Maxim Shelopugin

Final Project Report

Random Projection Trees and Their Applications

1. Paper Background

The paper by Omid Keivani and Kaushik Sinha “Improved Nearest Neighbor Search Using Auxiliary Information and Priority Functions” dives deep into the analysis of one of the most popular and profound machine learning algorithms - K-Nearest Neighbor classifier. The classification in this algorithm is performed by comparing the labels of the closest K neighbours of the training set to the query point. Training set is not used for the hyperparameter tuning here, since the model does not really learn anything, it is rather re-traversed with every query point. Due to such simplicity and high accuracy that it yields this algorithm attracts the attention of the researchers from various fields. It is used widely in Bioinformatics on non-sequential data, due to the generalizability and ease of use. A major drawback of KNN is the fact that the model has to perform a linear scan of the dataset for each query point - which is very slow when the dataset is big, or if there are many query points. To address that, this report focuses on the implementation of various optimizations of KNN algorithm, in particular : distance metric, representation of search space and searching methodology.

2. Method Proposed

The method proposed in the paper is to use Random Projection Trees as a data structure to represent search space for the algorithm. To navigate through such a complicated container, authors advise to use a special guided search function of their design. Both will be described below.

2.1. Random Projection Trees

As the name suggests, RPT is a tree which represents a particular part of the space as a leaf, and separates space according to some criteria. In essence, it randomly splits the space and does so recursively, until the size of the subspace becomes smaller than some integer *n*. This *n* is a static parameter, and needs to be tweaked from problem to problem. Authors of the paper used a specific value, to mask it as a constant, such that it does not add up to the time complexity of the construction, however given an always evolving nature of problems, such an important aspect of the algorithm cannot be left constant and needs to be picked dynamically with respect to some property of the dataset.

During the construction of the tree each branching generates a small *c* x *m* matrix which contains information for the future searches - instead of checking each leaf for the neighbor, only such matrix can be traversed and based on the matrix the prediction is done whether to take a left or a right path to the leaf. This prediction is of course approximate and does not guarantee an absolute optimal path to the child, hence when using this data structure it is assumed that the accuracy will suffer in return of higher retrieval speeds.

2.2. Guided Search

Guided search proposed in this paper is a sophisticated procedure which combines two methodologies - it builds a priority queue of nodes of higher importance in relation to the query node and it uses matrices of auxiliary information from unvisited nodes to represent the entirety of the tree through smaller subset. The result is a fast and accurate but difficult to follow procedure, which will be discussed below.

The algorithm takes a set of random vectors to build the prediction without any bias, and a parameter *t* - the amount of of tries that the search will make in order to retrieve the set of nodes. On every split point it projects the query node on the random set of points, and based on its position in regard to the median separator of the split it selects a left or a right subtree. Once the path is chosen the priority score is calculated, and the subtree is put on the priority queue.Once the leaf is hit, the leaf represents the most probable subspace where the querie's true neighbour is situated. If the parameter *t* is set to be higher than 1, we can search for the new leaf by extracting the node with the highest priority and reaching the most probable leaf again. In the end, once *t* iterations are reached, we can unionise these *t* nodes, into a set which with high probability contains the true closest neighbour. Of course increasing *t* increases the accuracy, but increases the time complexity in linear manner as well.

To represent the unvisited paths each subtree as well contains a small matrix of auxiliary information, with predefined dimensions of c x m. Each matrix consists of the c closest points to the median in the subtree, and is used for the proxy to keep the track of the population in the leaf. While traversing the tree such points are used further to determine the closest points to the query and are used in union with the leaf node vectors for the future computation of the nearest neighbor. This portion of the algorithm is not as significant on its own, however with the use of prioritised search, these proximity matrices build a bigger and more representative set of close point within the whole tree, as such it not only prioritizes some leafs over others but also contains the closest vectors in all the unvisited nodes, such that the final set has even higher probability of containing a true neighbors subset of the whole dataset to a query point.

3. Implementation Method

Everything that is discussed further in this paper was implemented from scratch using solely numpy as a vector manipulation library. Methods from the original paper were adapted and improved in places where the improvement seemed reasonable. Some mistakes that were made in the original paper were addressed and corrected.

3.1. Random Projection Trees

Since the Random Projection Tree is a data structure, it is reasonable to create a class which would store all the internals of the tree, and instantiate it only once per dataset. This provides the ability to do a hard calculation once, and retrieve necessary subsets in faster manner later, without the need to reconstruct it.

To build the tree, a random vector of the dataset's dimensionality is needed. Once it is generated, we have a direction in high-dimensional space which separates the dataset into two, not necessarily equal parts. To equalize such a distribution, each point in the dataset needs to be projected on this random vector - now every point is represented as a single coordinate on the vector so it is easier to manipulate. To create a precise boundary we can choose the median on this vector - it will separate all the projections into two equal parts, and once they are separated we can draw a vector perpendicular to the random vector which goes through the median. This ensures that the whole space is separated into two equal parts. This is done recursively until the desired leaf size is reached, which in this implementation is set to 100.

To widen the possibilities of selecting good neighbours, each selection of the median, namely a root of a subtree is accompanied with a small matrix which stores the *c* closest points to the median point of such a subtree. This parameter *c* was chosen to be 10, hence 10 closest points represent the sub-tree. This results in the ability of retrieval of many small subtest which are a good representation of a whole subtree. Once the tree is constructed in such a manner, it can be used to search through.

3.2. Guided Search

As it was discussed before the search consists of two major parts - a priority queue and a set containing all the representative points. A priority queue is a placeholder for the most promising candidates which lead to the leaf of the tree. The queue gets both sub-trees once a root is visited. Each sub-tree gets its priority by projecting the target point on the matrix of the sub-tree, which results in the the set of distances to closest points. Average this set to get the priority score. Such a distance metric sorts the sub-trees in the ascending order of relevance. Once the sub-tree is processed until the leaf node, the programmer has a choice to proceed with another sub-tree such that a new traversal generates a new leaf in the end, and the resulting union of leafs creates the set of possible solutions. In this implementation the amount of iterations was capped at 3 - hence only 3 leaf nodes are considered.

To combat the mistakes in such a calculation, it is decided to use the same auxiliary information matrices from the rejected sub-trees as the proxies for their according vector sets. In essence, if the tree is not considered for the traversal its matrix is stored in the set of representatives. By the end of traversals, this set contains all the candidates from rejected paths and summarizes the entire untraversed tree. This of course increases the performance of the algorithm incredibly, but again adds the parameter dependence, since each matrix needs to contain a specific number of representatives and such a number depends both of the problem and influences the time complexity.

3.3. Optimizations

As it is known, K-NN classifier is very sensitive to small perturbations in space so to increase the accuracy of a well formed data structure it is decided to work directly with a distance metric which is usually left as a vanila MSE. To tackle such an issue antialiasing was implemented as a modification to the query point. Antialiasing is a technique to alias the edges of the target image to make it more flexible to smaller mistakes between true neighbors, now the query vectors do not have sharp edges, and such blurriness gives an opportunity for a mistake in dislocation of the true neighbor, providing a leverage to achieve higher accuracy. The discussion of the results will be reported below.

3.4. Datasets

To test the algorithm with its optimizations it was decided to use a variety of experimental and real-life datasets, each of which will be discussed below.

3.4.1. MNIST

As a baseline for the algorithm performance it is decided to test it on a popular and contained dataset MNIST. It is the dataset of monochrome handwritten digits, which contains 60,000 examples for training and 10,000 examples for testing. Each frame is 28x28, but was flattened for the consistency across all the datasets.

3.4.2. SVHN

Another popular and large dataset which provides a digit recognition, but in more real "Street View house number". This dataset contains 73,257 training vectors and 26,032 test vectors. Each digit here is represented as 32x32 matrix with 3 channels representing colors. Again, the set was flattened, and the channels were averaged for the monochrome effect - color does not add any information here.

3.4.3. Heart Disease

To demonstrate the ability of the algorithm in biomedical field, it was decided to use the Heart Disease data set. This data set contains 303 different persons, each having 76 distinct attributes like sex, age etc. The last column of the dataset is the heart disease diagnosis. Since the dataset does not contain a specific test set, it was divided into two sets, and the RPT was constructed on the first 250 vectors of data, and the last 53 vectors were used for the testing purposes.

3.4.4. Mice Protein Expression

Since the previous real-life dataset was not as large, it is decided to expand the algorithm to a bigger data-set. The Mice Protein Expression dataset contains 1081 test subjects, each of which contains 77 real-valued measurements of the expression of the specific protein. The goal is based on these measurements to classify the mice into classes of learnability.

4. Results

The results of the algorithm ran on all of the datasets is shown below. It is seen that most of the RPT constructors hinder the accuracy, however are ran a lot faster.

	MNIST	SVHN	Heart	Mice
KNN	0.9691	0.4821	0.5849	0.912
RPT	0.8788	0.3526	0.5849	0.912

KNN time	5199 sec	79256 sec	0.11 sec	2.33 sec
RPT time	72 sec	738 sec	0.12 sec	1.6 sec

Below is shown the experiment with a antialiased data. It is seen that the accuracy increased almost everywhere after applying such a small optimization.

	MNIST	SVHN	Heart	Mice
KNN	0.9721	0.5120	0.5903	0.9060
RPT	0.8831	0.3695	0.6037	0.9072

5. Bioinformatics Applications

Since Bioinformatics field is populated with hard problems, and every one of them needs a special insight to work with, a more general algorithm like K-NN can be of a great use. Especially if the data is not sequentially dependent, such a simple algorithm can lead to great accuracy. To help such algorithms to achieve higher accuracy, it is possible to add a custom distance function which would describe true neighbours in more precise way. This experiment shows that the real biomedical data can be successfully classified by the RPT algorithm, and after applying a simple distance enhancement metric it can actually increase the performance even further.

6. Insight

Since the RPT data structure makes the K-NN classifier an actual learning model, its advantages and disadvantages need to be considered when applied. It takes a long time to be built if the dataset is large, and if the amount of query points is not considerably large, it would be much better to proceed with a vanilla K-NN, since the accuracy in this instance would be better. If however, the amount of test data is sufficient, RPT provides a far superior speed and by constructing a forest of such trees or by manipulating the parameters a higher accuracy can be achieved. On another hand, using a special kind of distance metric can greatly increase the accuracy of any similarity based algorithm, so the knowledge of the domain of the problem would help in building a more relevant heuristic. As it is seen, there are tradeoffs everywhere, and each situation needs to be addressed in a special manner.