or user-to-user interactions in latent space. For the supervised algorithms, learning happens with proper cost functions such as SGD (stochastic gradient descent) or ALS (Alternating Least Squares). Besides, there are Deep learning techniques applied for collaborative filtering and matrix factorization techniques that are more accurate than classical algorithms such as SVD or logistic regression that are not too good at handling sparse data.

2.1.3 Hybrid Approaches

CBF and CF techniques have their advantages and disadvantages; the idea of the hybrid approaches is to combine both CF and CBF to achieve good accuracy of rating prediction. (Seyednezhad et al., 2018) Unlike traditional recommender systems that solely rely on user preferences for some items, context-aware recommender systems use contextual information about the activities in addition to the user's preferences. Different techniques of incorporating context information into a recommender system model are contextual pre-filtering, contextual post-filtering, and contextual modeling. Contextual pre-filtering will help in reducing the item space from which recommendations made and Contextual post-filtering will help in cases when ranking items before displaying to the user.

$$Rating = Items\ X\ Users\ X\ Context$$ *Equation 2.1*

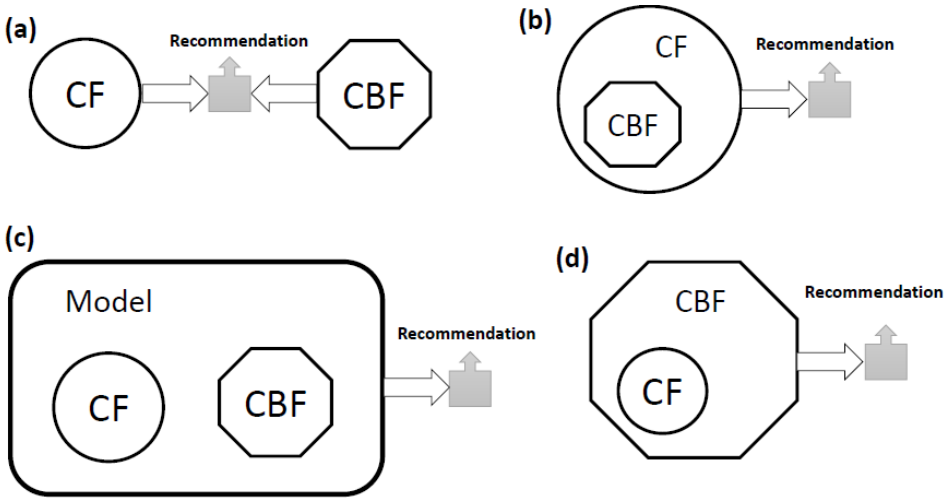


Figure 2.0.1 - Explaining the different types of hybrid recommender systems - (Seyednezhad et al., 2018)

Hybrid approaches as described in the above figure 2.0.1 are several ways of combining the two different built solutions. CF and CBF can work together in tandem or hierarchical way to provide recommendations. As CF captures the user and items that similar users liked, whereas CBF captures the item and similar items, just by combining these two ideas will give a good collection of items to recommend from very large itemsets.

2.1.4 Deep Learning in Recommendations

The recent trends in deep learning have also made its way to recommender systems. The most common use cases of DL in RS are the generation of the user or item embedding and learning the non-linear interactions between users and items. The common DL architectures in Recommender systems are Auto Encoders, which are good at learning embedding, Multilayer perceptron, for non-linear interactions. CNN's are used for extracting representations of items form text or images. RNNs are good at learning sequential decision-making by considering user's previous rated/liked items (Zhang et al., 2019). Besides, from the literature review, RBM is an alternative for MF. Summarizing the DL techniques, the advantages of using them are for flexibility, non-linear transformations but they are also complex to train and need more data. With DL in RS, we will not have interpretability. Deep learning architectures have successfully adopted classical approaches such as CF and CBF, one of the trending examples is Google's wide and deep network by (Cheng et al., 2016).

2.2 Survey on Restaurant Recommendation

As per the literature survey, the most common techniques used for restaurant recommendation comes from hybrid approaches to using collaborative and content-based filtering. A simple architecture of the Collaborative filtering technique proposed by (Farooque et al., 2014) talks about using the user's contextual information and K-means to recommend restaurants for the user. This is a classical user-user correlation analysis to provide recommendations. One more way of joining the CF and CBF technique is to concatenate the features generated by each technique. (Mustofa and Budi, 2018) (Shi et al., 2019) proposes this hybrid technique to extract the user and item features and build a model for recommendations. Users also provide very vital information such as images of the food/order received or reviews for the food in online systems. This data is a very good representation of the sentiment of the user for the food as proposed by (Ganun et al., 2013), where they use user reviews information for predicting the rating of the user. Following Table 2.1 from (Kumar et al., 2020) provides a good comparative study on the different types of recommender techniques and usage.

Sr. No.	Authors Name	Publication name and Year	Approach Used	Techniques and Algorithms Used	Dataset Used	Advantages
1.	Wubin Guo et al. [9]	IEEE, July 2007	Content Based Filtering Approach	Knowledge Based Reasoning.	Restaurant menu ratings from surveys.	Improve prediction quality.
2.	Chi-Chun Lo et al. [10]	IEEE, December 2008	Content Based Filtering Approach	TFIDF, Latent Semantic Analysis (LSA), K-Nearest Neighbour.	Restaurant menu ratings from surveys	Improve prediction quality.
3.	Yoosoo Oh et al. [11]	Springer US, July 2009	Content Based Filtering Approach	Context-awareness integration, Personalised food recommendation	Data collection using sensors.	Better recommendation quality by using sensed environmental information.
4.	AsamiYajima et al. [12]	IET, September 2009	Content Based Approach	Browsing log, Recipe Retrieval	AJINOMOTO Web recipe.	Better Recommendation quality.
5.	Yoko MINO et al. [13]	IEEE, November 2009	Content Based Approach	Recipe recommendation on the basis of nutritional balance.	AJINOMOTO Recipe Encyclopaedia.	Flexibility in the control of calorie intake depending on user's schedule.
6.	Jong-Hun Kim et al. [14]	IEEE, November 2009	Content Based Approach	Personalised diet recommendation service.	Data recorder that stores XML data of user's health.	Better recommendation based on specific diet by observing the nutrition condition of users.
7.	Jill Freyne et al. [15]	ACM, New York, USA, February 2010	Hybrid Recommendation technique	Personalised diet recommendation	Opinions of 183 users for the recipes and foods over a period of 3weeks	Accuracy in Recipe Recommendation
8.	Chang-Shing Lee et al. [16]	IEEE, February 2010	Hybrid Recommendation technique	Type 2 Fuzzy ontology model	Taiwanese Food dataset	Used Fuzzy ontology, much better than classical ontology. Effectively handled and processed uncertain data and knowledge
9.	Wahidah Husain et al. [17]	IEEE,2011	Data mining	Case based reasoning, Rule-based reasoning (CAMP,PRISM and CAMPER algorithm) and genetic algorithm	User Database, Knowledge database and Food database	High accuracy of recommendations when the data set is large
10.	M. A. El-Dosuky et al. [18]	Advanced machine learning technologies and applications :first international conference (AMLTA) 2012, Cairo Egypt, December 8-10,2012	Semantic Recommender system	Hermes framework, TFIDF, binary cosine, Jaccard.	Food database(Corpus of 300 food items) provided by United States Department of agriculture (USDA)	Proposed system had a better F-measure than existing semantic recommenders at that time.
11.	Sumodh Sawant et al. [19]	2012, Stanford University	Collaborative filtering approach	k-nearest neighbors clustering, weighted bipartite graph algorithm and their combinations	Yelp Dataset Challenge data	Compared algorithms in which "cascaded clustered multistep weighted bipartite graph projection" algorithm performed best hence improved predictions quality.
12.	Shilpa Dharkar et al. [20]	IJSER ,2012	Content Based filtering, Data mining	Decision tree algorithm(ID3 and C4.5),bagging	Content database collected from external web atmosphere	An effective web data mining solution to e-commerce for personalized diet recommendations.
13.	Achmad Arwan et al. [21]	IEEE , 2013	Domain Knowledge based recommender system	OWL,SWRL,SPARQL,JAVA SWING,JENA	Database containing data of 30 patients	Showed 73% accurate recommendations
14.	E-Seok Jo et al. [22]	International journal of advancements in computer technology(IJACT),August 2013	Content Based filtering approach	Termite Colony algorithm(TCA) k-means	Blogs acquired from http://kitwitobes.com/clusters/feedlist.text	Food recommendation based on similarity levels of blog and using TCA K-means algorithm that simulates habits of termites improved processing time of clustering
15.	Irshad Faiz et al. [23]	IEEE,2014	Knowledge- Based recommender system	modeled SHADE(semantic healthcare assistant for diet and exercise) architecture using OWL based ontology, SWRL and pellet reasoning engine	USDA and myfitnesspal	Better recommendation because of simultaneous diet and exercise suggestions
16.	Shreya B.Ahire et al. [24]	IEEE computer society,2015	Demographic and knowledge based recommender system	The Decision tree algorithm ,SWRL, OWL	Database maintaining user profiles based on health care information	Better understanding of User's query results in more accurate and relevant recommendations

Table 2.1 – Showing the different types of restaurant recommendation work from different papers - (Kumar et al., 2020)

2.3 Reinforcement learning for Recommendations

The classical methods of recommendation are static in nature as they capture the user's current preference. Supervised recommendation tries to maximize only immediate rewards and forget about future rewards while not considering explicit or implicit feedback from the users. To improvise on providing personalized recommendations to users (Shani SHANIGU et al., 2005) formulated recommendation as a sequential decision problem rather supervised prediction. MDP fits the scenario and initial work had performed to prove that recommendation achieved

through the RL setting. From there, many papers have been published which started using RL for RS. Following are the sections that will explain in detailed on the different approaches

2.3.1 Multi-Arm Bandit problems

The multi-arm-bandit problem is a retro reinforcement learning method, explained with a gambler given a slot machine with multiple arms and each arm is associated with a reward distribution. The objective is to pull the arms one by one to get an idea as to which arm can give him the most reward. This is an exploration vs exploitation trade-off problem too. When the MAB setting applied to RS, (Bouneffouf and Rish, 2019) each item becomes an arm and each item is associated with a reward. The learning policy will need to consider the exploration and exploitation as well as contextual information if the items are having any to perform recommendations. The algorithm observes the current user and a set of items or arms or actions together with their feature vectors, the utility or dot product of user and arm referred to as the context. Based on previous trials, actions chosen with the threshold of distribution parameter μ (mean) that defines upper confidence bonds of the rewards distribution. The standard evaluation metrics for this methodology are using cumulative regret (loss) or gain.

2.3.2 Formulation of MDP

Markov chains consist of a state and transition of those states to next with some probability. The extension of Markov chains to a stochastic decision process makes up with MDP. As such, it is widely used in applications where an autonomous agent is influencing its surrounding environment through actions. An MDP is by definition a four-tuple: (S, A, R, St) where S is a set of states, A is a set of actions, R is a reward function that assigns a real value to each state/action pair, and St is the state-transition function, which provides the probability of a transition between every pair of states given each action. There are several variations in adopting the MDP process for recommendations. As per the literature review, the RLCF algorithm (Lee et al., 2012) models the rating of an item as the MDP process in which state is K historic ratings of the user and acting with some policy, the transition will happen to next state. (Choi et al., 2018) proposes the formulation of MDP based on user-item similarity matrix with state representing the clusters of users and items. Using Q-learning and SARSA to learn optimal recommendation policy (Zhao et al., 2019) the basic MDP process for recommendation uses the users historic rated or liked items as the current state. The action corresponds to the recommendation of the items to the user based on the state. The user or Environment will accept the recommended item or select a new item outside of recommended actions, thus transitioning

to the next state. The rewards function is a compute utility (a simple dot product) of actions and states. Reinforcement learning's key thing is to find the optimal policy that maximizes immediate and future rewards. Maximized the rewards using the concepts of q-value and Bellman equations. Policy iteration learning happens for the optimal value function. Policy evaluation helps in choosing the best action that maximizes current and future rewards. There are several techniques such as SARSA, Q-Learning that are majorly used for policy iteration and finding the optimal policy considering states and actions. In addition to Q-Learning, there are Temporal Difference (TD) and Monte Carlo tree search are model-free approaches used for finding the optimal policy for a given state. (Zheng et al., 2018) propose a methodology of performing offline learning of recommendation strategies and use a dueling bandit technique for exploration and exploitation. The earlier mentioned approaches use epsilon greedy policy for the explore-exploit dilemma.

2.3.3 Deep Reinforcement Learning for Recommendation

Deep reinforcement learning successful implementations started with learning images for the environment like in Atari, solution built on popular DQN architecture (Mnih et al., 2013). There are two varieties of DQN architectures. The first case takes in the current state as input, all available actions, and their corresponding q-values as output. However, this architecture is not suited for the recommendation process with large and discrete action space. There is another version of the DQN that takes in the current state and possible actions as input and output of q-value. This is more suited for recommendation systems and popularly used in benchmarking the Deep reinforcement learning for recommendations (Zhao et al., 2019). There are variations of the DQN architecture with the adaptation of CNN and RNN architectures for forming relations between states and actions. In the case of very large action space, the following papers follow the Actor-Critic architecture that creates two Deep Learning models. The actor is responsible for generating a proto action that is specific to the current state. The output of the actor gets transacted to actions either a linear combination of weights to action space or through K-NN (Dulac-Arnold et al., 2015). The output of the Actor chained to Critic that replicates the DQN framework.

2.4 Discussion

Content-based filtering recommends items based on a comparison between the content of the items and a user profile. The user profile can have personal information and detailed

information about each item that the user has liked. Typically, context such as words that occur in a restaurant name, its attributes like cuisine is as contextual representations. Several issues or challenges faced while implementing a content-based filtering system are as follows. First, context needs to assigned to a particular restaurant for this all the restaurant data needs to be looked holistically and then a knowledge set of data needs to derived which has associations of overall contextual information to each restaurant. Second, the contextual information represented in both user data and the item data needs to provide meaningful representation. This can be achieved either by just performing similarity measures between users and items. However, Similarity effectiveness comes from the data types of contextual information. If the data is one-hot encoded, similarity measures such as Jaccard similarity works best. Else, if the vectors are real numbers, then cosine similarity works best. However, if the dimension of these contextual attributes is very large, then employ techniques such as PCA or representation of these contextual features as embedding. Thirdly, a learning algorithm that can learn user preferences based on items and provide recommendations based on the contextual information of users and items. Techniques such as supervised learning and/or ranking algorithm work well in case of recommendations. Neural networks and the Bayesian classifier are among the learning techniques for these purposes.

The ability of a learning method to adapt to changes in the user's preferences also plays an important role. The learning method has to be able to evaluate the training data, as instances do not last forever but become obsolete as the user's interests change. Another criterion is the number of training instances needed. A learning method that requires many training instances before it can make accurate predictions is only useful when the user's interests remain constant for a long period. The Bayesian classifier does not do well here. There are many training instances needed before the probabilities will become accurate enough to base a prediction on. Conversely, a relevance feedback method and the nearest neighbor method that uses a notion of distance can start making suggestions with only one training instance. The learning methods applied to content-based filtering try to find the most relevant documents based on the user's behavior in the past. Such an approach, however, restricts the user to documents similar to those already seen. This is the case of an over-specialization problem. As stated before the interests of a user are rarely static but change over time. Instead of adapting to the user's interests, after the system has received feedback one could try to predict a user's interests in the future and recommend documents that contain information that is entirely new to the user. A recommender system has to decide between two types of information delivery when providing the user with recommendations: Exploitation. The system chooses documents

similar to those for which the user has already expressed a preference. Exploration. The system chooses documents where the user profile does not provide evidence to predict the user's reaction.

2.4.1 Supervised Learning & Reinforcement Learning

Supervised Learning	**Reinforcement Learning**
In **supervised learning**, works with **objects or datasets**. There is **no interaction** with the environment and given a dataset, you are required to predict the target	In **reinforcement learning**, works with the **processes** where the **agent actively interacts** with the environment
supervised learning is **passive learning**, where the agent learns only by extracting features from a given dataset	RL is **active learning**, where the agent learns only by interacting
In supervised learning, there is a **teacher (ground-truth)** which tells you whether the result for a given observation is correct or not	The environment acts only as a **critic**, where it tells you how good or bad the action is by giving rewards.
Minimizes the loss function with the ground truth-value. In the case of the **regression mean square error** or in case of **classification sigmoid function**.	The objective function is to **maximize the total rewards** by interacting with the environment. Uses **exploration and exploitation** techniques.

Table 2.2 - Showing the difference between supervised and reinforcement learning

2.4.2 Q-Learning for Recommendation

The Q-Learning in RL defines techniques and methods to calculate reward by being in a state, Defined by equation 2.1. The recommendation strategy defined using this equation is as follows; an optimal policy to take action 'a' from a given state 's' which is the user's history of rated restaurants. To maintain the finite state, a window of length k of contextual information of restaurants can form state space. Once the Agent observes the current state 's' of the user, based on CF or CBF techniques, the recommendation of restaurants is given to the user. Based on the rewards 'r' observed from the environment, the agent will learn the policy through the epsilon-greedy strategy. To maximize the current and future rewards, γ will control the weightage given to future rewards. The parameter α used to control the learning rate of the policy. To perform the Q-Learning, a DQN can be used which will learn the Q-value based on current state and action.

$$Q(s,a):=Q(s,a) + \alpha(r + \gamma * \max Q(s',a) - Q(s,a))]$$

Equation 2.1 – Q value equation for updating DQN uses reward from current state and q-value from next state to calculate current q-value

Since the action space of restaurants to choose from is very large in number, the nearest neighbor approach can be used form the context-based recommendation process to reduce the number of restaurants to recommend. The input for the nearest neighbor will be restaurants in the current state. Based on epsilon-greedy policy, K random restaurants picked from reduced action space and sent to Environment. Another option for working with large discrete action spaces is the actor-critic model of the agent. The actor will take the current state of the user and then generates proto actions that represent weights associated with items; the linear combination of the proto actions with action space will give the next actions that will be input to the critic model. The critic is very similar to the DQN agent. A very different way to handle large action space is Contextual bandits, where all the restaurants will become the arms to pull. A linear function for user preference and recommendations provided will tell about the importance of the recommendations.

The environment will be a simulator in offline learning in RL. The environment will get the recommendations from the agent-based on actions the user will either choose an action and move to the next state or decide to remain in the current state. Based on the collaborative filtering techniques, the user with similar interests will provide a similar rating to restaurants. If the compute utility (dot product) of the embedding of recommended action is close to the next ground truth restaurant's embedding, then the rating copied from the ground truth as the reward. Then the state 's' will transition to 's``'.

CHAPTER 3

PROPOSED METHODOLOGY

3.1 Methodology

Formulating recommendations as a sequential decision problem in the RL framework is as follows. The system has a user history of rated restaurants, user ratings of restaurant data, and contextual information of restaurant data. The user interacts with the system and begins to get recommendations on where to order or visit the restaurant. The RL Agent will refer to user browsing history and then recommends some restaurants. Table-3.1 refers to an example flow of data from the user perspective. State transition will happen only if user orders/visits to the recommended restaurants. The goal of the system would be to make the most successful recommendations in a given sequence.

User History	Restaurant-R1	Restaurant-R2	Restaurant-R3	Restaurant-R4	Restaurant-R5	Restaurant-R6
States	R1	{R1,R2}	{R1,R2,R3}	{R2,R3,R4}	{R2,R3,R4}	{R3,R4,R6}
Recommendations	R7	R8	R4	R8	R6	R9

Table 3.1 - Showing an example sequential decision-making process with restaurant recommendations

The sequence of these recommendations is nothing but episodes in the RL framework. The RL agent's objective is to maximize the cumulative rewards in one episode considering current and future rewards. There are several techniques in RL (Sutton and Barto, 2015) To solve for optimal policy. Quoting an example from the grid-world problem and extending this analogy for the recommendation as follows. In the grid world problem the starting points can be any cell in the NxM matrix, some starting states are good and some are bad. One task of recommendation is very similar to one-step in the grid world and recommendations of restaurants are actions correspond to movements from a position in the grid (left, up, down, right). The user/agent to be in a particular position is dependent on the previous state (previous set of positions in the grid world). A user going to a restaurant is very similar in nature, as it determined by contextual parameters such as location, food or cuisine type. If the state in the RL framework captures this information then based on current state RL Agent can provide recommendations. Research for session-based recommendation by (Quadrana and Cremonesi, 2020) Implies on the logic of looking at a previous session to provide the next recommendation. Looking at the complete history of the user rated restaurants is not feasible, as the state space will become too large, hence this case study aims only 'K' number of restaurants to form a

state-space, as this will be sufficient to capture the dynamic changes from the user. Considering the above analogy, the recommendation process is first-order MDP and is a sequential decision process between a user and the RL agent with the state, action, and rewards revolving between user and agent. In session-based domains like those that news articles or product recommendations, one session for the user has multiple items/articles recommended (Zheng et al., 2018). However, user for the case of onsite visiting or online order use cases visits only one restaurant. Hence, in this study, the RL agent will be providing only one restaurant to the user. This will simplify the reward function and user environment-simulation flow. A detailed explanation for the same in the upcoming section 3.2.1. The second factor considered for the sequential decision process is the order of restaurants concerning time. The sequential optimization functions, namely the policy evaluation step in the RL framework will also be dependent on 'K' history of time ordinal restaurants that the user has rated. The intuition for this approach is that the next recommendation item is dependent on the previous steps. Optimizing policy iteration in the RL framework for the recommendation process, several techniques such as Nearest Neighbour, RNN, and simple item-based similarity used to recommend items based upon previous items. The case of news articles and product recommendations (Zhao et al., 2019). Extend this idea for the recommendation. Considering several methodologies in the RL, techniques such as Temporal Difference and Q-Learning are present for learning optimized policy. More details about this topic are in section 3.2.2.

3.1.1 Formulating MDP

The formal setting of an episode in reinforcement learning will be a tuple of (S(t), A, S(t+1), R). The nomenclature for the same tabulated below.

RL Component	Symbol	Recommender System Settings
State Space	s(t)	K-windowed previous state of the user at time t, where x(i) {x(1), x(2)……x(n)}. X(n) being feature vector of restaurant n.
Action	a(t)	Action from RL Agent based on state space s(t) at time t.
Reward	r	The environment based on action taken gives the reward.
Transition State	s(t+1)	If user likes the action then transition to s(t+1), else be at s(t)
Discount Factor	γ	The discount factor for future rewards
Policy	$\pi(a\|s)$	The probability to take action in a given state s.
State value function	$v\pi(s)$	represents the value of the state s.
Action value function	$q\pi(s, a)$	represents the value of performing a particular action while in state

Table 3.2 - Different components or functions in the RL framework

State space s(t): To start with, the users sampled from the user-review data and reviews for a set of restaurants will be chronologically ordered. For a defined 'k' number of restaurants will form the initial state space s(0). The state will be a matrix of K cross N where N is the dimension of the feature space, which is the latent dimension of the embedding of a restaurant. The representation of the current state of the user will be in the form of a matrix where each row represents the index of time. This state will go to the RL agent and based on this state agent will provide recommendations, which is synonymous to action from RL Agent.

Action a(t): Agent will have access to the full set of restaurants and their embedding. An agent will fist query all relevant restaurants from the data and based on optimal policy will provide a restaurant as a recommendation. Since the action is large and performing a random selection from this action space is not optimal, a CBF algorithm such as nearest neighbors based on contextual information used to select actions. The Nearest neighbor uses KD-Tree to represent all restaurant embeddings. The number of neighbors to query will be hyper-parameter that is tuned while experimentation. The policy will become deterministic if the state becomes static and Agent will learn only repetitive recommendations. To avoid this behavior, small perturbation applied to the current state to perform exploration will help in providing different recommendations. By using exploration and exploitation algorithms such as epsilon-greedy, optimal policy evaluation implemented using Q-learning techniques.

Reward r: The recommended restaurant a(t) will then move back to Environment where the user can like or rate the restaurant. Based on the collaborative filtering methodology, the user with similar interests will make a similar decision on the same restaurants. Using this principle, a reward function formulated will in actual mimics the user behavior. The reward function will take in state-s(t) and action a(t) and provide a reward r. The solution to simulate reward follows the cosine similarity approach by giving weightage to current state-s(t) and action a(t) that is part of the user's historical data. Alternatively, it can also be a parameterized model such as Matrix Factorization or a Deep-MF (Zhang et al., 2019). Since the current state is having contextual information, the parameterized model can use this information to provide a rating that is nothing but a reward.

Transition state s(t+1) : This will be the transition from state s(t) to s(t+1) for given action space a(t). If the provided action has a positive rating from the user, then action a(t) will be appended to the end of the current state s(t). Since the constraint to keep current state size as 'k', remove the earliest item from the current state as well. This is a similar analogy of the moving window function, the only difference is that window movement is conditional based upon the rating provided by the user.

3.2 Yelp's Business & Review Data

The Yelp data set consists of business, reviews, user and tips data. From the Yelp dataset, we will be filtering the using only the restaurant business data. The files are present in the form of JSON. The total data set contains around 4.5 million reviews provided by 1.1 million users on 74000 businesses. That makes the data size of 8GB. Using cloud-computing resources followed the data filtering to filter out only restaurant data using keywords. Save the data as CSV files for further processing. Explained the detailed data dictionary and pre-processing steps in the following sections.

3.2.1 Data Pre-Processing

Starting with the restaurant data attributes, processing rows level attributes into columns are tedious processes, many of the attributes have no values. Starting with the assumption that none value of data means, either the attribute not related to the business. Imputed the value with "false" in such cases. The attributed as exploded as rows in the data, which roughly constitutes to around 57 features. Some of the key features continue real value such as latitude, longitude, opening and closing hours. There are binary features as well such as if Wi-Fi present, car parking available, etc. As the features are very sparse in nature, we will be using dimensionality reduction to bring down to latent space with the appropriate number that will explain the variance in data. By following PCA or embedding shown in Figure 3.0.1. The category features in restaurant data are very useful for representing the taste and comfort food of the user. Using word embedding or by using count vectorizer, created additional features along with the features that represent the restaurant.

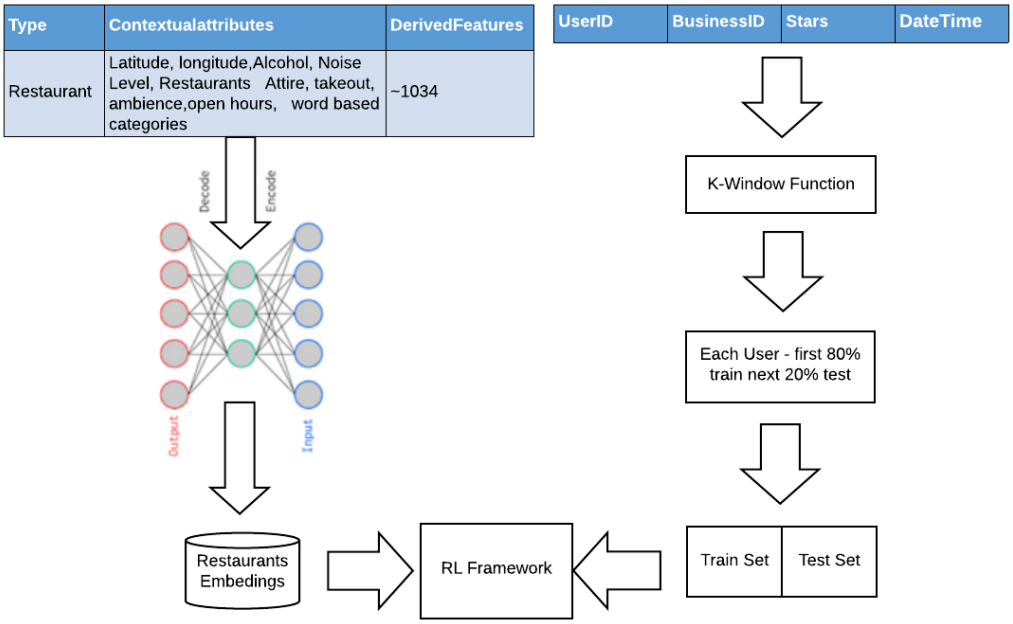


Figure 3.0.1 - Flow diagram of processing data from user reviews and restaurants to form input to RL Framework

The review data is very simple to pre-process as there are no missing data and it is a very complete data with ratings and time of the rating. Used this data for understanding the domain of using recommendations for restaurants based on ratings provided by a user to a restaurant. In the case of RL, using the same data by creating a window function of length "K" on historic ratings of the user. By moving the window function selecting only good ratings provided by the user to restaurants and create the state, then refer to the next K+1 point to be action and the reward will be the rating provided to k+1th restaurant. This forms the basis of the user review data that will feed into the RL framework. Take the first 80% of each session of the user as the training set and next 20% as a test set. The same setting works as the validation set as well. When selecting users, split the users into two parts one part for training and another set of users for testing.

3.3 Reinforcement System Design

In the following section, we will describe the components involved in the reinforcement learning framework design. To maximize the cumulative reward, the RL Agent will go through a training process to learn the optimal recommendation policy. The system has several components as described in Figure 3.0.2

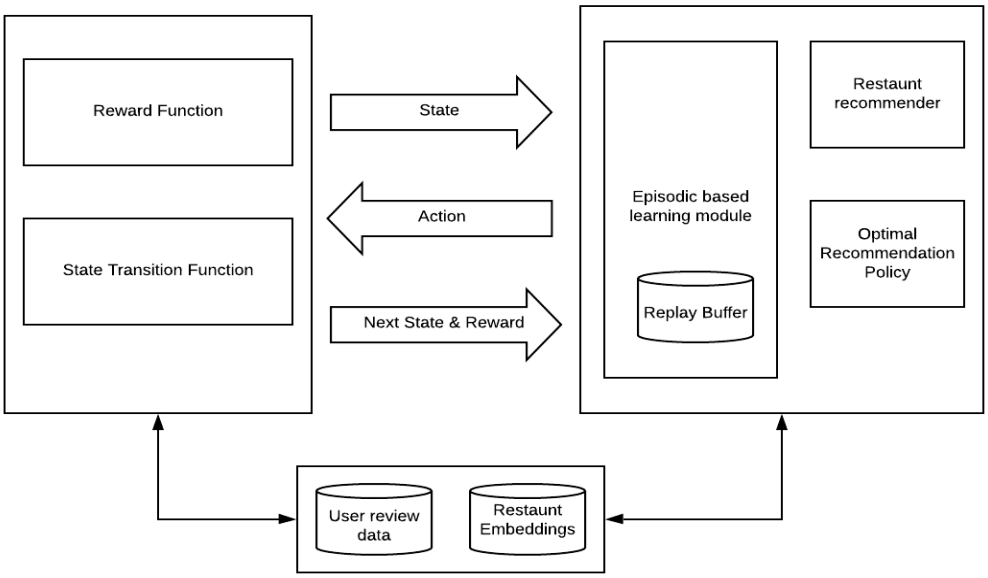


Figure 3.0.2 - Systems Integration diagrams and communications between user environment simulator and RL Agent. Also, show common data access by both of these modules.

3.3.1 User-Environment Simulation

To learn the optimal recommendation policy for the given user is the task of RL Agent. Users can give rewards as ratings to the recommendation provided by the RL agent. These two aspects of the RL framework arise the need of creating a broader user-environment simulation algorithm that can work for all user's data in general and provide details to RL agent based upon historic data of ratings provided by the user to respective restaurants. The simulator should provide a real number reward 'r' based upon state s(t) and action a(t) as given by the equation below.

$$r = f\big(s(t), a(t)\big) \qquad \textit{Equation 3.1}$$

The mentioned equation 3.1 is very similar to the form of collaborative filtering, wherein the input is users and items and the output is the reward. In the setting of the RL environment, implemented a similarity-based approach on the assumption that the user with similar interests will provide similar ratings to restaurants. There can be either unsupervised way of implementing the same using cosine similarity where the function 'f' is a weight-based state and action similarity as equation 3.2. The historic state, action and reward is formed with tuple data ((s(i), a(i) → r(i)). Based on the episodic task of the RL, when Agent provide recommendation a(t) for giving state s(t) for a given episode, the cosine similarity p(i) of the

state action pair is given by the equation 3.2, 'i' refers to state-action-reward from user review data.

$$\boldsymbol{cosine\ p(i) = \alpha \frac{s(t)s(i)}{\| s(t) \| \| s(i) \|} + (1-\alpha)\frac{a(t)a(i)}{\| a(t) \| \| a(i) \|}} \quad \textit{Equation 3.2}$$

Probability of assigning reward r(t):= r(i) is arg-max of p(i) over all the state action historic pairs of the user data. Generated reward from the user environment is an unsupervised methodology and uses a brute force technique to assign the reward; based on the historic interactions of state-action pairs a parameterized model can simulate the ratings. Then the direct r(t) can be given the below equation 3.3.

$$r(t) = f(s(t), a(t);\ \theta) \quad \textit{Equation 3.3}$$

The parameter θ can be a simple latent dimension in Matrix Factorization models trained on users and restaurant data by using RMSE error. Since data has to contextual information, a non-linear function approximation can also be adopted using Deep Neural Network architecture. In the case of DNN, parameters θ can be the weights of the network.

3.3.2 Learning Policy

The objective function for RL Agent is to maximize the cumulative rewards, formulating the learning policy with RL comes with basic 2 equations, one is action-value function and state action values (referred to as q-value) table 3.2. The objective function of the RL agent is to maximize the cumulative rewards over episodes. The reward from being in the state 's(t)' taking action 'a(t)' in episodic learning is given by equation 3.4 and 3.5. Where 'r' is the total expected reward from being in state s(t), followed policy pi. Where the π(a|s) is the probability of an action 'a' taken given state 's', p(s`, r | s, a) is the model of the environment that describes the state transition with some reward r.

$$v_\pi(s) = \sum a_\pi(a|s) q_\pi(s,a) \quad \textit{Equation 3.4}$$

$$q_\pi(s,a) = \sum s' \sum r p(s', r|s, a)(r + v_\pi(s')) \quad \textit{Equation 3.5}$$

In the recommendation process, the model of the environment is unknown; hence, methods such as Monte-Carlo, Temporal Difference and Q-Learning decide on optimal policy. Since all

the methods are tabular in nature and given the dimension of the state and actions, it will be impossible to calculate/tabulate all the state-action value pairs. Therefore, in this case, study the focus will be using Deep-Q Networks for finding an optimal policy. There are also DNN variants of classical TD method SARSA (Zhao et al., 2017). The difference between DQN and Deep-SARSA is the techniques followed to perform policy evaluation. In DQN, when the RL agent is playing episodes the state, action, reward, and next state goes into a buffer memory and later once the episode is completed a random batch is created from the experience buffer and Q-network is trained for the q value using the equation 3.6. The back prorogation happens in the network using MSE as the function with learning rate Equation 3.7. Similarly, in Deep-SARSA, the only difference is after each step in an episode the updated DQN network. This is on-policy learning and the latter is off-policy learning.

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma * maxaQ(s',a) - Q(s,a))] \quad \textit{Equation 3.6}$$

$$loss\ function\ =\ [Q(s,a) - (r + \gamma * maxaQ(s',a))]^2 \quad \textit{Equation 3.7}$$

In this study, the focus will be to perform a comparative study of the two techniques as which is better suited for restaurant recommendation using RL. Keeping Q-value architecture the same between DQN and Deep-SARSA. The architecture of DQN will follow the double DQN with a soft update (Zheng et al., 2018). Using a target network and a predictor network during the training process of an episode updates only the parameters of the predictor Q-network just as in DQN architecture. Do not update the parameters of the target network. Constantly shifting the predicted and target Q-values to update the network destabilize training by falling into feedback loops between the predicted and target Q-values.

The architecture of the DQN network needs to have an input layer of state and action, and an output layer of q-value. Since action space is large, the output layer training will be difficult the soft-max function and convergence of the RL solution will not happen. As shown in Figure 3.0.3, the state is the connection via the GRU layer to capture the t-1 and t time step of the same relations into evaluating the q-value. Later that state and action layers are concatenated and the fully connected layer will then output to a linear layer that is q-value. Each layer in the function uses a RELU activation function and weights follow uniform distribution initialization to achieve convergence on Q-Value.

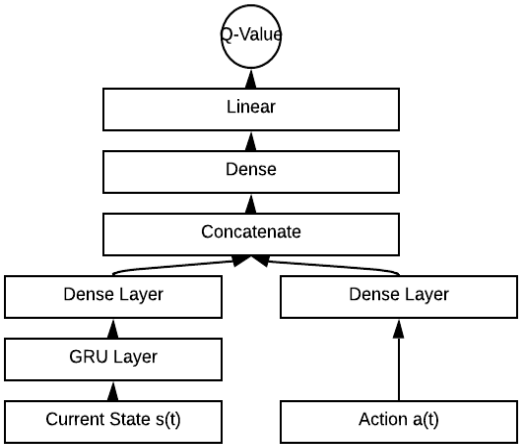


Figure 3.0.3 - Deep Q network architecture of state s(t) and action a(t) as input, then followed by GRU layer and concatenating the state, action and followed by a linear layer for q-value estimation.

The presented novel idea in this case study is the approach to reduce the large action space from the total count of unique restaurants to a predefined value so that the agents' policy will learn optimal recommendations. With the Epsilon-Greedy learning approach, RL Agent will perform exploration with decaying constant value and performs exploitation thereafter. Using embeddings of the contextual information of restaurants in the current state from the environment, in conjunction with CBF techniques from collaborative filtering to reduce the action space. Combining the above points into a Nearest Neighbour query on embedding's can provide more accurate and diversified recommendations to the user. By doing so, the recommendation policy will become deterministic in nature and hence small perturbations added to the current state and performed NN query to keep the recommendation policy quasi-deterministic in nature. Explained the algorithm as below Figure 3.0.4. The parameter ɵ is a hyper-parameter that defined the variance in the noise.

Algorithm 1 Get Nearest Neighbour Actions

```
function QueryNearestNeighbourActions(CurrentState)
    NNModel ←Load the Nearest Neighbour Model
    QueryActions ←Initialize empty list
    for State ∈ CurrentState do
        Noise ← GuassianNoise(μ = 0, σ = θ)
        ModifiedState ← Noise + CurrentState
        KActions ← NNModel(ModifiedState)
        KActions append to QueryActions
return QueryActions
```

Figure 3.0.4 – Showing the algorithm for selecting recommendations based on the current state

3.4 Evaluation Metrics

For recommender systems, the case of predicting a rating for the given user and items is evaluated using root mean square error (RMSE). Since this is a regression case, the cost

function will be to learn the difference between actual rating and predicting rating. Several model-based CF techniques use the RMSE metric for learning, such as MF, SVD and even DNN like wide and deep networks (Beel et al., 2016). Unlike the classical recommendation setting, the object of RL is to provide optimal recommendations to the user based on historic data as state RMSE and MSE metrics will perform the DQN network in this case. To capture the quality of recommendations used Mean Average Precision (MAP) and Normalized Discounted cumulative gain (nDCG). Precision in recommender systems is the ratio of "recommendations that are relevant" by "total recommendations given". Precision @N is a subset of recommendations from the initial time to the N^{th} element. In RL episodic learning task, each episode is a sequence in time of length N, where N is set of {1….t}.At each time step t recommendation provided to the user will be hit or miss based on the assumption that the user provides a rating of 3 and above is good else the recommendation is bad. Average precision is mean of the Average precision values across episodes constitute for MAP as defined in equation 3.8. The implementation of the MAP follows the details provided by (Turpin and Scholer, 2006).

$$MAP@N = \frac{1}{U}\sum\frac{1}{N}\sum_{t=1}^{N} P(t)\cdot rel(t) \qquad \textit{Equation 3.8}$$

Cumulative gain is the total sum of rewards received by the RL agent from a complete episode of recommendation (Järvelin and Kekäläinen, 2002). Discounted cumulative gain (DCG) measures the effectiveness of the recommendation with time position t in the episodic task. If the RL Agent learns a good recommendation strategy, DCG number per episode from the starting of the episode is a good indicator. Equation 3.9 gives the DCG. Normalized Discounted cumulative gain (nDCG) is the ration of the DCG and Ideal discounted cumulative gain (IDCG). In this case study, the best rating user can give to a restaurant is '5', hence for episodic length N the IDCG value of uniform vector of rating '5'.

$$\mathrm{DCG} = \sum_{t=1}^{N}\frac{rating_t}{log_2(t+1)} = rating_1 + \sum_{t=2}^{N}\frac{rating_t}{log_2(t+1)} \qquad \textit{Equation 3.9}$$

$$\boldsymbol{nDCG = \frac{DCG}{IDCG}} \qquad \textit{Equation 3.10}$$

3.5 Tools

In the following subsections, explained the various tools used in this case study.

3.5.1 Python

Python is a high-level programming language written for fast prototyping. The 3.7 version of python used for writing the proposed methodology. Python supports both an object-oriented programming style and a functional programming style for playing with data. Python provides an interface for configuration files as .ini extensions, which are useful for configuring multiple hyper-parameter settings for different use cases. In this study, the open-source version Anaconda community package used as the distribution contains all the essential packages that require to get started on developing an application using python.

3.5.2 Pandas & NumPy

Pandas is a data processing package, where the flat file formats such as CSV and TSV or even from MS-Excel files read as tables. The building blocks of the Pandas package are Series and Data frames. Some of the key features of pandas are abstractions to functions that used in pre-processing of data, representation of columns as a nominal, ordinal or binary variable. Representation of column as one-hot encoding columns. Apart from this, pandas generally provide SQL as functionalities, for example, join merges and selection transformations. Pandas recently have also included plotting and statistical functions that are very helpful for data pre-processing and analysis.

NumPy is a numerical python, which provides an abstraction on top of CPython. Python has Gil (Global Interpreter lock) limitation that provides a bottle next for mathematical operations such as vectorized matrix dot products and solving parameters for quadratic functions. Hence, there is a compiler for python on C language, which is faster for numerical processing and NumPy packages provide a large variety of mathematical functions starting from the mean, mode, variance to complex function such as random number generation from Gaussian distribution.

3.5.3 Keras

Keras is a deep learning library in python that provides various abstractions for quick prototyping of deep neural networks. There are two bases if Keras, one is TensorFlow and the second is Theano. In this study, the default backend configured by Keras is the Tensorflow package. Tensorflow is a mathematical or numerical processing library created by Google. Built the DQN networks, Auto-Encoders explained in the proposed methodology using Keras package.

CHAPTER 4

IMPLEMENTATION

4.1 Introduction

This section, starting with a detailed explanation of exploratory data analysis (EDA) of Yelp's data set. Then steps performed for data clean up and pre-processing, also a review of all the columns that are represented on contextual information of users and restaurants. Going forward, a discussion on the approach followed for dimensionality reduction techniques such as Auto-Encoders and PCA. Finally, how the data transform as a tuple of <State, Action, Reward> that in turn fuels training and testing RL framework. Next, sections will have a detailed layout of methods and implementation details of Models followed for estimating Q-Value [Q(s, a)], what are the different analogies and techniques followed from Deep Learning so that the loss function minimization happens. Later, a section is detailed Training and testing procedure is explained which follows off-policy or on-policy with DQN and Deep-SARSA. Finally, a summary of the complete implementation details ends the chapter.

4.2 Yelp's Dataset

In this study, Yelp Dataset Challenge from Kaggle (Yelp Inc, 2020) will be under the purview. The dataset contains four different files

1) User reviews data: this file has all the details of the user-provided ratings to the business. A time column indicates when the user-provided ratings to a restaurant in Date-Time format. The review has text that is very helpful for sentimental analysis.
2) Business data: The business data contains all the attributes of the restaurant, starting from latitude, longitude to contextual features such as categories and cuisine type.
3) User data: this has very interesting data of what all compliments that user-provided like, cool, hot, etc. This data is having social connections like friends with other users list. This is very helpful in case social context-based learning needs implementation.
4) Tip data: this file contains data in which the user provides a tip as feedback to the business and on what date and time. This helps to capture explicit feedback from the user to the business.

To create the tuple if <State, Action, Reward> for training and testing purposes. The study aims to minimize data elements that are required are user contextual data, business contextual data

and the user to business ratings/reviews data. By taking these files and pre-processing them, to obtain respective destination datasets as follows.

4.2.1 Business Data

The business data is the widest data set with many attributes present in JSON format. Initial processing performed formatted the business data from JSON to CSV with keeping all the attributes data intact. This intermediary file is further then processed for feature engineering using the Pandas package in python. The file contains around 74587 rows that are unique business-IDs from Yelp's dataset. One of the columns describes the name of the establishment and other categories, which had different categories in each business. Filtering the data based on keywords such as "Restaurant" and "food" yielded a total count of 19590 businesses that form restaurant data for the recommendation. In the next steps, the contextual information extracted from the rows and pivoted to columns that provided 51 contextual raw feature space. Table 4.1 are some examples from 51 columns and their data types. Some of the interesting columns from the dataset are Latitude and Longitude that represent the position of the restaurant/establishment. The majority of the data contains from US and CANADA countries and the states that covered in the dataset are [ON, AZ, NV, OH, NC, QC, PA, AB]. Most of the data come from the top three states Ontario from Canada and Arizona and Nevada from the USA. The missing values from the data are first replaced with the keyword 'none'. Later while imputing the missing values the 'none' keyword in case of Binary variable is translated to 'false'. However, in the case of ordinal variables, the 'none' keyword is given a value of zero which forms a low-level category in the order of the ordinal values.

Column Name	Data Type	Unique Values	Representation
Alcohol	Nominal Strings	['full_bar','none','beer_and_wine']	`[2 0 1]`
Restaurants Attire	Nominal Strings	['casual' 'none' 'formal' 'dressy']	`[3 0 2 1]`
romantic	Boolean	['False' 'none' 'True']	`[0 1]`
Wi-Fi	Nominal Strings	['no' 'none' 'free' 'paid']	`[0 1 2]`
Restaurants Take Out	Boolean	['true' 'false' 'none']	`[1 0]`
Good For Kids	Boolean	['true' 'none' 'false']	`[1 0]`
late-night	Boolean	['False' 'none' 'True']	`[0 1]`

Table 4.1 - Sample contextual features from the restaurant's data. Shows some attributes, their data types and how pre-processing of the data type.

4.2.2 User Review Data

The user reviews data file consists of Business-Id, User-Id, Stars and Date-Time columns. The data for reviews joins with business data and user data to add contextual information related to user and business to review data. This will help in forming the initial data set on which this case study will focus on and further process to extract training, validation and test sets. On performing univariate analysis on the Stars/Rating columns show that, the majority of the ratings are for number '5' as shown in Figure 4.0.1. It so happens that users show a long tail distribution for the count of ratings provided to a business. As insights provided by this data, some users have provided just a one-time rating to a business and some users have provided 500 reviews to all businesses.

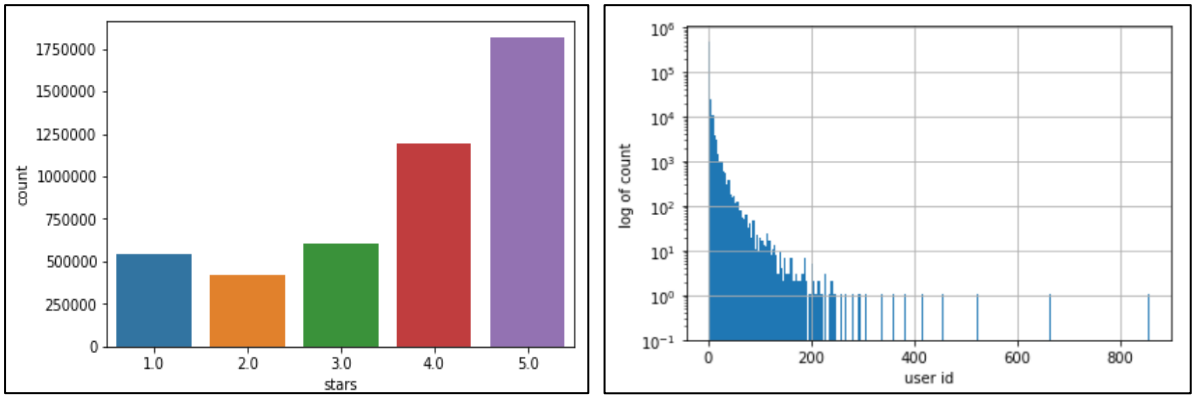


Figure 4.0.1 – Left: Showing the count plot for the start given by all users across the yelp dataset. Right: showing the long tail distribution of the count of reviews given by user

Since the assumption in the study for state represented in RL framework is to have historic ratings of restaurants/business of the user, the reviews count of the user will have lower bound of 10 and upper bound of 300 to have a good sample of users and business information gathered for training and testing. Extracted Business-Id's from Business data based on restaurant key words filtered from user review data. For performing a sanity check as weather the data of the reviews truly, give meaning for a recommendation, a simple MF model applied on the data and MAP calculated on top of it to verify this assumption. The MF model implemented using Keras Python package in which the input is the sparse matrices of user and review data. A simple pre-processing of the user-id and business-id to an embedding of real-valued number is performed and then a big sparse matrix is created in which the user represented by an index of the rows and columns identified as unique indices of the restaurants. The cells in the matrix have ratings provided by a user to a particular restaurant. Training of the MF model using SGD with a learning rate of 0.05, a total of 10 epochs was run and the RMSE is the metric used for calculating the regression error. At the end of the training-loss is 1.2706 and validation loss is 1.5081.

4.3 Data Pre-Processing

The initial steps involved in data pre-processing are reading of Yelp's Business data, filtered only for restaurants and User to Restaurants reviews data. The two files are loaded into memory, the shape of the user data is tuple value of (1324324, 4) where first represents total rows and the second value represents total columns. The columns of user data are User-Id, Business-Id, Stars and Date-Time from Figure3.1. Similarly, restaurant embedding's are loaded into memory too, and the shape of the data is (19590, 1034). The first value in the tuple is 19590 is the total count of restaurants, and 1034 is one hot encoding representation of all the contextual attributes that represent a business. Python's Pandas package reads the aforementioned data from CSV files. One hot encoded feature of business data is converted to latent features representation using techniques such as PCA and Auto-encoder. Observing the data of the business set, real-valued independent variables are latitude, longitude and hours of opening. Just by using, these independent variables will not be sufficient and PCA with one-hot encoded features does not capture the variance in data. Hence, in this study, the focus will be on building an autoencoder network using Keras package. The architecture of the network consists of input later, equal to 1034 dimension and then a hidden layer that represents features in latent space whose dimension is 10. The output layer follows a sigmoid function to map back to the input given. The network trained using Adam Optimizer from Keras with a learning rate of 0.01. After training and running the Auto-Encoder on business data, saved data used for further to create state, action and reward data points for the RL framework. Shown the sample data from the embeddings in table 4.2 below.

Business-Id	0	1	2	3	4	5	6	7	8	9
QXAEGFB4oINsVuTFxEYKFQ	4.411849	4.738783	9.329696	13.22289	1.758909	4.30665	5.077082	0.568811	0	6.865017
1Dfx3zM-rW4n-31KeC8sJg	2.127881	4.58382	8.213978	2.47098	3.485499	4.024045	3.452238	6.611355	0.600296	2.409309
1RHY4K3BD22FK7Cfftn8Mg	4.881931	0.23461	0.276599	6.049745	6.167278	2.42805	4.997796	6.372932	3.795377	3.091788
tstimHoMcYbkSC4eBA1wEg	3.093885	3.616591	2.917479	3.739008	9.100592	3.093931	5.404335	0.905019	0.892846	6.37306

Table 4.2 - Sample latent features of restaurant contextual information

Next steps involved are to create state column for each user, the following steps will elaborate on the process:

1. Initially read user reviews data, group the data by user and sort using date-time that will then create a base data set for user mapped to all the restaurants reviewed.
2. For each user in the data set, follow a 'K' window approach to scan and get all the business-ids from review data. In the current experimentation settings, the K size set to

number 5, as this number will give sufficient insight to user historic restaurants. It also will keep the state space confined not to use more memory.

3. The next item 'K+1' will be action taken by the user and this index will give the business id from user review data. The corresponding stars or ratings given by the user to this business will become the reward.
4. Collecting the state, action, and reward from the earlier steps will form the RL framework training and testing data. Now, How to split the data for training and testing?
5. Considering a session of the user, in which rows of state, action and reward tuple for a single transaction. First 80% of the total number of rows are, considered for training and the rest 20% of the rows are, considered for testing.

As per the above approach, the final data will have structure as mentioned in table 4.3 below. Additional reference section provided for sample code snippets for actual implementation details of code in python.

State	array([[-21.18446 , 0.1743042 , -1.2954022 , 0.25401723, 0.28795096, -0.9985907], [-21.18446 , 0.1743042 , -1.2954022 , 0.25401723, 0.28795096, -0.9985907], [-21.18446 , 0.1743042 , -1.2954022 , 0.25401723, 0.28795096, -0.9985907], [-21.18446 , 0.1743042 , -1.2954022 , 0.25401723, 0.28795096, -0.9985907], [-21.18446 , 0.1743042 , -1.2954022 , 0.25401723, 0.28795096, -0.9985907]], dtype=float32)
Action	array([-21.18446 , 0.1743042 , -1.2954022 , 0.25401723, 0.28795096, -0.99859065], dtype=float32)
Reward	5.0

Table 4.3 - A sample row of state action and reward from the RL training data set.

4.6 Model Implementation

Model implementation divided into three major segments; the building of the environment, building of agents and the complete training and testing framework that includes environment and agents. The implementation follows a classical Object-Oriented Programming approach that will help perform several different kinds of classes that can correspond to experimentation settings. This allows capturing each experimental setting as a python object/class and uses a configuration or enumeration driven approach to perform several types of experimentation. The implementation comes with several configurations values starting from parameters that are required for the RL framework as well as for DQN networks hyper-parameters. Tabulated the list in table 4.4.

Configuration Parameter	Nomenclature	Value	Comments
State-space	K	5	The historic state for each user, the dimension of state space in the RL framework
Max Query Size	N	200	Total number of nearest neighbors to query while performing action space reduction

User sample size	-	500	Total number of users based on which training and testing of RL recommender Agent
Episodes	-	1000000	Total number of epochs to train the Agent for the recommendation
Episode length	-	20	Length of each episode that defines now many recommendation sessions to be made between Environment and agent
Similarity Alpha	α	0.5	Weightage for similarity check on recommended action to existing user's state action tuple.
Epsilon Min	Min(ε)	0.01	A minimum value of epsilon to maintain while performing an epsilon greedy strategy.
Epsilon Max	Max(ε)	1	Starting value for epsilon greedy strategy for exploration and exploitation dilemma
Epsilon Decay	ε	0.000059	The decaying factor for epsilon value for episodic progression. This signifies exploratory vs exploitation nature of Agent while training
Discount	γ	0.8	The discount factor to consider for future rewards from the Q-Value equation.
Learning rate	β	0.02	Learning rate for Deep Q-Network architecture for Q-value prediction.
Training batch size	B	128	For off-policy learning type in RL, after the episode is completed a sample batch size that is required to be picked from memory buffer for training DQN
Gaussian noise variance	ɵ	0.05	While performing action space reduction, this signifies the amount of perturbation to add with the current state before querying the nearest neighbors.

Table 4.4 - Different parameters used in the RL framework and their sample values with the significance of this parameter

4.6.1 Building Environment

Environment class takes initializes with the user's historic data of the states, action, and rewards. Also, a Boolean flag used to indicate whether the training or testing process happens with the environment. Initialized the episode length to a respective value considering long and short-term recommendation strategies. The significance of this variable is to determine the end of the episode that will signal to RL Agent, and in turn, RL Agent can perform off-policy or on-policy learning on the recommendation and rewards based in current states. To help with the process of RL, three main methods created are; 'reset', 'step' and 'is episode done'. The first method reset will randomly pick a single user from the sampled user list initialized with the RL Environment. For this picked user current state is set as the first state of shape (5, 10), created earlier by the pre-processing data flow. Once the current state is sent to RL Agent, Environment will need to take in recommended action from Agent and provide a reward back along with moving from current state-s(t) to next state-s(t+1). This will be the responsibility of the 'step' function, the recommended action-a(t) is compared with the current user's all <state, action> historic tuples. As this is O(n) complexity algorithm, the historic state-action pairs are

grouped by rewards and average states and average actions are calculated. This has reduced the complexity of the algorithm from O(n) to O(5) as currently the rewards are set of 1 to 5 ratings. Based on the cosine similarity score, take a maximum of normalized values of the cosine similarity across all ratings will provide the actual rating of the user. This becomes a reward value. If the rating provided by the reward, the function is greater than or equal to 3 then action recommended by RL agent will be appended to the end of the current state and the earliest restaurant embedding from the state space is popped out. This becomes the current state again and the cycle continues until the end of the episode reaches.

4.6.2 Building Agent

The Agent has several methods that perform individual functionality, as one of the objectives is to find out the optimal recommendation policy for several Agents, but all the agents have some common functions that will enable them to integrate several policies at once for a given training data and compare the results. The two major RL agent policies implemented are Deep-SARSA and Soft update DQN. The "get action" function of the Agent is responsible for the recommendation of the restaurant to the User Simulation environment; also, it should be generic enough to support any online traffic. The "get action" function will use epsilon parameters to perform exploration epsilon times and 1-epsilon times exploitation. While recommending action from large action space, the current state modified by adding small perturbation becomes a query state for the Nearest Neighbour model. After querying K actions, if the epsilon value falls under exploration criteria, then random choice made on these K actions becomes a recommended action. Else, for all K actions, the maximum Q-value predicted from the DQN network becomes the recommended action. As and when the RL agent receives rewards and next state, the data pushed into memory buffer then used for training the RL Agent.

4.6.3 Training RL Agent

As an illustration of the proposed methodology of the RL training framework, there are two major stages in the process. One is a state-transition generation process and the second is the training process. State transition will start with state zero S(0) that is revived by calling the 'reset' function of user environment simulation. Based on this state RL agent will recommend an action a(t) according to Figure 4.3 Then the agent will observe reward r(t) for the action taken from the user simulator and updates the state s(t) to the next state s(t+1). In the end, the main function will update the RL Agent memory buffer with <state, action, reward, next state> data for the length of one episode from User Simulation functions and RL Agent's actions.

Apart from this, the epsilon value of the training process will decay with a factor of ε. Until now, the User environment and RL Agent have performed an episode and collected the data for training. Next, the training of the RL Agent will happen which whose target is to maximize the cumulative reward from an episode. In this study, two approaches Soft update DQN and Deep-SARSA compared to see convergence and performance of these policies.

For training the DQN network, once the episode is completed the RL Agent will sample a mini-batch of transactions from the replay memory buffer. In this DQN network, there are networks namely target DQN network and predictor DQN network. While performing the episodes the predictor DQN network is used for approximating the Q-value provided s(t) and a(t). In the training procedure, the next state s(t+1) is used to get the next action a(t+1) based on target DQN. Using the next state s(t+1) and next action a(t+1) the Q-value for next state action is predicted from target DQN, which is used to update the Q-value of the predictor DQN network based on Equation 3.6. Predictor DQN parameters are updated with Q-value derived from Equation 3.7 for input state s(t) and action a(t). After the predictor DQN is updated, the parameters of this are copied to target DQN with a weightage of τ=0.05. This will help smooth the learning process of DQN and avoid the divergence of parameters. This process continues to the end of the epochs. After this, run the test set with the same RL framework using the trained DQN network and estimated MAP and nDCG values for both training and testing sets. In the aforementioned process, RL Agent is the learning process need to converge and DQN loss and Q-values captured by the user and by episodes.

Similarly, for the Deep-SARSA process, learning will happen on-policy which means there will be no experience replace buffer present in this process. For state s(t) RL Agent receives a reward r(t) for taking action a(t). Using these details, a Deep Neural Network - DQN similar to architecture as Figure 3.2 will estimate the Q-value at time t, and the next state s(t+1) is used to get an action a(t+1) using the algorithm from Figure 4.0.2. In the same episodic task, the DQN trained with the Q-value and then for the next task in an episode, updated DQN used for selections based on Q-value. Compared to soft update DQN for Deep-SARSA there is no target network used as all the variations in Q-value by taking actions gets learned in the process.

Algorithm 1 Training Steps for Reinforcement Learning

```
function QueryNearestNeighbourActions(CurrentState)
    NNModel ←Load the Nearest Neighbour Model
    QueryActions ←Initialize empty list
    for State ∈ CurrentState do
        Noise ← GuassianNoise(μ = 0, σ = θ)
        ModifiedState ← Noise + CurrentState
        KActions ← NNModel(ModifiedState)
        KActions append to QueryActions
    return QueryActions

function GetRecommendation(CurrentState, I, DQN)
    QValueList ←Initialize empty list
    QueryAction ← QueryNearestNeighbourActions(CurrentState)
    for Action ∈ QueryActions do
        QValue ←DQN.Predict(CurrentState , QueryActions)
        QValue append to QValueList
    return Action corresponding to argmax(QValueList)

Initialize the K for nearest neighbour Query
Initialize Parameters tracking for Rewards, loss and q-value
Create User Environment Simulator with α parameter
Create DQN with learning rate β
Initialize buffer memory of D length
for epochs = 1, M do
    Reset action space of restaurants to I s(t) Initial state from user history
    for t = 1, T do
        Select an action a(t) from I using GetRecommendation(s(t))
        Execute action a(t) and observe reward r and next state s(t+1)
        Append s(t), a(t), r, s(t+1) to memory D
        s(t) = s(t+1)
        if policy = on-policy then
            get a(t+1) using GetRecommendation(s(t + 1))
            get Q(s(t+1), a(t+1) from DQN
            Q(s(t), a(t)) := r + discount-factor * Q(s(t+1), a(t+1))
            Update DQN parameters using Q(s(t), a(t))
    if policy = off-policy then
        sample mini-batch M from D
        for m = 1, M do
            get a(m+1) using GetRecommendation(s(m + 1))
            get Q(s(m+1), a(m+1) from DQN
            Q(s(m), a(m)) := r + γ * Q(s(m+1), a(m+1))
            Update DQN parameters using Q(s(m), a(m))
```

Figure 4.0.2 - Showing the complete RL learning steps

CHAPTER 5

RESULTS AND EVALUATION

5.1 Introduction

Following the implementation, results, and evaluation of the RL framework discussed in this chapter. The initial discussion will be a total runtime of the training algorithm as the action space is large. Next, the proposed nearest neighbor action search will be evaluated for the effectiveness and various experimental settings created to evaluate the RL framework. Deep-SARSA and DQN technique's observations discussed along with comparisons. Finally the effects of tuning the noise parameter.

5.2 Experimental Settings

The evaluation of the data in consideration is of Yelp's 2018 dataset from Kaggle. As per the process explained in section 3.2, initially all the users selected with the count of reviews provided to restaurants to be greater than 10 and less than 300, (15662) users are selected for experimentation. As per the initial analysis, the total action space or the number of restaurants to recommend is 19590. RL framework takes a lot of time to train. Hence in this study, the experimentation starts by randomly picking users from across 15662 users. The selected users provide ratings to restaurants from the action space. This is a validation step of data correctness based on the two experimentation settings used for the current case study on users and the number of epochs. Reference for the settings in table 5.1 defined all the required parameters. From the cross-validation procedure of the training data, derived the value of theta for experimentation. Some of the specific settings of user simulation are the nature of rewards. Based on user historic ratings the set of ratings that rewards function provides are {1, 2, 3, 4, 5}. In this study, positive ratings are Three, Four and Five and not good ratings are one and two. RL Agent's recommendation will be only one restaurant considering the complexity in calculating the reward function and one restaurant will be good enough to learn the user preferences. Next is the selection of the epsilon greedy decay value, this is a manual step that is performed by plotting a total number of epochs vs the decaying epsilon value function. In the current experimentation, to check the convergence (Figure 5.0.1), hyper-parameter setting and RL framework correctness, a user set of 15 selected. To check if the solution works for larger data, a user set of 1000 selected for the second experimentation. To evaluate the performance of the RL Agent, in this study MAP and nDCG as the chosen metrics.

Parameter	Nomenclature	Experiment-1	Experiment-2	Experiment-3
Max Query Size	N	100	150	200
User sample size	U	15	100	1000
Episodes	E	4000	20000	150000
Episode length	e	20	20	20
Similarity Alpha	α	0.5	0.5	0.5
Epsilon Min	Min(ε)	0.01	0.01	0.01
Epsilon Max	Max(ε)	1	1	1
Epsilon Decay	ε	0.0017	0.00029	0.000059
Discount	γ	0.8	0.8	0.8
Learning rate	β	0.01	0.02	0.02
noise variance	ɵ	0.05	0.05	0.05

Table 5.1 – Explaining the different experimental settings for the RL framework and training RL Agent.

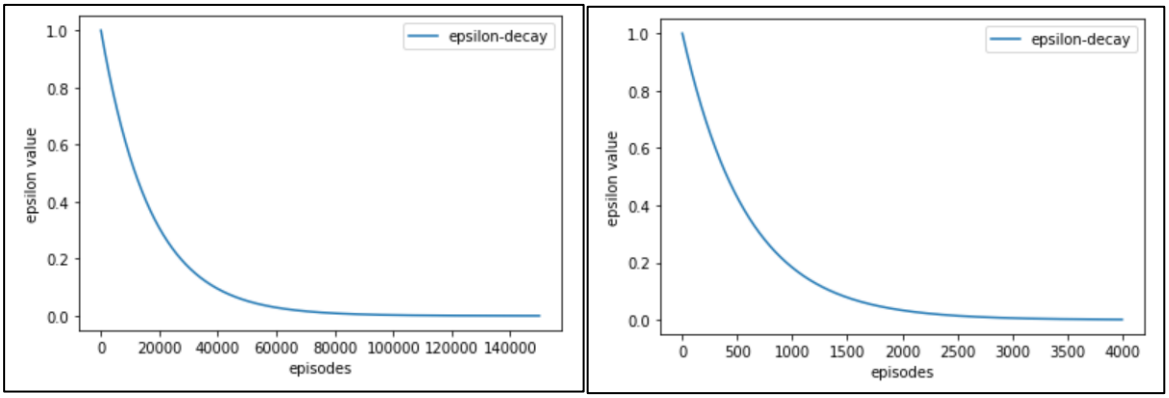


Figure 5.0.1 – Epsilon decay graphs for deriving epsilon greedy policy. Left is for 150K epochs and Right is for 4K

5.3 Observation and Results

The initial challenges faced while training the RL framework was the usage of brute force for loop on all restaurants (action space) and finding out Q-Value for recommendation, as the total number of episodes performed increased the total time taken by the RL framework to train even on Experiment-I settings was very much high as 1200 seconds per episode. As this was very time consuming, the nearest neighbor approach of content-based filtering not only provided a novel way to select restaurants from large action space, it also reduced the total time taken by the RL framework to 1.3 seconds per episode. This greatly increased the speed of training. The second challenge of the case study, it to achieve the convergence of the RL Agent learning. Convergence is evaluated either from loss of the Deep Neural Network used by RL Agent or by Q-value from the network. In these experiments, the RL framework code developed to capture the Q-value and rewards from both on user level and episodic level. This data plotted to see the convergence graphs are in the upcoming sections. The script automates to capture the precision per episode and average precision for all episodes. The mean of average precision

across all users provides MAP value. Similarly, ratings provided by User Environment Simulator captured to calculate nDCG values.

5.3.1 Deep-SARSA

Deep-SARSA implementation follows on-policy learning where current DQN used to predict the q-values for taking action (recommendation of the restaurant). Once the RL agent receives a reward and next state, again the current DQN used to predict the next action. Based on this <State, Action, Reward, Next-State, Next-Action> tuple, the DQN network is trained to minimize the error using the q-value equation 3.7. As this is a very quick process, only time taking component is a reward function based on similarity score. Secondly, the Nearest Neighbour query of ~100 actions from the action space of 19590 restaurants. Each episode of 20 recommendation steps takes around 1.5 seconds for Deep-SARSA on-policy learning algorithm.

1) DQN loss tracking: Starting with Experiment-I, the total number of users is 15 and epochs are 4000, on running training it is observed that the loss function converges but there are some small spikes seen in loss function where it is observed the model is still trying to learn the rewards of the user. Overall, we observed the convergence of loss function. In Experiment-2 and Experiment-3 where the users are comparatively more, the tuned learning rate and query size of Nearest Neighbours showed better results.

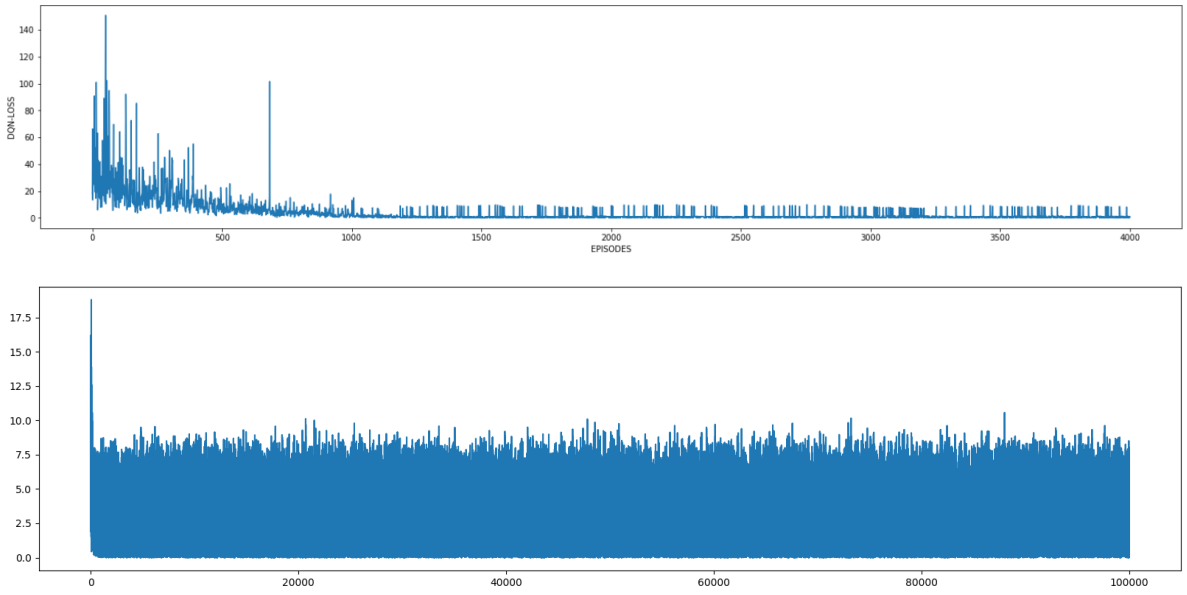


Figure 5.0.2 – Figure showing the MSE loss of DQN network used in Deep-SARSA, TOP for 4K Epochs and Bottom for 150K Epochs

2) Q-value Convergence: Q-value is the output of DQN's prediction value given state s(t) and action a(t). This output will evaluate what action selected for recommendation.

Hence, the convergence of this value observed for experiment settings. Initial the Q-value fluctuations signify the RL Agent performing exploration for epsilon greedy policy, as the episode progresses and RL Agent performs exploitation; the variance in Q-value reduces. Experiment-1 showed better convergence of Q-value as the user base was less.

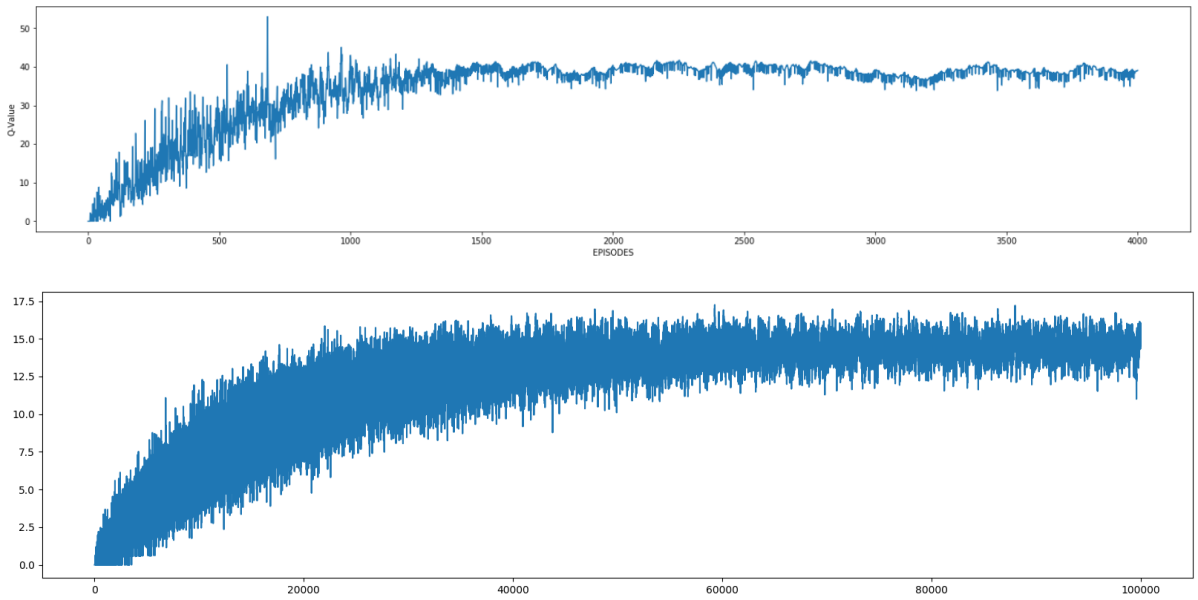


Figure 5.0.3 – Q-value convergence for the Deep Network used in Deep-SARSA RL Agent learning process, TOP for 4K epochs and bottom for 150K epochs

Rewards Convergence: This metric captured at the user level, as random user selection happens for each episode while learning. Gradually for epoch's progression, total rewards per each user should converge. The total rewards provided by the user simulation module per episode captures total rewards per episode for a user. Also, AP (average precision) and nDCG values show this as performance metrics.

5.3.2 Soft update DQN

Soft update DQN follows off-policy learning, once an episode is completed a memory buffer will be used to sample mini-batch of <state, action, reward, next state> data. Q-value loss function for current state and action, updates predictor DQN parameters based upon the discount factor, next state's q-value derived from target DQN. The mini-batch size also plays an important role here as the size of the mini-batch signifies the diversity of the learning data to the DQN network. The initial prototyping performed on a batch size of 32, 64 and 128. The mini-batch size of 128 showed optimal results and the same used for Experiment settings. The time taken for DQN is comparatively longer than Deep-SARSA as after each episode the training happens for mini-batch size whereas for latter it's just one record update to the DQN.

With this time taking aspect of training comes with the advantage of yielding good results, as deep-SARSA tends to give an overfit estimator of Q-value compared to soft update DQN, as it learns generalization from the mini-batch concept.

1) DQN loss tracking: The loss value tracking for DQN shows less variance in all experiment settings. This is because of the advantage of using mini-batch training with off policy methodology. Overall observed the convergence of the training loss for the predictor DQN.

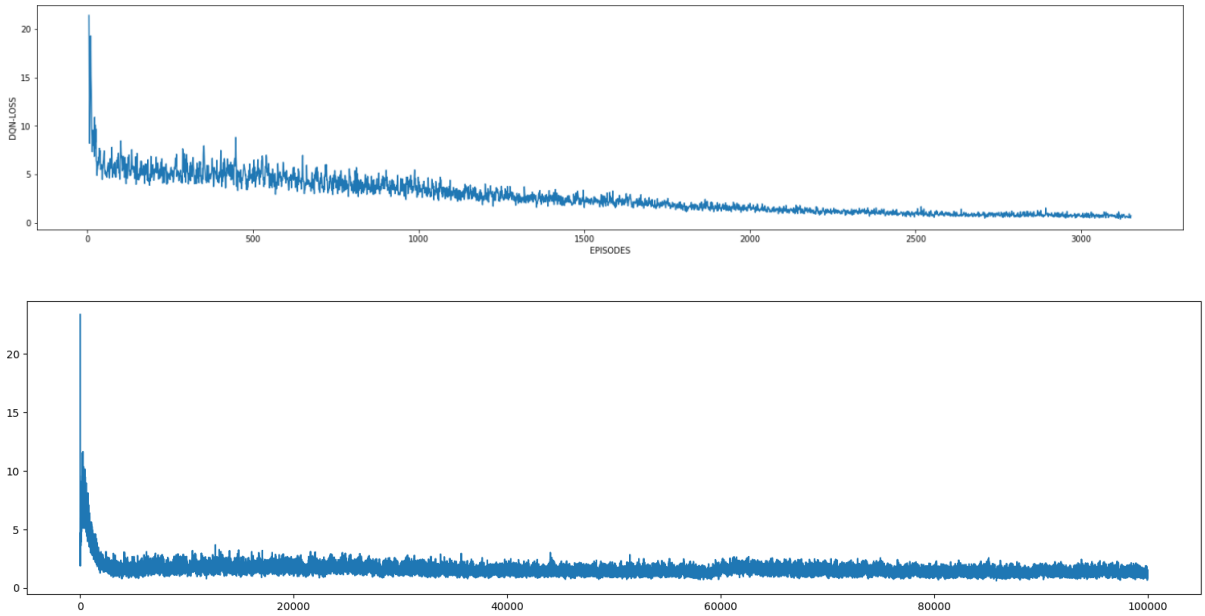
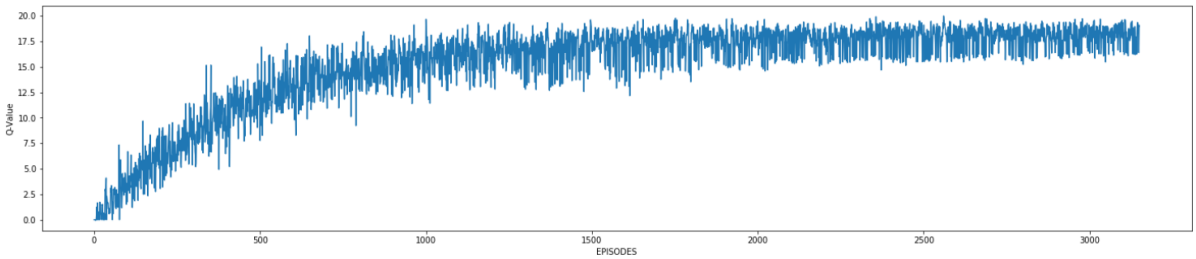


Figure 5.0.4 – Loss graphs for Deep Network using in soft update DQN, Top for 4k epochs and Bottom for 150K.

2) Q-value convergence: Q-value output for the DQN follows the approach of predicting q-value based on current state and action. The architectures of DQN are the same for Deep-SARSA and soft update DQN. There is some variance in the final stages of training along with the convergence of Q-value for epochs. This is because; using mini-batch training focuses on training DQN generally on sampled data. This implies that the DQN is performing generic in nature and is not over-fitted.



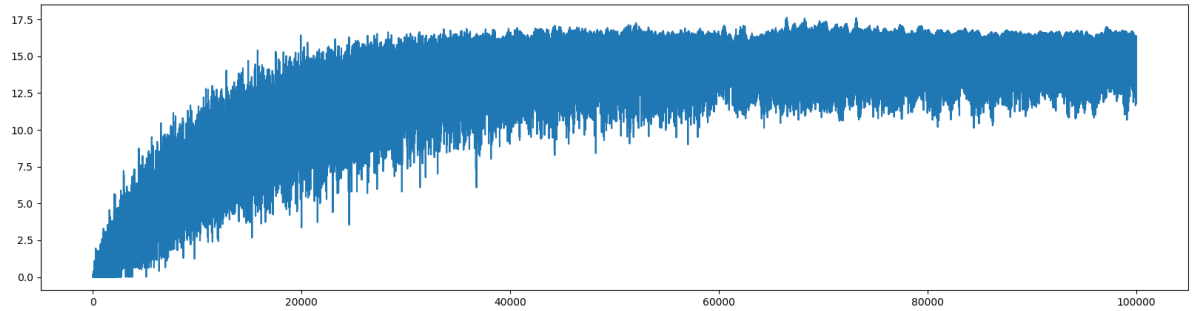


Figure 5.0.5 – Q-value convergence graphs for Deep Network used in Soft update DQN, Top for 4K epochs and bottom for 150K.

3) Rewards convergence: The total rewards per episode captured from the user simulation module for soft update DQN are a metric for rewards convergence. For all the experiments in the soft update DQN, the plot created per user with the y-axis being total rewards and x-axis being several epochs that happened for a particular user. The plots for the majority of the users are converged. Also, the metrics of MAP and nDCG capture the performance of the recommendations from the RL Agent. Observation from the histogram plots of MAP and nDCG by the count of users versus accuracy ranges is that the majority of the users all under the performance bucket of 0.7 to 1.

5.3.3 Comparison Of Results

The comparison of learning policies based on Deep-SARSA and soft update DQN tabulated as follows. There are factors of comparison of these two policies, come of the attributes of the RL learning framework affect the learning policy directly such as the total number of restaurants to query from action space using Nearest Neighbour. Secondly, the user environment simulation is an α parameter that provides weight for the importance of state and action similarities. These attributes discussed in detail in Section 5.3.4, but for the scope of this section for learning policies compared based on convergence graphs of Q-values and DQN learning loss. Also, the performance comparison using MAP and nDCG.

1) The graphs of Q-value convergence of Deep-SARSA are sharper than soft update DQN; the reason for this is the methodology of how these two learning policies implemented. Deep-SARSA follows on-policy learning while the latter uses off-policy. In on-policy, learning for each recommendation made updates the DQN parameters, this will tend to over-fit the network. Soft update DQN follows off-policy training; each episode of recommendation's data of state, action, reward, and next state goes into buffered memory. After the episodic task, the buffered memory accessed and a mini-batch sampled to train the DQN. Moreover, a hyper-parameter tau controls the divergence of

the DQN network. Hence, the Q-value convergence graphs for soft update DQN will contain a little bit of variance compared to Deep-SARSA.

2) The loss function graphs for Deep-SARSA also show that the DQN network is trying to learn but is having a large variance of loss, as its one network with on-policy learning for all data points of recommendations based on state and action. Unlike soft update DQN, loss variance is very less.

3) The performance metrics such as MAP and nDCG are better for soft update DQN compared to Deep-SARSA. The only advantage of the deep- SARSA is faster at run time for training compared to Deep Q-Learning.

Serial Number	Learning Policy	MAP-Train	MAP-Test	nDCG-Train	nDCG-Test
Experiment-1	Deep-SARSA	70.13	78.10	71.44	74.69
	SU-DQN	78.04	88.45	79.72	87.35
Experiment-2	Deep-SARSA	75.54	81.70	73.28	74.98
	SU-DQN	77.69	87.08	75.31	78.18
Experiment-3	Deep-SARSA	76.13	81.50	73.31	75.48
	SU-DQN	75.60	81.87	73.91	76.02

Table 5.0.2 – Comparing the results from different experiments performed

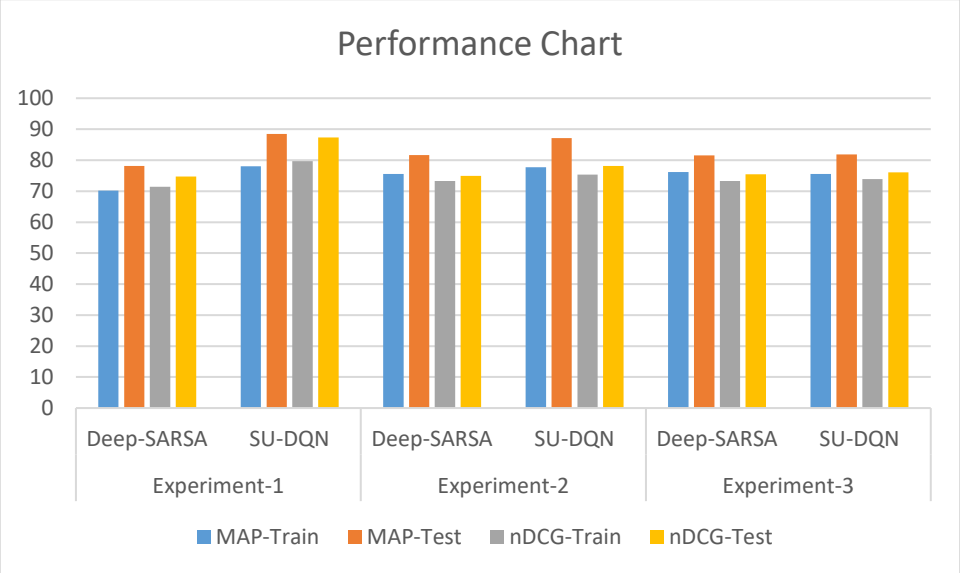


Figure 5.0.6 – Graph showing the experimental comparisons of MAP and nDCG for experiments

5.3.4 Effects of Hyper-Parameters

Some of the tuned hyper-parameters considered to get optimal results for the RL framework are the learning rate of the DQN and the variance of the Gaussian noise. The first discussion will be on the learning rate, initial the experimentation was with a learning rate of 0.01 and with that, many variances observed in the network loss. Moreover, to tackle the lot of variances in training process techniques such as batch normalization or drop out proved to be efficient, but only to an extent. The learning rate of the DQN proved to be more efficient in controlling the convergence of the Q-value in addition to minimizing the variance in the loss function of the DQN. Secondly, the variance theta from the mean zero Gaussian noise can be controlled to achieve good exploration, which in turn can increase the performance of the RL Agent. Experimentation for 15 users with 4000 epochs for both Deep-SARSA and Soft update DQN showed that around 0.05 value of the theta proved optimistic and yielded good results for the training and the test sets.

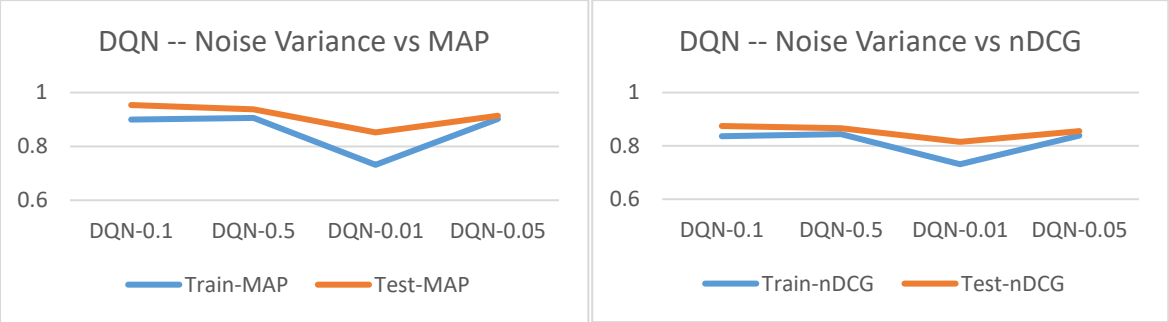


Figure 5.7 – Left shows the MAP value with respect to the noise variance for the NN-Epsilon Greedy policy. Right shows the nDCG value with respect to the noise variance hyper-parameter

5.4 Summary

In summary, the novel approach of using the CBF recommendation technique to reduce the action space for providing recommendations greatly contributed to the reduction in training time for RL's on-policy and off-policy learning methodologies. Implementation of additive mean zero and controlled variance Gaussian noise to the current state provided an improvement to the performance metrics of the RL Agent. This also helped to avoid the static nature of recommendation and provided a way to implement a quasi-deterministic model. From the perspective of RL Agent's learning algorithm, soft update DQN of Deep Q-Learning proved to be more effective than Deep-SARSA on Yelp's restaurant's data. Also, the learning algorithm provided some good numbers of MAP and nDCG for Yelp's data.

CHAPTER 6

CONCLUSION

6.1 Introduction

Considering the restaurant's recommendation domain and using Yelp's data set for recommendations, this case study started with the objective to formulate the problem of recommendation to sequential optimization problems. The idea of solving this as a sequential problem will capture the personalized user preferences and will try to utilize the historic reviewed restaurants for predicting the next recommendation. The literature survey reviewed classical recommendation systems like context-based filtering and collaborative filtering techniques. Then, advantages and disadvantages discussion happened on the classical approaches. Followed by a review of current work on restaurant recommendation showed the only usage of classical and hybrid recommendation systems for a restaurant recommendation. Later in the literature review, discussed Reinforcement Learning and its successful application in e-commerce, news article recommendations and concluded the chapter with the difference in supervised learning and reinforcement learning.

Next, I performed a detailed discussion of the proposed methodology of how the RL framework used to solve the sequential decision problem. Yelp's restaurant data were pre-processed using the contextual information to represent restaurants and represented the user review data as state, action, and reward. Using this data, this study proposed to tackle the challenges of RL on large action space by using the CBF technique of the Nearest Neighbour search. Also taking inspirations from deep learning in RL's Temporal Difference and Q-Learning methodologies, proposed the usage of DQN whose implementation followed Deep-SARSA and soft update DQN. Discussed the runtime of the algorithms and followed by convergence if the solutions, and performance of the RL Agent on training and testing data. Also, it discussed the significance of using noise on the current state of for providing recommendations as a technique to diversify recommendations to the user.

6.2 Contributions

The contribution of this case study for the recommendation of restaurants from Yelp's data set are:

1) Taking the inspiration from using RL for the recommendation in several e-commerce and production domains, the concepts applied to the restaurant's industry as well. This adds to the knowledge of how different e-commerce data compared to restaurants and techniques followed to apply the RL framework for restaurant recommendations.
2) Tackling the challenge of large action space, using CBF filtering techniques such as Nearest Neighbour search based on contextual information of the restaurants, achieved improvement in the quality of recommendations and training run time for epsilon greedy strategy.
3) By applying CBF techniques on the current state of the user, the recommendation becomes deterministic in nature. Hence in this study, the proposed methodology of using Noise as a parameter to convert the deterministic recommendation process to quasi-deterministic recommendation process. It also proved that tuning the variance of the noise controls the convergence of the RL Agent learning policy.
4) Successfully applied the Deep-SARSA and Deep Q-learning technique of soft update DQN for optimal recommendation policy.

6.3 Future Work

There are several aspects considered for future work on the current case study. Starting with data, Yelp's data set has review text data that effectively when incorporated into either user-environment simulation for considering sentiment analysis for rating a restaurant. Besides, the sentiment of the user to food can add one more dimension that represents user contextual information. In general, the existing work when extended to incorporate the sentimental and social relations between users from Yelp's data can make the RL learning framework consider all aspects of data and more personalized.

Secondly, the learning policies rely upon Deep Q learning and Temporal Difference techniques in RL. There is one more variety of RL action optimization using policy gradient algorithms. The objective of the Policy Gradient algorithm in RL is to optimize the action given state. No model is required for the environment in policy gradient too, but the actions tend to move towards maximizing the total reward accumulated by taking an action. The classical REINFORCE and Actor-Critic can capture non-linear interactions between current states of the user (restaurants rated by user) and provide the next recommendation items.

REFERENCES

Beel, J., Gipp, B., Langer, S. and Breitinger, C., (2016) Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 174, pp.305–338.

Beel, J., Gipp, B., Langer, S., and Breitinger, C., (2016) *Research-Paper Recommender Systems: A Literature Survey*.

Bouneffouf, D. and Rish, I., (2019) A Survey on Practical Applications of Multi-Armed and Contextual Bandits. [online] Available at: http://arxiv.org/abs/1904.10040 [Accessed 11 Jan. 2020].

Cheng, H.T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X. and Shah, H., (2016) Wide & deep learning for recommender systems. In: *ACM International Conference Proceeding Series*. Association for Computing Machinery, pp.7–10.

Choi, S., Ha, H., Hwang, U., Kim, C., Ha, J.-W. and Yoon, S., (2018) Reinforcement Learning based Recommender System using Biclustering Technique. [online] Available at: http://arxiv.org/abs/1801.05532 [Accessed 11 Jan. 2020].

Dulac-Arnold, G., Evans, R., Sunehag, P., Coppin, B. and Deepmind, G., (2015) *Reinforcement Learning in Large Discrete Action Spaces*.

Farooque, U., Khan, B., Junaid, A. bin and Gupta, A., (2014) Collaborative filtering based simple restaurant recommender. In: *2014 International Conference on Computing for Sustainable Global Development, INDIACom 2014*. IEEE Computer Society, pp.495–499.

Ganun, G., Kakodkar, Y. and Marian, A., (2013) Improving the quality of predictions using textual information in online user reviews. *Information Systems*, 381, pp.1–15.

Järvelin, K. and Kekäläinen, J., (2002) Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 204, pp.422–446.

Kumar, A., Tanwar, P., Nigam, S., and View All Authors, (2020) *Survey and evaluation of food recommendation systems and techniques - IEEE Conference Publication*. [online] Available at: https://ieeexplore.ieee.org/document/7724932 [Accessed 11 Jan. 2020].

Lee, J.-K., Oh, B.-H. and Yang, J.-H., (2012) A Reinforcement Learning Approach to Collaborative Filtering Considering Time-sequence of Ratings. *The KIPS Transactions: PartB*, 19B1, pp.31–36.

Li, L., Chu, W., Langford, J. and Schapire, R.E., (2010) A contextual-bandit approach to personalized news article recommendation. In: *Proceedings of the 19th International Conference on World Wide Web, WWW '10*. pp.661–670.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., (2013) *Playing Atari with Deep Reinforcement Learning*.

Mustofa, A.A. and Budi, I., (2018) Recommendation system based on item and user similarity on restaurants directory online. In: *2018 6th International Conference on Information and Communication Technology, ICoICT 2018*. Institute of Electrical and Electronics Engineers Inc., pp.70–74.

Quadrana, M. and Cremonesi, P., (2020) *RecSys 2018 – Tutorials – RecSys*. [online] Available at https://recsys.acm.org/recsys18/tutorials/#content-tab-1-4-tab [Accessed 18 Jan. 2020].

Seyednezhad, S.M.M., Cozart, K.N., Bowllan, J.A. and Smith, A.O., (2018) *A Review on Recommendation Systems: Context-aware to Social-based*.

Shani SHANIGU, G., Heckerman, D. and Brafman, R.I., (2005) *An MDP-Based Recommender System* *. [online] *Journal of Machine Learning Research*, Available at: www.mitos.co.il [Accessed 11 Jan. 2020].

Shi, Y., Zhao, Q., Wang, Y. and Cao, J., (2019) An Improved Restaurant Recommendation Algorithm Based on User's Multiple Features. In: *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*. IEEE Computer Society, pp.191–194.

Sutton, R.S. and Barto, A.G., (2015) *Reinforcement Learning: An Introduction Second edition, in progress*.

Turpin, A. and Scholer, F., (2006) *User Performance versus Precision Measures for Simple Search Tasks*. [online] Available at http://trec.nist.gov [Accessed 18 Jan. 2020].

Yelp Inc, (2020) *Yelp Dataset | Kaggle*. [online] Available at: https://www.kaggle.com/yelp-dataset/yelp-dataset [Accessed 18 Jan. 2020].

Zhang, S., Yao, L., Sun, A. and Tay, Y., (2019) *Deep learning based recommender system: A survey and new perspectives*. *ACM Computing Surveys*,

Zhao, D., Wang, H., Shao, K. and Zhu, Y., (2017) Deep reinforcement learning with experience replay based on SARSA. In: *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*. Institute of Electrical and Electronics Engineers Inc.

Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D. and Tang, J., (2019) *Deep Reinforcement Learning for Page-wise Recommendations*.

Zhao, X., Zhang, L., Xia, L., Ding, Z., Yin, D. and Tang, J., (2019) Deep Reinforcement Learning for List-wise Recommendations List-Wise Recommender System, Deep Reinforcement Learning, Actor-Critic, Online Environment Simulator. ACM Reference Format. [online] Available at: https://doi.org/10.1145/nnnnnnn.nnnnnnn [Accessed 11 Jan. 2020].

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N.J., Xie, X. and Li, Z., (2018) DRN: A Deep Reinforcement Learning Framework for News Recommendation. [online] Available at: https://doi.org/10.1145/3178876.3185994 [Accessed 11 Jan. 2020].

ADDITIONAL REFERENCES

[1] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[2] Charu C Aggarwal, (2016). Recommender Systems the Textbook, Springer ISBN 978-3-319-29657-9

[3] Sergey Levine, Deep Reinforcement Learning (2017), http://rail.eecs.berkeley.edu/deeprlcourse-fa17/index.html

[4] Xing Xie, Jianxun Lian, Zheng Liu, Xiting Wang, Fangzhao Wu, Hongwei Wang, and Zhongxia Chen (November 2, 2018), https://www.microsoft.com/en-us/research/lab/microsoft-research-asia/articles/personalized-recommendation-systems/

[5] The Lazy Programmer, Recommender Systems in python, (all videos) https://www.udemy.com/recommender-systems/

[6] Minmin Chen, ACM Channel, Reinforcement Learning for Recommender Systems: A Case Study on Youtube, (March 28, 2019) https://www.youtube.com/watch?v=HEqQ2_1XRTs