Towards Responsible Data-driven Decision Making in Score-Based Systems *

Abolfazl Asudeh[†], H. V. Jagadish[‡], Julia Stoyanovich[§]
[†]University of Illinois at Chicago, [‡]University of Michigan, [§]New York University
[†]asudeh@uic.edu, [‡]jag@umich.edu, [§]stoyanovich@nyu.edu

Abstract

Human decision makers often receive assistance from data-driven algorithmic systems that provide a score *for evaluating the quality of items such as products, services, or individuals. These scores can be obtained by combining different features either through a process learned by ML models, or using a weight vector designed by human experts, with their past experience and notions of what constitutes item quality. The scores can be used for different evaluation purposes such as ranking or classification.*

In this paper, we view the design of these scores through the lens of responsibility. We present technical methods (i) to assist human experts in designing fair and stable score-based rankings and (ii) to assess and (if needed) enhance the coverage of a training dataset for machine learning tasks such as classification.

1 Introduction

Big data technologies have affected every corner of human life and society. These technologies have made our lives unimaginably more shared, connected, convenient, and cost-effective. Using data-driven technologies gives us the ability to make wiser decisions, and can help make society safer, more equitable and just, and more prosperous. However, while having an enormous potential to help solve societal issues, *irresponsible* implementation of these technologies can not only fail to help, but may even make matters worse. Racial bias in predictive policing and data-driven judgeship, harming marginalized people and poor communities, and sexism in job recommendation systems are a few examples of such failures. In order to minimize societal harms of data-driven technologies, and to ensure that objectives such as fairness, equity, diversity, robustness, accountability, and transparency are satisfied, it is necessary to develop proper *tools, strategies, and metrics*.

Human decision makers often receive assistance from data-driven algorithmic systems that *provide a score for evaluating objects, including individuals*. The scores can be computed by combining different attributes either through a process learned by ML models (using some training data), or using a weight vector assigned by human experts. For example, a support vector machine learns the parameter values that define a linear separator in some regularized multi-dimensional feature space. Learning methods require that there be labeled data, and assume that there is some known ground truth.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work was supported in part by NSF Grants No. 1741022 and 1926250.

In contrast, an expert-specified method does not require any labeled data, but may be ad-hoc. The scores are used either to evaluate an object independently (by only looking at its score) or in comparison with others.

Two major categories of methods that use scores to supporting decisions are (i) *Classification* and (ii) *Ranking*[1]. Classification is often used for predicting future outcomes based on historical data. Predicting recidivism and classifying individuals based on how likely they will commit a crime in the future is a societally important example of this kind. Ranking, on the other hand, is used for assignment by comparison on the existing data. For example, a company may rank its employees, and then reward high-ranked employees (with a raise or promotion) and fire low-ranked employees. College admissions is another example where the top-k applicants may be admitted by a college. Similarly, the international football association FIFA considers its rankings as "a reliable measure" for seeding the international tournaments such as the World Cup [3].

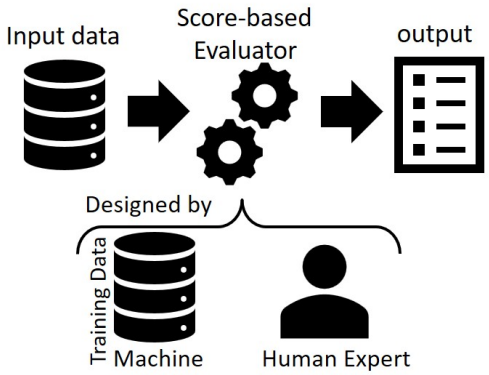

Figure 1: General architecture of score-based systems

Rankings are relative while labels in classification are absolute. That is, the rank of an individual depends on the others in the dataset, while class labels are assigned solely based on the score of an individual. The scoring mechanism for classification is usually learned by ML models. It can be a linear model such as regression and SVM, or a complex deep learning model. On the other hand, scoring mechanism for ranking is usually designed by human experts. US News university rankings, FIFA rankings, and CSRankings[2] are some of these examples.

Of course, the dichotomy above is not as clean as the preceding paragraph may suggest. Not all classification scoring is machine-learned. For example PSA (Public Safety Assessment) scores[3], used in data-driven judgeship, are human-designed. Similarly, ranking can be the basis for classification through the introduction of a cut-off rank, as in the case of college admissions.

Figure 1 shows the general architecture of score-based systems for data-driven decision making. The central component in these systems are the score-based evaluators that assign a score to each individual in the input data and generate the output by, for example, ranking or classifying the input. The output provides the evaluation of individuals that is used for decision making. Note that the scoring module can be designed by experts, or be learned by a machine, using some training data.

We, in our project Mithra, view *human experts* (for human-designed evaluators) and *training data* (for machine learned evaluators) as the keys to achieving responsibility in score-based systems. That is, for human-designed tasks such as ranking, we advocate designing assistive tools that help experts make sure their evaluators meet the objectives of fairness and stability. On the other hand, for machine learning tasks such as classification, we advocate assessing and repairing training data to make sure that, for example, the data is representative of minority groups [4], and models trained on that data do not reflect results of historical discrimination [5]. In the following, first in § 2, we explain our research for score-based ranking. Then in § 3, we provide our results for machine learning tasks such as classification by assessing and enhancing coverage for a (given) training dataset.

[1]Note that in some contexts ranking or classification is done without scoring. For instance, rank aggregation from partial ranked lists [1] or pairwise comparisons [2] is popular for group opinion collection. Our focus in this paper are evaluations (including ranking and classification) based on scores.

[2]csrankings.org

[3]www.psapretrial.org/about/factors

$\mathcal{D}$				f	f'
id	x_1	x_2	location	$\langle 1,1\rangle$	$\langle 1.11,.9\rangle$
t_1	0.63	0.71	Detroit	1.34	1.338
t_2	0.72	0.65	Chicago	1.37	1.384
t_3	0.58	0.78	Detroit	1.36	1.387
t_4	0.7	0.68	Chicago	1.38	1.389
t_5	0.53	0.82	Detroit	1.35	1.321
t_6	0.61	0.79	Chicago	1.4	1.388

Figure 2: Example 1-Data

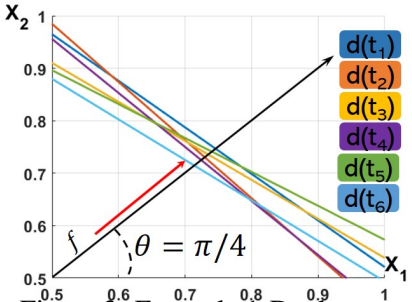

Figure 3: Example 1-Dual space

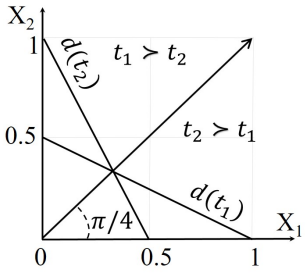

Figure 4: Ordering exchange between t_1:[1,2] and t_2:[2,1]

2 Responsible Ranking

Ranking of individuals is commonplace today, and is used, for example, for college admissions and employment. Score-based evaluators, designed by human experts, are commonly used for ranking, especially when there are multiple criteria to consider. The scores are usually computed by linearly combining (with non-negative weights) the relevant attributes of each individual from some dataset $\mathcal{D}$. Then, we sort the individuals in decreasing order of score and finally return either the full ranked list or its highest-scoring sub-set, the top-k.

Formally, we consider a dataset $\mathcal{D}$ to consist of n items, each with d scalar scoring attributes. In addition to the scoring attributes, the dataset may contain non-scoring attributes that are used for filtering, but they are not our concern here. Thus we represent an item $t \in \mathcal{D}$ as a d-length vector of scoring attributes, $\langle x_1, x_2, \ldots, x_d\rangle$. Without loss of generality, we assume that the scoring attributes have been appropriately transformed: normalized to non-negative values between 0 and 1, standardized to have equivalent variance, and adjusted so that larger values are preferred. A *scoring function* $f_{\vec{w}} : \mathbb{R}^d \to \mathbb{R}$, with weight vector $\vec{w} = \langle w_1, w_2, \ldots, w_d\rangle$, assigns the score $f_{\vec{w}}(t) = \Sigma_{j=1}^{d} w_j t[j]$ to any item $t \in \mathcal{D}$.

Linear scoring functions are straightforward to compute and easy to understand [6]. That is the reason they are popular for ranking, and for evaluation in general. However, it turns out that the rankings may highly depend on the design of these functions. To further explain this, let us consider the following toy example.

Example 1: *Consider a real estate agency with two offices in Chicago, IL and Detroit, MI. The owner assigns agents based on need (randomly) to the offices. By the end of the year, she wants to give a promotion to the "best" three agents. The criteria for choosing the agents are* x_1 : `sales` *and* x_2 : `customer satisfaction`. *Figure 2 shows the values in* $\mathcal{D}$*, after normalization. Considering the two criteria to be (roughly) equally important, the owner chooses the weights* $\vec{w} = \langle 1, 1\rangle$ *for scoring. That is, the score of every agent is computed as* $f = x_1 + x_2$. *The 5th column in Figure 2 shows the scores, based on this function. According to function* f*, the top-3 agents are* t_6*,* t_4*, and* t_2*, with scores 1.4, 1.38, and 1.37, respectively. Note that, according to* f*, all top-3 agents are located in Chicago and no agent from Detroit is selected.*

The specific weights chosen have a huge impact on the score and hence rank for an item. In Example 1, the owner chose the weight vector $\vec{w} = \langle 1, 1\rangle$, *in an ad-hoc manner*, without paying attention to the consequences. However, small changes in the weights could dramatically change the ranking. For example, the function f' with the weight vector $\vec{w'} = \langle 1.1, .9\rangle$ may be equally good for the owner and she may not even have a preference between $\vec{w}$ and $\vec{w}'$. Probably her choice of weights is only because $\vec{w}$ is more intuitive to human beings. The last column in Figure 2 shows the scores based on f', which produce the ranking $f' : \langle t_4, t_6, t_3, t_2, t_1, t_5\rangle$. Comparing it with the ranking generated by $f : \langle t_6, t_4, t_2, t_3, t_5, t_1\rangle$, one may notice that the rank of each and every individual has changed. More importantly, while according to f all promotions are given to the agents of the Chicago office, f' gives two promotions to Chicago and one to Detroit.

Many sports use ranking schemes. An example is the FIFA World Ranking of national soccer teams based on recent performance. FIFA uses these rankings as "a reliable measure for comparing national A-teams" [3].

Despite the trust placed by FIFA in these rankings, many critics have questioned their validity. University rankings is another example that is both prominent and often contested [7]: various entities, such as U.S. News and World Report, Times Higher Education, and QS, produce such rankings. Similarly, many funding agencies compute a score for a research proposal as a weighted sum of scores of its attributes. These rankings are, once again, impactful, yet heavily criticized. Similarly, in criminal justice, COMPAS [8] was originally intended to provide services and positive interventions, under resource constraints. That is, a score computed by COMPAS would then be used to rank individuals to prioritize access to services. Many other impactful examples can be mentioned, such as a company that evaluates its employees to promote some and let go some others, and a college admissions officer who decides to admit a small portion of the applicants.

Surprisingly, similar to Example 1, despite the enormous impact of score-based rankers, attribute weights are usually assigned in an ad-hoc manner, based only on intuitive reasoning and common-sense of the human designers. For instance, in the case of FIFA rankings, the scoring formula combines the past four years of performance of each team as $x_1 + 0.5x_2 + 0.3x_3 + 0.2x_4$, where x_i is the team's performance in the past i^{th} year. Of course, the designers tried to come up with a set of weights that make sense. For them $0.98x_1 + 0.51x_2 + 0.29x_3 + 0.192x_4$ would probably be equally acceptable, since the weight values are virtually identical: they choose the former formula simply because round numbers are more intuitive. This issue, in the context of university ranking, is further elaborated by Malcolm Gladwell in [7].

Assuming that the designers of rankings are willing to accept scoring functions similar to their initial functions, Mithra provides a toolbox and algorithms to help human experts practice responsible ranking. In the following, we start by providing some necessary background from computational geometry in § 2.2, followed by an explanation of fairness and stability in our framework in § 2.1.

2.1 Fairness and Stability Models

Decisions based on rankings may impact the lives of individuals and even influence societal policies. For this reason, it is essential to make the development and deployment of rankings transparent and otherwise principled. Also, since rankings highly depend on what weights are chosen in the scoring function, it is necessary to make sure that generated rankings are fair and robust.

2.1.1 Fairness

There is not a single universal definition of fairness. Impossibility theorems [9] have established that we cannot simultaneously achieve all types of fairness. Indeed, the appropriate definitions of fairness greatly depend on the context and on the perspective of the user. Sometimes, it may even be prescribed by law. As such, we consider a general definition of fairness in our work. Our focus is on societal (v.s. statistical) bias [10] and group fairness [11] (v.s. individual fairness [12]).

We consider some attributes, used for decision making (e.g. sales and customer satisfaction in Example 1), to be *non-sensitive*. Some other attributes, such as race and gender (and location in Example 1), we consider to be *sensitive*. We adopt the Boolean fairness model, in which a fairness oracle $\mathcal{O}$ takes as input an ordered list of items from $\mathcal{D}$, and determines whether the list satisfies a set of *fairness constraints*, defined over the sensitive attributes: $\mathcal{O} : \nabla_f(\mathcal{D}) \rightarrow \{\top, \bot\}$. A scoring function f that gives rise to a fair ordering over $\mathcal{D}$ is said to be *satisfactory*. For instance, in Example 1, assume that the owner knows that, because of some hidden factors, sales and customer satisfaction patterns are different in Chicago and Detroit. Hence, she considers the selection of the top-3 agents to be fair, if it assigns at least one of the promotions to each one of the offices. Note that according to this criterion, the ranking provided by $f = x_1 + x_2$ is not fair as it assigns all three promotions to the agents in Chicago. On the other hand the ranking generated by function $f' = 1.1x_1 + .9x_2$ assigns two of the promotions to Chicago and one to Detroit, and hence is considered to be fair.

2.1.2 Stability

We want a ranking to be *stable* with respect to changes in the weights used for scoring. Given a particular ranked list of items, one question a consumer will ask is: how robust is the ranking? If small changes in weights can change the ranked order, then there cannot be much confidence in the correctness of the ranking. We call a ranking of items *stable* if it is generated by a large portion of scoring functions in the neighborhood of the initial scoring function specified by the expert.

Every scoring function in a universe $\mathcal{U}^*$ of scoring functions induces a single ranking of the items. But each ranking is generated by many functions. For a dataset $\mathcal{D}$, let $\mathfrak{R}_\mathcal{D}$ be the set of rankings over the items in $\mathcal{D}$ that are generated by at least one scoring function $f \in \mathcal{U}^*$. Consider the set of scoring functions that generate a ranking $\mathfrak{r} \in \mathfrak{R}_\mathcal{D}$. Because this set of functions is continuous, we can think of it as a region in the space of all possible functions in $\mathcal{U}^*$. We use the region associated with a ranking to define the ranking's stability. The intuition is that a ranking is stable if it can be induced by a large set of functions. If the region of a ranking is large, then small changes in the weight vector are not likely to cross the boundary of a region and therefore the ranked order will not change. For every region R, let its volume, $vol(R)$, be the measure of its bulk. Given a ranking $\mathfrak{r} \in \mathfrak{R}_\mathcal{D}$, the stability of $\mathfrak{r}$ is the proportion of scoring functions in $\mathcal{U}^*$ that generate $\mathfrak{r}$. That is, stability is the ratio of the volume of the ranking region of $\mathfrak{r}$ to the volume of $\mathcal{U}^*$. Formally:

$$S_\mathcal{D}(\mathfrak{r}) = \frac{\text{vol}(R_\mathcal{D}(\mathfrak{r}))}{\text{vol}(\mathcal{U}^*)} \tag{20}$$

We emphasize that stability is a property of a ranking (not of a scoring function).

2.2 Geometric Interpretation

In the popular geometric model for studying data, each attribute is modeled as a dimension and items are interpreted as points in a multi-dimensional space. We transform this *Primal space* into a *dual space* [13].

We use the dual space in $\mathbb{R}^d$, where an item t is presented by a hyperplane $\mathsf{d}(t)$ given by the following equation of d variables $x_1 \dots x_d$:

$$\mathsf{d}(t):\ t[1] \times x_1 + \cdots + t[d] \times x_d = 1 \tag{21}$$

Continuing with Example 1, Figure 3 shows the items in the dual space. In $\mathbb{R}^2$, every item t is a 2-dimensional hyperplane (i.e. simply a line) given by $\mathsf{d}(t) : t[1]x_1 + t[2]x_2 = 1$. In the dual space, a scoring function $f_{\vec{w}}$ is represented as a ray starting from the origin and passing through the point $[w_1, w_2, ..., w_d]$. For example, the function f with the weight vector $\vec{w} = \langle 1, 1\rangle$ in Example 1 is drawn in Figure 3 as the origin-starting ray that passes through the point $[1, 1]$. Note that every scoring function (origin-starting ray) can be identified by $(d-1)$ angles $\langle \theta_1, \theta_2, \cdots, \theta_{d-1}\rangle$, each in the range $[0, \pi/2]$. Thus, given a function $f_{\vec{w}}$, its angle vector can be computed using the polar coordinates of w. For example, the function f in Figure 3 is identified by the angle $\theta = \pi/4$. There is a one-to-one mapping between these rays and the points on the surface of the origin-centered unit d-sphere (the unit hypersphere in $\mathbb{R}^d$), or to the surface of any origin-centered d-sphere. Thus, (the first quadrant of) the unit d-sphere represents the universe of functions $\mathcal{U}$.

Consider the intersection of a dual hyperplane $\mathsf{d}(t)$ with the ray of a function f. This intersection is in the form of $a \times \vec{w}$, because every point on the ray of f is a linear scaling of $\vec{w}$. Since this point is also on the hyperplane $\mathsf{d}(t)$, $t[1] \times a \times w_1 + \cdots + t[d] \times a \times w_d = 1$. Hence, $\sum t[j]w_j = 1/a$. This means that the dual hyperplane of any item with the score $f(t) = 1/a$ intersects the ray of f at point $a \times \vec{w}$. Following this, the ordering of the items based on a function f is determined by the ordering of the intersection of the hyperplanes with the vector of f. The closer an intersection is to the origin, the higher its rank. For example, in Figure 3, the intersection of the line t_6 with the ray of $f = x_1 + x_2$ is closest to the origin, and t_6 has the highest rank for f.

Consider the dual presentation of two items $t_1 : [1, 2]$ and $t_2 : [2, 1]$, shown in Figure 4, and a function that passes through this intersection. We name this function the *ordering exchange* between t_1 and t_2. That is because

this function partitions the space in two regions, where every function in the top-left region ranks t_1 higher than t_2 while every function in bottom-right ranks t_2 higher. In general, the ordering exchange between a pair of items t_i and t_j is identified by (the set of function on) the following origin-passing hyperplane:

$$\sum_{k=1}^{d}(t_i[k] - t_j[k])w_k = 0 \tag{22}$$

2.3 Designing Fair Ranking Schemes

We interpret fairness to mean that (a) disparate impact, which may arise as a result of historical discrimination, needs to be mitigated; and yet (b) disparate treatment cannot be exercised to mitigate disparate impact when the decision system is deployed. Disparate impact arises when a decision making system provides outputs that benefit (or hurt) a group of people sharing a value of a sensitive attribute more frequently than other groups of people. *Disparate treatment*, on the other hand, arises when a decision system provides different outputs for groups of people with the same (or similar) values of non-sensitive attributes but with different values of sensitive attributes. To avoid disparate treatment, it is desirable (and in many cases mandated by law) to not use information about an individual's membership in a protected group as part of decision-making. Following these, our goal is to build a system that helps human expert to design fair ranking schemes, in the sense that those *both* mitigate disparate impact (by ensuring that appropriate proportionality constraints are satisfied) *and* do not exercise disparate treatment (by not explicitly using information about an individual's membership in a protected group) during deployment. That is, a *single* evaluator will be used for all items in the dataset, irrespective of their membership in a protected group.

Our goal [4] is to build a system to assist a human designer of a scoring function in tuning attribute weights to achieve fairness. Formally, our *closest satisfactory function* problem is: Given a dataset $\mathcal{D}$ with n items over d scalar scoring attributes, a fairness oracle $\mathcal{O} : \nabla_f(\mathcal{D}) \rightarrow \{\top, \bot\}$, and a linear scoring function f with the weight vector $\vec{w} = \langle w_1, w_2, \cdots, w_d\rangle$, find the function f' with the weight vector $\vec{w'}$ such that $\mathcal{O}(\nabla_{f'}(\mathcal{D})) = \top$ and the angular distance between $\vec{w}$ and $\vec{w'}$ is minimized.

Since the tuning process does not occur too often, it may be acceptable for it to take some time. However, we know that humans are able to produce superior results when they get quick feedback in a design or analysis loop. Ideally, a designer of a ranking scheme would want the system to support her work through interactive response times. Our goal is to meet this need, to the extent possible, by providing a query answering system. From the system's viewpoint, the challenge is to propose similar weight vectors that satisfy the fairness constraints, in interactive time. To accomplish this, our solution operates with an offline phase and then an online phase. In the offline phase, we process the dataset, and develop data structures that will be useful in the online phase. In the online phase, the user specifies a *query* in the form of a scoring function f. If the ranking based on f does not meet the predefined fairness constraints, we suggest to the user an alternative scoring function that is both satisfactory and similar to f. The user may accept the suggested function, or she may decide to manually adjust the query and invoke our system once again.

The notion of ordering exchanges, explained in § 2.2 is a key in the preprocessing phase. Consider the set of ordering exchange hyperplanes between all pairs of items in $\mathcal{D}$. Similar to Figure 4, each hyperplane $h_{i,j}$ (the ordering exchange between t_i and t_j) partitions the function space $\mathcal{U}$ in two regions where in one region t_i outranks t_j while in the other t_j is ranked higher. The collection of these hyperplanes provide an arrangement [14] in the form of a dissection of the space into origin-starting connected d-cones with convex surfaces, we call *ranking regions*. All functions in a ranking region generate the same ranking while every ranking is generated by the functions of (at most) one ranking region. Hence, in the offline time it is enough to identify the satisfactory ranking regions whose rankings satisfy fairness constraints.

For 2D, we design a raw-sweeping algorithm. At a high level, using a min-heap for maintaining the ordering exchanges, the algorithm sweeps a ray from the x to y-axis. It first orders the items based on the x-axis and

gradually updates the ordering as it visits an ordering exchange along the way. As the algorithm moves from one ranking region to the other, it checks if the new ranking is fair, and if so, marks the region as satisfactory. Each satisfactory region in 2D is identified by two angles as its beginning and end. We construct a the sorted list of (the borders of) satisfactory regions in the offline phase. Given a query function f in online phase, we apply a binary search on the sorted list. If f falls in a satisfactory region, the algorithm returns f, otherwise it returns the closest satisfactory border to f.

Discovering the satisfactory regions in MD is challenging when there are more than two attributes. That is because the complexity of the arrangement of ordering exchanges is exponential in the number of attributes, d. Even given the satisfactory regions, answering user queries in interactive time is not possible. The reason is that we need to solve a non-linear programming problem for each satisfactory region, before answering each query. To address this issue, we propose an approximation algorithm for obtaining answers quickly, yet accurately. Our approach relies on first partitioning the function space, based on a user-controlled parameter N, into N equi-volume cells, where each cell is a hypercube of $(d-1)$-dimensions. During preprocessing, we assign a satisfactory function f'_c to every cell c such that, for every function f, the angle between f and f'_c is within a bounded threshold (based on the value of N) from f and its optimal answer. To do so, we first identify the cells that intersect with a satisfactory region, and assign the corresponding satisfactory function to each such cell. Then, we assign the cells that are outside of the satisfactory regions to the nearest discovered satisfactory function. In the online phase, given an unsatisfactory function f, we need to find the cell to which f belongs, and to return its satisfactory function. This can be done in $O(\log N)$ by performing binary searches on the partitioned space.

2.4 Obtaining Stable Rankings

Magnitude of the ranking regions that produces an observed ranking identify its stability. Stability is a natural concern for consumers of a ranked list. If a ranking is stable, then the same ranking would be obtained for many choices of weights. But if this region is small, then we know that only a few weight choices can produce the observed ranking. This may suggest that the ranking was engineered or "cherry-picked" by the producer to obtain a specific outcome. Human experts who produce scoring functions for generating the rankings desire to produce stable results. We argued in [15] that stability in a ranked output is an important aspect of algorithmic transparency, because it allows the producer to justify their ranking methodology, and to gain the trust of consumers. Of course, stability cannot be the only criterion in the choice of a scoring function: the result may be weights that seem "unreasonable" to the ranking producer. To support the producer, we allow them to specify a range of reasonable weights, or an *acceptable region* in the space of functions, and help the producer find stable rankings within this region.

We develop a framework [16] that can be used to assess the stability of a provided ranking and to obtain a stable ranking within the the acceptable region of scoring functions. We address the case where the user cares about the rank order of the entire set of items, and also the case where the user cares only about the top-k items. We focus on efficiently evaluating an operator we call GET-NEXT, which can be used to discover the stable rankings, ordered by their stability. Formally, for a dataset $\mathcal{D}$, a region of interest $\mathcal{U}^*$, and the top-$(h-1)$ stable rankings in $\mathcal{U}^*$, discovered by the previous GET-NEXT calls, our goal is to find the h-th stable ranking $\mathfrak{r} \in \mathfrak{R}$. Note that the GET-NEXT operator enables discovering the top-ℓ stable rankings, for any arbitrary ℓ. That simply can be done by calling the operator ℓ times. Our technical contribution for 2D is similar to the one for fair rankings. The 2D algorithm first discovers the ranking regions in $\mathcal{U}^*$ by sweeping a ray in it. The discovered rankings, along with their stabilities, are moved to a heap data structure. Then, every call of GET-NEXT returns the next stable ranking from heap.

For MD, we design a threshold-based algorithm that uses an arrangement [4] tree data structure, AKA cell tree [17], to partially construct the arrangement of ordering exchange hyperplanes. Specifically, given that our objective is to find stable rankings and that the user will likely be satisfied after observing *a few* rankings, rather than discovering all possible rankings, we target the discovery of only the next stable ranking and delay the

arrangement construction for other rankings. Arrangement construction is an iterative process that starts by partitioning the space into two half-spaces by adding the first hyperplane. The construction then iteratively adds the other hyperplanes; to add a new hyperplane, it first identifies the set of regions in the arrangement of previous hyperplanes with which the new hyperplane intersects, and then splits each such region into two new regions. The GET-NEXT operator, however, only breaks down the largest region at every iteration, delaying the construction of the arrangement in all other regions. Please refer to [16] for more details about the algorithm.

While being efficient in practice for medium-size settings, the algorithms based on arrangement construction are not scalable, as their worst-case complexities are cursed by the complexity of the arrangement. Next, we discuss function sampling as a powerful technique for aggregate estimation using Monte-carlo methods [18], as well as an effective technique for search-space exploration.

2.5 Function Sampling for Scalability

Uniform sampling from the scoring function space enables designing randomized algorithms for evaluating and designing score-based evaluators. In the following, we first discuss sampling from the complete function space and then propose an efficient unbiased sampling from a region of interest $\mathcal{U}^*$.

As explained in § 2.2, there is a 1-1 mapping between the universe of scoring functions and the points on the surface of (the first quadrant of) the unit d-sphere. That is, every point one the surface of the d-sphere correspond to a scoring function and vice versa. Hence, the problem of choosing functions uniformly at random from $\mathcal{U}$ is equivalent to choosing random points from the surface of a d-sphere. As also suggested in [19], we adopt a method for uniform sampling of the points on the surface of the unit d-sphere [20, 21]. Rather than sampling the angles, this method samples the weights using the *Normal distribution*, and normalizes them. This works because the normal distribution function has a constant probability on the surfaces of d-spheres with common centers [21]. Therefore, in order to generate a random function in $\mathcal{U}$, we set each weight as $w_i = |\mathcal{N}(0, 1)|$, where $\mathcal{N}(0, 1)$ draws a sample from the standard normal distribution.

In order to compute an aggregate (or conduct exploration) by perturbing in a region of interest $\mathcal{U}^*$ in the neighborhood of some function f, we need to only sample from the set of functions with the maximum angle around the ray of f. An acceptance-rejection method [22] can be used for this purpose. That is, to draw a sample, uniformly at random, from $\mathcal{U}$ and accept it, if it belongs to $\mathcal{U}^*$. The efficiency of this method, however, depends on the volume of $\mathcal{U}^*$, as if it is small, the algorithm will reject most of the generated samples. Alternatively, we propose a sampler that works based on the following observation: modeling $\mathcal{U}^*$ as the surface unit d-spherical cap, each Riemannian piece of the surface forms a $(d-1)$-sphere. Following this observation, our sampler first selects a Riemannian piece, randomly, proportional to its volume. Then it uses the Normal distribution to draw a sample from the surface of the Riemannian piece. Please refer to [16] for more details about the design of this sampler. We would like to emphasize that the proposed sampler has a *linear complexity* to the number of attributes d. It therefore provides a powerful tool for studying the score-based evaluators in higher dimensions.

We use function sampling for different purposes, including (i) evaluating the stability of a given ranking, (ii) designing a randomized algorithm for finding the stable rankings, and (iii) on-the-fly query processing for discovering fair functions. For (i) and (ii), in [16], we design Monte-carlo methods [18] that consume a set of N function samples for finding the rankings in $\mathcal{U}^*$ and computing their stabilities. For (iii), function sampling provides a heuristics for on-the-fly fair ranking scheme query processing in large-scale settings [4].

We used function sampling in MithraRanking [23], our web application[4], designed for responsible ranking design. After uploading a dataset, or choosing among available datasets, the application allows the user to specify the fairness constraints she wants to satisfy. For instance, in Figure 5, the user has added a constraint that the top-30% of the ranking should contain at most 30% with age more than 56 years old. Note that the interface gives the user the ability to add multiple fairness constraints. She then, as in Figure 6, specifies the weight vector

[4] http://mithra.eecs.umich.edu/demo/ranking/

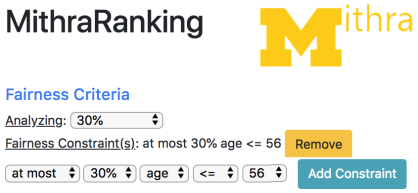

Figure 5: Specifying fairness constraints

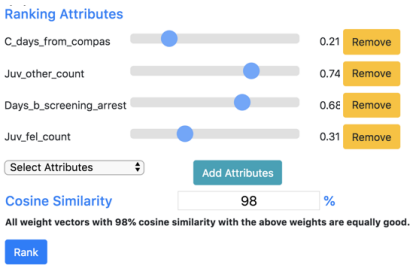

Figure 6: Specifying a weight vector

Figure 7: System results

of the initial scoring function and a region of interest around it, by specifying a cosine similarity. The system then ranks the data based on the specified function and checks if the ranking satisfies the fairness criteria. It also draws unbiased function samples from the region of interest to estimate the stability of the generated ranking. The system also uses the samples for finding the most stable rankings in the region of interest, the most similar fair function to the initial function, and a function (not necessarily the most similar) that generates a fair and stable ranking (Figure 7). The user can then accept any of those suggestions and change the ranking accordingly.

3 Coverage in Training Data

So far in this paper, we discussed responsible design of scoring functions by a human expert. Scoring models are also used for tasks such as classification and prediction Such scoring models can be complex and are often determined using machine learning techniques. An essential piece to the learning is a labeled training dataset. This dataset could be collected prospectively, such as through a survey or a scientific experiment. In such a case, a data scientist may be able to specify requirements such as representation and coverage. However, more often than not, analyses are done with data that has been acquired independently, possibly through a process on which the data scientist has limited, or no, control. This is often called "found data" in the data science context. It is generally understood that the training dataset must be representative of the distribution from which the actual test/production data will be drawn. More recently, it has been recognized that it is not enough for the training data to be representative: it must include enough examples from less popular "categories", if these categories are to be handled well by the trained system. Perhaps the best known story underlining the importance of this inclusion is the case of the "google gorilla" [24]. An early image recognition algorithm released by Google had not been trained on enough dark-skinned faces. When presented with an image of a dark African American, the algorithm labeled her as a "gorilla". While Google very quickly patched the software as soon as the story broke, the question is what it could have done beforehand to avoid such a mistake in the first place. The Google incident is not unique: there have been many other such incidents. For example, Nikon introduced a camera feature to detect whether humans in the image have their eyes open – to help avoid the all-too-common situation of the camera-subject blinking when the flash goes off resulting in an image with eyes closed. Paradoxically for a Japanese company, their training data did not include enough East Asians, so that the software classified many (naturally narrow) open Asian eyes as closed [25].

The problem becomes critical when the data is used for training models for data-driven algorithmic decision making. For example, consider a tool designed to help judges in sentencing criminals by predicting how likely an individual is to re-offend. Such a tool can provide insightful signals for the judge and have the potential to make society safer. On the other hand, a wrong signal can have devastating effects on individuals' lives. So it is important to make sure that the tool is trained on data that includes adequate representation of individuals similar to each person that will be scored by it. In [26], we study a real dataset of criminals used for building such a tool, published by Propublica [8]. We demonstrate that a model with an acceptable overall accuracy had an accuracy worse than random guess for Hispanic females, due to inadequate representation.

While Google's resolution to the gorilla incident was to "ban gorillas" [27], a better solution is to ensure that

the training data has enough entries in each category. Referring to the issue as "disparate predictive accuracy", [28] also highlights that the problem often is due to the insufficient or skewed sample sizes. If the only category of interest were race, as in (most of) the examples above, there are only a handful of categories and this problem is easy. However, in general, objects can have tens of attributes of interest, all of which could potentially be used to categorize the objects. For example, survey scientists use multiple demographic variables to characterize respondents, including race, sex, age, economic status, and geographic location. Whatever be the mode of data collection for the analysis task at hand, we must ensure that there are enough entries in the dataset for each object category. Drawing inspiration from the literature on diversity [29], we refer to this concept as *coverage*.

Lack of coverage in a dataset also opens up the room for adversarial attacks [30]. The goal in an adversarial attack is to generate examples that are misclassified by a trained model. Poorly covered regions in the training data provide the adversary with opportunities to create such examples. For instance, consider the gorilla incident again. Knowing that black people are under-represented in the dataset gives the adversary the information that the models trained using this dataset are not well-trained for this category. The adversary can use this information to generate examples that are misclassified by the model.

Our objective here is two-fold. First, we want to help the dataset users to assess the coverage, as a characterization, of a given dataset, in order to understand such vulnerabilities. For example, we use information about lack of coverage as a widget in the nutritional label [15] of a dataset, in our MithraLabel system[5][31]. Once the lack of coverage has been identified, next we would like to help data owners improve coverage by identifying the smallest number of additional data points needed to hit all the "large uncovered spaces".

Given multiple attributes, each with multiple possible values, we have a combinatorial number of possible *patterns*, as we call combinations of values for some or all attributes. Depending on the size and skew in the dataset, the coverage of the patterns will vary. Given a dataset, our first problem is to efficiently identify patterns that do not have sufficient coverage (the learned model may perform poorly in portions of the attribute space corresponding to these patterns of attribute values). It is straightforward to do this using space and time proportional to the total number of possible patterns. Often, the number of patterns with insufficient coverage may be far fewer. In [26], we develop techniques, inspired from set enumeration and association rule mining (*apriori*) [32], to make this determination efficient.

A more interesting question for the dataset owners is what they can do about lack of coverage. Given a list of patterns with insufficient coverage, they may try to fix these, for example by acquiring additional data. In the ideal case, they will be able to acquire enough additional data to get sufficient coverage for all patterns. However, acquiring data has costs, for data collection, integration, transformation, storage, etc. Given the combinatorial number of patterns, it may just not be feasible to cover all of them in practice. Therefore, we may seek to make sure that we have adequate coverage for at least any combination of ℓ attributes, where we call ℓ the *maximum covered level*. Alternatively, we could identify important pattern combinations by means of a *value count*, indicating how many combinations of attribute values match that pattern. Hence, our goal becomes to determine the patterns for the minimum number of items we must add to the dataset to reach a desired maximum covered level or to cover all patterns with at least a specified minimum value count.

We consider the low-dimensional categorical (sensitive) attributes $\mathcal{A} = \{A_1, A_2, ..., A_d\}$ such as `race`, `gender`, and `age` for studying coverage. Where attributes are continuous valued or of high cardinality, we consider using techniques such as (a) bucketization: putting similar values into the same bucket, or (b) considering the hierarchy of attributes in the data cube for reducing the cardinality of any one attribute. To capture lack of coverage in the dataset, we define a pattern P as a vector of size d, in which $P[i]$ is either X (meaning that its value is unspecified) or is a value of attribute A_i. We name the elements with value X as non-deterministic and the others as deterministic. We say an item t *matches* a pattern P (written as $M(t, P) = \top$), if for all i for which $P[i]$ is deterministic, $t[i]$ is equal to $P[i]$. For example, consider the pattern $P = X1X0$ on four binary attributes A_1 to A_4. It describes the value combinations that have the value 1 on A_2 and 0 on A_4. Hence, for example,

[5]http://mithra.eecs.umich.edu/demo/label/

$t_1 = [1, 1, 0, 0]$ matches P, while $t_3 = [1, 0, 1, 0]$ does not match it, because $P[2] = 1$ and $t_3[2] = 0$. Using the patterns to describe the space of value combinations, we define the coverage of a pattern P as the number of items in $\mathcal{D}$ that match it. We say a pattern P is dominated by another pattern P' if all value combinations matching it also match P'. Our (lack of coverage) identification problem is to discover *Maximal Uncovered Patterns* (MUPs), the set of uncovered patterns (patterns with coverage less than a threshold) that are not dominated by another uncovered pattern. This problem is #P-complete. At a high level, we define a directed acyclic graph (DAG) that captures the domination relation between the patterns and transform the problem into an enumeration on this graph while using the monotonicity property of coverage for pruning the search space.

We note that not all combinations of attribute values are of interest. Some may be extremely unlikely, or even infeasible. For example, we may find few people with attribute `age` as "teen" and attribute `education` as "graduate degree". A human expert, with sufficient domain knowledge, is required to be in the loop for (i) identifying the attributes of interest, over which coverage is studied, (ii) setting up a *validation oracle* that identifies the value combinations that are not realistic, and (iii) identifying the uncovered patterns and the granularity of patterns that should get resolved during the coverage enhancement.

Our coverage enhancement problem is: given a dataset $\mathcal{D}$, its set of material MUPs $\mathcal{M}_{\mathcal{D}}$, and a positive integer number λ, to determine the minimum set of additional tuples to collect such that, after the data collection, the maximum number of deterministic values in any MUP is at least λ. The problem, using a polynomial-time reduction from the vertex cover problem, turns out to be NP-complete. Since a single tuple could contribute to the coverage of multiple patterns, we shall show that this problem translates to a hitting set [33] instance. Using this transformation, we show that the greedy approach provides a logarithmic approximation-ratio for the problem. Given the exponential number of value combinations, the direct implementation of hitting set techniques can be very expensive. Hence, we also provide an efficient implementation of the greedy approach.

4 Final Remarks

In this article we explained our results towards responsible data-driven decision making in score-based systems. The scores, in these systems, are obtained by combining some features using either machine learning models or human-designed weight vectors. We provided our results for (i) assisting the experts to design fair and stable rankings, and (ii) assessing and enhancing coverage in a (given) training dataset for tasks such as classification.

So far, in (i) our focus has been on ranking, where the scores are used for comparing the items in a pool. Human-designed scores are also used for tasks such as classification. Extending our results for these tasks is part of our future work. Also, we would like to adopt the proposed techniques for linear machine learning models. The idea is to first train a machine learning model and then adjust the model to, for example, satisfy some fairness criteria. A similar idea can also be applied for designing ensemble methods for combining the outcome of multiple ML models. In (ii), we used a fixed threshold across different value combinations, representing "minor subgroups". We consider further investigations on identifying threshold value and minor subgroups for future work. We will also investigate other properties (in addition to coverage) for assessing and enhancing the fitness of training data for responsible data science tasks.

References

[1] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[2] A. Asudeh, G. Zhang, N. Hassan, C. Li, and G. V. Zaruba. Crowdsourcing pareto-optimal object finding by pairwise comparisons. In *CIKM*, pages 753–762. ACM, 2015.

[3] FIFA. Fifa/coca-cola world ranking procedure. www.fifa.com/fifa-world-ranking/procedure/men.html, 2008.

[4] A. Asudeh, H. Jagadish, J. Stoyanovich, and G. Das. Designing fair ranking schemes. In *SIGMOD*. ACM, 2019.

[5] B. Salimi, L. Rodriguez, B. Howe, and D. Suciu. Capuchin: Causal database repair for algorithmic fairness. In *SIGMOD*. ACM, 2019.

[6] A. Asudeh, N. Zhang, and G. Das. Query reranking as a service. *PVLDB*, 9(11):888–899, 2016.

[7] M. Gladwell. The order of things: What college rankings really tell us. *The New Yorker Magazine*, 2011.

[8] J. Angwin, J. Larson, S. Mattu, and L. Kirchner. Machine bias: Risk assessments in criminal sentencing. *ProPublica*, 2016.

[9] S. A. Friedler, C. Scheidegger, and S. Venkatasubramanian. On the (im) possibility of fairness. *arXiv preprint arXiv:1609.07236*, 2016.

[10] A. Narayanan. Translation tutorial: 21 fairness definitions and their politics. In *FAT**, 2018.

[11] M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. In *NIPS*, pages 3315–3323, 2016.

[12] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. Zemel. Fairness through awareness. In *ITCS*, 2012.

[13] H. Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10. Springer Science & Business Media, 2012.

[14] T. Jan. Redlining was banned 50 years ago. it's still hurting minorities today. Washington Post, 2018.

[15] K. Yang, J. Stoyanovich, A. Asudeh, B. Howe, H. Jagadish, and G. Miklau. A nutritional label for rankings. In *SIGMOD*, pages 1773–1776. ACM, 2018.

[16] A. Asudeh, H. Jagadish, G. Miklau, and J. Stoyanovich. On obtaining stable rankings. *PVLDB*, 12(3):237–250, 2018.

[17] B. Tang, K. Mouratidis, and M. L. Yiu. Determining the impact regions of competing options in preference space. In *SIGMOD*, 2017.

[18] C. P. Robert. *Monte carlo methods*. Wiley Online Library, 2004.

[19] A. Asudeh, A. Nazi, N. Zhang, G. Das, and H. Jagadish. RRR: Rank-regret representative. In *SIGMOD*. ACM, 2019.

[20] M. E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *CACM*, 2(4), 1959.

[21] G. Marsaglia et al. Choosing a point from the surface of a sphere. *The Annals of Math. Statistics*, 43(2), 1972.

[22] S. Lucidl and M. Piccioni. Random tunneling by means of acceptance-rejection sampling for global optimization. *Journal of optimization theory and applications*, 62(2):255–277, 1989.

[23] Y. Guan, A. Asudeh, P. Mayuram, H. Jagadish, J. Stoyanovich, G. Miklau, and G. Das. Mithraranking: A system for responsible ranking design. In *SIGMOD*, 2019.

[24] M. Mulshine. A major flaw in google's algorithm allegedly tagged two black people's faces with the word 'gorillas'. Business Insider, 2015.

[25] A. Rose. Are face-detection cameras racist? Time Business, 2010.

[26] A. Asudeh, Z. Jin, and H. Jagadish. Assessing and remedying coverage for a given dataset. *ICDE*, 2019.

[27] A. Hern. Google's solution to accidental algorithmic racism: ban gorillas. The Guardian, 2018.

[28] I. Chen, F. D. Johansson, and D. Sontag. Why is my classifier discriminatory? In *NeurIPS*, 2018.

[29] M. Drosou, H. Jagadish, E. Pitoura, and J. Stoyanovich. Diversity in big data: A review. *Big data*, 5(2), 2017.

[30] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *ECML PKDD*, 2013.

[31] C. Sun, A. Asudeh, H. V. Jagadish, B. Howe, and J. Stoyanovich. Mithralabel: Flexible dataset nutritional labels for responsible data science. In *CIKM*. ACM, 2019.

[32] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[33] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.