Technische Universität Berlin

Fakultät IV - Elektrotechnik und Informatik
Institut für Telekommunikationssysteme

Bachelor Thesis

Extension of the FA*IR Top-k Ranking Algorithm to Multinomial Use Cases

Hyerim Hwang
Matriculation Number: 370104

16.04.2018

Supervised by
Meike Zehlike
Technical University of Berlin
Department of of Telecommunication Systems
Complex and Distributed IT Systems (CIT)

Declaration:
Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, 16 April 2018 (Signature)

Abstract

Rankings are very widely used in our life to set priorities, which are closely related to making decisions. Accordingly, our concerns about algorithmic bias in rankings, in which the rankings are generated unfairly, have grown bigger. Therefore, generating fair rankings is an indispensable work as it builds the basis for our decisions and indeed, ranking algorithms have risen to a significant topic in machine learning. According to prior studies, the algorithm for generating a fair top-k ranking: FA*IR, which assures the appearance of specific percentage of protected group from a binomial protected attribute, has already been invented. Now our task remains to further develop this algorithm for multinomial protected attributes. Hence, this thesis presents the development of the previous work of FA*IR Top-k ranking algorithm in binomial use cases to multinomial use cases.

Contents

1 Introduction

Computer applications, which “learn” from given data to perform a specific task–formally named *Machine Learning*–, have been remarkably developed in recent years, for example an information filtering application that delivers the best matching information with users’ preferences. Accordingly, “learning” algorithms that form the base of the machine learning technology have become very significant and thus are being continuously studied. An algorithm is defined as “a sequence of instructions or a set of rules that are followed to complete a task” [9]. Algorithms are very often processed by computers, so-called computer algorithms, to provide desired results economically, compared to the case by human.

However, a bias in computer algorithms could draw undesirable consequences. Imagine that there is a bias in the information filtering system of an online shopping application, which prefers products from American brands. Every time when a user searches for a product, the shopping application will display various products from different brands, but the products from American brands will always be displayed first. Hence, this computer algorithm is biased against the non-American brands. Another example of bias in computer algorithms is Sabre’s and Apollo’s algorithms for controlling search and display functions. [4] In the algorithms, flights, which all segments are served by a single carrier, are preferred. Imagine that there is a traveller who wants to fly to Berlin from Seoul, transferring to another plane in Paris. If the person first flies to Paris with a French Airline, German Airlines’ flight from Paris to Berlin would be ranked lower than French Airlines’ flight from Paris to Berlin, even if they are both offered with the same conditions. Thus, the computer systems are biased against international carriers

Friedman and Nissenbaum [4] defines bias in computer systems as when it “systematically and unfairly discriminates against certain individuals or groups of individuals in favour of others” and a system discriminates unfairly if “an individual or group of individuals are discriminated on grounds that are unreasonable or inappropriate”. For instance, two groups of people, male and female, are applying for bank loan. Let all the other factors be ignored and only the group property, gender, be considered to decide if the person can get the loan. In the list of people accepted by the bank, if the female population falls far below the male population, the procedure of selecting people is biased against women as female is being discriminated on an unreasonable ground, gender. An Example of reasonable discrimination ground in this case could be debt rate of the people.

In this thesis, we focus on rankings that are generated by computer algorithms, which learn from given data set to understand the users’ needs to provide the most suitable data. We define a ranking as an ordered list of items, where each item is associated with a “quality” value indicating how suitable is the item for the users’ needs. Suppose that there are some specific requirements, that are relevant to the users’ needs. The items that fulfil these requirements are selected and ordered by how much they fulfil those requirements. Hence, in a ranking, the items that fulfil the requirements the most are ranked in the higher positions.

We classify the items either as *protected* or *non-protected*. The term *protected group* will be used for the groups of items that experience direct discrimination or are exposed to historic

discrimination, such as people with disabilities, racial or ethnic minorities, and for the privileged ones, *non-protected group*. In previous research of FA*IR [1], a fair ranking algorithm, in which one protected group is present, has been studied. Considering that the number of protected groups to be selected into the ranking could be arbitrary, this thesis focuses on generating a fair ranking not just with one protected group but also with multiple protected groups. In this thesis, we define an unfair ranking as a ranking that does not "fairly" represent all the groups of the items. In other words, if the protected groups are discriminated, so that the items of these groups are generally ranked below the non-protected group or even completely excluded from the ranking, we define the ranking as unfair.

On the other hand, a ranking could be said as fair if it fairly represents all the protected groups even if the model of learning algorithm produces biased results. The protected groups are not excluded from the ranking and a specific proportion of each protected group is guaranteed in any position of the ranking. For example, suppose that people with different ethnic background, White, Black, Asian and Hispanic, are applying for a job. Each person is associated with a value representing how suitable it is to hire the person and a ranking of 100 people based on this value is generated. In the ranking, the people with Black, Asian and Hispanic ethnicity are ranked lower than the people with White ethnicity, so that a lot of people with those three ethnic backgrounds are even excluded from the ranking. Now, let us say that we want to have at least 20 percent of each Black, Asian and Hispanic people in the ranking. In a fair ranking, the population of Black, Asian and Hispanic, should not fall below the minimum proportion of 20 percent in any position of the ranking. Yet the utility of the ranking must be preserved as high as possible, that is, the items that are selected into the ranking must be the better qualified ones than the items that are not being ranked.

This thesis endeavours to extend the prior FA*IR Top-k ranking algorithm by Zehlike et al [1] to multinomial use case. We introduce a ranking algorithm, which generates the above described fair ranking that fairly represents all groups, where more than one protected group are present, and at the same time the utility of the ranking is preserved as high as possible. A corresponding fairness test for the generated ranking is also conducted to prove its fair representation of each group in the ranking.

The rest of the thesis is organized as follows. In chapter 2, the fundamental information and research papers prior to this thesis are provided. In chapter 3, other case of fair ranking algorithm with multiple groups is introduced. In chapter 4, all the preliminaries and notions, that are used to explain the ranking algorithm are presented and the fairness constraints and formal problem statement are demonstrated. In chapter 5, the developed ranking algorithm is explained with its representation in pseudo code and the achievement of the fairness constraints is proven. In chapter 6, the application of the ranking algorithm to both a synthetic data set and real data set is conducted, and the results of the application are presented. Finally, in chapter 7, the total procedure and the results of the ranking algorithm are summed up and potential themes further to be studied are proposed.

2 Background

In this chapter, we show the development of fair top-k ranking algorithm (§2.1), which forms the foundation of the work of this thesis. Next, we present the fundamental concept of the Fair Top-k ranking algorithm in multinomial use case (§2.2) developed from the previous works.

2.1 Fair Top-k Ranking Algorithm

2.1.1 Concept of Group Fairness

Before understanding the idea of the fair ranking algorithm, it must be made clear how "fairness" could be defined and how it could be measured. Yang and Stoyanovich [6] provided the concept of fairness in ranking and studied fairness measures in rankings based on statistical parity: a requirement that the protected groups should have the same exposure as the non-protected groups. They introduce a *fairness probability* $f \in [0,1]$, which is a real-valued number between 0 and 1 specifying the relative preference between the protected group and the non-protected group, under the premise that there is only one protected group.

Further, they present a ranking algorithm, which takes a ranking and *fairness probability* as inputs and regenerates the input ranking. This ranking algorithm considers the given *fairness probability* as the probability of the protected items being selected into a certain position of the ranking, so that the relative preference between the protected group and the non-protected group can be controlled in the regenerated ranking.

Suppose that a ranking of size $n \in \mathbb{N}$, should be regenerated by the ranking algorithm by Yang and Stoyanovich. This procedure can then also be compared with n-times coin toss event. In each coin toss trial, there are two possible outcomes that can be produced, head and tail. Each outcome has a fixed probability to be thrown in each trial. In the case of coin, the probability of each head and tail being thrown in each trial is 0.5. Now suppose that we have a situation where the coin is biased. For example, the probability of throwing head in each trial is 0.3, while the probability of throwing tail is 0.7. If we throw the coin for n times, we will throw tail more than head, since the probability to throw tail is larger than to throw head.

With the above example, we can represent the coin toss procedure with *binomial distribution* [12]: Let there be $n \in \mathbb{N}$ independent and identical trials, where each trial results in one of two possible random variables, E_0, and E_1; The probability, $p \in [0,1]$, of E_0 to happen in each trial remains the same from trial to trial.

2.1.2 Fair Top-k Ranking Algorithm in Binomial Use Case

Inspired by Yang and Stoyanovich, a fair top-k ranking algorithm where one protected group is presented in the ranking, has been studied recently by Zehlike et al. [1] Zehlike defines a fair ranking as a ranking which fairly represents the protected group, that is, certain proportion of protected items should be assured to appear in the ranking. In her research

paper, she introduces the FA*IR algorithm that generates a fair ranking of $k \in \mathbb{N}$ items, where a specified proportion of the protected items is guaranteed in any position of the ranking.

According to the researchers of FA*IR, the algorithm is applied on the assumption that that the protected group in the ranking is being discriminated in the procedure of being selected. However, it cannot be distinguished by the algorithm itself whether the discrimination is based on some reasonable grounds. Therefore, during the procedure of the fair ranking algorithm, it is expected for the users of the algorithm to assume that the ranked outputs contains undesirable discrimination and hence to provide the minimum proportion p, where $p \in [0,1]$, of the protected group as an input, which could be originated in a legal mandate or in voluntary commitments. [1]

To test if the ranking fairly represents the protected group, Zehlike further presents the *Fair Representation Condition* based on the statistical hypothesis test [8]. The null hypothesis H_0 represents that the protected items are represented with a sufficient proportion, while the alternative hypothesis H_a states that the proportion of protected items is insufficient. To perform the test, a significance level α for the probability of declaring a fair ranking as unfair (Type I error) should be provided as input. Then the *Fair Representation Condition* can be tested as follows:

Let $F(x; n, p)$ be the cumulative distribution function for a binomial distribution of parameter n and p; Suppose that there is a ranking τ and let $\tau_p \in \mathbb{N}$ denote the number of protected items; Then the ranking τ fairly represents the protected group with minimum proportion p if $F(\tau_p; k, p) > \alpha$

Moreover, Zehlike also defines the *Ranked Group Fairness Condition*, which states that the *Fair Representation Condition* must be fulfilled in every position of the ranking.

k / p	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0.1	0	0	0	0	0	0	0	0	0	0	0	0
0.2	0	0	0	0	0	0	0	0	0	0	1	1
0.3	0	0	0	0	0	0	1	1	1	1	1	2
0.4	0	0	0	0	1	1	1	1	2	2	2	3
0.5	0	0	0	1	1	1	2	2	3	3	3	4
0.6	0	0	1	1	2	2	3	3	4	4	5	5
0.7	0	1	1	2	2	3	3	4	5	5	6	6

Table 1 Table of minimum target number of the protected group that must appear in the top k positions. [1]

Based on this observation, the FA*IR algorithm provides a way to produce a fair ranking that satisfies the *Ranked Group Fairness Condition*, where at least the given proportion of the protected group is assured in each position in the ranking. During the procedure of FA*IR algorithm, a table of minimum target number of the protected group in each position up to k-th position is computed based on the *Ranked Group Fairness Condition*. **Table 1** shows an example of such table. For example, for $p = 0.3$ at least one item from the protected group is required in the top 7 positions and two protected items in the top 12 positions. In every prefix of the ranking, the actual number of the protected items that are already in the

ranking and its minimum target number are compared to test if the fairness criterion is fulfilled: The fairness criterion is then satisfied when the ranking contains at least the minimum target amount of each protected group.

2.2 Fair Top-k Ranking Algorithm in Multinomial Use Case

In Fair top-k ranking algorithm in multinomial use case, there are more than one protected group. Hence, given that the users specify the minimum proportion for each of the protected group, the ranking must achieve this specified minimum proportion for all the protected groups. Note that when only one protected group is present, the procedure of selecting each item into the ranking can be compared to the coin toss event with binomial distribution (§2.1.1). Since there are more than one protected group in multinomial use case, we compare the procedure of selecting each item into the ranking to the dice rolling event.

Suppose that a ranking of size $k \in \mathbb{N}$ with $G \in \mathbb{N}_{>1}$ protected groups and one non-protected group should be generated by the fair top-k ranking algorithm. This procedure can then also be explained with the procedure of rolling a $(G+1)$ sided dice for k-times. In each dice rolling trial, there are $G+1$ possible outcomes that can be produced. Each outcome has a fixed probability to be thrown in each trial. In the case of dice, the probability of throwing each side in each trial is the same with $\frac{1}{G+1}$. Now suppose that we have a situation where we throw a biased dice, so that the probability of throwing each side is different. The sides of the dice with higher probability will then be thrown more than the sides with lower probability, which shows that we can control the proportion of the outcomes depends on the probability.

The dice rolling procedure can be expressed mathematically with *multinomial distribution* [7]: Let there be $n \in \mathbb{N}$ independent and identical trials, where each trial produces exactly one of the random variables, $E_1, E_2, \dots, E_\Upsilon$, where $\Upsilon \in \mathbb{N}$; In each trial, each random variable $E_1, E_2, \dots, E_\Upsilon$ occurs with probability, $p_1, p_2, \dots, p_\Upsilon \in [0,1]$ respectively.

Based on this observation, we present further development of the concept of the fair top-k ranking algorithm with multinomial use case deeper in the upcoming chapters.

3 Related Work

We present the work from L. Elisa Celis, who demonstrates a ranking maximization problem with fairness constraints with different approach (§3.1).

3.1 Ranking with Fairness Constraints

Generating rankings with fairness constraints with arbitrary number of groups of items has been researched by L. Elisa Celis [2]. Celis defines that an *algorithmic bias* is present when "one type of content is overrepresented at the expense of another" [2]. In her research paper, she states that in the procedure of optimizing rankings, algorithmic bias arises and leads to several problems, such as results of search engines that unconsciously establishes stereotypes by over- or underrepresenting sensitive attributes, like gender or race. Hence,

she emphasizes the need of rankings, where no type of content dominates, and further declares such rankings as fair. In her work, she presents some fairness constraints to avoid the algorithmic bias and introduces the *constrained ranking maximization problem* to generate optimized rankings satisfying the fairness constraints.

Celis demonstrates the fundamental situation by having a set of $m \in \mathbb{N}$ items in total and a ranking of $n \in \mathbb{N}$ items from them should be generated, where $n \leq m$. Each item $i \in [m]$ is associated with a *value* ω_{ij} of placing it at a specific position $j \in \{1,2, \dots, n\}$ of the ranking. More specifically, a *value* ω_{ij} represents how much the item i contributes to the ranking when i is ranked in the position j. She then defines the value of the ranking as the sum of the value of the items in the ranking, which can only increase by placing better qualified items higher than worse qualified items in the ranking. She states that there is a set of (possibly non-disjoint) properties $\{1,2, \dots, p\}$, such as gender, age or race, that are assigned to each item.

Based on the given situation, Celis represents a ranking with an assignment matrix x, which is a $m \times n$ binary matrix, where each row represents an item and each column represents a position in the ranking. The assignment matrix then represents a ranking by containing 1 in i-th column if item i is ranked in the position j, and 0 otherwise. Formally:

$x: m \times n \to \{0,1\}^{m \times n}$

$x = \begin{bmatrix} \alpha_{11} & \dots & \alpha_{1n} \\ & \vdots & \\ \alpha_{m1} & \dots & \alpha_{mn} \end{bmatrix}$, where $\alpha_{11}, \dots, \alpha_{mn} \in \{0,1\}$

Further, she introduces fairness constraints to prevent overrepresenting items with a specific property $l \in \{1,2, \dots, p\}$. There exists an upper-bound $u_{kl} \in \mathbb{Z}_{\geq 0}$ for each of the property, which controls the number of items that possess the property l in the top k positions of the ranking. In other words, the number of items with the property l must not exceed the upper-bound u_{kl} in the top k positions of the ranking. Celis formally presents the fairness constraints as follows:
$\sum_{1 \leq j \leq k} \sum_{i \in P_l} x_{ij} \leq u_{kl}$, where P_l is a set of items that possess the property l. In this thesis, different from her fairness constraints, we set a lower bound of items with property l, that is, the number of items with the property l must not be below the lower bound.

With the given situation and the fairness constraints, Celis defines the *Constrained Ranking Optimization Problem* as finding a ranking, in which the *value* is maximized, and the fairness constraints are fulfilled at the same time: $max \sum_{i \in [m], j \in \{1,2,\dots,n\}} \omega_{ij} x_{ij}$.

With the formal definitions and fairness constraints, Celis further presents the approximation algorithms based on linear programming that solves the constrained ranking optimization problem.

4 Problem Statement

In this chapter, we first present the notations (§4.1) used in this thesis. Then we explain the fair top-k ranking criteria (§4.2) and the utility (§4.3). Finally, we present the formal problem statement (§4.4).

4.1 Notation

Let m denotes an item and let $M = \{m_1, m_2, \dots, m_i\}$, where $i \in \mathbb{N}$, represent a set of items. Suppose that a request for a set of items based on a certain criterion can be made. Let us call this request *query* and denote with q and hence, $Q = \{q_1, q_2, \dots, q_n : n \in \mathbb{N}\}$ represents a set of all possible queries, where the size of Q is finite. As it is mentioned above (§1), an item $m \in M$ is specified either as protected, or non-protected. Suppose that there are more than one protected sets of items along with a non-protected set of items. Each item is assigned to exactly one of the sets and is associated with a *unique index*, a *quality*, which represents how relevant is the item to be queried based on a specific criterion, and a *protected attribute*, which indicates whether m is protected or non-protected.

Definition: Item Index

Let $m \in M$ be an item. We define

$i \in \mathbb{N}$, where $i \in \{1,2, \dots, |\mathrm{M}|\}$, a *unique index* an item m, which an item can be uniquely identified with.

Further, let m_i represent an item of index i.

Definition: Quality Function

Let $m_i \in M$ be an item of index $i \in \{1,2, \dots, |\mathrm{M}|\}$, and let $q \in Q$ represent a query.

We define $v: M \times Q \to \mathbb{R}_+$ and $v(m_i, q) \in [0,1]$, the *quality function* of an item $m_i \in M$, which computes the quality of m, referenced to the query q.

In other words: the *quality function* measures how relevant is the item to be requested by a query q.

Definition: Protected Attribute

We define the set of *protected attributes*:

$G: \{g \in \mathbb{N} \cup \{0\} \mid g > 0 \Leftrightarrow g\ is\ protected, g = 0 \Leftrightarrow g\ is\ non-protected\}$. Hence, we specify $g(m_i) \in G$ to represent the *protected attribute* of item $m_i \in M$.

Note that each protected set can be denoted in natural number with the *protected attribute*.

Definition: 3-Tuple Notation

Let $m_i \in M$ be an item. We denote m_i in 3-Tuple:

$I: (\mathbb{N} \times \mathbb{R}_+ \times G)$,

$I = (i, v(m_i, q), g(m_i))$

Note that the set of items in 3-Tuple notations can be represented as:

$M_I = \{(i, v(m_i, q), g(m_i)) : i \in \mathbb{N}, v(m_i, q) \in \mathbb{R}_+\ and\ v(m_i, q) \in [0,1], g(m_i) \in \mathrm{G}\}$

Definition: Ranking

Let I be an element of M_I. Suppose that an ordered set of k items must be generated from M_I with a query q, where $k \in \mathbb{N}$, for $k \leq |M_I|$, denotes the size of the ordered set. We define,

$T: Q \times M_I \to (\mathbb{N} \times \mathbb{R}_+ \times G)^k$ a *ranking* of size k. Note that:

$$T(q, M_I) = \begin{bmatrix} (i_1, v(m_{i_1}, q), g(m_{i_1})) \\ \vdots \\ (i_k, v(m_{i_k}, q), g(m_{i_k})) \end{bmatrix}, \text{ where } i_1, i_2, \dots, i_k \in \mathbb{N}$$

Example: Suppose that a ranking from a set of candidates should be generated based on their credit score and hence, $q \coloneqq Candidates\ with\ best\ credit\ scores$. There are four groups present, which each candidate can be classified into exactly one of them: married, divorced, single and widowed. In the procedure of generating the ranking, the married group is considered as non-protected, hence the *protected attribute* **is zero**. The other three groups are protected and consequently, the *protected attribute* is **bigger than zero**. If there are altogether 50 candidates for the procedure and five random candidates are chosen, m_1, m_2, m_3, m_4 and m_5, where the family status of each candidate is married, divorced, single, widowed, and divorced respectively, their properties are represented as follows:

Candidate	Family status	Protected attribute	Quality function	3-Tuple Notation
m_1	Married	$g(m_1) = 0$	$v(m_1, q)$	$I_1 = (1, v(m_1, q), 0)$
m_2	Divorced	$g(m_2) = 1$	$v(m_2, q)$	$I_2 = (2, v(m_2, q), 1)$
m_3	Single	$g(m_3) = 2$	$v(m_3, q)$	$I_3 = (3, v(m_3, q), 2)$
m_4	Widowed	$g(m_4) = 3$	$v(m_4, q)$	$I_4 = (4, v(m_4, q), 3)$
m_5	Divorced	$g(m_5) = 1$	$v(m_5, q)$	$I_5 = (5, v(m_5, q), 1)$

Table 2 Properties of the candidates

Definition: Minimum Target Group Proportion

Let I be an element of M_I. Let $T(q, M_I)$ denote a ranking and let k represent the size of the ranking. Let $PG_n = \{I \in M_I : g(m_i) = n, n \in G \ \wedge \ n \neq 0, \}$ be the set of *3-Tuple* items I, that are assigned to the identical *protected attribute* $g(m_i)$.

We define $p_n \in [0,1]$ the *minimum target group proportion* of the protected set PG_n, which should be achieved in the ranking $T(q, M_I)$.

Example: In the above example, let us assume that a ranking $T(q, M_I)$ of 20 candidates should be generated based on their credit score. If at least 15 percent of both divorced and single group and 10 percent of widowed group are to be assured in the generated ranking, the *minimum target group proportions* are $p_1 = 0.15$, $p_2 = 0.15$, and $p_3 = 0.1$.

Definition: Rank

We call in a vector $b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \in \mathbb{R}^n$, where $n \in \mathbb{N}$, the $position$ of an n-th entry is n. For instance, the $position$ of b_1 in b is 1.
Let I be an element of M_I. Let $T(q, M_I)$ be a ranking and let k be the size of the ranking. We define the rank of an item I in the ranking $T(q, M_I)$,

$r: M_I \times T(q, M_I) \to \mathbb{N}$

$$r(I, T(q, M_I)) = \begin{cases} position\ of\ I\ in\ T(q, M_I),\ I \in T(q, M_I) \\ |T(q, M_I)| + 1,\ I \notin T(q, M_I) \end{cases}'$$

Example: Continuing with the previous example. let us suppose that a ranking $T(q, M_I)$ of 20 candidates out of 50 candidates should be generated according to their credit score and the candidates m_1, m_2, m_3, m_4 and m_5 are also present. Candidate m_1 appears in the first position of the ranking, while the candidate m_4 was not selected into the ranking and let $I_1 = (1, v(m_1, q), 0)$ and $I_4 = (4, v(m_4, q), 3)$ be the *3-Tuple Notation* of the item m_1 and m_4 respectively. Therefore, their rank is represented as $r\big(I_1, T(q, M_I)\big) = 1$ and $r\big(I_4, T(q, M_I)\big) = 20 + 1 = 21$

Definition: Protected Group Count τ_{PG_n}

Let I be an element of M_I. Let $T(q, M_I)$ be a ranking and let k represent the size of the ranking. Let $PG_n = \{I \in M_I: g(m_i) = n, n \in G\ \wedge\ n \neq 0, \}$ be the set 3-Tuple items I, that are assigned to the identical *protected attribute* $g(m_i)$.

We define

$\tau_{PG_n}: T(q, M_I) \to \mathbb{N},$ with $\tau_{PG_n}(T(q, M_I)) = |\{I \in T(q, M_I): g(m_i) = n\}|$ the *protected group count* of a ranking $T(q, M_I)$.

Note that the *protected group count* of PG_n must always be equal or smaller than the size of PG_n: $\tau_{PG_n}(T(q, M_I)) \leq |PG_n|$

In other words: τ_{PG_n} denote the number of the items in the ranking $T(q, M_I)$, which belong to the protected set PG_n.

Example: In the above example, if there are five divorced candidates, four single candidates and two widowed candidates present in the generated ranking τ, the protected group count τ_{PG_n} for each group is represented as $\tau_{PG_1} = 5, \tau_{PG_2} = 4, and\ \tau_{PG_3} = 2.$

4.2 Fair Top-k Ranking Criteria

Definition: Multinomial Cumulative Distribution function [7]

Let us assume that there should be n independent trials, where $n \in \mathbb{N}$, and let $E_1, E_2, \dots, E_\gamma$, where $\gamma \in \mathbb{N}$, denote the possible random variables that can result from each trial. Each of the trials produces exactly one of the $E_1, E_2, \dots, E_\gamma$ random variables and each variable

occurs on each trial with the probability $p_1, p_2, \dots, p_r$, where each $p_\gamma \in [0,1]$ and $p_\gamma \in \mathbb{R}$ for $\gamma \in \mathbb{N}$.

Let $x = [x_1, x_2, \dots, x_\gamma]$, for each $x_\gamma \in \mathbb{N}$ and $\gamma \in \mathbb{N}$, be an array of observations of each random variable $E_1, E_2, \dots, E_\gamma$ and let $p = [p_1, p_2, \dots, p_r]$ be an array of the probabilities, which each variable can be produced on each trial. We define

$F(x; n, p) = P(E_1 \leq x_1, E_2 \leq x_2, \dots, E_\gamma \leq x_\gamma)$ the *multinomial cumulative distribution function*, which computes the probability that each random variable $E_1, E_2, \dots, E_\gamma$ occurs exactly $x_1, x_2, \dots, x_\gamma$ or fewer than $x_1, x_2, \dots, x_\gamma$ times in n trials with the given probability $p_1, p_2, \dots, p_\gamma$ of each variable to be produced on each trial.

Example: Suppose that a person is rolling a dice for 20 times. A dice has six sides that can be produced on each dice rolling procedure, $E_1, E_2, \dots, E_6$, and each of the side is thrown with the probability of $\frac{1}{6}$. Among the 20 times, the person throws 4 times one, 3 times two, 4 times three, 4 times four, 3 times five and 2 times six, hence $x = [4,3,4,4,3,2]$. The probability of the person throwing exactly x times or fewer than x times of each side is:

$$F\left([4,3,4,4,3,2]; 20, [\tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}]\right) = P(E_1 \leq 4, E_2 \leq 3, E_3 \leq 4, E_4 \leq 4, E_5 \leq 3, E_6 \leq 2) \approx 0.0009$$

To test if all the protected items are fairly represented in a ranking, we introduce the *Fair Representation Condition*. As presented above (§2.1.2), the *Fair Representation Condition* is based on the *statistical hypothesis test* [8] and hence we expect the users to provide a significance level α, to specify how significantly should the ranking represent all the protected items to be declared as fair.

Definition: Fair Representation Condition

Let $F(x; n, p)$ represent *multinomial cumulative distribution function* and let $\tau_{PG} = [\tau_{PG_1}, \tau_{PG_2}, \dots, \tau_{PG_n}]$ be the array of *protected group count* of each group in a ranking $T(q, M_I)$. Let $p = [p_1, p_2, \dots, p_n]$ denote the array of *minimum target group proportion* of each group and let α be the significance level. Both p and α are given as parameters.

We consider that the ranking $T(q, M_I)$ of k items fairly represents the protected sets if:

$$F(\tau_{PG}; k, p) > \alpha$$

Example: In addition to the already defined properties of the previous example with married, divorced, single and widowed groups for rating credit score, let us declare the significance level $\alpha = 0.1$.

$\tau_{PG} = [\tau_{PG_1}, \tau_{PG_2}, \tau_{PG_3}] = [5,4,2]$ and $p = [p_1, p_2, p_3] = [0.15, 0.15, 0.1]$

$F(\tau_{PG}; k, p) = F([5,4,2]; 20, [0.15,0.15,0.1]) \approx 0.5051$

$0.5051 > 0.1$

$\therefore F(\tau_{PG}; k, p) > \alpha$

Hence, the ranking $T(q, M_I)$ fairly represents all the protected groups.

Definition: Ranked Group Fairness Condition

In a vector $b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \in \mathbb{R}^n$, where $n \in \mathbb{N}$, we call the j-th *prefix* of b as $b(j) = \begin{bmatrix} b_1 \\ \vdots \\ b_j \end{bmatrix}$, where $j \in \mathbb{N}$ and $j \leq n$. In other words, j-th *prefix* of b represents all the elements of b up to the $position\ j$.

Let $T(q, M_I)|_k = \langle T(q, M_I)(1), T(q, M_I)(2), \dots, T(q, M_I)(k) \rangle$ denote each prefix of a ranking $T(q, M_I)$ and let k be the size of the ranking.

We declare that the ranking $T(q, M_I)$ satisfies the *ranked group fairness condition*, if the ranking fulfills the fair representation condition in every prefix $T(q, M_I)|_k$ with the parameters $\tau_{PG} = [\tau_{PG_1}, \tau_{PG_2}, \dots, \tau_{PG_n}]$ and $p = [p_1, p_2, \dots, p_n]$ for a given α.

Example: Let us suppose that a generated ranking from the above example was constructed up to the 15th position as follows, where the family status *married*, *divorced*, *single* and *widowed* are represented as *M, D, S* and *W* respectively; the multinomial cumulative distribution function $F(\tau_{PG}; k, p)$ is computed in the last row; and τ_{PG_n}, for $n \in [1,2,3]$, represents the protected group count for each divorced, single and widowed group, which counts the number of protected items of each protected set that are placed up to the current position:

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T(q, M_I)$	M	M	M	D	D	S	M	M	S	D	M	M	M	D	M
τ_{PG_1}	0	0	0	1	2	2	2	2	2	3	3	3	3	4	4
τ_{PG_2}	0	0	0	0	0	1	1	1	2	2	2	2	2	2	2
τ_{PG_3}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$F(\tau_{PG}; k, p)$	0.71	0.34	0.18	0.24	0.21	0.36	0.28	0.22	0.26	0.25	0.21	0.17	0.13	0.12	0.099

Table 3 Example of ranking with the calculated multinomial cumulative distribution function $F(\tau_{PG}; k, p)$ in each position

For example, in the position where $k = 1$, an item of married set is positioned and there exists no items of the protected sets. Hence $\tau_{PG_1}, \tau_{PG_2}, \text{and } \tau_{PG_3}$ are 0, 0, and 0 respectively. Consequently, $F(\tau_{PG}; k, p) = F([0,0,0]; 1, [0.15,0.15,0.1]) \approx 0.71$. In the position where $k = 4$, finally the first protected item from the divorced group is placed, and there are still no items from the other two protected groups. Therefore, $\tau_{PG_1}, \tau_{PG_2}, \text{and } \tau_{PG_3}$ are 1, 0, and 0 respectively. Hence, $F(\tau_{PG}; k, p) = F([1,0,0]; 4, [0.15,0.15,0.1]) \approx 0.24$.

Whether the *fair representation condition* is satisfied or dissatisfied completely depends on the given significance level α. To fulfill the *ranked group fairness condition*, the value computed by
$F(\tau_{PG}; k, p)$ must be bigger than the specified α in every position.

Since the *fair representation condition* should be tested in each prefix of the ranking, if forms the multiple statistical hypothesis testing. Consequently, the significance level α must be adjusted in each prefix. [13] However, the adjustment of the significance level will not be covered by this thesis and remains for the future work.

4.3 Utility

Definition: Utility

Let I be an element of M_I and let $T(q, M_I)$ denote a ranking. We define the *utility* of an item I in the ranking $T(q, M_I)$:

$utility: M_I \times T(q, M_I) \rightarrow \mathbb{R}_+$

$utility(I, T(q, M_I)) = d\left(r\left(I, T(q, M_I)\right)\right) * v(m_i, q)$ where

$d: \mathbb{N} \rightarrow \mathbb{R}_+$

$d\left(r\left(I, T(q, M_I)\right)\right) = \frac{1}{\log\left(1+r\left(I, T(q, M_I)\right)\right)}$.

Note that the d function is inversely proportional to the rank $r(I, T(q, M_I)$ of an item I.

Example: Let the candidates in the first, second, third and fourth position in the ranking $T(q, M_I)$ of the previous example be represented as $m_{k1}, m_{k2}, \mathrm{m_{k3}}$ and m_{k4} respectively, where $k_1, k_2, k_3, k_4 \in \mathbb{N}$. Suppose that each candidate has the credit score of $v(m_{k1}, q) = 0.99$, $v(m_{k2}, q) = 0.85$, $v(m_{k3}, q) = 0.82$ and $v(m_{k4}, q) = 0.88$. Hence in *3-Tuple Notation,* $I_{k1} = (k1, 0.99, g(m_{k1}))$, $I_{k2} = \left(k2, 0.85, g(m_{k2})\right)$, $I_{k3} = (k3, 0.82, g(m_{k3}))$ and $I_{k4} = (k4, 0.88, g(m_{k4}))$ represent each item $m_{k1}, m_{k2}, \mathrm{m_{k3}}$ and m_{k4} respectively.

The *utility* of each candidate is calculated as follows:

$$utility(I_{k1}, T(q, M_I)) = \frac{1}{\log\left(1+r\left(I_{k1}, T(q, M_I)\right)\right)} * v(m_{k1}, q) = \frac{1}{\log(1+1)} * 0.99 \approx 3.289$$

$$utility(I_{k2}, T(q, M_I)) = \frac{1}{\log\left(1+r\left(I_{k2}, T(q, M_I)\right)\right)} * v(m_{k2}, q) = \frac{1}{\log(1+2)} * 0.85 \approx 1.782$$

$$utility(I_{k3}, T(q, M_I)) = \frac{1}{\log\left(1+r\left(I_{k3}, T(q, M_I)\right)\right)} * v(m_{k3}, q) = \frac{1}{\log(1+3)} * 0.82 \approx 1.362$$

$$utility(I_{k4}, T(q, M_I)) = \frac{1}{\log\left(1+r\left(I_{k4}, T(q, M_I)\right)\right)} * v(m_{k4}, q) = \frac{1}{\log(1+4)} * 0.88 \approx 1.259$$

Definition: Selection Utility

Let $utility(I, T(q, M_I))$, for $I \in M_I$, be the *utility* of an item I in a ranking $T(q, M_I)$.
If $I_{in} \in T(q, M_I)$ denotes an item in the ranking $T(q, M_I)$, while $I_{out} \in \{I: I \in M_I \wedge I \notin T(q, M_I)\}$ is an item that is **not** present in the ranking $T(q, M_I)$, we define the selection utility of the ranking $T(q, M_I)$ as:

$SelectionUtility: T(q, M_I) \rightarrow \mathbb{R}$

$SelectionUtility\left(T(q, M_I)\right) = \min\left(utility\left(I_{in}, T(q, M_I)\right)\right) - \max\left(utility\left(I_{out}, T(q, M_I)\right)\right)$

Note that the selection utility is proportional to the minimum utility present in the ranking and inversely proportional to the maximum utility outside of the ranking.

In other words: *Selection utility* represents the utility difference between the item with the least utility **in** the ranking and the item with the best utility **out** of the ranking

Example: Let m_{k20} be the candidate, whose rank is 20 in the ranking $T(q, M_I)$ of 20 candidates from the previous example. $I_{k20} = \big(k20, v(m_{k20}, q), g(m_{k20})\big)$, where $k20 \in \mathbb{N}$, demonstrations the *3-Tuple Notation* of the candidate m_{k20} and consequently $r\big(I_{k20}, T(q, M_I)\big) = 20$. Let m_{k21}, hence $I_{k21} = \big(k21, v(m_{k21}, q), g(m_{k21})\big)$ and $k21 \in \mathbb{N}$, be the candidate who has the best credit score among the candidates that were not selected into the ranking. Suppose that m_{k20} has the worst credit score $v(m_{k20}, \mathrm{q}) = 0.55$ among the candidates in the ranking while m_{k21} has the best credit score $v(m_{k21}, \mathrm{q}) = 0.60$ among the non-ranked candidates.

$$utility(I_{k20}, T(q, M_I)) = \frac{1}{\log\big(1+r\big(I_{k20}, T(q, M_I)\big)\big)} * v(m_{k20}, q) = \frac{1}{\log(1+20)} * 0.55 \approx 0.416$$

$$utility(I_{k21}, T(q, M_I)) = \frac{1}{\log\big(1+r\big(I_{k21}, T(q, M_I)\big)\big)} * v(m_{k21}, q) = \frac{1}{\log(1+21)} * 0.60 \approx 0.447$$

$$\therefore SelectionUtility\big(T(q, M_I)\big) = 0.416 - 0.447 = -0.031$$

Definition: Ordering Utility

Let $I_c, I_d \in T(q, M_I)$ be the *3-Tuple Notation* of the items m_c and m_d, where $c, d \in \mathbb{N}$, that are present in the ranking $T(q, M_I)$ and the following condition is fulfilled:

$$r\big(I_c, T(q, M_I)\big) < r\big(I_d, T(q, M_I)\big) \wedge v(m_c, q) < v(m_d, q)$$

We define the *ordering utility* of the ranking $T(q, M_I)$ as:

$$OrderingUtility\colon T(q, M_I) \to \mathbb{R}$$

$$OrderingUtility\big(T(q, M_I)\big) = \begin{cases} \min(v(m_c, q) - v(m_d, q)), & I_c, I_d \in T(q, M_I) \\ 0, & I_c, I_d \notin T(q, M_I) \end{cases}$$

Note that the maximum *ordering utility* that can be calculated is 0.

Example: Extending the example of the *Definition: Utility*, suppose that no other candidates that are ranked below m_{k4} in the ranking $T(q, M_I)$ have higher credit score than $v(m_{k4}, q)$. m_{k3} has the quality $v(m_{k3}, q) = 0.82$ and ranked in the third position, while m_{k4} has the quality $v(m_{k4}, q) = 0.88$ and is ranked below m_{k3}.

$$\therefore Ordering\ Utility\big(T(q, M_I)\big) = v(m_{k3}, q) - v(m_{k4}, q) = 0.82 - 0.88 = -0.06$$

Definition: In-Group Monotonicity

Let $I = \big(i, v(m_i, q), g(m_i)\big) \in M_I$ be the *3-Tuple Notation* of item $m_i \in M$ of index i, where $i \in [1, |\mathrm{M}|]$, and let M_I represent a set of I. Let $g(m_i)$ represent the *protected attribute* of the item m_i. Further, if $T(q, M_I)$ represent a ranking, let $r(I, T(q, M_I))$ denote the *rank* of I

in the ranking $T(q, M_I)$. We declare that *in-group monotonicity* is preserved in a ranking $T(q, M_I)$ if

$\forall I_x, I_y, \text{where } x, y \in \mathbb{N}$

$$\text{s.t. } g(m_x) = g(m_y), r(I_x, T(q, M_I)) < r\left(I_y, T(q, M_I)\right) \Longrightarrow v(m_y, q) \leq v(m_x, q)$$

In other words: the *in-group monotonicity* is preserved in a ranking if the items that possess the same *protected attribute* are ranked in the decreasing order of their quality. No items with lower quality are ranked higher than the ones with higher quality within a group.

Example: Continuing with the previous example, suppose that each candidate with the family status *divorced* is positioned in the decreasing order of their credit scores in the ranking. Consequently, *in-group monotonicity* for the divorced group is preserved.

4.4 Formal Problem Statement

Based on the Fair Top-k Ranking Criteria that are presented above, the fair top-k ranking problem with multinomial protected groups could be stated formally:

PROBLEM - FAIR TOP-K RANKING IN MULTINOMIAL USE CASE. *Given a set of items [m] and the parameters k, p and α, construct a ranking τ that satisfies the following constraints:*

- *(i) In-group monotonicity is preserved;*
- *(ii) Ranked group fairness with parameter p and α is achieved;*
- *(iii) Selection utility subject to (i) and (ii) is maximized;*
- *(iv) Ordering utility subject to (i), (ii) and (iii) is maximized.*

5 Algorithm

We present the FA*IR algorithm in multinomial use case (§ 5.1) and the table of minimum target number of each protected group, which we introduce to reduce the expense of the FA*IR algorithm. (§ 5.2) Next, we demonstrate the fundamental functions used for the FA*IR algorithm, which are the multinomial inverse cumulative distribution function (§ 5.3) and the multinomial cumulative distribution function. (§ 5.4) Finally, we prove the correctness of the algorithm (§ 5.5).

5.1 FA*IR in Multinomial Use Case

Algorithm 1 shows the FA*IR algorithm in pseudocode. It takes as inputs: the data set M_I; the minimum target number of each protected group as an array of arrays, which each array shows the minimum target number of the protected groups in each position up to k-th position; the significance level α, and the query q. The algorithm outputs a ranking $T(q, M_I)$ that achieves the group fairness criteria and optimizes the selection- and ordering utility.

The data is separated into each group list $(P_0, P_1, \dots, P_G)$ and sorted in descending order of the quality. (line 5-10) The variables $\left(\tau_{PG_0}, \tau_{PG_1}, \tau_{PG_2}, \dots, \tau_{PG_G}\right)$ are declared and initialized with 0. (line 11) These variables are used to count the number of items of each group that

have been selected into the ranking. Then the algorithm starts to generate the ranking with the minimum target number L. (line 13) First, the minimum target number of current position l and the number of protected items $(\tau_{PG_1}, \tau_{PG_2}, \dots, \tau_{PG_G})$ are compared. (line 15) If any protected group has not achieved the minimum target number, the algorithm takes the first item of this group list, that has the highest quality in this group, and ranks it into the current position of the ranking. The protected group count $(\tau_{PG_1}, \tau_{PG_2}, \dots, \tau_{PG_G})$ of this group is increased by 1. (line 16-18) If all the protected groups have achieved the minimum target number, the algorithm compares the first items of each group list and places the one with the highest quality into the ranking. The group count $(\tau_{PG_0}, \tau_{PG_1}, \tau_{PG_2}, \dots, \tau_{PG_G})$ of this group is increased by 1. (line 20-26)

FA*IR in multinomial use case has the algorithm complexity of $O(n + k\ log\ k)$, where $O(n)$ denotes the complexity to separate the data set into each group list. Selecting an item into the ranking has the complexity of $O(log\ k)$ and since the algorithm constructs a ranking of k items, the procedure of selecting k items into the ranking has the complexity of $O(k\ log\ k)$. FA*IR succeeds to generate ranking as long as there are enough items of each group. If there are k items of each group, the algorithm is guaranteed to succeed.

Algorithm 1: This presents the FA*IR algorithm with multinomial protected groups. The FA*IR algorithm generates a ranking that satisfies the group fairness criteria and maximizes the utility subject to in-group monotonicity.

Input:

- M_I: Data set to be ranked in *3-Tuple Notation*
- L: 2-dimensional array; minimum target number of each protected group in each position up to k-th position
- α: Significance level
- q: query

Output: A ranking $T(q, M_I)$ achieving the group fairness criteria and optimizing selection- and ordering utility

1 $G \leftarrow number\ of\ protected\ groups$
2 // Empty lists, which each will be filled with items with same protected attribute. P_0 is for non-protected
3 $(P_0, P_1, \dots, P_G) \leftarrow ([\,], \dots, [\,])$
4 //sort the data set in descending order of quality.
5 $data \leftarrow sort(M_I)$
6 **for** m_i **in** $data$ **do**
7 $\quad n \leftarrow g(m_i)$
8 $\quad$ //separate items with their protected attribute
9 $\quad append(P_n, m_i)$
10 **end**
11 $(\tau_{PG_0}, \tau_{PG_1}, \tau_{PG_2}, \dots, \tau_{PG_G}) \leftarrow (0, 0, \dots, 0)$
12 //generate rankings according to the minimum target number of each protected group
13 **for** l **in** L **do**
14 $\quad$ **for** $j \leftarrow 0$ **to** $length(l) - 1$ **do**
15 $\quad\quad$ **if** $\tau_{PG_{j+1}} < l[j]$ **then**
16 $\quad\quad\quad \tau_{PG_{j+1}} \leftarrow \tau_{PG_{j+1}} + 1$
17 $\quad\quad\quad T(q, M_I)[idx(l)] \leftarrow pop(P_{j+1})$
18 $\quad\quad\quad$ **break**

```
19              else
20                  if j == length(l) − 1 then
21                      // every protected group has achieved the minimum target number
22                      //find the item m_b that has the best quality among the remaining items
23                      m_b ← max(peek(P_0), peek(P_1), … , peek(P_G))
24                      τ_{PG_{g(m_b)}} ← τ_{PG_{g(m_b)}} + 1
25                      T(q, M_I)[idx(l)] ← pop(P_{g(m_b)})
26                  else
27                      //do nothing and move on to compare the minimum target number of next
28                      //protected group
29                  end
30              end
31      end
32  end
33  return T(q, M_I)
```

5.2 Table of Minimum Target Number

FA*IR generates rankings according to the minimum target number of each protected group in each position. However, computing the minimum target number in each position during the algorithm is very expensive. Hence, having a pre-computed table of minimum target number can save the resources.

We present the function that computes the minimum target number in a specified position (line 6) in the next section (§5.3).

Algorithm 2: This presents the algorithm to compute the table of minimum target number of each protected group up to k-th position.

Input:

- k: Size of ranking
- p: Minimum target group proportion
- α: Significance level

Output: L: 2-dimensional array; minimum target number of each protected group in each position up to k-th position

```
1  L ← [ ]
2  // number of protected-groups present
3  G ← length(p)
4  τ ← [0,0, … ,0], array of size G
5  for i ← 1 to k do
6      τ_p ← F^{−1}(α; i, p, τ)
7      L[i − 1] ← τ_p
8      τ ← τ_p
9  return L
```

Example: Let us assume that there are four groups of people: German, French, British and Italian. Each person is rated with the school math grade and a ranking of size 50 is to be generated based on the math grade. The German is considered as the non-protected group, while the other three, French, British and Italian are the protected group. If the minimum target group proportion for French, British and Italian is 0.3, 0.2, and 0.1 respectively, the table of minimum target number of each protected group up to the 14-th position is computed as follows:

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\boldsymbol{p = 0.3}$	0	0	1	1	1	2	2	2	2	2	3	3	4	4
$\boldsymbol{p = 0.2}$	0	0	0	0	1	1	1	1	2	2	2	2	2	3
$\boldsymbol{p = 0.1}$	0	0	0	0	0	0	0	1	1	1	1	1	1	1

Table 4 Minimum Target Number up to the position k=14 when three protected groups are present, in which the minimum target group proportion is 0.3, 0.2 and 0.1 for each group respectively.

5.3 Multinomial Inverse Cumulative Distribution Function

As **Algorithm 1** shows, to fulfill the fair representation condition, FA*IR algorithm ranks the items depends on the minimum target number of each protected group in every position of the ranking. In other words, the minimum target number of each protected group should be computed in the way, in which the ranking satisfies the *ranked group fairness condition* (§ 4.2) if it achieves the minimum target number.

To compute the minimum target number, we use the multinomial inverse cumulative distribution function. On the contrary to the multinomial cumulative distribution function, we provide a target probability and the multinomial inverse cumulative distribution function computes the value of observations, in which the cumulative probability of the random variables occurring less than or equal to this computed value of observations is equal to the target probability. [17]

For example, suppose that there are n independent trials, where $n \in \mathbb{N}$, and let $E_1, E_2, \dots, E_\gamma$, where $\gamma \in \mathbb{N}$, denote the possible random variables that can result from each trial. Each of the trials produces exactly one of the random variables $E_1, E_2, \dots, E_\gamma$ and each variable occurs on each trial with the probability $p = [p_1, p_2, \dots, p_r]$, where each $p_\gamma \in [0,1]$ and $p_\gamma \in \mathbb{R}$ for $\gamma \in \mathbb{N}$. Suppose that $p_{target} \in [0,1]$ is the target probability. Then,

$F^{-1}(p_{target}; n, p) = [x_1, x_2, \dots, x_\gamma]$, where $x_1, x_2, \dots, x_\gamma \in \mathbb{N}$, which

$$P(E_1 \leq x_1, E_2 \leq x_2, \dots, E_\gamma \leq x_\gamma) = F([x_1, x_2, \dots, x_\gamma]; n, p) = p_{target}$$

For computing the minimum target number of FA*IR algorithm, the significance level α is provided as the target probability. Therefore, the value of observations computed by the multinomial inverse cumulative distribution function produces the value of multinomial cumulative distribution function equal to α. Hence, to derive the value of multinomial cumulative distribution function equal to or larger than α, at least this computed value of observations must happen.

In the situation where more than two random variables are present, the multinomial inverse cumulative distribution function computes more than one solution. Therefore, the most

suitable solution for the FA*IR algorithm must be chosen from them. **The chosen solution of multinomial inverse cumulative distribution function** must fulfill the following conditions:

(i) The computed minimum target number of each protected group for the position k must always contain **at least** the target number computed for the position $k-1$.
(ii) The computed solution must show higher minimum target number for the protected group with higher *minimum target group proportion*.
(iii) From the position k to $k+1$, the minimum target number of each protected group must increase by maximum of 1.

Example: Suppose that there are four groups of people that are classified by their nationality: German, Italian, Spanish and French. The credit score of the people is rated by a bank and the bank wants to generate a ranking of 20 people with the best credit score. During this process, the German is considered as the non-protected group, while the other three are the protected group. The bank specified the minimum target group proportion of each protected group to be 0.3, 0.2, and 0.2 for Italian, Spanish and French respectively.

(i) Let us assume that the minimum target number of each protected group is 1, 1, and 0 for Italian, Spanish and French in the fourth position of the ranking at the *significance level* α. If the minimum target number computed for the fifth position is 1, 0, and 1 for Italian, Spanish and French correspondingly, this solution is invalid for the algorithm since the target number of Spanish is **reduced** in the next position.
(ii) If the minimum target number is 0, 0, and 1 for Italian, Spanish and French in the fourth position, the solution is invalid since Italian has the higher minimum target group proportion than French but the computes solution for Italian is smaller than the one for French.
(iii) Suppose that the minimum target number computed in the fourth position is 1, 1, and 0 for Italian, Spanish and French respectively. If the minimum target number computed for the fifth position is 1, 1, and 2 for Italian, Spanish and French, the solution is invalid since the minimum target number for French is increased by 2 from fourth position to fifth position.

Algorithm 3 shows the pseudocode of the multinomial inverse cumulative distribution function that satisfies the above conditions. To fulfil the condition (i), the algorithm takes τ, the minimum target number computed in the previous position $k-1$, as an input, so that no solutions that contain the value smaller than the minimum target number of previous position can be produced. In the algorithm, τ_{temp} is initialized with the minimum target number of previous position. (line 3) This τ_{temp} is used for several trials of different potential solutions that can be derived from the initial τ.

First, the algorithm tests if the initial τ achieves the value computed by multinomial cumulative distribution function larger than the given significance level. If it does, the initial τ is returned, so that the minimum target number for current position stays the same as the last position. (line 5-6) Otherwise, the algorithm derives several potential solutions from τ_{temp} and compares them to take the most suitable solution. For simplicity, let us denote

the initial τ_{temp} as $[i_1, i_2, \dots, i_G]$, where $i_1, i_2, \dots, i_G \in \mathbb{N}$. First, τ_{temp} is updated with $[i_1 + 1, i_2, \dots, i_G]$. τ is updated with the increased τ_{temp} and the value computed by the multinomial cumulative distribution function with the updated τ is saved in $mcdf$. (line 9-13) Next, τ_{temp} is updated with $[i_1, i_2 + 1, \dots, i_G]$ and its value computed by the multinomial cumulative distribution function is saved in $mcdf_{temp}$. $mcdf$ and $mcdf_{temp}$ are compared and only if $mcdf_{temp}$ is larger than α and $mcdf$, τ is updated with τ_{temp}. (line 14-23) This procedure is repeated till every element of τ_{temp} is once increased by 1.

After the comparison of $mcdf$ and $mcdf_{temp}$, if they present the value larger than α, the updated τ_{temp} should be set back to the initial sate of $k - 1$ to derive another trial solution, such as $[i_1, i_2 + 1, i_3, \dots i_G]$, or $[i_1, i_2, i_3 + 1, \dots, i_G]$. (line 25-26) If the value presented by $mcdf$ or $mcdf_{temp}$ is below α, the τ_{temp} is not set back to the initial state to produce the solution that contains at least the current τ_{temp}, such as $[i_1 + 1, i_2 + 1, \dots, i_G]$ or $[i_1, i_2 + 1, i_3 + 1, \dots, i_G]$. (line 27-30)

Algorithm 3: This presents the algorithm to compute the multinomial inverse cumulative distribution function that fulfils the above conditions.

Input:

- k: Size of ranking
- p: Minimum target group proportion, where the one for non-protected group is placed in the first position
- α: Significance level
- τ: Array of size $length(p)$; the minimum target number of each protected group computed in the previous position $k - 1$. Required to produce the continuous solution, so that no solutions that contain the value smaller than any components of τ can be produced

Output: Minimum target number τ of each group that should be contained in the k-th position of the ranking

1 $mcdf \leftarrow F(\tau; k, p)$
2 $G \leftarrow length(p)$
3 $\tau_{temp} \leftarrow copy(\tau)$
4 // if the minimum target number in $k - 1$ is also sufficient for k, minimum target number stays the same
5 **if** $mcdf > \alpha$, **then**
6 **return** τ
7 **else**
8 **for** $i \leftarrow 0$ **to** $G - 1$ **do**
9 $\tau_{temp}[i] \leftarrow \tau_{temp}[i] + 1$
10 //For the first loop, update τ with increased τ_{temp} no matter if $mcdf$ is larger than α
11 **if** $i == 0$, **then**
12 $\tau \leftarrow copy(\tau_{temp})$
13 $mcdf \leftarrow F(\tau; k, p)$
14 **else**
15 $mcdf_{temp} \leftarrow F(\tau_{temp}; k, p)$
16 //After the first loop, update τ with increased τ_{temp} only if the value computed by
17 //multinomial cumulative distribution function is equal to or larger than α
18 //and if the value is larger than the one from previous loop
19 **if** $mcdf_{temp} \geq \alpha$ **and** $mcdf_{temp} \geq mcdf$, **then**
20 $\tau \leftarrow copy(\tau_{temp})$

```
21                              mcdf ← F(τ; k, p)
22                          end
23                  end
24      //put τ_temp back into the initial state of k − 1 if mcdf or mcdf_temp is equal to or larger than α
25                  if mcdf_temp ≥ α or mcdf ≥ α, then
26                          τ_temp[i] ← τ_temp[i] − 1
27                  else
28                          //if not, do not subtract any value from τ_temp and continue, so that the minimum
29                          //target number of the current group is increased for sure
30                  end
31          end
32  end
33  return τ
```

5.4 Multinomial Cumulative Distribution Function [3]

To compute the multinomial cumulative distribution function, the corresponding representation function by Bruce Levin [3] is used. Bruce proved his function based on the representation of multinomial random variables as independent Poisson random variables on the condition that the sum of their observations being fixed. [18]

Let the random variables $E_1, E_2, \dots, E_\gamma$, where $\gamma \in \mathbb{N}$, have γ-category multinomial distribution with $n \in \mathbb{N}$ independent trials, in which each random variable can be produced with the probability $p_1, \dots, p_\gamma$ in each trial. If $x_1, \dots, x_\gamma$ denote the observations of each random variable, the multinomial cumulative distribution function can be computed with Bruce's representation function:

$$P(E_1 \leq x_1, \dots, E_\gamma \leq x_\gamma) = \frac{n!}{s^n e^{-s}} \left\{ \prod_{i=1}^{\gamma} P(X_i \leq x_i) \right\} P(W = n),$$

where s is any real number $s > 0$, X_i represents independent Poisson random variable with mean sp_i and W is the sum of independent truncated Poisson random variables, namely $W = \sum_1^\gamma Y_j$, where Y_j denotes truncated $Poisson(sp_j)$ with range $0,1, \dots, x_j$.

In his research paper, he stated that the function is applicable to any probability p and any value of the observations of each random variable, providing excellent accuracy.

Algorithm 4 presents the pseudocode of the representation function of of multinomial cumulative distribution function by Bruce. Note that in both *protected group count* (line 3) and *minimum target group proportion* (line 4), the value for non-protected group is also inserted since the multinomial cumulative distribution function considers all the groups present.

Algorithm 4: This presents the algorithm to compute the representation function of multinomial cumulative distribution function.

Input:

- k: Size of ranking
- p: Minimum target group proportion
- τ_{PG_n}: Protected Group Count

Output: Value computed by *Multinomial Cumulative Distribution Function*

1 //no restriction on non-protected group. Hence for non-protected group, the value of the observations can
2 //be arbitrary below k: $\tau_{PG_0} \leq k$
3 $\tau_{PG_n} \leftarrow concate([k], \tau_{PG_n})$
4 $p \leftarrow concate([1 - sum(p)], p)$
5 $G \leftarrow length(p)$
6 $s \leftarrow k$
7 **return** $\frac{k!}{s^k e^{-s}} \left\{ \prod_{i=0}^{G-1} P\left(independent\ Poisson(sp[i]) \leq \tau_{PG_n}[i] \right) \right\} P(W = k)$

5.5 Algorithm Correctness

We show that the rankings generated by FA*IR algorithm in multinomial use case (§ 5.1) achieves the constraints we define in the *formal problem statement*. (§ 4.4)

The rankings generated by FA*IR algorithm fulfils *in-group monotonicity* (§ 4.3), since the algorithm first separates the items into each group and when an item is to be selected, the algorithm takes the one that has the best quality in its group.

The *ranked group fairness condition* (§ 4.2) is satisfied, because FA*IR algorithm constructs a ranking not only based on the quality of items, but also based on the minimum target number of each protected group in every position.

The *selection*- and *ordering utility* (§ 4.3) are also maximized in the rankings since the *in-group monotonicity* is preserved, which means that for each group, the items that appear in the ranking have better quality than the ones that have not been selected into the ranking, and the utility loss is caused by items with lower quality that are yet ranked higher due to the *ranked group fairness condition*.

Hence, FA*IR algorithm in multinomial use case constructs rankings that satisfy the constraints defined in the *formal problem statement*.

6 Experiments

We present the data set used for the experiments (§6.1). We also explain the baseline and metrics used to analyse the result of the experiments (§6.2) and finally we present the results of the experiments with diagrams and a result summarizing table (§6.3).

6.1 Data set

Table 5 summarizes the properties of data sets that are used for the experiments. We use two public data sets and one synthetic data set. The public data sets contain sets of people classified with demographic attributes, in which each person is associated with a quality attribute. The choice of protected groups is not arbitrary but determined by social disadvantages since the groups with social disadvantages are more likely to be discriminated. Hence, the group that have more social disadvantages are chosen as the protected groups and the group with less social disadvantage is then chosen as the non-protected group.

	Dataset	n	k	Quality	Non-protected Group	Protected group	Protected %
D1	German Credit [14]	1,000	500	Credit Rating	Male Single	1. Female divorced/separated/married 2. Male divorced/separated 3. Male married/widowed	1. 31 % 2. 5 % 3. 9.2 %
D2	COMPAS [15]	19,289	1000	Recidivism Score	Caucasian	1. African-American 2. Hispanic 3. Asian	1. 46.7 % 2. 15.1 % 3. 0.56 %
D3	Synthetic Dataset [1]	500	300	Quality	$(0,)$	1. $(1,)$ 2. $(2,)$ 3. $(3,)$	1. 24 % 2. 25.6 % 3. 22.8 %

Table 5 Data sets for the experiments

German Credit: The German Credit data set was originally provided by Professor Dr. Hans Hoffman. [14] The data set was generated based on credit ratings by SCHUFA, which is a German Credit Rating Agency. The data set provides the information of 1000 credit applicants in Germany with various categorial and numerical attributes, such as sex, family status or credit amount etc. We use, as Yang and Stoyanovich [6], the credit-worthiness as the quality of applicants and for the experiment, different family status and gender are used to denote different groups: *Male single, Female divorced/separated/married, Male divorced/separated, Male married/widowed*. In the data analysis, the three groups, *Female divorced/separated/married*, *Male divorced/separated, and Male married/widowed,* show relatively lower credit-worthiness than the *Male single* group. Hence, these three groups are treated as protected groups, while the *Male single* group is considered as non-protected group.

Note that two attributes, family status and gender, are used to specify each group of people. With the experiment on German Credit data set, we also present that combinations of attributes are also possible to classify the groups.

COMPAS: The data set was generated by the Correctional Offender Management Profiling for Alternative Sanctions (COMPAS), which is a tool to assess a criminal defendant's likelihood to become a recidivist. [15] The data set is created based on more than 10,000 criminal defendants in the USA and provides scores, which shows how likely is the observed defendant to reoffend, along with several categorical attributes, such as sex, ethnicity or family status. For the experiments, we generate a ranking of the criminal defendants, who are least likely to reoffend and to classify each defendant, we use their ethnic background: *Caucasian, African-American, Hispanic and Asian*. Since it has been analysed that that

Caucasian defendants were “often predicted to be less risky than they were”, [15] we consider the Caucasian to be the non-protected group, while the other three groups are considered as the protected groups.

Synthetic Data Set: The synthetic data set was generated by the data set generator implemented by Meike Zehlike. [1] The data set generator produces a data set which consists of the given number of groups and desired size of data set. Each generated item is composed of a protected attribute and a quality and each group has different quality mean and variance. For the experiment, we use a data set of four different groups. Each group is denoted as *(0,), (1,), (2,) and (3,)*, in which *(0,)* shows the highest quality mean. Hence, we use *(0,)* as the non-protected group while the other three groups are considered as the protected groups.

6.2 Baseline and metrics

Baseline: Color-blind ranking: The ranking which only considers the quality of the items without considering the ranked group fairness.

Utility: With the calculated *ordering*- and *selection utility* according to (§4.3), we report the loss of utility in the generated ranking.

NDCG: The Normalized Discounted Cumulative Gain (NDCG) is one of the most popular measure to evaluate the quality of ranked items in information retrieval. [16] With NDCG, we report a weighted summation of the quality of the items in the ranking: $\sum_{i=1}^{k} \frac{1}{\log(1+i)} * v(m, q)$, where $m \in T(q, M_I)$.

6.3 Results

Table 6 summarizes the results of the experiments with the baseline and metrics (§ 6.2) described above. The percentage of protected groups in the rankings generated by FA*IR algorithm achieves very close value to the provided target protected proportion p. Moreover, FA*IR shows not very dramatic utility change in the generated rankings with small ordering utility loss and very small selection utility loss. The NDCG in the rankings generated by FA*IR tends to show a value a bit lower than the NDCG in the color-blind rankings, but the drop of NDCG is also not very noticeable.

We illustrate the results of the experiments in diagrams in the following sections (§ 6.3.1-6.3.3). See appendix for the generated rankings, where each group is shown with different colors. (§ 9)

	Method	% Prot. Output	NDCG	Ordering utility loss	Selection utility loss
D1	Color-blind	1. 24.8 % 2. 4.4 % 3. 4.4 %	46.4079	0.0	0.0
	FA*IR p: 1. 30 % 2. 20 % 3. 10 %	1. 29.6 % 2. 20 % 3. 10.8 %	44.7806	-0.3864	- 0.0697
D2	Color-blind	1. 20.3 % 2. 18.7 % 3. 0.015 %	157.1045	0.0	0.0
	FA*IR p: 1. 20 % 2. 20 % 3. 10 %	1. 19.9 % 2. 19.9 % 3. 10.3 %	154.5328	-0.4249	- 0.0615
D3	Color-blind	1. 39.3 % 2. 17.3 % 3. 7.3 %	34.6345	0.0	0.0
	FA*IR p: 1. 30 % 2. 20 % 3. 20 %	1. 30.3 % 2. 20.3 % 3. 20.3 %	32.9477	-0.4561	- 0.0281

Table 6 Results of the experiments

6.3.1 German Credit

Figure 1 shows the data distribution of German Credit data set. The *Male single* group has the highest mean score of credit-worthiness, which we use as quality, while the *Male married/widowed* group has the lowest mean score. The *Male divorced/separated* group has the second highest mean score and the *Female divorced/separated/married* group has the third highest mean score.

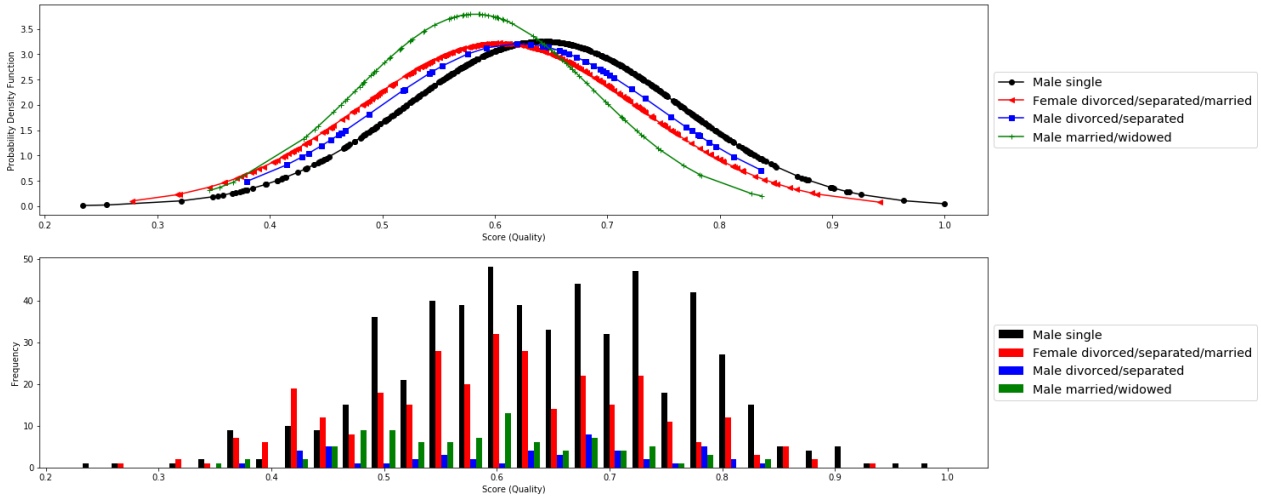

Figure 1 Data distribution of German Credit data set

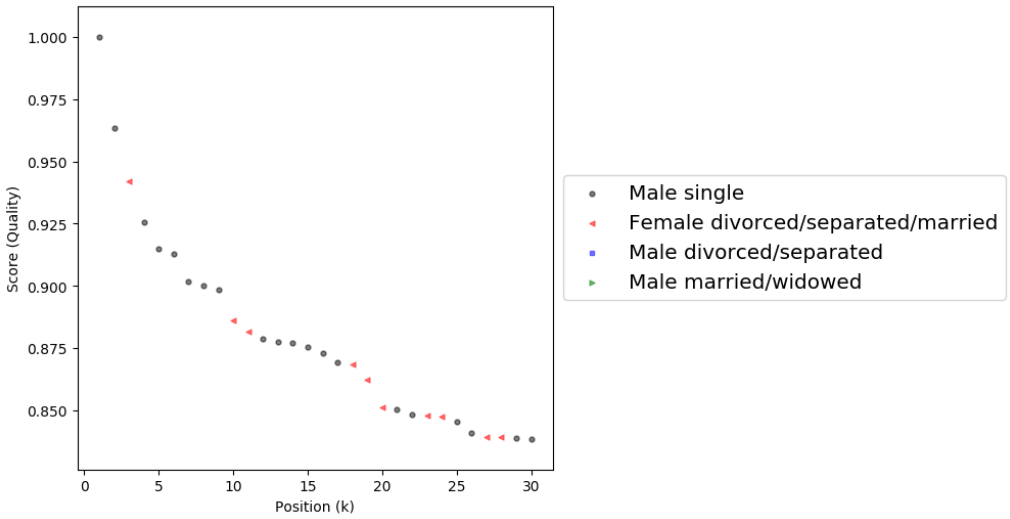

Figure 2 Color-blind ranking of German Credit data set with k=30

Despite the higher mean score, the highest score of the *Male divorced/separated* group is approximately 0.84, which is lower than the highest scores of *Male single* and *Female divorced/separated/married* groups. Hence, as **figure 2** shows, the people from *Male divorced/separated* group are not ranked in the first top 30 positions in the color-blind ranking. Instead, the people from *Male single* and *Female divorced/separated/married* groups appear in the top 30 positions of the ranking.

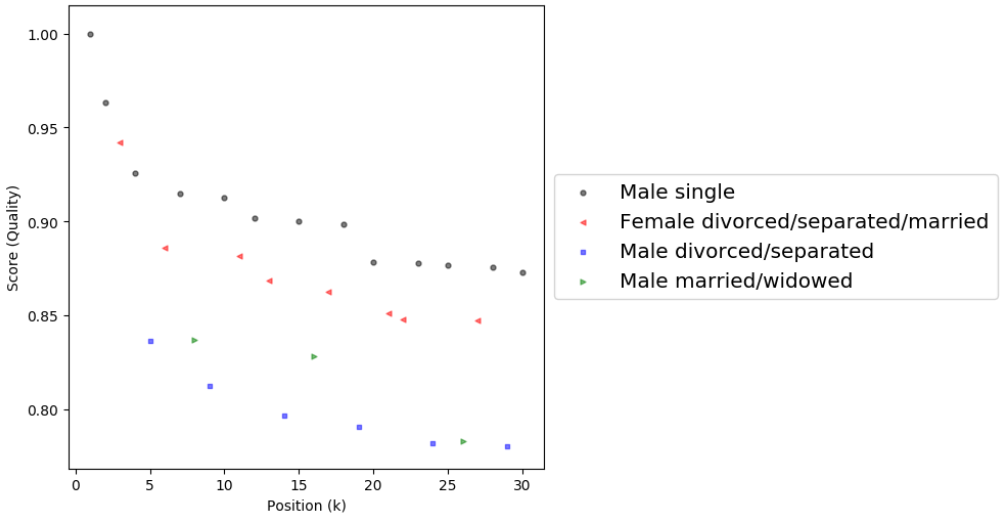

Figure 3 Ranking of German Credit data set generated by FA*IR with k=30

Figure 3 shows the top 30 positions of the ranking of German Credit data set generated by FA*IR. Different from the color-blind ranking, all four groups appear in the top 30 positions. Also, the points are falling along a curve within the group, which shows that the *in-group monotonicity* is fulfilled.

6.3.2 COMPAS

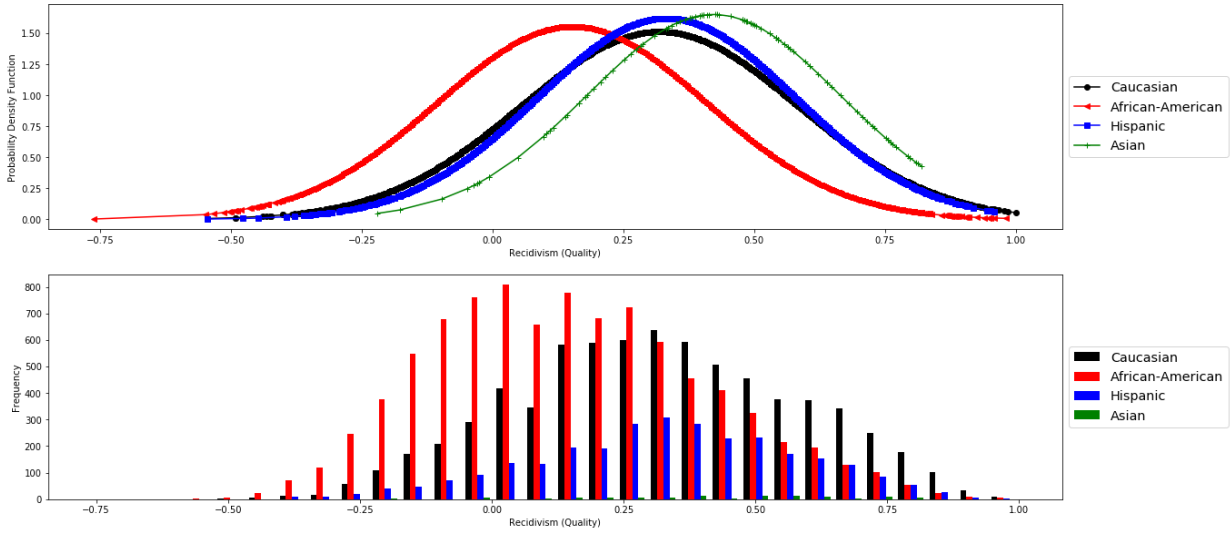

Figure 4 Data distribution of COMPAS data set

Figure 4 shows the data distribution of COMPAS data set. The *Asian* defendants have the highest mean quality representing how **unlikely** to reoffend, while the *African-American* defendants have the lowest mean quality. Yet, the highest quality of *Asian* defendants first appears approximately at 0.80, which is a lot lower than the highest qualities of the other three ethnic groups. The highest qualities among the entire defendants are instead possessed by the defendants of three ethnic groups, *Caucasian, African-American,* and *Hispanic,* showing few differences in the quality among them.

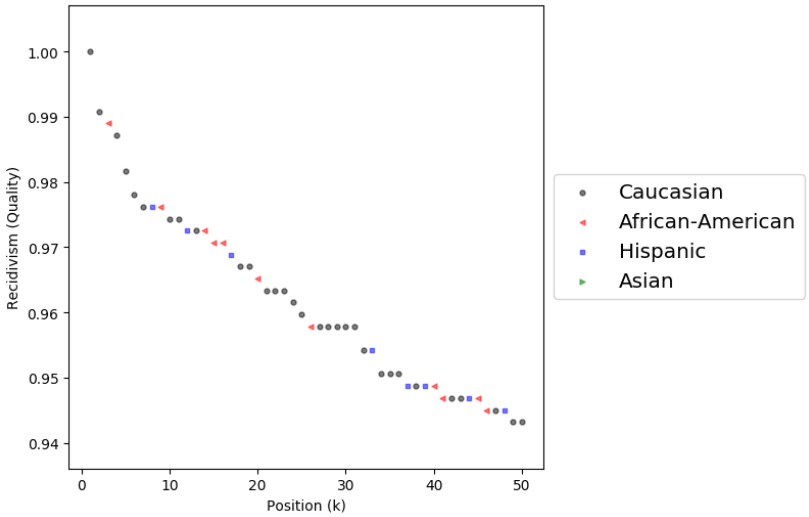

Figure 5 Color-blind ranking of COMPAS data set with k=50

Figure 5 shows the color-blind ranking of COMPAS data set up to the position 50. As it is shown in the data distribution, the three ethnic groups, *Caucasian, African-American,* and *Hispanic*, that possess the best qualities, appear in the first top 50 positions, while the *Asian*

defendants are not even ranked. Among the three ethnic groups, the *Caucasian* defendants tends to appear more in the ranking than the *Hispanic* and *African-American* defendants.

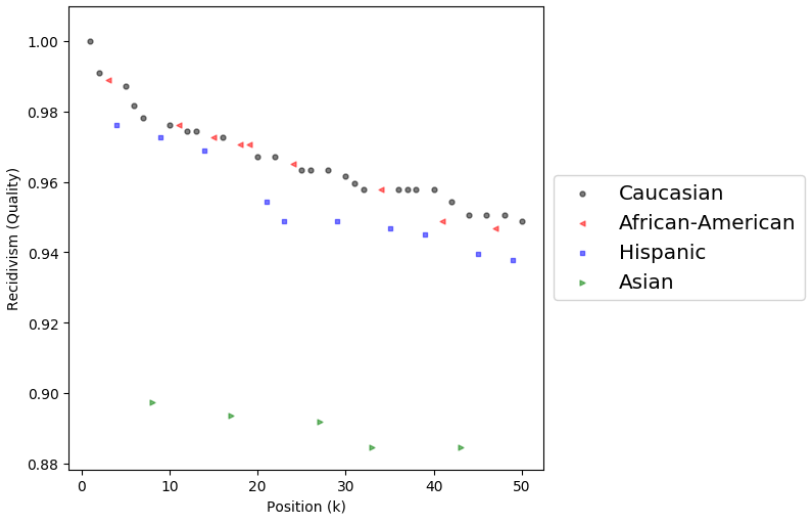

Figure 6 Ranking of COMPAS data set generated by FA*IR with k=50

In the ranking that is generated by FA*IR, which is shown by **Figure 6**, all four ethnic groups appear in the top 50 positions. Note that an Asian defendant is already ranked in the top 10 positions, even though the quality of this defendant is a lot lower than the other three ethnic groups. It is also observable in the figure that the *in-group monotonicity* is fulfilled.

6.3.3 Synthetic data set

Figure 7 illustrates the data distribution of the synthetic data set. It is very clearly shown that all the groups possess different mean quality and variance, where the group (0,) has both the highest mean quality and highest qualities among the data set. The group (3,) has the lowest quality mean, having its highest quality at about 0.45.

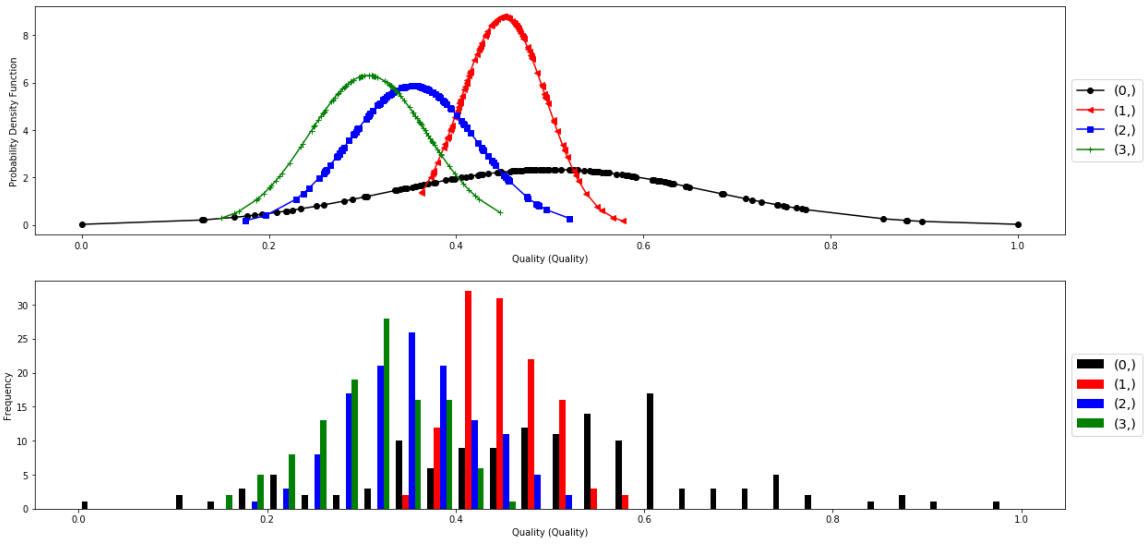

Figure 7 Data distribution of synthetic data set

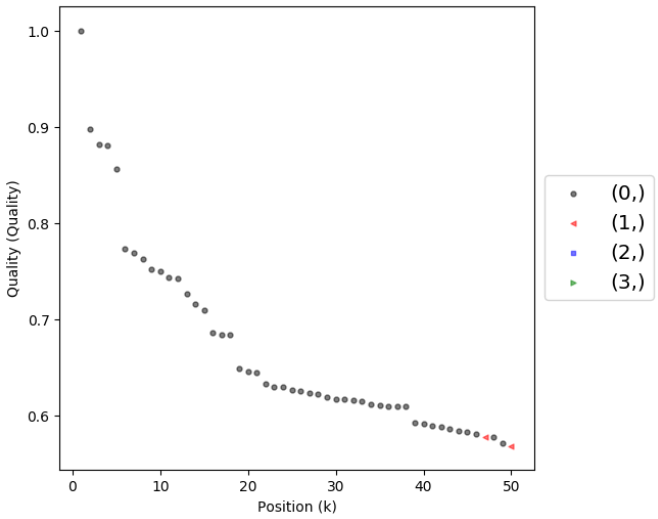

Figure 8 Color-blind ranking of synthetic data set with k=50

Figure 8 shows the color-blind ranking of the synthetic data set up to the position 50. The ranking is mostly filled with the group (0,) and only two items from the group (1,) appear in the position 47 and 50. The two other groups, (2,) and (3,), are completely excluded from the color-blind ranking.

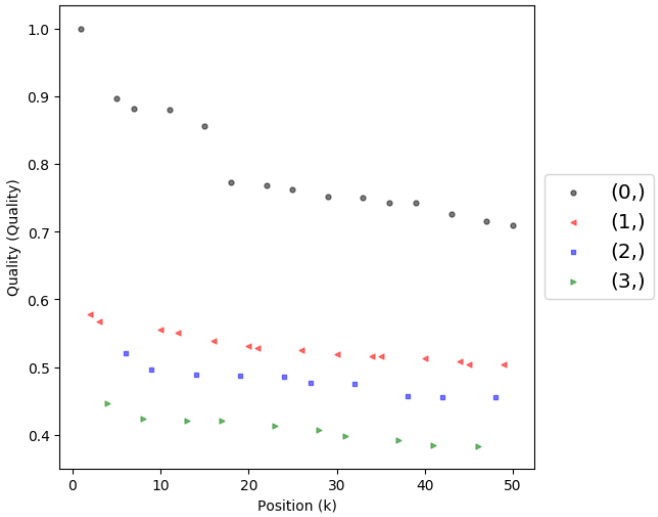

Figure 9 Ranking of synthetic data set generated by FA*IR with k=50

As it is shown by **Figure 9**, in the ranking generated by the FA*IR, all four groups appear in the top 50 positions, although the qualities of the three groups, (1,), (2,) and (3,), are a lot lower than the quality of the group (0,). It is also visible from the trends of the scatter plots that the *in-group monotonicity* is preserved.

7 Conclusion

The main challenge of this thesis is to further develop the FA*IR algorithm from binomial use case [1] to the multinomial use case. Indeed, the FA*IR algorithm in multinomial use case we present can generate rankings that guarantees the *ranked group fairness* with very small utility loss. As it is shown in the experiments chapter (§ 6), the algorithm is applicable on various shapes of data distribution. Further, the *ranked group fairness* and the utility loss in the generated rankings can be controlled with the *minimum target group proportion* p, which shows the flexibility of the algorithm.

As we mention in the problem statement (§ 4.2) the significance level must be adjusted in each prefix since the *ranked group fairness condition* forms multiple statistical hypothesis testing. [13] Therefore, we plan to include the adjustment of the significance level in the future work.

Especially for the experiments, complex cases with bigger number of protected groups and the combinations of more than two protected attributes for the classification of the groups are considered to be our future work.

8 References

[1] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates 2017. FA*IR: A Fair Top-k Ranking Algorithm. *arXiv: 1706.06369v2 (Sep 2017)*

[2] L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi 2017. Ranking with Fairness Constraints. *arXiv:1704.06840v1 (Apr 2017)*

[3] Bruce Levin 1981. A Representation for Multinomial Cumulative Distribution Functions. *The Annals of Statistics (1981) Vol. 9, No. 5, 1123-1126*

[4] Batya Friedman and Helen Nissenbaum. Bias in computer systems. *ACM Tansactions on Information Systems 14, 3(1996), 330-347*

[5] Xiaofang Zhou, Haruo Yokota, Ke Deng and Qing Liu. Database Systems for Advanced Applications. *DASFAA 2009, Brisbane, Australia, April 21-23, 2009. Proceedings*

[6] Ke Yang and Julia Stoyanovich 2016. Measuring Fairness in Ranked Outputs. *arXiv:1610.08559v1* (Oct 2016)

[7] The Pennsylvania State University Department of Statistics Online Programs. 1.7 – The Multinomial Distribution. *https://onlinecourses.science.psu.edu/stat504/node/40*

[8] The Pennsylvania State University Department of Statistics Online Programs. 3.2 - Hypothesis Testing (P-value approach). *https://onlinecourses.science.psu.edu/statprogram/node/138*

[9] Associate Professor at Adrews University hosting the Berrien Co. Math & Science Center for Berrien RESA - Andrews University. Applied Statistics – Lesson 8 Hypothesis Testing. *https://www.andrews.edu/~calkins/math/edrm611/edrm08.htm*

[10] BBC Bitesize- *What is an algorithm?* *http://www.bbc.co.uk/guides/zqrq7ty*

[11] Oxford Living Dictionaries- *ranking* *https://en.oxforddictionaries.com/definition/ranking*

[12] The Pennsylvania State University Department of Statistics Online Programs. 4.2 – Binomial Distributions. *https://onlinecourses.science.psu.edu/stat500/node/22*

[13] Yoav Benjamini and Daniel Yekutieli 2001, The Control Of The False Discovery Rate In Multiple Testing Under Dependency. *The Annals of Statistics (2001) Vol. 29, No. 4, 1165-1188*

[14] Statlog (German Credit - SCHUFA). *Fairness Measures*. *http://fairness-measures.org/Pages/Datasets/Schufa.html*

[15] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016 Machine Bias. *ProPublica (May 2016)*. *https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm*

[16] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, Wei Chen. A Theoretical Analysis of NDCG Type Ranking Measures. *arXiv: 1304.6480v1 (April 2013)*

[17] Information Technology Laboratory. *Engineering Statistics-1.3.6.2 Related Distributions Percent Point Function*. *https://www.itl.nist.gov/div898/handbook/eda/section3/eda362.htm*

[18] The Pennsylvania State University Department of Statistics Online Programs. 1.7.7 – Relationship between the Multinomial and Poisson. *https://onlinecourses.science.psu.edu/stat504/node/48*

9 Appendix

9.1 German Credit

9.1.1 Color-Blind

1. {'Index': 654, 'k': 1, 'Score': 1.0000000000000000, 'Group': 0, 'Utility': 1.4426950408889634}
2. {'Index': 891, 'k': 2, 'Score': 0.9634788985234082, 'Group': 0, 'Utility': 0.87699628746322411}
3. {'Index': 827, 'k': 3, 'Score': 0.9420066211780169, 'Group': 1, 'Utility': 0.67951414042904668}
4. {'Index': 769, 'k': 4, 'Score': 0.9257552084212175, 'Group': 0, 'Utility': 0.57520405184261703}
5. {'Index': 263, 'k': 5, 'Score': 0.9148915339734488, 'Group': 0, 'Utility': 0.51061068725235315}
6. {'Index': 30, 'k': 6, 'Score': 0.9129079211152771, 'Group': 0, 'Utility': 0.46914186739735608}
7. {'Index': 808, 'k': 7, 'Score': 0.9017068827878809, 'Group': 0, 'Utility': 0.43362934937784053}
8. {'Index': 243, 'k': 8, 'Score': 0.9002629392657567, 'Group': 0, 'Utility': 0.4097273207990329}
9. {'Index': 66, 'k': 9, 'Score': 0.8984551526487219, 'Group': 0, 'Utility': 0.39019411503288365}
10. {'Index': 803, 'k': 10, 'Score': 0.8861122927489402, 'Group': 1, 'Utility': 0.36953752851551236}
11. {'Index': 147, 'k': 11, 'Score': 0.8818414203072478, 'Group': 1, 'Utility': 0.35487909390176969}
12. {'Index': 958, 'k': 12, 'Score': 0.8786100728116797, 'Group': 0, 'Utility': 0.34254480317740743}
13. {'Index': 716, 'k': 13, 'Score': 0.8776749386857043, 'Group': 0, 'Utility': 0.33257138025631994}
14. {'Index': 673, 'k': 14, 'Score': 0.8769908749974441, 'Group': 0, 'Utility': 0.32384587059741282}
15. {'Index': 872, 'k': 15, 'Score': 0.875584840813496, 'Group': 0, 'Utility': 0.31580047692979579}
16. {'Index': 773, 'k': 16, 'Score': 0.873061748425369, 'Group': 0, 'Utility': 0.3081524906188095}
17. {'Index': 950, 'k': 17, 'Score': 0.869156815803223, 'Group': 0, 'Utility': 0.30070762123549893}
18. {'Index': 264, 'k': 18, 'Score': 0.8683481684573157, 'Group': 1, 'Utility': 0.29491124611559849}
19. {'Index': 293, 'k': 19, 'Score': 0.8623399919929922, 'Group': 1, 'Utility': 0.28785616111480949}
20. {'Index': 956, 'k': 20, 'Score': 0.8510703754589874, 'Group': 1, 'Utility': 0.27954150211334466}
21. {'Index': 55, 'k': 21, 'Score': 0.8503116382578004, 'Group': 0, 'Utility': 0.27508895496856239}
22. {'Index': 206, 'k': 22, 'Score': 0.8482735447242, 'Group': 0, 'Utility': 0.27053902393273233}
23. {'Index': 492, 'k': 23, 'Score': 0.8481272459841925, 'Group': 1, 'Utility': 0.26687000638102354}
24. {'Index': 320, 'k': 24, 'Score': 0.8475226174110342, 'Group': 1, 'Utility': 0.26329770501343797}
25. {'Index': 247, 'k': 25, 'Score': 0.845611392800435, 'Group': 0, 'Utility': 0.25954153995508755}
26. {'Index': 107, 'k': 26, 'Score': 0.841062589828185, 'Group': 0, 'Utility': 0.25518938710332406}
27. {'Index': 375, 'k': 27, 'Score': 0.839406055890173, 'Group': 1, 'Utility': 0.25190712434574969}
28. {'Index': 913, 'k': 28, 'Score': 0.8392351823226774, 'Group': 1, 'Utility': 0.24923120055241729}
29. {'Index': 453, 'k': 29, 'Score': 0.8387783785305106, 'Group': 0, 'Utility': 0.24661267324644418}
30. {'Index': 702, 'k': 30, 'Score': 0.8386686541591615, 'Group': 0, 'Utility': 0.2442259112275586}

9.1.2 FA*IR

1. {'Index': 654, 'k': 1, 'Score': 1.0000000000000000, 'Group': 0, 'Utility': 1.4426950408889634}
2. {'Index': 891, 'k': 2, 'Score': 0.9634788985234082, 'Group': 0, 'Utility': 0.87699628746322411}
3. {'Index': 827, 'k': 3, 'Score': 0.9420066211780169, 'Group': 1, 'Utility': 0.67951414042904668}
4. {'Index': 769, 'k': 4, 'Score': 0.9257552084212175, 'Group': 0, 'Utility': 0.57520405184261703}
5. {'Index': 500, 'k': 5, 'Score': 0.8366075105736831, 'Group': 2, 'Utility': 0.46691954190375745}
6. {'Index': 803, 'k': 6, 'Score': 0.8861122927489402, 'Group': 1, 'Utility': 0.45537163839713968}
7. {'Index': 263, 'k': 7, 'Score': 0.9148915339734488, 'Group': 0, 'Utility': 0.43996982633826376}
8. {'Index': 44, 'k': 8, 'Score': 0.8370719344477783, 'Group': 3, 'Utility': 0.3809678551213882}
9. {'Index': 170, 'k': 9, 'Score': 0.8125237519389911, 'Group': 2, 'Utility': 0.35287458188243037}
10. {'Index': 30, 'k': 10, 'Score': 0.9129079211152771, 'Group': 0, 'Utility': 0.3807121734928412}
11. {'Index': 147, 'k': 11, 'Score': 0.8818414203072478, 'Group': 1, 'Utility': 0.35487909390176969}
12. {'Index': 808, 'k': 12, 'Score': 0.9017068827878809, 'Group': 0, 'Utility': 0.35154958524416119}
13. {'Index': 264, 'k': 13, 'Score': 0.8683481684573157, 'Group': 1, 'Utility': 0.32903725080648777}
14. {'Index': 962, 'k': 14, 'Score': 0.7967230489334141, 'Group': 2, 'Utility': 0.29420542078914852}
15. {'Index': 243, 'k': 15, 'Score': 0.9002629392657567, 'Group': 0, 'Utility': 0.32470121949370734}
16. {'Index': 799, 'k': 16, 'Score': 0.8283012163519716, 'Group': 3, 'Utility': 0.29235398671605883}
17. {'Index': 293, 'k': 17, 'Score': 0.8623399919929922, 'Group': 1, 'Utility': 0.2983491620540431}
18. {'Index': 66, 'k': 18, 'Score': 0.8984551526487219, 'Group': 0, 'Utility': 0.30513627859357817}
19. {'Index': 503, 'k': 19, 'Score': 0.7906908695906277, 'Group': 2, 'Utility': 0.26393909648427649}
20. {'Index': 958, 'k': 20, 'Score': 0.8786100728116797, 'Group': 0, 'Utility': 0.28858715637145071}
21. {'Index': 956, 'k': 21, 'Score': 0.8510703754589874, 'Group': 1, 'Utility': 0.27533441817802512}
22. {'Index': 492, 'k': 22, 'Score': 0.8481272459841925, 'Group': 1, 'Utility': 0.27049236502350382}
23. {'Index': 716, 'k': 23, 'Score': 0.8776749386857043, 'Group': 0, 'Utility': 0.27616742369326486}
24. {'Index': 730, 'k': 24, 'Score': 0.7818367152672885, 'Group': 2, 'Utility': 0.24289123215845129}
25. {'Index': 673, 'k': 25, 'Score': 0.8769908749974441, 'Group': 0, 'Utility': 0.26917277151339636}
26. {'Index': 720, 'k': 26, 'Score': 0.7833385686282466, 'Group': 3, 'Utility': 0.23767516429838301}
27. {'Index': 320, 'k': 27, 'Score': 0.8475226174110342, 'Group': 1, 'Utility': 0.25434291767598394}
28. {'Index': 872, 'k': 28, 'Score': 0.875584840813496, 'Group': 0, 'Utility': 0.26002611146197185}
29. {'Index': 794, 'k': 29, 'Score': 0.7805089735657651, 'Group': 2, 'Utility': 0.22948064636705459}
30. {'Index': 773, 'k': 30, 'Score': 0.873061748425369, 'Group': 0, 'Utility': 0.25424140989374072}

9.2 COMPAS

9.2.1 Color-Blind

1. {'Index': 18363, 'k': 1, 'Utility': 1.4426950408889634, 'Group': 0, 'Recidivism': 1.00000000000000000}
2. {'Index': 13126, 'k': 2, 'Utility': 0.90190370257347807, 'Group': 0, 'Recidivism': 0.9908424908424909}
3. {'Index': 13351, 'k': 3, 'Utility': 0.71342062461542144, 'Group': 1, 'Recidivism': 0.989010989010989}
4. {'Index': 12765, 'k': 4, 'Utility': 0.61336910206525785, 'Group': 0, 'Recidivism': 0.9871794871794871}
5. {'Index': 2878, 'k': 5, 'Utility': 0.54788882020415475, 'Group': 0, 'Recidivism': 0.9816849816849816}
6. {'Index': 11019, 'k': 6, 'Utility': 0.50260387330667933, 'Group': 0, 'Recidivism': 0.978021978021978}
7. {'Index': 2123, 'k': 7, 'Utility': 0.46944838632101193, 'Group': 0, 'Recidivism': 0.9761904761904762}
8. {'Index': 16134, 'k': 8, 'Utility': 0.44428343204405152, 'Group': 2, 'Recidivism': 0.9761904761904762}
9. {'Index': 12791, 'k': 9, 'Utility': 0.42395413709603147, 'Group': 1, 'Recidivism': 0.9761904761904762}
10. {'Index': 13502, 'k': 10, 'Utility': 0.406339253182599, 'Group': 0, 'Recidivism': 0.9743589743589745}
11. {'Index': 2415, 'k': 11, 'Utility': 0.39211089657718201, 'Group': 0, 'Recidivism': 0.9743589743589745}
12. {'Index': 7213, 'k': 12, 'Utility': 0.37916049675536584, 'Group': 2, 'Recidivism': 0.9725274725274726}
13. {'Index': 17696, 'k': 13, 'Utility': 0.36851320417099653, 'Group': 0, 'Recidivism': 0.9725274725274726}
14. {'Index': 14963, 'k': 14, 'Utility': 0.35912461007245794, 'Group': 1, 'Recidivism': 0.9725274725274726}
15. {'Index': 16292, 'k': 15, 'Utility': 0.35010456578349386, 'Group': 1, 'Recidivism': 0.9706959706959707}
16. {'Index': 2027, 'k': 16, 'Utility': 0.34261308726799161, 'Group': 1, 'Recidivism': 0.9706959706959707}
17. {'Index': 4572, 'k': 17, 'Utility': 0.33520410176221871, 'Group': 2, 'Recidivism': 0.9688644688644689}
18. {'Index': 11760, 'k': 18, 'Utility': 0.32842690029417093, 'Group': 0, 'Recidivism': 0.9670329670329669}
19. {'Index': 6273, 'k': 19, 'Utility': 0.322803534738345, 'Group': 0, 'Recidivism': 0.9670329670329669}
20. {'Index': 10720, 'k': 20, 'Utility': 0.31702885590267016, 'Group': 1, 'Recidivism': 0.9652014652014651}
21. {'Index': 17668, 'k': 21, 'Utility': 0.31166507024945372, 'Group': 0, 'Recidivism': 0.9633699633699634}
22. {'Index': 3128, 'k': 22, 'Utility': 0.30724660835787421, 'Group': 0, 'Recidivism': 0.9633699633699634}
23. {'Index': 1769, 'k': 23, 'Utility': 0.30313204709452324, 'Group': 0, 'Recidivism': 0.9633699633699634}
24. {'Index': 18393, 'k': 24, 'Utility': 0.29871871853827492, 'Group': 0, 'Recidivism': 0.9615384615384616}
25. {'Index': 12747, 'k': 25, 'Utility': 0.29456062719668619, 'Group': 0, 'Recidivism': 0.9597069597069597}
26. {'Index': 15407, 'k': 26, 'Utility': 0.29063193866046155, 'Group': 1, 'Recidivism': 0.9578754578754579}
27. {'Index': 13769, 'k': 27, 'Utility': 0.28745998480900375, 'Group': 0, 'Recidivism': 0.9578754578754579}
28. {'Index': 11665, 'k': 28, 'Utility': 0.28446430199134165, 'Group': 0, 'Recidivism': 0.9578754578754579}
29. {'Index': 10108, 'k': 29, 'Utility': 0.28162889429467541, 'Group': 0, 'Recidivism': 0.9578754578754579}
30. {'Index': 5488, 'k': 30, 'Utility': 0.27893972832058672, 'Group': 0, 'Recidivism': 0.9578754578754579}
31. {'Index': 4164, 'k': 31, 'Utility': 0.27638443457323364, 'Group': 0, 'Recidivism': 0.9578754578754579}
32. {'Index': 6724, 'k': 32, 'Utility': 0.27290444463167279, 'Group': 0, 'Recidivism': 0.9542124542124543}
33. {'Index': 4319, 'k': 33, 'Utility': 0.27059412886216977, 'Group': 2, 'Recidivism': 0.9542124542124543}
34. {'Index': 7639, 'k': 34, 'Utility': 0.26735763534534068, 'Group': 0, 'Recidivism': 0.9505494505494505}
35. {'Index': 962, 'k': 35, 'Utility': 0.26525587470704881, 'Group': 0, 'Recidivism': 0.9505494505494505}
36. {'Index': 375, 'k': 36, 'Utility': 0.26324316241611168, 'Group': 0, 'Recidivism': 0.9505494505494505}
37. {'Index': 11409, 'k': 37, 'Utility': 0.26080975324287853, 'Group': 2, 'Recidivism': 0.9487179487179487}
38. {'Index': 3747, 'k': 38, 'Utility': 0.25896055269609497, 'Group': 0, 'Recidivism': 0.9487179487179487}
39. {'Index': 142, 'k': 39, 'Utility': 0.25718323423659539, 'Group': 2, 'Recidivism': 0.9487179487179487}
40. {'Index': 2712, 'k': 40, 'Utility': 0.25547314867647353, 'Group': 1, 'Recidivism': 0.9487179487179487}
41. {'Index': 8596, 'k': 41, 'Utility': 0.25333604721364633, 'Group': 1, 'Recidivism': 0.9468864468864469}
42. {'Index': 8510, 'k': 42, 'Utility': 0.25175114797417303, 'Group': 0, 'Recidivism': 0.9468864468864469}
43. {'Index': 292, 'k': 43, 'Utility': 0.25022172208267818, 'Group': 0, 'Recidivism': 0.9468864468864469}
44. {'Index': 15999, 'k': 44, 'Utility': 0.24874452343254067, 'Group': 2, 'Recidivism': 0.9468864468864469}

45. {'Index': 4147, 'k': 45, 'Utility': 0.24731656711301081, 'Group': 1, 'Recidivism': 0.9468864468864469}
46. {'Index': 12633, 'k': 46, 'Utility': 0.24545940644852038, 'Group': 1, 'Recidivism': 0.9450549450549451}
47. {'Index': 4104, 'k': 47, 'Utility': 0.24412448291681621, 'Group': 0, 'Recidivism': 0.9450549450549451}
48. {'Index': 7194, 'k': 48, 'Utility': 0.24283108485603608, 'Group': 2, 'Recidivism': 0.9450549450549451}
49. {'Index': 4746, 'k': 49, 'Utility': 0.24110886922563318, 'Group': 0, 'Recidivism': 0.9432234432234433}
50. {'Index': 1966, 'k': 50, 'Utility': 0.2398945251724941, 'Group': 0, 'Recidivism': 0.9432234432234433}

9.2.2 FA*IR

1. {'Index': 18363, 'k': 1, 'Utility': 1.4426950408889634, 'Group': 0, 'Recidivism': 1.0000000000000000}
2. {'Index': 13126, 'k': 2, 'Utility': 0.90190370257347807, 'Group': 0, 'Recidivism': 0.9908424908424909}
3. {'Index': 13351, 'k': 3, 'Utility': 0.71342062461542144, 'Group': 1, 'Recidivism': 0.989010989010989}
4. {'Index': 16134, 'k': 4, 'Utility': 0.60654124564152589, 'Group': 2, 'Recidivism': 0.9761904761904762}
5. {'Index': 12765, 'k': 5, 'Utility': 0.55095536210828244, 'Group': 0, 'Recidivism': 0.9871794871794871}
6. {'Index': 2878, 'k': 6, 'Utility': 0.50448628481719116, 'Group': 0, 'Recidivism': 0.9816849816849816}
7. {'Index': 11019, 'k': 7, 'Utility': 0.47032915252424085, 'Group': 0, 'Recidivism': 0.978021978021978}
8. {'Index': 6399, 'k': 8, 'Utility': 0.40844067861460648, 'Group': 3, 'Recidivism': 0.8974358974358975}
9. {'Index': 7213, 'k': 9, 'Utility': 0.42236331481799766, 'Group': 2, 'Recidivism': 0.9725274725274726}
10. {'Index': 2123, 'k': 10, 'Utility': 0.40710304877128806, 'Group': 0, 'Recidivism': 0.9761904761904762}
11. {'Index': 12791, 'k': 11, 'Utility': 0.39284794713465787, 'Group': 1, 'Recidivism': 0.9761904761904762}
12. {'Index': 13502, 'k': 12, 'Utility': 0.37987454665509346, 'Group': 0, 'Recidivism': 0.9743589743589745}
13. {'Index': 2415, 'k': 13, 'Utility': 0.3692072026722602, 'Group': 0, 'Recidivism': 0.9743589743589745}
14. {'Index': 4572, 'k': 14, 'Utility': 0.35777197500627167, 'Group': 2, 'Recidivism': 0.9688644688644689}
15. {'Index': 14963, 'k': 15, 'Utility': 0.35076514043591556, 'Group': 1, 'Recidivism': 0.9725274725274726}
16. {'Index': 17696, 'k': 16, 'Utility': 0.34325952705528973, 'Group': 0, 'Recidivism': 0.9725274725274726}
17. {'Index': 5774, 'k': 17, 'Utility': 0.30922419973527926, 'Group': 3, 'Recidivism': 0.8937728937728937}
18. {'Index': 16292, 'k': 18, 'Utility': 0.32967094158316401, 'Group': 1, 'Recidivism': 0.9706959706959707}
19. {'Index': 2027, 'k': 19, 'Utility': 0.32402627540023271, 'Group': 1, 'Recidivism': 0.9706959706959707}
20. {'Index': 11760, 'k': 20, 'Utility': 0.31763042868426916, 'Group': 0, 'Recidivism': 0.9670329670329669}
21. {'Index': 4319, 'k': 21, 'Utility': 0.30870247452464905, 'Group': 2, 'Recidivism': 0.9542124542124543}
22. {'Index': 6273, 'k': 22, 'Utility': 0.30841484641246686, 'Group': 0, 'Recidivism': 0.9670329670329669}
23. {'Index': 11409, 'k': 23, 'Utility': 0.29852167375468253, 'Group': 2, 'Recidivism': 0.9487179487179487}
24. {'Index': 10720, 'k': 24, 'Utility': 0.29985669460889691, 'Group': 1, 'Recidivism': 0.9652014652014651}
25. {'Index': 17668, 'k': 25, 'Utility': 0.295684904399727, 'Group': 0, 'Recidivism': 0.9633699633699634}
26. {'Index': 3128, 'k': 26, 'Utility': 0.29229904347113339, 'Group': 0, 'Recidivism': 0.9633699633699634}
27. {'Index': 15783, 'k': 27, 'Utility': 0.26767306424853693, 'Group': 3, 'Recidivism': 0.8919413919413918}
28. {'Index': 1769, 'k': 28, 'Utility': 0.28609602838899756, 'Group': 0, 'Recidivism': 0.9633699633699634}
29. {'Index': 142, 'k': 29, 'Utility': 0.2789364574467339, 'Group': 2, 'Recidivism': 0.9487179487179487}
30. {'Index': 18393, 'k': 30, 'Utility': 0.28000641944227156, 'Group': 0, 'Recidivism': 0.9615384615384616}
31. {'Index': 12747, 'k': 31, 'Utility': 0.27691289429517096, 'Group': 0, 'Recidivism': 0.9597069597069597}
32. {'Index': 13769, 'k': 32, 'Utility': 0.2739520624613529, 'Group': 0, 'Recidivism': 0.9578754578754579}
33. {'Index': 17449, 'k': 33, 'Utility': 0.25085789681464105, 'Group': 3, 'Recidivism': 0.8846153846153846}
34. {'Index': 15407, 'k': 34, 'Utility': 0.26941819515532406, 'Group': 1, 'Recidivism': 0.9578754578754579}
35. {'Index': 15999, 'k': 35, 'Utility': 0.26423369407233954, 'Group': 2, 'Recidivism': 0.9468864468864469}
36. {'Index': 11665, 'k': 36, 'Utility': 0.26527201145207402, 'Group': 0, 'Recidivism': 0.9578754578754579}
37. {'Index': 10108, 'k': 37, 'Utility': 0.26332722190352409, 'Group': 0, 'Recidivism': 0.9578754578754579}
38. {'Index': 5488, 'k': 38, 'Utility': 0.26146017193061327, 'Group': 0, 'Recidivism': 0.9578754578754579}

39. {'Index': 7194, 'k': 39, 'Utility': 0.25619024877622248, 'Group': 2, 'Recidivism': 0.9450549450549451}
40. {'Index': 4164, 'k': 40, 'Utility': 0.25793910571003026, 'Group': 0, 'Recidivism': 0.9578754578754579}
41. {'Index': 2712, 'k': 41, 'Utility': 0.25382605891038451, 'Group': 1, 'Recidivism': 0.9487179487179487}
42. {'Index': 6724, 'k': 42, 'Utility': 0.25369893248461156, 'Group': 0, 'Recidivism': 0.9542124542124543}
43. {'Index': 6854, 'k': 43, 'Utility': 0.23376613494377865, 'Group': 3, 'Recidivism': 0.8846153846153846}
44. {'Index': 7639, 'k': 44, 'Utility': 0.24970678464504564, 'Group': 0, 'Recidivism': 0.9505494505494505}
45. {'Index': 5924, 'k': 45, 'Utility': 0.2454030927059469, 'Group': 2, 'Recidivism': 0.9395604395604396}
46. {'Index': 962, 'k': 46, 'Utility': 0.24688649602089546, 'Group': 0, 'Recidivism': 0.9505494505494505}
47. {'Index': 8596, 'k': 47, 'Utility': 0.24459759237983328, 'Group': 1, 'Recidivism': 0.9468864468864469}
48. {'Index': 375, 'k': 48, 'Utility': 0.24424289348891998, 'Group': 0, 'Recidivism': 0.9505494505494505}
49. {'Index': 12627, 'k': 49, 'Utility': 0.23970435154082362, 'Group': 2, 'Recidivism': 0.9377289377289376}
50. {'Index': 3747, 'k': 50, 'Utility': 0.24129196900845035, 'Group': 0, 'Recidivism': 0.9487179487179487}

9.3 Synthetic Data Set

9.3.1 Color-Blind

1. {'Quality': 1.0000000000000000, 'k': 1, 'Utility': 1.4426950408889634, 'Group': 0, 'Index': 113}
2. {'Quality': 0.8973810048418225, 'k': 2, 'Utility': 0.81683139183683462, 'Group': 0, 'Index': 486}
3. {'Quality': 0.8814003593496779, 'k': 3, 'Utility': 0.6357959637357653, 'Group': 0, 'Index': 85}
4. {'Quality': 0.8804091067147484, 'k': 4, 'Utility': 0.54702893470629455, 'Group': 0, 'Index': 452}
5. {'Quality': 0.855879636319493, 'k': 5, 'Utility': 0.47767552007872588, 'Group': 0, 'Index': 79}
6. {'Quality': 0.7733088043978671, 'k': 6, 'Utility': 0.3974021127199977, 'Group': 0, 'Index': 475}
7. {'Quality': 0.7690895403299571, 'k': 7, 'Utility': 0.36985388861120055, 'Group': 0, 'Index': 447}
8. {'Quality': 0.7628029719597623, 'k': 8, 'Utility': 0.34716659363265356, 'Group': 0, 'Index': 403}
9. {'Quality': 0.7519076829072968, 'k': 9, 'Utility': 0.32654935758729897, 'Group': 0, 'Index': 416}
10. {'Quality': 0.7498244824664771, 'k': 10, 'Utility': 0.31270109707144284, 'Group': 0, 'Index': 150}
11. {'Quality': 0.7434928819892697, 'k': 11, 'Utility': 0.29920354635965929, 'Group': 0, 'Index': 169}
12. {'Quality': 0.7429459875947344, 'k': 12, 'Utility': 0.28965327733800117, 'Group': 0, 'Index': 8}
13. {'Quality': 0.7260448308481215, 'k': 13, 'Utility': 0.27511521735451266, 'Group': 0, 'Index': 204}
14. {'Quality': 0.7161545697692885, 'k': 14, 'Utility': 0.26445394899910074, 'Group': 0, 'Index': 143}
15. {'Quality': 0.7094004985486301, 'k': 15, 'Utility': 0.25586214531506674, 'Group': 0, 'Index': 355}
16. {'Quality': 0.6856339775265288, 'k': 16, 'Utility': 0.24199871109774235, 'Group': 0, 'Index': 384}
17. {'Quality': 0.6843898246380856, 'k': 17, 'Utility': 0.23678262935153965, 'Group': 0, 'Index': 465}
18. {'Quality': 0.6834292168776688, 'k': 18, 'Utility': 0.23210846674470564, 'Group': 0, 'Index': 110}
19. {'Quality': 0.6485863159179761, 'k': 19, 'Utility': 0.21650343111219511, 'Group': 0, 'Index': 180}
20. {'Quality': 0.6457383818976544, 'k': 20, 'Utility': 0.2120984144825396, 'Group': 0, 'Index': 346}
21. {'Quality': 0.6449479864802887, 'k': 21, 'Utility': 0.20865064010349449, 'Group': 0, 'Index': 318}
22. {'Quality': 0.6323040961008036, 'k': 22, 'Utility': 0.20166010604916113, 'Group': 0, 'Index': 227}
23. {'Quality': 0.6300043211901798, 'k': 23, 'Utility': 0.19823588737677375, 'Group': 0, 'Index': 74}
24. {'Quality': 0.6293777965308002, 'k': 24, 'Utility': 0.19552720601036871, 'Group': 0, 'Index': 175}
25. {'Quality': 0.6268122946274481, 'k': 25, 'Utility': 0.19238604114784377, 'Group': 0, 'Index': 456}
26. {'Quality': 0.6258223251377102, 'k': 26, 'Utility': 0.18988267641305284, 'Group': 0, 'Index': 145}
27. {'Quality': 0.6228259562521422, 'k': 27, 'Utility': 0.18691108374359494, 'Group': 0, 'Index': 114}
28. {'Quality': 0.6220009616670804, 'k': 28, 'Utility': 0.18471824071055226, 'Group': 0, 'Index': 226}
29. {'Quality': 0.6185909859282951, 'k': 29, 'Utility': 0.1818744743435006, 'Group': 0, 'Index': 254}
30. {'Quality': 0.6167700733862812, 'k': 30, 'Utility': 0.17960756306276127, 'Group': 0, 'Index': 217}

31. {'Quality': 0.6163327829077602, 'k': 31, 'Utility': 0.17783604988766391, 'Group': 0, 'Index': 96}
32. {'Quality': 0.6159461431704644, 'k': 32, 'Utility': 0.17616039214630747, 'Group': 0, 'Index': 78}
33. {'Quality': 0.6148185247585809, 'k': 33, 'Utility': 0.17434931013626376, 'Group': 0, 'Index': 382}
34. {'Quality': 0.6110147674077854, 'k': 34, 'Utility': 0.17185793256815982, 'Group': 0, 'Index': 358}
35. {'Quality': 0.6105725743171105, 'k': 35, 'Utility': 0.17038352100356527, 'Group': 0, 'Index': 105}
36. {'Quality': 0.6097238132136916, 'k': 36, 'Utility': 0.16885562839261539, 'Group': 0, 'Index': 206}
37. {'Quality': 0.6090249019360937, 'k': 37, 'Utility': 0.16742556057611122, 'Group': 0, 'Index': 149}
38. {'Quality': 0.6090067602794474, 'k': 38, 'Utility': 0.16623352330452251, 'Group': 0, 'Index': 237}
39. {'Quality': 0.5919357409956633, 'k': 39, 'Utility': 0.16046491850947334, 'Group': 0, 'Index': 243}
40. {'Quality': 0.5915617580037063, 'k': 40, 'Utility': 0.15929723387021835, 'Group': 0, 'Index': 347}
41. {'Quality': 0.5892797225150448, 'k': 41, 'Utility': 0.15765966034892309, 'Group': 0, 'Index': 155}
42. {'Quality': 0.5876643258923937, 'k': 42, 'Utility': 0.1562438338338796, 'Group': 0, 'Index': 238}
43. {'Quality': 0.5863443064203447, 'k': 43, 'Utility': 0.15494580429184956, 'Group': 0, 'Index': 402}
44. {'Quality': 0.5842322183535856, 'k': 44, 'Utility': 0.15347623276915104, 'Group': 0, 'Index': 223}
45. {'Quality': 0.5831397369184054, 'k': 45, 'Utility': 0.1523098343587756, 'Group': 0, 'Index': 468}
46. {'Quality': 0.5809877179634934, 'k': 46, 'Utility': 0.1509001155450366, 'Group': 0, 'Index': 104}
47. {'Quality': 0.5774295529529241, 'k': 47, 'Utility': 0.14916031260735355, 'Group': 1, 'Index': 197}
48. {'Quality': 0.5770305355014911, 'k': 48, 'Utility': 0.14826751784547293, 'Group': 0, 'Index': 158}
49. {'Quality': 0.5709588389132679, 'k': 49, 'Utility': 0.14594976515246239, 'Group': 0, 'Index': 333}
50. {'Quality': 0.5674764832045397, 'k': 50, 'Utility': 0.14432900545778793, 'Group': 1, 'Index': 83}

9.3.2 FA*IR

1. {'Quality': 1.0000000000000000, 'k': 1, 'Utility': 1.4426950408889634, 'Group': 0, 'Index': 113}
2. {'Quality': 0.5774295529529241, 'k': 2, 'Utility': 0.52559902971134997, 'Group': 1, 'Index': 197}
3. {'Quality': 0.5674764832045397, 'k': 3, 'Utility': 0.40934775407014928, 'Group': 1, 'Index': 83}
4. {'Quality': 0.4466567918258464, 'k': 4, 'Utility': 0.27752346851971843, 'Group': 3, 'Index': 417}
5. {'Quality': 0.8973810048418225, 'k': 5, 'Utility': 0.50083787486745734, 'Group': 0, 'Index': 486}
6. {'Quality': 0.5209135672593572, 'k': 6, 'Utility': 0.26769661873249734, 'Group': 2, 'Index': 202}
7. {'Quality': 0.8814003593496779, 'k': 7, 'Utility': 0.42386397582384355, 'Group': 0, 'Index': 85}
8. {'Quality': 0.4243094610000255, 'k': 8, 'Utility': 0.19311155781555669, 'Group': 3, 'Index': 262}
9. {'Quality': 0.4965353593502506, 'k': 9, 'Utility': 0.21564256663566203, 'Group': 2, 'Index': 424}
10. {'Quality': 0.555524331875896, 'k': 10, 'Utility': 0.23167164061656156, 'Group': 1, 'Index': 438}
11. {'Quality': 0.8804091067147484, 'k': 11, 'Utility': 0.35430268850938945, 'Group': 0, 'Index': 452}
12. {'Quality': 0.5505783084198035, 'k': 12, 'Utility': 0.21465465071197212, 'Group': 1, 'Index': 471}
13. {'Quality': 0.4205149810509688, 'k': 13, 'Utility': 0.15934287456812263, 'Group': 3, 'Index': 139}
14. {'Quality': 0.4885983152515538, 'k': 14, 'Utility': 0.18042439355544007, 'Group': 2, 'Index': 309}
15. {'Quality': 0.855879636319493, 'k': 15, 'Utility': 0.30869332672899552, 'Group': 0, 'Index': 79}
16. {'Quality': 0.5389359951382745, 'k': 16, 'Utility': 0.19022075985520315, 'Group': 1, 'Index': 234}
17. {'Quality': 0.4200539786709348, 'k': 17, 'Utility': 0.14532870296818928, 'Group': 3, 'Index': 184}
18. {'Quality': 0.7733088043978671, 'k': 18, 'Utility': 0.26263366633489815, 'Group': 0, 'Index': 475}
19. {'Quality': 0.48683672667295064, 'k': 19, 'Utility': 0.1625100917631038, 'Group': 2, 'Index': 343}
20. {'Quality': 0.5306360254198867, 'k': 20, 'Utility': 0.17429203964634796, 'Group': 1, 'Index': 484}
21. {'Quality': 0.5278042086142939, 'k': 21, 'Utility': 0.17075281772362974, 'Group': 1, 'Index': 337}
22. {'Quality': 0.7690895403299571, 'k': 22, 'Utility': 0.24528494947392232, 'Group': 0, 'Index': 447}
23. {'Quality': 0.41351900935872815, 'k': 23, 'Utility': 0.13011705636007256, 'Group': 3, 'Index': 285}
24. {'Quality': 0.48544576418414037, 'k': 24, 'Utility': 0.15081220606079682, 'Group': 2, 'Index': 6}
25. {'Quality': 0.7628029719597623, 'k': 25, 'Utility': 0.23412534375761115, 'Group': 0, 'Index': 403}

26. {'Quality': 0.5252071736885581, 'k': 26, 'Utility': 0.15935472386571339, 'Group': 1, 'Index': 46}
27. {'Quality': 0.4768454053009975, 'k': 27, 'Utility': 0.14310208267376887, 'Group': 2, 'Index': 380}
28. {'Quality': 0.4078050615717494, 'k': 28, 'Utility': 0.12110758369970347, 'Group': 3, 'Index': 154}
29. {'Quality': 0.7519076829072968, 'k': 29, 'Utility': 0.22107146352671883, 'Group': 0, 'Index': 416}
30. {'Quality': 0.5187433292528976, 'k': 30, 'Utility': 0.1510615207230139, 'Group': 1, 'Index': 57}
31. {'Quality': 0.3986598059465248, 'k': 31, 'Utility': 0.11502890500816154, 'Group': 3, 'Index': 102}
32. {'Quality': 0.47522321405679785, 'k': 32, 'Utility': 0.13591368120979644, 'Group': 2, 'Index': 411}
33. {'Quality': 0.7498244824664771, 'k': 33, 'Utility': 0.21263409604101507, 'Group': 0, 'Index': 150}
34. {'Quality': 0.5158117598166446, 'k': 34, 'Utility': 0.14508052401501295, 'Group': 1, 'Index': 93}
35. {'Quality': 0.5154046037260771, 'k': 35, 'Utility': 0.14382639315647908, 'Group': 1, 'Index': 152}
36. {'Quality': 0.7434928819892697, 'k': 36, 'Utility': 0.20590135250258854, 'Group': 0, 'Index': 169}
37. {'Quality': 0.3924782712098019, 'k': 37, 'Utility': 0.10789525085484809, 'Group': 3, 'Index': 300}
38. {'Quality': 0.45664078475059855, 'k': 38, 'Utility': 0.12464394730003102, 'Group': 2, 'Index': 144}
39. {'Quality': 0.7429459875947344, 'k': 39, 'Utility': 0.20140153584205123, 'Group': 0, 'Index': 8}
40. {'Quality': 0.5134959028682425, 'k': 40, 'Utility': 0.13827546460514928, 'Group': 1, 'Index': 298}
41. {'Quality': 0.3841464038798551, 'k': 41, 'Utility': 0.1027769822139297, 'Group': 3, 'Index': 497}
42. {'Quality': 0.45614451492496416, 'k': 42, 'Utility': 0.12127632162450135, 'Group': 2, 'Index': 439}
43. {'Quality': 0.7260448308481215, 'k': 43, 'Utility': 0.191862697455876, 'Group': 0, 'Index': 204}
44. {'Quality': 0.5079903171076616, 'k': 44, 'Utility': 0.1334476903252623, 'Group': 1, 'Index': 436}
45. {'Quality': 0.5043838608203681, 'k': 45, 'Utility': 0.13173964563066509, 'Group': 1, 'Index': 368}
46. {'Quality': 0.3835618294451192, 'k': 46, 'Utility': 0.099622629863530085, 'Group': 3, 'Index': 249}
47. {'Quality': 0.7161545697692885, 'k': 47, 'Utility': 0.18499544915166591, 'Group': 0, 'Index': 143}
48. {'Quality': 0.45520600948213424, 'k': 48, 'Utility': 0.11696480685480891, 'Group': 2, 'Index': 100}
49. {'Quality': 0.5037162479399434, 'k': 49, 'Utility': 0.12876106486107305, 'Group': 1, 'Index': 157}
50. {'Quality': 0.7094004985486301, 'k': 50, 'Utility': 0.18042521841363882, 'Group': 0, 'Index': 355}