RPT with Guided Search

By Maxim Shelopugin

Problem Statement

- K-Nearest Neighbor Search
  - Given a set of vectors, classify a query based on relationship to the set
  - Linear search through everything
  - Is simple and popular
- Bioinformatic Models need to be tuned
  - Not so many experts in the field
  - General Use algorithms are hence preferable
  - Many problems can be modeled as non-sequential data

Random Projection Tree Insight

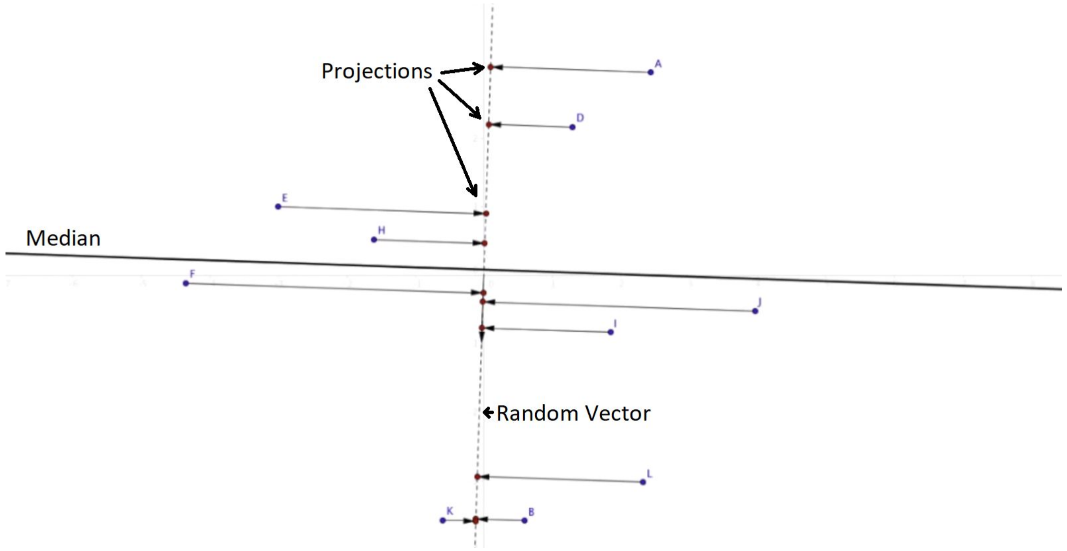
- Linear Time is not always feasible
  - Usually dataset are very big
  - Most models train only once, and predict fast after that
  - Knn needs to re-train every time
- RPT
  - Partition the space onto subspaces
  - End up with a random multi dimensional subspace
  - More focus on the speed rather than on accuracy
  - Is the most popular partition algorithm for KNN

RPT advantages and disadvantages

- A very fast search through the tree
  - Now the traversal is done in O(l) - size of the leaf
  - To find the leaf O(log(n/l)) - a very small value
- Is very sophisticated to construct
  - Every subspaces is constructed through O(n^2) comparisons
  - Once the class is built, does not have to be rebuilt again
  - Now KNN “learns”
  - Has many parameters
- Navigation through leaves
  - A creative field
  - Many different ways to navigate based on some heuristic
  - Most heuristics increase the time complexity

Random Projection Tree Construction

- Create a random vector
- Project all points on it
- Pick the median of projections
- Use median as a separator



- Recursively call until the target size is reached

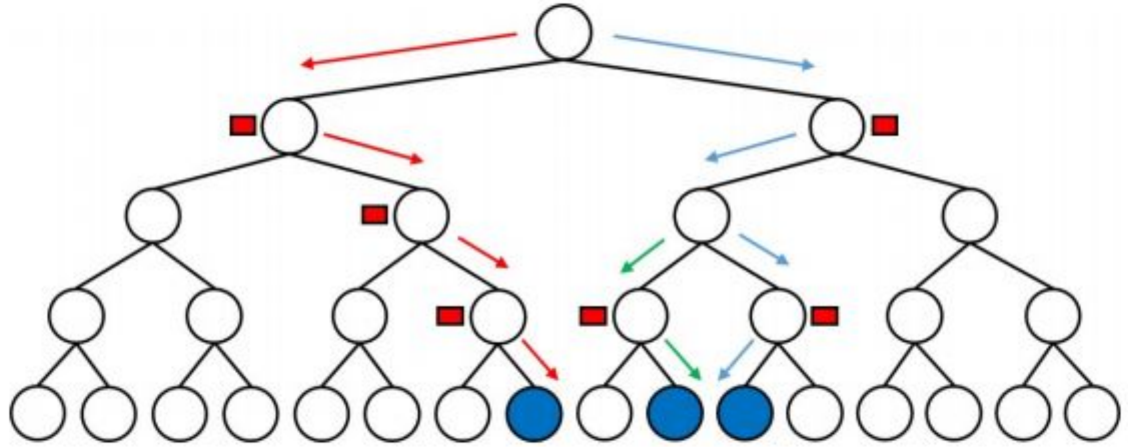
[1] https://www.youtube.com/watch?v=b9fg9Du8d1I

Guided Search

- When traversing the tree - keep the track of best subtrees
  - Each subtree has a matrix with closest points
  - Calculate the distance to the candidates in the matrix
  - Use distance as a priority
  - Keep all the candidates in the priority queue
  - Traverse through selected amount of candidates
- If the leaf is chosen - keep the track of closest point
  - Build a growing subset of close points
  - Union such a subset with the highest priority leaf
  - Results not only in best leaf, but a whole tree of close points

Resulting Algorithm

- Get k closest leafs
- Get the proxis of the whole unvisited tree
- Linear scan through the resultant subset



[2] Improved nearest neighbor search using auxiliary information and priority functions

Distance Metrics

- Typical distance metric is a Mean Squared Error
- Depending on data very different optimizations follow
  - If data is sequential - optimizations might harm
  - If data has spatial dependencies - might help a lot
  - Usual issue is an alignment
- In this implementation used smoothening
  - If vector contains 0 - smoothen it with the value next to it
  - Helps a lot in image processing
  - Does not help at all in most other domains

[3] https://subscription.packtpub.com/book/game_development/9781783981144/10/ch10lvl1sec69/adding-anti-aliasing

Dataset

- This implementation used MNIST and SVHN
  - Great playground for k-nn based algorithms
  - Both are free
  - Both have lots of examples
- To test on Biological data, Heart Disease and Mice Protein Expression
  - It was not extensive, rather proof of concept
  - A lot smaller than MNIST and SVHN
  - In a sense harder to pinpoint similarities
- Tests were done both on original and smoothened data

Results

	Original	Smoothened	RPT	Smoothened RPT
MNIST	0.9691	0.9721	0.8788	0.8831
	5199 seconds	5250 seconds	72 seconds	78 seconds
SVHN	0.4821	0.5120	0.3526	0.3695
	79256 seconds	82053 seconds	738 seconds	794 seconds
Heart	0.5849	0.5903	0.5849	0.6037
	0.11 seconds	0.15 seconds	0.12 seconds	0.16 seconds
Mice	0.912	0.9096	0.912	0.9072
	2.33 seconds	2.45 seconds	1.6 seconds	1.8 seconds

Conclusion

- As it is seen the accuracy suffers
  - Parameters can be tuned, but the time complexity will increase
  - More sophisticated distance comparison might be needed
- Very fast
  - On mnist as much as 70 times faster than vanila knn
  - Not as striking difference on smaller datasets
- When there are more candidates to be classified, a better choice
  - Build model only once, retrieval is fast
  - Can be built as a forest
  - If the model is too big and candidates set is small, better to use vanilla method

Questions?