Perturbation-based Detection and Resolution of Cherry-picking

Abolfazl Asudeh
UI Chicago
asudeh@uic.edu

You (Will) Wu
Google Research
wuyou@google.com

Cong Yu
Google Research
congyu@google.com

H. V. Jagadish
University of Michigan
jag@umich.edu

Abstract

In settings where an outcome, a decision, or a statement is made based on a single option among alternatives, it is popular to cherry-pick the data to generate an outcome that is supported by the cherry-picked data but not in general. In this paper, we use perturbation as a technique to design a support measure to detect, and resolve, cherry-picking across different contexts. In particular, to demonstrate the general scope of our proposal, we study cherry picking in two very different domains: (a) political statements based on trend-lines and (b) linear rankings. We also discuss sampling-based estimation as an effective and efficient approximation approach for detecting and resolving cherry-picking at scale.

1 Introduction

Often, an analysis, a decision, or a statement is made or justified based on a possible selection among a collection of valid alternatives. The selection can be a specific piece of data or choice of parameters. Let us consider two very different examples to understand the issues: trendline statements and multi-criteria rankings. Statements made by politicians are often justified based on evidences from data. For example, a politician may compare the unemployment rate on two dates to highlight the success of their policies or to criticize the other parties. As another example, rankings are also used to compare different entities such as universities. Rankings are often generated, using a weight vector that combines a set of criteria into a score, which is then used to sort the entities.

This enables (purposefully or not) cherry-picking to obtain an outcome that is supported by the cherry-picked data but perhaps not in general. In the political statements example, there are plenty of examples cherry-picking factual basis for making misleading conclusions [1]. For example, in his tweet [2] comparing his approval rate with President Obama's, President Trump cherry-picked a single poll source and a specific date which shows the highest approval for him. In such situations, the outcome based on selected data is valid, but the choice of data or parameters can be questioned. In other words, one can ask whether other alternatives *support* the final outcome. Likewise, rankings are both sensitive and have been highly criticized for cherry-picking. College rankings, for example, have a huge presence in Academia but have often been considered harmful [3, 4]. As M. Vardi nicely explains, each ranking is based on a *specific "methodology"* while the choice of methodology is completely arbitrary [4]. A similar concern has been cast by M. Gladwell [3], given that rankings depend on weights chosen for variables.

Our focus in this paper is on how cherry-picking in different settings can be detected, measured, and resolved. In particular, since data/parameters are carefully selected when cherry-picking, we note that *the outcome should*

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

change by perturbing around them. For example, in President Trump's tweet [2], by slightly changing the dates chosen for comparison, the statement that President Trump has a higher approval rate is not longer valid. To this end, we ask what other alternatives could have been chosen for a similar analysis. We can then look at the outcome from all such alternative options. If the outcome is not based on cherry-picking, it should not differ by much from the reported outcome; i.e., it is *stable*. In contrast, a outcome is presumed to be cherry-picked, if it differs greatly from most alternatives considered. Even if it is not intentionally chosen to mislead, there is no question that it does mislead its consumers about the observed trend.

Of course, this begs the question of what alternatives are valid to consider. In the simplest case, valid options are a set of data/parameters to select from. In other cases, a set of constraints may have to hold for an option to be valid. For instance, for a statement comparing the unemployment rate between two US presidents, a pair of dates form a valid trend if each fall in the range of date each president has been in office. We abstract the universe of valid alternatives for generating an outcome as a "*region of interest*".

Following the above argument, we define a notion of "*support*" to measure cherry-picking. That is, given an output O and a region of interest $\mathcal{U}$, we compute its support as the ratio of the valid alternatives in the region of interest that generate the same outcome. Formally,

$$\omega_{\mathcal{U}}(O) = \frac{\big|\{u_i \in \mathcal{U} | O(u_i) \sim O\}\big|}{|\mathcal{U}|} \tag{1}$$

where $O(u_i)$ is the outcome acquired using the option u_i and $O(u_i) \sim O$ indicates that $O(u_i)$ falls in the "acceptance range" of O. The support of a statement shows what portion of the data "agree" with the outcome O. If an outcome has a small support, it has been generated (whether intentionally or not) by cherry-picking. For example, the low support measure for the statement comparing the approval rate of two presidents verifies that it has been cherry-picked, not supported by the rest of the data.

Using the notion of support, our first mission is to detect if an outcome has been cherry-picked. Formally, we define the cherry-picking problem as follows:

Problem 1 (Cherry-picking Detection): Given an output O and a region of interest $\mathcal{U}$, compute $\omega_{\mathcal{U}}(O)$.

Besides detection, the support measure enables to mine data in order to find the most reliable outcome with the maximum support. Formally we define the cherry-picking resolution problem as following:

Problem 2 (Cherry-picking Resolution): Given a region of interest $\mathcal{U}$, find most supported outcome. That is

$$\underset{O \in \{O(u) | u \in \mathcal{U}\}}{\arg\max} \; w_{\mathcal{U}}(O) \tag{2}$$

Cherry-picking and our notion of support for detecting and resolving it are general, not being limited to a specific domain. Still, following the examples provided in this section, we provide a summary of our research findings for Problems 1 and 2 for (i) political statements base on trendlines [5] and (ii) linear rankings [6].

Paper Organization: First, in § 2, we elaborate on the notion of trendlines and carefully provide the formal definitions. We then discuss the design of exact algorithms both for detecting and resolving cherry-picking trendlines. Next, in § 3, we study cherry-picking in our other application domain, linear ranking, and provide exact algorithmic solutions to address Problems 1 and 2 for such rankings. In § 4, we discuss efficient and effective sampling-based approximation techniques for detecting and resolving cherry-picking. Finally, we conclude with brief sections on related work and future work, respectively.

2 Cherry-picked Trendlines

A *trendline* is a common form of statement that appears in many domains, comparing two windows of points in a timestamped data series. Cherry-picked trendlines are prevalent, for example, in politics, among many other different forms of cherry-picking [1]. The partisans on one side of an argument look for statements they can make about trends that support their position [7]. They would like not to be caught blatantly lying, so they cherry-pick the factual basis for their conclusion. That is, the points based on which a *statement* is made may be carefully selected to show a misleading *trendline* that is not a "reasonable" representation of the situation. Comparing with other forms of statements, the simplicity of a trendline may have also contributed to it being a popular form of cherry-picking. In this section, we focus on trendlines derived by comparing a pair of points in data to make a statement. Formally, such a trendline is defined as follows:

Definition 1 (Trendline): For a dataset $\mathcal{D}$, a trendline θ is a defined as a pair of trend points b (the beginning) and e (the end) and their target values in the form of $\theta = \langle (b, y(b)), (e, y(e)) \rangle$.

For example, in a trendline comparing the unemployment rate in two dates d_1 and d_2 is defined as $\theta = \langle (d_1, uemp(d_1)), (d_2, uemp(d_2)) \rangle$ where $uemp(d_i)$ is the unemployment rate at date d_i. We note that trendlines can be defined over based on the aggregate over a window of points, which as explained in [5] can be transform into the standard trendline form after linear preprocessing. Following the definition of trendline, a trendline statement, or simply a statement, is a claim that is made based on the choice of a trendline. Formally,

Definition 2 (Statement): Given a trendline $\theta = \langle (b, y(b)),\ (e, y(e)) \rangle$, a statement is made by proposing a condition that is satisfied by the target values $y(b))$ and $y(e)$. In this paper, we consider the conditions that are made based on the absolute difference between $y(b)$ and $y(e)$. Formally, given the trendline θ, the statement S_θ is a range $(\perp, \top)$ such that $y(e) - y(b) \in (\perp, \top)$.

For instance, the statement "Unemployment decreased" is made by proposing a condition: $(\perp = -\infty, \top = 0)$, which is satisfied by the selected trendline.

Given a statement S, a support region for S, $R_S = (R(b), R(e))$, is defined as a pair of *disjoint* regions, where every trendline θ_i with the beginning and end points b_i and e_i should satisfy the conditions $b_i \in R(b)$ and $e_i \in R(e)$ to be considered for computing the support of S. A support region may naturally be defined by the statement. For instance, for the statement comparing the approval rate of President Trump with President Obama, $R(b)$ (resp. $R(a)$) is any date when President Trump (resp. President Obama) has been in office.

Not all possible trendlines drawn in the support region may be valid or sensible. For example, for a statement comparing the temperature of location/dates, a trendline that compares the temperature of two different locations on different days may not be valid. Depending on the constraints the choice of one trend point enforces on the other, valid trendlines may categorize into unconstrained trendlines and constrained trendlines. In the rest of this section, we show-case our findings for unconstrained trendlines.

2.1 Cherry-picking Detection

Applying Equation 1 on trendlines, given a data set $\mathcal{D}$, a statement $S = (\top, \perp)$, and a support region $R_S = (R(b), R(e))$, the support for S can be computed as

$$\omega_{R_S}(S, \mathcal{D}) = \frac{\text{vol}(\{\text{valid } \langle p \in R(b), p' \in R(e) \rangle \mid y(p') - y(p) \in (\perp, \top)\})}{\text{vol}(\{\text{valid } \langle p, p' \rangle \mid p \in R(b), p' \in R(e)\})} \tag{3}$$

The denominator this equation is the universe of possible valid trendlines from $R(b)$ and $R(e)$. For unconstrained trendlines, this is the product of the "*volume*" of $R(b)$ and that of $R(e)$. Similarly, the numerator can be rewritten

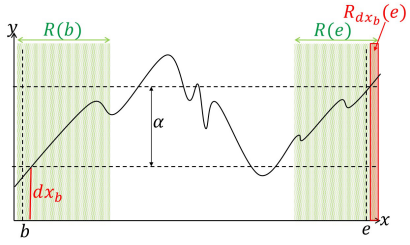


Figure 1: Illustration of a point dx_b and the set of points in $R(e)$ for which $y(dx_e) - y(dx_b) \geq \alpha$.

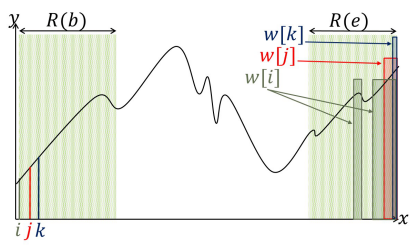


Figure 2: Illustration of weights for three points $dx[i]$, $dx[j]$, and $dx[k]$ in the example of Figure 1.

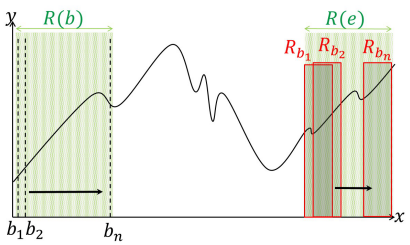


Figure 3: Illustration of the sliding window in $R(b)$ for constrained trendlines.

as a conditional integral as follows:

$$v = \text{vol}(\{\langle p \in R(b), p' \in R(e)\rangle \mid y(p') - y(p) \in (\bot, \top)\}) = \int_{R(b)} \Big(\int_{\{dx \in R(e) \mid y(dx_e) - y(dx_b) \in (\bot,\top)\}} dx_e \Big) dx_b \tag{4}$$

Consider the partitioning of the space into the Riemann pieces (the data records in the dataset $\mathcal{D}$). For a trend point dx_b, let $R_{dx_b}(e)$ be the points in $R(e)$ where $y(dx_e) - y(dx_b) \in (\bot, \top)$. Then, Equation 4 can be rewritten as the sum

$$v = \sum_{\forall dx_b \in R(b)} dx_b \Big(\sum_{\forall dx_e \in R_{dx_b}(e)} dx_e \Big) \tag{5}$$

Consider the example in Figure 1. The horizontal axis shows the trend attribute x while the vertical axis shows y. The trendline of interest is specified by the vertical dashed lines; the left green region identifies $R(b)$ while the one in the right shows $R(e)$, and the curve shows the y values. In this example, the range of the statement S is (α, ∞). A point dx_b in $R(b)$ is highlighted in red in the left of the figure. For dx_b, all points $dx_e \in R(e)$ for which $y(dx_e) - y(dx_b) > \alpha$ support S, forming $R_{dx_b}(e)$ (highlighted in red in the right-hand side of the figure), and therefore, are counted for dx_b. The summation of these counts for all points in $R(b)$ computes the numerator of Equation 5. Following this, the baseline solution sweeps a vertical line from left to right through $R(b)$ and counts the acceptable points in $R(e)$ for each dx_b (similar to highlighted dx_b and R_{dx_b} in Figure 1). For each point in $R(b)$, the baseline algorithm makes a pass over $R(e)$ and, therefore, is *quadratic*: assuming that $|R(e)|$ and $|R(b)|$ are $O(n)$, its run time is $O(n^2)$. In the following, we present an algorithm that pre-processes $R(e)$ in $O(n \log n)$ time, iterates over points in $R(b)$, and utilizes the pre-processed $R(e)$ to compute relevant component of result for each b in $O(\log n)$ time. The overall time complexity is improved significantly to $O(n \log n)$.

Consider Equation 5 once again. For a point $dx[i]$ in $R(b)$, let $w[i]$ be the number of points in $R(e)$ where $y(dx_e) - y(dx[i]) \in (\bot, \top)$, i.e. $\sum_{\forall dx_e \in R_{dx[i]}(e)} dx_e$. Then, Equation 5 can be rewritten as $v = \sum_{\forall dx[i] \in R(b)} w[i]$. For example, in Figure 1, the weight of the point dx_b is the width of the red rectangle $R_{dx_b}(e)$. In the following, we show how the construction of a cumulative function for $R(e)$ enables efficiently finding the corresponding weights for the points in $R(b)$.

In Figure 1, let $dx[1]$ to $dx[n']$ be the set of points in $R(b)$, from left to right. Figure 2 shows three points $dx[i]$, $dx[j]$, and $dx[k]$ where $y(dx[i]) < y(dx[j]) < y(dx[k])$. It also highlights $R_{dx[i]}(e)$, $R_{dx[j]}(e)$, and $R_{dx[k]}(e)$ in the right. Note that $R_{dx[i]}(e)$ consists of two disjoint rectangles. Looking at the figure, one can confirm that $R_{dx[k]}(e)$ is a subset of $R_{dx[j]}(e)$ and $R_{dx[j]}(e)$ is a subset of $R_{dx[i]}(e)$. Since all points in $R_{dx[k]}(e)$ belong to $R_{dx[j]}(e)$ and $R_{dx[i]}(e)$, we do not need to recount those points three time for $dx[i]$, $dx[j]$, and $dx[k]$. Instead, we could start from $dx[k]$, compute its width, move to $dx[j]$, only consider the parts of $R_{dx[j]}(e)$ that is not covered by $R_{dx[k]}(e)$, i.e. $R_{dx[k]}(e) \backslash R_{dx[k]}(e)$, and set $w[j]$ as $w[i]$ plus the width of the uncovered regions by $R_{dx[k]}(e)$. Similarly, in an incremental manner, we could compute $w[i]$, as we sweep over $R(e)$.

Let $dx[1]$ to $dx[n']$ be the set of points in $R(b)$, from left to right. Figure 2 shows three points $dx[i]$, $dx[j]$, and $dx[k]$ where $y(dx[i]) < y(dx[j]) < y(dx[k])$. It also highlights $R_{dx[i]}(e)$, $R_{dx[j]}(e)$, and $R_{dx[k]}(e)$ in the right. Note that $R_{dx[i]}(e)$ consists of two disjoint rectangles. Looking at the figure, one can confirm that $R_{dx[k]}(e)$ is a subset of $R_{dx[j]}(e)$ and $R_{dx[j]}(e)$ is a subset of $R_{dx[i]}(e)$. Since all points in $R_{dx[k]}(e)$ belong to $R_{dx[j]}(e)$ and $R_{dx[i]}(e)$, we do not need to recount those points three time for $dx[i]$, $dx[j]$, and $dx[k]$. Instead, we could start from $dx[k]$, compute its width, move to $dx[j]$, only consider the parts of $R_{dx[j]}(e)$ that is not covered by $R_{dx[k]}(e)$, i.e. $R_{dx[k]}(e) \backslash R_{dx[k]}(e)$, and set $w[j]$ as $w[i]$ plus the width of the uncovered regions by $R_{dx[k]}(e)$. Similarly, in an incremental manner, we could compute $w[i]$, as we sweep over $R(e)$.

Following the above discussion, if we could design a "cumulative" function $F : \mathbb{R} \to \mathbb{R}$, that for every value y, returns the number of points dx in $R(e)$ where $y(dx) < y$, we could use it to directly compute the weights for the points in $R(b)$. Formally, we seek to design the following function $F = |\{dx \in R(e) \mid y(dx) < y\}|$. Given such a function F, the weight of the point $dx[i] \in R(b)$ can be computed as following:

$$w[i] = F\big(y(dx[i]) + \top\big) - F\big(y(dx[i]) + \bot\big) \tag{6}$$

We use a sorted list $\mathfrak{F}$ as the implementation of F. $\mathfrak{F}$ contains the target values in $R(e)$ such that the i-th element in $\mathfrak{F}$ shows the y value for the i-th largest point in $R(e)$. Having the target values sorted in $\mathfrak{F}$, in order to find $F(y)$, it is enough to find index i for which $\mathfrak{F}[i] < y$ and $\mathfrak{F}[i+1] \geq y$. Then, $F(y) = i$. That is because, for all $j \leq i$: $\mathfrak{F}[j] < y$, while for all $j > i$: $\mathfrak{F}[j] \geq y$. Therefore, the number of points for which $y(x) < y$ is equal to i. Also, since the values in $\mathfrak{F}$ are sorted, we can use binary search for finding the index i.

Having the sorted list $\mathfrak{F}$ constructed, the weight of a point $dx \in R(b)$ can be computed using Equation 6 by applying two binary searches over $\mathfrak{F}$. Then making a pass over $R(b)$, we can compute the nominator of Equation 3 as $v = \sum_{\forall dx[i] \in R(b)} w[i]$. The sum is then is used to calculate $\omega(S, R_S)$. Considering $O(n)$ points in each region, the algorithm conducts $O(\log n)$ for each point in $R(b)$ for binary searches, and hence takes $O(n \log n)$ time.

2.2 Cherry-picking Resolution

An immediate question after detecting an statement based on cherry-picked trendlines is *if not this, what is the right statement supported by the data?* For instance, consider a fantastical statement that, cherry-picking a summer day and a winter day, claims in 2012 Summer was colder than winter in Northern Hemisphere. Apparently, using the 2012 weather data, this statement has very low support. Then, a natural question would be: what is the fair statement supported the most by data? For example, considering a 5 degrees Celsius range for the statement, is summer typically warmer than winter by 20-25 degrees Celsius, is it 15-20 degrees, or is it something else? How representative can it be if we would like to make such a statement with a 5 degree difference?

Formally speaking, adjusting Problem 2 for trendlines, given a dataset $\mathcal{D}$, a value d, and a support region R_S, we want to find the statement $S = (\bot, \bot + d)$ with the maximum support. Finding most supported statements (MSS) is challenging. That is because a brute force solution needs to generate *all* possible statements and check the support for each using the techniques provided in the previous sections. Let y_{min} and y_{max} be $\min(\ y(R(e)))$ and $\max(y(R(e)))$ respectively. For MSS, $(y_{max} - y_{min})$ provides a lower bound for $\bot$ and $(y_{max} - y_{min} - d)$ is an upper bound for it. The brute-force algorithm can start from the lower bound, check the support of $S(\bot, \bot + d)$, increase the value of $\bot$ by a small value ϵ, check the support of the new statement, repeat this process until $\bot$ reaches the upper bound, and return the statement with the maximum support. Note that in addition to the efficiency issue, this algorithm cannot guarantee the discovery of the optimal solution, no matter how small ϵ is.

Instead, we first create the "sorted distribution of trendlines." That is, we create a sorted list ℓ (from smallest to largest) where every value is the difference between the target values of a valid trendline. Constructing ℓ requires passing over the pairs of trendlines and then sorting them. Given that the number of pairs is $O(n^2)$, constructing the ordered list takes $O(n^2 \log n)$ time.

Having ℓ constructed, finding the MSS requires a single pass. Recall that every value in ℓ represents the target-value difference of a valid trendline. For a fixed statement range, the support window should contain all

trendlines that their target-value differences belong to the statement range; hence, the window size is variable. The algorithm for finding MSS starts from the beginning of ℓ the algorithm sweeps a window over ℓ. At every step i, it increases the value of j until it finds the index where $(\ell[j] - \ell[i]) \leq d$ while $(\ell[j+1] - \ell[i]) > d$. The support of the statement identified by the current window is $(j-i)/|\ell|$. In the end, the window with the maximum size (therefore maximum support) is returned. Note the values of i and j only get increased during the algorithm until they reach to the end of the list ℓ. As a result, after constructing the sorted list ℓ, the algorithm requires $O(n^2)$ to find the MSS.

3 Cherry-picked Rankings

Compared with trendlines, ranking is commonplace yet challenging, especially when there are multiple criteria to consider. When there is more than one attribute to be considered for ranking, it is common to use a weight vector to linearly combine the criteria into a score that is used for sorting the items. While complex function can also be used for scoring, in this section we will focus on linear ranking functions that are often used in human-designed rankers such as U.S. News and World Report, Times Higher Education, the National Research Council, etc.

Rankings are important as they may have a significant impact on individuals and society, when it comes to, for example, college admissions, employment, university ranking, sports teams/players ranking, etc. 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" [8]. 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 [3]: 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.

Example 1: Consider a real estate company with 5 agents that would like to rank them (for promotion) based on two criteria, x_1 : `customer satisfaction` and x_2 : `sales`. Figure 4 shows the candidates as well as their (normalized) values for x_1 and x_2. Claiming that the company values `sales` slightly more than `customer satisfaction`, they use the weight vector $\vec{w} = \langle 1.1, 1.3 \rangle$, computed as $f(t) = 1.1x_1 + 1.3x_2$ for ranking the agents. The scores generated for each agent is shown in the last column of Figure 1.

The ranking generated in Example 1 has been generated using the weights selected in an ad-hoc manner, while the outcome heavily depend in the selection of weights [3]. In other words, unstable outcomes can be generated by *cherry picking the weights*. In particular, as we shall evaluate it next, it turns out the selected ranking has a low support value, indicating that it (whether intentionally or not) has been cherry-picked.

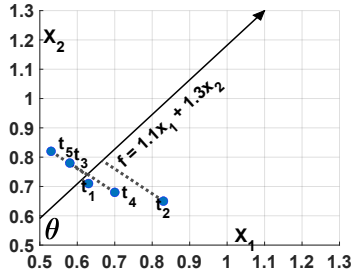
Following Example 1, in the rest of this section we limit our scope to the 2D ranking functions. First, in the following, we provide some background about the geometry of rankings. Next, adjusting the notions of support and region of interest for linear rankings, we propose two algorithms for detecting and resolving cherry-picking. Later in § 4, we will provide a sampling-based approximation approach for MD cases where there are more that than two criteria for ranking.

3.1 Geometry of Rankings

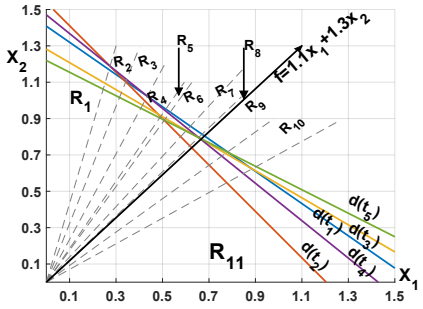
In the popular geometric model for studying data, each attribute is modeled as a dimension and items are interpreted as points in a multi-dimensional space (Figure 4b). This is called the *primal space* where a scoring function is modeled as an origin starting ray and the ranking of items based on it is determined by their projection on the line, as shown in Figure 4b. We transform this primal space into a *dual space* [9], in order to identify regions that help detecting cherry-picking. In the dual space, in $\mathbb{R}^2$, every item t is a line given by $\mathsf{d}(t) : t[1]x_1 + t[2]x_2 = 1$

$\mathcal{D}$			f
id	x_1	x_2	$1.1x_1 + 1.3x_2$
t_1	0.63	0.71	1.34
t_2	0.83	0.65	1.48
t_3	0.58	0.78	1.36
t_4	0.7	0.68	1.38
t_5	0.53	0.82	1.35

(a) A sample database, $\mathcal{D}$, of items with scoring attributes x_1 and x_2; and the result of scoring function $f = 1.1x_1 + 1.3x_2$.



(b) *Original space:* each item is a point. A scoring function is a ray which induces a ranking of the items by their projection.



(c) *Dual space:* items are the lines. Within a region bounded by the intersections of dual lines, all functions induce the same ranking.

Figure 4: A sample database and its geometric interpretation in the original space and dual space.

(Figure 4c). In the dual space, a scoring function f based on the vector $\vec{w}$ translates to an origin starting ray that passes through the point $\vec{w}$. For example, the function f with the weight vector $\vec{w} = \langle 1.1, 1.3 \rangle$ in Example 1 is drawn in Figure 4c as the origin-starting ray that passes through the point $[1.1, 1.3]$.

Every scoring function can then be identified by the angle θ it makes with the x-axis. For example, the function f in Figure 4c is identified by the angle $\theta = \arctan(\frac{1.3}{1.1})$. In other words, there is a one-to-one mapping between possible values for angle θ and the set of possible scoring functions in 2D. This observation enables extending the notion of support for rankings.

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 4c, the intersection of the line t_2 with the ray of $f = 1.1x_1 + 1.3x_2$ is closest to the origin, and t_2 has the highest rank for f.

One observation from the dual space is that the intersections between the dual lines of the items partition the space of possible scoring functions (different values of θ) into discrete regions, called *ranking regions*, where (a) all scoring functions in each region generate the same ranking and (b) no two regions generate the same ranking. In order words, there is a one-to-one mapping between possible rankings and the ranking regions. Formally, let $\mathfrak{R}_{\mathcal{D}}$ be the set of rankings over the items in $\mathcal{D}$ that are generated by at least one choice of weight vector. For a ranking $\mathfrak{r} \in \mathfrak{R}_{\mathcal{D}}$, we define its region, $R_{\mathcal{D}}(\mathfrak{r})$, as the set of functions that generate $\mathfrak{r}$:

$$R_{\mathcal{D}}(\mathfrak{r}) = \{f \mid \nabla_f(\mathcal{D}) = \mathfrak{r}\} \tag{7}$$

Figure 4b shows the boundaries (as dotted lines) of the regions for our sample database, one for each of the 11 feasible rankings. We use the ranking regions in order to extend the notion of *support* for rankings. Looking at the figure, one can observe that the weight vector $\vec{w} = \langle 1.1, 1.3 \rangle$ in Example 1 belongs to a narrow ranking region (R_8) and by slightly changing it, the ranking changes. In other words, it is evident from the figure that the ranking has been cherry-picked.

Every ranking region in 2D can be identified by the two angles in its boundary. Let $\theta_b(R)$ and $\theta_e(R)$ be the beginning and the end angles for the region R. We define the volume of the region, $vol(R) = \theta_e(R) - \theta_b(R)$ to measure the bulk of the region. Similarly, a region of interest in 2D, $\mathcal{U}$, is identified by two angles demarcating the edges of the pie-slice, i.e., $\mathcal{U} = \langle \theta_b, \theta_e \rangle$. For example, let a region of interest be defined by the set of constraints $\{w_1 \leq w_2, \sqrt{3}w_1 \geq w_2\}$. This defines the set of functions above the line $w_1 = w_2$ and below the line $\sqrt{3}w_1 = w_2$, limiting the region of interest to the angles in the range $[\pi/4, \pi/3]$. Similarly, a region defined

around $f = x_1 + x_2$ with the maximum angle $\pi/10°$ corresponds to the angles in the range $[3\pi/20, 7\pi/20]$. The volume of the region of interest can be computed as $\text{vol}(\mathcal{U}) = \theta_e - \theta_b$.

Following Equation 1, the support of a ranking $\mathfrak{r}$ can be measured as the ratio of the volume of its region to the volume of the region of interest: (Equation 1) as following:

$$\omega_{\mathcal{U}}(\mathfrak{r}, \mathcal{D}) = \frac{\text{vol}(R_{\mathcal{D}}(\mathfrak{r}))}{\text{vol}(\mathcal{U})} = \frac{\theta_e(R_{\mathcal{D}}(\mathfrak{r})) - \theta_b(R_{\mathcal{D}}(\mathfrak{r}))}{\theta_e - \theta_b} \tag{8}$$

We note that sometimes in practice not all parts of a ranking are important for studying the support. For example, if the end goal of a ranking is done to select the top-k items, the support value shall be defined on possible (unordered) top-k sets, not the rankings. Similarly, a partial ranking may only look into the top-k (or bottom-k) items. As a generalization of both examples above, inversions at different positions of a ranking may be of different levels of interest/importance (e.g., inverting 10th and 11th items still matters, but not as much as inverting the 1st and the 2nd), and a ranking can be considered as supporting another if the cumulative importance of their inversions is within an acceptable range. [6] elaborates on how to extend the notions of ranking region and support, as well as the detection and resolution algorithms for some of these cases.

3.2 Cherry-picking Detection

The intersections between the lines of items in the dual space, called *ordering exchanges*, are the key in identifying the ranking regions. Consider a ranking $\mathfrak{r}$. For a value of $i \in [1, n)$, let t and t' be the i-th and $(i+1)$-th items in $\mathfrak{r}$. If t dominates t' (i.e., $t[1] > t'[1]$ and $t[2] > t'[2]$) the dual lines $\mathsf{d}(t)$ and $\mathsf{d}(t')$ will not intersect. Otherwise, using the equations of dual lines, the ordering exchange between t and t' can be computed as:

$$\theta_{t,t'} = \arctan \frac{t'[1] - t[1]}{t[2] - t'[2]} \tag{9}$$

If $t[1] < t'[1]$ (resp. $t[1] > t'[1]$), all functions with angles $\theta < \theta_{t,t'}$ (resp. $\theta > \theta_{t,t'}$) rank t higher than t'. The reason is that if $t[1] > t'[1]$, $t[2]$ should be smaller than $t'[2]$, otherwise t dominates t'. Hence $\frac{t[1]}{t[2]} > \frac{t'[1]}{t'[2]}$, i.e. the dual line $\mathsf{d}(t)$ has a larger slope than $\mathsf{d}(t')$, and intersects the rays in range $[0, \theta_{t,t'})$ closer to the origin.

We use this idea for computing the support (and the region) of a given ranking $\mathfrak{r}$. The cherry-picking detection algorithm uses the angle range (θ_1, θ_2), where $0 \leq \theta_1 < \theta_2 \leq \pi/2$, for specifying the region of $\mathfrak{r}$. For each value of i in range $[1, n)$, the algorithm considers the items t and t' to be the i-th and $(i+1)$-th items in $\mathfrak{r}$, respectively. If t' dominates t, the ranking is not valid. Otherwise, if t does not dominate t', the algorithm computes the ordering exchange $\theta_{t,t'}$ and, based on the values of $t[1]$ and $t'[1]$, decides to use it for setting the upper bound or the lower bound of the ranking region. After traversing the ranked list $\mathfrak{r}$, the algorithm returns $(\theta_{min}, \theta_{max})$ as the region of $\mathfrak{r}$, and $\frac{\theta_{max} - \theta_{min}}{\theta_2 - \theta_1}$ as the support of $\mathfrak{r}$. Since the algorithm scans the ranked list only once, computing the support of a ranking in 2D takes $O(n)$ time.

3.3 Cherry-picking Resolution

Similar to other cherry-picking problems such as cherry-picked trendlines, a natural question followed by the detection problem is to find the most supported ranking. This should help the producers of rankings to reveal the ranking that is not just supported by a single function, but the one that has the most support among all possible rankings generated by the functions in the region of interest. Formally, for a dataset $\mathcal{D}$ with n items over d (here $d = 2$) scoring attributes, a region of interest $\mathcal{U}$ (in 2D, $\mathcal{U} = (\theta_b, \theta_e)$), find the ranking $\mathfrak{r}$ with maximum support.

Following the notion of ranking regions, we propose a *ray sweeping* algorithm for finding the most supported ranking, that is, the ranking with the largest region. Let $\mathcal{U} = (\theta_b, \theta_e)$ be the region of interest. The algorithm starts from the angle θ_b and, while sweeping a ray toward θ_e, uses the dual representation of the items for computing the ordering exchanges and finding the ranking regions.

To do so, the algorithm starts by ranking $\mathfrak{r}_{\theta_b}$ the items based on θ_b. It uses the fact that at any moment, an adjacent pair in the ordered list of items exchange ordering, and, therefore, computes the ordering exchanges between the adjacent items in the ordered list. The intersections that fall into the region of interest are added to a min-heap data structure that serve as the sweeper.

Next, it removes the ordering exchange with the minimum angle $\theta_{t,t'}$ from the heap, which together with θ_b for the first ranking region $R(\mathfrak{r}_{\theta_b})$. The support of the first region is $\omega_{\mathcal{U}}(\mathfrak{r}, \mathcal{D}) = (\theta_{t,t'} - \theta_b)/(\theta_e - \theta_b)$, which is the maximum support discovered so far. That is $\omega_{max} = \omega_{\mathcal{U}}(\mathfrak{r}, \mathcal{D})$. After identifying the first region, the algorithm updates its ranking by changing the order between t and t' in it list. The new ranking adds two new ordering exchanges between t and t' and their new neighbors in the ranking, which are added to the sweeper's heap. The algorithm then pops the next ordering exchange from the heap to identify the next ranking region; computes its support; and updates ω_{max} if the new region has a higher support than the best known solution. The algorithm stops when the heap is empty and returns the corresponding ranking with ω_{max} as the output. It also returns the region of the ranking, along with a scoring function that generates the ranking.

The maximum number of ranking regions is $O(n^2)$, since there are at most $\binom{n}{2}$ ordering exchanges between the items. Adding or removing an item from the sweeper's list takes $O(\log n)$, hence the complexity of the cherry-picking resolution algorithm is $O(n^2 \log n)$.

4 Sampling-based Approximation

In large-scale settings where perturbation space, i.e. $\mathcal{U}$, is sizable, it is challenging to either detect or resolve cherry-picking at interactive speed. That is because in such cases even a linear scan over the region of interest to consider all possible cases is time consuming. Consider cherry-picking trendlines as an example. In very large settings where the number of points in $R(b)$ and $R(e)$ is significant, or in the absence of explicit target values where acquiring the data is costly, exact algorithms may not be efficient. The situation is even worse for ranking. So far in this paper, we only considered 2D scoring functions that use two criteria for ranking. In practice, however, there often are more than two criteria for ranking. FIFA rankings, for example uses 4 criteria to rank the national soccer teams [8]. In such cases, due to the curse of dimensionality, the size of the region of interest exponentially grows with d (the number of criteria) and exact algorithms are no longer efficient (please refer to [6] for more details).

On the other hand, approximate estimations of support may often be enough to give the user a good idea about cherry-picking. Hence, a user may prefer to quickly find such estimates, rather than spending a significant amount of time for finding out the exact values. Sampling-based approached, in particular Monte-Carlo methods [10, 11] turn out to be both efficient and accurate for such approximations.

Monte-Carlo methods use repeated sampling and the central limit theorem [12] for solving deterministic problems. Based on the law of large numbers [12], the mean of independent random variables can serve for approximating integrals. That is because the expected number of occurrence of each observation is proportional to its probability. At a high level, the Monte-Carlo methods work as follows: first, they generate a large enough set of random inputs based on a probability distribution over a domain; then they use these inputs to estimate aggregate results.

An important observation is that *uniform sampling from a region of interest $\mathcal{U}$ allows sampling output O based on its support value*. This enables both detection and resolution of cherry-picking by observing different outputs based on the samples and estimating their supports. As a specific topic for the explanation, let us once again consider cherry-picking trendlines. Consider a statement $S = (\bot, \top)$ with the region of interest $R_S = \langle R(b), R(e) \rangle$. The universe of possible trendlines from $R(b)$ to $R(e)$ is the set of valid pairs $\langle p, p' \rangle$ where $p \in R(b)$ and $p' \in R(e)$. Let ω be the support of S in the region R_S, i.e., $\omega(S, R_S)$. For each uniformly sampled pair $\langle p, p' \rangle$, let the random Bernoulli variable $x_{\langle p,p' \rangle}$ be 1 if $y(p') - y(p) \in (\bot, \top)$, 0 otherwise. The probability

distribution function (pdf) of the Bernoulli variable x is:

$$p(x) = \begin{cases} \omega & x = 1 \\ 1 - \omega & x = 0 \end{cases} \tag{10}$$

The mean of a Bernoulli variable with the success probability of x is $\mu = \omega$ and the variance is $\sigma^2 = \omega\,(1-\omega)$. For every set ξ of N iid (independent and identically distributed) samples taken from the above binary variable x, let m_ξ be the random variable showing the average of ξ. Using the central limit theorem, m_ξ follows the Normal distribution $\mathcal{N}\big(\mu, \frac{\sigma}{\sqrt{N}}\big)$ – with the mean μ and standard deviation $\frac{\sigma}{\sqrt{N}}$. Given a confidence level α, the confidence error e identifies the range $[m_\xi - e, m_\xi + e]$ where

$$p(m_\xi - e \leq \mu \leq m_\xi + e) = 1 - \alpha$$

Using the Z-table,

$$e = Z(1 - \frac{\alpha}{2})\frac{\sigma}{\sqrt{N}}$$

For a large enough value of N, we can estimate σ as $\sqrt{m_\xi\,(1-m_\xi)}$. Hence, the confidence error can be computed as:

$$e = Z(1 - \frac{\alpha}{2})\sqrt{\frac{m_\xi\,(1-m_\xi)}{N}} \tag{11}$$

Following the above discussion, the algorithm to estimate the support $\omega(S, R_S)$ uses a budget of N sample trendlines from R_S. The algorithm computes m_ξ by ratio of samples that support S. It then computes the confidence error e, using Equation 11 and returns m_ξ and e. It is easy to see that, since the algorithm linearly scans over N samples, its running time is $O(N)$.

Similarly, the samples can be used to identify the most supported outcome. To see a different application, let us now consider cherry-picking in ranking. In MD where there are $d > 2$ criteria for ranking, every item is represented with a hyperplane $\mathsf{d}(t) : \sum_{i=1}^{d} t[i]x_i = 1$. A scoring function remains as a origin-anchored ray in $\mathbb{R}^d$, identified by $d-1$ angles. In such cases, a region of interest can be described as an origin-anchored hyper-spherical cone, identified by a cosine similarity around an original scoring function. Taking unbiased samples from such an environment becomes challenging, in particular when the region of interest is narrow. Such a sampler is provided in [6, 13]. Having the sampler designed, we can design a Monte-carlo method for identifying the most supported ranking. The algorithm uses a hash data structure that contains the aggregates of the rankings it has observed so far. Upon calling the algorithm, it first draws N sample functions from the region of interest $\mathcal{U}$. For each sampled scoring function, the algorithm finds the corresponding ranking and checks if it has previously been discovered. If not, it adds the ranking to the hash and sets its count as 1; otherwise, it increments the count of the ranking. The algorithm then chooses the ranking that has the maximum count. It computes the support and confidence error of the ranking (using Equation 11) and returns it. Note that following the Monte-carlo method, the algorithm approximately estimates the support of each region and, hence, may miss to return the actual ranking with the maximum support, especially when the number of ranking regions is not small and their supports are close to each other. Still, following the bounds provided by the confidence error, the algorithm guarantees a (user-controllable) upper bound on the difference between the actual maximum support and the algorithm's selection. Considering a budget of N samples while finding the ranking for each sample, the algorithm runs in $O(Nn \log n)$ time. This method has been used in our demo system [14] for responsible ranking design.

5 Related Work

Cherry-picking detection and resolution is closely related to, but not limited to *computational fact checking*, which originated in journalism, with an aim to detect fake news by comparing of claims extracted from the news content against the existing facts [15–22]. The initial fact checking efforts included manual methods based on the domain knowledge of human expert and crowdsourcing [18, 20]. Manual fact checking efforts, however, are not scalable and may not make full use of relevant data. As a result, computational approaches have emerged, with the "Holy Grail" being a platform that can automatically "evaluate" a claim in real-time [17]. Computational fact checking harnesses on techniques from various areas of research such as natural language processing [23, 24], information retrieval [25, 26], and graph theory [15], and spurred novel research including but not limited to multi-source knowledge extraction [27–29], data cleaning and integration [30–32], and credibility evaluation [33, 34]. Existing work also includes style-based [35–38], propagation-based [39–41], and credibility-based [40, 42–45] study of fake news. Further information about fake news and the detection mechanisms can be found in a comprehensive literature survey by Zhou and Zafarani [46].

Using perturbations for studying uncertainty has been studied in different context in data management [47–49]. Perturbation is an effective technique for studying the robustness of query outputs. For example, [6, 14, 50] use function perturbation for verifying the stability of ranking queries, as well as discovering fair and stable rankings. Query perturbation has also been used for retrieving more relevant query results [51–54].

The idea of query perturbation has also seen its applications in the context of the computational journalism, in both fact-checking [21] and lead-finding [55]. Compared with [21], whose focus is more on the modeling of a generic framework for perturbation-based fact-checking, we drill down on two common types of statements—trendlines and linear rankings. On the mining aspect, while [55] studied the representative points to capture the high-value regions of a complex surface, we have treated all points in the support region indifferently, proposed and studied the notion of "support," which is a natural measure that can be defined within the framework and complementary to those defined in [21].

6 Final Remarks and Future Work

In this article, we proposed a measure of support, based on perturbation, to detect and resolve cherry-picking in different contexts. We have demonstrated cherry-picking detection and resolution in two representative types of statements, namely trendlines and linear rankings, with applications in various domains, including but not limited to politics, environment, education, sports, and business intelligence. Besides the exact algorithms, we proposed sampling-based and Monte-Carlo methods as effective approximations for detecting and resolving cherry-picking at scale.

We only focused on the algorithmic aspect of cherry-picking in this paper, which simplifies the problem by assuming the existence of data and a query. Any successful attempt as a real-world system for detection and resolution of cherry-picking needs to address the challenges associated with such assumptions. In the context of trendlines, for example, the first challenge is to translate the (informal) human-language statements to formal trendline statement queries. This requires efficient interaction with human experts for statement formation or (semi-)automatic methods. The next major challenge is to discover the relevant data for evaluating the support of the statement. Discovering relevant data or unbiased samples that can be used for studying cherry-picking is often challenging for real-world scenarios. Fortunately, there have been extensive efforts in the database community, both in designing interactive query systems [56–58] as well as data discovery [59–62], which can be extended for the context of cherry-picking.

Acknowledgments

This research is supported in part by NSF 2107290, 1741022, 1934565, and the Google Research Scholar Award.

References

[1] L. Jacobson. The age of cherry-picking. PolitiFact, Feb. 5, 2018.

[2] L. Jacobson. Donald trump tweet on 50% approval cherry-picks polling data. PolitiFact, June 19, 2017.

[3] M. Gladwell. The order of things: What college rankings really tell us. The New Yorker Magazine, Feb. 14, 2011.

[4] M. Y. Vardi. Academic rankings considered harmful! *Communications of the ACM*, 59(9), 2016.

[5] A. Asudeh, H. V. Jagadish, Y. Wu, and C. Yu. On detecting cherry-picked trendlines. *PVLDB*, 13(6):939–952, 2020.

[6] A. Asudeh, H. Jagadish, G. Miklau, and J. Stoyanovich. On obtaining stable rankings. *PVLDB*, 12(3), 2019.

[7] C. Wardle. Fake news. it's complicated. First Draft News, Feb. 16, 2017.

[8] Fifa/coca-cola world ranking procedure. The Fédération Internationale de Football Association, 28 March 2008.

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

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

[11] F. J. Hickernell, L. Jiang, Y. Liu, and A. B. Owen. Guaranteed conservative fixed width confidence intervals via monte carlo sampling. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 105–128. Springer, 2013.

[12] R. Durrett. *Probability: theory and examples*. Cambridge university press, 2010.

[13] A. Asudeh and H. Jagadish. Responsible scoring mechanisms through function sampling. *CoRR, abs/1911.10073*, 2019.

[14] Y. Guan, A. Asudeh, P. Mayuram, H. Jagadish, J. Stoyanovich, G. Miklau, and G. Das. Mithraranking: A system for responsible ranking design. In *SIGMOD*, pages 1913–1916. ACM, 2019.

[15] S. Cohen, J. T. Hamilton, and F. Turner. Computational journalism. *CACM*, 54(10):66–71, 2011.

[16] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Toward computational fact-checking. *PVLDB*, 7(7):589–600, 2014.

[17] N. Hassan, B. Adair, J. T. Hamilton, C. Li, M. Tremayne, J. Yang, and C. Yu. The quest to automate fact-checking. In *Computation+Journalism Symposium*, 2015.

[18] N. Hassan, F. Arslan, C. Li, and M. Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *SIGKDD*, pages 1803–1812. ACM, 2017.

[19] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, et al. Claimbuster: the first-ever end-to-end fact-checking system. *PVLDB*, 10(12):1945–1948, 2017.

[20] N. Hassan, C. Li, and M. Tremayne. Detecting check-worthy factual claims in presidential debates. In *CIKM*, 2015.

[21] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Computational fact checking through query perturbations. *TODS*, 42(1):4, 2017.

[22] N. Hassan, A. Sultana, Y. Wu, G. Zhang, C. Li, J. Yang, and C. Yu. Data in, fact out: automated monitoring of facts by factwatcher. *PVLDB*, 7(13):1557–1560, 2014.

[23] Y. Li, I. Chaudhuri, H. Yang, S. Singh, and H. Jagadish. Danalix: a domain-adaptive natural language interface for querying xml. In *SIGMOD*, pages 1165–1168. ACM, 2007.

[24] Y. Li, H. Yang, and H. Jagadish. Constructing a generic natural language interface for an xml database. In *ICDT*, pages 737–754. Springer, 2006.

[25] P. A. Bernstein and L. M. Haas. Information integration in the enterprise. *CACM*, 51(9):72–79, 2008.

[26] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Elsevier, 2012.

[27] S. Pawar, G. K. Palshikar, and P. Bhattacharyya. Relation extraction: A survey. *CoRR, abs/1712.05191*, 2017.

[28] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, pages 601–610. ACM, 2014.

[29] R. Grishman. Information extraction. *IEEE Intelligent Systems*, 30(5):8–15, 2015.

[30] R. C. Steorts, R. Hall, and S. E. Fienberg. A bayesian approach to graphical record linkage and deduplication. *Journal of the American Statistical Association*, 111(516):1660–1672, 2016.

[31] A. Magdy and N. Wanas. Web-based statistical fact checking of textual documents. In *SMUC*. ACM, 2010.

[32] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014.

[33] D. Esteves, A. J. Reddy, P. Chawla, and J. Lehmann. Belittling the source: Trustworthiness indicators to obfuscate fake news on the web. *CoRR, abs/1809.00494*, 2018.

[34] X. L. Dong, E. Gabrilovich, K. Murphy, V. Dang, W. Horn, C. Lugaresi, S. Sun, and W. Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *PVLDB*, 8(9):938–949, 2015.

[35] G. D. Bond, R. D. Holman, J.-A. L. Eggert, L. F. Speller, O. N. Garcia, S. C. Mejia, K. W. Mcinnes, E. C. Ceniceros, and R. Rustige. 'lyin'ted','crooked hillary', and 'deceptive donald': Language of lies in the 2016 us presidential debates. *Applied Cognitive Psychology*, 31(6):668–677, 2017.

[36] S. Volkova, K. Shaffer, J. Y. Jang, and N. Hodas. Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on twitter. In *ACL-IJCNLP (Volume 2: Short Papers)*, pages 647–653, 2017.

[37] M. Potthast, J. Kiesel, K. Reinartz, J. Bevendorff, and B. Stein. A stylometric inquiry into hyperpartisan and fake news. *CoRR, abs/1702.05638*, 2017.

[38] D. Pisarevskaya. Deception detection in news reports in the russian language: Lexics and discourse. In *EMNLP Workshop*, pages 74–79, 2017.

[39] J. Ma, W. Gao, and K.-F. Wong. Rumor detection on twitter with tree-structured recursive neural networks. In *ACL-IJCNLP (Volume 1: Long Papers)*, pages 1980–1989, 2018.

[40] K. Wu, S. Yang, and K. Q. Zhu. False rumors detection on sina weibo by propagation structures. In *ICDE*, 2015.

[41] S. Vosoughi, D. Roy, and S. Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[42] Z. Jin, J. Cao, Y. Zhang, and J. Luo. News verification by exploiting conflicting social viewpoints in microblogs. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[43] M. Gupta, P. Zhao, and J. Han. Evaluating event credibility on twitter. In *ICDM*, pages 153–164. SIAM, 2012.

[44] J. Zhang, L. Cui, Y. Fu, and F. B. Gouza. Fake news detection with deep diffusive network model. *CoRR, abs/1805.08751*, 2018.

[45] K. Shu, S. Wang, and H. Liu. Exploiting tri-relationship for fake news detection. *CoRR, abs/1712.07709*, 2017.

[46] X. Zhou and R. Zafarani. Fake news: A survey of research, detection methods, and opportunities. *CoRR, abs/1812.00315*, 2018.

[47] C. C. Aggarwal. *Managing and mining uncertain data*, volume 35. Springer Science & Business Media, 2010.

[48] R. Jampani, F. Xu, M. Wu, L. Perez, C. Jermaine, and P. J. Haas. The monte carlo database system: Stochastic analysis close to the data. *TODS*, 36(3):18, 2011.

[49] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

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

[51] S. Chaudhuri. Generalization and a framework for query modification. In *ICDE*, pages 138–145. IEEE, 1990.

[52] J.-L. Koh, K.-T. Chiang, and I.-C. Chiu. The strategies for supporting query specialization and query generalization in social tagging systems. In *DASFAA*, pages 164–178. Springer, 2013.

[53] S.-Y. Huh, K.-H. Moon, and H. Lee. A data abstraction approach for query relaxation. *IST*, 42(6):407–418, 2000.

[54] C. S. Jensen and R. Snodgrass. Temporal specialization and generalization. *TKDE*, 6(6):954–974, 1994.

[55] Y. Wu, J. Gao, P. K. Agarwal, and J. Yang. Finding diverse, high-value representatives on a surface of answers. *PVLDB*, 10(7):793–804, 2017.

[56] C. Mishra and N. Koudas. Interactive query refinement. In *ICDT*, 2009.

[57] C. Wang, A. Cheung, and R. Bodik. Interactive query synthesis from input-output examples. In *SIGMOD*, 2017.

[58] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *SIGMOD*, pages 517–528, 2014.

[59] R. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE*, pages 1001–1012. IEEE, 2018.

[60] R. Fernandez, E. Mansour, A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*, 2018.

[61] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *SIGMOD*, pages 493–504, 2014.

[62] W.-C. Tan. Deep data integration. In *SIGMOD*, 2021.