LargeLSH: An Exploration of Locality Sensitive Hashing for Approximate Nearest Neighbours on High Dimensional Data using Apache Spark

Wei Yang, Zhucheng Tu | CS 798 | December 13, 2017

Outline

- Motivation for using LSH for ANN on high dimensional data
- Related work and background information
- Methodology
- Evaluation
- Summary and main takeaways

Motivation

- The **nearest neighbour problem** has applications in many areas, such as data mining, information retrieval, databases, and machine learning
- **Curse of dimensionality**: Can be efficiently solved at low-dimensions using computational geometry data structures, but space complexity grows at $n^{O(d)}$ [1]
- Instead look towards **approximate nearest neighbours** (ANN) - popular ways include kd trees, balltrees, LSH

Related Work

- Single Machine & Brute Force k-NN
    - Pros: high accuracy
    - Problems: low efficiency, especially when data grows large and distributed through network
- Distributed K-D tree [Mohamed et al. 2008]
    - Pros: high efficiency sacrificing little effectiveness
    - Cons1: low efficiency when data dimension grows large
    - Cons2: graph-based, not efficient to parallelize (due to iterations, think MapReduce implementation of PageRank)
- LSH at Large [Haghani et al. 2008]:
    - Pros: Built on top of Chord-like P2P network (data is distributed)
    - Cons: Designed for querying points one by one, not for batch workloads

LSH using Apache Spark

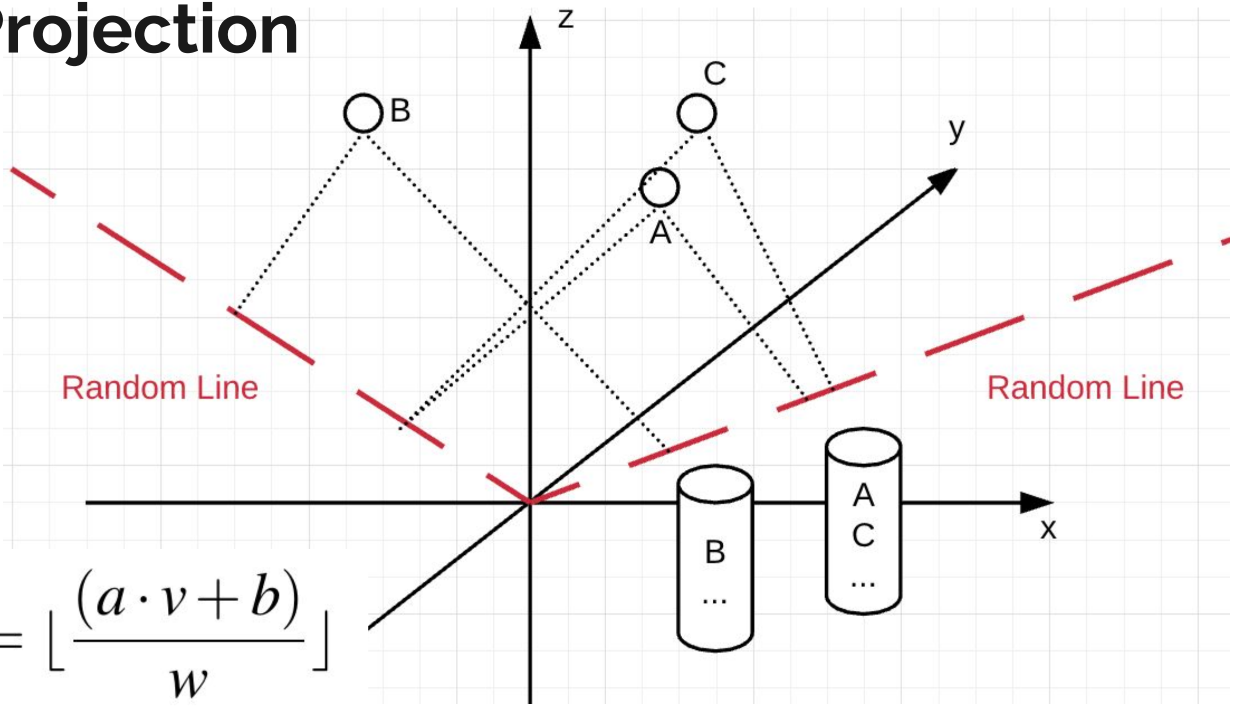
- Many modern workloads cannot fit inside the memory of a single machine - large datasets, each data point has many features
  - e.g. 80 million tiny images data set for object/scene recognition is 240 GB [2]
- Often, data is stored in a distributed file system, like HDFS
- Frameworks like MapReduce and Spark have become the de-facto for massive parallel computation on large datasets
- To our knowledge, there is little literature on distributed LSH using a modern parallel computation engine for the problem of ANN

LSH using Apache Spark cont'ed

- We found an open-source Spark implementation[1] of hybrid spill tree approach of solving (ANN)
- Want to compare LSH against this approach and examine accuracy vs. running time, scalability
- Spark has 2 family of APIs:
    - Traditional Resilient Distributed Dataset (RDD)-based API
    - New DataFrame API

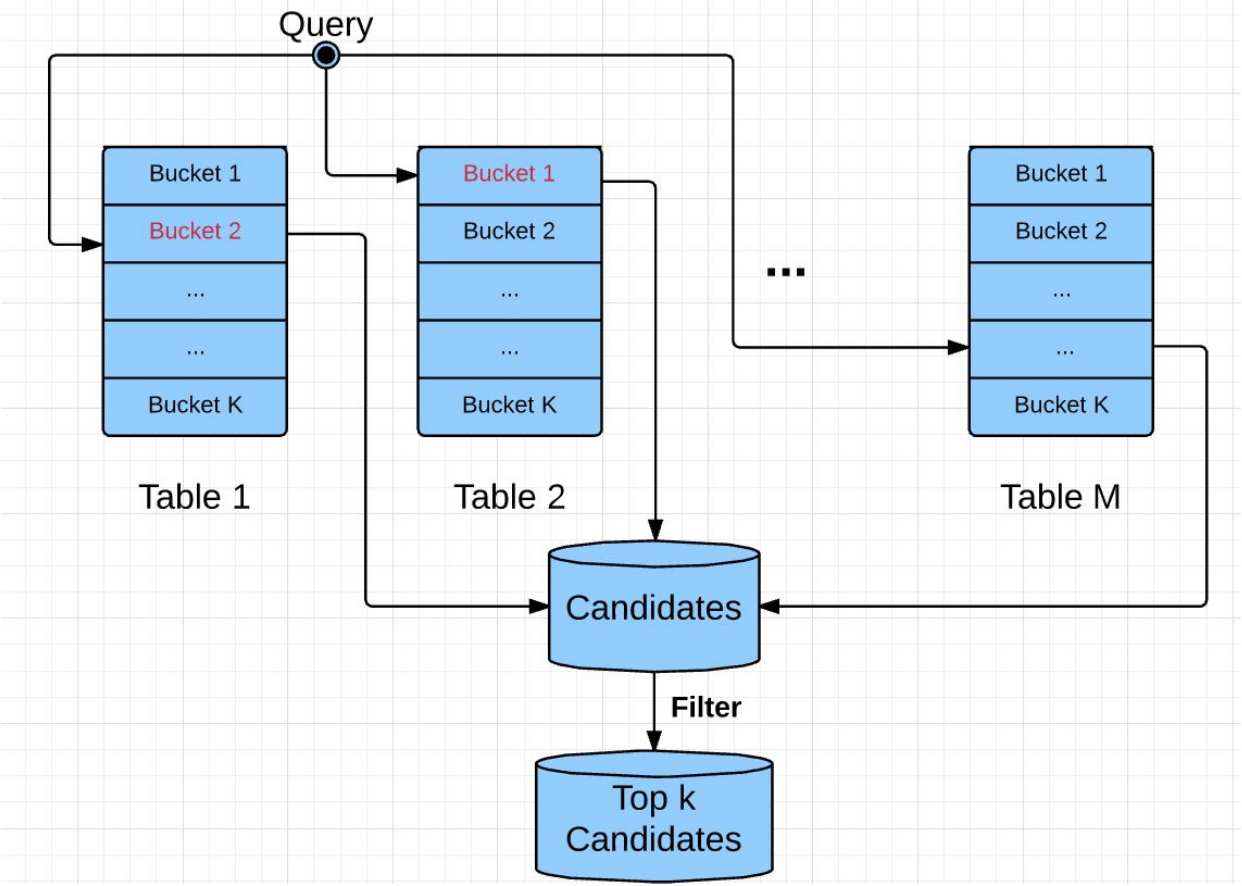
[1] https://github.com/saurfang/spark-knn

Random Projection



$$h_{a,b,w}(v) = \left\lfloor \frac{(a \cdot v + b)}{w} \right\rfloor$$

Query


Query
Bucket 1
Bucket 2
...
...
Bucket K
Table 1
Bucket 1
Bucket 2
...
...
Bucket K
Table 2
...
Bucket 1
Bucket 2
...
...
Bucket K
Table M
Candidates
Filter
Top k
Candidates

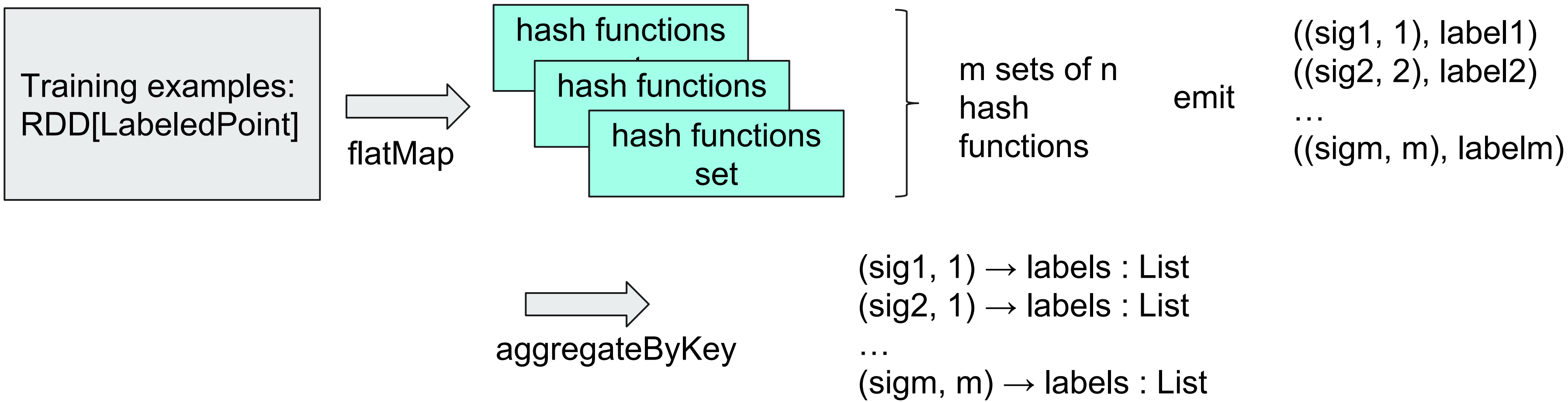
Algorithm 1: LSH-based Approximate k-Nearest Neighbor Search Algorithm

Function LSHkNN (H, Q);
Input : Hash Table H and Query Data Set Q
Output: Prediction List of labels L
$L \leftarrow \infty$
for $q = 1, 2, \ldots, n$ **do** `// for each query sample`
    for $i = 1, 2, \ldots, m$ **do** `// for each hash table`
        compute $h_{q,i} = g_i(Q_q)$
        find candidates $C_i = findHash(H, h_{q,i})$
    end
    collect candidates C = $\bigcup_{i=1}^{m} C_i$
    find best candidates by some distance matric $C' = findFirst(C, min(k, length(C)))$
    $L.insert(l_q)$
end
return L;

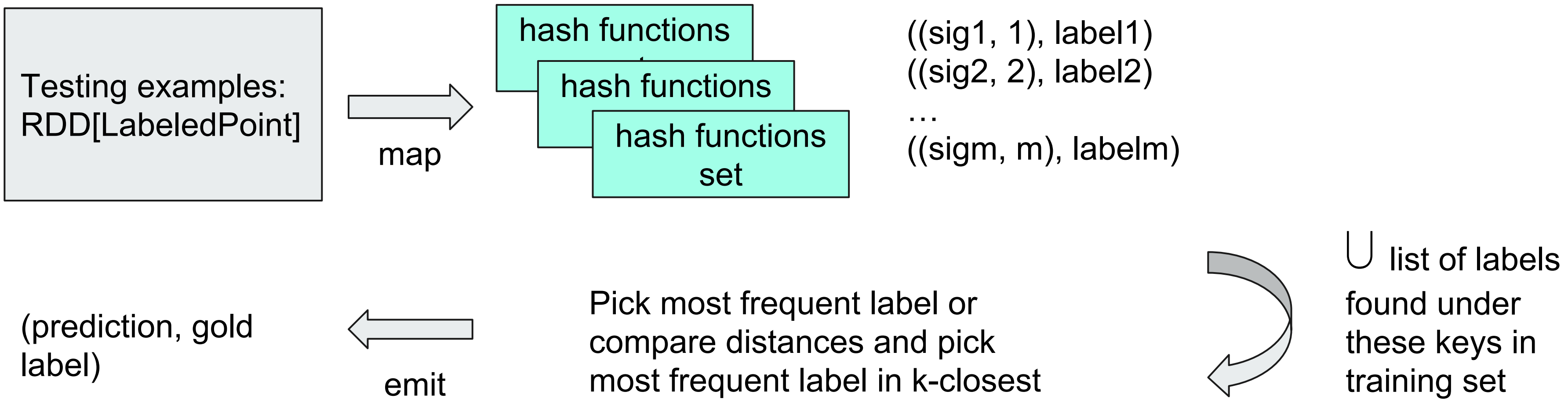
Algorithm 2: Distributed LSH

Function largeLSH (H, Q, k, d);
Input : Hash Table H, Query Data Set Q, Neighbor Number k, and distance threshold d
Output: Prediction List of labels L
generate ID column for H and Q
results $\leftarrow$ H.approxSimilarityJoin(Q) `// search the hash table and union the results through a map-reduce way`
resultsThres $\leftarrow$ filter(results, d) `// get rid of candidates far from the query points`
resultsSelected $\leftarrow$ `SELECT trainID, testID, distance FROM results_thres`
resultsPartitioned $\leftarrow$ findTopkByPartition(resultsSelected, k)
$L \leftarrow$ resultsPartitioned.map(groundtruthVector Intersect predictionVector)
return L;

RDD Random Projection Impl. Intuition



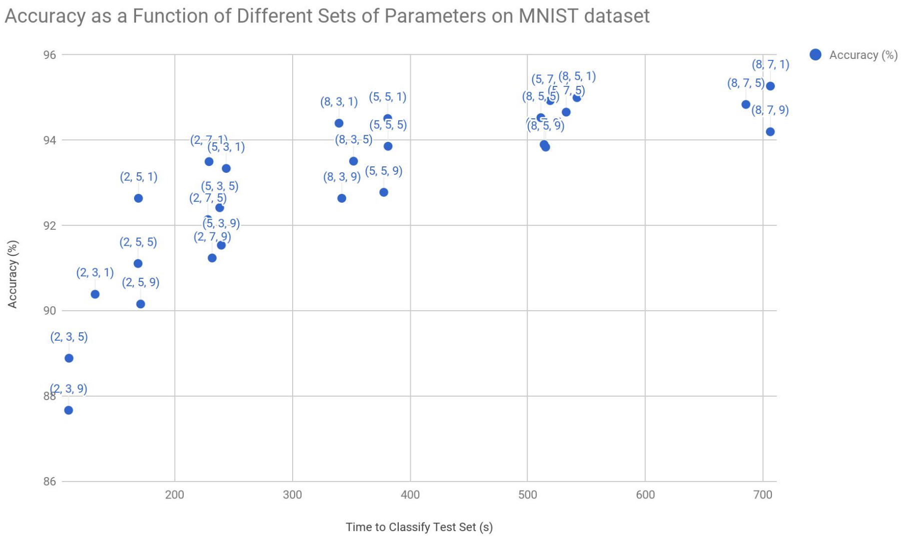
RDD Random Projection Impl. Intuition Cont'ed



Evaluation: Infrastructure and Dataset

- A small Spark cluster on Microsoft Azure
- 2 head nodes (A3), 4 cores, 285 GB disk each
- 2 worker nodes (A4), 8 cores, 14 GB RAM, 605 GB disk each
- Evaluated on
  - Standard Image Classification Task
    - MNIST: 60,000 * 784 Train, 10,000 * 784 Query
    - SVHN: 73,257 * 3,072 Train, 26,032 * 3,072 Query, 531,131 * 3,072 Extra
  - Large Scale Nearest Neighbor Search Task
    - SIFT1M: 1,000,000 * 128 Train, 10,000 * 128 Query
    - SIFT1B: 1,000,000,000 * 128 Train, 10,000 * 128 Query

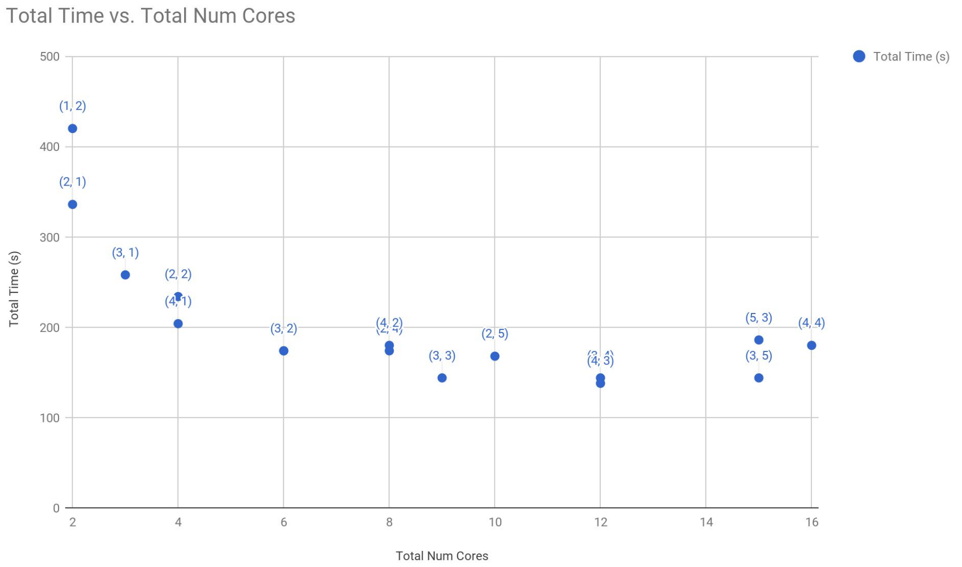
Evaluation: Accuracy vs. Time



- (bl, nht, k)
  - bl = bucket length
  - nht = # hash tables
  - k = # nearest neighbours
- ↑ bl, ↑ time, ↑ accuracy
- ↑ nht, ↑ time, ↑ accuracy
- ↑ k, negligible effect on time, doesn’t necessarily ↑ accuracy

spilltree implementation runs very slowly, need closer examination to ascertain result

Evaluation: Horizontal Scalability



- (# executors, # cores / executor)
- ↑ cores, ↓ time but with diminishing returns
- When the total # cores is fixed, more executors and fewer cores per executor is better than fewer executors and more cores per executor

Summary and Take Aways

- LSH-based approach:
  - can deliver high-accuracy results much faster than tree-based approaches
  - is flexible: can tune parameters to choose between the accuracy vs. running tradeoff
  - can be scaled horizontally, but in sublinear fashion (diminishing returns)
- Our distributed LSH approach and implementation:
  - presents a robust and scalable solution to the distributed k-Nearest Neighbor search problem over high dimensional data under the batch setting
  - can serve as the baseline of distributed ANN algorithm on two standard batch-retrieval tasks and show the tradeoff between effectiveness, efficiency, and resources

References

[1] SHALEV-SHWARTZ, S., AND BEN-DAVID, S. Understanding machine learning: From theory to algorithms. Cambridge university press, 2014.

[2] TORRALBA, A., FERGUS, R., AND FREEMAN, W. T. 80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE transactions on pattern analysis and machine intelligence 30, 11 (2008), 1958–1970.

[3] Aly, Mohamed, Mario Munich, and Pietro Perona. "Distributed kd-trees for retrieval from very large image collections." Proceedings of the British Machine Vision Conference (BMVC). Vol. 17. 2011.

[4] Haghani, Parisa, et al. "LSH At Large-Distributed KNN Search in High Dimensions." WebDB. 2008.