Computing Large-Scale Similarities : Distributed Locality Sensitive Hashing

Abstract

Millions of users visit commercial search engines and "query" their interests. To provide users with high quality of services, search engines such Yahoo!, Google, and Bing require intelligent analysis to realize users' implicit intents, in particular taking advantage of very large scaled query logs. One of the key interesting tasks involved in learning users' implicit intents often involves computing pairwise similarity between all queries. Due to the large scaled volume that is based on streaming querying, computing pairwise similarity is a computationally time intensive task. To challenge the aforementioned problem, we exploit several Locality Sensitive Hashing (a.k.a, LSH) methods and their novel variants. In particular, we develop the framework of LSHs on a distributed system (e.g. Hadoop) to take advantage of its computing efficiency.

1 Introduction

Many real-world problems on the Web like spelling correction (), finding related queries (Jones et al., 2006; Jain et al., 2011; Song et al., 2012), finding near-duplicate queries (), diversifying search results (Song et al., 2011) etc can benefit from query logs from a commercial search engine. We can take advantage of large-scale query logs by computing pairwise similarity between all queries. Each query point also has an associated feature vector, that is very high dimensional and sparse $-$ these are the set of webpages (urls) that get clicked for this query term.

However, computing pairwise similarity between all queries on a commercial large-scale query logs (consisting of hundreds of millions of queries) is a computationally time intensive task. To solve this problem, we exploit research advances in Locality Sensitive Hashing (Indyk and Motwani, 1998; Charikar, 2002; Andoni and Indyk, 2006; Andoni and Indyk, 2008) to propose an efficient approximate similarity Hadoop framework.

Spelling grouping mis-spellings in same group Similar entities LOTR, Finding related queries

2 Background

3 System

We are interested in finding out, using a batch process, a *small* set of neighbor candidates for each query such that: 1) the similarity of any point to a neighbor candidate returned is large and within a user-specified similarity threshold (τ). 2) We return an approximate set of neighbor candidates with a 100% precision and a large recall.

To solve the above problem, we propose a Hadoop-based framework based on Locality Sensitive Hashing. Our framework has three main steps: i) Generate 256 bit signature for each query. ii) Use the 256 bit signature for each query to generate L tables for each table with K bit signature. Next, generate query pairs which fall in the same table. iii) Compute cosine similarity between query points which fall in the same table and return the query pairs with similarity $\geq \tau$.

To minimize the number of hash functions computations (time intensive to read all features and evaluate hash functions to generate a single bit), we use a trick from Andoni and Indyk (2008) in which hash functions are reused to generate L tables. K vis assumed to be even and $R \approx \sqrt{L}$. We generate $f(q) = (h_1(q), h_2(q), \ldots, h_{k/2}(q)))$

Symbol	**Description**
N	# of query points
D	# of features i.e. all clicked unique urls
K	# of hash functions concatenated together $g(q) = (h_1(q), h_2(q), \ldots, h_k(q)))$ to generate the index of a table
L	# of tables generated independently with $g_j(q)$ index, $\forall 1 \le j \le L$
F	# of bits flipped, $\forall 1 \le j \le L$
τ	τ threshold
Recall	fraction of similar candidates retrieved
Comparisons	Avg # of pairwise comparisons per query

Table 1: Major Notations

L	τ	Comparisons	Recall
10	0.7	57.2255	0.63
	0.8	57.2255	0.84
	0.9	57.2255	0.98
28	0.7	152.3315	0.77
	0.8	152.3315	0.90
	0.9	152.3315	0.99
55	0.7	296.6195	0.89
	0.8	296.6195	0.97
	0.9	296.6195	0.99
120	0.7	629.5505	0.93
	0.8	629.5505	0.98
	0.9	629.5505	0.99

Table 2: $K = 16$ on AOL-logs.

of length $k/2$. Next, we define $g(q) = (f_a, f_b)$, where $1 \le a < b \le R$. This scheme generates $L = \frac{R(R-1)}{2}$. This scheme requires O($K\sqrt{L}$) hash functions, instead of O(KL).

4 Multi Probe LSH

One strategy that we have experimented with is the following variant of Multi-probe LSH: first we compute the initial LSH of a query q, which gives the L bucket ids. We then create alternate bucket ids by taking each of the L bucket ids and then creating alternate candidate buckets by flipping a set of coordinates in the LSH(q). We have experimented with two different methods to choose which coordinate to flip: randomly choosing a specified number of coordinates or choosing the set of coordinates based on the distance of q from the random hyperplane that was used in to create this coordinate.

Flip query as well as candidates Bits

F	τ	Comparisons	Recall
2	0.7	433.041	0.73
	0.8	433.041	0.88
	0.9	433.041	0.98
5	0.7	1556.5815	0.86
	0.8	1556.5815	0.95
	0.9	1556.5815	0.99
10	0.7	4138.3675	0.94
	0.8	4138.3675	0.99
	0.9	4138.3675	1.00

Flip query Bits only

F	τ	Comparisons	Recall
1	0.7	108.3585	0.65
	0.8	108.3585	0.85
	0.9	108.3585	0.98
2	0.7	158.9565	0.66
	0.8	158.9565	0.85
	0.9	158.9565	0.98
5	0.7	310.7435	0.70
	0.8	310.7435	0.86
	0.9	310.7435	0.98
10	0.7	556.796	0.75
	0.8	556.796	0.89
	0.9	556.796	0.99
16	0.7	838.8535	0.82
	0.8	838.8535	0.92
	0.9	838.8535	0.99

Table 3: $K = 16$ on AOL-logs. Random Flipping is tested here.

Flip query as well as candidates Bits

F	τ	Comparisons	Recall
2	0.7	405.239	0.86
	0.8	405.239	0.96
	0.9	405.239	0.99
5	0.7	1475.484	0.93
	0.8	1475.484	0.99
	0.9	1475.484	1.00
10	0.7	4059.0105	0.96
	0.8	4059.0105	0.99
	0.9	4059.0105	1.00

Flip query Bits only

F	τ	Comparisons	Recall
1	0.7	106.2105	0.72
	0.8	106.2105	0.88
	0.9	106.2105	0.98
2	0.7	155.354	0.75
	0.8	155.354	0.89
	0.9	155.354	0.98
5	0.7	303.0655	0.79
	0.8	303.0655	0.91
	0.9	303.0655	0.99
10	0.7	551.8475	0.81
	0.8	551.8475	0.92
	0.9	551.8475	0.99
16	0.7	838.8535	0.82
	0.8	838.8535	0.92
	0.9	838.8535	0.99

Table 4: $K = 16$ on AOL-logs. Query Driven Flipping is tested here.

Data	N	D
AOL-logs	0.3×10^6	0.7×10^6
Qlogs001	6×10^6	66×10^6
Qlogs010	62×10^6	464×10^6
Qlogs100	617×10^6	2.4×10^9

Table 5: Query-logs statistics

L	τ	Comparisons	Recall
10	0.7	1052.419	0.67
	0.8	1052.419	0.81
	0.9	1052.419	0.96
28	0.7	2908.1875	0.78
	0.8	2908.1875	0.90
	0.9	2908.1875	0.96
55	0.7	5648.186	0.84
	0.8	5648.186	0.92
	0.9	5648.186	0.97
120	0.7	12130.4065	0.91
	0.8	12130.4065	0.96
	0.9	12130.4065	0.99
253	0.7	25039.1265	0.95
	0.8	25039.1265	0.96
	0.9	25039.1265	0.99

Table 6: $K = 16$ on Qlogs001.

4.1 Random Flip LSH

Random Flips 1. Generate less number of hash functions 2. Have to compute similarity over less number of data pairs

4.2 Query Driven LSH

Based on random projection distance

5 Experiment

To evaluate our large-scale approximate similarity framework, we perform several experiments on query logs sampled from a commercial search engine.

5.1 Data

We remove all those queries which have less than or equal to 5 features.

The public dataset that we demonstrate result on is adapted from the query log of AOL search engine (Pass et al., 2006).

F	τ	Comparisons	Recall
2	0.7	8558.23	0.76
	0.8	8558.23	0.86
	0.9	8558.23	0.96
5	0.7	31119.1765	0.83
	0.8	31119.1765	0.90
	0.9	31119.1765	0.96
10	0.7	83196.2015	0.90
	0.8	83196.2015	0.93
	0.9	83196.2015	0.99
Flipping only bits for the query			
2	0.7	3091.9095	0.72
	0.8	3091.9095	0.84
	0.9	3091.9095	0.96
5	0.7	6095.7215	0.73
	0.8	6095.7215	0.86
	0.9	6095.7215	0.96
10	0.7	10976.4205	0.79
	0.8	10976.4205	0.87
	0.9	10976.4205	0.96
16	0.7	16634.244	0.81
	0.8	16634.244	0.89
	0.9	16634.244	0.96

Table 7: $K = 16$ on Qlogs001. Random Flipping is tested here.

F	τ	Comparisons	Recall
2	0.7	2979.8075	0.76
	0.8	2979.8075	0.87
	0.9	2979.8075	0.96
5	0.7	5903.528	0.78
	0.8	5903.528	0.87
	0.9	5903.528	0.96
10	0.7	10844.163	0.81
	0.8	10844.163	0.88
	0.9	10844.163	0.96
16	0.7	16634.244	0.81
	0.8	16634.244	0.89
	0.9	16634.244	0.96

Table 8: $K = 16$ on Qlogs001. Query Driven Flipping is tested here.

L	τ	Comparisons	Recall
10	0.7	10514.8925	0.64
	0.8	10514.8925	0.83
	0.9	10514.8925	0.95
28	0.7	29065.2255	0.74
	0.8	29065.2255	0.90
	0.9	29065.2255	0.97
55	0.7	56472.7575	0.83
	0.8	56472.7575	0.93
	0.9	56472.7575	0.98
120	0.7	121284.769	0.89
	0.8	121284.769	0.96
	0.9	121284.769	0.99

Table 9: $K = 16$ on Qlogs010.

L	τ	Comparisons	Recall
10	0.7	694.8745	0.53
	0.8	694.8745	0.76
	0.9	694.8745	0.92
28	0.7	1922.759	0.63
	0.8	1922.759	0.84
	0.9	1922.759	0.95
55	0.7	3761.9375	0.73
	0.8	3761.9375	0.89
	0.9	3761.9375	0.97
120	0.7	8166.7285	0.79
	0.8	8166.7285	0.93
	0.9	8166.7285	0.99

Table 10: $K = 20$ on Qlogs010.

F	τ	Comparisons	Recall

Table 11: $K = 20$ on Qlogs010. Query Driven Flipping is tested here.

L	τ	Comparisons	Recall
10	0.7	46.5485	0.49
	0.8	46.5485	0.74
	0.9	46.5485	0.91
28	0.7	129.22	0.57
	0.8	129.22	0.81
	0.9	129.22	0.94
55	0.7	250.617	0.63
	0.8	250.617	0.85
	0.9	250.617	0.95
120	0.7	545.12	0.77
	0.8	545.12	0.90
	0.9	545.12	0.97

Table 12: $K = 24$ on Qlogs010.

References

Alexandr Andoni and Piotr Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions.

Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*.

Moses S. Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC.

Alpa Jain, Umut Ozertem, and Emre Velipasaoglu. 2011. Synthesizing high utility suggestions for rare web search queries. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.

Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating query substitutions. In *Proceedings of the 15th International Conference on World Wide Web (WWW)*. ACM.

Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*. ACM Press.

Yang Song, Dengyong Zhou, and Li-wei He. 2011. Post-ranking query suggestion by diversifying search results. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. ACM.

Yang Song, Dengyong Zhou, and Li-wei He. 2012. Query suggestion by constructing term-transition graphs. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM)*. ACM.