Procedia Computer Science
Volume 53, 2015, Pages 64–73
2015 INNS Conference on Big Data

Fast ℓ_1-norm Nearest Neighbor Search Using A Simple Variant of Randomized Partition Tree

Kaushik Sinha[1]

Wichita State University, Wichita, Kansas, USA
kaushik.sinha@wichita.edu

Abstract
For big data applications, randomized partition trees have recently been shown to be very effective in answering high dimensional nearest neighbor search queries with provable guarantee, when distances are measured using ℓ_2 norm. Unfortunately, if distances are measured using ℓ_1 norm, the same theoretical guarantee does not hold. In this paper, we show that a simple variant of randomized partition tree, which uses a different randomization using 1-stable distribution, can be used to efficiently answer high dimensional nearest neighbors queries when distances are measured using ℓ_1 norm. Experimental evaluations on eight real datasets suggest that the proposed method achieves better ℓ_i-norm nearest neighbor search accuracy with fewer retrieved data points as compared to locality sensitive hashing.

Keywords: Nearest Neighbor Search, Big Data, Randomized Partition Tree.

1 Introduction

Nearest-neighbor search is an important problem in computer science and is used in wide variety of application areas. Given a set of n objects and a separate query object q, the task of nearest neighbor search is to return an object from the set of these n objects which is closest to q, measured using appropriate distance metric. While the naive solution, obtained simply by performing a linear scan over the given set of objects, guarantees to find the exact nearest neighbor of any query object in $O(n)$ time, difficulty lies in finding a solution in sub-linear time using a data structure of size $O(n)$. In the big data era, both n and number of features (d) of each object can be really large and finding a sub-linear time accurate nearest solution is of extreme importance. Over the past two decades, numerous solution techniques have been developed that achieve this goal and answer either approximate or exact version of nearest neighbor search problem when data have some low intrinsic dimensionality measured in the form of either *doubling dimension* or *doubling measure* [1, 11, 3, 16, 17, 9, 21]. Among these, the work in [9] analyzes the failure probability of nearest neighbor search for a popular and empirically successful data structure, namely randomized partition (RP) trees, which combines classical k-d tree partitioning with randomization and shows that for any query point, it possible

Selection and peer-review under responsibility of the Scientific Programme Committee of INNS-BigData2015

doi:10.1016/j.procs.2015.07.280

to find exact nearest neighbor in $O(\log n)$ time where the hidden constant depends only on low intrinsic data dimension d_0, where, $d_0 \ll d$. However, the data structure developed in [9] can answer nearest neighbor queries when distances are measured using ℓ_2 norm. A natural question to ask is whether a similar data structure as in [9] can be used for finding nearest neighbors in big data applications where distances are measured using ℓ_1 norm. Note that, in case of dimension reduction, while random projection based methods involving ℓ_2 norm [22] ensure nice theoretical guarantees towards pairwise distance preservation due to the celebrated JL Lemma [15, 8], no such guarantee is known for dimension reduction using ℓ_1 norm. In fact, the recent impossibility results [4, 18, 5] tell us that one can not even hope to preserve pairwise distances after random projection in ℓ_1 without allowing large errors. In spite of these negative results, in this paper, we show that a slight variant of randomized partition tree used in [9] can efficiently find nearest neighbors in high dimensions when distances are measured using ℓ_1 norm. The theoretical analysis of the proposed data structure presented in this paper relies heavily on the same techniques developed in [9]. However, it breaks down at one important juncture due to the unpleasant property of the ratio of two Cauchy distributions. We show how to overcome this hurdle and provide theoretical guarantee for the performance of the proposed data structure in answering nearest neighbor queries using ℓ_1 norm for big data applications. The rest of the paper is organized as follows. In section 2, we present the variant of randomized partition tree data structure proposed in this paper. Analysis of its failure probability in finding true nearest neighbor is presented in section 3. In section 4, we present the experimental evaluations and finally conclude in section 5.

2 Random Projection trees using 1-stable distribution

We start with the definition of p-stable distributions and then describe the RP tree data structure constructed using 1-stable distributions (i.e., for $p = 1$) that we study in this paper.

2.1 p-Stable distribution

Stable distributions [13] are defined as limits of normalized sums of independent and identically distributed variables. For any $p > 0$, a p-stable distribution $\mathcal{D}_p$, satisfies the following. For any X_1, X_2, X_3 sampled i.i.d from $\mathcal{D}_p$ and for any $a, b \in \mathbb{R}$, the random variables $aX_1 + bX_2$ and $(|a|^p + |b|^p)^{1/p}X_3$ have the same distribution. In particular, for 1-stable distribution, $aX_1 + bX_2$ and $(|a| + |b|)X_3$ have the same distribution. Example of an one dimensional 1-stable distribution is Cauchy distribution (with scale parameter 1 and location parameter 0) [13] whose probability density function is given by :

$$f(x) = \frac{1}{\pi(1+x^2)} \text{ for } -\infty < x < \infty. \tag{1}$$

It is known that, random variable having probability density function as in equation 1 can be generated by first choosing a parameter θ uniformly at random from an interval $[-\pi/2, \pi/2]$ and then computing $\tan(\theta)$ ([7]). Now, let $U \in \mathbb{R}^d$ be a vector such that each of its d coordinates are chosen independently at random from a Cauchy distribution as in equation 1. For any $x \in \mathbb{R}^d$, a random projection from $\mathbb{R}^d$ to $\mathbb{R}$ using U is given by the map, $x \mapsto U^\top x$. It is clear that $U^\top x$ has probability distribution $\|x\|_1 Z$, where Z is distributed according to a 1-stable distribution [13].

2.2 Random projection trees

An RP tree (of the variant that we consider in this paper) is a space partitioning data structure in which, the root node (cell) contains all the data points. At any level of the tree construction,

Algorithm 1 Space Partitioning Tree

```
Input : data $S = \{x_1, \ldots, x_n\}$, maximum number of data points in leaf node $n_0$
Output : tree data structure
function MakeTree($S, n_0$)

1: if $|S| \leq n_0$ then
2:     return leaf containing $S$
3: else
4:     Rule = ChooseRule($S$)
5:     LeftTree = MakeTree ($\{x \in S : \text{Rule} = \text{true}\}, n_0$)
6:     RightTree = MakeTree ($\{x \in S : \text{Rule} = \text{false}\}, n_0$)
7:     return (Rule, LeftTree, RightTree)
8: end if
```

Algorithm 2 Function ChooseRule for ℓ_1 norm Random Projection Tree

```
function ChooseRule($S$)
1: Pick $U$ such that each of its $d$ coordinates are chosen uniformly at random from a Cauchy distribution as follows
2: for $i = 1$ to $d$ do
3:     Pick $\theta$ uniformly at random from $[-\pi/2, \pi/2]$
4:     Set $U(i) = \tan(\theta)$
5: end for
6: Pick $\beta$ uniformly at random from $[1/4, 3/4]$
7: Let $v$ be the $\beta$-fractile point on the projection of $S$ onto $U$
8: Rule($x$) = ($U^\top x \leq v$)
9: return (Rule)
```

a cell is split by first choosing a direction whose coordinates are chosen uniformly at random from a Cauchy distribution (Equation 1), followed by projecting the points in this cell onto that direction, and finally splitting at the β fractile, for β chosen uniformly at random from $[1/4, 3/4]$, to construct left and right sub-cells which becomes the root nodes of left and right subtrees respectively. The process of splitting and growing the tree continues until the leaf nodes (cells) contain at most n_0 (any pre-specified input parameter) data points. The generic space partitioning tree procedure is shown in Algorithm 1. Careful choice of **ChooseRule** function in line 4 of Algorithm 1 leads to appropriate variant of such a generic space partitioning tree. For the variant of RP tree that we consider in this paper, Algorithm 2 provides the details of the **ChooseRule** function. In order to answer a nearest neighbor query using this data structure, the query point is routed to a particular leaf node, following the path from root to leaf using the same splitting rule, and its nearest neighbor among the points falling within that leaf node is returned.

3 Analysis of randomized partition trees using 1-stable distribution

The proposed variant of RP tree presented in this paper is the first to consider 1-stable distribution for choosing projection direction (lines 3, 4 of Algorithm 2). In previous versions [9, 10] each coordinate of projection direction was chosen i.i.d from standard normal distribution, where the goal was to find nearest neighbors based on ℓ_2 distance. Due to this, performance analysis of this proposed data structure follows similar technique as was developed in [9]. However, difficulty arises due to unpleasant property of the ratio of two Cauchy distributions resulting from this different randomization (see section 3.2 for details). We show how to overcome this hurdle, leading to a slightly different form of potential function (to be introduced later) and theoretical performance guarantee compared to [9]. We start the analysis in section 3.1 following the same technique as in [9].

3.1 How does Random projection using 1-stable distribution affect the relative position of three points?

Consider any three points $q, x, y \in \mathbb{R}^d$ such that q is closer to to x than to y in ℓ_1 distance, i.e., $\|q - x\|_1 \leq \|q - y\|_1$. Let U be a random vector in $\mathbb{R}^d$ generated by selecting each of its d coordinates uniformly at random according to equation 1. Under this random choice, we want to estimate the probability of the event that $U^\top y$ falls in between $U^\top q$ and $U^\top x$. Following [9], without loss of generality, we will assume that $q = 0$ is the origin and x is along the positive e_1 axis, that is, $x = \|x\|_1 e_1$. It will be helpful to split U into two pieces, the component U_1 along the e_1 direction and the remaining $d-1$ coordinate U_R, so that U_1 and U_R are independent. Likewise, we will write $y = (y_1, y_R)$. Our event of interest is :

$$E \equiv U^\top y \text{ falls between } U^\top q \text{ (that is 0) and } U^\top x \equiv y_1 U_1 + U_R^\top y_R \text{ falls between 0 and } \|x\|_1 U_1$$

Let Z and Z' be two independent Cauchy random variables. Because of 1-stability of Cauchy distribution, $U_R^\top y_R$ is distributed as $\|y_R\|_1 Z'$. Note that this distribution is symmetric hence assigns same probability mass to the intervals $(-y_1|U_1|, (\|x\|_1 - y_1)|U_1|)$ and $(-(\|x\|_1 - y_1)|U_1|, y_1|U_1|)$ [9]. Using 1-stability of Cauchy distribution again, it follows that $|U_1|$ is distributed as Z. Therefore, following similar arguments as in [9],

$$\begin{aligned} \Pr_U(E) &= \Pr_{U_1}\Pr_{U_R}(-y_1|U_1| < U_R^\top y_R < (\|x\|_1 - y_1)|U_1|) = \Pr(-y_1 Z < \|y_R\|_1 Z' < (\|x\|_1 - y_1)|Z) \\ &= \Pr\left(\frac{Z'}{Z} \in \left(-\frac{y_1}{\|y_R\|_1}, \frac{\|x\|_1 - y_1}{\|y_R\|_1}\right)\right) \end{aligned}$$

Unlike [9], where $\frac{Z'}{Z}$ follows a Cauchy distribution, in our case, $\frac{Z'}{Z}$ is ratio of two independent Cauchy distributions having the following density function ([20]) :

$$f(u) = \frac{\log u^2}{\pi^2(u^2-1)} \text{ for } -\infty < u < \infty. \tag{2}$$

The function $f(u)$ becomes infinite as u approaches 0 and its value is indeterminate at $u = \pm 1$. But using l'Hôpital's rule it can be shown [20] that $f(u)$ approaches $1/\pi^2$ as u approaches ± 1. The density given in Equation 2 along with Equation 1 is plotted in Figure 1. Due to "not so nice property" of Equation 2, unlike [9], it is not straightforward to evaluate $\Pr_U(E)$. In the following, we discuss how this probability can be bounded from above.

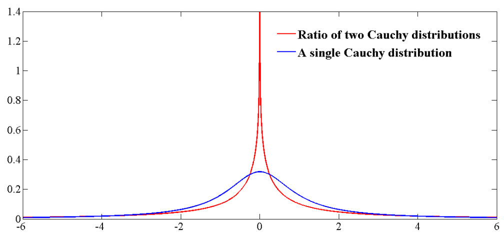

Figure 1: Probability densities of Equation 1 and 2.

3.2 Probability estimation

Let $f(u) = \frac{\log u^2}{\pi^2(u^2-1)}$. First, consider the case when $y_1 > 0$. Suppose $\frac{\|x\|_1}{\|y\|_1} = r$. Then,

$$\Pr_U(E) = \int_{-\frac{y_1}{\|y_R\|_1}}^{\frac{\|x\|_1 - y_1}{\|y_R\|_1}} f(u)du = \int_{-\frac{|y_1|}{\|y\|_1 - |y_1|}}^{\frac{r\|y\|_1 - |y_1|}{\|y\|_1 - |y_1|}} f(u)du = \int_{-\frac{k}{1-k}}^{\frac{r-k}{1-k}} f(u)du \tag{3}$$

where, $k = \frac{|y_1|}{\|y\|_1} \in [0, 1]$. When $y_1 < 0$, in a similar manner, it is easy to see that $\Pr_U(E) = \int_{\frac{k}{1-k}}^{\frac{r+k}{1-k}} f(u)du$. Note that the expression $\Pr_U(E)$ now is a function of two variables r and k both of which take values in $[0, 1]$. The combinations of r and k which are of interest are when, (a) r and k both are very close to zero. This corresponds to an interval of integration very close to origin where equation 2 differs significantly from equation 1, and, (b) when both r and k are close to 1, leading to an interval of integration which is larger than the symmetric half. To better understand the behavior $\Pr_U(E)$ as a function of k and r, we do the following : (1) Fix any $r \in [0, 1]$, (2) Numerically evaluate the integral in equation 3 for different values of $k \in [0, 1]$ and fixed value of r as in step 1, (3) Repeat step 1 and 2 above, for various values of r. We show these plots in Figures 2, 3, 4, 5 respectively. In each of these figures, we plot $\Pr_U(E)$ vs $k = \frac{|y_1|}{\|y\|_1}$ for various values of $r = \frac{\|x\|_1}{\|y\|_1}$.

Plot of $\Pr_U(E)$ for different values of r are shown using different colors. For each color (fixed r), the solid line represents the actual value of $\Pr_U(E)$ computed numerically for that fixed r and all k, while the dashed line of same color represents a constant value of $\Pr_U(E)$ for all k, which is set to be r in Figures 2, 3, and $0.8\sqrt{r}$ in Figures 4, 5 respectively.

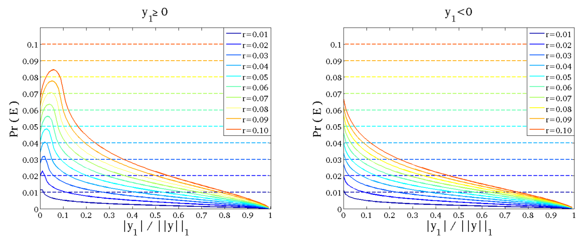

Figure 2: Behavior of $\Pr(E)$ vs $|y_1|/\|y\|_1$ in the range $r \in [0.01, 0.1]$ where the upper bound of $\Pr(E)$ is set to r represented by the dashed lines.

One might guess that since we are merely using ℓ_1 norm in place of ℓ_2 norm and using similar analysis technique as in [9], $\Pr_U(E)$ can be bounded from above by the quantity $\frac{\|x\|_1}{2\|y\|_1} = \frac{r}{2}$, as was established in case of ℓ_2 norm [9]. It turns out that, that is not the case here. In fact, as can be seen from Figure 2, for small values of r and k, the probability $\Pr_U(E)$ is not bounded from above by any $cr = c\frac{\|x\|_1}{\|y\|_1}$ for $c \leq 1$, which is understandable since according to Equation 3, this corresponds to integral of a small interval which is very close to the origin where equation 2 differs significantly from Equation 1 (Figure 1).

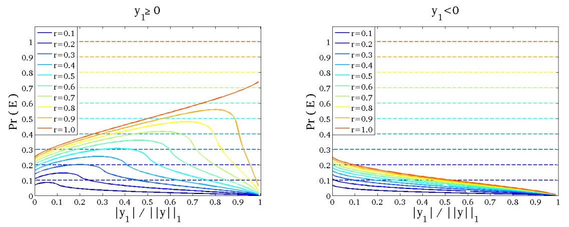

Figure 3: Behavior of $\Pr(E)$ vs $|y_1|/\|y\|_1$ in the range $r \in [0.1, 1.0]$ where the upper bound of $\Pr(E)$ is set to r represented by the dashed lines.

Geometrically, this happens when, x is very close to the query point (which is origin in this case) while y is far away from the query point and the component of y along the direction x is very very small. Since both $\Pr_U(E)$ and r take values in the interval $[0, 1]$, it is conceivable (from Figures 2 and 3) that $\Pr_U(E)$ may be bounded from above by a polynomial of r whose degree is less than one (as we have seen, the bound doesn't work[1] for r and a potential bound could be of the form cr^a for any $0 < a < 1$ and $0 < c < 1$ since $0 \leq r \leq r^a \leq 1$).

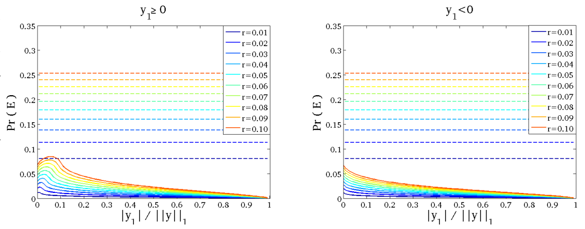

Figure 4: Behavior of $\Pr(E)$ vs $|y_1|/\|y\|_1$ in the range $r \in [0.01, 0.1]$ where the upper bound of $\Pr(E)$ is set to $0.8\sqrt{r}$ represented by the dashed lines.

One might guess that the bound can take the form $c\sqrt{r}$ for some $c \in [0, 1]$. It turns out that, it is indeed the case and the value of c is dictated by domain where $r = 1$ and k approaches 1, for $y_1 > 0$. Assume $k = 1 - \epsilon$ for some small positive ϵ, such that as $\epsilon \to 0, k \to 1$. It is easy to see from Equation 3 that for $r = 1$ and $k = 1 - \epsilon$, the interval of the integral for Equation 3 is $[-\frac{1}{\epsilon} + 1, 1]$. For small values of ϵ, this interval of integration approaches $(-\infty, 1]$ and due to symmetry of

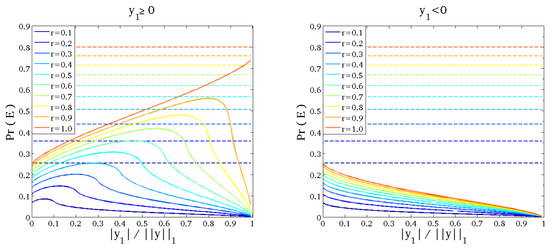

Figure 5: Behavior of $\Pr(E)$ vs $|y_1|/\|y\|_1$ in the range $r \in [0.1, 1.0]$ where the upper bound of $\Pr(E)$ is set to $0.8\sqrt{r}$ represented by the dashed lines.

[1]Note that, it may be possible to bound $\Pr_U(E)$ from above by $c'r$ for some $c' > 1$. However, for $r = 1$ that is when x and y are equidistant from the query point (measured using ℓ_1 norm), the bound becomes meaningless.

the density of ratio of two independent Cauchy random variables, the associated probability mass is clearly greater than 1/2, indicating that $\sqrt{r}/2$ can not be a valid upper bound either. It turns out, from Figures 4, 5 that $0.8\sqrt{r}$ works as a valid upper bound for all combinations of k and r. For non zero query point q, this insight leads to :

Proposition 3.1. *Pick any $q, x, y \in \mathbb{R}^d$ with $\|q-x\|_1 \leq \|q-y\|_1$. Pick a random direction $U \in \mathbb{R}^d$ whose coordinates are chosen uniformly at random from a Cauchy distribution. Then probability, over U, that $U^\top y$ falls (strictly) between $U^\top q$ and $U^\top x$ is $\frac{4}{5}\sqrt{\|q-x\|_1/\|q-y\|_1}$.*

3.3 Potential function and its effect on nearest neighbor search

For a query $q \in \mathbb{R}^d$ and data points $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$, let $x_{(1)}, x_{(2)}, \ldots$ denote a re-ordering of the points by increasing ℓ_1 distance from q. Define the potential function : $\tilde{\Phi}(q, \{x_1, \ldots, x_n\}) = \frac{1}{n}\sum_{i=2}^{n}\sqrt{\|q-x_{(1)}\|_1/\|q-x_{(i)}\|_1}$. Note that $\tilde{\Phi} \in [0,1]$ and this is very similar to the potential function introduced in [9] i.e., $\Phi(q, \{x_1, \ldots, x_n\}) = \frac{1}{n}\sum_{i=2}^{n}(\|q-x_{(1)}\|_2/\|q-x_{(i)}\|_2)$, except that here, (a) ℓ_2 norm is replaced by ℓ_1 norm, and (b) linear function is replaced by a square root function. Using $\tilde{\Phi}$, we analyze the failure probability of this new variant of RP tree next.

Lemma 3.2. *Pick any points $q, x_1, \ldots, x_n \in \mathbb{R}^d$. If these points are projected to a direction $U \in \mathbb{R}^d$ whose coordinates are chosen uniformly at random from a Cauchy distribution, then the expected fraction of the projected x_i that fall between q and $x_{(1)}$ is at most $(4/5)\tilde{\Phi}(q, \{x_1, \ldots, x_n\})$.*

Proof. Let Z_i be the event that $x_{(i)}$ falls between q and $x_{(1)}$ in the projection. Using Proposition 3.1, $\Pr_U(Z_i) \leq \frac{4}{5}\left(\sqrt{\|q-x_{(1)}\|_1/\|q-x_{(i)}\|_1}\right)$. The result now follows from linearity of expectation. □

Since the tree data structure partitions the input data space, any cell (at any level) of this tree contains only a subset of the n data points $x_1, \ldots, x_n$. For any cell that contains m ($m \leq n$) data points, the appropriate variant of the potential function is : $\tilde{\Phi}_m(q, \{x_1, \ldots, x_n\}) = \frac{1}{m}\sum_{i=1}^{m}\sqrt{\|q-x_{(1)}\|_1/\|q-x_{(i)}\|_1}$.

Corollary 3.3. *Pick any points $q, x_1, \ldots, x_n \in \mathbb{R}^d$ and let S denotes any subset of the x_i that includes $x_{(1)}$. If q and the points in S are projected to a direction $U \in \mathbb{R}^d$ whose coordinates are chosen uniformly at random from a Cauchy distribution, then for any $0 < \alpha < 1$, the probability (over U) that at least α fraction of the projected S falls between q and $x_{(1)}$ is at most $(4/5)\tilde{\Phi}_{|S|}(q, \{x_1, \ldots, x_n\})$.*

Proof. Apply Lemma 3.2 to S noting that the corresponding value of $\tilde{\Phi}$ is maximized when S consists of points closest to q measured using ℓ_1 distance. The result now follows by applying Markov's inequality. □

We are now ready to present the main result. In many of the statements below, we will drop the arguments $(q, \{x_1, \ldots, x_n\})$ of $\tilde{\Phi}_m(q, \{x_1, \ldots, x_n\})$ in the interest of readability.

Theorem 3.4. *Suppose a randomized partition tree is built using points $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ according to Algorithm 1 and 2, and is used to answer a query q. Define $\beta = \frac{3}{4}$ and $l = \log_{1/\beta}(n/n_0)$. Then over the randomizartion of tree construction, the probability that NN query does not return $x_{(1)}$ is at most $\frac{8}{5}\sum_{i=0}^{l}\tilde{\Phi}_{\beta^i n}\ln(5e/4\tilde{\Phi}_{\beta^i n})$.*

Proof. Consider any internal node of the tree that contains q as well as m of the data points, including $x_{(1)}$. What is the probability that the split at that node separates q from $x_{(1)}$? To analyze this, let F denote the fraction of the m points that fall between q and $x_{(1)}$ along the randomly-chosen split direction. Since the split point is chosen at random from an interval of mass 1/2, the probability that it separates q from $x_{(1)}$ is at most $F/(1/2)$. Integrating out F, we get : $\Pr\left(q \text{ is separated from } x_{(1)}\right) \leq \int_0^1 \Pr(F = f)\frac{f}{1/2}df = 2\int_0^1 \Pr(F > f)df \leq 2\int_0^1 \min\left(1, \frac{4}{5f}\right)df = 2\int_0^{4/5\tilde{\Phi}_m} df + 2\int_{4/5\tilde{\Phi}_m}^1 \frac{4\tilde{\Phi}_m}{5f}df = \frac{8}{5}\tilde{\Phi}_m \ln\left(\frac{5e}{4\tilde{\Phi}_m}\right)$, where, the second inequality uses Corollary 3.3. The result now follows by taking an union bound over the path that conveys q from root to leaf, in which the number of data points per level shrinks geometrically, by a factor of 3/4 or better. $\square$

The failure probability of nearest neighbor search using randomized partition trees as given in Theorem 3.4 is related to potential function $\tilde{\Phi}$ which takes vales in the range $[0, 1]$. The worst case happens when all points are equidistant from query point in ℓ_1 norm, in which case $\tilde{\Phi}$ is 1. However, under favorable condition, $\tilde{\Phi}_m$ can be bounded by a decreasing function of m. One such setting is when the query points are chosen arbitrarily but the data points used to construct the randomized tree data structure are drawn i.i.d. from an underlying distribution μ on $\mathbb{R}^d$ which is a *doubling measure*, i.e., there exists a constant $C > 0$ and a subset $\mathcal{X} \subseteq \mathbb{R}^d$ such that $\mu(B(x, 2r)) \leq C\mu(B(x, r))$ for all $x \in \mathcal{X}$ and all $r > 0$, where, $B(x, r) = \{y \in \mathbb{R}^d : \|x - y\|_1 \leq r\}$. This essentially says that probability mass of an ℓ_1 ball grows polynomially in its radius and an equivalent formulation is that, there exists a constant $d_0 > 0$ and a subset $\mathcal{X} \subset \mathbb{R}^d$ such that for all $x \in \mathcal{X}$, all $r > 0$ and al $\alpha \geq 1$, we have $\mu(B(x, \alpha r)) \leq \alpha^{d_0}\mu(B(x, r))$. Comparing this with standard formula for the volume of a ball, the degree of polynomial $d_0 = \log_2 C$ can be thought of *dimension* of measure μ. We next show an upper bound for $\tilde{\Phi}_m$ that depends on only on d_0 and becomes useful especially when $d_0 \ll d$.

Lemma 3.5. *Suppose μ is continuous on $\mathbb{R}^d$ and is a doubling measure of dimension $d_0 \geq 2$. Pick any $q \in \mathcal{X}$ and draw $x_1, \ldots, x_n$ iid from μ. Pick any $\delta \in (0, 1/2)$. With probability at least $1 - 3\delta$ over the choice of x_i, for all $2 \leq m \leq n$, $\tilde{\Phi}_m(q, \{x_1, \ldots, x_n\}) \leq 6(\sqrt{(2/m)\ln(1/\delta)})^{1/d_0}$*

The proof is omitted due to space limitation. Using the above lemma, we conclude the following.

Theorem 3.6. *There is an absolute constant c_0 for which the following holds. Suppose μ is a doubling measure on $\mathbb{R}^d$ of intrinsic dimension $d_0 \geq 2$. Pick any query $q \in \mathcal{X}$ and draw $x_1, \ldots, x_n$ i.i.d. from μ. With probability at least $1 - 3\delta$ over the choice of data the tree fails to return nearest neighbor is at most $c_0(d_0 + \ln n_0)(\sqrt{(2/n_0)\ln(1/\delta)})^{1/d_0}$ where $n_0 \geq c_0 9^{d_0} \ln(1/\delta)$.*

The above theorem can be proved using Theorem 3.4 above and Lemma 12 from [9]. Note that, failure probability of Theorem 3.6 becomes an arbitrary small constant by choosing $n_0 = O\left((d_0 + \ln d_0)^{2d_0} \ln(1/\delta)\right)$. Moreover, for any small $\delta_1 \in (0, 1)$, the failure probability can be made at most δ_1, by constructing only $O(\log(1/\delta_1)$ independent randomized partition trees. Note also that, the analysis presented here can be easily extended to k-nearest neighbor search for $k \geq 1$ which we do not discuss here due to space limitation.

4 Experimental evaluation

We evaluate empirical performance of our proposed algorithm on eight different real datasets[2] of different dimensions as listed in Table 1. For each of these eight datasets, except

[2]The USPS dataset contains hand written digits. The AERIAL dataset contains texture information of large areal photographs [19]. The COREL dataset is available at the UCI repository [2]. After removing the missing data and keep only 66,615 instances. MNIST is a dataset of handwritten digits. This SIFT and GIST dataset contains SIFT and GIST image descriptors, respectively, introduced in [14]. The original dataset contains 1 million image descriptors. We used 500000 image descriptors from this dataset for our experiments. The 20 NEWSGROUPS and REUTERS are common text datasets used in machine learning and are available in Matlab format in [6].

USPS, 20 NEWSGRUPS and REUTERS, we chose 50000 data points for our experiments. To generate a nearest neighbor search problem instance, we randomly selected 5000 data points as query points and used the remaining 45000 data points to construct data structures for performing nearest neighbor search. After preprocessing, REUTERS, 20 NEWSGROUPS ([6]) and USPS dataset contain less than 50000 data points each. In all these cases, we randomly selected 5000 data points as query points and used the remaining data points to construct necessary data structures. Before presenting the actual results, we provide quantitative measurements that indicate the difficulty of nearest neighbor search for the eight datasets that we consider. Recall that the potential function $\tilde{\Phi} \in [0, 1]$, and higher the value of $\tilde{\Phi}$, more difficult the nearest neighbor search problem is, as this corresponds to the scenario where more and more data points are located in the vicinity of the true nearest neighbor of q. To estimate distribution of $\tilde{\Phi}$ from data, we generated 20 nearest neighbor search problem instances (each having 5000 random query points) as described above and computed $\tilde{\Phi}$ values for these 100,000 query instances. The histogram of these $\tilde{\Phi}$ values are plotted in Figure 6. Quantitatively, Figure 6 suggests that three out of eight datasets, namely, GIST, 20 NEWSGROUPS and REUTERS are harder compared to the remaining five other datasets for answering nearest neighbor queries and we will see later that experimental evaluations agree with this observation.

Dataset	No. of examples	Dimensions
USPS	9298	256
AERIAL	275465	60
COREL	66615	89
MNIST	70000	784
SIFT	100000	128
GIST	50000	960
20 NEWSGROUPS	18846	26214
REUTERS	8293	18933

Table 1: Dataset description

For our comparison, we used the LSH scheme proposed in [11] for ℓ_1 norm in non-Hamming space. We used the random hash function, $h_{U,b}(x) = \lfloor \frac{U^\top x + b}{w} \rfloor$ that a maps vector $x \in \mathbb{R}^d$ to an integer, where $U \in \mathbb{R}^d$ is a random vector whose entries are chosen i.i.d. from a *1-stable* (Cauchy) distribution, w is the width of projection, and b is a real number chosen uniformly from the range $[0, w]$. We tried different values for the number of projections k, namely $k = 2, 4, 8, 16, 32, 64$ and 128 and report the results for $k = 32$ and $k = 64$ due to space limitations. It was observed that small values of k resulted in very large number of retrieved points as compared to RP tree, thus, essentially doing a linear search while large values of k resulted in small number of retrieved points however, yielding a poor nearest neighbor search accuracy. The results for eight different datasets are presented in Figure 7, where each panel plots number of retrieved data points vs 1-NN accuracy. These results are based on 20 random instances of nearest neighbor search problems, where in every problem instance, 5000 data points were chosen at random as query points and the rest were used to build the data structures (RP trees / hash tables). The 1-NN accuracy of nearest neighbor search was simply calculated as fraction of the query points for which the true nearest neighbor was within the retrieved set of points returned by respective methods (RP trees or LSH). In each of the plots in Figure 7, the markers (legends) represent number of trees / hash tables (N) used. We used seven different values for N, namely, 2, 4, 8, 16, 32, 64 and 128. The placement of marker and the associated N values are easy to identify since accuracy increases with N.

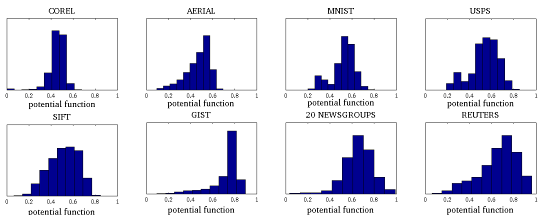

Figure 6: Histogram of potential function $\tilde{\Phi}$ for different datasets.

As can be seen from Figure 7, except for GIST, 20 NEWSGROUPS and REUTERS datasets, the proposed RP tree method clearly achieves better 1-NN accuracy with fewer retrieved data points as compared to LSH. Note that LSH as well as the proposed method do not work well

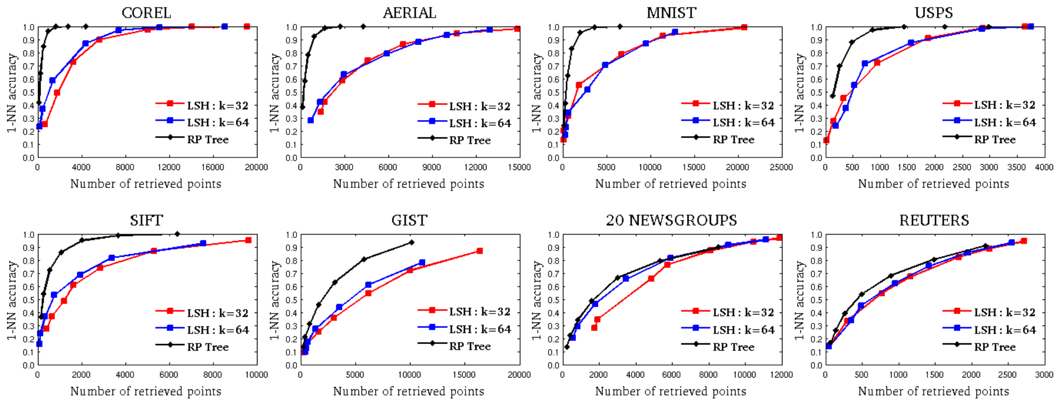

Figure 7: 1-NN accuracy vs number of retrieved data points.

for the GIST dataset. This can be attributed to the fact that majority of the query points have very high potential function values for the GIST dataset (Figure 6), indicating that performing nearest neighbor search on GIST dataset is a difficult problem. Performance of the proposed method is marginally better than that of LSH for the 20 NEWSGROUPS and REUTERS datasets, without clear distinct advantage over LSH. Moreover, in order to achieve close to 100% 1-NN accuracy for these two datasets, both the methods essentially scans the whole dataset. This phenomenon can most likely be explained due to the facts that : (a) for both datasets, majority of the query points have very high potential function values, and (b) these two datasets have very high dimensions compared to the other datasets and the number of data points used to construct the data structures is much less as compared to the other datasets.

Note that fiixing the parameters of LSH scheme, namely w, k, N, doesn't give an estimate of the number of retrieved data points and can vary quite widely based on the choice of w and k. RP trees on the other hand, give an easy-to-understand upper bound on the number of retrieved points, namely $n_0 N$, where, N is the number of trees and n_0 is maximum number of points in any leaf node. In addition, time complexity of nearest neighbor search depends on two factors, (a) time spent in reaching leaf node/hash bucket (number dot product computation required), and (b) time spent in distance computation for all points within the retrieved set. Under restricted setting, traditional analysis of LSH ([12, 1]) sets number of projection k to be $O(\log n)$. By construction, depth of our randomized partition tree is also $O(\log n)$. Assuming that tree/hash table data structures are constructed off-line, time required for a query point to reach the appropriate leaf node/hash bucket takes time of the same order. Because of smaller number of retrieved points, RP based method works faster with better accuracy. It was observed that changing different w, k, N combination can reduce the number of retrieved points in LSH and hence query time, at the expense of worse 1-NN accuracy.

5 Conclusion

In this paper, we have presented a new variant of randomized partition tree that can efficiently answer nearest neighbors queries in high dimensions when distances are measured using using ℓ_1 norm. We have presented analysis of its failure probability in finding true nearest neighbor and have shown that failure probability can be arbitrarily small constant when data are drawn i.i.d from a doubling measure and the query point is arbitrary. Our experimental evaluations suggest that the proposed method achieves better accuracy with fewer retrieved data points as compared to LSH using 1-stable distribution for ℓ_1 norm nearest neighbor search.

References

[1] A. Andoni and P. Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[2] K. Bache and M. Lichman. UCI machine learning repository, 2013. Available at : `http://archive.ics.uci.edu/ml`.

[3] A. Beygelzimer, S. Kakade, and J. Langford. Cover Trees for Nearest Neighbor. In *23rd International Conference on Machine Learning*, 2006.

[4] B. Brinkman and M. Charikar. On the Impossibility of Dimension Reduction in ℓ_1. In *44th Annual Symposium on Foundations of Computer Science*, 2003.

[5] B. Brinkman and M. Charikar. On the Impossibility of Dimension Reduction in ℓ_1. *Journal of the ACM*, 52(2):766–788, 2005.

[6] D. Cai. Text Dat sets in Matlab Format, 2009. Available at : `http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html`.

[7] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A Method for Simulating Stable Random Variables. *Journal American Statistical Association*, 71:340–3443, 1976.

[8] S. Dasgupta and A. Gupta. An Elementary Proof of a Theorem of Johnson-Lindenstrauss. *Random Structures and Algorithms*, 22(1):60–65, 2003.

[9] S. Dasgupta and K. Sinha. Randomized Partition Trees for Exact Nearest Neighbor Search. In *The 26th Annual Conference on Learning Theory*, 2013.

[10] S. Dasgupta and K. Sinha. Randomized Partition Trees for Nearest Neighbor Search. *Algorithmica*, 72(1):237–263, 2015.

[11] M. Datar, N. Immorlica, P. Indyk, and C. S. Mirrokni. Locality-Sensitive Hashing Based on p-Stable Dis. In *The 20th ACM Symposium on Computational Geometry*, 2004.

[12] A. Gionis, P. Indyk, and R. Motwan. Similarity search in high dimensions via hashing. In *25th International Conference on Very Large Databases*, 1999.

[13] P. Indyk. Stable Distributions, Pseudorandom Generators, Embeddings, and Data Stream Computation. *Journal of the ACM*, 53(3):307–323, 2006.

[14] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[15] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz Mapping into Hilbert Space. *Contemporary Mathematics*, 26:189 – 206, 1984.

[16] D. R. Karger and M. Ruhl. Finding Nearest Neighbors in Growth Restricted Metrics. In *34th Annual ACM Symposium on Theory of Computing*, 2002.

[17] R. Krauthgamer and J. Leel. Navigating Nets : Simple Algorithms for Proximity Search. In *15th Annual Symposium on Discrete Algorithms*, 2004.

[18] J. R. Lee and A. Naor. Embedding the Diamond Graph in ℓ_p and Dimension Reduction in ℓ_1. *Geometric and Functional Analysis*, 14(4):745–747, 2004.

[19] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieving of large image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[20] P. R. Rider. Distributions of Product and Quotient of Cauchy Variables. *The American mathematical Monthly*, 72(3), 1965.

[21] K. Sinha. LSH vs Randomized Partition Trees : Which One to Use for Nearest Neighbor Search? In *13th International Conference on Machine Learning and Applications*, 2014.

[22] S. Vempala. *The Random Projection Method*. American Mathematical Society, 2004.