open proceedings

Online Set Selection with Fairness and Diversity Constraints

Julia Stoyanovich*
Drexel University
Philadelphia, PA
stoyanovich@drexel.edu

Ke Yang
Drexel University
Philadelphia, PA
ky323@drexel.edu

HV Jagadish†
University of Michigan
Ann Arbor, MI
jag@umich.edu

ABSTRACT

Selection algorithms usually score individual items in isolation, and then select the top scoring items. However, often there is an additional diversity objective. Since diversity is a group property, it does not easily jibe with individual item scoring. In this paper, we study set selection queries subject to diversity and group fairness constraints. We develop algorithms for several problem settings with streaming data, where an online decision must be made on each item as it is presented. We show through experiments with real and synthetic data that fairness and diversity can be achieved, usually with modest costs in terms of quality.

Our experimental evaluation leads to several important insights in online set selection. We demonstrate that theoretical guarantees on solution quality are conservative in real datasets, and that tuning the length of the score estimation phase leads to an interesting accuracy-efficiency trade-off. Further, we show that if a difference in scores is expected between groups, then these groups must be treated separately during processing. Otherwise, a solution may be derived that meets diversity constraints, but that selects lower-scoring members of disadvantaged groups.

1 INTRODUCTION

Diversity is desired in many contexts, ranging from results of a Web search to admissions at a university. As algorithms are increasingly used to make decisions, there is growing interest in algorithms that can produce diverse results. Indeed, fairness and diversity are central to responsible data science practice [7, 17].

Diversity is a set concept: it makes no sense to talk about an individual item as being diverse. Fairness is less clearly a set concept; nevertheless, fairness is often stated with respect to some comparison standard, usually a group [5, 12, 19]. For example, in the context of racial discrimination, we frequently refer to under-represented minorities, which is a set construct, with fairness requiring proportional representation.

Most algorithmic decision-making is based on the individual: typically, a score is assigned to an individual item based on its attributes. However, since fairness and diversity are set concepts, they can only be guaranteed as part of a set selection procedure.

In this paper, we show how we can guarantee fairness and diversity in set selection. We begin by developing a simple general problem statement in Section 2, to maximize *utility* subject to a set of *diversity constraints*. We show that our problem formulation covers a wide range of fairness and diversity requirements. We then solve this problem in two settings. In Section 3, we present a baseline algorithm that make the assumption that all items are available before any selections have to be made. Then, in Section 4 we develop algorithms that decide whether to accept, reject or defer an item in an online manner, as the items are presented. We refer to this variant as the Diverse K-choice Secretary Problem.

Algorithms of Section 4 constitute the main technical contribution of this paper. These algorithms build upon a rich body of work on the Secretary Problem [8, 11, 14] — selecting the maximum element in a randomly-ordered sequence of N elements, and on its K-choice variant — selecting K elements out of N [4].

In Section 5, we show experimentally that the online algorithms of Section 4 produce solutions that both meet the diversity requirements and are very close to the baseline algorithm of Section 3 in terms of utility. Further, we demonstrate that theoretical guarantees on solution quality of online algorithms are conservative on real datasets. These algorithms start by observing the scores of the items in the stream without accepting any items, to develop a quality estimate; this is known as the warm-up period. We show that an interesting quality-efficiency trade-off can be achieved by tuning the length of the warm-up period. Finally, we show that if a difference in scores is expected between groups, then these groups must be treated separately during processing. Otherwise, a solution may be derived that meets diversity constraints, but that results in selecting lower-scoring members of historically disadvantaged groups.

We discuss related work in Section 6 and conclude in Section 7.

2 PROBLEM DEFINITION

The basic problem setting is that we have a set of items, each with associated attributes. From this set, we wish to select K items to maximize a utility score (to be defined below) subject to diversity constraints (also to be defined below). The items in the set may be presented to us together or one at a time.

We obtain the utility score for a set of K selected items as the sum of scores of each individual selected item. The *score* of an item may be pre-computed and stored as a physical attribute, or it may be computed on the fly, and possibly even be obtained as the result of an expensive scoring algorithm. In all cases, all we require is that we eventually have a single scalar score value for each item. The score is sometimes called the *utility score* or *utility value* in the literature.

The basic top-k problem is to choose K items with the highest score. That is, for any item j in the top-k, and any other item q not in the top-k, we have $s_j \geq s_q$, where s_q is the score of item q. This is equivalent to saying we choose $K \geq 0$ items such that $\forall k \in [0, K][argmin_{j \in [0,K]}(s_j)]$ is maximized. This is further equivalent to saying $\sum_{j \in [0,K]} s_j$ is maximized. We will use this last definition, since with added diversity constraints these three definitions are no longer equivalent, and the first two may not be appropriate.

Having described the utility maximization problem above, let us now turn to fairness and diversity constraints. Among the attributes associated with items, we assume that one discrete-valued attribute is of particular concern. We call this the *sensitive*

*This work was supported in part by NSF Grants No. 1741047 and 1464327.
†This work was supported in part by NSF Grants No. 1250880 and 1741022.

attribute. Our notions of fairness and diversity are defined with respect to the value of this sensitive attribute.

In practice, there may be multiple sensitive attributes, rather than just one. In this case, we could consider each independently, by making minor appropriate modifications to all statements below. If combinations of multiple attributes are of concern, or if dependencies between the sensitive attributes need to be captured explicitly, we could represent such combinations as a single (Cartesian product) attribute of concern. For example, if both race $\in \{W, B, H\}$ and gender $\in \{M, F\}$ are sensitive attributes, we could combine these into a single attribute of cardinality 6.

If a sensitive attribute is not discrete-valued, or takes on too many discrete values, then we can bucketize the attribute value into a finite number of discrete buckets. Attributes such as age and salary are often treated this way in practice for many applications. In fact, sensitive attributes may also have associated privacy concerns, and so may need to be converted to noisy histograms, e.g., to enforce differential privacy.

We further assume that the dataset is partitioned on the value of the sensitive attribute. That is, each item is associated with exactly one value of the sensitive attribute. For example, a person of mixed race should not be listed as having both White and Black as values for the race attribute: rather this value should be set to an appropriate single value, such as "White-Black-Mixed".

Let there be d distinct values of the sensitive attribute. Our requirement is to choose k_i elements for each distinct value $i \in [1...d]$, with each $k_i \in [0, K]$, and $\sum_i k_i = K$. Of course, this begs the question of what the k_i values should be. We next consider several notions of fairness and diversity and show how to capture these within this framework.

Fairness by proportional representation (of values of the sensitive attribute). Suppose that the number of items N is known, as is the number of items n_i in each sensitive category $i \in [1...d]$. Then, proportional representation requires that the desired size K of the selected set be prorated among the d categories. That is $k_i = K * n_i/N$. We call the right hand side of this equation $proportion_i$, for convenience.

A difficulty we run into is that k_i must be an integer: an item in some category is either selected or it is not. Thus, fractional values do not make sense, yet $proportion_i$ is not always an integer. We can round $proportion_i$ to the nearest integer to determine each k_i, hoping to return K items in total. But we may end up with rounding errors resulting in violation of $\sum_i k_i = K$. To avoid this, it is reasonable to provide some flexibility in choosing the value of each k_i, using the formula $\lfloor proportion_i \rfloor \leq k_i \leq \lceil proportion_i \rceil$, where $\lfloor . \rfloor$ is the floor function and $\lceil . \rceil$ is the ceiling function.

Even weaker constraints are often acceptable in practice. For example, in a class of 821 students, and with a binary assignment of the gender attribute, we may desire to see 410 students of one gender and 411 of the other. However, it is unlikely that an institution would be accused of discrimination if they admitted 407 women and 414 men. Generally, it is acceptable to set thresholds on the relative representation of different categories. This idea is a generalization of the 80% rule of disparate impact [10].

Another potentially appropriate fairness metric is the normalized difference: the mean difference normalized by the rate of positive outcomes, which in our case corresponds to being selected among the top-k. Another is the elift ratio: the ratio of positive outcomes for the historically disadvantaged demographic group over the general group. A ratio of 1 indicates no discrimination, while a ratio below 0.8 has been construed as discrimination by US courts. These and other proportional representation metrics can be found in a recent survey by Zliobaite [19].

Coverage-based diversity. A popular measure of diversity is coverage [7]: is there representation for every category in the selected set? Whether this is possible depends on how K, the number of items selected in total, compares to d, the number of categories of items. If $d \geq K$, then each $k_i \leq 1$. We cannot get full coverage, but by not choosing 2 from any category, we make sure to include a representative from as many categories as possible. If $d \leq K$, then each $k_i \geq 1$. Since K is large enough in this case, we can have multiple items from each category as long as we make sure that we have at least one from each category.

To avoid "tokenism" — selecting a single representative of each category, we may want to specify coverage diversity in terms of a larger minimum number per category. For example, we may require that there be at least 5 members of each race in the selected set. Such a choice would typically be made only if $5d \leq K$, and our requirement becomes that each $k_i \geq 5$.

Summarizing the scenarios considered above, we can state the specific diversity or proportionality constraint of interest as $floor_i \leq k_i \leq ceil_i$, where $floor_i$ and $ceil_i$ are integers that are determined, for each i, based on the particular constraint of interest. This formulation allows us to treat combinations of sensitive attributes (represented by a single Cartesian product attribute) in a way that captures attribute dependencies. For example, we can derive the constraint for the number of female candidates of a minority race to be higher or a lower than what would result from $proportion_{female} \times proportion_{minority}$.

The general statement of our problem is as follows:

Diverse Set Selection Problem Statement: Given N items, each with an associated utility score and an identified sensitive attribute, for each value i of the sensitive attribute, choose k_i items such that the summation utility of the selected set is maximized, subject to $floor_i \leq k_i \leq ceil_i$ and subject to $\Sigma_i k_i = K$. The $floor_i$ and $ceil_i$ values depend on the specific constraint to be applied. These values are computed prior to the optimization problem, and are assumed to be given.

All N items may be given together; we call this the static case and study it in Section 3. Alternatively, the items may arrive one at a time; we call this the online case and study it in Section 4.

The standard cost-metric in the top-k problem is the number of items examined: ideally, this should be much less than N. We carry over this metric to our problem domain as well. This metric, which we call *walking distance* (it is sometimes called *depth* in the top-k literature), is a simple surrogate for the incurred CPU cost, and has the advantage of being independent of the implementation and of the execution environment. We will discuss in Section 4 that walking distance relates to solution utility in the online case, and so is more informative than wall-clock time.

Another standard top-k cost metric is *buffer size*: the in-memory storage cost for running the algorithm. We do not present experimental results on buffer size, but note that all algorithms proposed here use buffers of constant size, under the assumption that K and $\sum_i ceil_i$ are constants.

Finally, as we shall see when we get to the online algorithms, we cannot always get the best answer if we are required to decide for each item on the spot. An *accuracy* metric we develop will reflect how close the online solution comes to the true optimum. We note that this optimum is the best we can do subject to the

Figure 1: An illustration of the static scenario. $N = 12$ items, labeled a through l, belong to one of two classes, blue and red. The goal is to select $K = 3$ items subject to $1 \le k_{blue} \le 2$ and $1 \le k_{red} \le 2$. Items arrive in score-sorted order, with scores ranging from 9 down to 1.

diversity or fairness constraints: a higher score may be possible without these constraints. We describe all metrics in Section 5.3.

3 THE STATIC PROBLEM

In this section, we solve the problem for the case when we have access to all items. We call this the *static* case. In the next section, we will turn to the online (streaming) case.

In the traditional set up for the top-k problem with multi-attribute criteria, the problem setting assumes that we have items sorted by attributes of interest, with our ranking criterion being some monotone aggregation function of these attribute values (e.g., weighted sum). We proceed down the sorted list(s), stopping when we can predict that an unseen item cannot possibly be included in the selected set [9].

In our problem setting, item scores are precomputed, and so we consume a single list of items, sorted by decreasing score. But we have a more complex selection criterion: Diversity constrains each k_i, the number of items with a value i for the sensitive attribute, to $floor_i \le k_i \le ceil_i$.

Recall that d is the number of distinct values of the sensitive attribute. Let us define $required = \sum_{i=1}^{d} floor_i$. For our set of floor and ceiling constraints to be feasible, we must have $required \le K$. The difference $K - required = slack$, represents the total slack that we have to choose items after all floor constraints are satisfied. To return a set of items with the highest utility (total score), we are best off filling the slack with items of highest utility, unconstrained by sensitive attribute value, as long as the number of items per category does not exceed the respective ceiling constraint. We use this observation in Algorithm 1. We illustrate the algorithm with an example.

Example 3.1. Consider the score-sorted list of items in Figure 1. $N = 12$ items are partitioned into $d = 2$ categories (blue and red), with 6 items per category. The goal is to select $K = 3$ items, with between 1 and 2 items per category. That is, $floor_{red} = floor_{blue} = 1$ and $ceil_{red} = ceil_{blue} = 2$.

We process items in order, left-to-right. At step 1, blue item a is accepted to meet the $floor_{blue}$ constraint. Among the remaining 2 items that will be accepted, one must be red, to meet the $floor_{red}$ constraint, and the other can be of either color. At step 2, blue item b is encountered and accepted. At this point, only one item remains to be accepted, and it must be red. At step 3, blue item c is skipped. Finally, at step 4, red item d is accepted, meeting the $floor_{red}$ constraint, and selecting the required $K = 3$ items. The algorithm terminates after consuming 4 items.

Let us now consider the pseudocode of Algorithm 1. As illustrated in Example 3.1, the algorithm accepts an item if the floor constraint of its category has not been met (line 7), or if the ceiling constraint of its category has not been met and some

Algorithm 1 Diverse top-k selection from a sorted list.

Require: List of items I sorted by score,
number of items to select K, number of categories d,
constraints $floor_i \le k_i \le ceil_i$ for each $i \in [1 \dots d]$.
{Initialize the output list L.}
1: $L = \emptyset$
{Initialize the counts of per-category selected items C.}
2: $C = [k_1 = 0, \dots, k_d = 0]$
{Compute the slack value s.}
3: $slack = K - \sum_{i=1}^{d} floor_i$
4: **while** $|L| < K$ **do**
5: $x = getNextItem(I)$
6: $i = category(x)$
7: **if** $k_i < floor_i$ **then**
8: $L \leftarrow x$
9: $k_i = k_i + 1$
10: **else if** $(k_i < ceil_i) \wedge (slack > 0)$ **then**
11: $L \leftarrow x$
12: $k_i = k_i + 1$
13: $slack = slack - 1$
14: **end if**
15: **end while**
16: **return** L

slack remains (line 10). Algorithm 1 terminates once K items are selected.

In general, we only have inequality constraints on each of the k_i values. However, note that in the special case that for each i, $floor_i = ceil_i$, we have the value of each k_i determined exactly. In this case, once the floor constraints are met for each category, we will have selected K items in total, since $K = \sum_i k_i = \sum_i floor_i$.

Algorithm 1 never examines any item with score lower than the smallest score included in the selected set. In this sense, the number of items examined is optimal — there is no way to examine fewer items if we proceed strictly in score order. This optimality result holds even though the worst case number of items examined is still N.

Besides the computational cost, we have one additional important notion of goodness to consider: that of the utility score. It is straightforward to establish the following Theorem.

Theorem 3.2. *Algorithm 1 produces a solution that has the highest possible utility score, subject to the given constraints.*

Even though Algorithm 1 is optimal, subject to the diversity constraints, in general it will return K items with the combined utility score that is lower than would be possible in the absence of these constraints. This cost of diversity was illustrated in Example 3.1: we skipped item c although accepting it would maximize utility, but would result in selecting all items from the same category, blue. We will quantify the cost of diversity experimentally in Section 5 (Figure 7).

4 THE ONLINE PROBLEM

In practice, even though a set has to be selected, not all items in the set may be available for evaluation at once. Rather, they may appear one at a time, with a decision to be made on the specific item instantaneously. For example, we may wish to hire a diverse set of employees. However, each hiring decision may have to be made individually on each job applicant when the job application arrives. The order of arrival of applications is not, in general, determined by the quality of the applicants. More generally, we

have to classify each *individual* item, as presented, into one of two buckets: "selected" or "not selected," subject to the utility and diversity criteria in our problem statement, for the selected *set*. Such situations motivate us to consider an online scenario, which is sometimes referred to as streaming.

Returning to the hiring example, we note that, while the quality of an applicant may be unknown ahead of the job interview, it is reasonable to assume that the number of applicants, both over-all and in each demographic category (e.g., by race, gender or some other sensitive attribute) can be known ahead of time, because these properties are declared by the applicants. The classic Secretary Problem and its variants, described next, and our proposed solution presented in the remainder of this section, rely on this information.

4.1 Background and Problem Statement

The problem of designing an online algorithm to optimize the probability of selecting the maximum element in a randomly-ordered sequence has been studied extensively [8, 11, 14], and is traditionally known as the Secretary Problem. In this problem, the goal is to hire one secretary from a pool of N candidates, where N is known, and candidates arrive in random order. When a candidate is interviewed, the decision must be made to hire or reject the candidate, and this decision is irreversible. It was shown by Lindley [14] and by Dynkin [8] that the optimal hiring strategy is to interview $m = \lfloor \frac{N}{e} \rfloor$ candidates without making any offers (this is called the *warm-up period*), and make an offer to the first candidate who is better than the best of the first m candidates (or accept the last candidate if no better candidate is seen). This strategy yields the best candidate with probability $\frac{1}{e}$, and is said to have *competitive ratio* e. Further, this is the best such strategy for the Secretary Problem, *i.e.*, with the highest competitive ratio [11].

A generalization of the Secretary Problem called the K-choice Secretary Problem is stated as follows: design an online algorithm for picking K out of N non-negative numbers presented in random order, to maximize their expected sum. While a straightforward extension of the Secretary Problem is natural here (with the same length of warm-up, $\lfloor \frac{N}{e} \rfloor$, remembering the scores of the K highest-scoring candidates), the exact optimal competitive ratio for this problem is not known for $K > 1$. This quantity is known to lie between $1 + c\sqrt{k}$ and $1 + C\sqrt{k}$ for some pair of constants $c < C$ [3].

Another interesting variant is the Poset Secretary Problem: If the elements of the permutation (candidates) are only partially ordered, how to maximize the probability of returning a maximal element in the poset? The incomparable elements present the main challenge: many simple modifications of the total order algorithm to handle incomparable elements were shown to have vanishing success probabilities [13].

In this section, we state, and then present a solution to, the online variant of the Diverse Set Selection Problem of Section 2:

Diverse K-choice Secretary Problem Statement: Design an online algorithm for picking K out of N items, each with an associated non-negative utility score and an identified sensitive attribute, presented in random order. Select items to maximize their expected sum, subject to diversity constraints of the form $floor_i \le k_i \le ceil_i$ for each value i of the sensitive attribute, and subject to $\Sigma_i k_i = K$.

Figure 2: An illustration of the online scenario. $N = 12$ items belong to one of two classes: blue and red, with $n_{blue} = n_{red} = 6$. The goal is to select $K = 3$ items subject to $1 \le k_{blue} \le 2$ and $1 \le k_{red} \le 2$.

4.2 Online Algorithm

We now present Algorithm 2 that solves the Diverse K-choice Secretary Problem. The basic idea of this algorithm is to solve d K-choice Secretary Problems [4] *in parallel*, one for each category, to satisfy the per-category $floor$ constraints. But that in itself is not enough: we also have to run a category-insensitive K-choice Secretary algorithm to select the remaining items, subject to *ceiling* constraints.

Algorithm 2 relies on the estimates of the number of items per category in the stream (n_i represents the estimate of the number of items from category i), and guarantees that diversity constraints are met if these estimates are accurate. We now illustrate Algorithm 2 with an example.

Example 4.1. Consider the stream of items in Figure 2. $N = 12$ items are partitioned into $d = 2$ categories, with 6 items per category: $n_{red} = n_{blue} = 6$. Like in Example 3.1, the goal is to select $K = 3$ items subject to $1 \le k_{blue} \le 2$ and $1 \le k_{red} \le 2$. In contrast to Example 3.1, items arrive in random order.

To start, we compute the lengths of the per-category warm-up periods: $r_{blue} = \lfloor \frac{n_{blue}}{e} \rfloor = 2$ and $r_{red} = \lfloor \frac{n_{red}}{e} \rfloor = 2$. Therefore, we will consider, and discard, 2 items in each category before accepting any items in that category. As we consider the warm-up items, we record $floor_{blue} = 1$ highest blue item score, and $floor_{red} = 1$ highest red item score in the respective per-category threshold heaps T_{red} and T_{blue}.

Similarly, we compute the length of the category-independent warm-up period $r = \lfloor \frac{N}{e} \rfloor = 4$. The number of items we will accept irrespective of their category membership corresponds to the difference between K and the sum of the floor constraints, and is 1 in our example. (We called this quantity *slack* in Algorithm 1.) Therefore, we will record the score of the highest-scoring item (of any category) among the first $r = 4$ items in the threshold heap T, setting $T = \{8\}$ (the score of item d).

The warm-up period for the blue category will terminate at step 3, after items a and c are considered, with $T_{blue} = \{6\}$. At step 4, a blue item d is encountered, with score 8, higher than $getMinElement(T_{blue}) = 6$, and this item is accepted.

The warm-up period for the red category will terminate at step 5, after items b and e are considered, with $T_{red} = \{4\}$ (score of b). We will reject the next red item, g, because its score is lower than $getMinElement(T_{red})$, and will accept the following red item i at step 9 to satisfy $floor_{red}$.

We are also looking to accept an item with a score higher than $getMinElement(T) = 8$ from any category, as long as its ceiling constraint is not exceeded. However, because i (score 9) was used to satisfy $floor_{red}$, which takes precedence, no such item is encountered. To return K items, we must accept the last item in the stream, l with score 5.

We terminate with the output {d, i, l}, with utility 8+9+5 = 22. This is only slightly lower than the best possible utility of 24.

Algorithm 2 Diverse K-choice Secretary Algorithm

Require: Stream of items I, total number of items to select K, input size N, number of categories d, constraints $floor_i \leq k_i \leq ceil_i$ and number of items per category n_i for $i \in [1 \ldots d]$.

{Initialize the output list L.}

1: $L = \emptyset$

{Initialize the array of counts of per-category selected items C.}

2: $C = [k_1 = 0, \ldots, k_d = 0]$

{Initialize counts of per-category seen items M.}

3: $M = [m_1 = 0, \ldots, m_d = 0]$

{Compute the length of per-category warm-up.}

4: $R = [r_1 = \lfloor \frac{n_1}{e} \rfloor, \ldots, r_d = \lfloor \frac{n_d}{e} \rfloor]$

{Initialize d MinHeaps, one per category, $T_1 \ldots T_d$.}

5: **for** i=1…d **do**

6: $T_i = MinHeap(floor_i)$

7: **end for**

8: $slack = K - \sum_{i=1}^{d} floor_i$

{Compute the length of category-independent warm-up.}

9: $r = \lfloor \frac{N}{e} \rfloor$

{Initialize a category-independent heap T.}

10: $T = MinHeap(slack)$

11: **while** $|L| < K$ **do**

12: $x = getNextItem(I)$

13: $i = category(x)$

14: **if** $\sum_i m_i < r$ **then**

15: $T \xleftarrow{\text{offer}} x$

16: **end if**

17: **if** $m_i < r_i$ **then**

18: $T_i \xleftarrow{\text{offer}} x$

19: **else if** $((k_i < floor_i) \wedge (score(x) > getMinElement(T_i)) \vee (n_i - m_i == floor_i - k_i)$ **then**

20: $deleteMinElement(T_i)$

21: $L \leftarrow x$

22: $k_i = k_i + 1$

23: **else if** $(\sum_i m_i \geq r) \wedge (score(x) > getMinElement(T) \wedge (k_i < ceil_i) \wedge (slack > 0))$ **then**

24: $deleteMinElement(T)$

25: $L \leftarrow x$

26: $k_i = k_i + 1$

27: $slack = slack - 1$

28: **else if** $(k_i < ceil_i) \wedge (numFeasibleItems() == K - |L|)$ **then**

29: $L \leftarrow x$

30: $k_i = k_i + 1$

31: $slack = slack - 1$

32: **end if**

33: $m_i = m_i + 1$

34: **end while**

35: **return** L

In Example 4.1, we happen to satisfy both floor constraints (at steps 4 and 9) before accepting a category-independent item at the end of the stream. Note, however, that this may not be the case in general. For example, we could have accepted a blue item with a score higher than 8, had one been encountered at steps 5, 6, 7, or 8 — any time after $floor_{blue}$ is met.

Processing of item d in Example 4.1 illustrates that different streams (per-category and category-independent) are consumed in parallel: d is part of the category independent warm-up (where it is discarded but its score is recorded in T), and it is also part of the post-warm-up stream for the blue category, where it is accepted, since its score exceeds $getMinElement(T_{blue})$.

Let us now consider the pseudocode for Algorithm 2. The algorithm uses a MinHeap data structure to keep track of the top-K elements seen thus far. We need one for each category (denoted T_i and initialized on line 6 with capacity $floor_i$), and an additional one for the extra elements after the floor constraints have been met (denoted T and initialized on line 10 with capacity $slack$). Each per-category heap T_i stores the best $floor_i$ scores seen among the first r_i items of category i. If $floor_i > r_i$, then T_i will store the first $floor_i$ elements observed, together with $floor_i - r_i$ elements of value -1. Heap T is initialized similarly, storing the best $slack$ scores seen among the first $r = \lfloor \frac{N}{e} \rfloor$ items, irrespective of category.

During the warm-up period, an item x is not accepted (not added to L) irrespective of its score, but rather is offered to the relevant per-category heap (on line 18) or to the category-independent heap (on line 15). Note that the same item x may be offered to its per-category heap and to the category-independent heap during warm-up.

An item of category i is added to the output after the warm-up period if $floor_i$ is not yet satisfied and either (a) the item has a sufficiently high score or (b) we are at the end of the stream for category i (line 19). The latter condition is evaluated by comparing the number of items remaining in the stream ($n_i - m_i$) to the number of items still required for i ($floor_i - k_i$).

An item is also added to the output if its score is sufficiently high according to the category-independent estimate and there is sufficient slack to meet all outstanding floor constraints (line 24). Note that Algorithm 2 uses the slack mechanism in a similar way as Algorithm 1.

Finally, an item is added to the output if it is *feasible*: accepting it would not violate the ceiling constraint for its category, and if exactly $K - |L|$ feasible items remain in the input. We compute the number of feasible items (line 28) as a sum of $n_j - m_j$ (the number of items that remain on the stream in category j) over all feasible categories (those in which $ceil_j - k_j > 0$).

This final set of conditions (line 28) is required to ensure that exactly K items are returned by Algorithm 2. Asserting that exactly $K - |L|$ feasible items remain in I relies on the estimates of the number of items in each category.

Optimality. Algorithm 2 specifies per-category lengths of the warm-up period on line 4. What is the competitive ratio of this algorithm? To reason about this, let us first consider the K-choice Secretary Problem, a generalization of the Secretary Problem where $K \geq 1$ rather than 1 item is to be chosen in an online manner. Recall from Section 4.1 that, when $K = 1$, the optimal competitive ratio is e, and it is achieved with a warm-up period of length $\lfloor \frac{N}{e} \rfloor$ [8, 14]. For $K > 1$, it is known that competitive ratio is no worse than e under the same warm-up period length [4], but the optimal competitive ratio is not known [3].

Our problem setting, and its solution presented in Algorithm 2, differ from the generalized K-choice Secretary Problem in that we are receiving items from multiple distinct categories. Algorithm 2 treats items that belong to different categories as different sub-streams of a common stream, and is guaranteed to have a competitive ratio no less than e for selecting $floor_i$ items in each category, by an immediate application of the result of Babaioff et al. [4]. The remaining $slack$ items are selected from the common

stream (subject to floor and ceiling constraints), and will have a competitive ratio no less than e (subject to the same constraints).

We will empirically compare the quality of the result returned by Algorithm 2 to that of the static algorithms of Section 3 in Section 5.4. We will also consider the impact of warm-up period length on accuracy in that section.

Impact of per-category warm-up on utility. An important point to note is that, by estimating scores on a per-category basis rather than for the entire set of items at once, Algorithm 2 accommodates the case when score is not independent of category membership. Consider an example in which there are two categories A and B, and where, for all pairs of items $a \in A, b \in B, score(a) < score(b)$. Suppose further that a and b occur in the input in approximately equal proportion. Then, if a common heap T of size K is maintained for both categories during warm-up (with $K < \lfloor \frac{N}{e} \rfloor$), T will contain scores of some K items from B, and so it will be the case that, at any point in time, $\forall a \in A, getMinElement(T) > score(a)$. As a result, an online algorithm will accept a subset of B with a high combined score, and it will accept $floor_A$ items from A that appear at the end of the stream. This represents the worst case for category A in terms of utility. We validate this claim experimentally in Section 5.4.

4.3 Online Algorithm with a Deferred List

In a true on-line setting, a decision must be made whether to accept or to reject an item once it is seen. In practice, it may be acceptable to keep a waiting list of modest size. For example, college admissions work this way.

We now introduce Algorithm 3, an optimized version of Algorithm 2 that will often return a set of K items of higher utility, subject to diversity constraints. This is accomplished by introducing per-category deferred lists D_i of bounded size. We now give an intuition behind this algorithm using an example.

Example 4.2. Consider again the stream of items in Figure 2. $N = 12$ items are partitioned into $d = 2$ categories, with 6 items per category: $n_{red} = n_{blue} = 6$, and arrive in random order. The goal is to select $K = 3$ items subject to $1 \le k_{blue} \le 2$ and $1 \le k_{red} \le 2$. We now highlight the differences in the processing of this input when a deferred list is allowed, as compared to that of Example 4.1 (Algorithm 2).

We conduct a warm-up period of length 2 for each category, storing $floor_{blue} = floor_{red} = 1$ highest score in each $T_{blue} = \{6\}$ and $T_{red} = \{4\}$. In contrast to Example 4.1, we do not conduct a category-independent warm-up period (there is no T).

Also in contrast to Example 4.1, items encountered during the warm-up period are not immediately discarded, but rather are placed into per-category deferred lists D_{blue} and D_{red}, which maintain up to $ceil_{blue} = 2$ and $ceil_{red} = 2$ items, respectively. Even beyond the warm-up period, we consider the scores of all encountered items, and always keep 2 best items of the appropriate category seen so far in each D_{blue} and D_{red}. Note that D_{blue} and D_{red} are MinHeaps, which we denote by sorting the elements in the order of increasing score.

In our example, $D_{blue} = \{c, a\}$ at step 3 (end of warm-up for blue), $D_{blue} = \{a, d\}$ at step 4 (d has a higher score than c and replaces c), and $D_{red} = \{e, b\}$ at step 5 (end of warm-up for red). We continue processing until a sufficient number of items post the warm-up period is seen (1 post-warm up item in each category in our example) and once there are at least K items in the union of deferred lists. We continuously update the per-category deferred lists as we process, evicting lower-scoring items and keeping 2 highest-scoring items in each category.

In our example, the algorithm terminates after step 9 (item i), with $D_{blue} = \{a, d\}$ and $D_{red} = \{b, i\}$. The union of these lists is then passed to Algorithm 1, which returns $\{a, d, i\}$, with combined utility 23. Per Example 4.1, this utility is 1 point higher than of executing Algorithm 2 on this input.

We will illustrate experimentally in Section 5.4 that Algorithm 3 usually returns a set of K items of higher utility than Algorithm 2. Note, however, that Algorithm 3 may sometimes

Algorithm 3 Diverse K-choice Secretary Algorithm with a deferred list

Require: Stream of items I, total number of items to select K, input size N, number of categories d, constraints $floor_i \le k_i \le ceil_i$ and number of items per category n_i for $i \in [1 \dots d]$.

{Initialize the output list L.}

1: $L = \emptyset$

{Initialize i deferred lists D_i of capacity $ceil_i$ for each category.}

2: **for** i=1…d **do**

3: $D_i = MinHeap(ceil_i)$

4: **end for**

{Initialize the list of counts of per-category selected items C.}

5: $C = [k_1 = 0, \dots, k_d = 0]$

{Initialize the list of counts of per-category seen items M.}

6: $M = [m_1 = 0, \dots, m_d = 0]$

{Compute the length of per-category warm-up.}

7: $R = [r_1 = \lfloor \frac{n_1}{e} \rfloor, \dots, r_d = \lfloor \frac{n_d}{e} \rfloor]$

{Initialize d MinHeaps, one per category, $T_1 \dots T_d$.}

8: **for** i=1…d **do**

9: $T_i = MinHeap(floor_i)$

10: **end for**

{Initialize the number of unsatisfied categories u.}

11: $u = d - \sum_{i=1}^{d} \mathbb{1}[floor_i == 0]$

{Initialize the total number of deferred items w (for "waiting").}

12: $w = 0$

13: **while** $(u > 0) \vee (w < K)$ **do**

14: $x = getNextItem(I)$

15: $i = category(x)$

16: **if** $m_i < r_i$ **then**

17: $T_i \xleftarrow{\text{offer}} x$

18: **else if** $(k_i < ceil_i) \wedge (score(x) > getMinElement(T_i))$ **then**

19: $k_i = k_i + 1$

20: $deleteMinElement(T_i)$

21: **if** $(floor_i > 0) \wedge (k_i == floor_i)$ **then**

22: $u = u - 1$

23: **end if**

24: **end if**

25: $D_i \xleftarrow{\text{offer}} x$

26: $m_i = m_i + 1$

27: $w = \sum_i |D_i|$

28: **end while**

29: $W = MaxHeap(w)$

30: $W \leftarrow \bigcup_i D_i$

31: Invoke Algorithm 1 on W (sorted by score), compute L.

32: **return** L

return items of lower combined utility, because it may terminate sooner than Algorithm 2.

We now describe Algorithm 3 in detail. To start, set u (unsatisfied) to the number of categories with $floor_i > 0$. Add the first $ceil_i$ items to D_i irrespective of their score, then maintain no more than $ceil_i$ items in D_i, replacing the lowest-scoring item $y \in D_i$ with item x if $score(y) < score(x)$.

A category becomes satisfied once $floor_i$ items in D_i have a sufficiently high score (post warm-up). If a sufficient number of high-scoring items cannot be found, we satisfy $floor_i$ by adding the required number of items of category i from the end of its stream.

The algorithm stops consuming its input once all unsatisfied categories become satisfied (i.e., once $u == 0$) **and** the total size of all D_i is at least K. Note that, even after a category i is satisfied, we can still add item x to D_i (while waiting for the remaining categories to be satisfied), if x happens to have a higher score than the lowest-scoring item currently in D_i.

Having filled the deferred lists, the algorithm will first add $floor_i$ items from each D_i to the output list L, and will then fill the remaining *slack* positions with the highest-scoring items from the remaining deferred lists, irrespective of their scores. Algorithm 1 can be invoked for this purpose, with $I = \bigcup_i D_i$.

Note that when Algorithm 1 is invoked on line 31, it is invoked on input W, a sorted list whose size is bounded by $\sum_i ceil_i$. We assume that $ceil_i << |I|$. In fact, setting any $ceil_i$ higher than K is not meaningful. Further, since K is commonly treated as a constant, then $\sum_i ceil_i$ can also be treated as such.

5 EXPERIMENTAL EVALUATION

In this paper, we have introduced diversity and fairness constraints into set selection queries under several different settings. Most importantly, we have introduced two streaming algorithms. For all our agorithms we are interested in evaluating the cost of introducing a diversity or fairness constraint in terms of the lower utility achieved. For streaming algorithms, we are further interested in how well we manage to satisfy a group constraint and make a group selection, while being forced to make decisions regarding individual items as they are presented.

5.1 Experimental Datasets

Our experimental evaluation is conducted on both real and synthetic datasets. The real data gives us a sense for what would happen in a real scenario, while the synthetic data let us vary parameters to dive deeper into understanding "what if" questions.

Forbes Richest. We selected two Forbes Richest People lists from 2016: US Richest with 400 individuals (https://www.forbes.com/forbes-400/list/) and World's Richest with 526 individuals (https://www.forbes.com/billionaires/list/). Both lists are naturally ranked by net worth. We used gender as the sensitive attribute in the US list, with a break-down of 27 female vs. 373 male individuals ($d = 2$ categories). We used country as the sensitive attribute in the World list, creating separate categories for US (197 individuals), Germany (44), China (43), Russia (25), and assigning the remaining 217 individuals to the category "other", resulting in $d = 5$ categories.

NASA Astronauts. This dataset is available at https://www.kaggle.com/nasa/astronaut-yearbook/data and consists of 357 astronauts, with their demographic information. We ranked this dataset by the number of space flight hours, and assigned individuals to categories based on their undergraduate major. A total of 83 majors are represented in the dataset, we assigned 9 most frequent - Physics (35), Aerospace Engineering (33), Mechanical Engineering (30) etc, to their individual categories, and combined the remaining 141 individuals into the category "other", resulting in $d = 10$ categories.

Pantheon. This dataset is a ranking of 11,341 individuals based on the popularity of their biographical page in Wikipedia, and is available at http://pantheon.media.mit.edu/rankings/people/all/all/-4000/2010/H15. Individuals in the dataset include historical and present-day figures, and are described with name, gender, birth year, place of birth, and occupation. Occupation is aggregated into a set of $d = 8$ cultural domains (http://pantheon.media.mit.edu/methods), which we use as the sensitive attribute to state diversity constraints.

Synthetic data. We also used synthetic data in our experiments, in cases where it was important to control dataset composition and assignments of scores to items in particular categories. Synthetic datasets consisted of three attributes: identifier of a tuple, value of the sensitive attribute and score. Additional details about specific datasets will be given as appropriate.

In our discussion, we will find it convenient to use the term *balanced* to describe datasets in which different categories are represented in the same proportion. For example, a dataset in which diversity is stated with respect to gender is balanced if about 50% of the individuals in the input are male and about 50% are female.

5.2 Diversity Constraints

Recall that our algorithms are designed for diversity constraints stated in terms of size limits on each category. The specific constraints chosen can implement different notions of diversity or fairness as discussed in Section 2. We explore several families of constraints, generated using the procedure described below, after a brief discussion of requirements on the constraints.

A single per-category constraint of the form $floor_i \le k_i \le ceil_i$ is satisfiable if $floor_i \le n_i$, where n_i is the number of items in category i in the input. While it is not incorrect to set $ceil_i > n_i$, this will not lead to sensible subset selection in practice, and will make satisfiability of a set of constraints more cumbersome to state. For these reasons, we also require that $ceil_i \le n_i$.

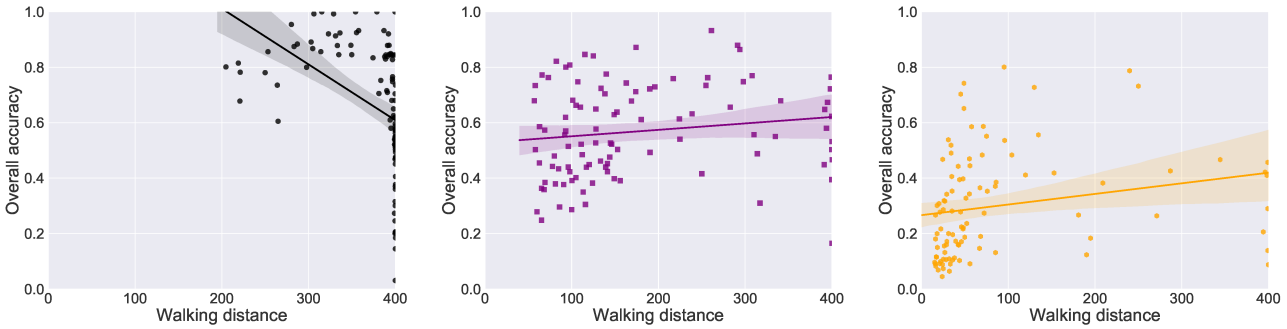
A set of per-category constraints is *satisfiable* if two conditions hold: $\sum_{i=1}^{d} floor_i \le K$ and $\sum_{i=1}^{d} ceil_i \ge K$.

We use several measures of diversity, listed below, to generate a set of per-category constraints of the form $floor_i \le k_i \le ceil_i$ for a given selected set size K and number of categories d. We generate constraints that are satisfiable individually and as a set, as discussed above. In what follows, we assume that each category i is represented in the input dataset, that is, that $n_i \ge 1$.

Minimum: (Cover as many categories as possible.)

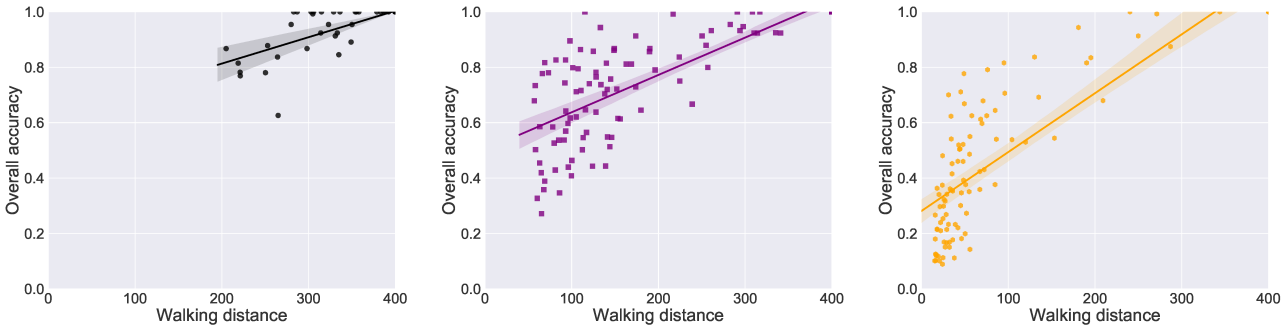
If $K \ge d$, set $floor_i = ceil_i = 1$ for all d categories. Next, compute $r = k - d$. If $r > 0$, assign the remaining r positions in the top-K to a random category j by setting $ceil_j = ceil_j + r$. Select category j from among categories in which $n_j \ge ceil_j + r$.

If $K < d$, assign $floor_i = ceil_i = 1$ to a random set of K out of d categories, and $floor_i = ceil_i = 0$ to the remaining $d - K$ categories.



(a) full warm-up (b) 1/4 warm-up (c) 1/16 warm-up

Figure 3: Accuracy of Algorithm 2 as function of walking distance, for different warm-up period lengths. Forbes US Richest, $K = 4$, $N = 400$ items, diversity on gender ($d = 2$), with average constraints $floor_i = ceil_i = K/d$.



(a) full warm-up (b) 1/4 warm-up (c) 1/16 warm-up

Figure 4: Accuracy of Algorithm 3 as function of walking distance, for different warm-up period lengths. Forbes US Richest, $K = 4$, $N = 400$ items, diversity on gender ($d = 2$), with average constraints $floor_i = ceil_i = K/d$.

Average: (Select equal numbers from each category.)

If $K \geq d$, set $floor_i = MIN(\lfloor K/d \rfloor, n_i)$ and $ceil_i = MIN(\lceil K/d \rceil, n_i)$ for all d categories. Next, compute $r = \sum_{i=1}^{d} ceil_i$. If $r < K$, assign the remaining r positions in the top-K to a random category j by setting $ceil_j = ceil_j + r$. Select category j from among categories in which $n_j \geq ceil_j + r$.

If $K < d$, set constraints as in **minimum** above.

Proportion: (Select equal proportions from each category.)

Recall that N denotes the size of the input.

If $K \geq d$, set $floor_i = \lfloor K * n_i/N \rfloor$ and $ceil_i = \lceil K * n_i/N \rceil$.

If $K < d$, set constraints as in **minimum** above.

Relaxed average: Let integer t denote the tightness threshold.

If $K \geq d$, set constraints as in **average** above. Next, set $floor_i = MAX(floor_i - t, 0)$ and $ceil_i = MIN(ceil_i + t, n_i)$.

If $K < d$, set constraints as in **minimum** above.

Relaxed proportion: Let integer t denote the tightness threshold.

If $K \geq d$, set constraints as in **proportion** above. Next, set $floor_i = MAX(floor_i - t, 0)$ and $ceil_i = MIN(ceil_i + t, n_i)$.

If $K < d$, set constraints as in **minimum** above.

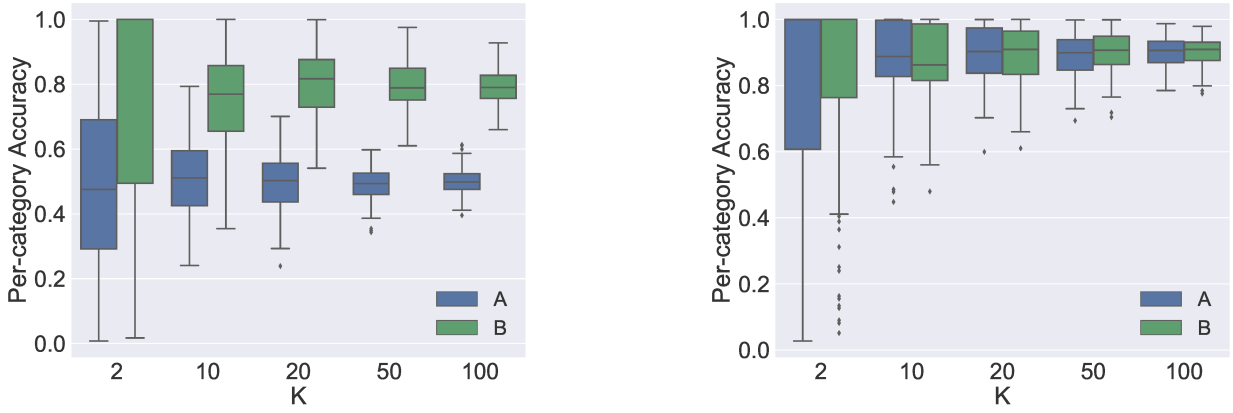
Note that in balanced datasets, average and proportion constraints are equivalent, as are relaxed average and relaxed proportion (for the same tightness threshold t).

5.3 Metrics

Recall that Algorithms 2 and 3 both consume the input one item at a time, and decide whether to accept or reject an item when it is encountered. Algorithm 3 differs from Algorithm 2 in that it can place an item on the deferred list, and decide after it has considered all feasible items which of these to accept. For a given input (a fixed set of items received in some fixed order), Algorithms 2 and 3 will stop consuming the input at some point. We refer to this point — the number of items considered from the input stream, as the *walking distance*, and use it as our primary measure of efficiency. This measure is sometimes called *depth* in the top-k literature.

In several experiments with online algorithms, we quantify the relationship between algorithm efficiency and accuracy. To quantify accuracy, we use an intuitive normalized measure that compares the scores of the K retrieved items with the best possible K scores, subject to diversity constraints. Based on our statement of optimality in Theorem 3.2, we use scores returned by Algorithm 1 as the gold standard.

To make accuracy insensitive to shifts in the score distribution, we subtract the minimum observed score from each value. For example, suppose that the lowest net worth of any individual



(a) Common warm-up.

(b) Per-category warm-up.

Figure 5: Per-category accuracy of Algorithm 2 as a function of K, on a balanced synthetic dataset of size $N = 10,000$ with average constraints. Category A items in the input have strictly lower scores than category B items. A common warm-up period leads to lower accuracy for both categories compared to per-category warm-up periods, and places items in category A at a particular disadvantage.

was 100, that $K = 2$ individuals were selected, with scores 225 and 200, and that the two highest-scoring individuals in the dataset, subject to diversity constraints, have scores 300 and 250, respectively. Then accuracy is computed as $\frac{(225-100)+(200-100)}{(300-100)+(250-100)}$.

5.4 Experimental Results: Online Algorithms

The most important question we are interested in is how well our streaming algorithms do despite being forced to meet set-oriented constraints while making decisions on individual items one at a time. The way the streaming algorithms are stated, they guarantee that the diversity constraints will be met, but do not guarantee optimality of utility score. We measure this in terms of accuracy, as described above.

The one parameter we can control in the streaming algorithms is the length of the warm-up period. Therefore, we start by presenting the relationship between warm-up period length and accuracy of the online algorithms of Section 4. To show this relationship, we will consider Figures 3 and 4, where 400 US Richest individuals (see Section 5.1 for dataset description) were randomly permuted, and where a diversity constraint was specified over the binary gender attribute, with $2 \leq k_F \leq 2$ and $2 \leq k_M \leq 2$ (a tight average constraint) and with $K = 4$. Each point in these figures corresponds to an execution of the relevant algorithm on a random permutation, with 100 executions per experiment (note that points may coincide). Other datasets in our experiments exhibited a similar trend.

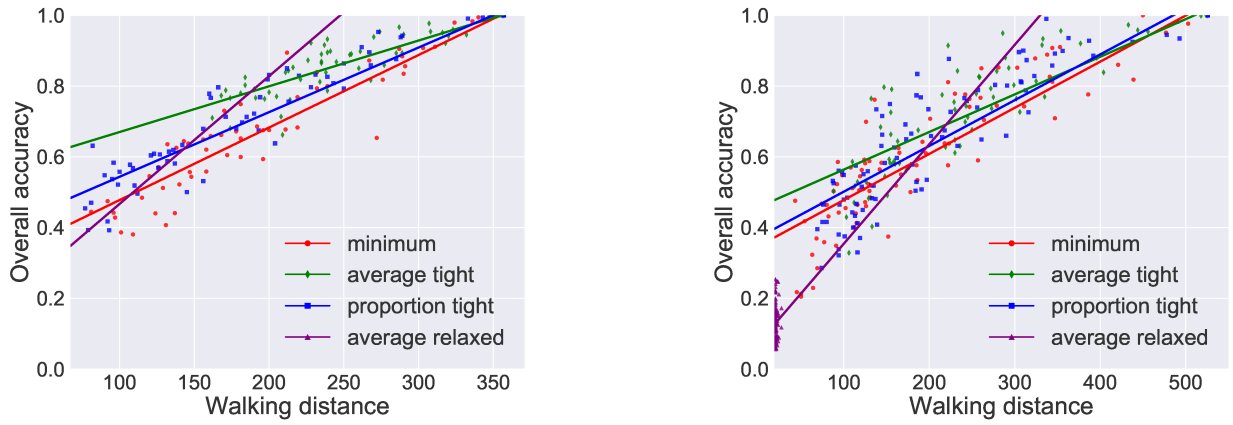
We see that Algorithm 3 gets very high accuracy, often equaling the gold standard, if given enough warm up. Algorithm 2 also does not do too badly in terms of accuracy, though Algorithm 3 does substantially better.

Walking distance takes on values between K (output size) and N (input size). Walking distance is made up of two parts: length of the warm-up period during which an on-line algorithm is estimating item scores, and post-warm-up, during which an algorithm is able to accept items. In both Algorithms 2 and 3, the per-category warm-up period has length $\lfloor \frac{n_i}{e} \rfloor$ for category i, while the total warm-up period length is the sum of per-category warm-up period lengths, and of $\lfloor \frac{N}{e} \rfloor$ for the category-independent warm-up.

Consider Figures 3a and 4a, which show the overall accuracy as a function of walking distance for Algorithms 2 and 3, respectively. Note that accuracy of Algorithm 2 varies significantly at walking distance 400 — the case when all N items were consumed from the stream. This effect is so pronounced that in Figure 3a the over-all trend in accuracy is decreasing, due exclusively to this end-of-stream effect. In contrast, Algorithm 3 is not forced to accept items from the end of the stream, and so its accuracy strictly increases as a function of walking distance.

We do not have explicit control of the post-warm-up walking distance (because we must return a valid set of results — K results that meet the diversity constraints). However, we can impact walking distance by changing the length of the per-category warm-up periods, set to $\lfloor \frac{n_i}{e} \rfloor$ by default. These settings provide a strong theoretical guarantee, but can be conservative in practice, particularly for Algorithm 3 (the deferred list variant). Figures 3b and 3c show accuracy when warm-up is abbreviated to a quarter and a sixteenth of the optimal for Algorithm 2, and Figures 4b and 4c correspond to Algorithm 3. Observe that reducing warm-up period length introduces a trade-off between walking distance (efficiency) and accuracy, and that accuracy is often comparable to that which results from the full warm-up period, but at a lower efficiency cost.

In the next experiment we demonstrate the importance of having per-category warm-up periods. Recall that Algorithm 2 considers, and rejects, $\lfloor \frac{n_i}{e} \rfloor$ items in each category before accepting any items. This warm-up period allows the algorithm to form an expectation on the score of an item. Suppose now that $K = 2$, that there are two categories A and B in the input, and that diversity constraints are such that exactly one item per category is to be selected. Further, suppose that scores of A-items are strictly lower than scores of B-items. If a common (rather than a per-category) warm-up period were used by the algorithm, with a common MinHeap of score thresholds T, then



(a) NASA Astronauts, $N = 357$, $K = 30$, $d = 10$.

(b) Forbes World Richest, $N = 526$, $K = 20$, $d = 5$.

Figure 6: Accuracy of Algorithm 3 under different diversity constraints, at 1/8 warm-up.

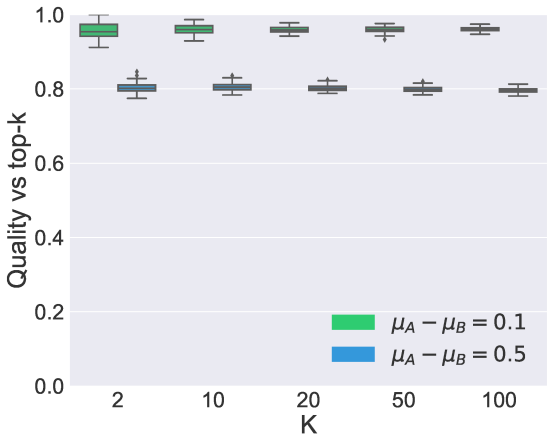


Figure 7: The cost of diversity: quality of the selected set is lower, in absolute terms, when items of a lower score must be included to satisfy diversity constraints. Synthetic datasets with $N = 10,000$, $d = 2$, under average constraints. Item scores are drawn from Gaussian distributions with the same standard deviation but different means (μ_A and μ_B).

T would contain the highes-scoring B-items that were encountered during warm-up. Since A-item scores are lower than B-item scores, no post-warm-up A-item will have a score that exceeds $getMinElement(T)$. This would force the algorithm to walk down the end of the stream in all cases (impacting performance), and to accept the very last A-item from the stream (impacting accuracy for category A).

To illustrate this point, we generate a synthetic dataset of $N = 10,000$ items in two categories, A and B, with a balanced breakdown — 5,000 A-items and 5,000 B-items, and with category-dependent scores. Scores of A-items are drawn uniformly at random from the $[0, 0.5)$ range, while scores of B-items are drawn uniformly at random from $[0.5, 1)$. We vary K between 2 and 100, and impose a (tight) average diversity constraint, setting $\lfloor \frac{K}{2} \rfloor \leq k_F \leq \lfloor \frac{K}{2} \rfloor$ and $\lfloor \frac{K}{2} \rfloor \leq k_M \leq \lfloor \frac{K}{2} \rfloor$.

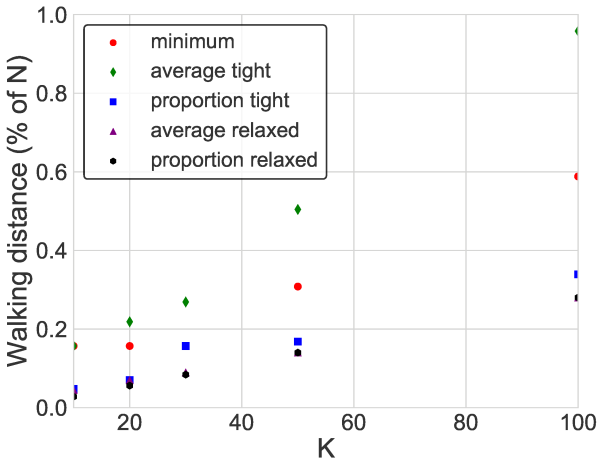
Figure 5 shows box-and-whiskers plots in which accuracy is presented as a function of K (see Section 5.3 for a description of how accuracy is computed). Observe that that accuracy for category A is lower than accuracy for category B in all cases when a common threshold is used (see Figure 5a). This is in contrast to the per-category threshold case in Figure 5b, where accuracy is comparable across the two categories.

Let us now compare performance of Algorithm 3 under different diversity constraints. Figure 6 presents accuracy as a function of walking distance, with warm-up period of length $\frac{1}{8}$ of the theoretically optimal (so, $\lfloor \frac{n_i}{8*e} \rfloor$ per category), for two real datasets: NASA Astronauts ($N = 327$, $d = 10$ and $K = 30$) and the World's Richest ($N = 526$, $d = 5$, $K = 20$). We also experimented with other real datasets, and with different values of K and warm-up period lengths, and present here results that are representative.

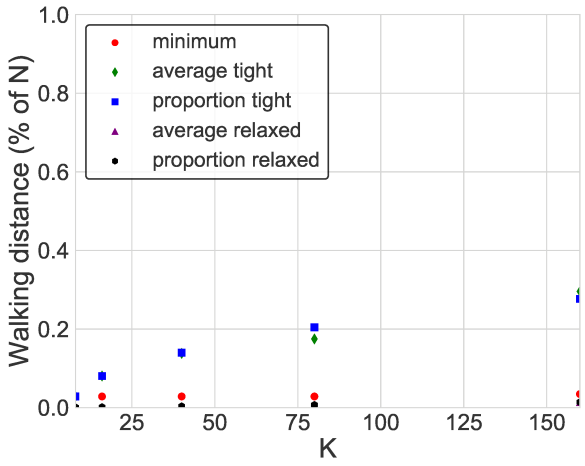
The average relaxed constraint (purple line in Figure 6) uses $t = \lfloor 0.3 * k \rfloor$ as the threshold (see Section 5.3 for a description of this and other constraints) and is easier to satisfy than the tight constraints, leading to somewhat lower walking distance. This difference was somewhat less pronounced in Figure 6a than in Figure 6b, and is sensitive to the variation in dataset composition (how balanced the categories are) and to the value of K.

In our final experiment with online algorithms, we quantified the impact of warm-up period length on the variance in accuracy. We generated a synthetic dataset with $N = 10,000$ items and with $d = 2$ categories. We requested that $K = 10$ items be returned by Algorithms 2 and 3, subject to proportion constraints.

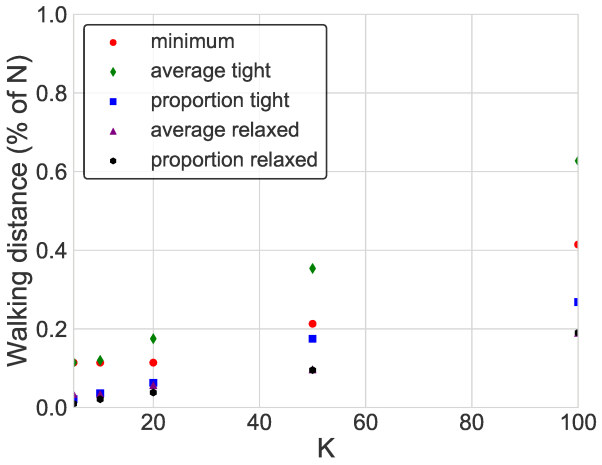
Scores of A-items were drawn uniformly at random from $[0, 0.5)$, while scores of B-items were drawn from $[0.5, 1)$. (Note that we executed per-category warm-up in both algorithms, and so differences in scores between A and B do not impact accuracy, as we saw in Figure 5.) We generated three such datasets, with A constituting 10%, 25% and 50% of the over-all dataset. For Algorithm 2, we observed, as expected, that higher variance in accuracy occurs when A appears in the dataset in lower proportion: variance was 0.080 for 10% proportion, 0.075 for 25% proportion and 0.019 for 50% proportion. Variance did not differ significantly for Algorithm 3.



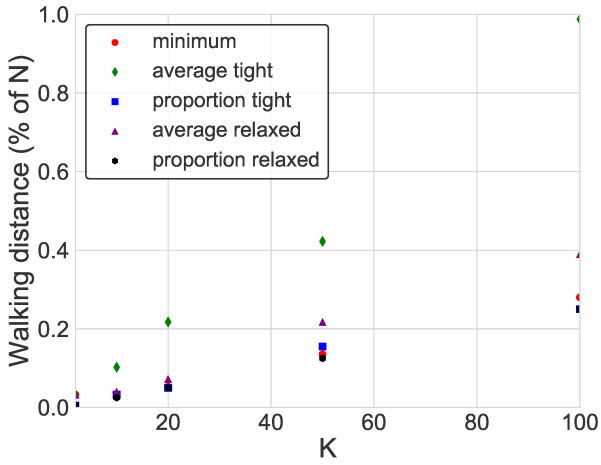
(a) NASA Astronauts, $N = 357$, $d = 10$.



(b) Pantheon, $N = 11,341$, $d = 8$.



(c) Forbes World's Richest, $N = 526$, $d = 5$.



(d) Forbes US Richest, $N = 400$, $d = 2$.

Figure 8: Walking distance of Algorithm 1 in proportion to N, as a function of K, for different diversity constraints.

5.5 Experimental Results: Static Algorithm

In this experiment we quantify the cost of imposing diversity constraints, in terms of a normalized measure we call *quality*: the sum of score of the diverse top-K divided by the sum of scores of the "vanilla" (category-agnostic) top-K. Figure 7 presents box-and-whiskers plots that quantify this cost of diversity as a function of K for two synthetic datasets, each of size N, with $d = 2$ categories represented in equal proportion, and with average diversity constraint. In the both datasets, item scores were drawn from per-category Gaussian distributions, with different means but the same standard deviations. The dataset presented in green had means of A-scores and of B-scores close to each other ($\mu_A - \mu_B = 0.1$), while in the dataset presented in blue, these distributions were further apart ($\mu_A - \mu_B = 0.5$). As expected, the cost of diversity is higher in the latter case, since items of lower scores (in absolute terms) must be included into the result to satisfy diversity constraints.

In our final experiment we use real datasets to support the claim that, while Algorithm 1 may walk to the end of the input in the worst case, this rarely happens in practice. Figure 8 presents walking distance of Algorithm 1 as a function of K for different diversity constraints, for the NASA Astronauts, Pantheon, Forbes World's Richest, and Forbes US Richest datasets. A value of 1 on the y-axis denotes that the algorithm walked to the end of the list (that is, walking distance equals N). We observe that this does not occur often, particularly for lower values of K.

6 RELATED WORK

There is considerable work on diverse top-k, starting with [2, 16], see also [18] (Sections 5.1, 5.6 and 6) for a survey. The work proposed here differs from prior work in that we consider a family of diversity constraints that can express coverage-based (rather than distance-based) diversity [7], and can also be used to compute

several fairness metrics — those based on proportional representation. To the best of our knowledge, diversity in combination with utility has not been considered in a fully online setting.

In [16] a generic method is proposed to extend top-k algorithms with a diversity criterion that is based on pair-wise similarity. The problem is formulated as: given a user-defined pairwise similarity function $sim(s_i, s_j)$, and a user-defined similarity threshold τ, return the highest-scoring set of k items such that $sim(s_i, s_j) \leq \tau$. When describing top-k methods, they refer to incremental methods (generate results in decreasing order of scores, stop once k results were generated) vs. bounding methods (generate results in some order, stop once the top-k are among the results, Fagin's TA is in this category).

More recently, diversity-aware top-k for pub/sub queries over text streams was considered in [6]. There, diversity is a pair-wise measure based on document similarity in the top-k (max-sum), quality is measured as the relevance of a document to a user's query, and there is additionally a per-document recency score that is appropriate for a stream of Twitter messages or Facebook status updates, and is based on an exponential decay function.

Another related line of work is [1], where max-sum diversity is maximized subject to a constraint on a variant of coverage-based diversity. The problem is posed as a partition matroid, a local search algorithm is proposed, and it is shown that it achieves a 0.5 approximation of the optimal solution.

Selection of diverse set results in an online setting is studied in [15]. The authors consider selection of subsets of items that are simultaneously diverse along multiple dimensions. For example, for program committee selection it is desirable to achieve coverage of topics, geographic diversity and gender diversity. The paper proposes and analyzes several diversity objectives and proposes heuristic and dynamic programming methods. The most important difference between our method and that of [15] is that they do not explicitly handle utility.

In [16] a set of diverse top-k items is determined that maximizes the total score of the K selected items, subject to a pair-wise diversity constraint. The problem is modeled by representing the items as vertices in a graph, and by including an edge between vertices s_i and s_j if their similarity is above a user-specified threshold τ. A diverse set K is the independent set of a graph — a set of vertices in which no two vertices are adjacent. This formulation can accommodate the coverage diversity version of our problem: Include an edge between two vertices if they belong to the same category, then identify an independent subset of size d that maximizes the total score. If $d < k$, we add $k - d$ vertices that maximize the total score. The algorithmic contributions of [16] are in (1) determining when the set of $k' > k$ items is sufficient to compute the true diverse top-k, and (2) efficiently identifying the independent set of a graph for a fixed k (finding an independent set of a graph is NP-hard).

7 CONCLUSIONS

Diversity and group fairness are important objectives in algorithmic decision-making. Since most algorithms are designed to score or classify items individually, it is not easy to support these objectives. In this paper, we showed how we can continue to select items individually and meet desired diversity and group fairness constraints, while paying a very small utility cost.

We demonstrated experimentally that the theoretically-motivated setting for warm-up period length can be conservative in practice. In our future work, we will investigate the interaction between expected (or observed) score variance and warm-up period length.

Further, we demonstrated that different categories must be treated separately in score estimation, to achieve comparable accuracy irrespective of expected score, and ultimately afford comparable opportunity to members of different groups.

REFERENCES

[1] Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. 2013. Diversity maximization under matroid constraints. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*. 32–40. DOI: http://dx.doi.org/10.1145/2487575.2487636

[2] Albert Angel and Nick Koudas. 2011. Efficient diversity-aware search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. 781–792. DOI: http://dx.doi.org/10.1145/1989323.1989405

[3] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. 2008. Online auctions and generalized secretary problems. *SIGecom Exchanges* 7, 2 (2008). DOI: http://dx.doi.org/10.1145/1399589.1399596

[4] Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. 2007. Matroids, secretary problems, and online mechanisms. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*. 434–443. http://dl.acm.org/citation.cfm?id=1283383.1283429

[5] Solon Barocas and Andrew D. Selbst. 2016. Big data's disparate impact. *California Law Review* 104 (2016).

[6] Lisi Chen and Gao Cong. 2015. Diversity-Aware Top-k Publish/Subscribe for Text Stream. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. 347–362. DOI: http://dx.doi.org/10.1145/2723372.2749451

[7] Marina Drosou, HV Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. 2017. Diversity in Big Data: A Review. *Big Data* 5, 2 (2017).

[8] E.B. Dynkin. 1963. The optimum choice of the instant for stopping a Markov process. *Sov. Math. Dokl.* 4 (1963).

[9] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*. DOI: http://dx.doi.org/10.1145/375551.375567

[10] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*. 259–268. DOI: http://dx.doi.org/10.1145/2783258.2783311

[11] Thomas S. Ferguson. 1989. Who Solved the Secretary Problem? *Statist. Sci.* 4, 3 (08 1989), 282–289. DOI: http://dx.doi.org/10.1214/ss/1177012493

[12] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. 2016. On the (im)possibility of fairness. *CoRR* abs/1609.07236 (2016). http://arxiv.org/abs/1609.07236

[13] Ravi Kumar, Silvio Lattanzi, Sergei Vassilvitskii, and Andrea Vattani. 2011. Hiring a secretary from a poset. In *Proceedings 12th ACM Conference on Electronic Commerce (EC-2011), San Jose, CA, USA, June 5-9, 2011*. 39–48. DOI: http://dx.doi.org/10.1145/1993574.1993582

[14] D. V. Lindley. 1961. Dynamic Programming and Decision Theory. *Journal of the Royal Statistical Society* 10, 1 (March 1961), 39–51.

[15] Debmalya Panigrahi, Atish Das Sarma, Gagan Aggarwal, and Andrew Tomkins. 2012. Online selection of diverse results. In *Proceedings of the Fifth International Conference on Web Search and Web Data Mining, WSDM 2012, Seattle, WA, USA, February 8-12, 2012*. 263–272. DOI: http://dx.doi.org/10.1145/2124295.2124329

[16] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2012. Diversifying Top-K Results. *PVLDB* 5, 11 (2012), 1124–1135. http://vldb.org/pvldb/vol5/p1124_luqin_vldb2012.pdf

[17] Julia Stoyanovich, Bill Howe, Serge Abiteboul, Gerome Miklau, Arnaud Sahuguet, and Gerhard Weikum. 2017. Fides: Towards a Platform for Responsible Data Science. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*. 26:1–26:6. DOI: http://dx.doi.org/10.1145/3085504.3085530

[18] Kaiping Zheng, Hongzhi Wang, Zhixin Qi, Jianzhong Li, and Hong Gao. 2017. A survey of query result diversification. *Knowl. Inf. Syst.* 51, 1 (2017), 1–36. DOI: http://dx.doi.org/10.1007/s10115-016-0990-4

[19] Indre Zliobaite. 2017. Measuring discrimination in algorithmic decision making. *Data Min. Knowl. Discov.* 31, 4 (2017), 1060–1089. DOI: http://dx.doi.org/10.1007/s10618-017-0506-1