An incremental preference elicitation for group recommendations using a Bayesian approach

Final Year Engineering Internship Report

AUTHOR: MAÉVA CAILLAT
caillat.maeva@gmail.com

SUPERVISOR: CRISTINA MANFREDOTTI (ASSOCIATE PROFESSOR AT AGROPARISTECH)
cristina.manfredotti@agroparistech.fr

SUPERVISOR: PAOLO VIAPPIANI (CNRS RESEARCHER)
Paolo.Viappiani@lip6.fr

SUPERVISOR: NICOLAS DARCEL (ASSOCIATE PROFESSOR AT AGROPARISTECH)
nicolas.darcel@agroparistech.fr

HEAD OF THE TRAINING PROGRAM IN 'MATHEMATICS AND APPLICATIONS' AT CENTRALE NANTES: ANTHONY NOUY
anthony.nouy@ec-nantes.fr

Internship realized between April 6th, 2020 and September 25th, 2020 (25 weeks)

AgroParisTech - 16 rue Claude Bernard, 75005 Paris - FRANCE
Ecole Centrale de Nantes - 1 Rue de la Noë, 44300 Nantes - FRANCE

Abbreviations

CER	Comité d'Ethique pour la Recherche
CNRS	French National Centre for Scientific Research
CROUS	Centre régional des œuvres universitaires et scolaires
DA2PL	From Multiple Criteria Decision Aid to Preference Learning
EkINocs	Expert Knowledge, INteractive modelling and learning for understanding and decisiOn making in dynamic Complex Systems
EL	Expected Loss
EM	Expected Maximum
ESB	Expected Score heuristic for Borda
EV	Expected Value
EVOI	Expected Value Of Information
IDE	Integrated Development Environment
IG	Information Gain
IGB	Information Gain heuristic for Borda voting
i.i.d.	independent and identically distributed
JFN	Journées Francophones de Nutrition
MDP	Markov Decision Process
MIA	Applied Mathematics and Computer Science
NW	Necessary Winner
pmax	The Borda possible maximum
pmin	The Borda possible minimum
PNCA	Nutrition Physiology and Eating Behavior
Spyder	Scientific PYthon Development EnviRonment
WEM	Weighted Expected Maximum
WIG	Weighted Information Gain

Abstract

This report gathers and presents the work that I have accomplished for six months, as part of my final year engineering internship, at the Applied Mathematics and Computer Science laboratory of AgroParisTech (Paris). I have worked on preference elicitation for group decisions using voting rules. In this report, my supervisors and I propose a general, domain-free framework for preference management, where the goal is to minimize the communication cost with users. We introduce heuristics that have proved their worth in the literature and combine some of them to get an even more efficient elicitation strategy. We assess the precision of the Borda aggregation protocol thanks to real-life group preferences data that we have gathered. We compare the performances of four heuristics on ranked data, under the Borda voting protocol. We suggest an interactive incremental framework where, at each stage, one user sorts two items. We propose an approach that determines which query to select next until finding a winning candidate. To do so, we combine a heuristic computing the expected value of information of future requests, and another heuristic using the information gain of these queries. To reduce the interactions with users, we also use a stopping algorithm criterion based on the expected loss. This elicitation strategy relies on probabilistic rating distributions. These permutation distributions are updated iteratively in a Bayesian way, allowing their accuracy to increase. We show the effectiveness of our system by evaluating it on two real-world datasets: the already-existing sushi dataset and the tailor-made CROUS dataset.

Acknowledgement

Throughout my internship and while writing this report, I received a lot of support and help.

I would first like to thank my supervisors, Paolo Viappiani, Cristina Manfredotti, and Nicolas Darcel, for their guidance through my internship. Thanks to his valuable expertise in preference elicitation, decision-making, and recommender systems, Paolo continuously helped me to formulate the research questions and to look in the right direction for implementing our model. Thanks to her regular follow-up and relevant feedback, Cristina helped me sharpen my reasoning and brought my work to a higher level. Nicolas helped me to anchor my work in a concrete context, applying our framework to the field of nutrition, and adopt a multidisciplinary vision of research.

I would like to acknowledge Doctor Lihi Dery for exchanging with me on her very inspiring paper (Dery et al. 2015).

I would like to thank Stéphane Dervaux for running my code on the INRAE's servers and thus saving me a considerable amount of time.

I would like to acknowledge Professor Mathieu Ribatet for inspiring my interest in Bayesian statistics.

I would like to thank my working partner and friend, Sai Mourya Surabhi, for the help and good advice she has given me throughout the year.

I would like to thank my mother for her unconditional support during these long and intense years of study. Her dedication is one of the reasons I can now write these acknowledgments.

Finally, I would like to thank my partner for his love and continuing support, and for always making me look on the bright sides of things.

Contents

Abbreviations **3**

Abstract **5**

Acknowledgement **7**

Introduction **11**

1 Background information **13**
- 1.1 Context of the internship 13
- 1.2 Basic notions and State-of-the-Art 13
- 1.3 Bayesian Elicitation Approach 15
  - 1.3.1 General process 15
  - 1.3.2 Probabilistic Voter Permutations Distribution Model 15
  - 1.3.3 Finding a possible or a necessary winner 16
  - 1.3.4 The Expected Loss 16
  - 1.3.5 The item winning probability 18
  - 1.3.6 Information Gain Heuristic for Borda Voting 19
  - 1.3.7 Highest Expected Score Heuristic for Borda Voting 19
  - 1.3.8 Expected Value of Information Heuristic for Borda Voting 20
  - 1.3.9 EVOI and IGB mixed 20

2 Materials and Methods **21**
- 2.1 The programming environment 21
- 2.2 The sushi dataset 22
- 2.3 The CROUS dataset 22

3 Results and Discussion **23**
- 3.1 Results 23
  - 3.1.1 Performances of the heuristics 23
  - 3.1.2 Relevance of the Borda aggregation protocol 23
- 3.2 Discussion 26

Conclusion **29**

A Our publications **37**

B Our nutrition experiment **41**

C Datasets **43**

D Noticeable Python functions **45**

Introduction

The issue of choosing an item to recommend to a group of people has been addressed in both the classic social choice literature (Chevaleyre et al. 2007) and more recently, in the recommender systems' literature (Felfernig et al. 2018). The process is simple when the system knows all the preferences of the users over the items. However, surveying the users about all their tastes might be constraining and time-consuming.

Fortunately, one can find an item that is likely to satisfy a group without knowing every user's preference (Konczak and Lang 2005). We introduce an incremental elicitation process that asks a reduced number of questions to users until finding an item suitable for the group. Studies have shown it is easier for users to state opinions when the queries are pair wise (Balakrishnan and Chopra 2012). To sort items, one can use the Borda aggregation voting protocol that assigns points to the objects of fixed ranked lists of preferences (Chevaleyre et al. 2007). One can assess the regret of recommending an item without knowing all the preferences, thanks to the expected loss (Chajewska, Koller, and Parr 2000). This stopping criterion leans on the utility theory (Braziunas and Boutilier 2008).

Previous research works have already implemented algorithms that use probabilistic methods to guide preference elicitation and minimize the overall number of queries asked. However, they never combined the utility theory (Braziunas and Boutilier 2008) with strategies based on the entropy or the items' winning probabilities (Dery et al. 2015). These heuristics use a single stopping criterion while one can mix multiple breaking conditions and thus reduce the number of queries. Finally, to the best of our knowledge, nobody has already implemented group recommender systems in nutrition.

My report aims to present the food recommender system for groups we have implemented. Our model combines a heuristic based on the expected value of information and a strategy founded on the information gain. The program stops when it finds a necessary or an approximate winner, or when the expected loss is getting close to a defined value. We compare our model to other approaches using the features' winning probabilities, the information gain, or the expected value of information, considered separately. To do so, we use the sushi dataset (Kamishima, Kazawa, and Akaho 2005). We expect the superiority of our model in terms of communication cost, compared to the methods quoted above. We test the precision of the Borda voting protocol by analyzing the results following a real nutrition group experiment that we have carried out. We also expect the Borda protocol to be an accurate aggregation rule.

I begin with background information related to our work: the context of my internship, the problem formulation, and the preference elicitation model. The following chapter describes the materials and methods we used to carry out experiments. These trials aimed at comparing elicitation strategies and assessing the accuracy of the Borda voting protocol. I conclude by analyzing the results and discussing future research directions.

Chapter 1

Background information

The goal of my internship was to design and implement a Bayesian group recommendation system based on voting theory with applications to food choices. After contextualizing my internship, I present preliminary concepts pulled out of the social choice literature. Then, I introduce different Bayesian elicitation strategies and our general framework.

1.1 Context of the internship

To complete my final academic year at Centrale Nantes, a French top-ranking Engineering school, in the Applied Mathematics option, I wanted to go on a research training course, especially I wanted to work in Bayesian statistics. Thus, I did a 6-month internship at AgroParisTech (*AgroParisTech website* 2020), Paris institute of technology for life, food and environmental sciences. This institution is under the care of the French Ministries of Agriculture and Education. AgroParisTech is a member of the consortium Université Paris-Saclay (*Université Paris-Saclay webpage* 2020), which is the 14th best university in the world, according to the 2020 Shanghai ranking. I was part of the Applied Mathematics and Computer Science (MIA) (*MIA webpage* 2020) research unit. In particular, I joined the Expert Knowledge, INteractive modelling and learning for understanding and decisiOn making in dynamic Complex Systems (EkINocs) team, and I worked in close collaboration with the Nutrition Physiology and Eating Behavior (PNCA) (*PNCA webpage* 2020) research unit. Three researchers supervised my internship. My main supervisor, Cristina Manfredotti, is an associate professor at AgroParisTech, in the MIA research unit. Her research interests are probabilistic reasoning, relational models and multi-target tracking. Paolo Viappiani, who is a French National Centre for Scientific Research (CNRS) researcher, also led my internship. He is working in the DECISION team of the LIP6 laboratory of computer science at Sorbonne University. His research interests are preference elicitation, decision-making under uncertainty, sequential decision-making, recommender systems and computational social choice. Dr Manfredotti and Dr Viappiani guided me on the mathematical and computational aspects of my work. My third supervisor is Nicolas Darcel, a research associate at AgroParisTech, in the PNCA team. His research interests are behavioral sciences applied to food decisions. He helped me adapt my recommender system to group food choices. A PhD student will take over our work, as part of her thesis whose purpose is to create a recipe recommender system, in association with the Group SEB (*Groupe SEB* 2020), a global company selling small household equipment.

Because of the COVID-19 pandemic, I teleworked the first two months, and worked partly remotely, partly on-site for the last four months. My tutors mostly worked from home. This context enabled me to gain greater independence in my work, as well as resilience.

1.2 Basic notions and State-of-the-Art

We want to simulate the decision making of a group, for example, the selection of a dish from a menu for a group of friends at the restaurant. According to the paper of Chevaleyre et al. 2007, voting is one of the most popular ways of reaching common decisions. Even if there are many voting rules, we only focus on the Borda rule for our model because it has been widely studied in social choice theory and it is easy to compute.

We want to represent the preference profiles of the set of voters $V = \{v_1, ..., v_m\}$ over the set of candidate items $C = \{c_1, ..., c_n\}$. For every voter, Borda assigns $n-1$ points to the first-ranked item, $n-2$ points to the second item, and so forth until the last item which gets 0. Note that the Borda score, as it is classically used in social choice theory, assumes that the users' rankings are known.

A ranking r is a permutation over the items. $r(a)$ denotes the position (rank) of item a in ranking r and we use r_j to show the item in the j-th position in ranking r. Therefore $r(r_j) = j$ and $r_{r(a)} = a$. A ranking can also be explicitly expressed as a tuple $\langle r_1, r_2, r_3\rangle$.

Example 1. $r = \langle b, c, a\rangle$ corresponds to $r_1 = b$, $r_2 = c$ and $r_3 = a$ and $r(a) = 3$, $r(b) = 1$, $r(c) = 2$.

$a \succ_r b$ represents the event "item a is preferred to b in r"; it means that the position of a in r is before the position of b: $r(a) < r(b)$. The position of an alternative can be expressed as n minus the number of alternatives that have ranks lower than $r(a)$ (i.e. they are preferred to a), or equivalently one plus the number of alternatives whose positions are higher than $r(a)$. Formally:

$$r(a) = n - \sum_{b\in C; b\neq a} \mathbb{1}_{r(b)>r(a)} = 1 + \sum_{b\in C; b\neq a} \mathbb{1}_{r(a)>r(b)} \tag{1.1}$$

$$= n - \sum_{b\in C; b\neq a} \mathbb{1}_{a\succ_r b} = 1 + \sum_{b\in C; b\neq a} \mathbb{1}_{b\succ_r a} \tag{1.2}$$

where $\mathbb{1}_.$ is the indicator function (it is worth one when the condition is true).

To determine a winning item, that is to say to select a candidate for all the voters, we use an aggregation protocol. We aggregate the Borda scores given by all the voters to items and select the candidate with the highest total number of points. Therefore, the Borda score of an alternative is the sum of the points that the alternative receives in each ranking $r^1, \dots, r^m$.

Definition 1.1. The Borda score of item a with respect to input rankings $r^1, \dots, r^m$ is:

$$s(a) = \sum_{i=1}^{m} [n - r^i(a)] = mn - \sum_{i=1}^{m} r^i(a)$$

Notice that ordering alternatives in decreasing order of the Borda score is equivalent to sort by increasing order of the sum of ranks.

Now that we know how to represent preferences and rank alternatives, we want to gather these preferences. To do so, we use an incremental method consisting in retrieving the preferences by asking a query after another to voters. The Borda protocol requires that, for a particular voter, two items cannot receive the same rank. Our model uses pair-wise comparison of items to comply with that constraint. In addition, it is easier for people to decide between two objects rather than to rank the entire set (Balakrishnan and Chopra 2012). We suppose there is transitivity closure in preferences. For instance, if $a \succ b$, and $b \succ c$ then $a \succ c$. The request for the i-th voter's preference between candidates c_j and c_k is the voter-item-item query $q^i_{j,k}$.

The winning item is the one with the highest aggregated Borda score:

$$c^{\text{winner}} = \arg\max_{c\in C} s(c) = \arg\min_{c\in C} \sum_{i=1}^{m} r^i(c) \tag{1.3}$$

Example 2. Let's consider 3 items c_1, c_2, c_3 and 3 voters v_1, v_2, v_3. The ranking preferences are $r^1 = \langle c_1, c_2, c_3\rangle$, $r^2 = \langle c_2, c_1, c_3\rangle$ and $r^3 = \langle c_1, c_3, c_2\rangle$. Thus, c_1 gets $2+1+2=5$ points, c_2 gets $1+2+0=3$ points and c_3 $0+0+1=1$. The winning item is c_1.

Voters	$c_1 \succ c_2 \succ c_3$	$c_2 \succ c_1 \succ c_3$	$c_1 \succ c_3 \succ c_2$	$c_3 \succ c_1 \succ c_2$	$c_2 \succ c_3 \succ c_1$	$c_3 \succ c_2 \succ c_1$
v_1	0.2	0.1	0.3	0.2	0.1	0.1
v_2	0.3	0.1	0.1	0.2	0.2	0.1

Table 1.1: Voter permutation distributions for 2 voters and 3 items

1.3 Bayesian Elicitation Approach

1.3.1 General process

Thanks to the Borda voting rule, one can easily recommend a winning item for a group regarding the voters' preferences over a set of candidates. However, the collection of preferences may be tough, and might repel users. Fortunately, one can find an item that is likely to satisfy a group without knowing every user's preference (Konczak and Lang 2005). We now suppose that every ranking follows a probability distribution. We might derive the initial distributions from the users' history of preferences or from other users' preferences on the items in question. We use an incremental method to refine our knowledge of the group preferences. First, the program asks a user to consider two items and to select the one he prefers. Second, the system updates the distribution of the user based on his answer in a Bayesian way. Third, we compute the current best alternative c^*. If we meet the termination condition, the elicitation process stops and returns c^*. Otherwise, the system asks another question, determined with the help of different strategies detailed later in this report, and so forth until we find a winning item. To avoid over-soliciting the voters, we want to find a winning candidate in a minimal number of queries. Figure shows the general functioning of our framework. I will describe the different parts of the model in the following sections.

1.3.2 Probabilistic Voter Permutations Distribution Model

We are now interested in the scenarios where the rankings are not known precisely, but belong to some probabilistic model. The system maintains a probability distribution $\mathbb{P}(r^1, ..., r^n)$ over the rankings $r^1, ..., r^n$ of the voters. Thus, voter v_i's permutation distribution, denoted by r^i, becomes a discrete random variable, taking the values in the set of permutations of n elements. We assume that the voters preferences are independent, so $\mathbb{P}(r^1, ..., r^n) = \mathbb{P}(r^1)...\mathbb{P}(r^n)$. The probability of a pair-wise preference, as the event "$a \succ b$", can be extracted by aggregating the probabilities of the rankings where a comes before b:

$$\mathbb{P}(a \succ b) = \sum_{r \in R | a \succ_r b} \mathbb{P}(r).$$

A naïve representation of a ranking distribution is a table that directly specifies the value of $\mathbb{P}(r)$ for every permutation r. It requires the specification of $n!$ probability values, our model cannot handle a large number of candidates.

Example 3. In Table 1.1, $\mathbb{P}(r^2 = (c_1 \succ c_2 \succ c_3)) = 0.3$ and for user v_1, $\mathbb{P}(c_1 \succ_{r^1} c_2) = 0.2 + 0.3 + 0.2 = 0.7$.

The initial distributions are derived from the users' history of preferences or from other users' preferences on the items in question, then they are updated from the voters' answers along the elicitation process. Since we cannot compute the Borda score, we are interested in evaluating the alternatives regarding their expected Borda score.

Definition 1.2. The expected Borda score of item a with respect to input rankings $r^1, \ldots, r^m$ is:

$$\mathbb{E}[s(a)] = mn - \sum_{i=1}^{m} \mathbb{E}[r^i(a)]$$

Example 4. In Table 1.1, $\mathbb{E}[s(c_1)] = (0.2 + 0.3) * 2 + (0.1 + 0.1) * 1 + (0.3 + 0.1) * 2 + (0.2 + 0.2) * 1 + (0.1 + 0.2) * 0 + (0.1 + 0.1) * 0 = 2.4$

We identify the 'approximate winner' as the candidate that yields the highest expected Borda score under the current probability distribution of preference rankings:

$$c^* = \arg\max_{c \in C} \mathbb{E}[s(c)] \tag{1.4}$$

and $s^* = \mathbb{E}(c^*) = \max_{c \in C} \mathbb{E}[s(c)]$.

1.3.3 Finding a possible or a necessary winner

The Borda possible maximum (pmax) represents the possible highest score of an item based on the known preferences. Before the elicitation process begins: $\forall c_j \in \mathcal{C}\, \text{pmax}(c_j) = (n-1)m$. At every round, a voter states his preferences over two items. The least-liked candidate has its possible maximum decreased by 1. Formally:

Definition 1.3. Borda Possible Maximum

$$\forall c_j \in \mathcal{C}\, \text{pmax}(c_j) = m(n-1) - \sum_{\forall q^i_{j,k}} \text{pmax}(q^i_{j,k}) \text{ where: } \text{pmax}(q^i_{j,k}) = \begin{cases} 1 & \text{if } r^i(c_k) < r^i(c_j) \\ 0 & \text{otherwise} \end{cases} \tag{1.5}$$

The Borda possible minimum (pmin) represents the possible lowest score of an item based on the known preferences. Before the elicitation process begins: $\forall c_j \in \mathcal{C}\, \text{pmin}(c_j) = 0 * m$. At every round, a voter states his preferences over two items. The best-liked candidate has its possible minimum increased by 1. Formally:

Definition 1.4. Borda Possible Minimum

$$\forall c_j \in \mathcal{C}\, \text{pmin}(c_j) = \sum_{\forall q^i_{j,k}} \text{pmin}(q^i_{j,k}) \text{ where: } \text{pmin}(q^i_{j,k}) = \begin{cases} 1 & \text{if } r^i(c_k) > r^i(c_j) \\ 0 & \text{otherwise} \end{cases} \tag{1.6}$$

According to the paper of Konczak and Lang 2005, a Necessary Winner (NW) is a candidate which wins no matter what the future answers are, and thus no matter what the updates of the ratings distributions are. According to the article of Xia and Conitzer 2008), a possible winner is candidate which wins at least in one scenario regarding the current rating distributions. A sufficient (but not necessary) condition for an alternative $c_j \in \mathcal{C}$ to be a 'necessary winner' is:

$$\forall c_k \in \mathcal{C}, c_k \neq c_j, \text{pmin}(c_j) \geq \text{pmax}(c_k) \tag{1.7}$$

Example 5. Let's consider 3 items c_1, c_2, c_3, 3 voters v_1, v_2, v_3 and their concealed preferences: $r^1 = \langle c_1, c_2, c_3 \rangle$, $r^2 = \langle c_1, c_2, c_3 \rangle$ and $r^3 = \langle c_1, c_3, c_2 \rangle$.
Initial stage: Pmax = [6 6 6] ; Pmin = [0 0 0]
Query 1: $q^1_{1,2}$. Answer: $c_1 \succ_{r^1} c_2$. Pmax = [6 5 6] ; Pmin = [1 0 0]
Query 2: $q^1_{2,3}$. Answer: $c_2 \succ_{r^1} c_3$. Then by transitivity: $c_1 \succ_{r^1} c_3$. Pmax = [6 5 4] ; Pmin = [2 1 0]
Query 3: $q^3_{1,2}$. Answer: $c_1 \succ_{r^3} c_2$. Pmax = [6 4 4] ; Pmin = [3 1 0]
Query 4: $q^2_{1,2}$. Answer: $c_1 \succ_{r^2} c_2$. Pmax = [6 3 4] ; Pmin = [4 1 0]
c_1 is an approximate winner as $\text{pmin}(c_1) \geq \max(\text{pmax}(c_2), \text{pmax}(c_3))$.

1.3.4 The Expected Loss

We have implemented another stopping criterion for the elicitation process for finding a necessary or an approximate winner. The idea is to cut the communication with the users even more. Our algorithm stops when the expected loss is null.

Let $\mathbb{P}(r^1, ..., r^m)$ be the posterior distribution over the users' rankings, and c^* its associated optimal candidate. We need to estimate the regret or loss of stopping the elicitation process at a particular round. The loss $l(r^1, ..., r^m)$ is the regret of choosing c^* when the true users preferences are $(r^1, ..., r^m)$:

$$l(r^1, ..., r^m) = \max_{c \in C} s(c; r^1, ..., r^m) - s(c^*; r^1, ..., r^m) \tag{1.8}$$

Voters	$c_1 \succ c_2$	$c_2 \succ c_1$
v_1	0.8	0.2
v_2	0.4	0.6

Table 1.2: Voter permutation distributions for 2 voters and 2 items

Since we do not know the true preferences, but we know their distributions, we consider the expected loss $\mathbb{E}(l)$ (in a way analogous to Chajewska, Koller, and Parr 2000 that considered Bayesian elicitation in influence diagrams) that quantifies how far we are from the true optimum in expectation:

The Expected Loss (EL) is:

$$\mathbb{E}_{r^1\sim\mathbb{P}(r^1),\dots,r^m\sim\mathbb{P}(r^m)}[l(r^1,...,r^m)] = [\sum_{r^1}...\sum_{r^m}\mathbb{P}(r^1)...\mathbb{P}(r^m)\max_{c\in C}s(c;r^1,...,r^m)] - s^* \tag{1.9}$$

Example 6. Let's consider two items and two voters whose rankings distributions are in Table 1.2. The expected Borda scores are: $\mathbb{E}(s(c_1)) = \frac{0.8+0.4}{2} = 0.6$ and $\mathbb{E}(s(c_2)) = \frac{0.2+0.6}{2} = 0.4$. c_1 is the local winner. $\mathbb{E}(\mathit{loss}) = 0.8*0.4*(2-2) + 0.8*0.6*(1-1) + 0.2*0.4*(1-1) + 0.2*0.6*(2-0) = 0.24$

One can approximate expected loss thanks to Monte Carlo sampling. To do so, at each round, for every user, we sample a ranking permutation based on the distribution over the rankings. We determine the expected Borda scores of the items regarding the selected permutations. The local loss equals the difference between the highest expected Borda score of these sampled preferences and the highest expected Borda score of the current round preferences without sampling permutations. The approximate expected loss is the sum of the local losses divided by the sample size.

When the expected loss is null, the algorithm returns the item having the highest expected Borda score.

Let's introduce some notations:

- N the number of samples needed to approximate the expected loss.
- l the worst loss case. $l = (n-1)*m - s^*$. $(n-1)*m$ is the best Borda score an item can receive and s^* is the expected Borda score of the winning item of the current state.
- ϵ the desired threshold for expected loss. The parameter ϵ controls the error tolerated in our estimate. It is often worth 5% or 1%.
- δ the confidence parameter. It controls the confidence we want to have in our estimate. It is often worth 5% or 1%.

With the Chebyshev's inequality, we get:

$$N \geq \frac{l^2}{\epsilon^2\delta} \tag{1.10}$$

Proof. Lower bound of the sample size for the expected loss estimation
Let EL denote the expected loss.

$$\forall i \in \{1,...,N\} EL_i = \mathbb{E}_{r^{1,i}\sim\mathbb{P}(r^{1,i}),\dots,r^{m,i}\sim\mathbb{P}(r^{m,i})}[l(r^{1,i},...,r^{m,i})]$$

The EL_i are independent and identically distributed (i.i.d.). $\mathbb{E}(EL_i) = \mu$, $\mathbb{V}(EL_i) = \sigma^2$.
Our estimate of the expected loss could be $A_N = \frac{1}{N}\sum_{i=1}^{N} EL_i$, $\mathbb{E}(A_N) = \mu$, $\mathbb{V}(A_N) = \frac{\sigma^2}{N}$.
We want to determine the sample size N so that our estimate A_N will be within $\pm\epsilon$ (in absolute terms) of the true value with probability at least δ.

$$\mathbb{P}(EL - \epsilon < A_N < EL + \epsilon) \geq 1 - \delta \Leftrightarrow \mathbb{P}(|A_N - EL| \geq \epsilon) \leq \delta$$

Using the Chebyshev's inequality, we find:

$$\mathbb{P}(|A_N - EL| \geq \epsilon) \leq \frac{\mathbb{V}(A_N)}{\epsilon^2} = \frac{\sigma^2}{N\epsilon^2} \leq \frac{l^2}{N\epsilon^2}$$

$\mathbb{E}_{r^1\sim\mathbb{P}(r^1),...,r^m\sim\mathbb{P}(r^m)}[l(r^1,...,r^m)] \leq l$ so $\sigma^2 \leq l^2$
Then $\delta \geq \frac{l^2}{N\epsilon^2} \Leftrightarrow N \geq \frac{l^2}{\delta\epsilon^2}$. □

1.3.5 The item winning probability

To calculate the items winning probabilities, one has to add the probabilities of winning for each item for the $(n!)^m$ combinations of rankings. To prevent a high completion time, we use Monte Carlo sampling to estimate a winner at every round. For each user v_i, a permutation is sampled out of all the user permutations. A winner is then determined thanks to the Borda voting protocol. This process is repeated γ times. Finally, the winning probability of an item is the number of times this item has won divided by γ. For further details, please refer to the function win_proba here below.

```python
import operator
import numpy as np
from numpy import random as rd
from borda_voting_protocol import borda


def win_proba(v, c, vc, gamma, distrib):
    """
    Return the estimated winning probabilities of the candidates.

    Parameters
    ----------
    v : ARRAY
        The set of voters.
    c : ARRAY
        The set of candidate items.
    vc : ARRAY
        The set of possible permutations.
    gamma : INT
        The sample size.
    distrib : ARRAY
        The current rankings probability distributions.

    Returns an array.
    -------
    pr_win : ARRAY
        The winning proba array.

    """
    # Initialize.
    pr_win = np.zeros(len(c))
    # Loop on the sample size.
    for _ in range(gamma):
        rd_permut = []
        # Loop on the number of voters.
        for i in range(len(v)):
            # Draw a permutation for voter i regarding distrib[i].
            rd_permut.append(vc[rd.choice(len(vc), 1, p=distrib[i])])
        all_rd_permut = np.array(rd_permut).flatten().reshape((len(v), len(c)))

        # Compute the items expected Borda scores regarding the drawn rankings.
        local_borda_scores = borda(all_rd_permut)
        # The local winner is the item with the highest expected Borda score.
        local_winner = int(max(local_borda_scores.items(),
                               key=operator.itemgetter(1))[0])
        # Add 1 to the local winning candidate.
        pr_win[local_winner] += 1

    # Divide the number of times the items won by the number of iterations.
    pr_win /= gamma
    # Return the winning probabilities array.
    return pr_win
```

1.3.6 Information Gain Heuristic for Borda Voting

We consider different exploration strategies to decide which query to ask at any stage of the elicitation process. The strategies aim at reducing uncertainty over the voters preferences in such a way to improve the quality of the approximated winner.

The first heuristic introduced is the Information Gain heuristic for Borda voting (IGB). To implement it, we have leaned on the article of Dery et al. 2015. The goal of this strategy is to maximize the information in terms of entropy at each stage, which amounts to reduce the entropy to its minimum value. First, we compute the information gain of every answer $q^i_{c_j \succ c_k}$. It is the difference between the prior entropy and the posterior entropy given this answer to the probability of winning for an item. The next selected query is the one maximizing the weighted information gain. In case of equality, we choose the item with the smallest ID. Given the item winning probabilities array PrWin, the entropy function is:

$$H(PrWin) = -\sum_{j=1}^{n} PrWin[j]log(PrWin[j]) \tag{1.11}$$

The Information Gain (IG) is:

$$IG(q^i_{c_j \succ c_k}) = H(PrWin) - H(PrWin|q^i_{c_j \succ c_k}) \tag{1.12}$$

The Weighted Information Gain (WIG) is:

$$WIG(q^i_{j,k}) = IG(q^i_{c_j \succ c_k})\mathbb{P}(q^i_{c_j \succ c_k}) + IG(q^i_{c_j \prec c_k})\mathbb{P}(q^i_{c_j \prec c_k}) \tag{1.13}$$

The chosen query is one that maximizes the WIG:

$$q^{next} = \underset{v_i \in \mathcal{V}, (c_j, c_k) \in \mathcal{C}^2}{\arg\max} \ WIG(q^i_{j,k}) \tag{1.14}$$

In the end, IGB aims at maximizing the decrease in the entropy induced by a query.

1.3.7 Highest Expected Score Heuristic for Borda Voting

The second heuristic introduced is the Expected Score heuristic for Borda (ESB). To implement it, we have leaned again on the paper of Dery et al. 2015. This strategy relies on the hypothesis that it is better to select a query $q^i_{j,k}$ where one item c_j or c_k is expected to win. If we pick an item that has significant chances to win, the possible minimum will increase faster, and a necessary winner will stand out quickly. Thus, ESB selects the queries containing the item with the highest winning probability. The Expected Maximum (EM) of an answer $q^i_{c_j \succ c_k}$ represents the difference between the highest winning probability given voter v_i prefers c_j over c_k and the highest winning probability without asking any query:

$$EM(q^i_{c_j \succ c_k}) = \max(PrWin|q^i_{c_j \succ c_k}) - \max(PrWin) \tag{1.15}$$

The Weighted Expected Maximum (WEM) is:

$$WEM(q^i_{j,k}) = EM(q^i_{c_j \succ c_k})\mathbb{P}(q^i_{c_j \succ c_k}) + EM(q^i_{c_j \prec c_k})\mathbb{P}(q^i_{c_j \prec c_k}) \tag{1.16}$$

The chosen query is the one that maximizes the WEM:

$$q^{next} = \underset{v_i \in \mathcal{V}, (c_j, c_k) \in \mathcal{C}^2}{\arg\max} \ WEM(q^i_{j,k}) \tag{1.17}$$

In the end, ESB aims at maximizing the increase in the highest winning probability induced by a query.

Voters	$c_1 \succ c_2$	$c_2 \succ c_1$
v_1	1	0
v_2	0.6	0.4
v_3	0.6	0.4

Table 1.3: Voter permutation distributions for 3 voters and 3 items

1.3.8 Expected Value of Information Heuristic for Borda Voting

The third strategy adopts myopic Expected Value Of Information (EVOI), that has been shown to be very effective in single-user preference elicitation (Viappiani and Boutilier 2020). To implement it, we have leaned on the theory of the value of information, discussed in the paper of Howard 1965. We also inspired from the paper of Braziunas and Boutilier 2008. This strategy relies on the same hypothesis as the ESB heuristic: it is better to select a query $q^i_{j,k}$ where one item c_j or c_k is expected to win. If we pick an item that has significant chances to win, the possible minimum will increase faster, and a necessary winner will stand out quickly. Thus, EVOI selects the queries containing the item with the highest expected Borda score. The Expected Value (EV) of an answer $q^i_{c_j \succ c_k}$ represents the difference between the highest expected Borda score given voter v_i prefers c_j over c_k and the highest expected Borda score without asking any query:

$$EV(q^i_{c_j \succ c_k}) = \max_{c \in \mathcal{C}} \mathbb{E}(s(c)|q^i_{c_j \succ c_k}) - s^* \tag{1.18}$$

The EVOI is:

$$EVOI(q^i_{j,k}) = \mathbb{P}(q^i_{c_j \succ c_k})EV(q^i_{c_j \succ c_k}) + \mathbb{P}(q^i_{c_k \succ c_j})EV(q^i_{c_j \prec c_k}) \tag{1.19}$$

The chosen query is the one that maximizes the EVOI:

$$q^{next} = \operatorname*{arg\,max}_{v_i \in \mathcal{V}, (c_j, c_k) \in \mathcal{C}^2} EVOI(q^i_{j,k}) \tag{1.20}$$

In the end, EVOI aims at maximizing the increase in the highest expected Borda score induced by a query.

1.3.9 EVOI and IGB mixed

There are many rounds in the elicitation problem when the weighted expected maximum, the information gain, or the expected value of information are null. This is because of the myopic approach we chose. Sometimes, asking another query does not increase the local information the program has. Thus, some queries are randomly chosen, and that might affect the communication cost. As the winning probability and the expected value grow in the same way, they are null at the same time. Contrary to ESB and EVOI heuristics, the IGB heuristic does not aim at maximizing the increase in the highest winning probability or the expected Borda score. It selects the query that decreases the local entropy the most. Therefore, the EVOI and the WEM are simultaneously null, but not necessarily the WIG. That is why we combine the EVOI and the IGB heuristics. At each round, we select the query with the highest expected value of information. If its value is null, we select the query with the highest weighted information gain, and so on.

Example 7. Let's consider 3 voters, 3 items and the current distribution over rankings in Table 1.3. Let's suppose we have already asked $q^1_{1,2}$, and taken the answer $q^1_{c_1 \succ c_2}$ into account.
$EVOI(q^2_{1,2}) = 0.6 * (1 + 1 + 0.6) + 0.4 * (1 + 0.6) - (1 + 0.6 + 0.6) = 0$
$EVOI(q^3_{1,2}) = 0.6 * (1 + 0.6 + 1) + 0.4 * (1 + 0.6) - (1 + 0.6 + 0.6) = 0$
Therefore, all the queries have a null EVOI, and our program calculates the weighted information gain of these queries ($q^2_{1,2}$ or $q^3_{1,2}$) to continue the elicitation process.

Chapter 2

Materials and Methods

The goal of my internship was to create a food recommender system. It had to suggest dishes to groups of people based on their individual preferences, with a minimal amount of queries. The first part of our work was to implement existing heuristics, compare them, and create a model using them to be able to limit the number of questions asked to users. To test these heuristics, we focused on:

1. The communication cost - lower communication costs are desirable. This is done by logging the number of queries asked and the percentage of dataset queried. The upper bound to the queries amount is the amount of queries a naïve voting center would have asked: $\frac{n*m*(n-1)}{2}$. The percentage of dataset queried is: $1 - \frac{2*queries}{n*m*(n-1)}$.

2. The runtime per query. In the long run, the algorithm has to make an instant recommendation, so its runtime must be quick.

To test these criteria, we used the Sushi Dataset (Kamishima, Kazawa, and Akaho 2005). It contains 5,000 rankings on ten different sushi. I implemented the heuristics with the programming language Python; I handled my code with the version-control system Git, and we ran my experiments on the server of AgroParisTech.

This part helped us compute the optimal strategy regarding the models described above. After that, we wanted to assess the accuracy of the food recommendations made. To do so, we made up a food preferences dataset that we have called the CROUS dataset. This database contains 130 rankings over five starters, five courses and five desserts.

In all domains, we explored scenarios that included a group of 4 to 15 users, that were required to choose among 5-6 items. We assumed these objects were the top-ranked candidates returned by a previous group recommender system.

2.1 The programming environment

During my internship, I worked on the Linux distribution Ubuntu 20.04.1 LTS (Focal Fossa). I programmed on Spyder 4.1.5, a Python Integrated Development Environment (IDE) with advanced editing, interactive testing, debugging and introspection features. I programmed in Python, version 3.7.6, and complied with the PEP8 coding conventions (Rossum, Warsaw, and Coghlan 2001). To handle the evolution of my code, I used Git, a free and open source distributed version control system. One can access my Git repository via the following link: `https://forgemia.inra.fr/stephane.dervaux/prefelicitgroup`. To compute the results obtained with the heuristics, we ran my code on the servers of the INRAE. I wrote approximately 2000 lines of Python code for this project.

2.2 The sushi dataset

We examined a scenario of users who were required to decide between ten types of sushi. The Sushi dataset (Kamishima, Kazawa, and Akaho 2005) contains 5,000 preference rankings over ten kinds of sushi. We derived six different random matrices of the size of ten users $*$ six sushis. These six sushis were chosen randomly among the ten items ranked in the dataset. To create an initial permutation probability distribution, we aggregated the number of appearances of each permutation in the training set and divided it by the total number of voters. Then, we normalized the initial distribution, so that there was no permutation receiving a null probability. Thus the initial permutation distribution was equal for all voters. Then, we randomly selected users in the sushi dataset and extracted their rankings over the six previous sushis. As users answered more queries based on the selected preferences, the distributions were updated for each voter. Over time, a unique permutation distribution pattern emerged for each user.

2.3 The CROUS dataset

After assessing the heuristics and making up my model, we wanted to test the quality of the recommendations returned by the algorithm. To do so, we examined a scenario of users who were required to decide between courses. The CROUS dataset contains 130 preference rankings over five starters, five main courses and five desserts. These 15 dishes are top-ranked items from the menu of the CROUS of Versailles, a college refectory near Paris. We created this dataset by ourselves through a real-life experiment. We put together a dossier addressed to the Research Ethics Committee of Paris-Saclay (Comité d'Ethique pour la Recherche (CER)) to ask the permission for carrying out an online nutrition experiment. After that, we created an online questionnaire on LimeSurvey (Caillat 2020). 130 French-Speaking persons over the age of majority - 75 women and 55 men - completed it. They had to rank five starters, five main dishes and five desserts in order of preference. If they did not want to eat one or more items - for taste, allergies or belief motives - they should not rank it. Over a second phase, 18 participants out of 130 gathered in four virtual tables of four or five persons on WhatsApp. Each chat group had to select one starter, one main dish and one dessert after discussions.

For every virtual table, we calculated the Borda score of the food items and compared the dishes suggesting by following the Borda voting protocol with the dishes selected by the tables. This stage allowed us to assess the relevance of the Borda aggregation method. Normally, with Borda, all the items should be ranked, which was not the case in our questionnaire. To tackle this issue, we attributed negative scores to items that participants did not want to eat. After that, for every table, we checked if there were excluded items. If there were, we would consider the rankings without these missing items to create an initial permutation distribution. Thus, the preferences and permutations would be only on the non-excluded items. We aggregated the number of appearances of each permutation in the training set and divided it by the total number of voters that had ranked all the items in question. Then, we normalized the initial distribution, so that there is no permutation receiving a null probability. Thus the initial permutation distribution is equal for all voters. As more queries are answered based on the selected preferences, the distributions are updated for each voter. Over time, a unique permutation distribution pattern emerges for each user. At the end of the elicitation process, an item was recommended.

Chapter 3

Results and Discussion

3.1 Results

3.1.1 Performances of the heuristics

We compared four heuristics, IGB, ESB, EVOI and a mix of EVOI and IGB, which determined a necessary winner under the Borda voting protocol. In this section, we gauge the heuristics performance. Since the heuristics use sampling, we ran each experiment 25 times. The γ parameter in the Item Sampling algorithm (the algorithm that sets the item winning probabilities) was set to 300 (Dery et al. 2015). Initially, the confidence parameter δ was fixed to 0.05, and the desired threshold for the expected loss ϵ to 0.01. However, the runtimes were very long, so we set the sample size for the estimate of the expected loss N to 1000, and not $\lceil \frac{x^2}{\epsilon^2 \delta} \rceil$ anymore.

Figures 3.1 and 3.2 display a comparison of the performance between the heuristics on the sushi dataset. Axis x in both figures presents a varying size of voters $*$ items. Axis y in figure 3.1 represents the percentage of cut in the communication; high reduction in the communication cost, that being high percentages in this figure, is desirable. Axis y in figure 3.2 represents the number of queries asked; low numbers of queries are desirable. EVOI+IGB is at most around 15% better than ESB with a 95% confidence interval according to a t-test. EVOI+IGB reduces communication up to 57% in total.

Runtime on the sushi dataset is presented in figure 3.3. Axis x presents a varying size of voters $*$ items, and axis y presents the runtime per query in seconds. EVOI has a linear runtime while IGB, ESB and EVOI+IGB have an exponential runtime. There is no significant difference between the runtimes of IGB, ESB and EVOI+IGB.

Figure 3.4 shows the evolution of the expected loss versus the number of queries asked for 10 voters and 6 sushis. EVOI+IGB reduces the expected loss more quickly than IGB, ESB and EVOI. For example, in Table 3.1, I reported the number of queries asked for getting to some expected loss values for all the heuristics, 6 sushi and 10 voters. EVOI+IGB is way faster for diminishing the expected loss than the other heuristics. One can see that stopping the elicitation process when the expected loss attains a determined value and not 0 anymore, for instance, 5% or 1% of its initial value, might really lessen the communication cost. For example, for 6 sushi, 10 voters, the EVOI+IGB heuristic and an expected loss worth 1% of its initial value, the program may ask 16 fewer queries than when it waits until the expected loss is null.

3.1.2 Relevance of the Borda aggregation protocol

No matter the heuristic or the dataset size, the elicitation framework we implemented always suggested the item with the highest Borda score. The only difference was the number of queries asked before the algorithm converged. Therefore, to test the precision of the recommendations made by our program, we only had to assess the precision of the Borda voting protocol. We compared the items selected by the

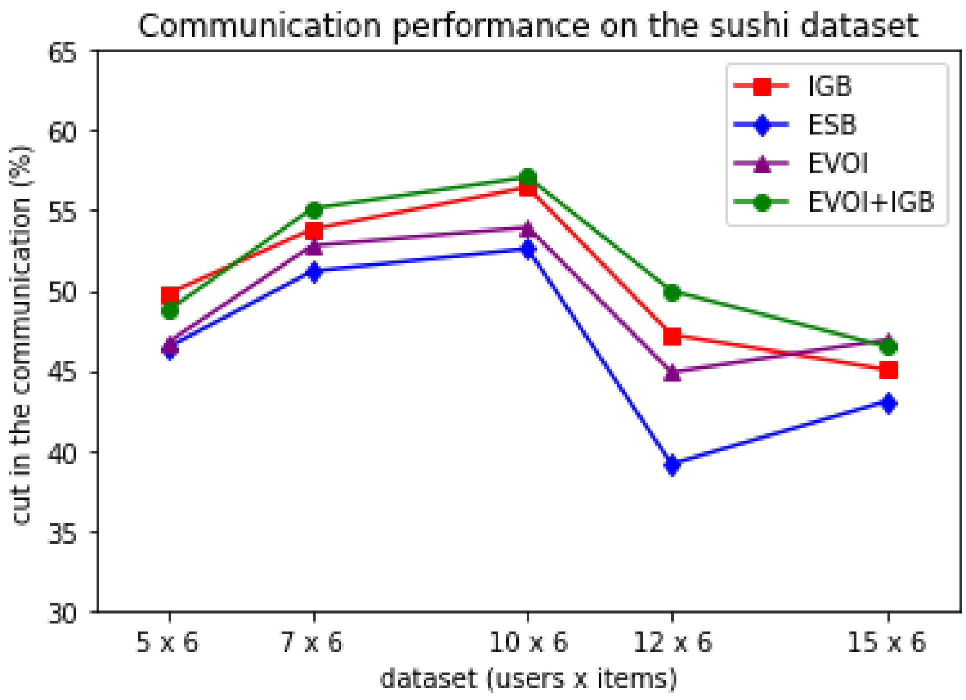


Figure 3.1: Communication performance on the sushi dataset (expected loss = 0)

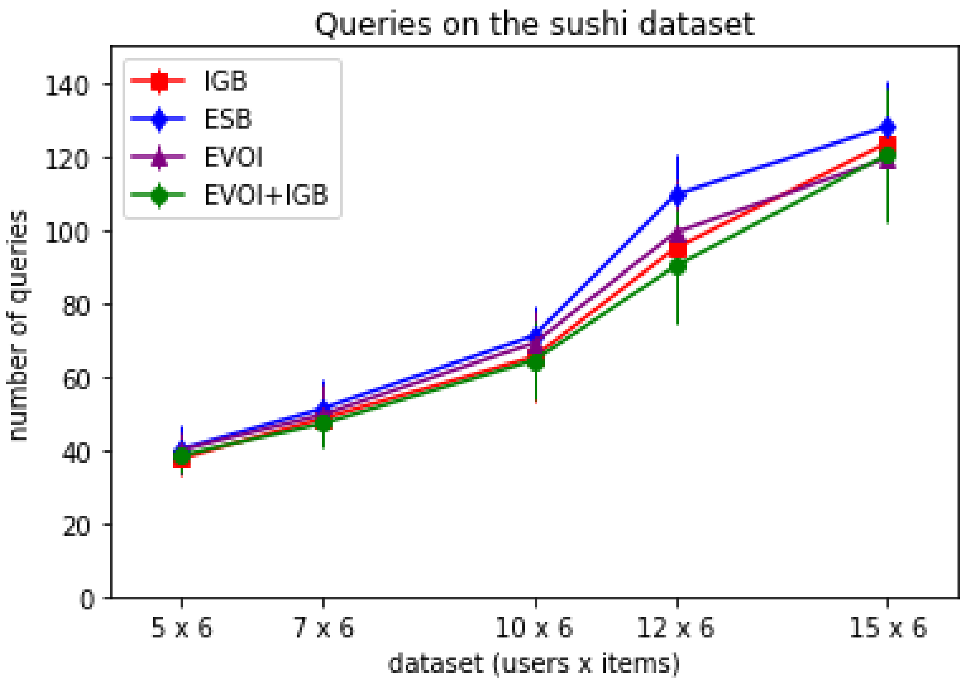


Figure 3.2: Number of queries on the sushi dataset (expected loss = 0)

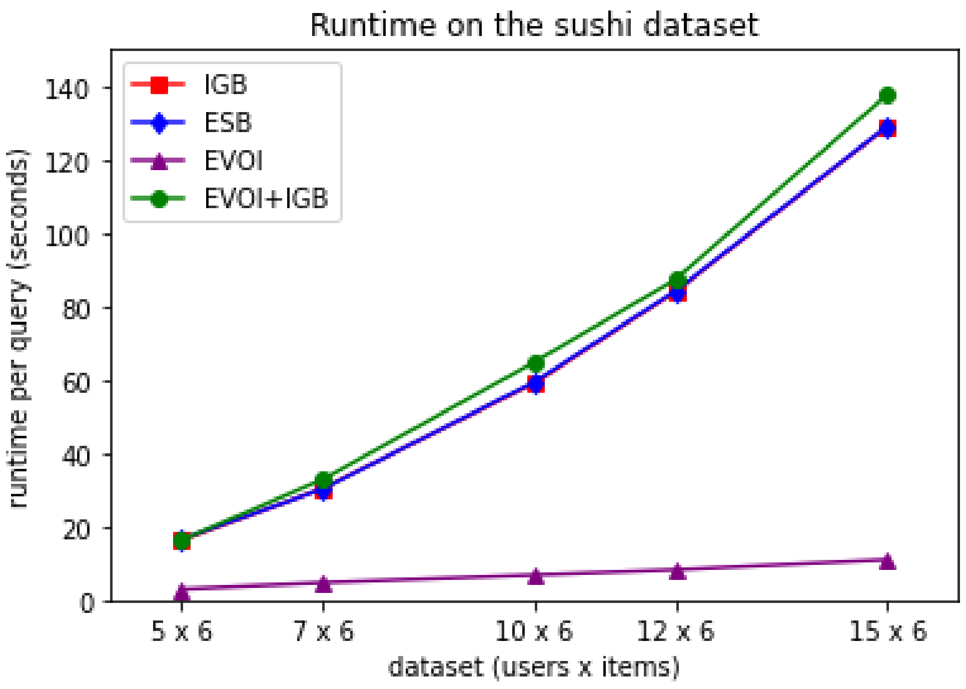


Figure 3.3: Runtimes on the sushi dataset (expected loss = 0)

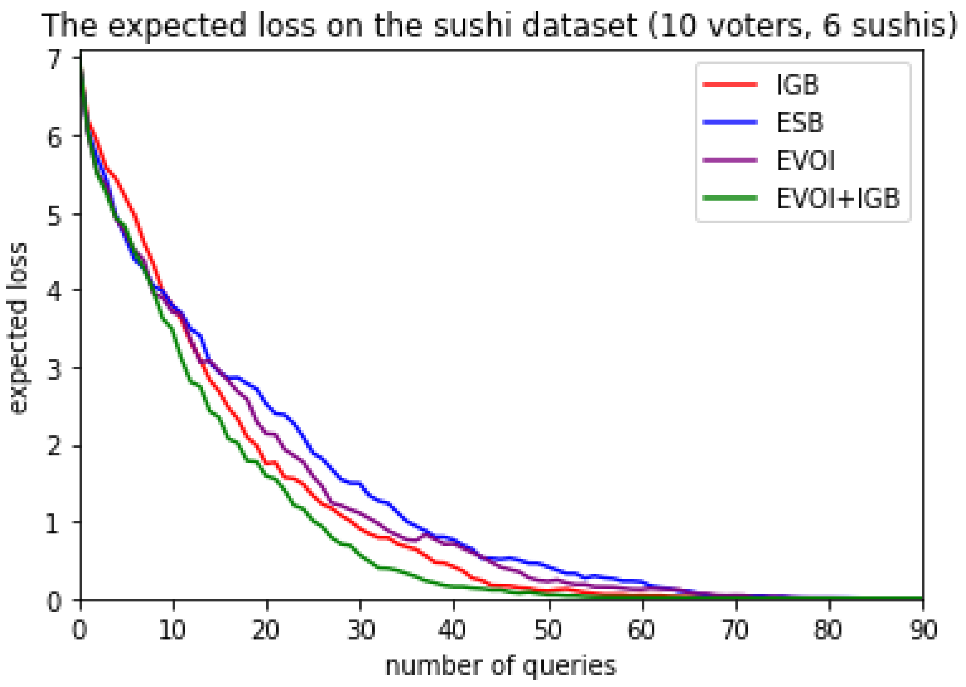


Figure 3.4: Expected losses on the sushi dataset (10 voters, expected loss = 0)

Cut in the expected loss	EVOI+IGB	IGB	EVOI	ESB
$\mathbb{E}(loss) \leq 50\%\mathbb{E}_{query=0}(loss)$	11	13	13	13
$\mathbb{E}(loss) \leq 5\%\mathbb{E}_{query=0}(loss)$	36	43	48	53
$\mathbb{E}(loss) \leq 1\%\mathbb{E}_{query=0}(loss)$	48	56	67	67
$\mathbb{E}(loss) = 0$	64	65	69	71

Table 3.1: Number of queries to get to various percentages of initial expected loss for 6 sushi and 10 voters

participants of the WhatsApp experiment with the items having the highest Borda scores. 4 items out of 12 (33%) were the winning elements for both the Borda protocol and the WhatsApp experiment. 2 (17%) items selected by the tables had the second best Borda score; 5 (42%) the third best Borda score and 1 the fourth best Borda score (8%). In summary, in this experiment, the Borda voting protocol matched reality half the time and got the wrong suggestions half the time too. When the Borda model matched reality (6 cases out of 12), the variances of the Borda scores divided by the numbers of eaters was high (around 5). The cases (6 out of 12) where there were substantial mistakes had lower variances of the Borda scores divided by the numbers of eaters (around 2). Neither the Borda voting protocol nor the real-world tables suggested items excluded by at least one eater on the table.

3.2 Discussion

Our experiments on the sushi dataset show the superiority of the heuristic EVOI+IGB regarding the communication cost. It reduces the number of queries up to 57%. Even if IGB is better than EVOI, itself better than ESB, we tested the heuristic EVOI followed by IGB when the EVOI is null, rather than the opposite (IGB+EVOI) for two reasons. First, the elicitation process is running faster with EVOI than with the other heuristics, which is important for a real-time recommender system. Second, EVOI is more precise than IGB and ESB because it does not use Monte Carlo sampling for computing the expected Borda scores, contrary to IGB and ESB that calculate approximate item winning probabilities. Finally, EVOI+IGB turns out to be even better than IGB because it reduces the randomness of the questions. There are many situations in which asking a new query does not improve the expected value of information when using the EVOI heuristic, or the item winning probabilities vector when using the IGB heuristic, for example. In these cases, the algorithm selects random unasked queries, and this damages the efficiency of the heuristics. Using IGB when the EVOI is null avoids many random questions.

IGB aims at reducing the entropy vector. The minimum entropy is reached when the candidate winning probability vector (PrWin) becomes an indicator vector (an item has a winning probability of one, and the other of 0). In this situation, an approximate winner is revealed. The pre-eminence of IGB over EVOI and ESB proves that a reduction in the entropy, combined with the stopping criterion of null expected loss, brings the program closer to finding a winning item. While EVOI and IGB are looking for candidates whose possible minimums are greater than the possible maximums of the other items, IGB is trying to reduce the overall uncertainty, and thus works efficiently when combined with the expected loss. On the contrary, IGB and EVOI select the queries that potentially provide the highest increase in the maximum entry in the probability vector or in the expected Borda scores vector. That approach turns out to be more efficient if the algorithm stops when it finds a possible winner (Dery et al. 2015). However, improving the winning probabilities or the expected Borda scores will not automatically decrease the expected loss; this explains why IGB and ESB will prove to be more costly of queries.

Our experiments on the CROUS dataset imply that the Borda aggregation rule is efficient when preferences over items are very marked. For example, Borda never suggested dishes that were excluded by at least one eater at the table. When a great majority of the participants strongly preferred one item over the others (e.g. brownie, see in the appendices), Borda always recommended this item. However, when the group preferences were not clear, and thus the variance of the Borda scores divided by the number of voters was low, Borda committed mistakes in its recommendations. We looked closely at the WhatsApp chats, in particular at the cases for which there were huge differences between Borda and reality. We realized that, once participants excluded really disliked dishes, and except for the cases everyone agrees over an item, people did not state clearly their preferences. They would rather try to make everyone more or less happy than openly express their tastes. Sometimes, people might even agree over a particular dish regarding their hidden personal rankings, but the group would rather choose an item that the voters think others prefer. In that case, Borda was mistaken.

The major disadvantage of our framework might be its lack of scalability, since we store distributions over permutations for every voter in an array, and we update them at every round. Thus, our algorithm is not suitable for many candidate items. Another drawback of our model is its lack of precision in its recommendations. Indeed, Borda might not operate well when preferences are unmarked.

Conclusion

The aim of this internship report is to present the Bayesian group recommender system based on voting theory with applications to food choices we have implemented. Our algorithm extracts personal preferences with a minimal communication cost and suggests dishes to groups of eaters. Our framework gathers information to initialize the rankings probability distributions over candidate items. Then, it computes locally optimal queries. Users' answers enable the system to update the rankings distributions in a Bayesian way, increase possible minimums, and decrease possible maximums and the expected loss. We use the Borda voting rule to compute the local best recommendations. Our experiments on the Sushi dataset show the superiority of the EVOI+IGB heuristic combined with a stopping criterion based on the expected loss to compute queries. On the example of 10 voters, 6 sushis, with a final expected loss below 5% of its initial value, this heuristic even cuts communication up to 75%. Our experiments on the CROUS dataset show the precision of the highest Borda score aggregation rule when preferences are marked, but its lack of accuracy otherwise. A major drawback of our model is its lack of scalability since the rankings distributions are based upon permutations of items Bayesian updated at every round.

One can imagine many improvements in our model. First, as choosing the item with the highest Borda score was not accurate in all the scenarios we have considered, one may contemplate other aggregation methods, in particular, one may add fairness constraints to the set of winning items. For example, we may select the item with the smaller worst rank. In our model, we only use pair wise queries, but one can implement other types of questions. For example, the system could ask 'Among these five candidates, which one do you prefer? Which one do you dislike the most?'. The program often selects questions randomly because the EVOI or the WIG are null. One may introduce new heuristics to cope with that issue, for example, the paper of Lu and Boutilier 2020 uses the minimax regret to compute queries and quickly reduce regret. Storing the probability distributions of all the rankings for all the voters prevents our framework from being scalable. One may imagine a way of considering only a little part of the rankings distributions during the elicitation process. Our model only takes account of the pleasure to eat a dish. One may also consider other attributes, such as the nutritional value, the hunger level or the influence of potential eating companions (empathetic decision making).

My internship at the MIA laboratory of AgroParisTech was very enriching. It has been a whole new experience in the world of research for me. I took part in a multidisciplinary project, mixing statistics, computer science and nutrition. My supervisors and I submitted two publications. First, we published an abstract for the Journées Francophones de Nutrition (JFN), which is a 'digital event from 25 to 27 November 2020 bringing together researchers, doctors, dietitians, students, experts and industrialists to discuss the fundamentals of nutrition' (*JFN website* 2020). If our summary (Darcel et al. 2020) is selected, we will present our work at this symposium. Second, we published an abstract for the From Multiple Criteria Decision Aid to Preference Learning (DA2PL) 2020 workshop, which will take place on November 5 and 6 (Viappiani, Caillat, et al. 2020). This symposium aims to 'bring together researchers from decision analysis and machine learning' (*DA2PL website* 2020). My tutors offered me to continue my internship with a thesis, but I declined their proposal because I don't want to engage in a thesis at the moment. However, the internship I did with them made me want to work on Bayesian preference elicitation.

List of Figures

3.1 Communication performance on the sushi dataset (expected loss = 0) 24
3.2 Number of queries on the sushi dataset (expected loss = 0) 24
3.3 Runtimes on the sushi dataset (expected loss = 0) 25
3.4 Expected losses on the sushi dataset (10 voters, expected loss = 0) 25

B.1 Starter dishes survey 42
B.2 Main dishes survey 42
B.3 Desserts survey 42

C.1 Fragment of the Sushi database 44
C.2 Fragment of the CROUS database (desserts) 44

List of Tables

1.1 Voter permutation distributions for 2 voters and 3 items 15
1.2 Voter permutation distributions for 2 voters and 2 items 17
1.3 Voter permutation distributions for 3 voters and 3 items 20

3.1 Number of queries to get to various percentages of initial expected loss for 6 sushi and 10 voters 26

Bibliography

Publications

Howard, R. (1965). “Information Value Theory”. In: *IEEE Transactions on systems science and cybernetics*, pp. 22–26.

Chajewska, U., D. Koller, and R. Parr (2000). “Making Rational Decisions using Adaptive Utility Elicitation.” In: *American Association for Artificial Intelligence.* URL: https://www.aaai.org/.

Konczak, K. and J. Lang (2005). “Voting procedures with incomplete preferences.” In: *Proceedings of the multi- disciplinary workshop on advances in preference handling in the 19th international joint conference on artificial intelligence (IJCAI), vol 20, Edinburgh, Scotland.*

Chevaleyre, Y. et al. (2007). “A Short Introduction to Computational Social Choice”. In: *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science*, pp. 51–69.

Braziunas, D. and C. Boutilier (2008). “Elicitation of Factored Utilities.” In: *AI Magazine*, pp. 79–92.

Xia, L. and V. Conitzer (2008). “Determining Possible and Necessary Winners under Common Voting Rules Given Partial Orders”. In: *Association for the Advancement of Artificial Intelligence*, pp. 25–67.

Balakrishnan, S. and S. Chopra (2012). “Two of a kind or the ratings game? Adaptive pairwise preferences and latent factor models.” In: *Front Comput Sci*, pp. 197–208.

Dery, L. et al. (2015). “Preference Elicitation for Group Decisions Using the Borda Voting Rule.” In: *Springer Science+Business Media*, pp. 1015–1033.

Felfernig, A. et al. (2018). “Group Recommender Systems. An Introduction.” In: *Springer.*

Darcel, N. et al. (2020). “Conception, mise en œuvre et évaluation d’un algorithme de recommandations alimentaires pour des groupes”. In: *Journées Francophones de Nutrition 2020.*

Lu, T. and C. Boutilier (2020). “Preference elicitation and robust winner determination for single- and multi-winner social choice”. In: *Artificial Intelligence, Volume 279.*

Viappiani, P. and C. Boutilier (2020). “On the equivalence of optimal recommendation sets and myopically optimal query sets”. In: *Artificial Intelligence, 286, 103328.*

Viappiani, P., M. Caillat, et al. (2020). “Bayesian Vote Elicitation for Group Recommendations”. In: *DA2PL 2020.*

Other Sources

Rossum, G. van, B. Warsaw, and N. Coghlan (2001). *PEP8 - Style Guide for Python Code.* URL: https://www.python.org/dev/peps/pep-0008/.

Kamishima, T., H. Kazawa, and S. Akaho (2005). *Sushi Preference Data Sets.* URL: http://www.kamishima.net/sushi/.

AgroParisTech website (2020). URL: http://www2.agroparistech.fr/.

Caillat, M. (2020). URL: https://enquetes.agroparistech.fr/limesurvey/index.php/667213?lang=fr.

DA2PL website (2020). URL: https://event.unitn.it/da2pl2020/.

Groupe SEB (2020). URL: https://www.groupeseb.com/en.

JFN website (2020). URL: https://www.lesjfn.fr/.

MIA webpage (2020). URL: http://www2.agroparistech.fr/Mia-Mathematiques-et-informatique.html.

PNCA webpage (2020). URL: http://www2.agroparistech.fr/PNCA-Physiologie-de-la-nutrition.html.

Université Paris-Saclay webpage (2020). URL: https://www.universite-paris-saclay.fr/en.

Appendix A

Our publications

Bayesian Vote Elicitation for Group Recommendations

Maeva Caillat [1] and **Nicolas Darcel** [2] and **Cristina Manfredotti** [3] and **Paolo Viappiani** [4]

Abstract. Elicitation of preferences is a critical task in modern application of voting protocols such as group recommender systems.

This paper introduces a Bayesian elicitation paradigm for social choice. The system maintains a probability distribution over the preferences (rankings) of the voters. At each step the system asks the question to one of the voters, and the distribution is conditioned on the response. We consider strategies to pick the next question based on value of information, conditional entropy, and a mix of these two notions.

We develop this idea focusing on scoring rules and compare different elicitation strategies in the case of Borda rule.

1 Introduction

Aggregation of preferences is an important task studied in social choice [5] and as well in the more recent research field of group recommender systems [1].

In modern applications of social choice and as well in recommender systems, it cannot be given for granted that preferences are readily available. Realizing this fact, a number of researchers considered voting procedures with incomplete preferences and as well elicitation procedures for voting; to our knowledge the first of these work is the one by Konczak and Lang [3] that introduced the notion of possible winners and necessary winners. Previous approaches considered robust winner approximation and elicitation with the minimax regret criterion [4]. A key insight is that, often, not all user preferences are needed in order to reach a "winning item".

In our work, we consider a Bayesian approach, as Dery et al. [6], but our soulution differs in a number of key points. We provide a principled quantification of uncertainty using the notion of expected loss allowing a principled termination condition, and we develop more effective elicitation strategies, as shown by our experiments.

We focus on Borda count, but our method can be extended to other social choice functions.

Notation and preliminaries We assume that $V = \{1, \dots, n\}$ is the set voters and C the set of candidates; with $|C| = m$. A profile $(r^1, \dots, r^n)$ is a vector of linear orders on C, one for each voter $i \in V$. The set of all $m!$ linear orders on C is denoted as L; hence the set of all possible profiles is L^n. We use $r(x)$ to denote the position of alternative x in ranking r; and r_i to denote the alternative in the i-th position in r.

A social choice function (or voting rule) $f : L^n \to 2^C \setminus \emptyset$ maps a profile to a non-empty subset of candidates (the "winners"). Scoring rules are social choice functions that rank alternatives according to their score, computed by summing up the number of points they receive in each ranking; the score of x with respect to rankings $r^1, \dots, r^n$ is:

$$s(x; r^1, \dots, r^n) = \sum_{i=1}^{n} w(r^i(x)), \tag{1}$$

where w is a function that assigns each position $1, \dots, m$ to a number of points.

A special type of scoring rule is Borda, that assumes the weights to be: $w(i) = m - i$; the first ranked item is assigned $m - 1$ points, the second obtains $m - 2$, etc.

2 Bayesian vote elicitation

We adopt a Bayesian approach to preference elicitation and approximate winner determination. The system maintains a probability distribution $\mathbb{P}(r^1, \dots, r^n)$ over the preferences (rankings) $r^1, \dots, r^n$ of the voters. The distributions give zero probability to all rankings that are not completion of the known preferences of the voters. We assume that the voters preferences are independent, thus $\mathbb{P}(r^1, \dots, r^n) = \mathbb{P}(r^1) \cdot \dots \cdot \mathbb{P}(r^n)$.

Our incremental elicitation approach is based on looping through the following steps:

- It computes the current best alternative x^* achieving the highest score *in expectation*
- If x^* meets the stopping criterion, the procedure stops and outputs x^*
- Otherwise, it selects an elicitation question to ask the user, and asks it.
- It conditions the probability distribution on the response.

We now discuss the different steps.

Computing expected scores We identify the "approximate winner" as the candidate that yields the highest expected score under the current probability distribution of preference rankings. Each alternative x is associated to its expected score $\mathbb{E}[s(x)]$, that we denote as $\bar{s}(x)$. We observe that for scoring rules $\bar{s}(x)$ can be efficiently computed as:

$$\bar{s}(x) = \sum_{i=1}^{n} \sum_{r^i \in L} \mathbb{P}(r^i) w(r^i(x))$$

and for Borda, the expression further simplifies to:

$$\bar{s}(x) = mn - \sum_{i=1}^{n} \sum_{r^i \in L} \mathbb{P}(r^i) r^i(x).$$

Let $s^* = \max_{x \in C} \mathbb{E}[s(x)]$ be the maximum value of expected score given the current uncertainty, and x^* the associated alternative (the "winner in expectation"); i.e. $s^* = \bar{s}(x^*)$.

[1] AgroParisTech and Ecole Centrale Nantes, email: caillat.maeva@gmail.com
[2] AgroParisTech, email: nicolas.darcel@agroparistech.fr
[3] AgroParisTech, email: cristina.manfredotti@agroparistech.fr
[4] LIP6, CNRS and Sorbonne université, email: paolo.viappiani@lip6.fr

Expected loss and stopping criterion At some point of the interaction, we have a posterior distribution over the rankings, and an associated alternative maximizing the expected score. We estimate the regret or loss of stopping the elicitation and recommending x^*. The user's loss is the difference between her expected utility, under true preferences, of the optimal alternative x^*, and her expected utility under the recommended alternative x.

The loss $\ell(r^1, \dots, r^n)$ is the regret of choosing x^* occurred when the true voters preferences are $(r^1, \dots, r^n)$:

$$\ell(r^1, \dots, r^n) = \max_{y \in C} s(y; r^1, \dots, r^n) - s(x^*; r^1, \dots, r^n).$$

Since we do not know the true preferences, but we know their distributions, we consider the *expected loss* $\mathbb{E}[\ell]$ (in a way analogous to [2] that considered Bayesian elicitation in influence diagrams) that quantifies how far we are from the true optimum in expectation:

$$\mathbb{E}_{r^1 \sim \mathbb{P}(r^1), \dots, r^n \sim \mathbb{P}(r^n)}[l(r^1, \dots, r^n)] =$$
$$= \Big[\sum_{r^1 \in L} \dots \sum_{r^n \in L} \mathbb{P}(r^1) \dots \mathbb{P}(r^n) \max_{y \in C} s(y; r^1, \dots, r^n) \Big] - s^*.$$

In order to compute the above expression we approximate it using a Monte Carlo method. We sample the voters preference rankings from $\mathbb{P}(r^1), \dots, \mathbb{P}(r^n))$, compute the scores of alternatives, and compute the loss for these preferences. We repeat the procedure N times and take the average. To set N (the number of samples) we use Chebyshev inequality. N should be at least $\frac{b^2}{4\delta\epsilon^2}$ where ϵ is the required precision, δ is the confidence and b is an upper bound of the variance; in our case we set $b = n(m-1) - s^*$ (the highest possible Borda score less the current best expected score).

The elicitation procedure continues until the expected loss is lower than a given threshold. If the goal is to find a necessary winner with certainty, the threshold can be set to zero.

Elicitation strategies We consider different strategies to decide which query to ask at any stage of the elicitation process. The strategies aim at reducing uncertainty over the voters preferences in such a way to improve the quality of the approximated winner.

We focus on pairwise comparisons. In the following we denote with $q^v_{a,b}$ the query asking voter v to compare alternatives a and b.

The first approach, adopted from [6], is called *Information Gain for Borda (IGB)* and is based on the notion of entropy. Define $P_{\text{win}}(a)$ as the probability that a wins, that can be found by summing up the probability of all preference combinations that make a a winner. Let $H(W)$ be the entropy associated to the distribution P_{win}. The query $q^v_{a,b}$ is associated with its *information gain* (i.e. the conditional entropy):

$$\text{IG}(q^v_{a,b}) = p^v_{a \succ_v b} H(W | a \succ_v b) + p^v_{b \succ_v a} H(W | b \succ_v a).$$

where $p^v_{a \succ_v b}$, the probability that voter v prefers a to b, can be computed by marginalization.

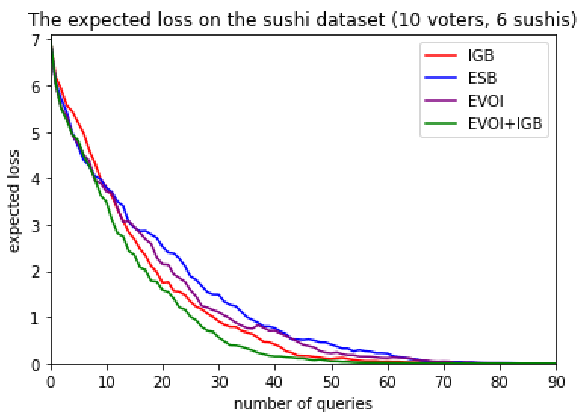
The second approach, *ESB*, also from [6], computes the *a posteriori* improvement of the maximum of P_{win} (details omitted).

The third strategy adopts myopic *Expected Value of Information (EVOI)*, that has been shown to be very effective [7] in single-user preference elicitation. EVOI$(q^v_{a,b})$ is

$$p^v_{a \succ b} \max_{x \in C} \mathbb{E}[s(x) | a \succ b] + p^v_{b \succ a} \max_{x \in C} \mathbb{E}[s(x) | b \succ a] - s^*.$$

The selected query is the one with highest EVOI.

While EVOI can often identify very informative queries, in preliminary tests we realized that it can sometimes happen that *myopic* EVOI of all candidate queries is null. Motivated by this observation, we designed *EVOI+IGB* that asks the query with highest EVOI if its value is positive, and otherwise asks a question using IGB.



Updating the distributions Whenever a query is answered, the distributions are updated using Bayes theorem. In fact, since there is no noise in user feedback, this means assigning zero probability to rankings that are inconsistent with the user's input and to renormalize.

3 Experiments

We provide some preliminary experimental results evaluating the performance of the Bayesian elicitation approach comparing the different elicitation strategies. In the Figure above we plot the expected loss as a function of the number of questions asked using the sushi dataset [5].

The experiments show that a necessary winner can be found with relatively few questions. Somewhat surprinsingly, we found ESB to perform worse than IGB, contrary to what reported in [6]. EVOI+IGB is the most efficient query strategy in our tests.

Future Works We are currently testing the approach in a realistic setting of food recommendations.

Since the current *flat* representation of distributions is not scalable, we are planning to adopt probabilistic ranking models, such as Plackett-Luce. We are also interested in dealing with other aggregation methods and handling more query types.

REFERENCES

[1] M. Stettinger A. Felfernig, L. Boratto and M. Tkalcic, *Group Recommender Systems. An Introduction*, Springer, 2018.

[2] Urszula Chajewska, Daphne Koller, and Ronald Parr, 'Making rational decisions using adaptive utility elicitation', in *Proceedings of AAAI 2000*.

[3] Kathrin Konczak and Jerome Lang, 'Voting procedures with incomplete preferences', *Proceedings of the Multidisciplinary IJCAI-05 Workshop on Advances in Preference Handling*, (01 2005).

[4] Tyler Lu and Craig Boutilier, 'Preference elicitation and robust winner determination for single- and multi-winner social choice', *Artif. Intell.*, **279**, (2020).

[5] Nicolas Maudet, Ulle Endriss, and Yann Chevaleyre, 'A short introduction to computational social choice', in *Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, (01 2005).

[6] Lihi Naamani-Dery, Inon Golan, Meir Kalech, and Lior Rokach, 'Preference elicitation for group decisions using the Borda voting rule', *Group Decision and Negotiation*, **24**(6), 1015–1033, (2015).

[7] Paolo Viappiani and Craig Boutilier, 'On the equivalence of optimal recommendation sets and myopically optimal query sets', *Artificial Intelligence*, **286**, 103328, (2020).

[5] `http://www.kamishima.net/sushi/`

Comportement alimentaire et TCA
JFN2020-ABS-333

Conception, mise en œuvre et évaluation d’un algorithme de recommandations alimentaires pour des groupes

Maéva Caillat[1], Cristina Manfredotti[1], Paolo Viappiani[2], Nicolas Darcel[*3]
[1]UMR 518 MIA-Paris, INRA AgroParisTech Université Paris-Saclay, [2]UMR 7606 LIP6 Laboratoire d'informatique de Paris 6, CNRS, Sorbonne Université, [3]UMR 914 PNCA, INRA AgroParisTech Université Paris-Saclay, Paris, France

Discipline: Expérimental/mécanismes cellulaires et moléculaires
Présentation préférée: Indifférent
Introduction et but de l’étude: Choisir un aliment dans un menu est à première vue une tâche ordinaire et triviale. Cette apparente simplicité cache en réalité un problème de décision complexe. Cette difficulté s'accroît notamment quand il faut sélectionner un aliment acceptable pour un ensemble de mangeurs. L’étude de ces procédures dites “d'agrégation de préférences” constitue un domaine de recherche au carrefour de la théorie du choix social et de l’intelligence artificielle. Elles fournissent de précieuses informations pour comprendre les comportements alimentaires humains. Cette étude avait pour but de développer un algorithme de recommandation de plats, capable de choisir dans un menu l’aliment le plus satisfaisant possible pour un groupe.

Matériel et méthodes: L’algorithme développé s’appuie sur un système de vote (méthode de Borda), où les préférences individuelles ne sont que partiellement connues. Le programme peut interroger chaque utilisateur sur ses préférences deux-à-deux. Par exemple, il peut demander : “Préférez-vous les tomates ou le melon ?”. Pour explorer efficacement l’ensemble très vaste des questions possibles, l'heuristique proposée identifie les questions les plus pertinentes. Cette stratégie se fonde notamment sur une représentation bayésienne des préférences des individus du groupe, qui est mise à jour au fil de leurs interactions avec le programme. Dans un second temps, une expérience de validation a été réalisée en ligne et en temps réel avec 130 adultes volontaires (75 femmes, 55 hommes). Après avoir classé cinq entrées, cinq plats et cinq desserts par ordre de préférence, 18 des participants se sont regroupés en “tablées virtuelles” de quatre ou cinq personnes sur une application de messagerie instantanée (Whatsapp®). Pour chaque groupe de discussion, la sélection d’un repas (entrée, plat et dessert) a été réalisée à la fois par les participants - en échangeant via la messagerie instantanée -, par l’algorithme de recommandation - en interrogeant la base de donnée des préférences des convives -, ainsi que par un opérateur ayant accès à l’intégralité des préférences des participants.

Résultats et Analyse statistique : Nous avons évalué la précision de notre système en comparant les plats sélectionnés par les tablées virtuelles avec ceux renvoyés par l’algorithme, ainsi qu’avec les plats suggérés par l'opérateur omniscient.

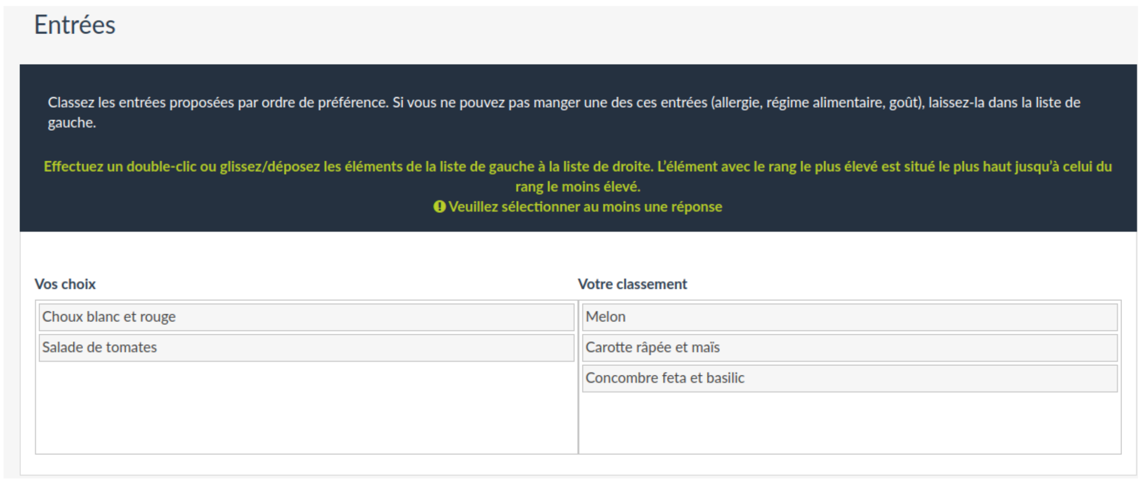
Conclusion: Les systèmes de recommandation peuvent prédire avec justesse les choix réalisés par les individus dans une situation de recherche de consensus. Les travaux de modélisation des décisions de groupe aident à comprendre et à caractériser expérimentalement les critères qui président à nos choix alimentaires en situation d’interaction sociale. Le travail présenté ici s’est concentré sur une prise de décision “simple”, basée sur la seule considération des préférences individuelles. Cependant, il gagnera en justesse s’il est complété par l’intégration des effets des interactions sociales à l’œuvre pendant un repas (“décisions empathiques”).

Conflits d’intérêts: Aucun conflit à déclarer

Appendix B

Our nutrition experiment

130 adults completed the following online questionnaire on LimeSurvey (Figures B.1, B.2 and B.3). One month later, we gathered 18 participants in four WhatsApp discussions. They had to agree on a starter dish, a main course, and a dessert for the group. I don't include the discussion text files in this report, but one may email me to access them. We drew 10 persons out of these 18 participants, who will receive a twenty euros gift voucher.

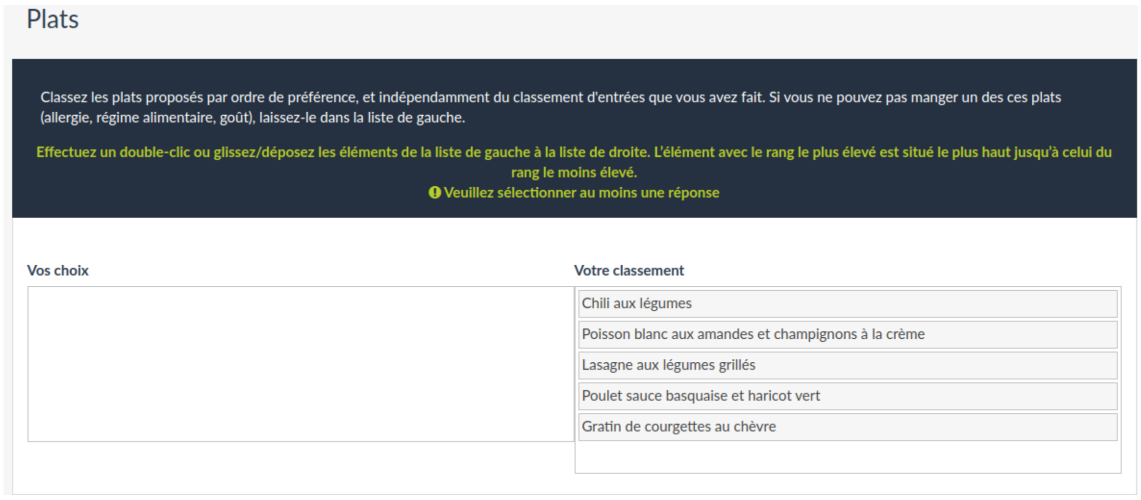

Entrées

Classez les entrées proposées par ordre de préférence. Si vous ne pouvez pas manger une des ces entrées (allergie, régime alimentaire, goût), laissez-la dans la liste de gauche.

Effectuez un double-clic ou glissez/déposez les éléments de la liste de gauche à la liste de droite. L'élément avec le rang le plus élevé est situé le plus haut jusqu'à celui du rang le moins élevé.
Veuillez sélectionner au moins une réponse

Vos choix	Votre classement
Choux blanc et rouge	Melon
Salade de tomates	Carotte râpée et maïs
	Concombre feta et basilic

Figure B.1: Starter dishes survey

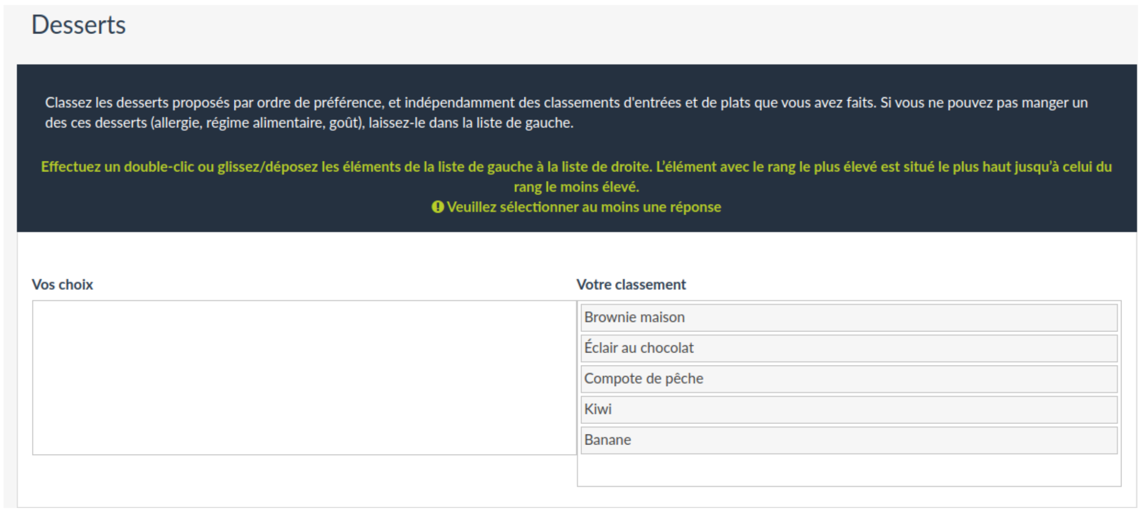

Plats

Classez les plats proposés par ordre de préférence, et indépendamment du classement d'entrées que vous avez fait. Si vous ne pouvez pas manger un des ces plats (allergie, régime alimentaire, goût), laissez-le dans la liste de gauche.

Effectuez un double-clic ou glissez/déposez les éléments de la liste de gauche à la liste de droite. L'élément avec le rang le plus élevé est situé le plus haut jusqu'à celui du rang le moins élevé.
Veuillez sélectionner au moins une réponse

Vos choix	Votre classement
	Chili aux légumes
	Poisson blanc aux amandes et champignons à la crème
	Lasagne aux légumes grillés
	Poulet sauce basquaise et haricot vert
	Gratin de courgettes au chèvre

Figure B.2: Main dishes survey


Desserts

Classez les desserts proposés par ordre de préférence, et indépendamment des classements d'entrées et de plats que vous avez faits. Si vous ne pouvez pas manger un des ces desserts (allergie, régime alimentaire, goût), laissez-le dans la liste de gauche.

Effectuez un double-clic ou glissez/déposez les éléments de la liste de gauche à la liste de droite. L'élément avec le rang le plus élevé est situé le plus haut jusqu'à celui du rang le moins élevé.
Veuillez sélectionner au moins une réponse

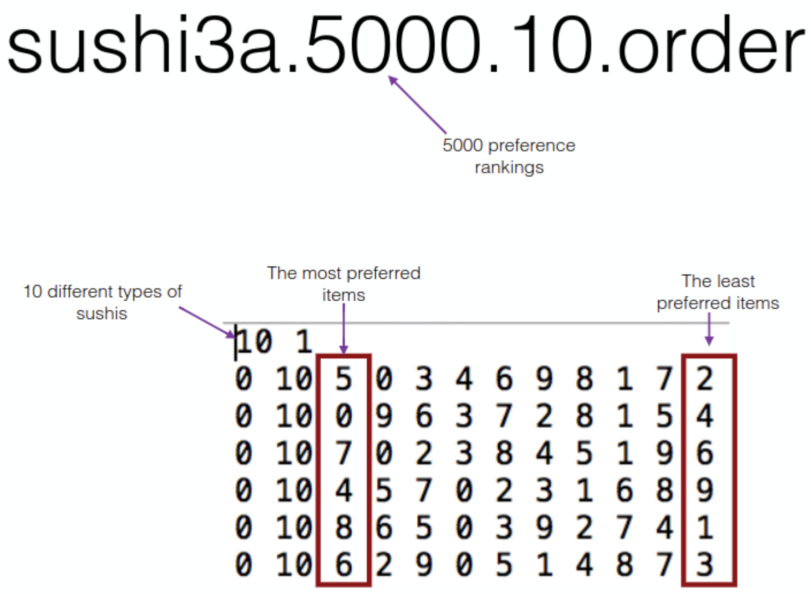
Vos choix	Votre classement
	Brownie maison
	Éclair au chocolat
	Compote de pêche
	Kiwi
	Banane

Figure B.3: Desserts survey

Appendix C

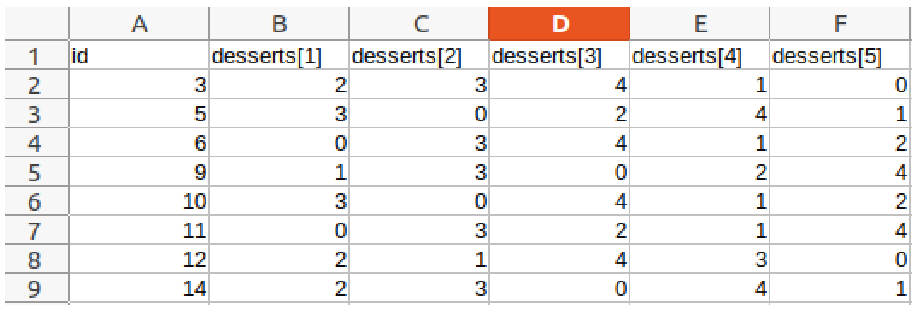
Datasets

Figure C.1 shows a fragment of the Sushi dataset. Figure C.2 shows a fragment of the CROUS dataset, in particular of the 'Desserts' section. There are 130 rankings over 5 starter dishes, 5 main courses and 5 desserts. The first column represents the most preferred items and the last filled cell on a line represents the least preferred item for a voter. When a user cannot eat an item, there is a blank cell.



```
10 1
0 10 5 0 3 4 6 9 8 1 7 2
0 10 0 9 6 3 7 2 8 1 5 4
0 10 7 0 2 3 8 4 5 1 9 6
0 10 4 5 7 0 2 3 1 6 8 9
0 10 8 6 5 0 3 9 2 7 4 1
0 10 6 2 9 0 5 1 4 8 7 3
```

Figure C.1: Fragment of the Sushi database



	A	B	C	D	E	F
1	id	desserts[1]	desserts[2]	desserts[3]	desserts[4]	desserts[5]
2	3	2	3	4	1	0
3	5	3	0	2	4	1
4	6	0	3	4	1	2
5	9	1	3	0	2	4
6	10	3	0	4	1	2
7	11	0	3	2	1	4
8	12	2	1	4	3	0
9	14	2	3	0	4	1

Figure C.2: Fragment of the CROUS database (desserts)

Appendix D

Noticeable Python functions

```python
def borda_permut(distrib, vc):
    """
    Return the expected Borda scores regarding distrib.

    Parameters
    ----------
    distrib : ARRAY
        The current permutation distribution.
    vc : ARRAY
        The set of permutations.

    Returns
    -------
    stats : DICT
        The expected borda scores of the candidates.

    """
    # The maximal number of points equals to nb_candidates - 1.
    max_points = len(vc[0]) - 1
    count_points = np.zeros(len(vc[0]))

    for i in range(len(vc)):
        for j in range(len(vc[0])):
            # The preferred item receives max_points*P(permutation),
            # the second-preferred (max_points-1)*P(permutation)...
            count_points[
                int(vc[i, j])
                ] += (max_points-j) * sum(distrib[:, i])

    stats = {str(r): count_points[r] for r in range(len(count_points))}
    return stats
```

```python
def posterior_distrib(vc, cj, ck, init_distrib, vi):
    """
    Return the posterior probability distributions knowing qi, cj>ck.

    Parameters
    ----------
    vc : ARRAY
        The set of possible permutations.
    cj : INT
        Candidate j.
    ck : INT
```

```
        Candidate k.
    init_distrib : ARRAY
        The initial permutation distribution.
    vi : INT
        Voter i.

    Returns
    -------
    distrib : ARRAY
        The posterior distrib knowing qi, cj>ck.

    """
    distrib = np.copy(init_distrib)
    index_cj_ck = index_query(vc, cj, ck)

    s = sum(distrib[vi][index_cj_ck])
    if s != 0:
        p = 1/s
        for r in range(len(distrib[vi])):
            if r in index_cj_ck:
                distrib[vi][r] *= p
            else:
                distrib[vi][r] = 0
    return distrib
```

```
def expected_loss(v, c, vc, n, distrib):
    """
    Return the expected loss estimated with Monte Carlo.

    Parameters
    ----------
    v : ARRAY
        The set of voters.
    c : ARRAY
        The set of candidates.
    vc : ARRAY
        The set of permutations.
    n : INT
        The sample size for the expected loss (Monte Carlo).
    distrib : ARRAY
        The current permutation distribution.

    Returns
    -------
    expect_loss : INT
        The expected loss estimated with Monte Carlo.

    """
    # Initialize the expected Borda scores.
    eu_array = np.array(list(borda_permut(distrib, vc).values()))
    # The candidate with the highest expected Borda score.
    winner = np.argmax(eu_array)
    expect_loss = 0
    # Loop on the samples.
    for _ in range(n):
        rd_permut = []
        # For voter i, sample a permutation from vci.
        for i in range(len(v)):
            # Choose a permutation using the probabilities associated.
            rd_permut.append(vc[rd.choice(len(vc), 1, p=distrib[i])])
        all_rd_permut = np.array(rd_permut).flatten().reshape((len(v), len(c)))
        # Find the local scores using the Borda voting protocol.
        local_scores = np.array(list(borda(all_rd_permut).values()))
        # Compute the local expected loss.
        local_loss = max(local_scores) - local_scores[winner]
        # Add the local expected loss to the previous expected losses.
        expect_loss += local_loss
    # Divide the accumulated Borda scores by the sample size.
    expect_loss /= n
    return expect_loss
```

```
def info_gain(v, c, vc, gamma, distrib):
    """
    Return the information gains of all the qi,cj>ck.

    Parameters
    ----------
    v : ARRAY
        The set of voters.
    c : ARRAY
        The set of candidates.
    vc : ARRAY
        The set of permutations.
    gamma : INT
        The sample size.
    distrib : ARRAY
        The current permutation distribution.

    Returns
    -------
    dict
        IG(vi, cj>ck)

    """
    # The list of cj > ck.
    comp_cand = np.array(list(permutations(c, 2)))
    # The information gains of the queries.
    ig_array = np.zeros((len(v), len(comp_cand)))
    # The winning probability array regarding the current distribution.
    pr_win = win_proba(v, c, vc, gamma, distrib)
    # The entropy function.
    entropy = st.entropy(pk=pr_win, base=2)

    # Query the i-th voter.
    for i, _ in enumerate(v):
        # Ask the query 'cj > ck ?'.
        for q, _ in enumerate(comp_cand):
            # The posterior probability distribution knowing  qi,cj>ck.
            post_distrib = posterior_distrib(vc,
                                             comp_cand[q][0],
                                             comp_cand[q][1],
                                             distrib,
                                             v[i])
            # The winning proba array knowing qi,cj>ck.
            post_pr_win = win_proba(v, c, vc, gamma, post_distrib)
            # The posterior entropy function.
            ig_array[i][q] = st.entropy(pk=post_pr_win, base=2)

    ig_array = entropy - ig_array
    ig_dict = {'IG(%s,c%s>c%s)' % (v[i], comp_cand[q][0], comp_cand[q][1]):
```

```
                ig_array[v[i]][q]
                for i in range(len(v))
                for q in range(len(comp_cand))
                }
    return ig_dict


def weighted_info_gain(v, c, vc, gamma, distrib, queries):
    """
    Return the weighted information gains of the queries qi,cj,ck.

    Parameters
    ----------
    v : ARRAY
        The set of voters.
    c : ARRAY
        The set of candidates.
    vc : ARRAY
        The set of permutations.
    gamma : INT
        The sample size.
    distrib : ARRAY
        The current permutation distribution.

    Returns
    -------
    dict
        WIG(vi,cj,ck)

    """
    # The information gains of the queries.
    ig_dict = info_gain(v, c, vc, gamma, distrib)
    # Unordered permutations.
    comb = np.array(list(combinations(c, 2)))
    # The weighted information gains of the queries.
    wig_array = np.zeros((len(v), len(comb)))

    for i, _ in enumerate(v):
        for q, _ in enumerate(comb):
            p_1 = proba_query(vc, comb[q][0], comb[q][1], distrib, v[i])
            p_2 = proba_query(vc, comb[q][1], comb[q][0], distrib, v[i])
            # The posterior entropy function.
            wig_array[i][q] = ig_dict[
                'IG(%s,c%s>c%s)'
                % (v[i], comb[q][0], comb[q][1])
                ]*p_1 + ig_dict[
                    'IG(%s,c%s>c%s)'
                    % (v[i], comb[q][1], comb[q][0])
                    ]*p_2

    wig_dict = {'WIG(%s,%s,%s)' % (v[i], comb[q][0], comb[q][1]):
                round(wig_array[i][q], 2)
                for i in range(len(v))
                for q in range(len(comb))
                if [v[i], comb[q][0], comb[q][1]] not in queries
                }
    return wig_dict


def optimal_wig_query(v, c, vc, gamma, distrib, queries):
```

```python
    """
    Return the query with the highest WIG.

    Parameters
    ----------
    v : ARRAY
        The set of voters.
    c : ARRAY
        The set of candidates.
    vc : ARRAY
        The set of permutations.
    gamma : INT
        The sample size.
    distrib : ARRAY
        The current permutation distribution.


    Returns
    -------
    chosen_query : LIST
        WIG(vi,cj,ck)
    max_chosen_query : FLOAT
        The ingo gain of the chosen query.

    """
    wig_dict = weighted_info_gain(v, c, vc, gamma, distrib, queries)
    # Choose the query with the highest WIG.
    max_chosen_query = max(wig_dict.values())
    print('WIG of the query asked: ', max_chosen_query)
    chosen_query_list = [k for k, v in wig_dict.items()
                         if v == max_chosen_query]
    # Randomly choose a query among the ones with the highest WIG.
    chosen_query = rd.choice(chosen_query_list)
    return([int(s) for s in re.findall(r'\b\d+\b', chosen_query)],
           max_chosen_query)
```

```python
def transitivity_complete(answer,
                          vi,
                          cj,
                          ck,
                          p_min,
                          p_max,
                          vc,
                          distrib,
                          queries,
                          list_alternative_worst):
    """
    Use the transitivity of preferences.

    Parameters
    ----------
    answer : INT
        1 if cj>ck, 0 otherwise.
    vi : INT
        The ith user.
    cj : INT
        The jth candidate.
    ck : INT
        The kth candidate.
    p_min : ARRAY
```

```
        The possible minima array.
    p_max : ARRAY
        The possible maxima array.
    vc : ARRAY
        The set of permutations.
    distrib : ARRAY
        The permutation distribution.
    queries : ARRAY
        The answers already known.
    list_alternative_worst : LIST OF LISTS
        One list per user, in each one list per candidate
        containing the candidates inferior to this very candidate.

    Returns
    -------
    p_min : ARRAY
        The possible minimma array updated.
    p_max : ARRAY
        The possible maxima array updated.
    distrib : ARRAY
        The permutation distribution updated.
    queries : ARRAY
        The list of answers updated.
    list_alternative_worst : LIST of LISTS
        The list of inferior candidates for every candidate
        and every user updated.

    """
    # If cj is preferred to ck.
    if int(answer) == 1:
        c_best = cj
        c_worst = ck
    # If ck is preferred to cj.
    else:
        c_best = ck
        c_worst = cj

    print("User v" + str(vi)
          + " prefers c" + str(c_best)
          + " to c" + str(c_worst) + ".")

    # Update the rankings distribution regarding this answer.
    distrib = posterior_distrib(vc, c_best, c_worst, distrib, vi)

    # Add c_worst to the list of inferior candidates of c_best for vi.
    list_alternative_worst[vi][c_best].append(c_worst)
    # The possible min of c_best increases of 1.
    p_min[c_best] += 1
    # The possible max of c_worst decreases of 1.
    p_max[c_worst] -= 1

    # Loop on the list of inferior items of c_worst for vi.
    for alt in list_alternative_worst[vi][c_worst]:
        # If c_worst>alt and alt not compared to c_best yet, add c_best>alt.
        if alt not in list_alternative_worst[vi][c_best]:
            list_alternative_worst[vi][c_best].append(alt)
            p_min[c_best] += 1
            p_max[alt] -= 1
            queries.append([vi, min(alt, c_best), max(alt, c_best)])
    for a in range(len(list_alternative_worst[vi])):
        # If a>c_best for vi, c_worst not compared to a yet, add a>c_worst.
        if (c_best in list_alternative_worst[vi][a] and
                c_worst not in list_alternative_worst[vi][a]):
            p_min[a] += 1
            p_max[c_worst] -= 1
            queries.append([vi, min(a,
                                    c_worst),
                            max(a,
                                c_worst)])
            list_alternative_worst[vi][a].append(c_worst)
    return(p_min,
           p_max,
           distrib,
           queries,
           list_alternative_worst)
```

```
def find_preferences(v,
                     c,
                     vc,
                     gamma,
                     rating,
                     distrib,
                     heuristic,
                     termination_value,
                     epsilon,
                     delta,
                     israeli):
    """
    The main function.

    Parameters
    ----------
    v : ARRAY
        The set of voters.
    c : ARRAY
        The set of candidates.
    vc : ARRAY
        The set of permutations.
    gamma : INT
        The sample size for PrWin algo.
    rating : ARRAY
        The rankings of the candidates by the users.
    heuristic : STRING
        The heuristic used to find a winner.
    termination_value : FLOAT
        The algo stops when EU loss is higher than this value.
    epsilon : FLOAT
        The desired threeshold for EU loss,
        used to calculate the sample size for EU loss.
    delta : FLOAT
        The confidence parameter (often 95% or 99%).
    database : STRING
        Dataset used (random, sushi or netflix).
    nb_matrix : INT
        Number of matrices needed for generating
        an initial permutation distribution for the sushi dataset.
    nb_user_init_distrib : INT
        Number of users needed for generating
        an initial permutation distribution for the sushi dataset.
    israeli : BOOL
```

```
        If True, apply the Israeli methods else, use the expected loss too.

    Returns
    -------
    nw : INT
        A necessary winner.
    runtime : TIME
        Total runtime in seconds.
    communication_cut : FLOAT
        Percentage of dataset queried.
    loss_array : ARRAY
        The expected loss array.
    time_array : ARRAY
        The array of time when expected losses are saved.
    nb_queries : INT
        Number of queries.

    """
    # Initialize time.
    starttime = timeit.default_timer()

    # The initial possible maximums of items.
    p_max = np.ones(len(c)) * ((len(c)-1) * len(v))

    # The initial possible minimums of items.
    p_min = np.zeros(len(c))

    # The list of possible winners.
    nw_list = []

    # The list of queries asked.
    queries = []

    # The real Borda scores.
    eu_array = np.array(list(borda(rating).values()))
    print("The real expected Borda scores are: ", eu_array)

    # Stopping criterion booleans.
    stop_loss = True
    stop_nw = True
    stopping_criterion = True

    # If we want to compute the expected loss.
    if not israeli:
        # The worst case loss.
        # x = (len(c) - 1) * len(v) - max(eu_array)
        # Minimum number of samples needed.
        n = 1000
        # n = int(round((x ** 2) / ((epsilon ** 2) * delta))) + 1
        print('Number of samples needed:', n)
        # The expected loss.
        expect_loss = expected_loss(v, c, vc, n, distrib)
        print("The initial expected loss is: ", expect_loss)

        # The expected losses vs. time.
        expect_losses = [expect_loss]

    nb_queries = 0
    # The list of least-liked items for every item and every voter.
    list_alternative_worst = [[[] for _ in range(len(c))]
```

```
                                for _ in range(len(v))]
    time = [timeit.default_timer()]
    print("\n")
    while stopping_criterion:
        # Find the next query qi,j,k thanks to an heuristic.

        # Highest Expected Score Heuristic for Borda Voting
        if heuristic == 'ESB':
            query, value_query = optimal_wem_query(v,
                                                   c,
                                                   vc,
                                                   gamma,
                                                   distrib,
                                                   queries)
        # Information Gain Heuristic for Borda Voting
        elif heuristic == 'IGB':
            query, value_query = optimal_wig_query(v,
                                                   c,
                                                   vc,
                                                   gamma,
                                                   distrib,
                                                   queries)
        # Expected Value of Information Heuristic for Borda Voting
        elif heuristic == 'EVOI':
            query, value_query = optimal_evoi_query_no_mc(v,
                                                          c,
                                                          vc,
                                                          distrib,
                                                          queries)
        # EVOI heuristic, then IGB heuristic if EVOI=0
        elif heuristic == 'EVOI+IGB':
            query, value_query = optimal_evoi_query_no_mc(v,
                                                          c,
                                                          vc,
                                                          distrib,
                                                          queries)
            if value_query == 0:
                query, value_query = optimal_wig_query(v,
                                                       c,
                                                       vc,
                                                       gamma,
                                                       distrib,
                                                       queries)
        else:
            sys.exit('Error in the name of the heuristic!')

        vi = query[0]
        cj = query[1]
        ck = query[2]
        query = [vi, cj, ck]
        print("The question selected is: \'User v"
              + str(vi) + ", do you prefer c"
              + str(cj) + " or c"
              + str(ck) + "? \'")

        # If query not already asked.
        if query not in queries:
            # Add the query to list of queries.
            queries.append(query)
            nb_queries += 1
            # We ask user vi to answer cj>ck.
            answer = deterministic_answers_to_query(vi, cj, ck, rating)
            # We use transitivity closure in answers to queries.
            (p_min,
             p_max,
             distrib,
             queries,
             list_alternative_worst) = transitivity_complete(answer,
                                                             vi,
                                                             cj,
                                                             ck,
                                                             p_min,
                                                             p_max,
                                                             vc,
                                                             distrib,
                                                             queries,
                                                             list_alternative_worst)
            print("Pmax = ", p_max)
            print("Pmin = ", p_min)

            # Update the possible winner array.
            nw_list = [j for j in range(len(c))
                       if p_min[j] >= max(np.delete(p_max, j))]
            # False if no approximate winner, True otherwise.
            stop_nw = (not nw_list)
            print("Number of different questions asked: ", nb_queries)

            if not israeli:
                # The current expected Borda scores.
                eu_array = np.array(list(borda_permut(distrib, vc).values()))
                # The worst case loss.
                # x = (len(c) - 1) * len(v) - max(eu_array)
                # Minimum number of samples needed.
                n = 1000
                # n = int(round((x ** 2) / ((epsilon ** 2) * delta)))+1
                print('Number of samples needed:', n)
                # The expected loss.
                expect_loss = expected_loss(v, c, vc, n, distrib)
                expect_losses.append(expect_loss)
                print("Current EU: ", eu_array)
                print("Current expected loss: ", expect_loss)
                stop_loss = np.any(expect_loss > termination_value)

            stopping_criterion = (stop_loss and stop_nw)
            time.append(timeit.default_timer())
            print("\n")

        else:
            print("Question already asked before.")
            print("\n")
    # if a possible winner is found, return it.
    if not stop_nw:
        nw = nw_list[0]
    # if the expected loss is null, return the item with the best expected score.
    else:
        nw = np.argmax(eu_array)

    runtime = timeit.default_timer() - starttime
    # The cut in the communication cost.
    communication_cut = 100 * (1 - (2*nb_queries/(len(c)*len(v)*(len(c)-1))))
    time_array = np.array(time)-starttime

    if israeli:
        return(nw,
               runtime,
               communication_cut,
               np.array([]),
               time_array,
               nb_queries)

    return(nw,
           runtime,
           communication_cut,
           np.array(expect_losses),
           time_array,
           nb_queries)
```