Spotify ANNOY/Nearest Neighbors

Eric Hunzeker[#1], Brian Choromanski[*2]

[#]*Department of Computer Science, University of Pittsburgh*
6127 Sennott Square, University of Pittsburgh, Pittsburgh, PA 15260, USA
[1]erichunzeker@pitt.edu
[3]Choromanski@pitt.edu

***Abstract*— The N Nearest Neighbors algorithm and specifically Spotify's ANNOY algorithm provide a simple machine learning solution to reducing repetitive or similar calculation This paper reviews how the N Nearest Neighbors algorithm works, identifies the concepts and characteristics of the algorithm, and finally describes how the algorithm has been adapted by Spotify.**

***Keywords*— Nearest Neighbors; Spotify ANNOY; Pattern Recognition; Lazy Learning, Algorithms;**

I. Introduction

With applications like Spotify, performing calculations on users can be very expensive and repetitive. Specifically, music recommendations should take a lot of computational power to perform. However, Spotify has developed their own implementation of the K Nearest Neighbors, or kNN algorithm, to solve this problem. Between the algorithm, Hadoop, and other cloud computing tools, Spotify has provided a scalable solution to mass music recommendation on their platform.

The development of Spotify's ANNOY algorithm is not the fastest kNN algorithm, but they believe that it is the best implementation of kNN to integrate with the platform. The major advantage of ANNOY is that indexes are stored as static files, and can be passed between CPUs during concurrent programming. This allows computation of indexes to be minimized, while also reaping the benefits of parallelism.

II. Related Work

A. Instance-Based Learning

ANNOY is a implementation of the K Nearest Neighbors algorithm, which is an instance-based learning algorithm. Instance-based learning is also commonly referred to as lazy learning - where the generalization of training data is delayed to the point of a query. This is separate from eager learning, where the algorithm makes a generalization of the training data before a query. The lazy learning aspect of KNN makes it among one of the simpler machine learning algorithms.

B. Feature Vectors

In machine learning algorithms, feature vectors serve the purpose of classifying data into an individual measurable property. ANNOY stores these feature vectors as static files so that different processes can share the same data. This reduces the computation power and time to recompute vectors on each process.

C. Distance Metrics

One of the most essential parts of the KNN algorithm is computing the distance between vectors. The algorithm to determine distance varies based on the vector's type. If the vector is discrete, then a hamming distance can be used. Otherwise, if the vector is continuous, the euclidean algorithm is most often used. Hamming distance is the number of substitutions needed to make to turn one discrete value into the second discrete value. The euclidean distance refers to the straight line distance from one point to another. This is the metric that determines the classification of a data point. In the ANNOY algorithm, the metric can be "angular", "euclidean", "manhattan", "hamming", or "dot" [1]. Hence, there will be a different distance algorithm used depending on the metric that is selected.

D. Parameter Selection

Parameter selection refers to the problem of selecting the 'k' part of KNN. This number determines how many of the closest items that a new item will be classified with. Higher k values will produce smoother, more defined boundaries across different classifications [3]. The value of k should also be kept an odd number so that there is never a tie between classifications of two equidistant points.

III. Implementation of ANNOY Based on KNN Algorithm

A. Starting Point

The problem starts with high dimensional point sent - or a large quantity of points present in a high dimensional space. The goal from here is to be able to build a data structure that can be queried for any item's nearest neighbor in sublinear time [7]. For Spotify in particular, there are millions of users and millions of tracks. Each user has a listening history composed of a different set of tracks, so users with a larger intersection of tracks will be dimensionally closer to each other. This is represented in Fig. 1 as a 2 dimensional representation of points in a high dimensional space.

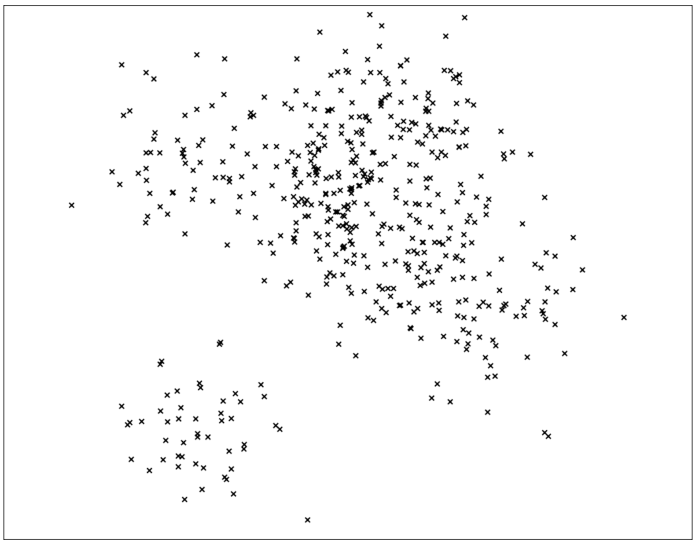


Fig. 1 Example of items scattered in high dimensional space

B. Binary Trees

Annoy splits the space recursively - splitting the plane equidistant from two randomly chosen points at each call. The split is by a hyperplane, which reduces the dimensionality at each split. The splits keep occurring until there is at most k items left in each node. While the splits occur, a binary tree is built up, with the nodes containing each item that they have in them. After the binary tree is created, it can be searched in logarithmic time. The only problem with the binary tree is it's likelihood to separate items that are close to each other into separate planes. This is solved by adding binary tree nodes that are "the wrong side" of the plane of the item that is being searched for to priority queues. With this approach, the parameter of how far away you are willing to look in the binary tree can be varied to produce either more refined or more broad results. The resulting representation of planes will resemble Fig. 2, with each plane consisting of no more than k closest items.

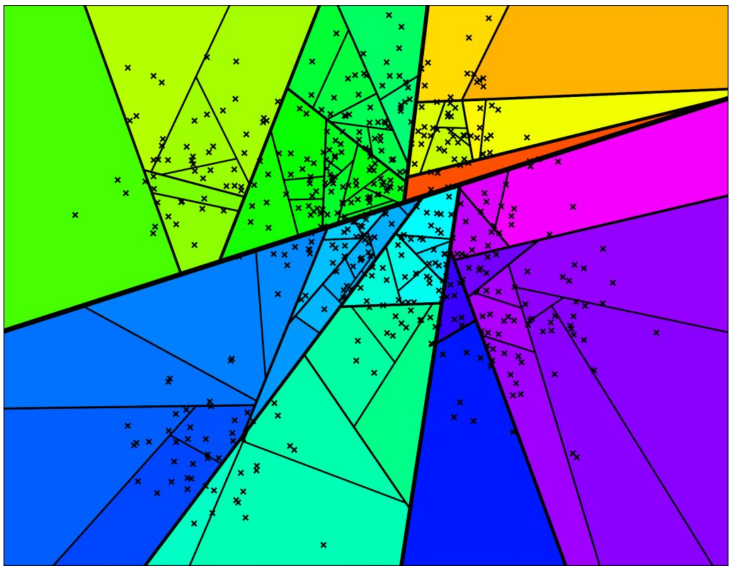


Fig. 2 Example of items scattered in high dimensional space

Each of these planes is represented as leaves on a binary tree, with each random series of splits comprising a separate binary tree. The collection of binary trees from recursive hyperplane splits make up the forest that is then searched from with a priority queue.

C. Solutions to Near Items Being Separated During Splits

One solution is by adding binary tree nodes that are "the wrong side" of the plane of the item that is being searched for to priority queues. With this approach, the parameter of how far away you are willing to look in the binary tree can be varied to produce either more refined or more broad results. Another approach to this is to build a forest, with each tree representing a set of random splits. After the forest is built, all of the trees are searched with a singular priority queue, and the union of leaf nodes will signify a close approximation of an items neighborhood, or k nearest neighbors. Sometimes, the forests' union of leaf nodes will amount to a close, but not completely correct representation of the nearest neighbors. This is shown in Fig. 3 where the majority of points nearest to the center of the circle are captured within the polygon, but there are still a few missed points. This is why the algorithm is called “Approximate Nearest Neighbors”, as it sacrifices minor accuracy for performance.

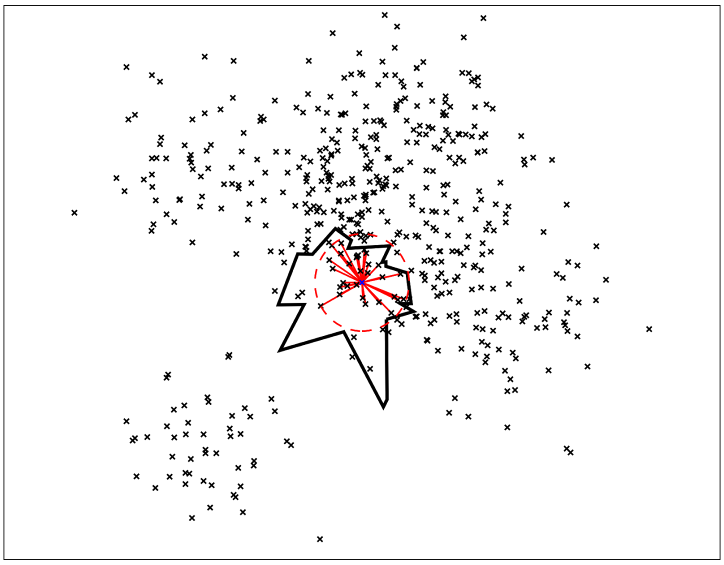


Fig. 3 Example of approximated neighborhood

D. *K Nearest Neighbors*

Spotify uses ANNOY to recommend music to people with this representation of data as vectors and an algorithm to call k nearest neighbors to a point. The usage of this library is quite simple from here, only requiring a query point and a k value. The item that is used to find the nearest neighbors can either be an index or a vector that exists in the forest. Other parameters that are to be specified are n, search_k, and include_distances. N refers to the amount of neighbors, search_k allows specification of number of nodes to be searched, and include_distances is a switch to toggle the return of either just nearest neighbors, or a tuple of nearest neighbors along with each distance. Between the three parameters, the query is very simple and lightweight. After the forest is built, it is mmapped to memory from static files. This allows for parallel programming and the sharing of static index files between different processes.

IV. ANNOY Cloud Computing and Performance

All paragraphs must be indented. All paragraphs must be justified, i.e. both left-justified and right-justified.

A. *Performance Against Other KNN Implementations*

Fig. 4 marks the performance of ANNOY against other KNN algorithms based on queries/accuracy. One of the features of ANNOY is being able to give up some accuracy for speed, which is shown in the upper left corner of the graph. Annoy is competitive with other KNN implementations especially at higher precision, which is ideal for it’s application.

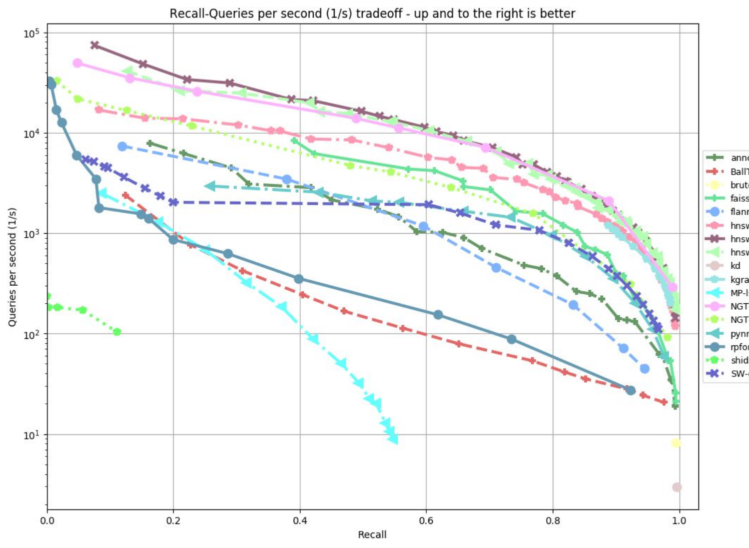
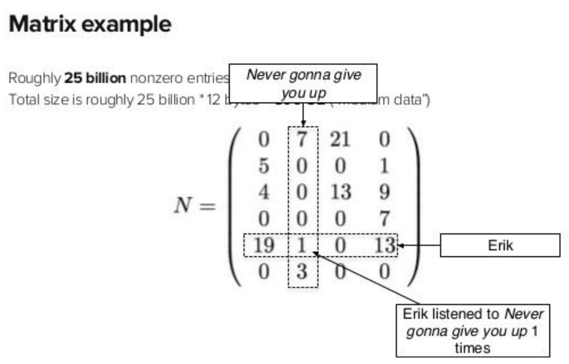


Fig. 4 Performance of different KNN algorithms

B. *Map/Reduce*

One of the heaviest parts of the ANNOY algorithm is completing the matrix factorization to vectorize items. In Fig. 5, there is a sample matrix to demonstrate the interaction between users (rows) and tracks (columns). The job of the matrix factorization is to organize each user into its own vector so that consistent computations can be made between different users. This process features the usage of both Map/Reduce functions, as well as Hadoop. Fig. 6 shows how the tracks that are marked as ‘1’ for any given user are mapped to the respective user. Fig. 6 also shows the reduce step, where the resulting map of users and their streamed songs are then reduced to vectors.


Matrix example
Roughly 25 billion nonzero entries
Total size is roughly 25 billion * 12 b m data")
Never gonna give you up
Erik
Erik listened to Never gonna give you up 1 times

$$N = \begin{pmatrix} 0 & 7 & 21 & 0 \\ 5 & 0 & 0 & 1 \\ 4 & 0 & 13 & 9 \\ 0 & 0 & 0 & 7 \\ 19 & 1 & 0 & 13 \\ 0 & 3 & 0 & 0 \end{pmatrix}$$

Fig. 5 Matrix representation of user space

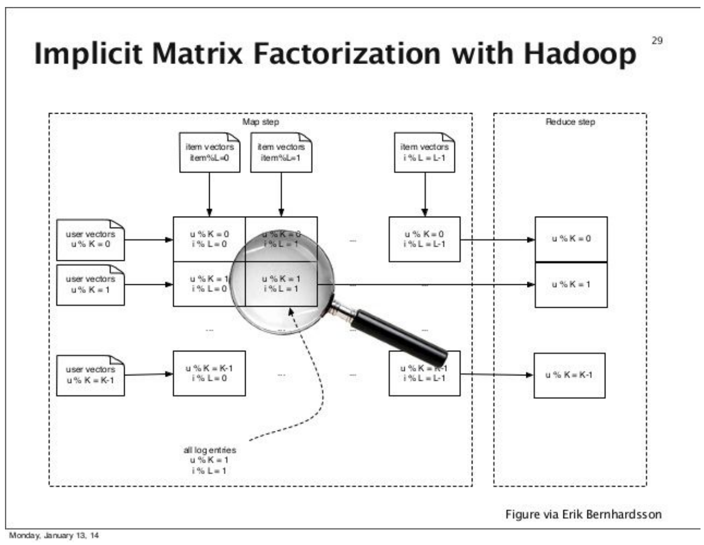


Fig. 6 Matrix Factorization using Map/Reduce

This process takes advantage of Hadoop by being able to make as many computations at once as one's budget allows. This means that for every user that needs to be vectorized, Hadoop can sort out the work to however many CPUs are needed to complete the task in a reasonable amount of time. Once the matrix factorization is complete, distances between items can be computed with a simple dot product. This computation is very simple and lightweight, making it ideal for high volume queries in applications such as Spotify.

V. Conclusions

This paper realizes the design of the K Nearest Neighbors algorithm and specifically Spotify's ANNOY algorithm. By using a lazy machine learning algorithm, the number of calculations for music recommendations is greatly reduced. ANNOY only needs to calculate indexes for items once, then can share static index files across CPUs to maintain parallelism. By using static indexes, each CPU doesn't have to recompute items which are already indexed. There is also efficient work being done to generate the indexes that are stored, involving a series of recursive splits using hyperplanes and the resulting forest of binary trees. This method is ideal for scalability, speed, and the assumption that exact precision is not necessary desired.

References

[1] Spotify, "spotify/annoy," GitHub, 12-Dec-2018. [Online]. Available: https://github.com/spotify/annoy. [Accessed: 06-Feb-2019].

[2] T. Srivastava, "Introduction to KNN, K-Nearest Neighbors : Simplified," Analytics Vidhya, 27-Mar-2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/. [Accessed: 06-Feb-2019].

[3] "K-Nearest Neighbours," GeeksforGeeks, 13-Nov-2018. [Online]. Available: https://www.geeksforgeeks.org/k-nearest-neighbours/. [Accessed: 06-Feb-2019].

[4] Bullinaria, John A. Bullinaria A. "Bias and Variance, Under-Fitting and Over-Fitting." University of Birmingham, 2015, www.cs.bham.ac.uk/~jxb/INC/l9.pdf.

[5] Altman, N. S. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression." The American Statistician, vol. 46, no. 3, 1992, pp. 175–185. JSTOR, www.jstor.org/stable/2685209.

[6] Bernhardsson, Erik. "Nearest Neighbor Methods and Vector Models – Part 1." Erik Bernhardsson, 24 Sept. 2015, erikbern.com/2015/09/24/nearest-neighbor-methods-vector-models-part-1.html.

[7] Bernhardsson, Erik. "Nearest Neighbors and Vector Models – Part 2 – Algorithms and Data Structures." Erik Bernhardsson, 1 Oct. 2015, erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html.

[8] Bernhardsson, Erik. "Approximate Nearest Neighbor Methods and Vector Models – NYC ML Meetup." LinkedIn SlideShare, 25 Sept. 2015, www.slideshare.net/erikbern/approximate-nearest-neighbor-methods-and-vector-models-nyc-ml-meetup.

[9] Erikbern. "Erikbern/Ann-Benchmarks." GitHub, 6 Dec. 2018, github.com/erikbern/ann-benchmarks.