Data Mining and Knowledge Discovery
https://doi.org/10.1007/s10618-020-00732-6

ForestDSH: a universal hash design for discrete probability distributions

Arash Gholami Davoodi[1] · Sean Chang[1] · Hyun Gon Yoo[1] · Anubhav Baweja[1] · Mihir Mongia[1] · Hosein Mohimani[1]

Received: 8 November 2019 / Accepted: 21 December 2020

Abstract
In this paper, we consider the problem of classification of high dimensional queries to high dimensional classes from discrete alphabets where the probabilistic model that relates data to the classes is known. This problem has applications in various fields including the database search problem in mass spectrometry. The problem is analogous to the nearest neighbor search problem, where the goal is to find the data point in a database that is the most similar to a query point. The state of the art method for solving an approximate version of the nearest neighbor search problem in high dimensions is locality sensitive hashing (LSH). LSH is based on designing hash functions that map near points to the same buckets with a probability higher than random (far) points. To solve our high dimensional classification problem, we introduce distribution sensitive hashes that map jointly generated pairs to the same bucket with probability higher than random pairs. We design distribution sensitive hashes using a forest of decision trees and we analytically derive the complexity of search. We further show that the proposed hashes perform faster than state of the art approximate nearest neighbor search methods for a range of probability distributions, in both theory and simulations. Finally, we apply our method to the spectral library search problem in mass spectrometry, and show that it is an order of magnitude faster than the state of the art methods.

Keywords Classification · Discrete probability distribution · Hash design · High dimensional data · Locality sensitive hashing

Responsible editor: Ira Assent, Carlotta Domeniconi, Aristides Gionis, Eyke Hüllermeier.

The work of Arash Gholami Davoodi was supported by Lane fellowship and a research fellowship from Alfred P. Sloan Foundation. The work of Hosein Mohimani was supported by a research fellowship from Alfred P. Sloan Foundation and a National Institute of Health New Innovator Award DP2GM137413.

Extended author information available on the last page of the article

1 Introduction

Consider the problem of classifying a large number of high dimensional data $Y = \{y^1, \ldots, y^M\} \subset \mathcal{B}^S$ into high dimensional classes $X = \{x^1, \ldots, x^N\} \subset \mathcal{A}^S$, given a known joint probability distribution $\mathbb{P}(x, y)$, where $\mathcal{A}$ and $\mathcal{B}$ are discrete alphabets. Given a point $y \in Y$, the goal is to find the class $x \in X$ that maximizes $\mathbb{P}(y \mid x)$;

$$\arg\max_{x \in X} \mathbb{P}(y \mid x), \tag{1}$$

where $\mathbb{P}(y \mid x)$ is factorizable to i.i.d. components, i.e.,[1]

$$\mathbb{P}(y \mid x) = \prod_{s=1}^{S} p(y_s \mid x_s). \tag{2}$$

In this paper, we refer to $X = \{x^1, \ldots, x^N\}$ as database points and $Y = \{y^1, \ldots, y^M\}$ as queries. This problem has applications in various fields, including the clustering of spectra generated by mass spectrometry instruments (Frank et al. 2011). Consider the problem where there are billions of data points (mass spectra) and given a query spectrum y the goal is to find the spectrum x that maximizes known probability distribution $\mathbb{P}(y \mid x)$ (Aebersold and Mann 2003; Kim and Pevzner 2014). This is similar to the nearest neighbor search problem (Dasarathy and Sheela 1977; Yianilos 1993), where instead of minimizing a distance, we maximize a probability distribution. Nearest neighbour search problem has applications in machine learning (Duda et al. 1973), database querying (Guttman 1984) and computer vision (Mori et al. 2001; Shakhnarovich et al. 2003) .

A naive approach to solve this problem is to compute $\mathbb{P}(y \mid x)$ for each $x \in X$, and find the maximum. The runtime for this algorithm is $O(NS)$ which is very slow when the number of classes, N, is large.

In order to address this problem where the number of classes is massive, multiclass classification methods have been established (Bentley 1975; Friedman et al. 1977; Prabhu and Varma 2014; Choromanska and Langford 2015; Bhatia et al. 2015; Rai et al. 2015; Jain et al. 2016; Yen et al. 2016; Liu and Tsang 2017; Nam et al. 2017; Tagami 2017; Niculescu-Mizil and Abbasnejad 2017; Zhou et al. 2017). For example, in Choromanska and Langford (2015) an approach to construct a tree with logarithmic (in terms of number of classes) depth is presented and logarithmic train and test time complexity is achieved. However, these methods are limited to low-dimensional data (when the number of dimensions is less than 10), and their complexity grows exponentially with the dimension. The methods in Bentley (1975), Friedman et al. (1977) and Prabhu and Varma (2014) speed up the prediction by rejecting a significant portion of classes for each query. However, all these methods suffer from the curse of dimensionality, i.e., when the dimension of the data increases, the complexity of these

[1] Here, we assume that $\mathbb{P}(x)$ is also factorizable to i.i.d. components.

methods increases linearly with the number of classes.[2] Dimension reduction can also be utilized before applying nearest neighbor search algorithms to high dimensional data (Beyer et al. 1999; Castelli et al. 2000; Min 2005; Anagnostopoulos et al. 2015). Moreover, Structure Preserving Embedding (SPE), which is a low-dimensional embedding algorithm for Euclidean space is also used as a dimension reduction tool (Shaw and Jebara 2009). Set similarity in nearest neighbor search literature is studied extensively (Charikar 2002; Christiani and Pagh 2017). In (Christiani and Pagh 2017), the authors develop a data structure that solves the problem of approximate set similarity search under Braun–Blanquet similarity $B(x, y) = \frac{|x \cap y|}{\max(|x|,|y|)}$ in sub-quadratic query time.

A similar problem has been investigated in the field of nearest neighbor search. In this problem, given a set of points in a database, the goal is to find the point in the database that is closest to a query point. A popular approach to this problem is locality sensitive hashing (Indyk and Motwani 1998; Gionis et al. 1999). This approach solves ϵ-approximate nearest neighbor search problem by designing a family of hashes in a way that near points are hashed to the same bucket with probability much higher than random points. In ϵ-approximate nearest neighbor search problem, given a query point y, the goal is to find $x \in X$ for which $d(x, y) \leq (1 + \epsilon)d(x', y)$ for all $x' \in X$ and X is the set of all feasible points (Indyk and Motwani 1998; Bawa et al. 2005). One of the most popular methods for approximate nearest neighbor search is LSH (Indyk and Motwani 1998; Gionis et al. 1999; Shrivastava and Li 2014; Andoni et al. 2017; Christiani and Pagh 2017; Rubinstein 2018). For any metric space $\mathcal{M} = (M, d)$, a family of hash functions $h : \mathcal{M} \rightarrow S$ is $\mathcal{F}(R, cR, p_1, p_2)$-sensitive if for any two points $x, y \in \mathcal{M}$:

$$\text{If } d(x, y) \leq R, \text{ then } h^{\mathcal{A}}(x) = h^{\mathcal{B}}(y) \text{ with probability at least } p_1. \tag{3}$$

$$\text{If } d(x, y) \geq cR, \text{ then } h^{\mathcal{A}}(x) = h^{\mathcal{B}}(y) \text{ with probability at most } p_2. \tag{4}$$

A family is interesting when $p_1 > p_2$. In the case of hamming distance, i.e., when data are in the form of d-dimensional vectors from $\{0, 1\}^d$, the family of hashes may be defined simply as $H = \cup_{i=1}^{d}\{h : \{0, 1\}^d \rightarrow \{0, 1\} \mid h(x) = x_i\}$ where x_i is i-th coordinate of x. From (3) and (4), we conclude that $p_1 = 1 - \frac{R}{d}$ and $p_2 = 1 - \frac{cR}{d}$.[3]

In this paper, in addition to going from minimizing a distance to maximizing a probabilistic measure, our approach differs from the classic LSH in the following way. The proposed hashes in this paper are defined to be a subset of integers instead of an integer and our goal is to ensure that[4]

$$Prob(H^{\mathcal{A}}(x) \cap H^{\mathcal{B}}(y) \neq \varnothing \mid (x, y) \sim \mathbb{P}) \geq \alpha, \tag{5}$$

$$Prob(H^{\mathcal{A}}(x) \cap H^{\mathcal{B}}(y) \neq \varnothing \mid (x, y) \sim \mathbb{Q}) \leq \beta. \tag{6}$$

[2] The curse of dimensionality holds for non-deterministic probability distributions. When $p(y \mid x)$ is deterministic, i.e., when it takes only a single value with probability one, there is no curse of dimensionality. In this paper, we are interested in the case of non-deterministic probability distributions.

[3] Note that $d(x, y) \leq R$ is equivalent to x and y being different for at most R coordinates.

[4] In fact, to control the complexity, two more conditions are defined in Definition 2. $\mathbb{P}(x, y)$ is joint probability distribution while $\mathbb{P}^{\mathcal{A}}(x)$ and $\mathbb{P}^{\mathcal{B}}(y)$ are marginal probability distributions of $\mathbb{P}(x, y)$. $\mathbb{Q}(x, y)$ is also defined as $\mathbb{Q}(x, y) = \mathbb{P}^{\mathcal{A}}(x)\mathbb{P}^{\mathcal{B}}(y)$.

In words, these hashes hash the jointly-generated pairs of points to the same buckets with probabilities higher than α while hash random pairs of points to the same buckets with probabilities lower than β. Note that, in this case for data points x and y, collision happens when we have $H^{\mathcal{A}}(x) \cap H^{\mathcal{B}}(y) \neq \varnothing$, while in the classic LSH, x and y have collision if $H^{\mathcal{A}}(x) = H^{\mathcal{B}}(y)$. The idea of defining hash collision as the intersection of hash sets has been previously proposed in Christiani and Pagh (2017).

Currently, the locality sensitive hashing approach does not generalize to the cases where the triangle inequality, i.e., $d(x, y) \leq d(x, z) + d(z, y)$ does not hold (Indyk and Motwani 1998; Gionis et al. 1999; Miltersen 1999; Charikar 2002; Chakrabarti and Regev 2010; Andoni and Razenshteyn 2015; Andoni et al. 2015, 2018). These papers are based on constructing balls on some specific points in the metric space and the notion of balls is well defined only for metric spaces that satisfy triangle inequality. Recently, high dimensional approximate nearest neighbor search in a known probabilistic distribution setting have been investigated in (Bawa et al. 2005; Dubiner 2012). However, currently it is not possible to design efficient algorithms based on these methods, due to the large number of parameters involved.

The problem of finding high dimensional approximate nearest neighbors in a known probabilistic setting using a bucketing tree algorithm has been studied previously in Dubiner (2010) and Dubiner (2012). Dubiner uses a strategy to hash the data points from an arbitrary joint probability distribution into the leafs of the tree in a way that the paired data collide with a probability higher than the random pairs. Here, paired data refers to the pairs coming from a joint probability distribution, i.e., $(x, y) \sim \mathbb{P}$ and the random pairs are the pairs coming from an independent probability distribution, i.e., $x \sim \mathbb{P}^{\mathcal{A}}$ and $y \sim \mathbb{P}^{\mathcal{B}}$. However, the algorithm introduced in Dubiner requires solving computationally intractable optimizations, making it impossible to implement [e.g. see equation (126) from Dubiner (2012)]. In this paper, for the specific case where the distribution for any $s \in \{1, 2, \ldots, S\}$ is $p(x_s = 0, y_s = 0) = p(x_s = 1, y_s = 1) = \frac{p}{2}$, $p(x_s = 1, y_s = 0) = p(x_s = 1, y_s = 0) = \frac{1-p}{2}$, our theoretical and practical results are compared to Dubiner (2012) for the range of $0 \leq p \leq 1$.

In this paper, we propose to solve (1) by defining a family of distribution sensitive hashes satisfying the following property. They hash the jointly-generated pairs of points to the same buckets with probabilities much higher than random pairs. We further design an algorithm to solve (1) in sub-linear time using these families of hashes. Next, a method to find optimal family of hashes is presented to achieve minimum search complexity using multiple decision trees. Note that, these decision trees have the same tree structure while they apply to different permutations $\{1, 2, \ldots, S\} \rightarrow \{1, 2, \ldots, S\}$ of the data. This way, we design forest of decision trees where each decision tree captures a very small ratio α of true pairs for some $\alpha \in \mathbb{R}^+$ and by recruiting $\#bands = O(\frac{1}{\alpha})$ independently permuted decision trees we can reach near perfect recovery of all the true pairs. In this paper, we refer to each decision tree as a band and $\#bands$ is referred to as the number of bands.

The main idea is that we construct decision-tree hashes, in a way that the chance of true pairs hashing to the same leaf nodes in these decision trees is higher than random points (Fig. 1). The decision tree is built in a way that the ratio $\frac{\mathbb{P}(x,y)}{\mathbb{Q}(x,y)}$ is higher than a minimum threshold, while the ratios $\frac{\mathbb{P}(x,y)}{\mathbb{P}^{\mathcal{A}}(x)}$ and $\frac{\mathbb{P}(x,y)}{\mathbb{P}^{\mathcal{B}}(y)}$ and the number of nodes in the graph are lower than a maximum thresholds, see Algorithm 4. We further determined the optimal tree among many trees that can be constructed in this way. Two theorems are presented here on the complexity of the decision tree built in Algorithm 4.

1. No decision tree exists with the overall search complexity below $O(N^{\lambda^*})$ where λ^* is derived analytically from the probability distribution $\mathbb{P}(x, y)$.
2. The decision tree construction of Algorithm 4 described in (57) results in the overall search with complexity $O(N^{\lambda^*})$.

Our results show that our approach, Forest-wise distribution sensitive hashing (ForestDSH), provides a universal hash design for arbitrary discrete joint probability distributions, outperforming the existing state of the art approaches in specific range of distributions, in theory and practice. Moreover, we applied this method to the problem of clustering spectra generated from mass spectrometry instruments.

An alternative strategy for solving (2) is to reformulate the problem as minimum inner product search problem (MIPS) (Shrivastava and Li 2014) by transferring the data points into a new space. However, as we show in Sect. 6 the transferred data points are nearly orthogonal to each other making it very slow to find maximum inner product using the existing method (Shrivastava and Li 2014).

Note that, the algorithms presented in this paper are based on the assumption that the true pairs are generated from a known distribution $\mathbb{P}$. The distribution $\mathbb{P}$ can be learned from a training dataset of true pairs. In practice, training data can be collected by running a brute force search on smaller data sets or portion of the whole data. For example, in case of mass spectrometry search, we collect training data by running brute-force search on small portion of our data (Frank et al. 2011). Note that, we can never perfectly learn the probability distribution that the data is generated from. In Theorem 3, we prove that our algorithm is robust to noise in the distribution $\mathbb{P}$ and demonstrate this in an experiment.

Notation The cardinality of a set A is denoted as $|A|$. The sets $\mathbb{N}$ and $\mathbb{R}$ stand for the sets of natural and real numbers, respectively. We use $\mathbb{P}(\cdot)$ and $\mathbb{Q}(\cdot)$ to denote the probability function Prob$(\cdot)$. $\mathbb{E}(v)$ denotes the expected value of the random variable v. Moreover, we use the notation $f(x) = O(g(x))$, if $\limsup_{x\to\infty} \frac{|f(x)|}{g(x)} < \infty$ and the notation $f(x) = \Omega(g(x))$, if $\limsup_{x\to\infty} \frac{|f(x)|}{g(x)} > 0$. In this paper, $\log x$ is computed to the base e.

2 Definitions

Definition 1 Define the S-dimensional joint probability distribution $\mathbb{P}$ as follows:

$$\mathbb{P} : \mathcal{A}^S \times \mathcal{B}^S \to [0, 1], \quad \mathbb{P}(x, y) = \prod_{s=1}^{S} p(x_s, y_s), \tag{7}$$

where $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$, $\mathcal{B} = \{b_1, b_2, \ldots, b_l\}$, $x = (x_1, x_2, \ldots, x_S) \in \mathcal{A}^S$ and $y = (y_1, y_2, \ldots, y_S) \in \mathcal{B}^S$ and $k, l < \infty$. On the other hand, we assume that the probability distribution function $p(a, b)$ is independent of s and satisfies

$\sum_{a\in[k],b\in[l]} p_{a,b} = 1$. Similarly, we define the marginal probability distributions $\mathbb{P}^{\mathcal{A}} : \mathcal{A}^S \rightarrow [0,1]$, $\mathbb{P}^{\mathcal{B}} : \mathcal{B}^S \rightarrow [0,1]$ and $\mathbb{Q} : \mathcal{A}^S \times \mathcal{B}^S \rightarrow [0,1]$ as $\mathbb{P}^{\mathcal{A}}(x) = \prod_{s=1}^{S} p^{\mathcal{A}}(x_s)$, $\mathbb{P}^{\mathcal{B}}(y) = \prod_{s=1}^{S} p^{\mathcal{B}}(y_s)$ and $\mathbb{Q}(x,y) = \prod_{s=1}^{S} q(x_s, y_s)$, where

$$p^{\mathcal{A}}(x_s) = \sum_{j=1}^{l} p(x_s, b_j), \tag{8}$$

$$p^{\mathcal{B}}(y_s) = \sum_{i=1}^{k} p(a_i, y_s), \tag{9}$$

$$q(x_s, y_s) = p^{\mathcal{A}}(x_s) \times p^{\mathcal{B}}(y_s). \tag{10}$$

We use p_{ij} instead of $p(a_i, b_j)$ for simplicity. Moreover, $p_i^{\mathcal{A}}$, $p_j^{\mathcal{B}}$ and q_{ij} are defined as $\sum_{j=1}^{l} p_{ij}$, $\sum_{i=1}^{k} p_{ij}$ and $q(a_i, b_j)$, respectively. Finally, we use compact notations $P = [p_{ij}]$ and $Q = [q_{ij}]$ as the $k \times l$ matrices with p_{ij} and q_{ij} in their ith row and jth column, respectively.

We define family of distribution sensitive hashes as follows.

Definition 2 (*Family of Distribution Sensitive Hashes*) Assume that the four scalar parameters α, β, $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$ along with the probability distributions $\mathbb{P}$, $\mathbb{P}^{\mathcal{A}}$, $\mathbb{P}^{\mathcal{B}}$, $\mathbb{Q}$ and finite set $V_{Buckets}$ are given where $\mathbb{P}^{\mathcal{A}}$ and $\mathbb{P}^{\mathcal{B}}$ are marginal distributions of $\mathbb{P}$ and $\mathbb{Q} = \mathbb{P}^{\mathcal{A}}\mathbb{P}^{\mathcal{B}}$. A family of hashes $H_z^{\mathcal{A}} : \mathcal{A}^S \rightarrow 2^{V_{Buckets}}$ and $H_z^{\mathcal{B}} : \mathcal{B}^S \rightarrow 2^{V_{Buckets}}$ is called $(\mathbb{P}, \alpha, \beta, \gamma^{\mathcal{A}}, \gamma^{\mathcal{B}})$-distribution sensitive, if the following hold

$$\alpha = \sum_{v \in V_{Buckets}} Prob(v \in H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid (x,y) \sim \mathbb{P}), \tag{11}$$

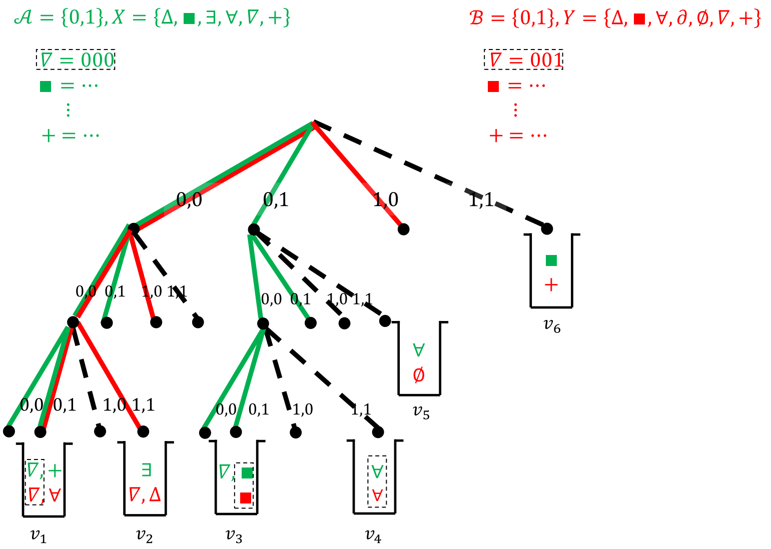
$$\beta = \sum_{v \in V_{Buckets}} Prob(v \in H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid (x,y) \sim \mathbb{Q}), \tag{12}$$

$$\gamma^{\mathcal{A}} = \sum_{v \in V_{Buckets}} Prob(v \in H_z^{\mathcal{A}}(x) \mid x \sim \mathbb{P}^{\mathcal{A}}), \tag{13}$$

$$\gamma^{\mathcal{B}} = \sum_{v \in V_{Buckets}} Prob(v \in H_z^{\mathcal{B}}(y) \mid y \sim \mathbb{P}^{\mathcal{B}}), \tag{14}$$

$$\mid H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid \leq 1, \tag{15}$$

where $1 \leq z \leq \#bands$ and $\#bands$ represent the number of bands, while $V_{Buckets}$ represent a set of indices for the buckets. We show how to choose $\#bands$ in (51) in Sect. 5 and how to select $V_{Buckets}$ in Algorithm 3. Intuitively, α represents the chance of a true pair falling in the same bucket, while β represents the chance of random pairs falling in the same bucket. We will show that $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$ represent the complexity of computing which buckets the data points fall into. In the next section, we will describe how the families of distribution sensitive hashes can be used to design an efficient solution for (1). Note that, in Christiani and Pagh (2017) definitions similar



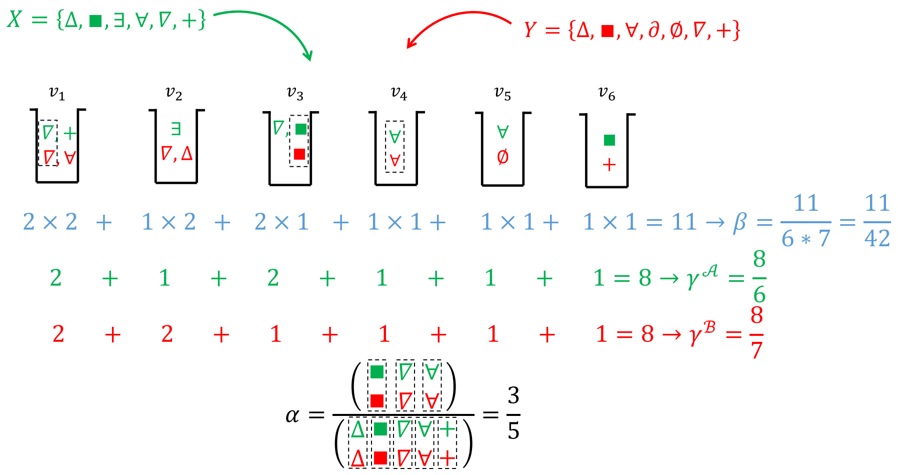
$\mathcal{A} = \{0,1\}, X = \{\Delta, ■, \exists, \forall, \nabla, +\}$
$\mathcal{B} = \{0,1\}, Y = \{\Delta, ■, \forall, \partial, \emptyset, \nabla, +\}$
$\nabla = 000$
$■ = \cdots$
$+ = \cdots$
$\nabla = 001$
$■ = \cdots$
$+ = \cdots$
0,0
0,1
1,0
1,1
v_1
v_2
v_3
v_4
v_5
v_6

Fig. 1 Distribution sensitive hashing is based on a decision tree G and a set of buckets $V_{Buckets}$ which are a subset of the leaf nodes of G. Construction of the tree and $V_{Buckets}$ are explained in Algorithm 4. Here, $V_{Buckets}$ has six buckets in it. We map data points to buckets by propagating them down the tree, starting from the root using Algorithm 3. If we reach a bucket node, we insert the data point. On the other hand, only a subset of leaf nodes are labeled as bucket and if we reach a leaf node that is not a bucket node, we will simply ignore it. The mapping of $\triangledown = 000$ and $\triangledown = 001$ are shown in the figure for $S = 3$. Each of the data points $\triangledown = 001$ and $\triangledown = 000$ are mapped to two buckets. Consider data point $\triangledown = 000$. Mapping this point by propagating down the tree, we end up with the two buckets v_1 and v_3. Note that, the first character on each of the edges in the paths from root to v_1 and v_3 is 0. Similarly mapping $\triangledown = 001$ by propagating down the tree, we end up with the two buckets v_1 and v_2 as the second alphabet on the edges should be 0 in all the first two depths and 1 at the third depth (Color figure online)

to Definition 2 have been made in order to speed up set similarity search. How the data points are mapped through a decision tree is sketched in Fig. 1. The decision tree and the set of buckets are explained in Algorithm 4. Finally, in Fig. 2, we show how the rate of positive calls for true and random pairs, i.e., α and β are derived. The average number of times that data points and queries fall in buckets, i.e., $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$ are also derived and shown in this figure.

Remark 1 In classic LSH, we have $|H^{\mathcal{A}}(x)| = 1$ and $|H^{\mathcal{B}}(y)| = 1$. Therefore, $\gamma^{\mathcal{A}} = \gamma^{\mathcal{B}} = 1$. Our approach is more flexible in the following two ways. First, we allow for $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$ to be larger than one. Moreover, even in the case of $\gamma^{\mathcal{A}} = \gamma^{\mathcal{B}} = 1$, our method can optimize over the larger set of possible hashes.

Here we present a general algorithm on how ForestDSH algorithm works.



$X = \{\Delta, ■, \exists, \forall, \nabla, +\}$
$Y = \{\Delta, ■, \forall, \partial, \emptyset, \nabla, +\}$
v_1
v_2
v_3
v_4
v_5
v_6
$2\times 2 + 1\times 2 + 2\times 1 + 1\times 1 + 1\times 1 + 1\times 1 = 11 \rightarrow \beta = \frac{11}{6*7} = \frac{11}{42}$
$2 + 1 + 2 + 1 + 1 + 1 = 8 \rightarrow \gamma^{\mathcal{A}} = \frac{8}{6}$
$2 + 2 + 1 + 1 + 1 + 1 = 8 \rightarrow \gamma^{\mathcal{B}} = \frac{8}{7}$
$\alpha = \frac{3}{5}$

Fig. 2 Positive calls refer to the pairs of data points that are mapped to the same bucket (see Algorithm 2). The rate of positive call for true and random pairs are shown as α and β. The average number of times x and y data points appear in buckets are also shown as $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$. For formal definition of α, β, $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$, see Definition 2. Here, we derive the empirical probabilities (11)–(14) in Definition 2. For example, β is derived by counting the number of data points combinations in all the six buckets, i.e., $2\times 2+1\times 2+2\times 1+1\times 1+1\times 1+1\times 1$ over all the possible number of data points which is 6×7. Here the size of X is six, and the size of Y is 7. Similarly, $\gamma^{\mathcal{A}}$ is derived by counting the number of data points from the set X designated to the buckets divided by $|X|$

Algorithm 1 ForestDSH

```
**Inputs:** Probability distribution $\mathbb{P}$, $X = \{x^1, \dots, x^N\} \subset \mathcal{A}^S$, $y \in \mathcal{B}^S$, true positive rate $TP$ and threshold $\Delta$.
**Output:** Classes $x \in X$ satisfying $P(y \mid x) > \Delta$.
**Procedure:**
    **Run Algorithm 5.** This algorithm takes probability distribution $\mathbb{P}$, $M$ and $N$ as
    inputs and generates parameters for decision tree design such as $\mu^*$, $\nu^*$, $\eta^*$, $\lambda^*$, and $\delta$.
    **Run Algorithm 4.** Here, the parameters generated in Algorithm 5 are given as an
    input while it generates $G = (V, E, f)$ and a subset of leaf nodes of the decision tree
    $V_{Buckets}$ as outputs.

    **Run Algorithm 3.** In this algorithm, list of buckets $V_{Buckets}$, permutations
    $perm_z, 1 \le z \le \#bands$, and a set of data points $X = \{x^1, \dots, x^N\}$ are given as
    input while the outputs are $H_z^{\mathcal{A}}(x)$ for each $x \in X$ and each band $1 \le z \le \#bands$.
    Similarly, $H_z^{\mathcal{B}}(y)$ are derived from this algorithm.

    **Run Algorithm 2.** This algorithm takes $X = \{x^1, \dots, x^N\} \subset \mathcal{A}^S$, $y \in \mathcal{B}^S$, $\#bands$,
    $V_{Buckets}$, $H_z^{\mathcal{A}}(x)$, $H_z^{\mathcal{B}}(y)$ and threshold $\Delta$ and finds classes $x \in X$ satisfying
    $P(y \mid x) > \Delta$ as an output.
```

3 Forest distribution sensitive hashing algorithm

In this section, we assume an oracle has given us a family of distribution sensitive hashes $H_z^{\mathcal{A}}$, $H_z^{\mathcal{B}}$, $1 \le z \le \#bands$ that satisfies (11)–(15). Inspired by LSH method, we present Algorithm 2 for solving (1) using this family.

Algorithm 2 Solving maximum likelihood classification (1) by ForestDSH

```
Inputs: $X = \{x^1, \ldots, x^N\} \subset \mathcal{A}^S$, $y \in \mathcal{B}^S$, $\#bands$, $V_{Buckets}$, $H_z^{\mathcal{A}}(x)$, $H_z^{\mathcal{B}}(y)$ and threshold $\Delta$.
Output: Classes $x \in X$ satisfying $P(y \mid x) > \Delta$.
For $z \in \{1, 2, \ldots, \#bands\}$
    For $v \in V_{Buckets}$
        For $\{x \in X \mid v \in H_z^{\mathcal{A}}(x)\}$
            If $v \in H_z^{\mathcal{B}}(y)$
                Call $(x, y)$ a positive, compute $\mathbb{P}(y \mid x)$ and report $x$ if $\mathbb{P}(y \mid x) > \Delta$.
```

Remark 2 Note that, the number of times $\mathbb{P}(y \mid x)$ is computed in the brute force method to solve (1) is $|X|$. Note that, in the optimization problem (1), the point $y \in Y$ is given. The goal of Algorithm 2 is to solve (1) with a much smaller number of comparisons than the brute force.

Remark 3 In the special case when the hashes are coming from a decision tree, we analyze the complexity of Algorithm 2 in Sect. 5. We show that the number of positive calls in Algorithm 2 is proportional to β, while the complexity of computing $|H_z^{\mathcal{A}}(x)|$ and $|H_z^{\mathcal{B}}(y)|$ are proportional to $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$. Moreover, the chance of true pairs being called positive grows with α. Therefore, in the next sections, we attempt to design buckets such that α is maximized, while β, $\gamma^{\mathcal{A}}$ and $\gamma^{\mathcal{B}}$ are minimized.

Now, the question is how we can design these families in a way to minimize the complexity, and efficiently map data points to these families. We investigate these questions in Sects. 4 and 4.1 .

4 Designing ForestDSH using a decision tree structure

In the previous section, we assumed that an oracle has given us a family of distribution sensitive hashes. In this section, we design buckets that satisfy (11)–(15) using a forest of decision trees with the same structure. Here, we focus on the probability distributions that can be factorized as the product of i.i.d. components.

Each of our decision trees recovers ratio α of true pairs and by recruiting $\#bands = O(\frac{1}{\alpha})$ decision trees we can recover nearly all true pairs. This can be more efficient than using a single decision tree classifier as achieving near perfect true pair recovery by the single decision tree would require near brute-force complexity. By allowing $\alpha < 1$, we can select decision tree that avoid paths with low $\mathbb{P}(x, y)$ resulting in complexities much lower than the brute-force search. Note that α is the true positive rate, e.g. the probability that a true pair (x, y) fall in the same bucket, therefore we always have $\alpha \leq 1$, see Fig. 2.

Assume that a decision tree $G = (V, E, f)$ is given where V is the set of nodes, E is the set of edges, $V_l \subset V$ is the set of leaf nodes in the decision tree and $f : V/V_l \times \mathcal{A} \times \mathcal{B} \to V$ is the decision function, see Fig. 1. For the two nodes $v_1, v_2 \in V$, v_1 is called an ancestor of v_2 if v_1 falls within the path from v_2 to root. In this case, v_2 is called a descendant of v_1. Furthermore, assume that a subset of leaf nodes $V_{Buckets} \subset V_l$ is given and at depth s in the decision tree, the decisions depend only on x_s and y_s where $x = (x_1, \ldots, x_S)$ and $y = (y_1, \ldots, y_S)$ are S-dimensional data point and query, respectively. We define functions $Seq^{\mathcal{A}} : V \to \cup_{s=0}^{S} \mathcal{A}^s$ and $Seq^{\mathcal{B}} : V \to \cup_{s=0}^{S} \mathcal{B}^s$ recursively as:

$$Seq^{\mathcal{A}}(root) = \varnothing$$
$$Seq^{\mathcal{B}}(root) = \varnothing \tag{16}$$
$$Seq^{\mathcal{A}}(f(v,a,b)) = [Seq^{\mathcal{A}}(v), a] \tag{17}$$
$$Seq^{\mathcal{B}}(f(v,a,b)) = [Seq^{\mathcal{B}}(v), b] \tag{18}$$

where $[S, a]$ stands for the concatenation of string S with character a, i.e., for a string $S = s_1, \ldots, s_n$ of length n, $[S, a]$ would be $s_1, \ldots, s_n, a$ which is a string of length $n + 1$. Moreover, given permutations $p_z : \{1, \ldots, S\} \to \{1, \ldots, S\}, 1 \leq z \leq \#bands$, $x = (x_1, \ldots, x_S)$ and $y = (y_1, \ldots, y_S)$ define $perm_z(x) = (x_{p_z(1)}, x_{p_z(2)}, \ldots, x_{p_z(S)})$ and $perm_z(y) = (y_{p_z(1)}, y_{p_z(2)}, \ldots, y_{p_z(S)})$. Finally, the family of buckets $H_z^{\mathcal{A}}(x)$ and $H_z^{\mathcal{B}}(y)$ are defined as

$$H_z^{\mathcal{A}}(x) = \{v \in V_{Buckets} \mid Seq^{\mathcal{A}}(v) \text{ is a prefix of } \mathrm{perm}_z(x)\}, \tag{19}$$
$$H_z^{\mathcal{B}}(y) = \{v \in V_{Buckets} \mid Seq^{\mathcal{B}}(v) \text{ is a prefix of } \mathrm{perm}_z(y)\}. \tag{20}$$

Note that the permutation $perm_z$ is a deterministic function of z, i.e., at each band the same permutation is used to randomly permute the data points x and y. These permutations are chosen before mapping the data points. In Fig. 3, we show how both x and y data points are first permuted randomly (using the same permutation) and then are mapped to the buckets in decision trees. Note that, here we call a pair positive, if they fall into the same bucket in at least one of the bands. Now, we show that these hashes are distribution sensitive.

Definition 3 The functions $\Phi : V \to \mathbb{R}$, $\Psi^{\mathcal{A}} : V \to \mathbb{R}$ and $\Psi^{\mathcal{B}} : V \to \mathbb{R}$ are defined as follows. At root, $\Phi(root) = 1$, $\Psi^{\mathcal{A}}(root) = 1$ and $\Psi^{\mathcal{B}}(root) = 1$, and for $a_i \in \mathcal{A}$, $b_j \in \mathcal{B}$ and $v \in V$, $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$, and $\Psi^{\mathcal{B}}(v)$ are defined recursively as

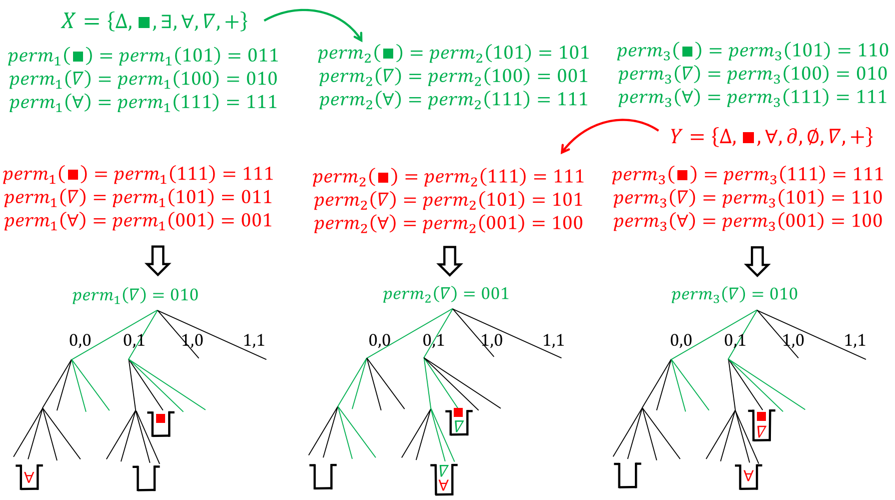
$$\Phi(f(v, a_i, b_j)) = \Phi(v) p_{ij}, \forall v \in V, \tag{21}$$
$$\Psi^{\mathcal{A}}(f(v, a_i, b_j)) = \Psi^{\mathcal{A}}(v) p_i^{\mathcal{A}}, \forall v \in V, \tag{22}$$
$$\Psi^{\mathcal{B}}(f(v, a_i, b_j)) = \Psi^{\mathcal{B}}(v) p_j^{\mathcal{B}}, \forall v \in V. \tag{23}$$

Moreover, $\Psi : V \to \mathbb{R}$ is defined as

$$\Psi(v) = \Psi^{\mathcal{A}}(v) \Psi^{\mathcal{B}}(v), \forall v \in V. \tag{24}$$

Lemma 1 *The following properties hold:*

$$\Phi(v) = Prob(v \in H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid (x, y) \sim \mathbb{P}), \tag{25}$$
$$\Psi(v) = Prob(v \in H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid (x, y) \sim \mathbb{Q}), \tag{26}$$



$X = \{\Delta, ■, \exists, \forall, \nabla, +\}$
$perm_1(■) = perm_1(101) = 011$
$perm_1(\nabla) = perm_1(100) = 010$
$perm_1(\forall) = perm_1(111) = 111$
$perm_2(■) = perm_2(101) = 101$
$perm_2(\nabla) = perm_2(100) = 001$
$perm_2(\forall) = perm_2(111) = 111$
$perm_3(■) = perm_3(101) = 110$
$perm_3(\nabla) = perm_3(100) = 010$
$perm_3(\forall) = perm_3(111) = 111$
$Y = \{\Delta, ■, \forall, \partial, \emptyset, \nabla, +\}$
$perm_1(■) = perm_1(111) = 111$
$perm_1(\nabla) = perm_1(101) = 011$
$perm_1(\forall) = perm_1(001) = 001$
$perm_2(■) = perm_2(111) = 111$
$perm_2(\nabla) = perm_2(101) = 101$
$perm_2(\forall) = perm_2(001) = 100$
$perm_3(■) = perm_3(111) = 111$
$perm_3(\nabla) = perm_3(101) = 110$
$perm_3(\forall) = perm_3(001) = 100$
$perm_1(\nabla) = 010$
$perm_2(\nabla) = 001$
$perm_3(\nabla) = 010$
0,0
0,1
1,0
1,1

Fig. 3 With only a single band, many of the true pairs are missed (not called positives). Refer to (50) and (51) in order to see how multiple bands improve true positive rate. While the decision tree are not affected in any band $1 \leq z \leq \#bands$, classes and queries are permuted by $perm_z$ (using the same permutation $perm_z$) and then mapped to the buckets in decision trees. In this figure, all classes and queries are permuted through three permutations $perm_1(abc) = bac$, $perm_2(abc) = cba$ and $perm_3(abc) = cab$. We showed how three of classes, i.e., ■, ▽, ∀ and three of queries ■, ▽, ∀ are permuted. How ▽ is mapped through these three trees is shown similar to Fig. 1. In the end, all the classes and queries designated to the buckets are shown. Note that, a pair (x, y) is called positive if x and y land in the same bucket in at least one band (Color figure online)

$$\Psi^{\mathcal{A}}(v) = Prob(v \in H_z^{\mathcal{A}}(x) \mid x \sim \mathbb{P}^{\mathcal{A}}), \tag{27}$$

$$\Psi^{\mathcal{B}}(v) = Prob(v \in H_z^{\mathcal{B}}(y) \mid y \sim \mathbb{P}^{\mathcal{B}}). \tag{28}$$

Remark 4 Note that the left side of (25)–(28) is independent of z while it seems like the right side depend on band z. The proof of Lemma 1 in "Appendix 1" shows that in fact it is independent of band z.

Lemma 2 *For any decision tree G, satisfying the condition that for any pair of buckets $v_1, v_2 \in V_{Buckets}$, v_1 is not an ancestor or descendant of v_2, $H_z^{\mathcal{A}}(x)$ and $H_z^{\mathcal{B}}(y)$ defined in (19) and (20) are $(\mathbb{P}, \alpha(G), \beta(G), \gamma^{\mathcal{A}}(G), \gamma^{\mathcal{B}}(G))$-sensitive where*

$$\alpha(G) = \sum_{v \in V_{Buckets}(G)} \Phi(v), \tag{29}$$

$$\beta(G) = \sum_{v \in V_{Buckets}(G)} \Psi(v), \tag{30}$$

$$\gamma^{\mathcal{A}}(G) = \sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{A}}(v), \tag{31}$$

$$\gamma^{\mathcal{B}}(G) = \sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{B}}(v). \tag{32}$$

Proofs of Lemmas 1 and 2 are relegated to "Appendix 1". So far, we showed how to design ForestDSH using a decision tree structure. However, it is not yet clear how to map data points to these buckets. In the next section, we investigate this and provide an algorithm to design optimal decision trees.

4.1 Mapping data points

In the previous sections, we presented Algorithm 2 for solving (1) where we need to compute $H_z^{\mathcal{A}}(x)$ and $H_z^{\mathcal{B}}(y)$ by mapping data points to the buckets. We did not clarify how this mapping can be done efficiently. We present Algorithm 3 for mapping data points to the buckets using hash-table search, see Fig. 2.

Algorithm 3 Mapping data points to the buckets using hash-table search

```
Inputs: List of buckets $V_{Buckets}$, permutations $perm_z, 1 \leq z \leq \#bands$, and a set of data points $X = \{x^1, \ldots, x^N\}$.
Outputs: $H_z^{\mathcal{A}}(x)$ for each $x \in X$ and each band $1 \leq z \leq \#bands$.
Procedure:
    Create an empty hash-table.
    Initialize $H_z^{\mathcal{A}}(x) = \varnothing$ for all $x \in X$ and $1 \leq z \leq \#bands$.
    For $v \in V_{Buckets}$
        Insert $Seq^{\mathcal{A}}(v)$ into the hash-table.
    For $z = 1$ to $\#bands$
        For $x \in X$
            Search $perm_z(x)$ in the hash-table to find all $v \in V_{Buckets}$ for which $Seq^{\mathcal{A}}(v)$
            is a prefix of $perm_z(x)$, and insert $v$ into $H_z^{\mathcal{A}}(x)$ (see Fig. 3).
```

Remark 5 Queries are mapped to buckets using hash-table search through Algorithm 3.

Note that we slightly modify the hash-table to search for values that are prefix of a query, rather than being exactly identical. In Sect. 5, we show that the complexity of Algorithms 2 and 3 can be formulated as:

$$c_{tree}|V(G)| + \left(\frac{c_{hash}N}{\alpha(G)} + \frac{c_{hash}M}{\alpha(G)} + \frac{c_{insertion}N\gamma^{\mathcal{A}}(G)}{\alpha(G)} + \frac{c_{insertion}M\gamma^{\mathcal{B}}(G)}{\alpha(G)} + \frac{c_{pos}MN\beta(G)}{\alpha(G)}\right) \log \frac{1}{1-TP}, \tag{33}$$

where c_{tree}, c_{hash}, $c_{insertion}$ and c_{pos} are constants not depending on N. For the intuition behind (33), note that the first term $c_{tree}|V(G)|$ stands for the time required for calculating and storing the tree. The second and third terms, i.e., $\left(\frac{c_{hash}N}{\alpha(G)} + \frac{c_{hash}M}{\alpha(G)}\right) \log \frac{1}{1-TP}$ denote the time needed for inserting data points to the hash-table. The fourth and fifth terms, i.e., $\left(\frac{c_{insertion}N\gamma^{\mathcal{A}}(G)}{\alpha(G)} + \frac{c_{insertion}M\gamma^{\mathcal{B}}(G)}{\alpha(G)}\right)$ $\log \frac{1}{1-TP}$ stand for the time required for mapping the data points from the hash-table to buckets. Finally, the last term, i.e., $\left(\frac{c_{pos} MN\beta(G)}{\alpha(G)}\right)\log\frac{1}{1-TP}$ is the time of brute-force checking within each bucket.

In order to bound (33) with $O(N^{\lambda})$ for some $\lambda \in \mathbb{R}^{+}$, it is necessary and sufficient to find a tree G that satisfies the following constraints:

$$|V(G)| = O(N^{\lambda}), \tag{34}$$

$$\frac{\alpha(G)}{\beta(G)} = \Omega(N^{1+\delta-\lambda}), \tag{35}$$

$$\frac{\alpha(G)}{\gamma^{\mathcal{A}}(G)} = \Omega(N^{1-\lambda}), \tag{36}$$

$$\frac{\alpha(G)}{\gamma^{\mathcal{B}}(G)} = \Omega(N^{\delta-\lambda}), \tag{37}$$

$$\alpha(G) = \Omega(N^{\max(1,\delta)-\lambda}), \tag{38}$$

where $\delta = \frac{\log M}{\log N}$.

Theorem 1 *Complexity of Algorithms 2 and 3 is equal to (33). Moreover, there exists a decision tree G that is optimal and satisfies (34)–(38) for λ defined in Definition* 4.

Theorem 1 is proved in three steps. In Sect. 5, we prove that the complexity is equal to (33). Theorem 2 shows that the tree G constructed by Algorithm 4 and presented in Sect. 6 satisfies (34)–(38) for λ defined in Definition 4. Theorem 3 shows that this is optimal.

5 Complexity analysis

In this section, we bound the complexity of Algorithms 2 and 3 by (33). Note that, the complexity of Algorithms 2 and 3 is the summation of the following terms.

1. **Tree construction complexity** $c_{tree}|V(G)|$ is the tree construction complexity where c_{tree} is a constant representing per node complexity of constructing a node and $|V(G)|$ is the number of nodes in the tree.
2. **Data mapping complexity** The complexity of this hash-table search grows with

$$c_{hash}(\#bands)|X| + c_{insertion}\sum_{z=1}^{\#bands}\sum_{x\in X}|H_z^{\mathcal{A}}(x)| \tag{39}$$

$$+c_{hash}(\#bands)|Y| + c_{insertion}\sum_{z=1}^{\#bands}\sum_{y\in Y}|H_z^{\mathcal{B}}(y)|, \tag{40}$$

where c_{hash} and $c_{insertion}$ represent complexity of insertion in the hash-table and insertion in the buckets, respectively.
3. **Complexity of checking positive calls** $(\#bands)c_{pos}\sum_{x\in X, y\in Y}|H_z^{\mathcal{A}}(x)\cap H_z^{\mathcal{B}}(y)|)$ is the complexity of checking positive calls where c_{pos} is a constant

representing the complexity of computing $\mathbb{P}(y \mid x)$ for a positive. Note that, c_{tree}, c_{hash}, $c_{insertion}$ and c_{pos} are constants not depending on N. From (11)–(15), we have

$$\begin{aligned}
&\mathbb{E}\left(\sum_{z=1}^{\#bands}\sum_{x\in X}|H_z^{\mathcal{A}}(x)|\right)\\
&\quad=\mathbb{E}\left(\sum_{z=1}^{\#bands}\sum_{x\in X, v\in V_{Buckets}(G)} 1_{x\in v}\right) &&(41)\\
&\quad=(\#bands)N\sum_{v\in V_{Buckets}(G)}\Psi^{\mathcal{A}}(v)=(\#bands)N\gamma^{\mathcal{A}}(G) &&(42)\\
&\mathbb{E}\left(\sum_{z=1}^{\#bands}\sum_{y\in Y}|H_z^{\mathcal{B}}(y)|\right)\\
&\quad=\mathbb{E}\left(\sum_{z=1}^{\#bands}\sum_{y\in Y, v\in V_{Buckets}(G)} 1_{y\in v}\right) &&(43)\\
&\quad=(\#bands)M\sum_{v\in V_{Buckets}(G)}\Psi^{\mathcal{B}}(v)=(\#bands)M\gamma^{\mathcal{B}}(G). &&(44)
\end{aligned}$$

Note that, the total number of collision for random pairs is the sum of number of buckets that they intersect at. Therefore, we conclude that

$$\begin{aligned}
&\mathbb{E}\left(\sum_{z=1}^{\#bands}\sum_{x\in X, y\in Y}|H_z^{\mathcal{A}}(x)\cap H_z^{\mathcal{B}}(y)|\right)\\
&\quad=\sum_{z=1}^{\#bands}\sum_{x\in X, y\in Y} Prob\big(|H_z^{\mathcal{A}}(x)\cap H_z^{\mathcal{B}}(y)|=1\big) &&(45)\\
&\quad=\sum_{z=1}^{\#bands}\sum_{x\in X, y\in Y}\sum_{v\in V_{Buckets}(G)} Prob\big(v\in H_z^{\mathcal{A}}(x), v\in H_z^{\mathcal{B}}(y)\big) &&(46)\\
&\quad=(\#bands)MN\sum_{v\in V_{Buckets}(G)}\Psi(v)=(\#bands)MN\beta(G), &&(47)
\end{aligned}$$

where (45) is concluded as $\mid H^{\mathcal{A}}(x)\cap H^{\mathcal{B}}(y)\mid\leq 1$.

Now, the question is how we can select $\#bands$ such that the true positive rate, defined as the ratio of true pairs that are called positive is high. In each band, the chance of a pair $(x, y)\sim\mathbb{P}$ being called positive is computed as $\alpha(G)=\sum_{v\in V_{Buckets}(G)}\Phi(v)$. Therefore, the overall true positive rate can be computed as:

$$TP = Prob((x, y) \text{ called positive in Algorithm 2 } \mid (x, y)\sim\mathbb{P}) \tag{48}$$

$$= 1 - \prod_{z=1}^{\#bands} \left(1 - \sum_{v \in V_{Buckets}} Prob(v \in H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid (x, y) \sim \mathbb{P})\right) \quad (49)$$

$$= 1 - (1 - \alpha(G))^{\#bands}. \quad (50)$$

Using (50), and the inequality $(1-x)^{\frac{c}{x}} < e^{-c}$, the minimum possible value of $\#bands$ to ensure true positive rate TP can be computed as

$$\#bands = \lceil \frac{\log \frac{1}{1-TP}}{\alpha(G)} \rceil, \quad (51)$$

where $\lceil r \rceil$ stands for the smallest integer greater than or equal to r. Therefore, the total complexity is equal to (33).

6 Constructing optimal decision trees for ForestDSH

In this section, we present an algorithm to design decision trees with complexity $O(N^{\lambda^*})$, where λ^* is defined below, and we show that it is the optimal decision tree.

Definition 4 Given probability distributions $P = [p_{ij}]$ and $Q = [q_{ij}]$, $1 \le i \le k$, $1 \le j \le l$, and number of queries and classes M and N define $\delta = \frac{\log M}{\log N}$ and

$$\mathcal{I} = \{(\mu, \nu, \eta) \in \mathbb{R}^3 | \min(\mu, \nu) \ge \eta \ge 0, \sum_{1 \le i \le k, 1 \le j \le l} p_{ij}^{1+\mu+\nu-\eta} (p_i^{\mathcal{A}})^{-\mu} (p_j^{\mathcal{B}})^{-\nu} = 1\}, \quad (52)$$

$$(\mu^*, \nu^*, \eta^*) = \arg\max_{\mathcal{I}} \frac{\max(1, \delta) + \mu + \nu\delta}{1 + \mu + \nu - \eta}, \quad (53)$$

$$r_{ij}^* = p_{ij}^{1+\mu^*+\nu^*-\eta^*} (p_i^{\mathcal{A}})^{-\mu^*} (p_j^{\mathcal{B}})^{-\nu^*}, \quad (54)$$

$$n^* = \frac{(\max(1, \delta) - \lambda^*) \log N}{\sum r_{ij}^* \log \frac{p_{ij}}{r_{ij}^*}}, \quad (55)$$

$$\lambda^* = \frac{\max(1, \delta) + \mu^* + \nu^*\delta}{1 + \mu^* + \nu^* - \eta^*}. \quad (56)$$

Remark 6 For any probability distribution $\mathbb{P}$, the parameters μ^*, ν^*, η^* and λ^* can be derived numerically from Algorithm 5 in "Appendix 2". The intuition behind the definition of $\mathcal{I}$ and (μ^*, ν^*, η^*) is that in Lemma 4 in Sect. 6.1 we show that for any decision tree G and the variables $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$, $\Psi^{\mathcal{B}}(v)$ and $\Psi(v)$ defined in (21)–(24), we have $\sum_{v \in V_{Buckets}(G)} \left(\Phi(v)\right)^{1+\mu+\nu-\eta} \left(\Psi^{\mathcal{A}}(v)\right)^{-\mu+\eta} \left(\Psi^{\mathcal{B}}(v)\right)^{-\nu+\eta} \left(\Psi(v)\right)^{-\eta} \le 1$ if $(\mu, \nu, \eta) \in \mathcal{I}$. Moreover, in proof of Theorem 2 we show that (μ^*, ν^*, η^*) are Lagrangian multipliers in an optimization problem to minimize the search complexity in Algorithm 2 while retaining a nearly perfect recovery. Consider the optimal decision tree and all the nodes v satisfying

1. Depth of the node v is n^*.
2. For any $1 \le i \le k$, $1 \le j \le l$, the ratio of times we have a_i at s-th position of x and b_j at s-th position of y in all $1 \le s \le n^*$ is r_{ij}.

The node v or one of its ancestors is designated as a bucket by Algorithm 4. This is proved in Sect. 6.2.

In Algorithm 4, we provide an approach for designing decision trees with complexity $O(N^{\lambda^*})$. The algorithm starts with the root, and at each step, it either accepts a node as a bucket, prunes a node, or branches a node into kl children based on the following constraints[5]:

$$\begin{cases} \frac{\Phi(v)}{\Psi(v)} \ge C_1 N^{1+\delta-\lambda^*} & : \text{Accept bucket,} \\ \frac{\Phi(v)}{\Psi^{\mathcal{A}}(v)} \le C_2 N^{1-\lambda^*} & : \text{Prune,} \\ \frac{\Phi(v)}{\Psi^{\mathcal{B}}(v)} \le C_3 N^{\delta-\lambda^*} & : \text{Prune,} \\ \quad otherwise & : \text{Branch into the } kl \text{ children.} \end{cases} \tag{57}$$

Note that, for the cases where $p_{ij} = 0$, we simply remove that branch, therefore, we assume all the p_{ij} are positive real numbers. In Sect. 5, we prove that in order to bound the complexity in (33) with $O(N^{\lambda})$ for some $\lambda \in \mathbb{R}^+$, it is necessary and sufficient to find a decision tree G that satisfies the constraints (34)–(38).

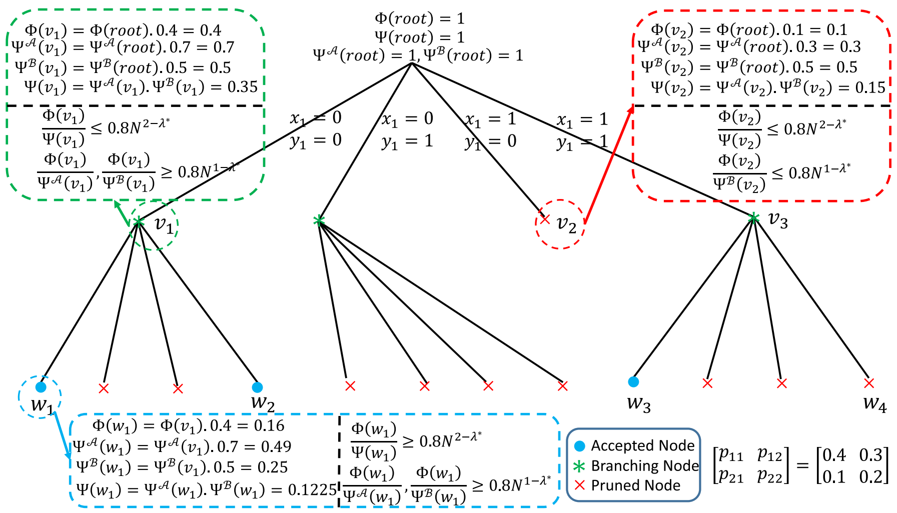
Remark 7 In Theorem 3, we prove that the decision tree construction of Algorithm 4 results in a tree with complexity $O(N^{\lambda^*})$ by setting $C_1 = C_2 = C_3 = p_0 q_0$ where p_0 and q_0 are defined as $\prod_{i,j} p_{ij}$ and $\min(\prod_{i,j} q_{ij}, \prod_i (p_i^{\mathcal{A}})^l, \prod_j (p_j^{\mathcal{B}})^k)$. Pessimistic lower bounds $C_1 = C_2 = C_3 = p_0 q_0$ are derived in proof of Theorem 3 and in practice C_1, C_2 and C_3 are chosen in a way that the best complexity is achieved.

Example 1 Here, we focus on the case where $\mathcal{A} = \{0, 1\}$, $\mathcal{B} = \{0, 1\}$, $P = \begin{bmatrix} 0.4 & 0.3 \\ 0.1 & 0.2 \end{bmatrix}$, $Q = \begin{bmatrix} 0.35 & 0.35 \\ 0.15 & 0.15 \end{bmatrix}$, $\delta = 1$, and $M = N = 4$. From Algorithm 5, we have $\mu^* = 12.0791$, $\nu^* = 13.4206$, $\eta^* = 11.0959$, $\lambda^* = 1.7203$. The decision tree is constructed from Algorithm 4 and is depicted in Fig. 4. The nodes in the tree that are selected as bucket, i.e., satisfying $\frac{\Phi(v)}{\Psi(v)} \ge C_1 N^{1+\delta-\lambda^*}$, are shown with a green check mark, and the pruned nodes, satisfying either $\frac{\Phi(v)}{\Psi^{\mathcal{A}}(v)} \le C_2 N^{1-\lambda^*}$ or $\frac{\Phi(v)}{\Psi^{\mathcal{B}}(v)} \le C_3 N^{\delta-\lambda^*}$ are shown with a red cross. For the non-leaf (intermediate) nodes, none of the above constraints holds.

Theorem 2 *No decision tree exists with overall complexity below* $O(N^{\lambda^*})$.

Theorem 3 *The decision tree construction of Algorithm 4 described in Remark 7 results in a tree with complexity* $O(N^{\lambda^*})$.

[5] If for any node the constraints for accepting as a bucket and pruning hold simultaneously, the algorithm accepts the node as a bucket.



Φ(root) = 1
Ψ(root) = 1
Ψ^A(root) = 1, Ψ^B(root) = 1
x1 = 0, y1 = 0
x1 = 0, y1 = 1
x1 = 1, y1 = 0
x1 = 1, y1 = 1
Φ(v1) = Φ(root).0.4 = 0.4
Ψ^A(v1) = Ψ^A(root).0.7 = 0.7
Ψ^B(v1) = Ψ^B(root).0.5 = 0.5
Ψ(v1) = Ψ^A(v1).Ψ^B(v1) = 0.35
Φ(v1)/Ψ(v1) ≤ 0.8N^{2−λ*}
Φ(v1)/Ψ^A(v1), Φ(v1)/Ψ^B(v1) ≥ 0.8N^{1−λ*}
Φ(v2) = Φ(root).0.1 = 0.1
Ψ^A(v2) = Ψ^A(root).0.3 = 0.3
Ψ^B(v2) = Ψ^B(root).0.5 = 0.5
Ψ(v2) = Ψ^A(v2).Ψ^B(v2) = 0.15
Φ(v2)/Ψ(v2) ≤ 0.8N^{2−λ*}
Φ(v2)/Ψ^B(v2) ≤ 0.8N^{1−λ*}
v1
v2
v3
w1
w2
w3
w4
Φ(w1) = Φ(v1).0.4 = 0.16
Ψ^A(w1) = Ψ^A(v1).0.7 = 0.49
Ψ^B(w1) = Ψ^B(v1).0.5 = 0.25
Ψ(w1) = Ψ^A(w1).Ψ^B(w1) = 0.1225
Φ(w1)/Ψ(w1) ≥ 0.8N^{2−λ*}
Φ(w1)/Ψ^A(w1), Φ(w1)/Ψ^B(w1) ≥ 0.8N^{1−λ*}
● Accepted Node
✱ Branching Node
× Pruned Node
[p11 p12; p21 p22] = [0.4 0.3; 0.1 0.2]

Fig. 4 The decision tree and functions $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$, $\Psi^{\mathcal{B}}(v)$ and $\Psi(v)$ are illustrated for $\mathcal{A} = \{0, 1\}$, $\mathcal{B} = \{0, 1\}$, $C_1 = C_2 = C_3 = 0.8$ and $N = 5$. For all of the nodes v_1, v_2 and w_1, how the decisions are made are explained in the figure. For instance, consider the node w_1. For this node, $\Phi(w_1)$, $\Psi^{\mathcal{A}}(w_1)$, $\Psi^{\mathcal{B}}(w_1)$ are derived from (21)–(23). On the other hand, from (57) this node is accepted as a bucket as $\frac{\Phi(w_1)}{\Psi(w_1)} = 1.31 \geq 0.8N^{2-\lambda^*} = 1.26$. Note that, $\lambda^* = 1.72$ from (56) and Algorithm 5

In other words, Theorem 3 proves that the tree G constructed by Algorithm 4 satisfies (34)–(38), and Theorem 2 shows that this is the optimal decision tree. For proofs of Theorems 2 and 3 , see Sects. 6.1 and 6.2 . Note that, not only Theorem 3 guarantees that the number of the nodes in our decision tree is bounded by $O(N^{\lambda^*})$ but also it guarantees that the runtime for mapping the data points to the decision tree and the number of comparisons that we need to do for the nodes with the collision is bounded by $O(N^{\lambda^*})$, see complexity equation (33) in Sect. 4.1.

Theorem 4 (Noise Robustness) *Assume that the decision tree described in Algorithm 4 is constructed based on distribution $P = [p_{ij}]$ while the data is actually generated from an unknown distribution $P' = [p'_{ij}]$. Assume the distribution $P' = [p'_{ij}]$ is satisfying $\frac{p_{ij}}{1+\epsilon} \leq p'_{ij} \leq p_{ij}(1+\epsilon)$ for all $1 \leq i \leq k, 1 \leq j \leq l$ and some $\epsilon > 0$. Then, it is possible to design an algorithm with complexity $O(N^{\lambda^*(p)+3c_d\log(1+\epsilon)})$ where $c_d = \frac{(\lambda^*-\min(1,\delta))}{\log(\max_{i,j}\min(\frac{p_{ij}}{p_i^{\mathcal{A}}}, \frac{p_{ij}}{p_j^{\mathcal{B}}}))}$ while maintaining arbitrary high true positive rate TP.*

In other words, Theorem 4 proves that the tree G constructed by Algorithm 4 is noise robust, i.e., for the case where our understanding from the distribution is not exact, the arbitrary high true positive rate TP can still be maintained while the complexity increases linearly with the noise. For proof of Theorem 4, see Sect. 6.3. For a high level intuition behind the relationship between Algorithm 4 and Theorems 2 and 3 , note that: Every constraint in the decision tree defined by constraints (57) directly relates to the constraints (34)–(38). Therefore, we expect this decision tree to be optimal.

Algorithm 4 Recursive construction of the decision tree

```
Inputs: $C_1, C_2, C_3, \delta, \mathcal{A}, \mathcal{B}, P, M$ and $N$. We use Algorithm 5 to derive $\mu^*, \nu^*, \eta^*, \lambda^*, \delta$.
Outputs: $G = (V, E, f)$ and a subset of leaf nodes of the decision tree $V_{Buckets}$.
Initialization:
    $Seq^{\mathcal{A}}(root) \leftarrow \varnothing$, $Seq^{\mathcal{B}}(root) \leftarrow \varnothing$.
    $\Phi(root) \leftarrow 1, \Psi^{\mathcal{A}}(root) \leftarrow 1, \Psi^{\mathcal{B}}(root) \leftarrow 1, \Psi(root) \leftarrow 1$.
    Recursive $TreeConstruction(root)$.
Procedure $TreeConstruction(v)$:
    For $a_i \in \mathcal{A}$
        For $b_j \in \mathcal{B}$
            Create a new node $w$.  # The node is created only if $p_{ij} \neq 0$.
            $\Phi(w) \leftarrow \Phi(v) p_{ij}$
            $\Psi^{\mathcal{A}}(w) \leftarrow \Psi^{\mathcal{A}}(v) p_i^{\mathcal{A}}$
            $\Psi^{\mathcal{B}}(w) \leftarrow \Psi^{\mathcal{B}}(v) p_j^{\mathcal{B}}$
            $\Psi(w) \leftarrow \Psi^{\mathcal{A}}(w) \Psi^{\mathcal{B}}(w)$
            $f(v, a_i, b_j) \leftarrow w$
            $Seq^{\mathcal{A}}(w) \leftarrow Seq^{\mathcal{A}}(v)$, $Seq^{\mathcal{B}}(w) \leftarrow Seq^{\mathcal{B}}(v)$
            $Seq^{\mathcal{A}}(w).append(a_i)$, $Seq^{\mathcal{B}}(w).append(b_j)$
            If $\frac{\Phi(w)}{\Psi(w)} \geq C_1 N^{1+\delta-\lambda^*}$                                #Accept bucket
                $V_{Buckets}.insert(w)$
            Else If $\frac{\Phi(w)}{\Psi^{\mathcal{A}}(w)} \geq C_2 N^{1-\lambda^*}$ and $\frac{\Phi(w)}{\Psi^{\mathcal{B}}(w)} \geq C_3 N^{\delta-\lambda^*}$          #Branch
                $TreeConstruction(w)$
            Else                                                    #Prune
                $f(v, a_i, b_j) \leftarrow null$.
```

Here, we present an intuition behind proof of Theorem 2. For rigorous proof of Theorem 2, see “Appendix 3”.

6.1 Intuition behind proof of Theorem 2

Note that, from definition of $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$, $\Psi^{\mathcal{B}}(v)$ and $\Psi(v)$ in (21)–(24), we have $\Phi(w_{ij}) = p_{ij}\Phi(v)$ where v is the parent of w_{ij}. Similar equations hold for $\Psi^{\mathcal{A}}(v)$, $\Psi^{\mathcal{B}}(v)$ and $\Psi(v)$. On the other hand, from (52), note that for any $(\mu, \nu, \eta) \in \mathcal{I}$ we have $\sum_{1\leq i\leq k, 1\leq j\leq l} p_{ij}^{1+\mu+\nu-\eta} (p_i^{\mathcal{A}})^{-\mu} (p_j^{\mathcal{B}})^{-\nu} = 1$. Therefore, we expect to have similar relation between $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$, $\Psi^{\mathcal{B}}(v)$, $\Psi(v)$, i.e.,

$$\sum_{v \in V_{Buckets}(G)} \big(\Phi(v)\big)^{1+\mu+\nu-\eta} \big(\Psi^{\mathcal{A}}(v)\big)^{-\mu+\eta} \big(\Psi^{\mathcal{B}}(v)\big)^{-\nu+\eta} \big(\Psi(v)\big)^{-\eta} \leq 1, \quad (58)$$

for any $(\mu, \nu, \eta) \in \mathcal{I}$. On the other hand, from (29)–(32), we have $\alpha(G) = \sum_{v \in V_{Buckets}(G)} \Phi(v)$. Similar definitions hold for $\beta(G)$, $\gamma^{\mathcal{A}}(G)$ and $\gamma^{\mathcal{B}}(G)$. Therefore, from convexity of the function $f(\theta, \theta_1, \theta_2, \theta_3) = \theta^{1+\rho_1+\rho_2+\rho_3} {\theta_1}^{-\rho_1} {\theta_2}^{-\rho_2} {\theta_3}^{-\rho_3}$ we conclude

$$1 \geq \big(\alpha(G)\big)^{1+\mu^*+\nu^*-\eta^*} \big(\gamma^{\mathcal{A}}(G)\big)^{-\mu^*+\eta^*} \big(\gamma^{\mathcal{B}}(G)\big)^{-\nu^*+\eta^*} \big(\beta(G)\big)^{-\eta^*}, \quad (59)$$

using the lower bounds on $\frac{\alpha(G)}{\beta(G)}$, $\frac{\alpha(G)}{\gamma^{\mathcal{A}}(G)}$, $\frac{\alpha(G)}{\gamma^{\mathcal{B}}(G)}$ and $\alpha(G)$ in (34)–(38) and (59), we conclude $\lambda \geq \lambda^*$.

Here, we present an intuition behind proof of Theorem 3. For rigorous proof of Theorem 3, see "Appendix 4".

6.2 Intuition behind proof of Theorem 3

Here, our goal is to prove that the tree construction steps in Algorithm 4, i.e., (57) result in a tree that satisfies the following three statements:

1. There is at least one node in the tree which is accepted as a bucket.
2. [(35)–(38)] holds.
3. Number of nodes in the tree is bounded by $O(N^{\lambda^*})$.

Let us intuitively prove all these three statements one by one.

1. Consider a node with the depth equal to n^* given in (55). Moreover, assume that the number of times we have a_i at sth position of x and b_j at sth position of y for all $1 \leq s \leq n^*$ are $n^* r_{ij}^*$. Then, we argue that this node is not pruned and accepted as a bucket. In order to understand why intuitively it is true, we verify that this node is accepted as a bucket, i.e., we have to verify that $\frac{\Phi(v)}{\Psi(v)} \geq N^{1+\delta-\lambda^*} p_0 q_0$. This is true as
$$\frac{\Phi(v)}{\Psi(v)} = \Omega(e^{\sum_{i,j} n^* r_{ij}^* \log \frac{p_{ij}}{q_{ij}}}) \tag{60}$$
$$\geq \Omega(N^{1+\delta-\lambda^*}), \tag{61}$$
(60) follows from (21), (22) and the definition of node v, i.e., the number of times we have a_i at s-th position of x and b_j at s-th position of y are $n^* r_{ij}^*$. (61) is concluded from the definition of r_{ij}^* in (54). For further details, see "Appendix 4".
2. Let us prove (35) as the rest of [(36)–(38)] follow similarly. From Algorithm 4, for all the buckets we have
$$\frac{\Phi(v)}{\Psi(v)} \geq N^{1+\delta-\lambda^*} p_0 q_0. \tag{62}$$
Note that, at least there is one node that is accepted a a bucket. On the other hand, $\alpha(G) = \sum_{v \in V_{Buckets}(G)} \Phi(v)$ and $\beta(G) = \sum_{v \in V_{Buckets}(G)} \Psi(v)$. Therefore, we conclude that
$$\frac{\alpha(G)}{\beta(G)} = \frac{\sum_{v \in V_{Buckets}(G)} \Phi(v)}{\sum_{v \in V_{Buckets}(G)} \Psi(v)} \geq N^{1+\delta-\lambda^*} p_0 q_0. \tag{63}$$
Note that $\frac{\sum_i a_i}{\sum_i b_i} \geq c$ if $\frac{a_i}{b_i} \geq c$ and $b_i > 0$ for any i.
3. Finally, we prove that number of nodes in the tree is bounded by $O(N^{\lambda^*})$. Note that $\sum_{i,j} p_{ij} = 1$, therefore, we expect to have $\sum_{v \in V_l(G)} \Phi(v) = 1$. On the

other hand, $\Phi(v)$ is expected to be greater than $N^{-\lambda^*}$ for the intermediate nodes from (57). Therefore, it is concluded that $|V_l(G)|$ is at most N^{λ^*}. This results in $|V(G)| = O(N^{\lambda^*})$ as for any decision tree we have $|V(G)| \leq 2|V_l(G)|$. For details, see "Appendix 4".

.

Below, we present an intuition behind proof of Theorem 4. For rigorous proof of Theorem 4, see "Appendix 5".

6.3 Intuition behind proof of Theorem 4

Define $\Phi'(v), \alpha'(G), \beta'(G), \#bands'$ for p'_{ij} the same way as $\Phi(v), \alpha(G), \beta(G)$, $\#bands$ for p_{ij}. As $\frac{p_{ij}}{1+\epsilon} \leq p'_{ij} \leq p_{ij}(1+\epsilon)$, we expect $\Phi(v)$ to be bounded as

$$\Phi(v)\frac{1}{(1+\epsilon)^d} \leq \Phi'(v) \leq \Phi(v)(1+\epsilon)^d \tag{64}$$

where $d = depth(G)$. Therefore, from (29) we have

$$\alpha(G) = \sum_{v \in V_{Buckets}(G)} \Phi(v) \tag{65}$$

$$\leq \sum_{v \in V_{Buckets}(G)} \Phi'(v)(1+\epsilon)^d \tag{66}$$

$$\leq \alpha'(G)(1+\epsilon)^d. \tag{67}$$

We conclude similar inequalities for $\gamma^{\mathcal{A}}(G)$ and $\gamma^{\mathcal{B}}(G)$, while for $\beta(G)$ we have

$$\beta'(G)(1+\epsilon)^{2d} \leq \beta(G) \leq \beta'(G)\frac{1}{(1+\epsilon)^{2d}}. \tag{68}$$

From (51), $\#bands$ is inversely related to $\alpha(G)$. Therefore, $\#bands'$ can be bounded from above by $(1+\epsilon)^d\#bands$. As a result, from (33) the total complexity is bounded by $(1+\epsilon)^{3d}N^{\lambda^*}$ from above. Assume that d is bounded by $c_d \log N$ for some constant c_d. Therefore, we conclude Theorem 4. In order to see why $d \leq c_d \log N$, note that for all the leaf nodes, we have $\frac{\Phi(v)}{\Psi^{\mathcal{A}}(v)} \leq N^{1+\delta-\lambda^*} \max_{i,j} \frac{p_{ij}}{p_i^{\mathcal{A}}} p_0 q_0$. On the other hand, $\frac{\Phi(v)}{\Psi^{\mathcal{A}}(v)}$ can be bounded by $\left(\min_{i,j} \frac{p_{ij}}{p_i^{\mathcal{A}}}\right)^d$ from below. Therefore, we conclude that $d = c_d \log N$ for some constant c_d.

7 Experiments and observations

Experiment 1 In this experiment, we compare the complexity of ForestDSH with the algorithm proposed by Dubiner in Dubiner (2012). Here, we set $\mathcal{A} = \mathcal{B} = \{0, 1\}$, $S = 1000$, $M = N = 1{,}000{,}000$. For ForestDSH, we use Algorithm 5 to

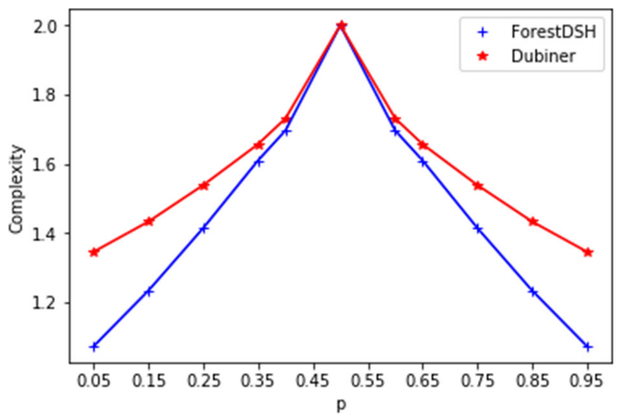


Fig. 5 Comparing the practical performances of Dubiner algorithm in Dubiner (2012) with ForestDSH for $S = 1000$. ForestDSH outperforms Dubiner's method for all values of p. The pseudo code for the algorithm presented in Dubiner (2012) for the case of hamming distance is generated in this paper in Algorithm 6 in "Appendix 6"

derive μ^*, ν^*, η^*, λ^* and δ while training C_1, C_2 and C_3 to get the best complexity in Algorithm 4. For Dubiner's algorithm, equation (126) in Dubiner (2012) is computationally intractable which makes it impossible to compute the complexity for the algorithm presented there for general probability distributions. However, in the special case where $P(p) = \begin{bmatrix} \frac{p}{2} & \frac{1-p}{2} \\ \frac{1-p}{2} & \frac{p}{2} \end{bmatrix}$, $0.5 \le p \le 1$ (hamming distance), a solution has been provided for computing the complexity in Dubiner (2012). We implemented that solution (see "Appendix 6" for the detail of implementation), and compared it to ForestDSH. Note that currently no source code for the implementation of Dubiner algorithm is available. Figure 5, shows that Dubiner algorithm's performance is worse than that of ForestDSH.

Observation 1 In this observation, we reformulate the problem of solving (2) to the minimum inner product search problem (MIPS) (Shrivastava and Li 2014) by transferring the data points from $\mathcal{A}^S$ and $\mathcal{B}^S$ to R^{klS} in a way that $\log\left(\frac{\mathbb{P}(x,y)}{\mathbb{Q}(x,y)}\right)$ is equal to the dot product in this new space. We transformed $x \in \mathcal{A}^S$ and $y \in \mathcal{B}^S$ to $T(x) \in \mathbb{R}^{klS}$ and $T(y) \in \mathbb{R}^{klS}$ as follows:

$$T(x) = \left(f_{s,i,j}\right), 1 \le s \le S, 1 \le i \le k, 1 \le j \le l, \tag{69}$$

$$f_{s,i,j} = \begin{cases} \frac{\log\left(\frac{p_{ij}}{q_{ij}}\right)}{\omega_{ij}} & \text{if } x_s = a_i \\ 0 & \text{o.w.} \end{cases}, \tag{70}$$

$$T(y) = \left(g_{s,i,j}\right), 1 \le s \le S, 1 \le i \le k, 1 \le j \le l, \tag{71}$$

$$g_{s,i,j} = \begin{cases} \omega_{ij} & \text{if } y_s = b_j \\ 0 & \text{o.w.} \end{cases}. \tag{72}$$

Then, we have $\log\left(\frac{\mathbb{P}(x,y)}{\mathbb{Q}(x,y)}\right) =< T(x), T(y) >$ where $< ., . >$ stands for the inner product in $\mathbb{R}^{klS}$. In other words, given any $x = (x_1, \dots, x_S)$, each x_s is transformed into a $kl \times 1$ vector with l non-zero elements. Similarly, given any $y = (y_1, \dots, y_S)$, each y_s is transformed to a $kl \times 1$ vector with k non-zero elements. Therefore, finding pairs of data points with large $\frac{\mathbb{P}(x,y)}{\mathbb{Q}(x,y)}$ is equivalent to finding transformed data points with large dot product. Using this transformation, in "Appendix 7" we show that the angle between both the true pairs and false pairs will be nearly $\frac{\pi}{2}$ for almost all the probability distributions [$\frac{S_0}{M^2} \approx 0$, using the notation from Shrivastava and Li (2014)]. It is well known that MIPS performs poorly in detection of pairs that are nearly orthogonal (Shrivastava and Li 2014). Therefore, solving (2) by transforming it to a MIPS problem and using existing approaches fails. Note that, Shrivastava and Li (2014) is based on data independent hashes for Euclidean distance that are not the state of the art. Recently, data dependent hashes have been introduced for Euclidean distance that improve on their data independent counterparts (Andoni et al. 2017; Rubinstein 2018), while currently there is no data dependent strategy for maximum inner product search. Therefore, MIPS is currently unable to solve (1) efficiently.

Experiment 2 In this experiment, we compared the complexity for the three algorithms LSH-hamming, MinHash and ForestDSH for a range of probability distributions. We benchmark the three methods using matrices $P(t) = P_1(1-t) + P_2 t$ where $0 \le t \le 1$, $P_1 = \begin{bmatrix} 0.345 & 0 \\ 0.31 & 0.345 \end{bmatrix}$, $P_2 = \begin{bmatrix} 0.019625 & 0 \\ 0.036875 & 0.9435 \end{bmatrix}$, and $\delta = 1$, i.e., $M = N$. The selection of P_1 was such that the complexity of MinHash minus the complexity of LSH-hamming was maximized. P_2 was selected such that the complexity of LSH-hamming minus the complexity of MinHash was maximized. Figure 6a shows the theoretical complexities of MinHash, LSH-hamming and ForestDSH for each matrix. See "Appendix 8", for the details on the derivation of complexities for MinHash, LSH-hamming and ForestDSH. For instance, for $P_1 = \begin{bmatrix} 0.345 & 0 \\ 0.31 & 0.345 \end{bmatrix}$, the theoretical per query complexities of MinHash, LSH-hamming and ForestDSH are equal to 0.5207, 0.4672 and 0.4384, respectively. We further consider N data points of dimension S, $\{x^1, \dots, x^N\}$ and $\{y^1, \dots, y^N\}$ where each (x^i, y^i) is generated from $P(t)$, and x^i is independent from y^j for $i \ne j$ ($N = 2000$, $S = 2000$). Then, we used ForestDSH, LSH-hamming and MinHash to find the matched pairs. In each case, we tuned $\#rows$[6] and $\#bands$ to achieve 99% true positive (recall) rate. Total simulation time for each of the three methods is plotted for each probability distribution in Fig. 6b. The simulation times in Fig. 6b are consistent with the theoretical guarantees in Fig. 6a. Figure 6b, c show that for sparse matrices, ($t \approx 1$), MinHash and ForestDSH outperform LSH-hamming. In denser cases, ($t \approx 0$), LSH-hamming and ForestDSH outperform MinHash. For ($t \le 0.4$), ForestDSH outperforms both MinHash and LSH-hamming. Note that, for the case when the data is coming from sparse distribution, MinHash beats ForestDSH in practice (as opposed to Theory).

[6] In MinHash and LSH-Hamming, we start with $\#bands \times \#rows$ randomly selected hashes from the family of distribution sensitive hashes, where $\#rows$ is the number of rows and $\#bands$ is the number of bands. We recall a pair (x, y), if x and y are hashed to the same value in all $\#rows$ rows and in at least one of the $\#bands$ bands.

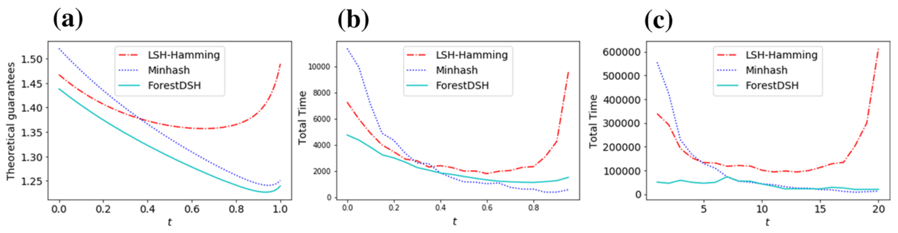


Fig. 6 Total (opposed to per query) complexities of LSH-hamming, MinHash, and ForestDSH are plotted for all the probability distribution matrices $P(t) = P_1(1-t) + P_2 t$ where $0 \leq t \leq 1$. **a** Theoretical guarantees, **b** simulation time for $N = 2000$ and $S = 2000$, **c** simulation time for $N = 20{,}000$ and $S = 2000$. Note that ForestDSH performs faster than MinHash if the data is not sparse. While for $N = 2000$ MinHash is superior to ForestDSH on sparse data, when N = 20,000, ForestDSH is performing the same as MinHash. This is mainly due to the fact that constant and $\log N$ terms vanish compared to N as N grows

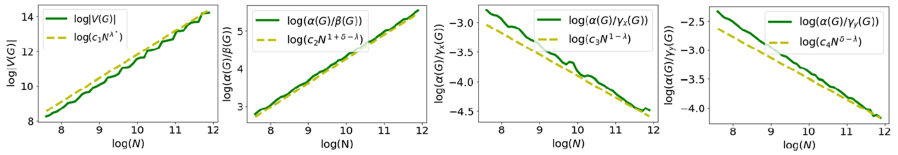


Fig. 7 In this figure, $V(G(N))$, $\frac{\alpha(G(N))}{\beta(G(N))}$, $\frac{\alpha(G(N))}{\gamma^{\mathcal{A}}(G(N))}$ and $\frac{\alpha(G(N))}{\gamma^{\mathcal{B}}(G(N))}$ are depicted as a function of N and constants c_1, c_2, c_3 and c_4 not depending on N. The parameters are chosen from Experiment 2. This figure shows how these functions grow/decay proportional to N^{λ^*}, $N^{1+\delta-\lambda^*}$, $N^{1-\lambda^*}$ and $N^{\delta-\lambda^*}$, respectively. (See proof of Theorem 3 in Sect. 6.2 for the justifications. This confirms [(34)–(38)] for trees $G(N)$ constructed by Algorithm 4)

This is because for sparse data the total complexity tends to its minimum, i.e., $O(N)$. ForestDSH is inefficient compared to MinHash for small number of N and when the data is sparse as the constant terms and $logN$ terms which play a significant role in this case are not optimized in ForestDSH. In Fig. 7, we further plotted $V(G(N))$, $\frac{\alpha(G(N))}{\beta(G(N))}$, $\frac{\alpha(G(N))}{\gamma^{\mathcal{A}}(G(N))}$ and $\frac{\alpha(G(N))}{\gamma^{\mathcal{B}}(G(N))}$ as a function of N for trees $G(N)$ constructed by Algorithm 4 for $P(t = 0.25)$, where $M = N$. As predicted by Theorem 3, we observed that these quantities grow/decay proportional to N^{λ^*}, $N^{1+\delta-\lambda^*}$, $N^{1-\lambda^*}$ and $N^{\delta-\lambda^*}$, respectively. In Fig. 8, true positive rate and total complexity are plotted in terms of $\#bands$ for the case of $N = 20{,}000$, $S = 10{,}000$ and the probability distribution P_2. From (50), i.e., $TP = 1 - (1 - \alpha(G))^{\#bands}$, we expect true positive rate to increase while $\#bands$ increases (see Fig. 8a). On the other hand, from (51) and (33), total complexity (N^{λ^*}) is expected to linearly increase while $\#bands$ increases (see Fig. 8b).

Experiment 3 In this experiment, we examine the robustness of our algorithm from Theorem 4 for matrices considered in Experiment 2 while $M = N = 300{,}000$. For each $P(t)$, we generated random matrices $P'(t)$ within $1+\epsilon$ of $P(t)$, see Theorem 4. We derive the practical complexity and theoretical complexity, i.e., λ^* from Definition 4. Moreover, for each of these matrices, we derive the worst case complexity in the case of $\epsilon = 0.03$. This is sketched in Fig. 9.

Experiment 4 In this experiment, we used the mass spectral data from human microbiome (McDonald et al. 2018), extracted using MSCluster (Frank et al. 2011). In

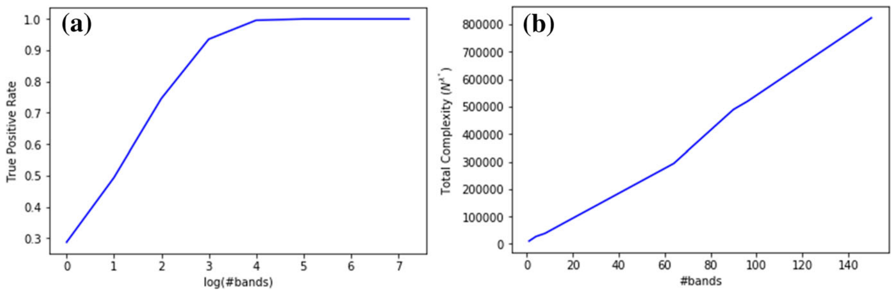
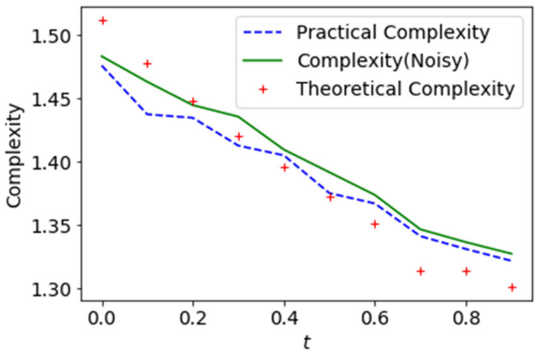


Fig. 8 **a** True positive rate and **b** total complexity (N^{λ^*}) are plotted for the probability distribution matrix $P_2(t)$, $N = 2000$ and $S = 2000$ in terms of #*bands*

Fig. 9 In this figure, practical, theoretical and noisy complexity are compared for $P(t) = P_1(1 - t) + P_2 t$ given in Experiment 4. Here, $M = N = 300{,}000$ and the noisy complexity is derived from Theorem 4 with $\epsilon = 0.03$



this problem, there are N spectra $X = x^1, \ldots, x^N$, and a query spectrum y. Our goal is to find the spectra x^i that maximize a probabilistic model $\mathbb{P}(y|x^i)$. $\mathbb{P}(y|x)$ is learned assuming that it can be factorized to i.i.d. components. To learn $p(y \mid x)$, each mass spectra is sorted based on the peak intensities, and in order to reduce the number of parameters we need to learn, instead of the peak ranks we use log of the peak ranks.[7] Using these data, we learn the joint probability distribution $p(\log Rank(y_s) = i \mid \log Rank(x_s) = j)$. Among 90,753 data points, we used 70,753 for training, and 20,000 for the test. The joint probability distribution of matching pairs are shown in Fig. 10, before and after logRank transformation.

After learning this probabilistic model, we applied ForestDSH method to the test mass spectra (after log $Rank$ transformation), and we were able to speed up the search of 20000 mass spectra nine times, in comparison to brute force search while maintaining true positive rate of 90%. For ForestDSH method, the total runtime is $705, 197ms$, while for brute force, the total runtime is 6,022,373ms (eight times slower than ForestDSH), and for LSH, the runtime for achieving 90% TP rate is 5,495,518ms (seven times slower than ForestDSH). The amount of memory used peaks at 220MB.

[7] $\log_b Rank$ of a peak is defined as the $\log_b$ of its $rank$. For any natural number n, $\log_b Rank = n$ for the peaks at rank $\{b^{n-1}, \ldots, b^n - 1\}$, e.g., $\log_2 Rank(m) = 3$ for $m \in \{4, 5, 6, 7\}$. Joint probability distribution of logRanks for the data from Frank et al. (2011) is shown in Fig. 10a (4×4 data matrix is obtained using $\log_4 Rank$), b (8×8 data matrix is obtained using $\log_2 Rank$) and c (51×51 data matrix is obtained not using any $\log_b Rank$).

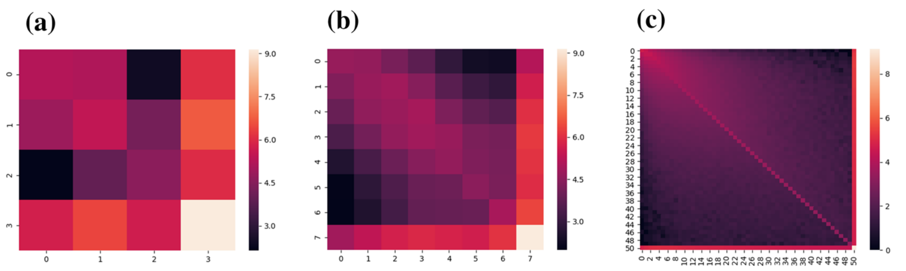


Fig. 10 Mass spectrometry joint probability distribution in the case of **a** $\log_4$ *Rank*, **b** $\log_2$ *Rank*, and **c** no log *Rank* filter. For further details on mass spectrometry joint probability distributions, see "Appendix 9"

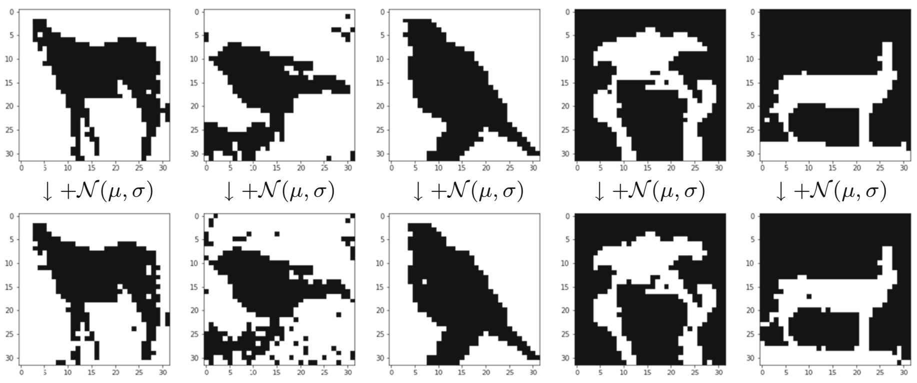


Fig. 11 In this figure, five grey-scale 32 × 32 pixels images along with their noisy versions are shown. Here, ForestDSH is able to detect true pairs with the success rate of 99% while being nine times faster than brute force and seven times faster than LSH

Experiment 5 In this experiment, we considered a set of 50,000 images (25,000 pairs of images) from CIFAR-10 database (Krizhevsky 2009). Each pair of images consists of a grey-scale 32 × 32 pixels image and a noisy version of the image constructed by adding independent Gaussian noise $\mathcal{N}(\mu = 0.5, \sigma = 0.039)$ and discretizing pixels to binary. ForestDSH is able to detect true pairs with the success rate of 99% in 1970 s while brute force approach detects true pairs in 19,123 s (nine times slower than ForestDSH) and LSH detects true pairs within 15,080 s (seven times slower than ForestDSH).

7.1 Codes

For the codes, see https://github.com/mohimanilab/ForestDSH.

8 Conclusion

ForestDSH algorithm proposed in this paper is comprehensive and efficient in the sense that for a wide range of probability distributions it enables us to capture the difference between pairs coming from a joint probability distributions and independent pairs. This algorithm is built upon a family of distribution sensitive hashes that are designed using a decision tree structure which is constructed recursively. Moreover, we prove that the decision tree introduced here has a complexity of $O(N^{\lambda^*})$ and there is no decision tree with lower overall complexity. We prove that this algorithm outperforms existing state of the art approaches in specific range of distributions, in theory and practice and enabled speeding up the spectral library search in mass spectrometry by a factor of nine. Finally, we should note that ForestDSH has some limitations which are discussed as follows. ForestDSH assumes that the probability distribution function is factorizable to i.i.d. components. Generalizing ForestDSH to the case of general probability distribution function is an open problem and our results here open a path towards solving the general probability mass function problem. Moreover, the special case of factorizable models are simple but crucial models that have been widely used in computational biology, e.g., see Kim and Pevzner (2014), and other areas of data sciences. In the future, we will address the more general problem of Markov chain models. Note that ForestDSH performs faster than MinHash if the data is not sparse. While for a small number of datapoints MinHash is superior to ForestDSH on sparse data, for a larger number of datapoints ForestDSH is performing the same as MinHash. This is mainly due to the fact that constant and logarithmic terms vanish in ratio as the number of datapoints grows.

Appendix 1: Proof of Lemmas 1 and 2

Appendix 1.1: Proof of Lemma 1

Lemma 1 is proved by induction on the depth of the node v. For example, consider v and its children $w_{ij} = f(v, a_i, b_j)$. From (21), we have $\Phi(w_{ij}) = \Phi(v)p_{ij}$. Therefore, (25) is proved by induction as follows. Assume (25) holds for any node with depth less than d. Consider the node w_{ij} which is a child of v, i.e., $w_{ij} = f(v, a_i, b_j)$ and $depth(w_{ij}) = d$.

$$
\begin{aligned}
\Phi(w_{ij}) &= \Phi(v)p_{ij} && (73)\\
&= Prob(v \in H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid (x, y) \sim \mathbb{P})p_{ij} && (74)\\
&= Prob\big(Seq^{\mathcal{A}}(v) \text{ is a prefix of } \mathrm{perm}_z(x),\\
&\quad Seq^{\mathcal{B}}(v) \text{ is a prefix of } \mathrm{perm}_z(y) \mid (x, y) \sim \mathbb{P}\big)p_{ij} && (75)\\
&= Prob\big(Seq^{\mathcal{A}}(v) \text{ is a prefix of } \mathrm{perm}_z(x),\\
&\quad Seq^{\mathcal{B}}(v) \text{ is a prefix of } \mathrm{perm}_z(y), a_i = (perm_z(x))_d, b_j\\
&= (perm_z(y))_d \mid (x, y) \sim \mathbb{P}\big)\\
& && (76)\\
&= Prob(w_{ij} \in H_z^{\mathcal{A}}(x) \cap H_z^{\mathcal{B}}(y) \mid (x, y) \sim \mathbb{P}). && (77)
\end{aligned}
$$

See Sect. 4 for the Definition of $perm_z$. Note that $(perm_z(x))_d$ stands for d-th entry of the vector $(perm_z(x))$. (74) follows from induction assumption for the nodes with depth less than d, (76) is a result of i.i.d. assumption (2), i.e., $\mathbb{P}(y \mid x) = \prod_{s=1}^{S} p(y_s \mid x_s)$ and the defnition of $H_z^{\mathcal{A}}(x)$ in (19). [(26)–(28)] follow similarly.

Appendix 1.2: Proof of Lemma 2

Using [(25)–(28)], constraints [(11)–(14)] hold for $\alpha = \alpha(G)$, $\beta = \beta(G)$, $\gamma^{\mathcal{A}} = \gamma^{\mathcal{A}}(G)$ and $\gamma^{\mathcal{B}} = \gamma^{\mathcal{B}}(G)$. Since no two buckets are ancestor/descendant of each other, we have

$$\mid H^{\mathcal{A}}(x) \cap H^{\mathcal{B}}(y) \mid \leq 1. \tag{78}$$

Therefore, (15) holds. This completes the proof that $H_z^{\mathcal{A}}(x)$ and $H_z^{\mathcal{B}}(y)$ defined in (19) and (20) are $(\alpha(G), \beta(G), \gamma^{\mathcal{A}}(G), \gamma^{\mathcal{B}}(G))$-sensitive.

Appendix 2: Deriving μ^*, ν^*, η^*, λ^*, p_0, q_0 and δ for $\mathbb{P}$, M and N

In this section, an algorithm for deriving μ^*, ν^*, η^*, λ^*, p_0, q_0 and δ for $\mathbb{P}$, M and N for the probability distribution $\mathbb{P}$ and δ is presented for a given probability distribution $\mathbb{P}$.

Algorithm 5 Deriving μ^*, ν^*, η^*, λ^*, p_0, q_0 and δ for $\mathbb{P}$, M and N

```
Inputs: The probability distribution $\mathbb{P}$, $list_\mu$, $list_\nu$, $list_\eta$, threshold $T$, $M$ and $N$.
Outputs: $\mu^*$, $\nu^*$, $\eta^*$, $\lambda^*$, $p_0$, $q_0$ and $\delta$.
Procedure:
  $\delta \leftarrow \frac{\log M}{\log N}$
  $\lambda^* = 0$
  For $\mu \in list_\mu$   # For some set of $list_\mu$, e.g., $\{0, 0.1, 0.2, \dots, 10\}$.
    For $\nu \in list_\nu$   # For some set of $list_\nu$, e.g., $\{0, 0.1, 0.2, \dots, 10\}$.
      For $\eta \in list_\eta$   # For some set of $list_\eta$, e.g., $\{0, 0.1, 0.2, \dots, 10\}$.
        If $|\sum_{i,j} p_{ij}^{1+\mu+\nu-\eta} (p_i^{\mathcal{A}})^{-\mu} (p_j^{\mathcal{B}})^{-\nu} - 1| \leq T$ and $\frac{\max(1,\delta)+\mu+\delta\nu}{1+\mu+\nu-\eta} > \lambda^*$.
          $\mu^* \leftarrow \mu$, $\nu^* \leftarrow \nu$, $\eta^* \leftarrow \eta$, $\lambda^* \leftarrow \frac{\max(1,\delta)+\mu+\delta\nu}{1+\mu+\nu-\eta}$
  #$p_0$, $q_0$ are computed as follows.
  $p_0 \leftarrow 1$, $q_0 \leftarrow 1$, $p_0^{\mathcal{A}} \leftarrow 1$, $p_0^{\mathcal{B}} \leftarrow 1$
  For $a_i \in \mathcal{A}$
    For $b_j \in \mathcal{B}$
      $p_0 \leftarrow p_0 p_{ij}$
      $q_0 \leftarrow q_0 q_{ij}$
      $p_0^{\mathcal{A}} \leftarrow p_0^{\mathcal{A}} p_i^{\mathcal{A}}$
      $p_0^{\mathcal{B}} \leftarrow p_0^{\mathcal{B}} p_j^{\mathcal{B}}$
  $q_0 \leftarrow \min(q_0, p_0^{\mathcal{A}}, p_0^{\mathcal{B}})$.
```

Remark 8 In Algorithm 5, the parameters μ^*, ν^*, η^* and λ^* could be derived from newton method too.

Appendix 3: Proof of Theorem 2

In order to prove Theorem 2, we first state the following two lemmas.

Lemma 3 *The function* $f(\theta, \theta_1, \theta_2, \theta_3) = \theta^{1+\rho_1+\rho_2+\rho_3} {\theta_1}^{-\rho_1} {\theta_2}^{-\rho_2} {\theta_3}^{-\rho_3}$ *is a convex function on the region* $(\theta, \theta_1, \theta_2, \theta_3) \in \mathbb{R}^{4+}$ *where* $(\rho_1, \rho_2, \rho_3) \in \mathbb{R}^{3+}$.[8]

Lemma 4 $\sum_{v \in V_{Buckets}(G)} \big(\Phi(v)\big)^{1+\mu+\nu-\eta} \big(\Psi^{\mathcal{A}}(v)\big)^{-\mu+\eta} \big(\Psi^{\mathcal{B}}(v)\big)^{-\nu+\eta} \big(\Psi(v)\big)^{-\eta} \leq 1$ *for any* $(\mu, \nu, \eta) \in \mathcal{I}$.

Proof of Lemma 3 and Lemma 4, are relegated to "Appendices 3.1 and 3.2", respectively. Consider (μ^*, ν^*, η^*) that satisfy (53). For any decision tree satisfying (34)–(38), we have:

$$\begin{aligned}
&\left(\frac{\sum_{v \in V_{Buckets}(G)} \Phi(v)}{|V_{Buckets}(G)|}\right)^{1+\mu^*+\nu^*-\eta^*} \left(\frac{\sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{A}}(v)}{|V_{Buckets}(G)|}\right)^{-\mu^*+\eta^*} \\
&\times \left(\frac{\sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{B}}(v)}{|V_{Buckets}(G)|}\right)^{-\nu^*+\eta^*} \left(\frac{\sum_{v \in V_{Buckets}(G)} \Psi(v)}{|V_{Buckets}(G)|}\right)^{-\eta^*} \\
&\quad \leq \frac{\sum_{v \in V_{Buckets}(G)} \big(\Phi(v)\big)^{1+\mu^*+\nu^*-\eta^*} \big(\Psi^{\mathcal{A}}(v)\big)^{-\mu^*+\eta^*} \big(\Psi^{\mathcal{B}}(v)\big)^{-\nu^*+\eta^*} \big(\Psi(v)\big)^{-\eta^*}}{|V_{Buckets}(G)|} \qquad (79) \\
&\quad \leq \frac{1}{|V_{Buckets}(G)|}, \qquad (80)
\end{aligned}$$

where (79) holds due to the convexity of $f(\theta, \theta_1, \theta_2, \theta_3) = \theta^{1+\rho_1+\rho_2+\rho_3} {\theta_1}^{-\rho_1} {\theta_2}^{-\rho_2}$ ${\theta_3}^{-\rho_3}$ in Lemma 3 and (80) follows from Lemma 4. Therefore, we have

$$\begin{aligned}
&\left(\sum_{v \in V_{Buckets}(G)} \Phi(v)\right)^{1+\mu^*+\nu^*-\eta^*} \left(\sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{A}}(v)\right)^{-\mu^*+\eta^*} \\
&\times \left(\sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{B}}(v)\right)^{-\nu^*+\eta^*} \left(\sum_{v \in V_{Buckets}(G)} \Psi(v)\right)^{-\eta^*} \\
&\leq 1. \qquad (81)
\end{aligned}$$

On the other hand, using (80) and the definitions of $\alpha(G)$, $\beta(G)$, $\gamma^{\mathcal{A}}(G)$ and $\gamma^{\mathcal{B}}(G)$ in [(29)–(32)], we have

$$\big(\alpha(G)\big)^{1+\mu^*+\nu^*-\eta^*} \big(\gamma^{\mathcal{A}}(G)\big)^{-\mu^*+\eta^*} \big(\gamma^{\mathcal{B}}(G)\big)^{-\nu^*+\eta^*} \big(\beta(G)\big)^{-\eta^*} \leq 1. \qquad (82)$$

Therefore, from (34)–(38) and (82) we have

1

[8] For any natural number n, $\mathbb{R}^{n+}$ denotes as the set of all n-tuples non-negative real numbers.

$$
\begin{aligned}
&\geq \left(\alpha(G)\right)^{1+\mu^*+\nu^*-\eta^*}\left(\gamma^{\mathcal{A}}(G)\right)^{-\mu^*+\eta^*}\left(\gamma^{\mathcal{B}}(G)\right)^{-\nu^*+\eta^*}\left(\beta(G)\right)^{-\eta^*} \\
&= \alpha(G)\left(\frac{\alpha(G)}{\gamma^{\mathcal{A}}(G)}\right)^{\mu^*-\eta^*}\left(\frac{\alpha(G)}{\gamma^{\mathcal{B}}(G)}\right)^{\nu^*-\eta^*}\left(\frac{\alpha(G)}{\beta(G)}\right)^{\eta^*} && (83) \\
&\geq \left(N^{\max(1,\delta)-\lambda}\right)\left(N^{1-\lambda}\right)^{\mu^*-\eta^*}\left(N^{\delta-\lambda}\right)^{\nu^*-\eta^*}\left(N^{1+\delta-\lambda}\right)^{\eta^*} && (84) \\
&= N^{\max(1,\delta)+\mu^*+\delta\nu^*-(1+\mu^*+\nu^*-\eta^*)\lambda}. && (85)
\end{aligned}
$$

Therefore, we conclude Theorem 2 as follows

$$\lambda \geq \lambda^* = \frac{\max(1,\delta)+\mu^*+\delta\nu^*}{1+\mu^*+\nu^*-\eta^*}. \tag{86}$$

Appendix 3.1: Proof of Lemma 3

The Hessian matrix for $f(\theta,\theta_1,\theta_2,\theta_3)$ is represented as

$$
\begin{aligned}
&H(\theta,\theta_1,\theta_2,\theta_3) \\
&\quad= f(\theta,\theta_1,\theta_2,\theta_3) \\
&\quad\times\begin{bmatrix}
\frac{(1+\rho_1+\rho_2+\rho_3)(\rho_1+\rho_2+\rho_3)}{\theta^2} & \frac{-\rho_1(1+\rho_1+\rho_2+\rho_3)}{\theta\theta_1} & \frac{-\rho_2(1+\rho_1+\rho_2+\rho_3)}{\theta\theta_2} & \frac{-\rho_3(1+\rho_1+\rho_2+\rho_3)}{\theta\theta_3} \\
\frac{-\rho_1(1+\rho_1+\rho_2+\rho_3)}{\theta\theta_1} & \frac{\rho_1(\rho_1+1)}{\theta_1^2} & \frac{\rho_1\rho_2}{\theta_1\theta_2} & \frac{\rho_1\rho_3}{\theta_1\theta_3} \\
\frac{-\rho_2(1+\rho_1+\rho_2+\rho_3)}{\theta\theta_2} & \frac{\rho_1\rho_2}{\theta_1\theta_2} & \frac{\rho_2(\rho_2+1)}{\theta_2^2} & \frac{\rho_2\rho_3}{\theta_2\theta_3} \\
\frac{-\rho_3(1+\rho_1+\rho_2+\rho_3)}{\theta\theta_3} & \frac{\rho_1\rho_3}{\theta_1\theta_3} & \frac{\rho_2\rho_3}{\theta_2\theta_3} & \frac{\rho_3(\rho_3+1)}{\theta_3^2}
\end{bmatrix} \\
&\quad= f(\theta,\theta_1,\theta_2,\theta_3) \\
&\quad\times\begin{bmatrix}
W^2+\frac{\rho_1+\rho_2+\rho_3}{\theta^2} & WW_1-\frac{\rho_1}{\theta\theta_1} & WW_2-\frac{\rho_2}{\theta\theta_2} & WW_3-\frac{\rho_3}{\theta\theta_3} \\
W_1W-\frac{\rho_1}{\theta\theta_1} & W_1^2+\frac{\rho_1}{\theta_1^2} & W_1W_2 & W_1W_3 \\
W_2W-\frac{\rho_2}{\theta\theta_2} & W_2W_1 & W_2^2+\frac{\rho_2}{\theta_2^2} & W_2W_3 \\
W_3W-\frac{\rho_3}{\theta\theta_3} & W_3W_1 & W_3W_2 & W_3^2+\frac{\rho_3}{\theta_3^2}
\end{bmatrix},
\end{aligned}
\tag{87}
$$

where $W = \frac{\rho_1+\rho_2+\rho_3}{\theta}$, $W_i = \frac{\rho_i}{\theta_i}$ for any $i \in \{1,2,3\}$. In order to show that the function $f(\theta,\theta_1,\theta_2,\theta_3)$ is a convex function it is necessary and sufficient to prove that $H(\theta,\theta_1,\theta_2,\theta_3)$ is positive semidefinite on $\mathbb{R}^{4+}$ On the other hand, for positive semidefinite matrices we have

1. For any non-negative scalar a and positive semidefinite matrix M, aM is positive semidefinite.
2. For positive semidefinite matrices M_1 and M_2, $M_1 + M_2$ is positive semidefinite.

As $f(\theta,\theta_1,\theta_2,\theta_3) > 0$ for any $\theta,\theta_1,\theta_2,\theta_3$, it is sufficient to prove that $\frac{H(\theta,\theta_1,\theta_2,\theta_3)}{f(\theta,\theta_1,\theta_2,\theta_3)}$ is positive semidefinite. Define, $M_1 = \begin{bmatrix} W^2 & WW_1 & WW_2 & WW_3 \\ W_1W & W_1^2 & W_1W_2 & W_1W_3 \\ W_2W & W_2W_1 & W_2^2 & W_2W_3 \\ W_3W & W_3W_1 & W_3W_2 & W_3^2 \end{bmatrix}$ and

$M_2 = \begin{bmatrix} \frac{\rho_1+\rho_2+\rho_3}{\theta^2} & -\frac{\rho_1}{\theta\theta_1} & -\frac{\rho_2}{\theta\theta_2} & -\frac{\rho_3}{\theta\theta_3} \\ -\frac{\rho_1}{\theta\theta_1} & \frac{\rho_1}{\theta_1^2} & 0 & 0 \\ -\frac{\rho_2}{\theta\theta_2} & 0 & \frac{\rho_2}{\theta_2^2} & 0 \\ -\frac{\rho_3}{\theta\theta_3} & 0 & 0 & \frac{\rho_3}{\theta_3^2} \end{bmatrix}$. Therefore, we have $M_1 + M_2 = \frac{H(\theta,\theta_1,\theta_2,\theta_3)}{f(\theta,\theta_1,\theta_2,\theta_3)}$.

In order to prove that $f(\theta,\theta_1,\theta_2,\theta_3)$ is positive semidefinite, it is sufficient to prove that M_1 and M_2 are positive semidefinites. The matrices M_1 and M_2 are positive semidefinite as for any non-zero vector $z = \begin{bmatrix} a \; b \; c \; d \end{bmatrix}$, we have $zM_1z^T \geq 0$ and $zM_2z^T \geq 0$, i.e.,

$$\begin{aligned} &\begin{bmatrix} a \; b \; c \; d \end{bmatrix} \begin{bmatrix} W^2 & WW_1 & WW_2 & WW_3 \\ W_1W & W_1^2 & W_1W_2 & W_1W_3 \\ W_2W & W_2W_1 & W_2^2 & W_2W_3 \\ W_3W & W_3W_1 & W_3W_2 & W_3^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \\ &\quad = (Wa + W_1b + W_2c + W_3d)^2 \qquad (88) \\ &\geq 0, \qquad (89) \\ &\begin{bmatrix} a \; b \; c \; d \end{bmatrix} \begin{bmatrix} \frac{\rho_1+\rho_2+\rho_3}{\theta^2} & -\frac{\rho_1}{\theta\theta_1} & -\frac{\rho_2}{\theta\theta_2} & \frac{\rho_3}{\theta\theta_3} \\ -\frac{\rho_1}{\theta\theta_1} & \frac{\rho_1}{\theta_1^2} & 0 & 0 \\ -\frac{\rho_2}{\theta\theta_2} & 0 & \frac{\rho_2}{\theta_2^2} & 0 \\ -\frac{\rho_3}{\theta\theta_3} & 0 & 0 & \frac{\rho_3}{\theta_3^2} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \\ &\quad = \rho_1\Big(\frac{a}{\theta} - \frac{b}{\theta_1}\Big)^2 + \rho_2\Big(\frac{a}{\theta} - \frac{c}{\theta_2}\Big)^2 \\ &\qquad +\rho_3\Big(\frac{a}{\theta} - \frac{d}{\theta_3}\Big)^2 \qquad (90) \\ &\quad \geq 0, \qquad (91) \end{aligned}$$

where (91) is concluded as $\rho_1, \rho_2, \rho_3 \geq 0$.

Appendix 3.2: Proof of Lemma 4

First, note that $\Psi(v) = \Psi^{\mathcal{A}}(v)\Psi^{\mathcal{B}}(v)$. Let us define

$$D(v) = \sum_{v \in V_{Buckets}(G)} \Phi(v)^{1+\mu+\nu-\eta}\big(\Psi^{\mathcal{A}}(v)\big)^{-\mu}\big(\Psi^{\mathcal{B}}(v)\big)^{-\nu}. \qquad (92)$$

We show

$$\sum_{v \in V_{Buckets}(G)} D(v) \leq 1, \tag{93}$$

by induction on the number of nodes in the tree. If the tree has only one node, i.e., *root*, then (93) is holds as $\Phi(root) = 1$, $\Psi^{\mathcal{A}}(root) = 1$ and $\Psi^{\mathcal{B}}(root) = 1$ from the definition of $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$ and $\Psi^{\mathcal{B}}(v)$ in [(21)–(23)]. Assume that (93) holds for any decision tree with $|G| < Z$. Our goal is to prove that (93) holds for a decision tree with $|G| = Z$. Assume v_{11} is the node with maximum length in G and consider a tree G' constructed by removing v_{11} and all its siblings v_{ij}, $1 \leq i \leq k$, $1 \leq j \leq l$ belonging to the same parent v. In other words, for the tree G' we have[9]

$$V(G') = V(G)/\{v_{11}, \ldots, v_{kl}\}, \tag{94}$$

$$V_{Buckets}(G') = (V_{Buckets}(G) \cup \{v\})/\{v_{11}, \ldots, v_{kl}\}. \tag{95}$$

(95) is true as for the gragh G', the node v in now a leaf node while the nodes $v_{11}, \ldots, v_{kl}$ are removed. Then, we have

$$\begin{aligned}
&\sum_{v \in V_{Buckets}(G)} D(v) \\
&\leq \sum_{v \in V_{Buckets}(G')} D(v) - \Phi(v)^{1+\mu+\nu-\eta}\left(\Psi^{\mathcal{A}}(v)\right)^{-\mu}\left(\Psi^{\mathcal{B}}(v)\right)^{-\nu} \\
&\quad + \sum_{i,j} \Phi(v_{ij})^{1+\mu+\nu-\eta} \Psi^{\mathcal{A}}(v_{ij})^{-\mu} \Psi^{\mathcal{B}}(v_{ij})^{-\nu} &(96) \\
&= \sum_{\in V_{Buckets}(G')} D(v) - \Phi(v)^{1+\mu+\nu-\eta}\left(\Psi^{\mathcal{A}}(v)\right)^{-\mu}\left(\Psi^{\mathcal{B}}(v)\right)^{-\nu} \\
&\quad + \sum_{i,j} \Big(\Phi(v)^{1+\mu+\nu-\eta}\left(\Psi^{\mathcal{A}}(v)\right)^{-\mu+\eta}\left(\Psi^{\mathcal{B}}(v)\right)^{-\nu} \\
&\quad \times {p_{ij}}^{1+\mu+\nu-\eta}(p_i^{\mathcal{A}})^{-\mu}(p_i^{\mathcal{B}})^{-\nu}\Big) &(97) \\
&= \sum_{v \in V_{Buckets}(G')} D(v) - \Phi(v)^{1+\mu+\nu-\eta}\left(\Psi^{\mathcal{A}}(v)\right)^{-\mu}\left(\Psi^{\mathcal{B}}(v)\right)^{-\nu} \\
&\quad \times \left(1 - \sum_{i,j} {p_{ij}}^{1+\mu+\nu-\eta}(p_i^{\mathcal{A}})^{-\mu}(p_j^{\mathcal{B}})^{-\nu}\right) &(98) \\
&= \sum_{v \in V_{Buckets}(G')} D(v) &(99) \\
&= 1, &(100)
\end{aligned}$$

[9] Note that, $V_b(G')$ satisfies bucket-list property, e.g., there is no bucket in the tree that is ancestor of another bucket.

where (96) holds from the definition of tree G', (97) follows from the recursive definition of $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$ and $\Psi^{\mathcal{B}}(v)$ in [(21)–(23)] and (99) holds, note that from the definition of μ, ν and η in (52), i.e.,

$$\sum_{i,j} {p_{ij}}^{1+\mu+\nu-\eta} \left(p_i^{\mathcal{A}}\right)^{-\mu} \left(p_j^{\mathcal{B}}\right)^{-\nu} = 1. \tag{101}$$

Therefore, we conclude that

$$\sum_{v \in V_{Buckets}(G)} D(v) \leq 1. \tag{102}$$

Note that, the inequality (96) becomes an equality only in cases where none of the children are pruned.

Appendix 4: Proof of Theorem 3

In order to prove Theorem 3, we first present the following lemma.

Lemma 5 *Given a fixed $N \in \mathbb{N}$ and probability distribution $\mathbb{P}$, consider the following region R_λ*

$$\mathcal{R}(\lambda, r_{ij}, n) = \Bigg\{ \lambda, r_{ij}, n \text{ s.t. } \lambda \geq 0, \sum_{i,j} r_{ij} = 1, r_{ij} \geq 0, r_{ij} \in \mathbb{R}^+, n \in \mathbb{R}^+, \tag{103}$$

$$\sum_{i,j} r_{ij} \log p_{ij} - \sum_{i,j} r_{ij} \log r_{ij} \geq \frac{(\max(1,\delta) - \lambda) \log N}{n}, \tag{104}$$

$$\sum_{i,j} r_{ij} \log p_{ij} - \sum_{i,j} r_{ij} \log p_i^{\mathcal{A}} \geq \frac{(1 - \lambda) \log N}{n}, \tag{105}$$

$$\sum_{i,j} r_{ij} \log p_{ij} - \sum_{i,j} r_{ij} \log p_j^{\mathcal{B}} \geq \frac{(\delta - \lambda) \log N}{n}, \tag{106}$$

$$\sum_{i,j} r_{ij} \log p_{ij} - \sum_{i,j} r_{ij} \log q_{ij} \geq \frac{(1 + \delta - \lambda) \log N}{n} \Bigg\} \tag{107}$$

[10] *Then, $(\lambda^*, r_{ij}^*, n^*)$ defined in Definition 4 is a member of $\mathcal{R}(\lambda, r_{ij}, n)$.*

The proof of Lemma 5 is relegated to "Appendix 4.4". Let us prove that the following tree construction steps in Algorithm 4 result in a tree that satisfies (34)–(38).

[10] Recall that, for simplicity we use the notation $\sum_{i,j}$ and $\prod_{i,j}$ instead of $\sum_{1 \leq i \leq k, 1 \leq j \leq l}$ and $\prod_{1 \leq i \leq k, 1 \leq j \leq l}$, respectively.

$$\begin{cases} \frac{\Phi(v)}{\Psi(v)} \geq N^{1+\delta-\lambda^*} p_0 q_0 & : accept\ bucket, \\ \frac{\Phi(v)}{\Psi^{\mathcal{A}}(v)} \leq N^{1-\lambda^*} p_0 q_0 & : prune, \\ \frac{\Phi(v)}{\Psi^{\mathcal{B}}(v)} \leq N^{\delta-\lambda^*} p_0 q_0 & : prune, \\ \quad otherwise & : branch\ into\ the\ kl\ children. \end{cases} \tag{108}$$

Consider the set of $r^*_{ij} = p_{ij}^{1+\mu^*+\nu^*-\eta^*} (p_i^{\mathcal{A}})^{-\mu^*} (p_j^{\mathcal{B}})^{-\nu^*}$ and $n^* = \frac{(\max(1,\delta)-\lambda^*)\log N}{\sum r^*_{ij} \log \frac{p_{ij}}{r^*_{ij}}}$. Note that we assume p_{ij} and q_{ij} are non-zero.[11] Consider $n_{ij} = \lceil n^* r^*_{ij} \rceil$ if $r^*_{ij} > \frac{1}{2}$ and $n_{ij} = \lfloor n^* r^*_{ij} \rfloor$ if $r^*_{ij} \leq \frac{1}{2}$. Therefore, we have $n^* - kl < \sum_{ij} n_{ij} \leq n^*$. For any $v \in V(G)$, define the set $S_{ij}(v)$ as follows

$$S_{ij}(v) = \{s \mid 1 \leq s \leq depth(v), Seq_s^{\mathcal{A}}(v) = a_i \& Seq_s^{\mathcal{B}}(v) = b_j\}, \tag{109}$$

where $depth(v)$ is the depth of node v in the tree, $Seq_s^{\mathcal{A}}(v)$ and $Seq_s^{\mathcal{B}}(v)$ stand for the character at position s in the strings $Seq^{\mathcal{A}}(v)$ and $Seq^{\mathcal{B}}$(v), respectively. Now, consider a node v^* in the graph that satisfies the following constraints:

$$|S_{ij}(v)| = n_{ij}, \forall 1 \leq i \leq k, 1 \leq j \leq l. \tag{110}$$

The number of nodes v that satisfy this constraint is $\binom{n^*}{n_{11},\ldots,n_{kl}}$. Moreover, define

$$|V_{n_{11},\ldots,n_{kl}}| = \{v \in V(G) \mid |S_{ij}(v)| = n_{ij}, \forall 1 \leq i \leq k, 1 \leq j \leq l\}. \tag{111}$$

Appendix 4.1: Node *v** or one of its ancestors is designated as a bucket by Algorithm 4

Here, we prove that the node v or one of its ancestors is designated as a bucket by Algorithm 4. In order to show this, we need to prove that:

$$\frac{\Phi(v^*)}{\Psi(v^*)} \geq e^{\sum n^* r^*_{ij} \log p_{ij}} e^{-\sum n^* r^*_{ij} \log q_{ij}} p_0 q_0 \geq N^{1+\delta-\lambda^*} p_0 q_0, \tag{112}$$

$$\frac{\Phi(v^*)}{\Psi^{\mathcal{A}}(v^*)} \geq e^{\sum n^* r^*_{ij} \log p_{ij}} e^{-\sum n^* r^*_{ij} \log p_i^{\mathcal{A}}} p_0 q_0 \geq N^{1-\lambda^*} p_0 q_0, \tag{113}$$

$$\frac{\Phi(v^*)}{\Psi^{\mathcal{B}}(v^*)} \geq e^{\sum n^* r^*_{ij} \log p_{ij}} e^{-\sum n^* r^*_{ij} \log p_j^{\mathcal{B}}} p_0 q_0 \geq N^{\delta-\lambda^*} p_0 q_0, \tag{114}$$

where p_0 and q_0 are defined as $\prod_{i,j} p_{ij}$ and $\min(\prod_{i,j} q_{ij}, \prod_i (p_i^{\mathcal{A}})^l, \prod_j (p_j^{\mathcal{B}})^k)$. Note that, $\Phi(v^*)$, $\Psi(v^*)$, $\Psi^{\mathcal{A}}(v^*)$ and $\Psi^{\mathcal{B}}(v^*)$ are computed as follows

$$\Phi(v^*) = \prod_{i,j} p_{ij}^{n_{ij}} = e^{\sum_{i,j} n_{ij} \log p_{ij}} \geq e^{\sum_{i,j} (n^* r^*_{ij}+1) \log p_{ij}}$$

[11] Note that in the cases where q_{ij} is zero, then from the definition of q_{ij}, p_{ij} would also be equal to zero. Therefore, we will ignore those branches during the tree construction.

$$\geq e^{\sum_{i,j} n^* r^*_{ij} \log p_{ij}} (\prod_{i,j} p_{ij}) = e^{\sum_{i,j} n^* r^*_{ij} \log p_{ij}} p_0, \tag{115}$$

$$\begin{aligned}\Psi(v^*) &= \prod_{i,j} q_{ij}^{n_{ij}} = e^{\sum_{i,j} n_{ij} \log q_{ij}} \\ &\leq e^{\sum_{i,j} (n^* r^*_{ij} - 1) \log q_{ij}} \leq \frac{e^{\sum_{i,j} n^* r^*_{ij} \log q_{ij}}}{\prod_{i,j} q_{ij}} = \frac{e^{\sum_{i,j} n^* r^*_{ij} \log q_{ij}}}{q_0}, \end{aligned} \tag{116}$$

$$\begin{aligned}\Psi^{\mathcal{A}}(v^*) &= \prod_{i,j} (p_i^{\mathcal{A}})^{n_{ij}} = e^{\sum_{i,j} n_{ij} \log p_i^{\mathcal{A}}} \\ &\leq e^{\sum_{i,j} (n^* r^*_{ij} - 1) \log (p_i^{\mathcal{A}})} \leq \frac{e^{\sum_{i,j} n^* r^*_{ij} \log p_i^{\mathcal{A}}}}{\prod_{i,j} p_i^{\mathcal{A}}} = \frac{e^{\sum_{i,j} n^* r^*_{ij} \log p_i^{\mathcal{A}}}}{q_0}, \end{aligned} \tag{117}$$

$$\begin{aligned}\Psi^{\mathcal{B}}(v^*) &= \prod_{i,j} (p_j^{\mathcal{B}})^{n_{ij}} = e^{\sum_{i,j} n_{ij} \log p_j^{\mathcal{B}}} \\ &\leq e^{\sum_{i,j} (n^* r^*_{ij} - 1) \log p_j^{\mathcal{B}}} \leq \frac{e^{\sum_{i,j} n^* r^*_{ij} \log p_j^{\mathcal{B}}}}{\prod_{i,j} p_j^{\mathcal{B}}} = \frac{e^{\sum_{i,j} n^* r^*_{ij} \log p_j^{\mathcal{B}}}}{q_0}. \end{aligned} \tag{118}$$

Therefore, from Lemma 5 and [(115)–(118)] we conclude [(112)–(114)]. This means v^* or one of its ancestors is an accepted bucket.

Appendix 4.2: Proof of bounds [(35)–(38)]

First, we derive a lower bound on $\alpha(G)$ as follows.

$$\begin{aligned}\alpha(G) &= \sum_{v \in V_{Buckets}(G)} \Phi(v) \\ &\geq \sum_{v \in V_{n_{11},\ldots,n_{kl}}} \Phi(v) && (119) \\ &\geq |V_{n_{11},\ldots,n_{kl}}| \Phi(v) && (120) \\ &\geq |V_{n_{11},\ldots,n_{kl}}| e^{\sum_{i,j} n^* r^*_{ij} \log p_{ij}} p_0, && (121)\end{aligned}$$

where $V_{n_{11},\ldots,n_{kl}}$ is the set of nodes that satisfies (110). $|V_{n_{11},\ldots,n_{kl}}|$ is lower bounded as

$$\begin{aligned}|V_{n_{11},\ldots,n_{kl}}| &= \binom{n^*}{n_{11},\ldots,n_{kl}} \\ &\geq \frac{n^*!}{(kl)! \prod_{i,j} n_{ij}!} \geq \frac{(\frac{n^*}{e})^n \sqrt{2\pi n^*}}{(kl)! \prod_{i,j} (\frac{n_{ij}}{e})^{n_{ij}} \sqrt{2\pi n_{ij}} e} && (122) \\ &\geq \prod_{i,j} (\frac{n_{ij}}{n^*})^{-n_{ij}} (n^*)^{\frac{1-kl}{2}} \frac{(2\pi)^{\frac{1-kl}{2}} e^{-kl}}{(kl)!} && (123)\end{aligned}$$

$$= ce^{-\sum_{i,j} n_{ij} \log(\frac{n_{ij}}{n^*})} (n^*)^{\frac{1-kl}{2}} \tag{124}$$
$$\geq ce^{-n^* \sum_{i,j} r_{ij}^* \log r_{ij}^*}, \tag{125}$$

for some constant $c = \frac{(2\pi)^{\frac{1-kl}{2}} e^{-kl}}{(kl)!} (n^*)^{\frac{1-kl}{2}}$ depending on n, k and l. (122) is true as for any natural number m we have $\sqrt{2\pi m}(\frac{m}{e})^m \leq m! < \sqrt{2\pi m}(\frac{m}{e})^m e$. (125) follows as $a \log \frac{1}{a}$ is an increasing function for $0 \leq x \leq 0.5$, and a decreasing function for $0.5 \leq x \leq 1$. Therefore, from (121) and (125), $\alpha(G) \geq N^{\max(1,\delta)-\lambda^*}$ is concluded. Similarly, (34)–(38) are proved as follows.

$$\frac{\alpha(G)}{\beta(G)} = \frac{\sum_{v \in V_{Buckets}(G)} \Phi(v)}{\sum_{v \in V_{Buckets}(G)} \Psi(v)} \geq N^{1+\delta-\lambda^*} p_0 q_0, \tag{126}$$
$$\frac{\alpha(G)}{\gamma^{\mathcal{A}}(G)} = \frac{\sum_{v \in V_{Buckets}(G)} \Phi(v)}{\sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{A}}(v)} \geq N^{1-\lambda^*} p_0 q_0, \tag{127}$$
$$\frac{\alpha(G)}{\gamma^{\mathcal{B}}(G)} = \frac{\sum_{v \in V_{Buckets}(G)} \Phi(v)}{\sum_{v \in V_{Buckets}(G)} \Psi^{\mathcal{B}}(v)} \geq N^{\delta-\lambda^*} p_0 q_0, \tag{128}$$

where (126) and (128) are concluded from (112)–(114) and the fact that $\frac{\sum_i a_i}{\sum_i b_i} \geq c$ is true for any i if $\frac{a_i}{b_i} \geq c$ and $b_i > 0$.

Appendix 4.3: Bounding number of nodes in the tree, i.e., (34)

The number of nodes in the decision tree defined in (57), is bounded by $O(N^{\lambda^*})$ using the following three lemmas.

Lemma 6 *For any leaf node v of the decision tree defined in (57), we have*

$$\Phi(v) \geq N^{-\lambda^*} \min_{i,j} p_{ij} p_0 q_0. \tag{129}$$

Lemma 7 *For any tree G, the summation of $\Phi(v)$ over all the leaf nodes is equal to one, i.e., $\sum_{v \in V_l} \Phi(v) = 1$.*

Lemma 8 *The number of nodes in the decision tree defined in (57) is at most two times of the number of leaf nodes.*

For proof of Lemmas 6, 7 and 8 , see "Appendices 4.5, 4.6 and 4.7". Therefore, we have

$$1$$
$$= \sum_{v \in V_l(G)} \Phi(v) \tag{130}$$
$$\geq \sum_{v \in V(G)} N^{-\lambda^*} \min_{i,j} p_{ij} p_0 q_0 \tag{131}$$

$$= |V_l(G)|N^{-\lambda^*} \min_{i,j} p_{ij} p_0 q_0 \tag{132}$$

$$\geq \frac{|V(G)|}{2} N^{-\lambda^*} \min_{i,j} p_{ij} p_0 q_0, \tag{133}$$

where (130) follows from Lemma 7, (131) is true from (167) and (133) is concluded from Lemma 8. Therefore, we conclude that $|V(G)| = O(N^{\lambda^*})$.

Appendix 4.4: Proof of Lemma 5

Consider the optimization problem of finding the member of $(\lambda, r_{ij}, n) \in \mathcal{R}(\lambda, r_{ij}, n)$ with minimum λ. This optimization problem is a convex optimization problem.[12] Therefore, writing the KKT conditions, we have

$$\begin{aligned}
&F(r_{ij}, n, \lambda) \\
&= \lambda + \sum_{i,j} \mu_{1ij}(-r_{ij}) + \mu_2\Big(\frac{(\max(1,\delta)-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} \\
&\quad + \sum_{i,j} r_{ij}\log r_{ij}\Big) \\
&\quad + \mu_3\Big(\frac{(1-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log p_i^{\mathcal{A}}\Big) \\
&\quad + \mu_4\Big(\frac{(\delta-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log p_j^{\mathcal{B}}\Big) \\
&\quad + \mu_5\Big(\frac{(1+\delta-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log q_{ij}\Big) \\
&\quad + \mu_6\Big(\sum_{i,j} r_{ij} - 1\Big),
\end{aligned} \tag{137}$$

where

$$\mu_2, \mu_3, \mu_4, \mu_5, r_{ij} \geq 0,\ \mu_{1ij} r_{ij} = 0,\ \sum_{i,j} r_{ij} - 1 = 0, \tag{138}$$

[12] Note that $f_1(n) = \frac{1}{n}$, $f_2(r) = r\log r$ and $f_3(r) = ar$ are convex functions. Therefore, as the sum of the convex functions is a convex function, the optimization problem (103)–(107) is in the form of convex optimization problem

$$\underset{\lambda}{\text{Minimize}}\ \ f(x), \tag{134}$$

$$\text{subject to}\ \ \{g_i(x) \leq 0, i \in \{1, \ldots, m\}, \tag{135}$$

$$h_j(x) = 0, j \in \{1, \ldots, p\}\}, \tag{136}$$

where $x \in \mathbb{R}^n$, $f(x)$ and $g_i(x)$ are convex functions and $h_j(x)$ is affine functions.

$$\frac{(\max(1,\delta)-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log r_{ij} \leq 0, \quad (139)$$

$$\mu_2\left(\frac{(\max(1,\delta)-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log r_{ij}\right) = 0, \quad (140)$$

$$\frac{(1-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log p_i^{\mathcal{A}} \leq 0, \quad (141)$$

$$\mu_3\left(\frac{(1-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log p_i^{\mathcal{A}}\right) = 0, \quad (142)$$

$$\frac{(\delta-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log p_j^{\mathcal{B}} \leq 0, \quad (143)$$

$$\mu_4\left(\frac{(\delta-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log p_j^{\mathcal{B}}\right) = 0, \quad (144)$$

$$\frac{(1+\delta-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log q_{ij} \leq 0, \quad (145)$$

$$\mu_5\left(\frac{(1+\delta-\lambda)\log N}{n} - \sum_{i,j} r_{ij}\log p_{ij} + \sum_{i,j} r_{ij}\log q_{ij}\right) = 0. \quad (146)$$

From (138), μ_{1ij} is zero if r_{ij} is a non-zero number. Therefore, we only keep i and j where $r_{ij} \neq 0$ and $\mu_{1ij} = 0$.

$$\frac{dF(r_{ij},n,\lambda)}{dr_{ij}} = 0 \rightarrow \mu_2 + \mu_2\log r_{ij} + \mu_3\log p_i^{\mathcal{A}} + \mu_4\log p_j^{\mathcal{B}} + \mu_5\log q_{ij} + \mu_6$$
$$-(\mu_2+\mu_3+\mu_4+\mu_5)\log p_{ij} = 0 \quad (147)$$
$$\rightarrow r_{ij}^{\mu_2} = p_{ij}^{\mu_2+\mu_3+\mu_4+\mu_5}(p_i^{\mathcal{A}})^{-\mu_3}(p_j^{\mathcal{B}})^{-\mu_4} q_{ij}^{-\mu_5} e^{-\mu_2-\mu_6}. \quad (148)$$

Consider the following two cases.

1. $\mu_2 = 0$. In this case, all the constraints are affine functions and therefore we have a linear programming problem and the feasible set of this linear programming problem is a polyhedron. From (103), the polyhedron is bounded, i.e., $0 \leq r_{ij} \leq M$ for some constant M. Assume that the polyhedron is nonempty, otherwise the solution is ∞. Moreover, a nonempty bounded polyhedron cannot contain a line, thus it must have a basic feasible solution and the optimal solutions are restricted to the corner points.
2. $\mu_2 \neq 0$. As $r_{ij} \neq 0$ and $\mu_{1ij} = 0$, we have

$$\frac{dF(r_{ij},n,\lambda)}{dr_{ij}} = 0 \rightarrow \mu_2 + \mu_2\log r_{ij} + \mu_3\log p_i^{\mathcal{A}}$$

$$+\mu_4 \log p_j^{\mathcal{B}} + \mu_5 \log q_{ij} + \mu_6$$
$$-(\mu_2+\mu_3+\mu_4+\mu_5)\log p_{ij} = 0 \tag{149}$$
$$\to r_{ij}^{\mu_2} = p_{ij}^{\mu_2+\mu_3+\mu_4+\mu_5} (p_i^{\mathcal{A}})^{-\mu_3} (p_j^{\mathcal{B}})^{-\mu_4} q_{ij}^{-\mu_5} e^{-\mu_2-\mu_6} \tag{150}$$
$$\to r_{ij} = c p_{ij}^{\frac{\mu_2+\mu_3+\mu_4+\mu_5}{\mu_2}} (p_i^{\mathcal{A}})^{-\frac{\mu_3}{\mu_2}} (p_j^{\mathcal{B}})^{-\frac{\mu_4}{\mu_2}} q_{ij}^{-\frac{\mu_5}{\mu_2}}, \tag{151}$$
$$\frac{dF(r_{ij},n,\lambda)}{dn} = 0$$
$$\to -(\mu_2(\max(1,\delta)-\lambda) + \mu_3(1-\lambda) + \mu_4(\delta-\lambda)$$
$$+\mu_5(1+\delta-\lambda))\frac{\log N}{n^2} = 0 \tag{152}$$
$$\to \mu_2(\max(1,\delta)-\lambda) + \mu_3(1-\lambda)$$
$$+\mu_4(\delta-\lambda) + \mu_5(1+\delta-\lambda) = 0 \tag{153}$$
$$\to \lambda = \frac{\mu_2\max(1,\delta) + \mu_3 + \mu_4\delta + \mu_5(1+\delta)}{\mu_2+\mu_3+\mu_4+\mu_5}. \tag{154}$$

Summing (140), (142), (144) and (146) and using (151), we have

$$(\mu_2(\max(1,\delta)-\lambda) + \mu_3(1-\lambda) + \mu_4(\delta-\lambda) + \mu_5(1+\delta-\lambda))\frac{\log N}{n}$$
$$= \mu_2 \log c. \tag{155}$$

From (154) and $\mu_2 \neq 0$, we conclude that $c = 1$. Moreover, we have $\lambda = \frac{\max(1,\delta)+\mu+\nu\delta}{1+\mu+\nu+\eta}$, where μ, ν and η are defined as:

$$\mu = \frac{\mu_3+\mu_5}{\mu_2}, \tag{156}$$
$$\nu = \frac{\mu_4+\mu_5}{\mu_2}, \tag{157}$$
$$\eta = \frac{\mu_2'-\mu_5}{\mu_2}. \tag{158}$$

Assume that $\eta > 0$, therefore $r_{ij} < p_{ij}$ as $p_{ij} \le \min(q_i, q_j)$. On the other hand, $\sum_{i,j} r_{ij} = \sum p_{ij} = 1$ which contradicts our assumption that $\eta > 0$. Thus, $\eta \le 0$. Define (μ^*, ν^*, η^*) as follows

$$(\mu^*,\nu^*,\eta^*) = \arg \max_{\min(\mu,\nu)\ge\eta\ge 0, \sum_{i,j} p_{ij}^{1+\mu+\nu-\eta}(p_i^{\mathcal{A}})^{-\mu}(p_j^{\mathcal{B}})^{-\nu}=1} \frac{\max(1,\delta)+\mu+\nu\delta}{1+\mu+\nu-\eta}. \tag{159}$$

Therefore, from (140), (151) and (154) we conclude Lemma 5 as follows

$$r_{ij}^* = p_{ij}^{1+\mu^*+\nu^*-\eta^*}(p_i^{\mathcal{A}})^{-\mu^*}(p_j^{\mathcal{B}})^{-\nu^*}, \tag{160}$$

$$\lambda^* = \frac{\max(1,\delta) + \mu^* + \nu^*\delta}{1 + \mu^* + \nu^* - \eta^*}, \tag{161}$$

$$n^* = \frac{(\max(1,\delta) - \lambda^*)\log N}{\sum r_{ij}^* \log \frac{p_{ij}}{r_{ij}^*}}. \tag{162}$$

Appendix 4.5: Proof of Lemma 6

In order to prove Lemma 6, consider a leaf node v_0 and its parent v_1. From (57), for the node v_1 we have

$$\frac{\Phi(v_1)}{\Psi(v_1)} \leq N^{1+\delta-\lambda^*} p_0 q_0, \tag{163}$$

$$\frac{\Phi(v_1)}{\Psi^{\mathcal{A}}(v_1)} \geq N^{1-\lambda^*} p_0 q_0, \tag{164}$$

$$\frac{\Phi(v_1)}{\Psi^{\mathcal{B}}(v_1)} \geq N^{\delta-\lambda^*} p_0 q_0. \tag{165}$$

(163)–(165) follow from (57) and the fact that the node v_1 is niether pruned nor accepted as it is not a leaf node. Therefore, from (164)×(165)/(163), we conclude that,

$$\Phi(v_1) \geq N^{-\lambda^*} p_0 q_0. \tag{166}$$

Therefore, from definition of $\Phi(v)$, for the leaf node v_0 we have

$$\Phi(v_0) \geq N^{-\lambda^*} \min_{i,j} p_{ij} p_0 q_0. \tag{167}$$

(167) is true for all the leaf nodes as (163)–(167) were derived independent of the choice of the leaf node.

Appendix 4.6: Proof of Lemma 7

The proof is straightforward based on proof of Lemma 4. $\sum_{v \in V_l} \Phi(v) = 1$ is proved by the induction on depth of the tree. For the tree G with $depth(G) = 1$, $\sum_{v \in V_l} \Phi(v) = 1$ is trivial as for the children v_{ij} of the root we have $\Phi(v_{ij}) = p_{ij}$ and $\sum_{ij} p_{ij} = 1$. Assume that $\sum_{v \in V_l} \Phi(v) = 1$ is true for all the trees G with $depth(G) \leq depth$. Our goal is to prove that $\sum_{v \in V_l} \Phi(v) = 1$ is true for all the trees G with $depth(G) = depth + 1$. Consider a tree G with $depth(G) = depth + 1$ and the tree G' obtained by removing all the nodes at depth $depth + 1$.

$$\sum_{v \in V_l(G)} \Phi(v)$$
$$= \sum_{v \in V_l(G), depth(v) = depth+1} \Phi(v)$$

$$
\begin{aligned}
&- \sum_{w\in V(G), w \text{ is a parent of a leaf node}, depth(w)=depth} \Phi(w) \\
&+ \sum_{v\in V_l(G')} \Phi(v) && (168)\\
&= \sum_{v\in V_l(G')} \Phi(v) && (169)\\
&= 1. && (170)
\end{aligned}
$$

(168) is a result of definition of G', i.e., G' is obtained from G by removing all the nodes at depth $depth + 1$. (169) is true as for any node w and its children v_{ij} we have $\Phi(w) = \sum_{ij} \Phi(v_{ij})$ which is a result of the fact that $\Phi(v_{ij}) = \Phi(w)p_{ij}$. (170) is concluded from the induction assumption, i.e., $\sum_{v\in V_l} \Phi(v) = 1$ is true for all the trees G with $depth(G) \leq depth$.

Appendix 4.7: Proof of Lemma 8

For any decision tree which at each node either it is pruned, accepted or branched into kl children, number of nodes in the tree is at most two times number of leaf nodes, i.e., $|V(G)| \leq 2|V_l(G)|$. This is true by induction on the depth of the tree. For a tree G with $depth(G) = 1$, we have $|V_l(G)| = kl$ and $|V(G)| = kl+1$. Therefore, Lemma 8 is true in this case. Assume that $|V(G)| \leq 2|V_l(G)|$ is true for all the trees G with $depth(G) \leq depth$. Our goal is to prove that $|V(G)| \leq 2|V_l(G)|$ is true for all the trees G with $depth(G) = depth+1$. Consider a tree G with $depth(G) = depth+1$. Consider the tree G' obtained by removing all the nodes v where $depth(v) = depth + 1$. Assume there are klr of them (each intermediate node has kl children). Therefore, we have $|V(G)| = |V(G')| + klr$, $|V_l(G)| = |V_l(G')| + (kl - 1)r$ and $|V(G')| \leq 2|V_l(G')|$. This results in $|V(G)| \leq 2|V_l(G)|$.

Appendix 5: Proof of Theorem 4

First, note that from $\frac{p_{ij}}{1+\epsilon} \leq p'_{ij} \leq p_{ij}(1+\epsilon)$, we conclude that $\frac{p_i^{\mathcal{A}}}{1+\epsilon} \leq p'^{\mathcal{A}}_i \leq p_i^{\mathcal{A}}(1+\epsilon)$, $\frac{p_j^{\mathcal{B}}}{1+\epsilon} \leq p'^{\mathcal{B}}_j \leq p_j^{\mathcal{B}}(1+\epsilon)$ and $\frac{q_{ij}}{(1+\epsilon)^2} \leq q'_{ij} \leq q_{ij}(1+\epsilon)^2$. Assume that $depth(G) = d$. Define the variables $\Phi(v)$, $\Psi^{\mathcal{A}}(v)$, $\Psi^{\mathcal{B}}(v)$, $\Psi(v)$, $\alpha(G)$, $\gamma^{\mathcal{A}}(G)$, $\gamma^{\mathcal{B}}(G)$, $\beta(G)$ and TP for the tree with the distribution p' as $\Phi'(v)$, $\Psi'^{\mathcal{A}}(v)$, $\Psi'^{\mathcal{B}}(v)$, $\Psi'(v)$, $\alpha'(G)$, $\gamma'^{\mathcal{A}}(G)$, $\gamma'^{\mathcal{B}}(G)$, $\beta'(G)$ and TP'. Therefore, from (29)–(32) we have

$$
\begin{aligned}
\alpha(G) &= \sum_{v\in V_{Buckets}(G)} \Phi(v) && (171)\\
&\leq \sum_{v\in V_{Buckets}(G)} \Phi'(v)(1+\epsilon)^d && (172)\\
&\leq \alpha'(G)(1+\epsilon)^d, && (173)
\end{aligned}
$$

(172) follows from $\Phi(v) \leq \Phi'(v)(1+\epsilon)^d$ which is a result of the definition of $\Phi(v)$ in (22), i.e., $\Phi(f(v, a_i, b_j)) = \Phi(v)p_{ij}, \forall v \in V$. Similarly, we have

$$\frac{\alpha(G)}{(1+\epsilon)^d} \leq \alpha'(G) \leq \alpha(G)(1+\epsilon)^d, \tag{174}$$

$$\frac{\gamma^{\mathcal{A}}(G)}{(1+\epsilon)^d} \leq \gamma'^{\mathcal{A}}(G) \leq \gamma^{\mathcal{A}}(G)(1+\epsilon)^d, \tag{175}$$

$$\frac{\gamma^{\mathcal{B}}(G)}{(1+\epsilon)^d} \leq \gamma'^{\mathcal{B}}(G) \leq \gamma^{\mathcal{B}}(G)(1+\epsilon)^d, \tag{176}$$

$$\frac{\beta(G)}{(1+\epsilon)^{2d}} \leq \beta'(G) \leq \beta(G)(1+\epsilon)^{2d}. \tag{177}$$

On the other hand, from (48)–(51), we have

$$TP = 1 - (1-\alpha'(G))^{\#bands} \geq 1 - (1 - \frac{\alpha(G)}{(1+\epsilon)^d})^{\#bands}, \tag{178}$$

Due to the inequality $(1-x)^{\frac{c}{x}} < e^{-c}$, the minimum possible value of $\#bands$ to ensure true positive rate TP can be computed as

$$\#bands = \lceil \frac{\log \frac{1}{1-TP}}{\frac{\alpha(G)}{(1+\epsilon)^d}} \rceil. \tag{179}$$

Thus, the total complexity is computed as

$$\begin{aligned} & c_{tree}|V(G)| + \left(\frac{c_{hash}N}{\alpha(G)} + \frac{c_{hash}M}{\alpha(G)} + \frac{c_{insertion}N\gamma'^{\mathcal{A}}(G)}{\alpha(G)} + \frac{c_{insertion}M\gamma'^{\mathcal{B}}(G)}{\alpha(G)} \right. \\ & \quad \left. + \frac{c_{pos}MN\beta'(G)}{\alpha(G)} \right) .(1+\epsilon)^d \log \frac{1}{1-TP} \qquad (180) \\ & \leq (1+\epsilon)^{3d} N^{\lambda^*} \qquad (181) \\ & \leq N^{\lambda^* + 3c_d \log(1+\epsilon)}, \qquad (182) \end{aligned}$$

where (181) follows from (174)–(177) and the fact that the total complexity for distribution p' is $O(N^{\lambda^*(p')})$. Finally, (182) is obtained from the following lemma in which we prove that the depth of the tree is bounded by $c_d \log N$ where c_d is a constant depending on the distribution.

Lemma 9 *For the decision tree G, $depth(G) = c_d \log N$ where c_d =* min $\left(\frac{(\lambda^*-1)}{\log(\max_{i,j} \frac{p_{ij}}{p_i^{\mathcal{A}}})}, \frac{(\lambda^*-\delta)}{\log(\max_{i,j} \frac{p_{ij}}{p_j^{\mathcal{B}}})}\right)$.

For proof of Lemma 9, see “Appendix 5.1”.

Appendix 5.1: Proof of Lemma 9

From (57), for the decision tree G we have

$$\begin{cases} \frac{\Phi(v)}{\Psi(v)} \geq N^{1+\delta-\lambda^*} p_0 q_0 & : \text{Accept bucket,} \\ \frac{\Phi(v)}{\Psi^{\mathcal{A}}(v)} \leq N^{1-\lambda^*} p_0 q_0 & : \text{Prune,} \\ \frac{\Phi(v)}{\Psi^{\mathcal{B}}(v)} \leq N^{\delta-\lambda^*} p_0 q_0 & : \text{Prune,} \\ \quad otherwise & : \text{Branch into the } kl \text{ children.} \end{cases} \tag{183}$$

Our goal here is to prove that for any pruned node v and any accepted node v, $depth(v) \leq c_d \log N$ where c_d is a constant depending on the distribution. Consider a pruned or accepted node v. For its parent w, we have

$$N^{1-\lambda^*} p_0 q_0 < \frac{\Phi(v)}{\Psi^{\mathcal{A}}(v)}. \tag{184}$$

Therefore, we conclude that

$$d \leq \frac{(\lambda^* - 1)\log N}{\log(\max_{i,j} \frac{p_{ij}}{p_i^{\mathcal{A}}})}. \tag{185}$$

Similarly, we have

$$N^{\delta-\lambda^*} p_0 q_0 < \frac{\Phi(v)}{\Psi^{\mathcal{B}}(v)} \tag{186}$$

$$\rightarrow d \leq \frac{(\lambda^* - \delta)\log N}{\log(\max_{i,j} \frac{p_{ij}}{p_j^{\mathcal{B}}})}. \tag{187}$$

Thus, c_d is defined as $\min\left(\frac{(\lambda^*-1)}{\log(\max_{i,j}\frac{p_{ij}}{p_i^{\mathcal{A}}})}, \frac{(\lambda^*-\delta)}{\log(\max_{i,j}\frac{p_{ij}}{p_j^{\mathcal{B}}})}\right)$. (185) and (187) are true as $p_0, q_0 \leq 1$.

Appendix 6: Pseudo code

Here, we present the pseudo code to compute the complexity of the algorithm in Dubiner (2012) in the case of hamming distance (see Experiment 1).

Algorithm 6 Algorithm for Dubiner (2012) for the case of hamming distance

```
Inputs: Probability $p$, number of data points $N$, and dimension $S$.
Output: Complexity of dubiner algorithm.
Procedure:
  For $d \in \{1, 2, \dots, S\}$
    Generate data points $x \in \{0,1\}^S$ and buckets $b \in \{0,1\}^S$ from Bernoulli($\frac{1}{2}$).
    $P_1(d) \leftarrow$ ratio of data points $x$ falling within distance $d$ of $b$.
    Generate data points $x, y \in \{0,1\}^S$ by $\begin{bmatrix} 0.5-p & p \\ p & 0.5-p \end{bmatrix}$ and buckets $b \in \{0,1\}^S$
    from Bernoulli($\frac{1}{2}$).
    $P_2(d) \leftarrow$ ratio of pairs of data points $(x, y)$ both falling within distance $d$ of $b$.
  Select $d_0$ such that $P_1(d_0) \approx \frac{1}{N}$.
Return $\frac{\log(P_2(d_0))}{\log(P_1(d_0))}$.
```

Appendix 7: Further discussion on MIPS

In order to use MIPS to solve this problem, i.e., (2), we need to derive optimal weights ω_{ij} to minimize the norm M^2 in Shrivastava and Li (2014). The term M stands for the radius of the space which is computed as follows: $M^2 = \mathbb{E}\big(||x||\big)^2 + \mathbb{E}\big(||y||\big)^2$. Therefore, from (69)–(72) we conclude that $M^2 = \sum_{ij}\left(p_j^{\mathcal{B}}\omega_{ij}^2 + \frac{p_i^{\mathcal{A}}\log^2(\frac{p_{ij}}{q_{ij}})}{\omega_{ij}^2}\right)$ which results in optimal $\omega_{ij} = \left(\frac{p_i^{\mathcal{A}}}{p_j^{\mathcal{B}}}\right)^{0.25}\left(\mid \log\frac{p_{ij}}{q_{ij}} \mid\right)^{0.5}$. On the other hand, for $(x, y) \sim Q(x, y)$ we have

$$\mathbb{E}\big(||x||||y||\big) \geq \mathbb{E}(< T(x), T(y) >) = S\sum_{ij} q_{ij} |\log(\frac{p_{ij}}{q_{ij}})\;|. \tag{188}$$

In order to have nearly one true positive rate and sub-quadratic complexity we need $S_0 \leq Sd_{KL}(p_{ij}||q_{ij})$ and $cS_0 \geq -Sd_{KL}(q_{ij}||p_{ij})$ where d_{KL} stands for kullback leibler divergence. Moreover, we should have $M^2 \geq S\sum_{ij}\sqrt{q_{ij}}|\log(\frac{p_{ij}}{q_{ij}})|$. Setting $c = 0$, S_0 and M as above, the complexity will be more than 1.9 for any 2×2 probability distribution matrix. The reason is that the transferred data points are nearly orthogonal to each other and this makes it very slow to find maximum inner product using the existing method (Shrivastava and Li 2014).

Appendix 8: Complexities of MinHash, LSH-hamming and ForestDSH

In this section, we derive the complexities of MinHash, LSH-hamming and ForestDSH in the case of $P_1 = \begin{bmatrix} 0.345 & 0 \\ 0.31 & 0.345 \end{bmatrix}$. Complexities are computed for any 2×2 probability distributions similarly.

Appendix 8.1: Complexity of MinHash

For MinHash the query complexity is

$$N^{\min(mh_1, mh_2, mh_3, mh_4)}, \tag{189}$$

where $mh_1 = \frac{\log \frac{p_{00}}{1-p_{11}}}{\log \frac{q_{00}}{1-q_{11}}}$, $mh_2 = \frac{\log \frac{p_{01}}{1-p_{10}}}{\log \frac{q_{01}}{1-q_{10}}}$, $mh_3 = \frac{\log \frac{p_{10}}{1-p_{01}}}{\log \frac{q_{10}}{1-q_{01}}}$ and $mh_4 = \frac{\log \frac{p_{11}}{1-p_{00}}}{\log \frac{q_{11}}{1-q_{00}}}$. For P_1, the per query complexity is derived and is equal to 0.5207.

Appendix 8.2: Complexity of LSH-hamming

In the case of LSH-hamming, the query complexity is

$$O\left(N^{\min\left(\frac{\log(p_{00}+p_{11})}{\log(q_{00}+q_{11})}, \frac{\log(p_{01}+p_{10})}{\log(q_{01}+q_{10})}\right)}\right), \tag{190}$$

and the storage required for the algorithm is $O(N^{1+\min(\frac{\log(p_{00}+p_{11})}{\log(q_{00}+q_{11})}, \frac{\log(p_{01}+p_{10})}{\log(q_{01}+q_{10})})})$. Similarly for P_1, the per query complexity is derived and is equal to 0.4672.

Appendix 8.3: Complexity of ForestDSH

From Definition 4, we derive λ^* as follows

$$(\mu^*, \nu^*, \eta^*) = \arg \max_{\min(\mu,\nu)\geq\eta>0, \sum_{i,j} p_{ij}^{1+\mu+\nu-\eta} (p_i^{\mathcal{A}})^{-\mu} (p_j^{\mathcal{B}})^{-\nu}=1} \frac{1+\mu+\nu}{1+\mu+\nu-\eta} \tag{191}$$

$$= (4.6611, 4.6611, 3.1462) \tag{192}$$

$$\lambda^* = \frac{1+\mu^*+\nu^*}{1+\mu^*+\nu^*-\eta^*} \tag{193}$$

$$= 1.4384. \tag{194}$$

Note that $\delta = 1$ and the per query complexity is equal to 0.4384.

Appendix 9: Joint probability distributions learned on mass spectrometry data

The mass spectrometry data for experiment 4, is shown in Fig. 10a–c in case of $\log Rank$ at base 4 (a 4×4 matrix), $\log Rank$ at base 2 (an 8×8 matrix), and no $\log Rank$ transformation (a 51×51 matrix). For the mass spectrometry data shown in Fig. 10a, the probability distribution $P_{4\times 4}$ is given in (195). Note that, in the case of LSH-hamming the query complexity for these 4×4, 8×8 and 51×51 matrices are 0.901, 0.890 and 0.905, respectively. Similarly, per query complexity for MinHash for these 4×4, 8×8 and 51×51 matrices are 0.4425, 0.376 and 0.386, respectively.

For the mass spectrometry data shown in Fig. 10a, b, the probability distribution $p(x, y)$ is represented as

$$P_{4\times 4} = \begin{bmatrix} 0.000125 & 5.008081 \times 10^{-5} & 9.689274 \times 10^{-8} & 0.000404 \\ 5.008082 \times 10^{-5} & 0.000209 & 6.205379 \times 10^{-6} & 0.001921 \\ 9.689274 \times 10^{-8} & 6.205379 \times 10^{-6} & 2.688879 \times 10^{-5} & 0.000355 \\ 0.000404 & 0.001921 & 0.000355 & 0.994165 \end{bmatrix}, \tag{195}$$

$$P_{8\times 8} = \left[\begin{matrix} 3.458 \times 10^{-5} & 1.442 \times 10^{-5} & 5.434 \times 10^{-6} & 1.723 \times 10^{-6} \\ 1.442 \times 10^{-5} & 3.708 \times 10^{-5} & 2.550 \times 10^{-5} & 8.706 \times 10^{-6} \\ 5.434 \times 10^{-6} & 2.550 \times 10^{-5} & 3.907 \times 10^{-5} & 2.948 \times 10^{-5} \\ 1.723 \times 10^{-6} & 8.706 \times 10^{-6} & 2.948 \times 10^{-5} & 4.867 \times 10^{-5} \\ 2.921 \times 10^{-7} & 1.561 \times 10^{-6} & 6.442 \times 10^{-6} & 1.813 \times 10^{-5} \\ 7.496 \times 10^{-8} & 4.809 \times 10^{-7} & 2.008 \times 10^{-6} & 6.098 \times 10^{-6} \\ 6.718 \times 10^{-8} & 2.680 \times 10^{-7} & 1.251 \times 10^{-6} & 4.531 \times 10^{-6} \\ 5.023 \times 10^{-5} & 1.574 \times 10^{-4} & 3.671 \times 10^{-4} & 5.539 \times 10^{-4} \end{matrix}\right. \tag{196}$$

$$\left.\begin{matrix} 2.920 \times 10^{-7} & 7.496 \times 10^{-8} & 6.718 \times 10^{-8} & 5.023 \times 10^{-5} \\ 1.561 \times 10^{-6} & 4.809 \times 10^{-7} & 2.680 \times 10^{-7} & 1.575 \times 10^{-4} \\ 6.442 \times 10^{-6} & 2.008 \times 10^{-6} & 1.251 \times 10^{-6} & 3.672 \times 10^{-4} \\ 1.813 \times 10^{-5} & 6.098 \times 10^{-6} & 4.532 \times 10^{-6} & 5.539 \times 10^{-4} \\ 2.887 \times 10^{-5} & 6.892 \times 10^{-6} & 5.309 \times 10^{-6} & 4.138 \times 10^{-4} \\ 6.892 \times 10^{-6} & 2.123 \times 10^{-5} & 5.826 \times 10^{-6} & 3.246 \times 10^{-4} \\ 5.309 \times 10^{-6} & 5.826 \times 10^{-6} & 6.411 \times 10^{-5} & 8.364 \times 10^{-4} \\ 4.138 \times 10^{-4} & 3.246 \times 10^{-4} & 8.364 \times 10^{-4} & 0.994 \end{matrix}\right]. \tag{197}$$

From (56), for $P_{4\times 4}$, $(\mu^*, \nu^*, \eta^*, \lambda^*)$ are derived as

$$\mu^* = 1.151016, \tag{198}$$
$$\nu^* = 1.151016, \tag{199}$$
$$\eta^* = 0.813168, \tag{200}$$
$$\lambda^* = 1.326723. \tag{201}$$

Similarly, for $P_{8\times 8}$, we have

$$\mu^* = 0.871147, \tag{202}$$
$$\nu^* = 0.871147, \tag{203}$$
$$\eta^* = 0.624426, \tag{204}$$
$$\lambda^* = 1.294837. \tag{205}$$

For the mass spectrometry data shown in Fig. 10c, $(\mu^*, \nu^*, \eta^*, \lambda^*)$ are

$$\mu^* = 0.901208, \tag{206}$$
$$\nu^* = 0.901208, \tag{207}$$
$$\eta^* = 0.615797, \tag{208}$$
$$\lambda^* = 1.281621. \tag{209}$$

References

Aebersold R, Mann M (2003) Mass spectrometry-based proteomics. Nature 422(6928):198–207

Anagnostopoulos E, Emiris IZ, Psarros I (2015) Low-quality dimension reduction and high-dimensional approximate nearest neighbor. In: 31st international symposium on computational geometry (SoCG 2015), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

Andoni A, Indyk P, Laarhoven T, Razenshteyn I, Schmidt L (2015) Practical and optimal LSH for angular distance. In: Advances in neural information processing systems, pp 1225–1233

Andoni A, Laarhoven T, Razenshteyn I, Waingarten E (2017) Optimal hashing-based time-space trade-offs for approximate near neighbors. In: Proceedings of the twenty-eighth annual ACM-SIAM symposium on discrete algorithms. SIAM, pp 47–66

Andoni A, Naor A, Nikolov A, Razenshteyn I, Waingarten E (2018) Data-dependent hashing via nonlinear spectral gaps. In: Proceedings of the 50th annual ACM SIGACT symposium on theory of computing, pp 787–800

Andoni A, Razenshteyn I (2015) Optimal data-dependent hashing for approximate near neighbors. In: Proceedings of the forty-seventh annual ACM symposium on theory of computing, pp 793–801

Bawa M, Condie T, Ganesan P (2005) LSH forest: self-tuning indexes for similarity search. In: Proceedings of the 14th international conference on world wide web, pp 651–660

Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517

Beyer K, Goldstein J, Ramakrishnan R, Shaft U (1999) When is “nearest neighbor” meaningful? In: International conference on database theory. Springer, pp 217–235

Bhatia K, Jain H, Kar P, Varma M, Jain P (2015) Sparse local embeddings for extreme multi-label classification. In: Advances in neural information processing systems, pp 730–738

Castelli V, Li CS, Thomasian A (2000) Searching multidimensional indexes using associated clustering and dimension reduction information. U.S. Patent No. 6,134,541

Chakrabarti A, Regev O (2010) An optimal randomized cell probe lower bound for approximate nearest neighbor searching. Soc Ind Appl Math SIAM J Comput 39(5):1919–1940

Charikar MS (2002) Similarity estimation techniques from rounding algorithms. In: Proceedings of the thirty-fourth annual ACM symposium on theory of computing, pp 380–388

Choromanska AE, Langford J (2015) Logarithmic time online multiclass prediction. In: Advances in neural information processing systems, pp 55–63

Christiani T, Pagh R (2017) Set similarity search beyond MinHash. In: Proceedings of the 49th annual ACM SIGACT symposium on theory of computing, pp 1094–1107

Dasarathy BV, Sheela BV (1977) Visiting nearest neighbors—a survery of nearest neighbor pattern classification techniques. In: Proceedings of the international conference on cybernetics and society, pp 630–636

Dubiner M (2010) Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. IEEE Trans Inf Theory 56(8):4166–4179

Dubiner M (2012) A heterogeneous high-dimensional approximate nearest neighbor algorithm. IEEE Trans Inf Theory 58(10):6646–6658

Duda RO, Hart PE, Stork DG (1973) Pattern classification and scene analysis, vol 3. Wiley, New York

Frank AM, Monroe ME, Shah AR, Carver JJ, Bandeira N, Moore RJ, Anderson GA, Smith RD, Pevzner PA (2011) Spectral archives: extending spectral libraries to analyze both identified and unidentified spectra. Nat Methods 8(7):587–591

Friedman JH, Bentley JL, Finkel RA (1977) An algorithm for finding best matches in logarithmic expected time. ACM Trans Math Softw TOMS 3(3):209–226

Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: International conference on very large data bases, VLDB, vol 99, pp 518–529

Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on management of data, pp 47–57

Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on theory of computing, pp 604–613

Jain H, Prabhu Y, Varma M (2016) Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp 935–944

Kim S, Pevzner PA (2014) MS-GF+ makes progress towards a universal database search tool for proteomics. Nat Commun 5:5277

Krizhevsky A (2009) Learning multiple layers of features from tiny images. Master's thesis, University of Toronto

Liu W, Tsang IW (2017) Making decision trees feasible in ultrahigh feature and label dimensions. J Mach Learn Res 18(1):2814–2849

McDonald D, Hyde E, Debelius JW, Morton JT, Gonzalez A, Ackermann G, Aksenov AA, Behsaz B, Brennan C, Chen Y, Goldasich LD (2018) American Gut: an open platform for citizen science microbiome research. Msystems 3(3):e00031-18

Miltersen PB (1999) Cell probe complexity-a survey. In: Proceedings of the 19th conference on the foundations of software technology and theoretical computer science, advances in data structures workshop, p 2

Min R (2005) A non-linear dimensionality reduction method for improving nearest neighbour classification. University of Toronto, Toronto

Mori G, Belongie S, Malik J (2001) Shape contexts enable efficient retrieval of similar shapes. In: Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, IEEE, vol 1, pp I

Nam J, Mencía EL, Kim HJ, Fürnkranz J (2017) Maximizing subset accuracy with recurrent neural networks in multi-label classification. In: Advances in neural information processing systems, pp 5413–5423

Niculescu-Mizil A, Abbasnejad E (2017) Label filters for large scale multilabel classification. In: Artificial intelligence and statistics, pp 1448–1457

Prabhu Y, Varma M (2014) Fastxml: a fast, accurate and stable tree-classifier for extreme multi-label learning. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining, pp 263–272

Rai P, Hu C, Henao R, Carin L (2015) Large-scale Bayesian multi-label learning via topic-based label embeddings. In: Advances in neural information processing systems, pp 3222–3230

Rubinstein A (2018) Hardness of approximate nearest neighbor search. In: Proceedings of the 50th annual ACM SIGACT symposium on theory of computing, pp 1260–1268

Shakhnarovich G, Viola P, Darrell T (2003) Fast pose estimation with parameter-sensitive hashing. In: Proceedings of the ninth IEEE international conference on computer vision. IEEE, vol 2, p 750

Shaw B, Jebara T (2009) Structure preserving embedding. In: Proceedings of the 26th annual international conference on machine learning, pp 937–944

Shrivastava A, Li P (2014) Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In: Advances in neural information processing systems, pp 2321–2329

Tagami Y (2017) Annexml: approximate nearest neighbor search for extreme multi-label classification. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 455–464

Yen IEH, Huang X, Ravikumar P, Zhong K, Dhillon I (2016) Pd-sparse: a primal and dual sparse approach to extreme multiclass and multilabel classification. In: International conference on machine learning, pp 3069–3077

Yianilos PN (1993) Data structures and algorithms for nearest neighbor search in general metric spaces. In: Symposium on discrete algorithms, SODA, vol 93, pp 311–321

Zhou WJ, Yu Y, Zhang ML (2017) Binary linear compression for multi-label classification. In: Proceedings of the twenty-sixth international joint conference on artificial intelligence, IJCAI, pp 3546–3552

Affiliations

Arash Gholami Davoodi[1] · Sean Chang[1] · Hyun Gon Yoo[1] · Anubhav Baweja[1] · Mihir Mongia[1] · Hosein Mohimani[1]

✉ Arash Gholami Davoodi
agholami@andrew.cmu.edu

Sean Chang
szchang@andrew.cmu.edu

Hyun Gon Yoo
hyungony@andrew.cmu.edu

Anubhav Baweja
abaweja@andrew.cmu.edu

Mihir Mongia
mmongia@andrew.cmu.edu

Hosein Mohimani
hoseinm@andrew.cmu.edu

1 Carnegie Mellon University, Pittsburgh, PA 15213, USA