GMDA: kd-trees and intrinsic dimension

Derwiche Nayef, Elazhari Lina

March 1, 2018

Abstract

In this project, we explore the properties of kd-trees and how they adapt to the intrinsic dimension of point sets in euclidean spaces. We have seen in class that random projection trees (RP trees) adapt to the intrinsic dimension in the sense that the diameter of the nodes shrinks with a speed related to the Assouad or Doubling Dimension only, and not to the ambient space in which the data is encoded. However, an approach presented in [9] may allow us to have at the same time the practical advantages of axis-aligned basic kd-trees, and retain the good properties of RP kd-trees. The simple idea is to perform a random rotation on the data before applying a basic axis-aligned kd-tree. We will discuss the approach, its claim, and evaluate it with experiments.

1 Initial Definitions and Problem Setting

1.1 Kd-trees

Kd-trees are a common data structure, introduced in [2] that are designed to be useful for elementary tasks such as nearest neighbors search for example. Those tasks are important primitives to many algorithms and problems, as diverse as for example Machine Learning, Computational Geometry, or Rigid Body Physics simulation.

Kd-trees are binary trees (although they can be generalized with quad-trees, octo-trees etc...), they recursively divide space into increasingly fine partitions. The basic kd-trees do so with axis-parallel cuts. The hierarchical structure built can then be used afterwards for the task at hand. Nearest Neighbor search for example, use its properties and different search strategies (defeatist, descending...) to make query of different accuracy and speed.

Algorithm 1 Generic Binary Partition Tree

```
1: procedure PartitionTree(dataset A)
2:     if $|A| \leq$ minLeafSize then return leaf
3:     $(A_{\text{left}}, A_{\text{right}}) \leftarrow$ SplitFollowingSomeRule(A)
4:     leftTree ← PartitionTree(A)
5:     rightTree ← PartitionTree(A)
6:     return (leftTree, rightTree)
```

Broadly speaking, the effectiveness of the tree is closely related to its ability to consistently reduce the diameter of the child nodes as the depth increases, because we benefit from points being in the same cell being close together, it is a bound on the error of representation of points in the cell by a point in this cell. If the space dimension is D, it will take D splits to divide the diameter by two, which will require 2^D data points. If D is high, it may pose a problem! And indeed, a lot of of the time, the data we want to work with is encoded in a relatively high dimensional space. However, often if not always, this encoding ambient space is quite expansive and gluttonous, and the data have an underlying structure, or intrinsic dimension, that is much more compact. Can we take advantage of this? Regular axis-aligned kd-trees can fail to do so, it has been empirically shown that it happens, and we can easily build dataset in which they consistently fail. What happens, is that they make "wasteful splits", that don't separate the data well.

Random projection tree, as described in [3], is an answer to this problem. The main principle is to make the split with random directions (projection on a random unit vector) and not split exactly at the median but add some randomness (the splits are "jittered"). The idea is that the randomness, both in the split direction and its position (the splits are "jittered") will, on average and with good probability, produce "good splits". To go further we must introduce the notion of intrinsic dimension.

1.2 Intrinsic Dimension

In our context, to define intrinsic dimension, we use the notion of Doubling/Assouad Dimension, which is defined as such:

Definition 1. Doubling Dimension: The doubling dimension, or Assouad Dimension of a set $S \subset \mathbb{R}^D$ is the smallest integer d such that for any ball $B(x,r) \subset \mathbb{R}^D$, the set $B(x,r) \cap S$ can be covered by 2^d balls of diameter r.

Data that are, on or near, a low-dimensional manifold in a high-dimensional ambiant space have low doubling dimension. Our hope is that structures like random projection trees can take advantage of it.

Similarly, we can also use Local Covariance Dimension, which is defined as such:

Definition 2. Local Covariance Dimension: Let μ be be any measure over $\mathbb{R}^D$ and let S be its covariance matrix. We say that μ has covariance dimension (d, ϵ) if the largest d eigenvalues of S account for $(1-\epsilon)$ fraction of its trace. That is, if the eigenvalues of S are $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$, then $\lambda_1 + \lambda_2 + \cdots + \lambda_d \geq (1-\epsilon)(\lambda_1 + \lambda_2 + \cdots + \lambda_D)$.

We will focus on Assouad dimension for theoretical part, but similar results have been established using local covariance dimension.

1.3 Random Trees and intrinsic dimension

Random Projection trees are very powerful because, as established in [3], they take advantage of the intrinsic dimension. Suppose we have a data set $S \subset \mathbb{R}^D$ of Assouad dimension d and let C be a cell in a tree built on this dataset. Given the Assouad dimension, the set can be covered with N balls, $N = O(d^{d/2}$, and we make "good splits", in the sense that we make splits that separates completely pairs of this covering balls that are of distance at least $(\Delta/2) - (\Delta/\sqrt{d})$, we can reduce the diameter in $\Theta(dlog(d))$ splits. In [3] they showed that random projection tree do so with good probability. As mentioned before, a similar result has been established with local covariance dimension. There is also similar results with other types of trees.

In the project, we discuss the approach proposed in the Vempala 2012 [9]. The simple idea is to perform a random rotation on the data before applying a basic axis-aligned kd-tree. We will call it "random kd-tree" or "random tree" in the rest of the report as opposed to "random projection tree" or "axis-aligned kd-tree".

Here is the main theorem of the paper that is central to the project. Its main merit is to provide guarantees very similar to those established with RP-trees to the random tree introduced in [9].

Theorem 3. ***Main Theorem :*** *Let $S \subset \mathbb{R}$ be a finite set with m points and doubling dimension d. Assuming $dlog(d) \leq c_0 n$, for the randomized k-d tree, with probability at least $1 - me{-}c_1 n$, for any cell C of the tree and very cell C' that is at least $c_2 d\log(d)$ levels below C, we have:* $\text{diam}(C' \cap S) \quad \leq \quad \frac{1}{2}\text{diam}(C \cap S)$

We will discuss the paper and its main result in a later section, first we will answer some preliminary questions.

2 Preliminary Questions

In this section, we will tackle first two sub-problems that are critical to the task.

2.1 Point Set Diameter Computation in Euclidean Space

2.1.1 Definition and Motivation

In this question we are asked to develop a procedure to compute the diameter of a point set, the diameter being defined as the maximum distance between any pair of point in a point set. The naive brute-force approach would be to compute all pairwise distances and then take the maximum. This is not practical, the complexity is $O(n^2)$. Also it would not make sense because we could as well just use this pairwise distance matrix to do any of the distance computation we need on the point set, and indeed the goal of using a kd-tree is to do better than this.

Better algorithm do exist, especially for 2D and 3D cases. But for higher dimensions it quickly becomes very difficult and complex. In [4] and [10] this difficulty is acknowledged, and they propose to cope with it by either using sampling to have a distance related, but not strictly similar, to the diameter which is the average furthest distance from a random point. Or, when it is simple to compute, use the diameter of the tree's cell. Those are also alternative definition of diameter that are used to prove analogous results about diameter reduction in [4] and [10].

Here, given that the question explicitly ask us to compute the diameter, we try a method described in [6] that is fast in practice for exact computation and even faster for good approximates.

The computation of the diameter is used mainly at two levels. First in the construction of the kd-tree to determine the random jitter scale. Second, when we have to evaluate the performance of the algorithm in adaptation to intrinsic dimension.

2.1.2 Possible approaches

The classic approach to diameter computation in $2D$ or $3D$ space starts with the computation of the convex hull in $O(nlog(n))$, such as quickhull [1] (which is worst case $O(n^2)$, but it is usually considered that its average complexity in reasonable settings is $O(nlog(n))$) and then use the rotating calipers algorithm which runs in $O(n)$ [8]. However for $d \geq 4$, despite being extensible, those approach become impractical because of the dimensionality curse.

Other provably optimal approach exist [7], but they too use data structures and algorithms that are too complex to be used in practice when the dimension becomes a little bit too high.

[6] propose an algorithm that is not worst-case optimal but performs very well in practice. It is based on the fast iterative computation of double normals (see next section) to reduce the search space.

Approximate approaches exist also, and may be very much relevant depending on the underlying application setting. The approach we chose can also be extended to provide an approximate result with the benefit of faster computation.

2.1.3 The approach we chose

The [6] approach to find the diameter Δ has been chose because it is fast in practice for exact solution and can also give an approximate solution even faster. The idea is to find double-normals of increasing length, which can be done very fast. A double normal is a segment $[p, q]$ such that p is the farthest point from q and q is the farthest point from p. $[p, q]$ then belongs to the convex hull of the set. Also, if $[p, q]$ is not of length Δ, and therefore is not a maximal segment, a maximal segment must have either of its endpoint outside of the ball B that have $[p, q]$ as a diameter. We use that property and other small lemmas, properties and tricks, to reduce the search space as much as we can before we must do an exhaustive search.

By making small adaptations, we can have an approximate search which gives us a result arbitrarily close to the true diameter and which in practice considerably reduce the computation time.

Performance is displayed in a later section.

2.2 Generation of a rotation

Another important part of the procedure we want to implement is to do a random rotation of the data in the euclidean space of dimension d, ie to chose a random orthonormal basis.

To do so, we first implemented a numerically stable and fast gramm-schmidt algorithm that we apply on d vectors that are sampled independently in hypercube with uniform distribution. It is worth nothing than the resulting rotation is not uniformly sampled (according to the Haar measure on the rotation group $SO(d)$). To have a uniform sampling, we should have used properties of nested subgroups and iterative methods that are significantly more costly in term of computation time with no clear gain for the task at hand. So we kept our first approach, which is quick and easy.

Please note that if the rotation procedure was a bottleneck in term of computation time, we could, instead of using our own implementation of gram-schmidt, have used a very fast QR factorization algorithm in numpy linear algebra library. But after optimization, our custom procedure was of the same order of magnitude, and entirely satisfactory for reasonable dimensions. Thus we used it to keep up with the spirit of the project which is to try out and learn as most as possible by doing it ourself.

Performance is displayed just at next section.

2.3 Performances

Performance comparison for rotation matrix computation:

Method	Speedup (Vanilla = 1)
gram-schmidt vanilla	1
fast gram-schmidt	11
QR-factorization	23

Performance comparison for diameter computation:

Method	Speedup (Brute Force = 1)
Brute Force	1
Malandain Boissonnat	9
Malandain Boissonnat Optimized	88
Malandain Boissonnat Optimized Approx ϵ=0.01	251
Malandain Boissonnat Optimized Approx ϵ=0.1	513
Malandain Boissonnat Optimized Approx ϵ=0.25	564
Malandain Boissonnat Optimized Approx ϵ=0.5	640

3 Random kd-tree

3.1 Discussion on the paper

The paper starts by stating that the classic axis-aligned kd-tree structure remains widely used, despite its vulnerability to the so-called curse of dimensionality and the fact that the main intrinsic dimension adaptability results use random projection trees (RP trees) or other trees more complex than axis-aligned kd-trees. Random Projection trees loose two very attractive properties of kd-trees: First, the traversal is more costly because instead of looking at one coordinate, one has to compare with a whole direction (by doing the projection) which is at least a factor D more costly. Secondly, the cells are no longer hyperrectangles, but polyhedras, which makes the computation of the distance of a point to a cell more involved. Indeed, as pointed by [9], Dasgupta and Freund did not provide an implementation of nearest neighbor search with the RP tree.

[9] establish a result (see Theorem 3) with kd-trees similar to the one established by [3] for RP-trees, on data that has been randomly rotated. This result can justify the usage of the following procedure: preprocess the data by rotating it and then applying an axis-aligned kd-tree. By retaining the intrinsic dimension adaptation of the RP-trees and the simplicity in implementation, transformation and usage of axis-aligned kd-trees at the same time, this is a simple, yet potentially a very useful idea. The overhead is not costly, one initial rotation on the whole dataset, and the inverse rotation on any point that results from a given query.

Also it explains why kd-trees work in practice on real-world data despite their limitations, if one makes the reasonable assumption that real-world data is coming from a randomly picked basis.

The paper used techniques and results similar to those used in [3] [4] [10], the main contribution and the main difficulty they had to overcome was the fact that the splits were not independent from one another (even if the basis is random, when depth $> D$, the split direction starts "cycling" through the base), unlike with RP trees.

The more important result they use, is the Johnson-Lindenstrauss Lemma [5], more precisely this version:

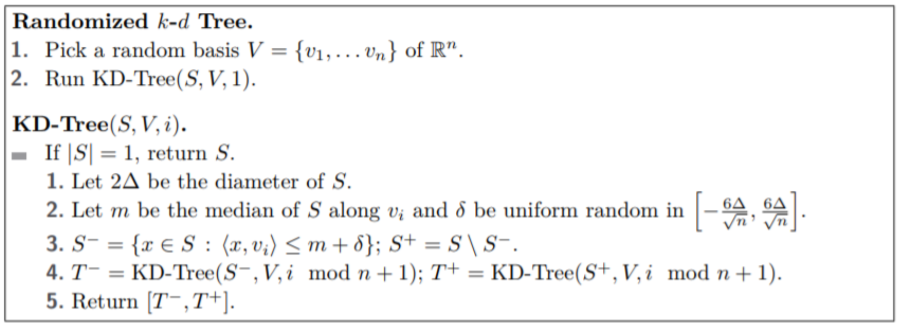
Lemma 4. *Fix a unit vector $u \in \mathbb{R}^n$, for a subspace V of $\mathbb{R}^n$ let $\pi_V(.)$ denote orthogonal projection to the subspace V. Let V be a randomized k dimensional subspace where $k < n$ and $\epsilon > 0$ then:*

$$\Pr(||\pi_V(u)||^2 > (1+\epsilon)\frac{k}{n}) \leq e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$$

$$\Pr(||\pi_V(u)||^2 < (1-\epsilon)\frac{k}{n}) \leq e^{-\frac{k}{4}(\epsilon^2-\epsilon^3)}$$

3.2 The implementation

For the implementation, we used the same algorithm used in the paper:



```
Randomized k-d Tree.
1. Pick a random basis $V = \{v_1, \dots v_n\}$ of $\mathbb{R}^n$.
2. Run KD-Tree($S, V, 1$).

KD-Tree($S, V, i$).
- If $|S| = 1$, return $S$.
    1. Let $2\Delta$ be the diameter of $S$.
    2. Let $m$ be the median of $S$ along $v_i$ and $\delta$ be uniform random in $\left[-\frac{6\Delta}{\sqrt{n}}, \frac{6\Delta}{\sqrt{n}}\right]$.
    3. $S^- = \{x \in S : \langle x, v_i \rangle \leq m + \delta\}$; $S^+ = S \setminus S^-$.
    4. $T^-$ = KD-Tree($S^-, V, i \mod n + 1$); $T^+$ = KD-Tree($S^+, V, i \mod n + 1$).
    5. Return $[T^-, T^+]$.
```

We only built the kd-tree through a function that returns it as a nested tuple object with numpy arrays or None object in leafs. This is because a complete implementation with a proper class and all possible queries was not needed for the project. As a follow-up it could be a good exercise to build a complete Class, with all its possible queries.

4 Experimental evaluation

4.1 The data used to evaluate the tree

To make our experiment we generated the following data:

- **Normally distributed points on an affine subspace**: Generate n points in a space of ambient dimension D on an affine subspace of dimension d. The affine subspace is aligned with the first axes.

- **Uniformly distributed points on an affine subspace**: Generate n points in a space of ambient dimension D on an affine subspace of dimension d. The affine subspace is aligned with the first axes.

- **Hollow Hypersphere**: Generate n points uniformly on the surface of a sphere, in a space of dimension D. Thus $d = D - 1$.

- **Robot Arm**: Generate data points encoding the position of a robot arm with a given number of joins. The root join is attached at the origin, and each joint can rotate through one given axis, so the intrinsic dimension and degrees of freedom are equal to the number of joins and the position is encoded in an ambient space of dimension $D = n \times \text{numberofjoins}$ with n the dimension of the space in which the arm exists.

- **Swiss Roll**: Generate points located on the famous swiss roll manifold. $D = 3$ and $d = 2$.

- **Picure rotation**: Generate data points by rotating a picture (encoded in grayscale). The intrinsic dimension should be 1, significantly lower than the ambient dimension which is equal to $x \times y$ the dimensions of the picture.

- **Smooth 1D Manifold**: The data we used the most in the experiments, it is a smooth 1D manifold described in [10] generated by sampling values uniformly in $[0, 2\pi]$ and mapping them with $M : t \rightarrow \sqrt{\frac{2}{D}}(sin(t), cos(t), sin(2t), cos(2t) \cdots sin(Dt/2), cos(Dt/2))$. With ambient space dimension D and intrinsic dimension $d = 1$.

4.2 The Experiments

Our experiments were done with the following protocol: we generated datasets with certain ambient dimensions D, and expected intrinsic dimensions d. Then we generated random kd-tree as in [9] for each dataset and we looked at how the average diameter of cell of the trees evolved as a function of the depth of the tree for the different datasets. According to the theoretical properties, the slope should be very similar for data of same intrinsic dimension d, and it should also be independent of the ambient dimension D.

We also did a comparison with random projection trees and axis-aligned tree, and an experiment with local covariance dimension.

4.3 Results and discussion

Results are plotted in fig.(1).

The most illustrative is (a), where the data is the 1D smooth manifold and the intrinsic dimension d is always equal to 1, but the ambient dimension varies from 2 to 32. We can see that, after a certain depth is reached, the slope of all the curves is very similar, evolves in parallel and is independent of the ambient dimension D.

What seems to depend on D is the fact that dataset of highest D the diameter starts to shrink later than for smaller D. This is because, intrinsic dimension may vary at different scales, and when at a large scale, where the is encompassed by the scale considered for instance, and the data is not on an affine subspace, the ambient dimension D dominates. Indeed, for d dimensional Riemannian manifold that have a certain curvature, we must go a at smaller scale to retrieve an intrinsic dimension d because Assouad dimension (and of course Local Covariance Dimension) rely on local "flatness", thus at larger scales, the curvature prevails and the manifold may "fill up" the space and cause us to consider that the intrinsic dimension is bigger than d and can even be as high as D. So what wee see see in (a), is that with bigger D, the tree takes some more time to arrive at the scale in which the intrinsic dimension d prevails over the curvature/ambient dimension. This relationship on intrinsic dimension, ambient dimension and curvature is very interesting and explored a little bit in the beginning of [10] in which they compare Assouad dimensions, Covering dimensions, Manifold dimensions with max curvature (Riemannian manifolds) and affine dimension.

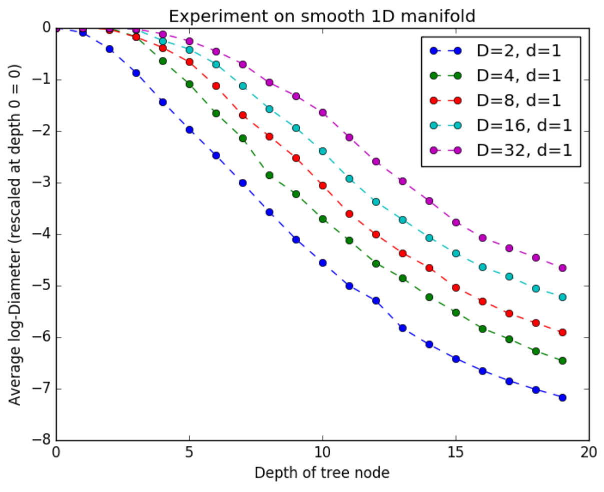
Indeed, in (b) and (c), we conducted similar experiments, but the data is on an affine manifold, thus without curvature, and the intrinsic dimension is d at all scales, therefore we are immediately, right at the root of the tree, in the "intrinsic dimension regime" and the diameters evolve similarly and have immediately the same slope, they are aligned.

What happen if we try different type of data, from different sources and ambient dimension but same intrinsic dimension? This is what we did in (d) and (e). We can see that again, we enter the "intrinsic regime" at different times, later for more complex data and data of higher D, but when we are in it, the slope depends on d only and is the same for data of similar d, independently of D.

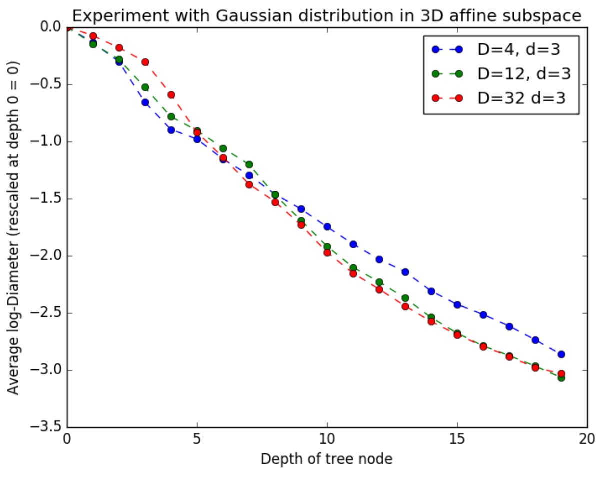
In (f), we vary simultaneously D, d and the data source, and we see clearly that the slope that are similar are those of the data of same d, despite having both different D and origin.

In fig. (2) we also compared the approach of [9] with a Random Projection tree and an axis-aligned kd-tree. We see that the axis-aligned tree fails on the axis-aligned affine subspace because it wastes a lot of its splits, it is an extreme example, but it shows how the randomness is useful in breaking those kind of unwanted symmetries and particular cases. The Random Projection Tree appears to be slightly better and detach itself after a certain depth, but the it remains comparable to the.

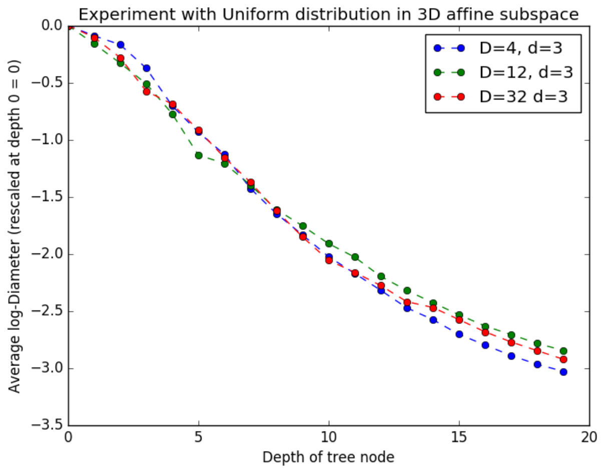
Finally, in fig. (3) we try estimate the local covariance dimension of a smooth 1D manifold with a little bit of noise by making samples at different scales. We see that at some scale the ambient dimension starts to prevail as we discussed before.



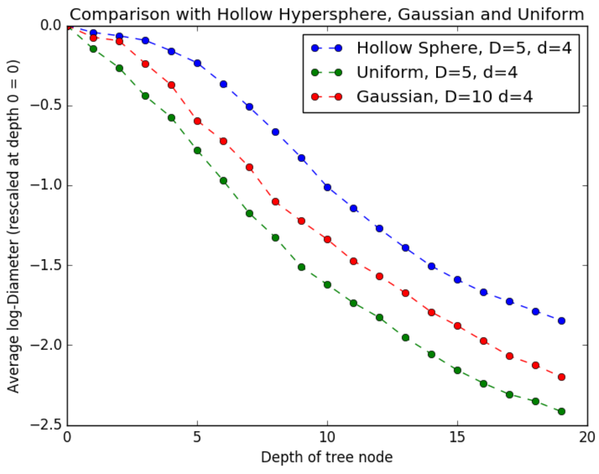
(a)



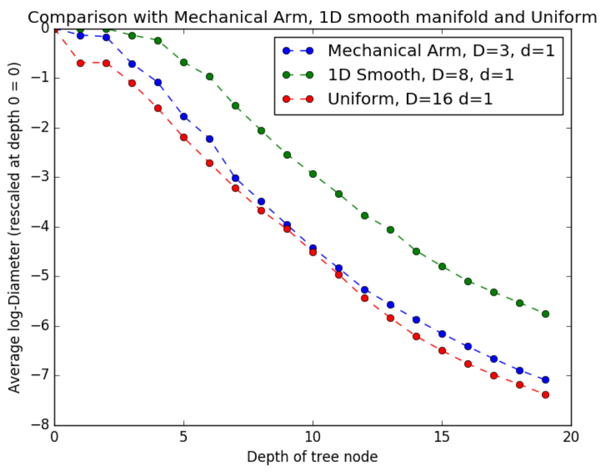
(b)



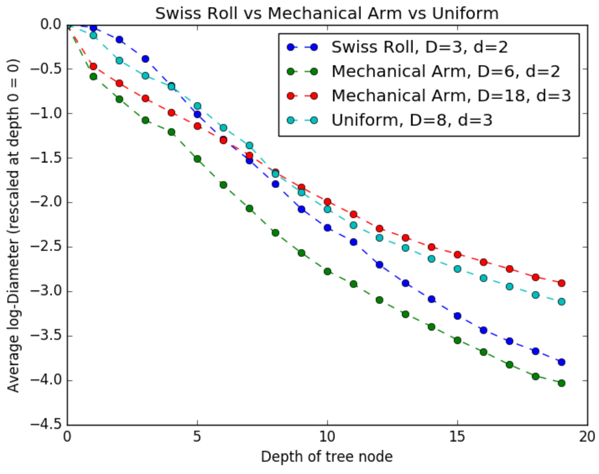
(c)



(d)



(e)



(f)

Figure 1: Intrinsic dimension experiments results

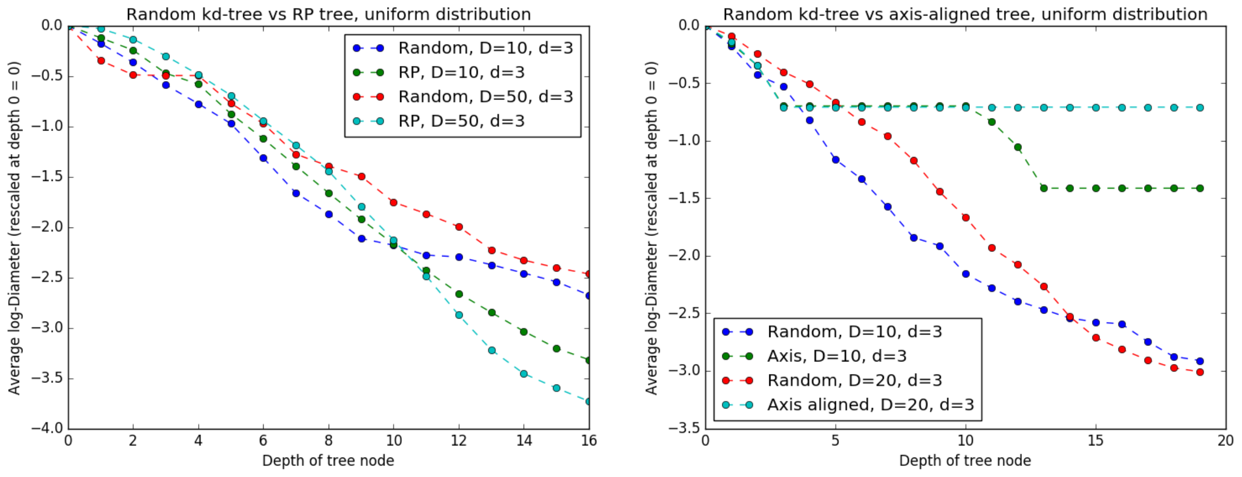


Figure 2: Comparison with other tree structures

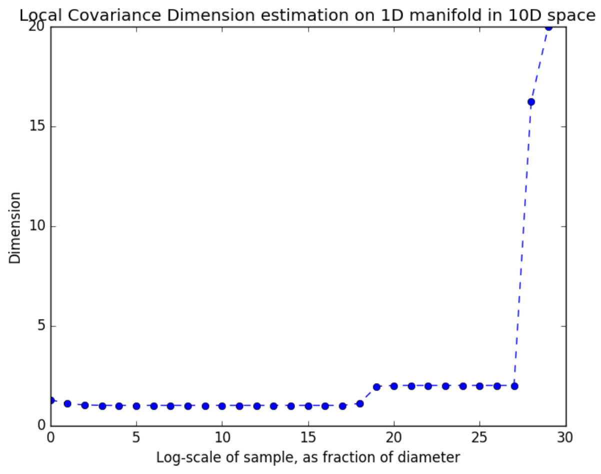


Figure 3: Local Covariance Dimension sampling 1D smooth manifold

5 Conclusions

In this project, we explored a simple yet powerful ideas, applying random rotation on data before applying kd-trees on it. It appears that the result established in [9], similar to the one [3] established from RP trees, holds very well in experiments. It provides a very nice and rigorous theoretical justification to the fact that kd-trees work well in practice. Additionally, if it does not in a particular case, we can now try to just apply a random rotation before looking for more involved techniques.

References

[1] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546. ACM, 2008.

[4] Yoav Freund, Sanjoy Dasgupta, Mayank Kabra, and Nakul Verma. Learning the structure of manifolds using random projections. In *Advances in Neural Information Processing Systems*, pages 473–480, 2008.

[5] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[6] Grégoire Malandain and Jean-Daniel Boissonnat. Computing the diameter of a point set. *International Journal of Computational Geometry & Applications*, 12(06):489–509, 2002.

[7] Edgar A Ramos. An optimal deterministic algorithm for computing the diameter of a three-dimensional point set. *Discrete & Computational Geometry*, 26(2):233–244, 2001.

[8] Godfried T Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, volume 83, page A10, 1983.

[9] Santosh S Vempala. Randomly-oriented kd trees adapt to intrinsic dimension. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 18. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[10] Nakul Verma, Samory Kpotufe, and Sanjoy Dasgupta. Which spatial partition trees are adaptive to intrinsic dimension? In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 565–574. AUAI Press, 2009.