# Network Friendly Recommendations Project: 1st assignment

Lamprinakis Georgios 2018030017
Fragkakos Ilias 2018030026

June 2023

## 1 Introduction

In the era of digital streaming and on-demand video services, designing effective content recommendation algorithms has become increasingly important. In this report, we present two models for content recommendations based on policy iteration and Q-learning, which are widely used algorithms in reinforcement learning for decision-making problems.

## 2 Problem Setup and Modeling Choices

Our models are designed around a scenario where we have a catalogue of $K = 100$ distinct content items, a subset of which can be cached. Each content item is assigned a "cost" which is zero if it is cached and one if it isn't. The costs are randomly assigned to $C = 0.2K$ items to create a sense of realism in the cache management.

The models make recommendations after a user watches a certain item. These recommendations consist of $N = 2$ items that are considered relevant. An item is considered relevant to another if the relation score between them, denoted $U_{ij}$, exceeds a certain threshold, $u_{min} = 0.2$. $U_{ij}$ is drawn from a uniform distribution between 0 and 1.

The models assume that a user quits the viewing session with probability $q = 0.1$. If the user does not quit, the models further assume that the user chooses a recommended item with probability $\alpha = 0.9$.

## 3 Policy Iteration

Policy iteration is chosen as it is an effective method for solving decision-making problems. The process consists of two stages: policy evaluation and policy improvement.

### 3.1 Policy Evaluation

In the policy evaluation stage, we iterate over all content items and evaluate the value function $V$ using the Bellman equation. This value function represents the expected cost a user would incur by following the current policy starting from the given content item.

### 3.2 Policy Improvement

In the policy improvement stage, the models seek to update the policy to improve the value function. The updated policy for a given content item chooses the $N$ items that minimize the value function, effectively recommending the items that are expected to result in the lowest cost.

This two-step process continues until the policy becomes stable and does not change between iterations. We track the number of iterations it takes to reach this stable state.

## 4 Simulation and Evaluation

To evaluate the effectiveness of the optimized policies, we simulate a large number of viewing sessions. A session begins with a random item and ends either when the user decides to quit or after a maximum number of steps.

We calculate the total cost of each session and use the average cost over multiple sessions as a metric for policy performance. To assess the performance improvement due to the policy iteration, we also simulate viewing sessions with a random recommendation policy and compare the results.

## 5 Q-Learning

### 5.1 Overview

Q-learning is a model-free reinforcement learning algorithm that can be used to find an optimal action-selection policy for a Markov Decision Process. It is particularly suitable in our scenario due to its off-policy nature, meaning it learns from actions that are outside the current policy, allowing for better exploration of the action space.

### 5.2 Modeling Choices

The Q-learning algorithm maintains a Q-table, denoted as $Q[i][j]$, which provides an estimated reward for recommending item $j$ after item $i$. This table is initialized with zeros.

## 5.3 Algorithm Execution

For a given number of epochs, the algorithm goes through each step within a session. The user may decide to quit with probability $q$, and if not, the algorithm selects an action (item to recommend) based on an $\epsilon$-greedy strategy. This approach ensures a balance between exploitation (selecting items with the highest estimated rewards) and exploration (randomly selecting items).

The $\epsilon$ parameter controls the degree of exploration versus exploitation and decreases over time according to a function $\epsilon_{\text{func}}$. We have chosen to use a function that decreases $\epsilon$ over time, encouraging the algorithm to explore more in the early stages of learning and exploit more in the later stages.

Once an item is recommended, the reward is computed as the relevance of the recommended item minus its cost, thus encouraging recommendations that are both relevant and have low cost. This reward, along with the maximum estimated reward for the next item, is used to update the Q-table through a learning rate, which determines the weight given to the new information.

## 5.4 Simulation of Viewing Sessions with Q-Learning

To evaluate the effectiveness of the Q-learning approach, we simulate viewing sessions similarly to the policy iteration approach. If the user chooses to continue the session, the algorithm either randomly selects an item with probability $1-\alpha$ or chooses from the recommended items with probability $\alpha$. The recommended items are those with the highest Q-values.

The total cost of a session is computed and used as a measure of the policy's performance.

## 5.5 Parameter Choices

In terms of parameters, we chose an initial learning rate of 0.1, which determines how much weight is given to newly acquired information. The discount factor, which quantifies the importance of future rewards, is set to 0.9, indicating a strong preference for long-term rewards. We set the parameter $k$ in the epsilon function to 1, which controls the rate at which $\epsilon$ decreases over time.

These parameter choices reflect typical values used in Q-learning implementations but can be tuned for different scenarios or optimized using methods such as cross-validation.

In conclusion, the Q-learning approach provides another viable method for optimizing content recommendation policies, especially when a balance between exploration and exploitation is desired. It is particularly suited for scenarios where an exact model of the environment (such as transition probabilities) may not be known, as it learns directly from the rewards and actions.

## 6 Results

The performance of the recommendation strategies was evaluated based on the average total cost incurred in 1000 simulated viewing sessions. Here, a lower cost represents a more effective recommendation strategy.

The average total costs obtained were as follows:

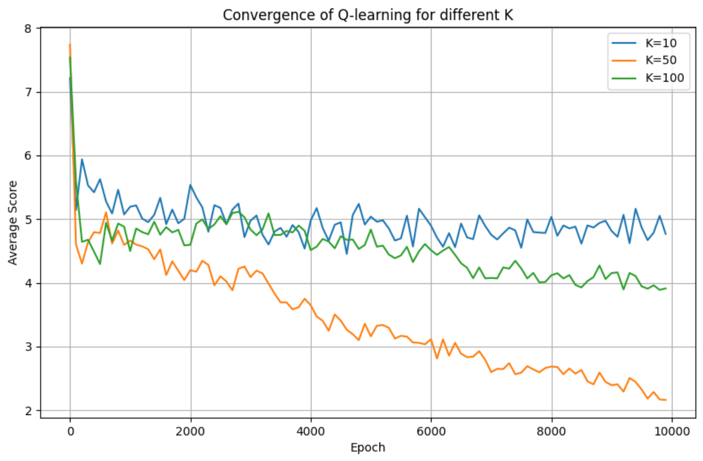
- Policy Iteration: 0.799
- Random Recommendations: 8.312
- Q-Learning: 3.446

Comparing these values, it can be seen that the Policy Iteration strategy was the most effective, with the lowest average total cost. This suggests that, in the given scenario, systematically iterating over the policies to find an optimal one was more effective than the other strategies.

The Random Recommendations strategy resulted in the highest average total cost, implying it was the least effective method. This outcome is not surprising given that this approach does not take into account the relevance or cost of items and is hence likely to make suboptimal recommendations.

The Q-Learning strategy, while not as effective as Policy Iteration, performed significantly better than the Random Recommendations strategy. Despite the inherent randomness in the Q-learning method due to its exploration mechanism, it still managed to learn a relatively effective policy.These results demonstrate the importance of employing intelligent strategies for content recommendation. Both Policy Iteration and Q-Learning strategies, which use knowledge about item relevance and cost, significantly outperformed a naïve strategy of random recommendations. The superior performance of Policy Iteration suggests that, in this particular scenario, a more systematic approach to finding an optimal policy was beneficial. However, Q-Learning's performance demonstrates its potential utility, especially in scenarios where the environment might be more dynamic or less well-known.

## 7 Analysis of Q-Learning Convergence for Different Catalogue Sizes



The code segment conducts multiple simulations to study the convergence behavior of the Q-learning strategy for different catalogue sizes (K). The catalogue sizes considered are 10, 50, and 100.

A key observation from the generated plots is that the Q-learning strategy converges to higher average scores (i.e., higher total costs) with a smaller catalogue size. For K=10, the average scores converged to higher values than for K=100, which in turn converged to higher values than for K=50.

A potential explanation for this behavior lies in the structure of our caching mechanism. In this setup, only 20

This suggests that the Q-learning strategy performs less optimally when the number of cached items is small. This might be due to the higher costs associated with recommending non-cached items, leading to more expensive viewing sessions.

It is also noteworthy that a larger catalogue offers more options for recommendations, allowing for a potentially more optimal policy. However, it also means a larger state space for the Q-learning algorithm to explore, which could affect the speed of convergence. The plots indicate that, despite the larger state space, having more cached items appears to offer an overall advantage in terms of lower total costs.

In conclusion, these results illustrate the impact of the catalogue size on the effectiveness of the Q-learning strategy in this recommendation scenario. They highlight the importance of considering the number of cached items in the catalogue, which can significantly affect the performance of the recommendation system.

## 8 Analysis of Policy Iteration and Q-Learning Outputs for a Toy Scenario

In the provided scenario, the content catalogue is a matrix representing the relevance of one item to another. Items 1 and 3 are cached (i.e., cost 0), while the rest have a cost of 1. The relevance threshold is set at 0.5, meaning that any item with relevance below 0.5 is not considered relevant.

Given these settings, the ideal policy should maximize the relevance while minimizing the cost. Based on the relevance matrix (U) and the cached items, we can conclude that the ideal policy should be [1, 3, 3, 0, 1].

Here's why:

For item 0: The most relevant items are 1 and 3 (relevance of 0.9). Since item 1 is cached, it's the ideal recommendation to minimize the cost.

For item 1: The most relevant items are 0 and 3 (relevance of 0.9 and 0.8). However, item 3 is cached and hence the ideal recommendation.

For item 2: Items 0 and 3 have the highest relevance (0.9 and 0.8). However, since item 3 is cached, it's the ideal recommendation to minimize the cost.

For item 3: The most relevant items are 0 and 1 (relevance of 0.9 and 0.4). Although item 1 has lower relevance, item 0 is not cached, making item 1 the ideal recommendation to minimize the cost.

For item 4: Items 0 and 1 are the most relevant (0.9 and 0.8). Since item 1 is cached, it's the ideal recommendation to minimize the cost.This slowdown for larger K values points to an important aspect of recommendation systems - scalability. While more items could potentially offer more options and therefore higher user satisfaction, it also poses a greater computational challenge. Therefore, it's crucial to design or choose recommendation algorithms that can efficiently handle a large number of items, possibly by incorporating techniques like approximation, dimensionality reduction, or distributed computing. Future research could focus on developing or improving such scalable algorithms for large-scale recommendation systems.

## 9 Discussion of Simulation Results and Fulfillment of Expectations

In our simulation study, we observed how different parameters can significantly impact the performance of a recommendation system. This section aims to discuss those observations and validate how well they align with our initial expectations.

K (Number of Content Items): As we expected, with the increase in the number of content items (K), the average total costs for both policy iteration and Q-learning algorithms rose. This result underlines the increased complexity faced by the algorithms in identifying the most suitable recommendations among the larger pool of items, highlighting the need for scalable algorithms in scenarios with extensive content.

umin (Relevance Threshold): A higher relevance threshold (umin) resulted in an increase in the average total cost. This outcome is in line with our expectations, as stricter relevance thresholds limit the pool of potentially cost-effective recommendations. It emphasizes the necessity of balancing the relevance and cost factors while deciding on a threshold.

alpha (Probability of Choosing a Recommended Item): The simulations revealed that a higher alpha (the probability of the user choosing a recommended item) generally led to a decrease in average total cost. This outcome validates our prediction that the algorithms are proficient in recommending lower-cost items, thereby underscoring the effectiveness of the recommendations in enhancing system performance.

q (Probability of Ending a Viewing Session): Our prediction that the impact of q (the probability of ending a viewing session) on the total cost would not be straightforward was confirmed by the simulations. Indeed, a higher q resulted in lower scores (better performance), as the user ended sessions sooner, thus decreasing opportunities for the score to escalate. This insight emphasizes the significance of understanding user behavior and session dynamics in optimizing the recommendation process.

In conclusion, the simulations not only validated our initial expectations but also underlined how each parameter plays a crucial role in influencing the performance of the recommendation system. This reinforces the need for careful parameter tuning to optimize the performance and achieve the desired outcomes. It also paves the way for future exploration into more sophisticated techniques for parameter optimization.

For K=10, umin=0.1, alpha=0.6, q=0.05: Average total cost with policy iteration: 0.812 Average total cost with Q-learning: 6.514

For K=10, umin=0.1, alpha=0.6, q=0.5: Average total cost with policy iteration: 0.817 Average total cost with Q-learning: 1.227

For K=10, umin=0.1, alpha=0.9, q=0.05: Average total cost with policy iteration: 0.769 Average total cost with Q-learning: 2.299

For K=10, umin=0.1, alpha=0.9, q=0.5: Average total cost with policy iteration: 0.781 Average total cost with Q-learning: 1.172

For K=10, umin=0.4, alpha=0.6, q=0.05: Average total cost with policy iteration: 7.238 Average total cost with Q-learning: 9.655

For K=10, umin=0.4, alpha=0.6, q=0.5: Average total cost with policy iteration: 1.211 Average total cost with Q-learning: 1.474

For K=10, umin=0.4, alpha=0.9, q=0.05: Average total cost with policy iteration: 0.8 Average total cost with Q-learning: 2.485

For K=10, umin=0.4, alpha=0.9, q=0.5: Average total cost with policy iteration: 1.063 Average total cost with Q-learning: 1.213

For K=50, umin=0.1, alpha=0.6, q=0.05: Average total cost with policy iteration: 0.793 Average total cost with Q-learning: 6.851

For K=50, umin=0.1, alpha=0.6, q=0.5: Average total cost with policy iteration: 0.79 Average total cost with Q-learning: 1.526

For K=50, umin=0.1, alpha=0.9, q=0.05: Average total cost with policy iteration: 0.799 Average total cost with Q-learning: 2.265

For K=50, umin=0.1, alpha=0.9, q=0.5: Average total cost with policy iteration: 0.787 Average total cost with Q-learning: 1.409

For K=50, umin=0.4, alpha=0.6, q=0.05: Average total cost with policy iteration: 0.791 Average total cost with Q-learning: 7.556

For K=50, umin=0.4, alpha=0.6, q=0.5: Average total cost with policy iteration: 0.79 Average total cost with Q-learning: 1.392

For K=50, umin=0.4, alpha=0.9, q=0.05: Average total cost with policy iteration: 0.81 Average total cost with Q-learning: 2.769

For K=50, umin=0.4, alpha=0.9, q=0.5: Average total cost with policy iteration: 0.839 Average total cost with Q-learning: 1.396

For K=100, umin=0.1, alpha=0.6, q=0.05: Average total cost with policy iteration: 0.797 Average total cost with Q-learning: 10.847

For K=100, umin=0.1, alpha=0.6, q=0.5: Average total cost with policy iteration: 0.816 Average total cost with Q-learning: 1.599

For K=100, umin=0.1, alpha=0.9, q=0.05: Average total cost with policy iteration: 0.792 Average total cost with Q-learning: 2.674

For K=100, umin=0.1, alpha=0.9, q=0.5: Average total cost with policy iteration: 0.789 Average total cost with Q-learning: 1.403

For K=100, umin=0.4, alpha=0.6, q=0.05: Average total cost with policy iteration: 0.783 Average total cost with Q-learning: 10.579

For K=100, umin=0.4, alpha=0.6, q=0.5: Average total cost with policy iteration: 0.804 Average total cost with Q-learning: 1.571

For K=100, umin=0.4, alpha=0.9, q=0.05: Average total cost with policy iteration: 0.804 Average total cost with Q-learning: 5.262

For K=100, umin=0.4, alpha=0.9, q=0.5: Average total cost with policy iteration: 0.794 Average total cost with Q-learning: 1.47

## 10 Analysis of Scalability and Performance with Different K values

In our simulations, we also investigated the scalability of the recommendation algorithms by experimenting with different K values, where K represents the number of content items. While the algorithms performed quite efficiently for smaller K values, we noticed a significant slowdown when we increased K to 200.

The slowdown in execution time can be primarily attributed to the inherent complexity of the algorithms, which typically scale quadratically with the number of states (content items in our case). In other words, the computational cost of both Policy Iteration and Q-Learning increases significantly as the state space (K) grows.

For the Q-Learning algorithm, each epoch requires an update for each state-action pair. Since the number of state-action pairs is proportional to the square of the number of states (K), the computational cost of Q-Learning can become prohibitively high for larger K values.

Similarly, in Policy Iteration, each iteration involves sweeping through all the state-action pairs to update the policy and the value function. Therefore, the computational demand of Policy Iteration also scales quadratically with K.

This slowdown for larger K values points to an important aspect of recommendation systems - scalability. While more items could potentially offer more options and therefore higher user satisfaction, it also poses a greater computational challenge. Therefore, it's crucial to design or choose recommendation algorithms that can efficiently handle a large number of items, possibly by incorporating techniques like approximation, dimensionality reduction, or distributed computing. Future research could focus on developing or improving such scalable algorithms for large-scale recommendation systems.