# Food Recommender System for Your Daily Dilemma

Reinforcement Learning Using Two Q-Learning Agents

Dongsoo Seo

**Abstract**

The model was designed to recommend food for users. The model includes two Q-learning agents that are merged at the end for the recommendation users receive. To provide better accuracy of food recommendations, three different objective attributes of each food were analyzed. The hypothesis was that more attributes would bring better accuracy and hoped that this can be a solution for future improvement of the model. However, the study variation found that it is not always the case and another option for future improvement was needed. Fortunately, options of Double Deep Q-learning and Kubernetes showed potential room for improvement, and I believe this can be a viable option to further improve the model.

## Introduction

Food is an essential part of everyone's life. Food is not just a source of nutrition to supplement energy to your body, but it also provides joy in our life. The thought of tasty food that you are desperately waiting for lunch might be the only hope to survive your day at work. Deciding what to eat for the day is a very exciting task, but it can also be a great stress for some people. Surely, you will run out of new food ideas eventually and will have to squeeze your brain for a new idea. Or, there are simply too many options to choose from and you cannot decide which one is the best. Or, someone is interested in a protein-based diet for their health and he or she may need some help finding that food. Hence, I made a model to handle such problems in the hope to reduce the stress of finding the food they want.

The goal of this project is to construct a model that (a) users can interact easily within a short time and (b) predicts users' preferences and provide an accurate recommendation. An additional goal is testing for a potential improvement option by adding more attributes for preference consideration, changing the number of attributes to see if more attributes can improve the model accuracy.

## Problem

Food recommender systems have been implemented in the past, but they are far from being effective. People tried choosing a random menu by spinning a wheel, but it did not guarantee the food that users like. Food lists or books were also used, but it was too time-consuming because users needed to read through all the lists until they find their favorites. The issue of such options in the past is users cannot process through all food options and they often miss great food choices because they have not thought of it.

To understand the food selection process of a user, three major information of food were selected: ingredients, nutrients, and culture. Although only three attributes were considered to simplify the food selection process, the combination of these attributes can provide a great number of combination list. There are thousands of ingredients, nutrition, and culture combinations, and if users were to list all their preferred attributes of their food preferences, the user will be exhausted before they can describe them all. Answers that users can provide in a short time is limited, and an exhaustive comparison among the combinations would not be appropriate. Instead of an exhaustive survey, a reinforcement learning agent can be implemented to learn the user to predict the item the user wants the most. Additionally, instead of choosing ingredients, nutrients, and food culture of their preference,

users will only have to choose the name of the food they like to reduce the time users spend comparing options.

**Q-learning**

Q-learning is a model-free reinforcement learning algorithm. "Model-free" means that the transition probability distribution is not used to learn the environment. The transition probability is unknown in this recommender system because it is unknown how an ingredient is related to another or whether users who like an ingredient also like another ingredient. Therefore, this algorithm is better suited for this environment.

Q-learning algorithm uses the following update expression (Russell p.884):

$$Q(a,s) \leftarrow Q(a,s) + \alpha\left(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)\right)$$

Where $\alpha$ is the learning rate, $\gamma$ is the discount factor, $R(s)$ is the reward, and $Q(a,s)$ is the action value or the action-utility function. Instead of a transition probability $T(s,a,s')$, Q-learning algorithm predicts the expected value $U(s)$ using the predicted maximum action value $\max_{a'} Q(a',s')$. This model-free algorithm is certainly slower than model-based reinforcement learning, but solving unknown environments makes the trade-off worth it.

**Implementation**

This model imports the code for Markov Decision Process and Q-learning from *Artificial Intelligence: A Modern Approach*. Each state of a Markov Decision Process contains one item from each attribute: ingredient, culture, and nutrition. These three attributes are used to pick a candidate from the food list. The action is limited to change only one attribute at a time. The environment is analogous to a 3D grid environment that agents can move to any state that is in the same line as the current state.

Two Q-learning agents are in the model, one for the left side, and another for the right side. For both agents, the learning rate $\alpha$ is set to 0.1, and the discount factor $\gamma$ is set to 0.99. Random initial states are given to the agents so that the agents can cover as many states as possible without overlapping. The terminal state does not exist. Instead, the model stops with a given number of iterations. You can compare the agents as two agents in a 3D grid environment, where the agents move to different states for a fixed number of iterations and learn the utility of each state.

In each iteration, each agent chooses a food item that matches its current state to the user. Whenever the user chooses one side, the respective agent receives a fixed amount of reward, and the other agent receives the same amount of penalty. This approach enables the model to receive two utilities at the same time. At the end of the iterations, the utilities of both agents are summed together to predict the attributes that both agents think is the best. Using the summed utility, the model predicts the best food.

The ingredients and the nutrients are gathered from *FoodData Central* by *USDA Agricultural Research Service*. For clarity, each food is manually assigned to the corresponding cuisine culture.

I believe this approach is simpler and more efficient than implementing a single Q-learning agent trying to administer two sides, two states, and two utilities at the same time. Additionally, because the model receives two inputs per iteration, the convergence should be twice faster than a normal Q-learning agent receiving only one input per iteration.

A method of comparing two items is an easy and useful way to receive users' input. The strengths and weaknesses of this model are in its simplicity. Compare to a 5-star rating system, users can make easier answers. Also, the 5-star rating has a downside of being affected by individuals' rating tendencies, but the simple selection method of this model can avoid this issue. However, this simplicity

may also induce assumption issues because the selection of this model needs more answer assumptions than the 5-star rating system.

**Experiments**

In the experiments, an automated user was used to provide instant results to cover a thousand iterations. The user had one favorite item in each attribute. These three items were used to determine which food was better between the two when the two agents presented their food items for comparison. More specifically, the number of favorite items the food information contains determined the value. The comparison was done a thousand times.

The experiments tested the general quality of this model as well as the model in different environments and with a different state and action space. Unless stated, a thousand iterations were given to learn the user for each experiment. Each attribute was weighted equally. i.e. When predicting the best food, each attribute was considered equally important.

For each table, the first three columns represent the attributes a user liked, and the "Result" column represents the food the model predicted. The accuracy was measured by the number of attributes the "Result" food contained or matched.

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Ground Beef | Chinese | Carbs | Kung Pao Beef | 2 |
| Soup | Korean | Protein | Pho | 1 |
| Parsley | American | Protein | Hamburger | 1 |
| Milk | Indian | Carbs | Chicken Curry | 2 |
| Tomatoes | Italian | Fat | Chicken Curry | 1 |
| Cauliflower | Japanese | Carbs | Tempura | 3 |
| Peanuts | Italian | Protein | Pad Thai | 1 |
| Cabbage | Indian | Protein | Chicken Curry | 1 |
| Potatoes | Italian | Protein | Chicken Curry | 1 |
| Beef Steak | American | Carbs | Hamburger | 2 |

Default model result

The result shows that it was hard to predict all three attributes. However, each test found at least one matching attribute. I think the model performed well enough to predict the food users may like.

**a) Number of Iteration**

Instead of a thousand iterations, the models ran several different numbers of iterations to see if the accuracy persists. I hypothesized that the models perform worse because the model had fewer iterations to learn the environment.

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Cauliflower | Mexican | Carbs | Tempura | 2 |
| Wheat flour | Korean | Carbs | Sushi | 1 |
| Peppers | Thai | Protein | Sushi | 0 |
| Salt | Chinese | Fat | Sushi | 1 |
| Chickpeas | American | Protein | Hamburger | 1 |
| Mung beans | Indian | Carbs | Hamburger | 1 |
| Cucumber | Italian | Fat | Spaghetti | 1 |
| Ground beef | Mediterranean | Fat | Teriyaki Chicken | 0 |
| Soy sauce | Italian | Carbs | Teriyaki Chicken | 1 |
| Lettuce | Korean | Fat | Dukboki | 1 |

100 iterations

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Ground beef | Chinese | Carbs | Kung Pao Beef | 2 |
| Sweet potato | Mediterranean | Protein | Tempura | 1 |
| Chicken | Thai | Carbs | Falafel | 1 |
| Vegetable oil | American | Carbs | Hamburger | 2 |
| Soup | Japanese | Protein | Hamburger | 0 |
| Pasta | Indian | Protein | Spaghetti | 1 |
| Lentils | Chinese | Carbs | Spaghetti | 1 |
| Tortillas | Vietnamese | Fat | Spaghetti | 0 |
| Water | Italian | Protein | Tempura | 1 |
| Potatoes | Korean | Carbs | Bibimbap | 2 |

10 iterations

The difference between 100 iterations and 1,000 iterations is noticeable. Some results show that none of the items matched the result. However, no significant differences between 10 iterations and 100 iterations were found. My hypothesis was somewhat obvious but I learned that if a human user were to use this model, about ten iterations may be enough to perform like when a hundred iterations are done.

**b) Weight Variances**

Given a state, the food was chosen with equal weight in each attribute. In this experiment, one attribute was given a 60% weight (ratio of 3:1:1). I looked for a possibility if the varying weight would help the model to focus on more important attributes.

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Sauce | American | Protein | French Fries | 1 |
| Peppers | Italian | Protein | Tempura | 1 |
| Water | Mexican | Protein | Enchilada | 2 |
| Parsley | Indian | Carbs | Chicken Curry | 2 |
| Ground Beef | Chinese | Protein | Dumpling | 1 |
| Tomatoes | American | Carbs | Chicken Curry | 2 |
| Seeds | Indian | Carbs | Chicken Curry | 2 |
| Water | Indian | Fat | Dumpling | 1 |
| Sugars | American | Fat | Pho | 1 |
| Squash | Mediterranean | Carbs | Tempura | 2 |

High ingredient weight

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Cauliflower | Mexican | Fat | Quesadilla | 1 |
| Seeds | Korean | Protein | Bibimbap | 1 |
| Rice | Chinese | Fat | Bibimbap | 1 |
| Yogurt | Thai | Protein | Chicken Curry | 1 |
| Chicken | Japanese | Protein | Teriyaki Chicken | 2 |
| Peppers | Korean | Carbs | Bibimbap | 2 |
| Ground beef patty | Mediterranean | Carbs | Falafel | 2 |
| Cabbage | Mexican | Carbs | Quesadilla | 2 |
| Pasta | Thai | Fat | Spaghetti | 1 |
| Vegetable oil | Mexican | Protein | French Fries | 1 |

High culture weight

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Ginger root | Indian | Carbs | Chicken Curry | 3 |
| Cornstarch | American | Protein | Hamburger | 1 |
| Cucumber | Thai | Protein | Bibimbap | 1 |
| Peppers | Japanese | Fat | Kung Pao Beef | 1 |
| Pasta | Japanese | Fat | Sushi | 1 |
| Mushrooms | Japanese | Protein | Bibimbap | 1 |
| Mung beans | Mediterranean | Fat | Bibimbap | 1 |
| Spices | Mexican | Protein | Pad Thai | 1 |
| Cheese | Mediterranean | Carbs | Quesadilla | 2 |
| Mung beans | Indian | Fat | Bibimbap | 1 |

High nutrient weight

The tables show that higher weight on the ingredient attributes led to higher accuracy. I believe that the higher weight enabled the model to focus on the respective attribute. Because the nutrient attribute has 1/3 chance of predicting a correct item if the item is chosen at random, focusing on the nutrient attribute by putting more weight on it did not help the model. Instead, the accuracy was lowered because other attributes had comparably lower weights.

**c) Attribute Exclusion**

The attributes can be ignored by assigning the weight as zero. In this experiment, one attribute was given a 0% weight (ratio of 0:1:1).

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Egg | Indian | Protein | Chicken Curry | 1 |
| Chickpeas | Thai | Carbs | Pad Thai | 2 |
| Beef | Mediterranean | Protein | Falafel | 1 |
| Margarine | Vietnamese | Carbs | Pho | 2 |
| Cornstarch | Chinese | Protein | Kung Pao Beef | 2 |
| Shortening | Mediterranean | Carbs | Falafel | 2 |
| Onions | Thai | Fat | Pho | 1 |
| Ginger root | American | Carbs | Pho | 2 |
| Tortillas | Mediterranean | Protein | Quesadilla | 1 |
| Cauliflower | Mexican | Fat | Chicken Curry | 1 |

No ingredient weights

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Ginger root | Indian | Fat | Kung Pao Beef | 1 |
| Soy sauce | Japanese | Fat | Kung Pao Beef | 1 |
| Sugars | Korean | Carbs | Dukboki | 3 |
| Basil | Mexican | Protein | Quesadilla | 1 |
| Vegetable oil | Japanese | Fat | Tempura | 2 |
| Yogurt | Korean | Protein | Bibimbap | 1 |
| Tortillas | Chinese | Carbs | Dumpling | 2 |
| Lime juice | Indian | Protein | Chicken Curry | 1 |
| Chicken | Italian | Protein | Chicken Curry | 1 |
| Spices | Korean | Fat | Dukboki | 2 |

No culture weights

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Lime juice | American | Protein | Pad Thai | 1 |
| Basil | Korean | Carbs | Bibimbap | 2 |
| Egg | Thai | Fat | Tempura | 1 |
| Sauce | American | Fat | Bibimbap | 1 |
| Milk | Korean | Fat | Bibimbap | 1 |
| Ground beef | Italian | Fat | Spaghetti | 2 |
| Onions | Japanese | Fat | Pad Thai | 1 |
| Carrots | Mediterranean | Fat | Spaghetti | 1 |
| Tomatillos | American | Fat | Hamburger | 1 |
| Cornstarch | Vietnamese | Protein | Kung Pao Beef | 1 |

No nutrient weights

The tables show that ignoring attributes led to lower accuracies. Such decreases are the most noticeable when the nutrient attribute was ignored. With the previous experiment, I concluded that the

nutrient attribute should neither be focused because such attribute is easy to predict, nor ignored because the model should not lose the opportunity to learn the easy prediction.

**d) State with two ingredients**

Instead of one item for each state, states were given two ingredients. I hypothesized that the model would perform better because the model would learn two possible ingredients instead of one. Also, because the previous result proved that states focusing on fewer attributes showed lower accuracy, I hastily believed that this experiment would increase the performance.

| Ingredient | Culture | Nutrient | Result | Attributes matched |
|---|---|---|---|---|
| Shortening | Thai | Fat | Sushi | 0 |
| Seaweed | Mexican | Protein | Dumpling | 0 |
| Salt | Vietnamese | Fat | Sushi | 1 |
| Sugars | Japanese | Carbs | Tempura | 2 |
| Lentils | Japanese | Carbs | Chicken Curry | 2 |
| Peanuts | Mediterranean | Fat | Kung Pao Beef | 1 |
| Cheese | American | Protein | Lasagna | 1 |
| Cabbage | Mediterranean | Fat | Dukboki | 1 |
| Sauce | Mexican | Fat | Hamburger | 1 |
| Ground beef | Korean | Carbs | Spaghetti | 1 |

Two ingredients

The result turns out to be against my hypothesis. I realized that, because one dimension was added, more iterations were required to fully learn the environment. The result is almost the same as the previous experiments that had fewer iterations.

To learn users' preferences better, I believed that more attributes are needed to learn about the environment. However, the results show that adding more attributes may lower the accuracy, and there need to be improvements more than just configurations. For example, a new concept of a model-free algorithm is needed, or the transition model should be predefined or assumed to make the model achieve better accuracy.

Because Q-learning does not use any transition models, the algorithm may suffer from overfitting more easily than other model-based reinforcement learning. Additionally, if the model is changed, the new model may require more resources and time. If this model is used as an online service or a platform like Spotify, any resource management is required.

**Literature Survey**

**a) Double Q-learning (Van Hasselt)**

Overestimation sometimes occurs in the Q-learning algorithm. Such error is caused mainly by maximum action value being used for the expected maximum action value. Instead of one set of estimators, Double Q-learning uses two sets of estimators to get the expected maximum action value. Double Q-learning is not too different from the Q-learning algorithm except the expected maximum action value is found with the estimator from the different sides to reduce the noise.

The double estimator can also be applied to Deep Q Network, which utilizes a deep neural network to estimate the maximum action value. Double DQN has proven to outperform the Deep Q Network (Van Hasselt *et al*.).

The idea of using two Q agents from my project looks the same as Double Q-learning, except the updaters do not interact with each other, and I think that such interaction is the factor to the reduction of overestimation.

Double Q-learning Algorithm

Initialize $Q^A, Q^B, s$
Repeat
  Choose $a$, based on $Q^A(s,\cdot)$ and $Q^B(s,\cdot)$, observe $r, s'$
  Choose (at random) UPDATE(A) or UPDATE(B)
  if UPDATE(A) then
    Define $a^* = \arg\max_a Q^A(s', a)$
    $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)\big(r + \gamma Q^B(s', a^*) - Q^A(s,a)\big)$
  else if UPDATE(B) then
    Define $b^* = \arg\max_a Q^B(s', a)$
    $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)\big(r + \gamma Q^A(s', b^*) - Q^B(s,a)\big)$
  end if
  $s \leftarrow s'$
until end

**b) Kubernetes and Kubeflow**

Like virtual machines, Kubernetes is a container-orchestration system. It uses “containers” to allocate spaces to each application. Compared to virtual machines, containers are lighter, faster, and portable across operating systems. Because of its flexibility, Kubernetes can be deployed, managed, and scaled easily.

A Spotify Labs article claims that the transition from TensorFlow to Kubeflow benefitted Spotify to “better manage workloads and accelerate the pace of experimentations and roll out.” Kubeflow is a platform designed to integrate machine learning with Kubernetes. It enables machine learning pipelines to manage workflows running on Kubernetes. Because Spotify has about 250,000,000 users with more than 50,000,000 tracks and 500,000 podcast titles, such transition made Spotify to easily manage their highly scaled real-time environment.

Because the food recommender system runs with only one user and the limited size of the food list, such a gigantic scale is not needed, but it is good to know how personalization is managed for each user.

**Conclusion**

Although the experiments have shown that the agents do not converge easily on the perfect results in a small number of iterations, most of the experiments have proven that the model is useful enough that it predicts at least one preference the users have. Because people usually look for variations of their favorite food, I think this model has potential. The food recommender system has room for improvement. For example, filters such as vegetarian options can be added to let users enjoy a variety of vegetable-based meals.

However, the experiments also showed that the model has its limits even with different environment settings. To make this project useful to everyone, the idea of deploying two parallel Q-learning agents was not enough. Fortunately, I saw the potential improvement of this model while I was studying about algorithms of Double Q-learning and Double Deep Q-learning during the survey. The implementation of such algorithms may improve the project by reducing its overestimation.

**Works Cited**

*FoodData Central*, USDA Agricultural Research Service, https://fdc.nal.usda.gov/.

*Kubernetes*, https://kubernetes.io/.

Norvig, Peter, et al. *AIMA3e-Java*, https://github.com/aimacode/aima-java.

Popkin, Barry M. “The World Is Fat.” Scientific American, Sept. 2007, pp. 88–95.

Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2010.

*Spotify Labs*. “For Your Ears Only: Personalizing Spotify Home with Machine Learning.” https://labs.spotify.com/2020/01/16/for-your-ears-only-personalizing-spotify-home-with-machine-learning/.

Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning." Thirtieth AAAI conference on artificial intelligence. 2016.

Van Hasselt, Hado. "Double Q-learning." Advances in neural information processing systems. 2010.