Geometric Methods for Data Analysis

Paul Dechorgnat, Romain Boyer*

E-mail: paul.dechorgnat@student.ecp.fr,romain.boyer1@essec.edu

Abstract

Kd-trees have shown to be an insteresting data structures to store multi-dimensional datasets of large size. They provide efficient way to perform different machine learning tasks such as KNN based classification and regression.

They work as follows: at each step, we split our data according to the median value following a certain axis. During the following step, we will take the next dimension to perform the same operation.

Some refinements of this data structure implies different splitting values: one can choose to split the data according to a random fractile, to make an overlap of the value between the two children nodes. Some other version offers to split the data following a randomly chosen direction at each step or following the variance maximizing dimension.

At each step, we expect the diameter of the cell - or node - to decrease: we are indeed splitting our points in two, so the largest distance between two points in a given cell should decrease as we go deeper into the tree.

The question that we face here is how to make this diameter decrease quicker. One possible solution to this problem has been brought by Vempala:[4] randomly rotating the data before building the tree should lead to a decrease in the time needed to halve the original diameter of the dataset.

This is what we want to verify in this project.

Procedure

Main process

To verify the efficiency of randomly rotated KD-trees in diameter reducing, we will implement two different trees builders: one with random rotation and one without. Both will have a jittered split around the median such as suggested by Vempala.[4] Then, we will build those trees onto different datasets. We will run them several times in order to obtain some statistical significance. We will collect at each node the diameter of the cell and the depth of the node. We then will be able to see whether the diameter decrease more rapidly with a rotation.

For each dataset, we will run both algorithms 50 times. We will try also to modify the jittered split to see if this enhance or not the effect.

Code implementation

KD-tree building algorithm

For computing efficiency, we have implemented the code in C++. We have followed the following algorithm given by Vempala:[4]

Algorithm 1 - KD-Tree*(S,V,i,minCellSize)*

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

Iteration description

To compare so-called regular and rotated trees, we just have to chose the basis carefully: for regular trees it is going to be the canonical basis while for rotated trees, we have to

implement a random orthonormal basis.

At each step of the simulation, we generate a random orthonormal basis, build two trees, save the diameter for every node of both trees.

Diameter computation

After looking for an efficient algorithm to compute the diameter of a set of point, we could not find a solution that was obvious for us. We have decided to compute the diameter with a greedy solution. This means a lot of computation time as it is quadratic in the number of points of the dataset.

Chosen datasets

To perform this experiment, we have chosen to use 3 different datasets.

- MNIST dataset[1] : a collection of hand-written digits: they are represented as a 784 sized vector of scales of grey values. We only took the first 1000 digits to avoid killing the machine.
- Swiss roll[2] : a dataset of a swiss roll comporting 1600 vectors of three dimensions.
- Auslan[3] : Australian sign language representation in 22 dimensions. We took only the first 2000 points.

Those datasets are of very different shapes and we hope we can identify some shift within the behaviour of the trees as they should have very different intrinsic dimension.

Remarks

This study is provided with full code, input data and output data. It is available at *https://github.com/pauldechorgnat/GMDA-CS*.

Structure

- Input Data : input data is available in the input_data folder.
- Output Data : output data is stored in the folders results_*dataset_name*. Subfolders rotated_tree and regular_tree contain the corresponding data. Output data from the jittered split study are contained in the folder results_swissroll_jittered and subfolders correspond to the different types of trees and to the different values of the jittered split.
- Code : the code to run the experiment on the different datasets are in the *dataset_name*_main.cpp files. The script for the jittered split study is contained in the file swissroll_main_jittered.cpp
- Others : python scripts that were used to treat the output data.

Coding

As we are not C++ experts, the code may seem a bit raw. Some functions certainly available in some packages were hand coded. We tried to provide some comments to help understand it but it still might be hard to understand.

Results

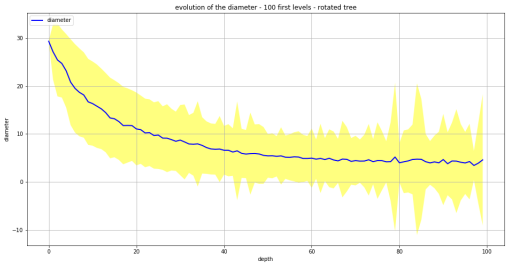
Randomly rotated trees vs regular tree

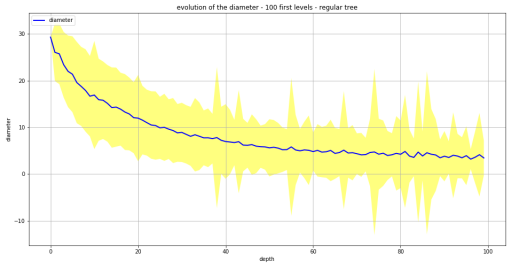
In this section, we will compare the diameter evolution of diameter for randomly oriented KD-tree against non-oriented KD-tree.

Swiss Roll Dataset

We can see that there is no clear effect of the random rotation of the data on the diameter evolution: the decrease at the beginning is more important but the effect is not very important.

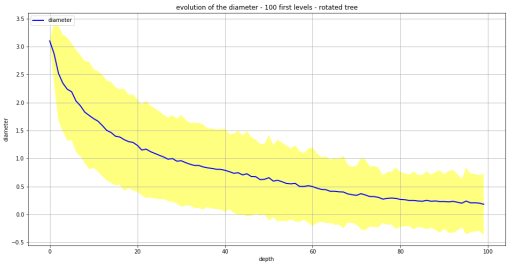
Here we can see the average diameter value on the first 100 levels of the trees in blue, and its standard deviation in yellow. Deepest nodes have been left apart as only very few tree are reaching those depths.

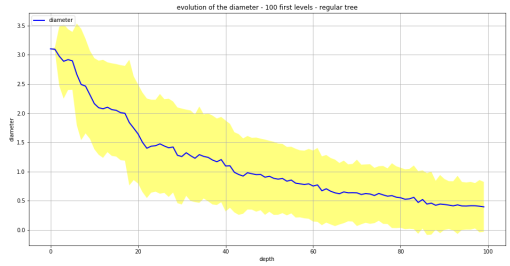


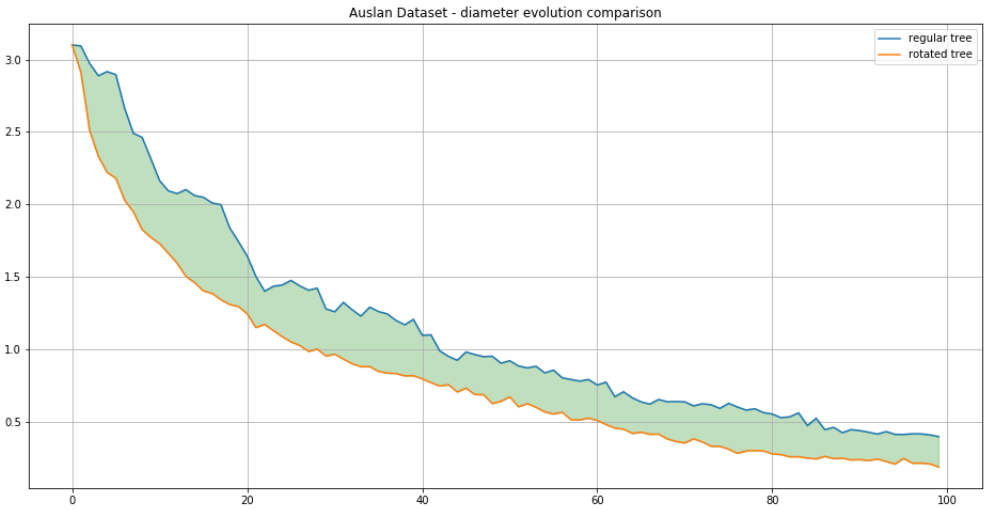


Auslan Dataset

Here the effect is more sensible. We can see a clear speed up in decrease when imposing a random rotation on the dataset.







MNIST Dataset

Unfortunately, we cannot get the results for MNIST dataset as it is way too big to compute the trees within a reasonable time: we estimated the computation time for 50 iterations to 40h, expected that the hardware can handle this amount of data.

Conclusion

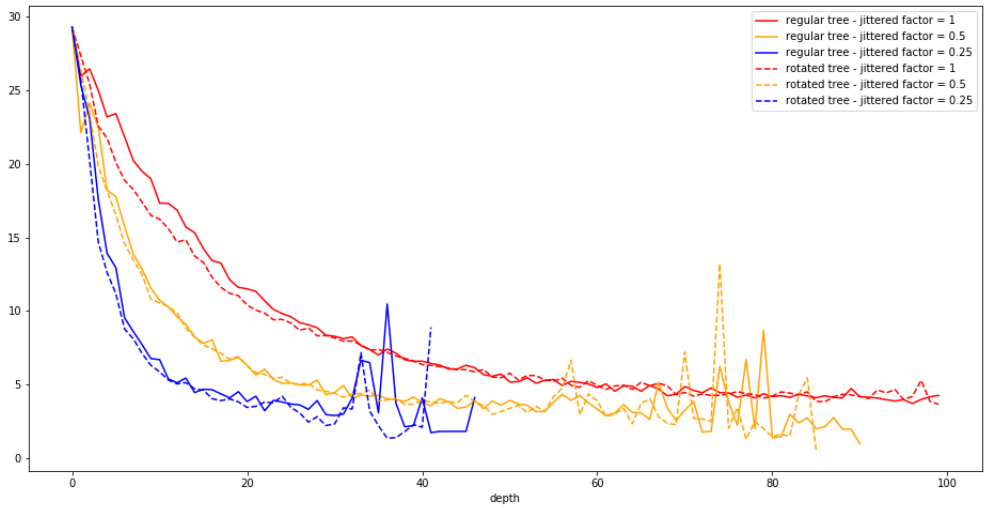
We can see that the effect announced by Vempala[4] can be seen even on those small datasets: it is not very visible on the Swiss Roll dataset but more obvious on the Auslan Dataset. Indeed, the intrinsic dimension of the Swiss Roll Dataset is smaller than the one of the Auslan Dataset. When we project the data onto randomly chosen direction and on the canonical direction, the difference is more important if the dataset is in a larger dimension space. We could expect that the drop in diameter wil be more important with the MNIST dataset as the dimension of the space it's in is even bigger than for the Auslan dataset.

Jittered split study

To study the effect of the jittered split we are going to use the Swiss Roll Dataset, although the effect of the random rotation has not proven very obvious. As the Auslan Dataset requires a computation time of 7h we cannot compute it on this dataset.
We are going to perform the same algorithm as Algorithm 1 but we will multiply δ by the following factors: 1, 0.5, 0.25 and 0.125.

Results

The following plot shows the evolution of the diameter given different jittered split factors.



We can see that the decrease of the factor means a speed up in the diameter reduction. It seems logical as the tree is therefore splitted in a more even way. We can see the effect talked about in the previous section: rotated trees decrease the diameter of the nodes quicker than the regular tree although it is not very sensitive.

We can see strange behaviours of the average diameter of nodes as depth grows. This is caused by the fact that some trees perform badly and take a lot of steps to split the data into small clusters. They will still have an important diameter for a deep depth and as good trees are done, the average diameter is not smoothed.

Conclusion and perspective

We can see that the effects predicted by Vempala[4] can be seen in large dimension datasets. It is more difficult to see it in smaller dimension dataset. A random rotation at the beginning of the construction of the tree helps the tree to decrease the diameter of nodes quicker. Moreover we have seen that this effect sticks when we decrease the jittered split effect.

This project has been limited by our computing power and the lack of optimization of some of our algorithms. Once improved, those algorithms could be run on other datasets (the

teapot dataset for example) and on larger slice of points. Moreover, we could run a finer study of the jittered split diminution effect. Finally to improve the statistical significance of the project, it could be interesting to run more iterations of the tree construction.

References

(1) MNIST Dataset : *http://yann.lecun.com/exdb/mnist/*

(2) Swiss Roll Dataset : *http://people.cs.uchicago.edu/ dinoj/manifold/swissroll.dat*

(3) Auslan DataSet : *https://archive.ics.uci.edu/ml/datasets/Australian+Sign+ Language+signs+(High+Quality)*

(4) S. Vempala , *Randomly-oriented k-d Trees Adapt to Intrinsic Dimension*, 2012

(5) Verma, Kpotufe, Dasgupta : *Which Spatial Partition Trees are Adaptive to Intrinsic Dim ension?*, 2009