

BOOTSTRAP SEQUENTIAL PROJECTION MULTI KERNEL LOCALITY SENSITIVE HASHING

*Thesis submitted in partial fulfillment of the requirements for the award of degree
of*

Master of Engineering
in
Software Engineering

Submitted By
Harsham Mehta
(Roll No. 801231011)

Under the supervision of:
Dr. Deepak Garg
Associate Professor



**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004**

July 2014

CERTIFICATE

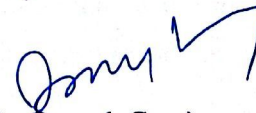
I hereby certify that the work which is being presented in the thesis entitled, "*Bootstrap Sequential Projection Multi Kernel Locality Sensitive Hashing*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Deepak Garg* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.



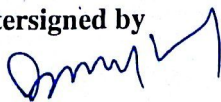
(Harsham Mehta)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.



(Dr. Deepak Garg)
Associate Professor,
Head, Computer
Science and Engg.
Department

Countersigned by



(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala



(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

It is a great pleasure for me to acknowledge the guidance, assistance and help I have received from my guide Dr. Deepak Garg, Head of Computer Science Department. I am thankful for his continual support, encouragement, motivation and invaluable suggestions that inspired me for the thesis work. He not only provided me help whenever needed, but also the resources required to complete this thesis report on time.

I wish to express my gratitude to Dr. S. K. Mohapatra ,Senior Professor, Dean of Academic Affairs for introducing me to my thesis topic and providing me an opportunity to be a part of state of art research.

I also wish to express my gratitude to Dr. Damandeep Kaur, Software Engineering P.G. Coordinator, Computer Science and Engineering Department, for cooperating and inspiring me for research work.

I would also like to thank all the staff members of Computer Science and Engineering Department for providing me all the facilities required for the completion of my thesis work.

Most of all, I would like to thank my parents and friends for their inspiration and ever encouraging moral support, which went a long way in successful completion of my thesis.

Above all, I would like to thank the almighty God for his blessings and for driving me with faith, hope and courage.

ABSTRACT

In Recommender system we have similarity search as a key part for making efficient recommendations. Similarity search have always been a tough task in a high dimensional space. Locality Sensitive Hashing which is most suitable for extracting data in a high dimensional data (Multimedia data) has been most suitable for it. The Idea of locality sensitive hashing is that it decreases the high dimensional data to low dimensions using distance functions and then store this data using hash functions which ensures that distant data is placed much further. This technique has been extended to kernelized Locality sensitive hashing (KLSH). One limitation of regular LSH is they require vector representation of data explicitly. This limitation is addressed by kernel functions. Kernel functions are capable of capturing similarity between data points. KLSH is a breakthrough in content based systems. This method takes a kernel function, a high dimensional database for data inputs and size of hash functions to be built. These kernel functions that are being used may give different degree of result precision. Hence we try to combine these kernels with a bootstrap approach to give an optimal result precision. In this paper we present the related work that has been done in locality sensitive hashing and at the end we propose algorithms for data preprocessing and query evaluation.

Table of Contents

Certificate	i
Acknowledgement	ii
Abstract	iii
Table of Content	iv
List of Figures	vi
List of Tables	vii
Chapter 1	
Introduction	1
1.1 Definition of nearest neighbor problem	1
1.2 Recommendation System – A Near Neighbor Problem	2
1.3 Features of Recommender System	3
1.3.1 Accuracy	3
1.3.2 User Satisfaction	3
1.3.3. Satisfaction of the Recommendation Provider	3
1.4 Classification of Recommender System	4
1.4.1 Collaborative Filtering	4
1.4.2 Content Based Approach	5
Chapter 2	
Literature Review	10
Chapter 3	
Problem Statement	23
Chapter 4	
Bootstrap and Kernel Based LSH	25
4.1 Analysis of Kernelized Locality Sensitive Hashing	25
4.2 Multi kernel Locality Sensitive Hashing	27
4.3 Bootstrap Sequential Projection Multi – Kernel	
Locality Sensitive Hashing	29
4.3.1 Bootstrap sequential projection Learning	29

Chapter 5

Experiment	33
5.1. Test Bed	33
5.2 Experimental Setup	33
5.2.1 Image Feature extraction	33
5.2.2 Kernel function	34
5.2.3 Experimental Procedure	34
5.3 Algorithm comparison	36
5.4 Experimental Result	37

Chapter 6

Conclusion and Future Scope	39
References	40
List of Publications	46

List of Figures

Figure 1

Representation of a 58 x 58 kernel matrix based on single feature set 34

Figure 2

GUI of Bootstrap Sequential Projection Multi kernel
Locality Sensitive Hashing. 35

Figure 3

Query extraction and processing. 35

Figure 4

Results of the queries 36

Figure 5

mAP of n query for BSP MK LSH. 37

List of Tables

Table 1	
Description of notations that are used	25
Table 2	
Algorithms complexity comparison.	36
Table 3	
mAP and standard deviations of different algorithms	38

Chapter 1

Introduction

At this modern age we have numerous of technologies for our services and what comes from them is a large amount of data. This data is usually media data, genomic, network, medical or astronomical data. From this data we try to figure out some useful information. Mostly by that we mean labeling of information. But labeling also becomes a tough job without any prior information (audio information, visual information) about the data. But suppose we have a subset of data already labeled then we can label rest of data by finding the most similar data among the already labeled data accordingly.

Hence we compute the following problem called All – Pairs – Nearest – Neighbor problem: For a collection of objects already given to us, ' q ' is an unlabeled object and we find a labeled object p which is under some notion of similarity is most similar to q so we label it as the category of p . This might look as an over effort but there is a problem with the data mentioned above. It is that it keeps getting bigger and bigger and hence becomes vitally important to design an algorithm to process this data efficiently. So instead of solving the problem efficiently for polynomial time we try to find answers whatever we can get in linear time. In various previous years many methods have been proposed that uses approximation to overcome the running time bottleneck. It formulates the no. of result points present up to the distance of ' c ' where $c > 1$ is approximation factor.

1.1 Definition of nearest neighbor problem

We define the Nearest Neighbor problem as optimization problem:

Out of all the solution points to an objective, find the point that minimizes our particular objective solution uncertainty (i.e. distance to query point).

R – Near neighbors of q includes all the points that are present up to distance R from the query point q . We can simply check if the returned point as nearest neighbor point is a

R – Near neighbor point of the query. But calculating the other way i.e. checking among the R – Near neighbor points if particular is nearest point to the query is quite ubiquitous.

There is another issue related to the near neighbor problem which is best known as “Curse of Dimensionality”. Data points are d – dimensional where d can range up to hundreds or even thousand. We have a no. of solutions for low dimensional data point. Say when $d = 1$, we can have sort all data points in the preprocessing stage and then executing query with binary search. So the solution takes $O(n)$ space $O(\log n)$ and query time. But what if d is of higher value i.e. higher dimensional data points. In that case none of the previously proposed methods are going to come handy. Example kd – trees [1] is a well known data structure but it only supports up to 10 or 20 dimensional value. As a successor to kd – trees several other data structures were proposed that supported multi – dimensional data including R – tree, R^* – tree, X – tree, SS – tree, VP – tree, SR – tree, metric – trees [2]. All these solutions possess a complexity either space or query time exponential in d . All these algorithms are able to achieve little improvement over linear time algorithm that compare query with each data point in database [3]. This phenomenon is called “curse of dimensionality”.

1.2 Recommendation System – A Near Neighbor Problem

We have a various applications, domains that use the nearest neighbor approach. Such kind of an application is Recommendation system that uses the approximation theory of near neighbor concept. Recommendation system is an application but also serves as domain to various research topics as it includes and invites various concepts that are involved to give a better recommendation.

Recommender System has been a domain of interest since first paper on collaborative filtering emerged in 1995[4]. It constitutes problem – rich research areas and huge no. of practical application that help users to deal with information overload and provide services for real-life application. Examples of such kind of application would include Amazon.com which recommends CDs, books, other items to the users [5], MovieLens [6] recommending movies to the users and AdaptiveInfo.com which is now known as VERSIFI [7] recommending news to the users according to their interest.

Recommendation system is mainly about the problem of estimating ratings of the items that have not been rated by the users which sounds similar to the nearest neighbor problem of labeling the unlabeled data with the help of already known data. The current generation of recommender system requires much more improvements to make recommendations methods more effective and applicable to much wider scope like recommending vacations, certain financial services or purchasing products while shopping for example “smart” [8]. We specify heuristics that define utility function for extrapolation from known to unknown by optimizing various performance criteria recommending N best items matching criteria [9].

1.3 Features of Recommender System

1.3.1 Accuracy:

Accuracy is related to the capacity to satisfy the user’s need of information and it may vary among users due to different context, preferences, goals, knowledge and background [10]. Hence a good recommender system is the one that provide the items that satisfy the information needs, in one word – they are “relevant” to the user. To achieve high accuracy we require high coverage of the available items [11].

1.3.2 User Satisfaction:

At first one might think that an accurate recommender system that recommends the most relevant items would also satisfy the user. However, many other factors also influence user satisfaction. One of a factor is serendipity. There may be a chance that even recommender system after being accurate it might not be satisfactory. For example user may have to wait for too long to receive recommendations or if the presentation is unappealing. Some systems want users to specify their interests manually. There are a few system that collect user’s interests automatically.

1.3.3. Satisfaction of the Recommendation Provider:

Typically for any system, it is assumed that providers of systems are satisfied when their users are satisfied. But one other interest of the provider is to make low cost system, where costs may refer to labor, disk storage, memory, CPU power, and traffic [12]. Hence a good recommender system is developed, operated, and maintained at a low cost. Other providers may want to generate a profit from the recommender system A news-

website might suggest articles of longer length so as to keep their readers a longer time on their website [13].

To evaluate the performance of recommendation algorithms we check coverage and accuracy metrics of the algorithm. Coverage measures the percentage of items for which predictions can be made by recommender system. Accuracy can be either statistically or by decision-support. Statistical accuracy metrics mainly compare the actual ratings R in the User Item matrix with the estimated ratings and include various parameters like Mean Absolute Error (MAE), root mean squared error, and correlation between predictions and ratings. Decision-support measures determine the degree of relevance to which a recommender system can make predictions of items [14].

1.4 Classification of Recommender System

Recommender systems are broadly classified in following categories:

Collaborative recommendations

Content – based recommendations

1.4.1 Collaborative Filtering

Collaboration based recommender system predicts the utility of items for a particular user based on the items previously rated by other users. There has been various collaborative recommender system developed. Grundy system [15] is considered to be first recommender system recommending books. It uses stereotypes for building user models from limited information. . GroupLens [16] [17]. Video Recommender and Ringo were the first systems to use collaborative filtering algorithms to automate prediction.

We will not get into the details of collaborative system, but would limit it to a brief overview of how it works and its classification.

Let the utility of item s for user c is denoted by $u(c, s)$. It is estimated on the bases of utilities $u(c_j, s)$ assigned to item s by the users $c_j \in C$ who are “similar” to user c . For example, for recommending movies to user c , the collaborative recommender system will create set of user that are similar to user c by comparing their user profiles. The movies they liked the most would be recommended to user c [18], [14].

Collaborative recommender system can be grouped into two general classes [17]:

1. Memory-based (or heuristic-based) and
2. Model-based.

Memory-based algorithms

Memory-based algorithms predict rating of items on the bases of previously rated items by all the users i.e. is the aggregate of the ratings of N most similar users for the same item s . To measure distance there are various similarity functions that can be used as a weight. The two most popular approaches to calculate similarity are correlation and cosine-based approach. In this paper we would use Euclidean distance.

In Cosine-based approach, the two users x and y are treated as two vectors. The similarity between two vectors is the cosine of the angle between them [14]. The same correlation-based and cosine based techniques can be used to compute similarities between items and obtain the ratings from them.

Model-based algorithms

In Model – based algorithm, with the help of collection of ratings we learn a model for making rating predictions. We use probabilistic approach to collaborative filtering [17]. Rating values are integers between 0 and n . We can calculate the probability that user c will rate item s to a particular value given all ratings of different items rated by user c with some expressions [17].

In some applications, model-based methods might excel over memory-based approaches when we talk about accuracy of recommendations. As model-based techniques does not calculate utility (rating) predictions according to some heuristic rules, but it calculates according to a model learned from the underlying data by using statistical and machine learning techniques. There are certain theories that support argument that method combining both memory-based and model-based approaches can provide better recommendations.

1.4.2 Content Based Approach

This approach to recommendations has its roots in information retrieval [19], [17] and information filtering [20] research. Our topic, Locality Sensitive Hashing is part of information filtering, but we would focus on it later in our work. These systems are

mostly used for recommending items with textual information. Fab system [19], Syskill and Webert system [15] recommend pages with important words. These systems are used for organizational purposes for eg. Educational institutes recommending papers.

There are various techniques involved in a content based recommender system. Finding and specifying keywords weights in information retrieval technique is mostly done by TF IDF measure [17].

We give a little demonstration of content based approach to show what it is about. Let if w_{ij} be the weight of i^{th} keyword in document d_j , then content of d_j document could be defined as $Content(d_j) = \{w_{i1}, \dots, w_{ik}\}$. Where k is the total no. of keywords in document d_j .

All the items that are previously rated by the user are compared with the candidate items that could be recommended and the items that match the best are usually recommended.

If say content based systems recommend on the basis of user profile defined as $ContentBasedProfile(user)$ which is created by analyzing $Content(items)$ of those items usually seen and rated by a user. It contains tastes and preferences of that particular user.

Let us say for a user c , $ContentBasedProfile(c)$ contains $\{w_{c1}, \dots, w_{ck}\}$.

Where w_{ci} denotes importance of i^{th} word to user c . Profile can be computed by Rocchio algorithm [16], giving an “average” vector from an individual content vectors [19], [17]. And in case of documents probability of liked by the user or not can be found using Bayesian classifier [15] or Winnow algorithm [21] but latter works well with large feature set of documents [22].

Both of these functions when compounded provide us our basic output function $u(user, d)$ which is termed as utility function which is basically a score function defined as:

$$u(user, d) = score(ContentBasedProfile(user), Content(d)).$$

This scoring function is usually defined for information retrieval method in heuristic measure like cosine similarity [17]. Bayesian classifier also can be used for content based

recommendation with some other machine learning techniques like clustering, artificial neural networks, decision trees [15] etc. These learning techniques use underlying data to learn a model and use that model to calculate utility prediction and not some heuristic function. All these recommendation techniques in content based recommender systems mostly are related to text retrieval. Example, adaptive filtering [23] observes documents one by one in a continuous document stream to identify relevant documents more accurately and threshold setting for a query to find the relevant document for a user it decides the extent up to which a query should be matched.

As stated earlier Recommender system comprises of various research fields hence there is wide scope of future work in almost every field. For example for comprehensive understanding of users and items, we need to adopt some advanced profiling techniques [14]. An important field is multi – dimensionality of recommendations. This will be the domain in which we will be interested in the further discussion as we introduce topic of locality sensitive hashing. Most of the recommendation examples Operates in the two-dimensions like $user \times item$ space. There is always a chance of some crucial contextual information for most of the applications that are not taken into consideration. SAY, the utility of a various products for a user may depend significantly on time (e.g., the time of the year, season or month, or the day of the week). In such cases we can define the utility (or ratings) function for users over a multidimensional space $D_1 \times D_2 \times \dots \times D_n$ [14].

As we discussed the text retrieval techniques are most vocal in content based information retrieval based on which recommendations are mostly done. This information retrieval procedure has now become more prominent in image processing in the past decade. There are various methods proposed to extract information from images which are usually termed as ‘features’ and recommend or search similar images based on those features which require some processing. Now these features of images that are extracted as a part of information retrieval are usually found to be high dimensional (in hundreds) and processing of this high dimensional data in database that contains millions or billions of images does not come really very handy. A lot of techniques like kd – trees, metric – trees that have been discussed before were tried on this data too but they all were a

disaster mostly because of the “Curse of Dimensionality” phenomenon. So a newer technique was introduced in 98 called “Locality sensitive hashing” by P. Indyk et.al.[24]. It has become important to address these images due to the fact that they are high dimensional and their size is almost in billions and is increasing at a faster rate. Hence labeling all these images has become necessary.

Locality sensitive hashing (LSH) technique tackles this problem with ease. LSH has also been used in various other domains due to its tremendous results, high level performance and compatibility with dimensionality. Various extensions of LSH have been introduced in the past decade. These extensions can be grouped into two groups based the method of how they implement LSH technique. First is linear projection method and second is kernel based method. We describe both methods here but our focus will be on kernel based method in the further discussion.

Linear projection method implement a family of locality sensitive hashing functions which map same/similar objects no nearby buckets with high probability and dissimilar products to farther buckets with high probability. Hence probability is the greatest factor in finding similar objects. LSH obtains a query time of $O\left(dn^{\frac{1}{\epsilon}}\right)$ in worst case. LSH is able to achieve a query time of $O\left(dn^{\frac{1}{1+\epsilon}}\right)$ after improvising the technique [25]. Its performance can be improved by tweaking the I/O disk access by the use of b – trees [26]. We can also reduce space requirement of LSH by introducing multi probe LSH method [27]. LSH also encouraged an idea of compact binary codes to which many studies are focused on [28]. Many other issues are being addressed by different works that are related to LSH which include the issues related to Hamming distance [24], normalized partial matching, lp norms, learned Mahalanobis metrics.

The regular LSH has some limitations with it that it assumes the data that is being processed come from multidimensional vector space and their underlying embeddings are also known and computable. Hence we require a framework that processes the underlying embeddings of data implicitly by exploring the similarity/kernel function. This framework is provided by kernel based methods particularly known as Kernel based LSH [29]. A variant of KLSH is proposed in which the hamming distance between the two

vector's binary codes is strongly related to the shift invariant kernel [30]. KLSH technique is speeded up by introducing an algorithm for learning binary codes with kernels [31]. But most of the techniques used only a single kernel for kernel based hashing methods [29] ,[30] ,[31],. Several attempts have been made to produce a multi kernel hashing methods but they were not able to explore the true potential. The problem with single kernel is that they explore only a single feature set, but what if the data involves a large amount of features to describe itself. Similar is the case of images. There is a large no. of feature extraction techniques available for images and can be analyzed using various similarity functions. So they require to be accessed and analyzed using most of the feature set [33], [32] and for that we require to implement multiple kernel function. Now the question that comes is how to configure multiple kernel function.

For this two algorithms are proposed one is weighted multi kernel LSH that calculated weight of each kernel and dedicated the hashing bits accordingly and the other one is boosting multi kernel LSH that used boosting learning technique to learn the bits allocation of each kernel [35]. BMKLSH introduced the concept of boosting rounds for learning of the bit size of each kernel. The optimization is done according to the following problem:

$$\max_{b_1, \dots, b_m \in |b|} \sum_{i=1}^{n_q} \exp \left(\sum_{l=1}^m AP_l(i) \right) \text{ subject to } \sum_l b_l = b.$$

To this end we propose our novel multi kernel LSH method that uses a refined learning technique of bootstrap sequential projection. Bootstrap learning algorithm performs regularized learning based hashing [34]. Learning technique is used which corrects the error effectively from holistically learned bits in previous projections with no computational overhead.

Chapter 2

Literature Review

Piotr Indyk et. al. [24] first introduced locality sensitive hashing in 1998. He introduced for $\{0,1\}^d$ Hamming metric space of dimension d defined as

$$H^d = (\{0,1\}^d, d_H). \quad (1)$$

Let $\mathcal{M} = (X, d)$ be a metric space defined where $X = \mathbb{R}^d$. If $P \subset X$ contains n data points i.e. $(p_1, p_2, \dots, p_n)$. Then distance of any point p to the rest of the set could be represented as

$$d(p, P) = \min_{q \in P} d(p, q). \quad (2)$$

Where $d(p, q)$ is Euclidian distance. Diameter of set P is

$$\Delta P = \max_{p, q \in P} d(p, q). \quad (3)$$

A ball of similarity measure D is a set that contains all those points that are similar to a particular point say q . It is represented as

$$B(q, r) = \{p: d(p, q) \leq r\}. \quad (4)$$

A family of hash function is defined as

$$\mathcal{H} = \{h: S \rightarrow U\} \quad (5)$$

The hash function family is said to be (r_1, r_2, p_1, p_2) - sensitive for D if for any $q, p, p' \in S$

If $p \in B(q, r_1)$ then $\Pr_{\mathcal{H}}[h(p) = h(q)] \geq p_1$

If $p \notin B(q, r_2)$ then $\Pr_{\mathcal{H}}[h(p) = h(q)] \leq p_2$

Where $S \subset P$ is (γ, δ) - cluster for P i.e. for every $p \in S$, $|P \cap B(p, \gamma \Delta S)| \leq \delta |P|$

And $\Pr_{\mathcal{H}}[h(p) = h(q)] = \text{sim}(x, y)$ i.e. is a probability based similarity estimation technique. The complexity for this technique for ϵ -approximate result is

$O(dn + n^{1+1/\epsilon})$ space and $O(n^{1/\epsilon})$ hash function evaluation for each query and $O(d)$ operations for evaluating hash function.

Venu satuluri et.al. [39] In their paper mentioned two phases to a similarity search algorithms i.e. candidate generations phase and candidate verification phase. They have also developed two algorithms bayes LSH and bayes LSH- Lite. Bayes LSH performs both candidate searching and similarity estimation, while Bayes LSH- Lite only performs candidate searching and computes the similarities of unpruned candidates exactly. The algorithm Bayes LSH returns pairs of objects with similarity estimation where as Bayes LSH - lite returns objects that are exactly similar.

Classical similarity estimation: If we compare n hashes and have observe m agreements in hash values, the maximum likelihood estimator for the similarity is:

$$\hat{s} = \frac{m}{n}.$$

But there were certain limitations with this approach like, difficulty of tuning no. of hashes. To get the same level of accuracy for different similarities, we will need to use different number of hashes. It could be easily done but we don't know the true similarity of each pair.

Paper also uses different similarity measures like jaccard similarity and cosine similarity for bayes LSH.

Junhao Gan et. al. [40] gave another variant of LSH which based on Currently, the primary choice of constructing an LSH function for Euclidean distance is to project data objects (represented as vectors $\vec{o}$ in R^d) along a randomly chosen line (identified by a random vector $\vec{a}$) that is segmented into equi-width intervals of size w , and then project data objects to the same interval which are termed as “colliding” in the hashing scheme, for this we take each interval as a bucket.

LSH function is of form $h_{\vec{a},b}(o) = \left\lfloor \frac{\vec{a} \cdot \vec{o} + b}{w} \right\rfloor$ where b is real no. chosen from $[0, w]$ uniformly.

Exact Euclidean LSH (E2LSH) exploits LSH as : a set of k LSH functions $h_1, h_2, \dots, h_k$ are randomly chosen from LSH family (defined by equation 5) then they are joined together to form a compound hash function

$$G(o) = (h_1(o), h_2(o), \dots, h_k(o)) \text{ for any object } o.$$

$G(o)$ is used to hash all data objects into hash table T . We use a compound function for hashing because it reduces the probability of collision of two “distant” data object.

It is difficult to design good compound hash function which would drop every pair of “close” objects in the same bucket for at least one hash table. Hence several hundred several hundred of compound hash functions and hash tables are needed in the E2LSH method to guarantee good search accuracy.

Another LSH defined is Collision Counting LSH (C2LSH). It chooses a set of m LSH functions with appropriately small interval size w (say $w = 1$) and form a function base, denoted as B with $|B| = m$. Intuitively, if a data object o is close to a query object q , then the two objects are very likely to collide under every single LSH function in B . Accordingly, o should collide with q under a large number of LSH functions.

C2LSH exploits only a single base B of m LSH functions $\{h_1, h_2, \dots, h_m\}$, where each h_i is randomly selected from an (R, cR, p_1, p_2) -sensitive LSH family. Here m is the base cardinality of B . $h_i()$ builds a hash table T_i by hashing each data object o in the database D with $h_i()$ to an integer. Each hash table T_i is surely a sorted list of buckets, and each bucket contains a set of object IDs of the objects that will fall in the bucket.

When a query q arrives, first locate bucket in which query q will fall.

Compute $h_i(q)$ for $i = 1, \dots, m$ and find union of data objects colliding with q , for every data object (say o) we can compute $\#collision(o)$.

$$\#collision(o) = |\{h | h \in B \wedge h(o) = h(q)\}|$$

Identify set C of all frequent objects. If C has less than βn objects, (n is the cardinality of the database and β is allowable percentage of false positives) we have to compute distance of each object of C , otherwise compute distance of first βn objects.

Bahman Bahmani et. al. [41] gives another enhancement to LSH distributed locality sensitive hashing. Two main instances of distributed frameworks are batched processing system MAP REDUCE [61] and real time processing system active distributed hash tables (Active DHT). Set S of n data points arriving as batch or real time.

Parameters k is the no. of hash function in a hash set H taken from hash family as defined in eq (5). L is the max no. of hash table that can be used. w is the max no. of bits of hash function. $H = Hw$ and $H' = H'w$ are the hash function families.

If a machine processes a (key, value) pair and let h be randomly chosen hash function $h \in H$ for any data point $p \in S$ a (key, value) pair $(h(p), p)$ is generated.

For each query point q , they generate the offsets $q + \delta_i (1 \leq i \leq L)$, for each of them a (Key, Value) pair $(H(q + \delta_i), q)$ is generated. Then, for any received query point q , the machine with id $H(q + \delta_i)$ retrieves all data points p , if any, with $H(p) = H(q + \delta_i)$ which are within distance of cr from q .

But the new idea is to use another layer of locality sensitive hashing to distribute the data and query points over the machines. More specifically, for a parameter value $D > 0$, they sample an LSH function $G: \mathbb{R}^k \rightarrow \mathbb{Z}$. $G(v) = \left\lfloor \frac{\alpha \cdot v + \beta}{D} \right\rfloor$ where $\alpha \in N^k$ and β is chosen uniformly from $[0, D]$.

Then from $G(H())$ they generate (key, value) pair $G(h(p)), < h(p), p >$. Then add p to a bucket $h(p)$ for each query point $q \in Q$, for each unique value x in the set

$$\{GH(q + \delta_i) | 1 \leq i \leq L\}$$

We generate a (Key, Value) pair (x, q) Then, all the data points p will be on machine x such that $GH(p) = x$ as well as the queries $q \in Q$ one of whose offsets gets mapped to x by $GH()$.

Narayan Sunderam et al [42] has given a most popular and widely implemented application of LSH., but to make PLSH work well there were few algorithmic and technical contributions required to make.

1. Both hash table construction and searching in it are distributed across multiple cores and multiple nodes in PLSH.

2. Within a single node, multiple cores will access concurrently the same set of hash tables. For this techniques to batch and rearrange accesses to data are developed to maximize cache locality and improve throughput.
3. They also develop a novel cache-efficient variant of the LSH hashing algorithm, which improves index construction time. We try to minimize cache miss effects by performing software pre fetching.
4. They propose a new hybrid approach that buffers inserts in an insert optimized LSH delta table and merge these into our main LSH structure periodically, to handle streaming arrival and expiration of old documents,.
5. To accurately project the performance of single- and multi-node LSH algorithms to within 25% of obtained performance they developed a detailed analytical model.

System consists of multiple nodes, each node store a portion of the original data in-memory for processing speed. Hence memory size determines the total capacity of a node. Coordinator broadcast queries from different clients to all nodes, with each node querying its data. Responses from each structure are concatenated by the coordinator node and sent back to the user

Two main steps in LSH table construction

- (1) Hash functions are applied on every data point to generate the k-bit indices into each of the L hash tables.
- (2) Insert each data point into all L hash tables.

Each hash function is the dot-product between randomly generated hyper plane in the high-dimensional vocabulary space and the sparse term vector representing the tweet.

To construct a hash tables are required to consist of contiguous arrays with exactly enough space to store all of the records that hash to (collide in) each bucket, and it is to be stored in memory and in parallel in a way that maximizes cache locality.

Process of insertion into the hash tables can be viewed as a partitioning operation. This involves three main steps:

- (1) Generate a histogram of entries by scanning each element of the table and in the various hash buckets;
- (2) Obtain the starting offsets for each bucket in the final table by perform a cumulative sum of this histogram
- (3) Re compute the histogram and perform an additional scan of the data, this time adding it to the starting offsets to get the final offset for each data item.

This computation needs to be performed for each of the L hash tables.

Hao Xia et. al. [35] has provided another extension of LSH with kernel function over a large collection of image database. They have discussed every implementation of LSH using Kernel function. It includes scalable image retrieval based on Locality Sensitive Hashing using Single Kernel, using multiple kernels and at the end they proposed a boosting algorithm for multi- kernel implementation of LSH.

The kernel function used by them is defined as

$$k(x, x') = \exp(-\gamma^{-1}d(x, x')) \quad (6)$$

Where $d(.,.)$ is a distance function, for the sake of simplicity they have considered Euclidean distance. In the experimental setup they have used a sequence of 9 kernel functions.

The various multi kernel algorithms proposed combine the result of separate single kernel function. This approach fails to fully explore the power of multi kernel and most of all this technique is completely unsupervised. The boosting approach followed provides the optimal combination of multiple kernels.

To find the optimal allocation of bit sizes

If b_l is the no. of bits allocated to l^{th} kernel are:

$$\max_{b_1, \dots, b_m \in |b|} \sum_{i=1}^{n_q} \exp \left(\sum_{l=1}^m AP_l(i) \right) \text{ subject to } \sum_l b_l = b.$$

This is a NP hard problem where $AP_l(i)$ is the average precision performance of applying the l^{th} kernel. After each round of kernel algorithm, best kernel with largest weighed Average precision performance (wAP) over the current distribution D_t is:

$$wAP_l = \sum_{i=1}^{n_q} D_t(i) AP_l(i).$$

R. Zhang et al. [43] proposed a non linear non stationary multiple kernels combination algorithm E2LSH –MKL. Exact Euclidean LSH- MKL extends the Group Sensitive – MKL by using E2LSH method for clustering instead of K-means method for clustering. Multi – kernel learning (MKL) tries to describe the complex data. The conventional method tries it by applying linear and stationary kernel combination format. E2LSH method makes clusters and partition image into different clusters. E2LSH based multiple kernel learning classifier is trained on these clusters and assigns different weights to different clusters. Hence captures intra class diversity of images effectively. A function set of k – LSH functions defined as: $\mathcal{B} = \{g: S \rightarrow U^k\}$ where $g(v) = \{h_1(v), \dots, h_k(v)\}$ for each $v \in \mathbb{R}^d$ derives a k – dimensional vector a . Then primary and secondary hash functions are used to produce a hash table saving data points. On this data E2LSH performs feature point clustering. E2LSH also supports dynamic expansion of features. It uses the hadamard product to produce nonlinear combination of multiple kernels.

$$K(x_i, x_j) = \sum_{m=1}^{M(M+1)/2} \beta_m^{c(x_i)} \beta_m^{c(x_j)} K_m(x_i, x_j)$$

$$K_m(x_i, x_j) = \begin{cases} K_{m1} \otimes K_{m2}, & m1 \neq m2 \\ K_{m1}, & m1 = m2 \end{cases}$$

M is no. of base kernels and $K_{m1} \otimes K_{m2}$ is the hadamard product of K_{m1} and K_{m2} . $\beta_m^{c(x_i)}$ Represent the m^{th} kernel weight that is derived by statistical property of data group $c(x_i)$. This weighting function is termed as:

$$\beta_m^{c(x_i)} = \frac{\exp(K_m^{c(x_i)} + 1)}{\sum_{m'=1}^{M(M+1)/2} \exp(K_{m'}^{c(x_i)} + 1)}$$

Where $K_m^{c(x_i)}$ represents statistical property of $c(x_i)$ group over m^{th} kernel function.

Guangtao Xue et. al. [44] proposed a scheme to construct structured overlay network by using LSH scheme. This mechanism is termed as TSO and the network is a two layer topology aware network. TSO clustered the physical nodes into P2P ring at local level. This ring is regarded as the virtual node when we take the overall P2P overlay network as the high level chord ring. It reduces the overhead. The LSH used in TSO reduces the mismatching problem. The paper simulates the network to find the results in comparison of the traditional structured overlay.

Joly et. al.[45] proposed an indexing technique used for approximate similarity search called multi probe LSH which probes multiple buckets of hash table. They define posteriori model that takes into account info of queries and objects that are returned which results in controlled search and probable buckets are selected more accurately. They proved that posteriori technique is better multi probe LSH Technique in terms of quality control. Contextual and personalized knowledge can also be adapted as prior information to the algorithm which focuses the retrieval on context aware objects making search more efficient.

M.datar et. al. [46] also proposed scheme for approximate nearest neighbor problem based on LSH using p - stable distribution. They have proved that the algorithm can find exact nearest neighbor in $O(\log n)$ for data that satisfies certain bounded growth condition. Due to the LSH scheme the resulting time bound of the query is free of large factors and is 40 times faster than kd trees. The algorithm is generalized to arbitrary l_p -norm for $p \in [0,2]$. LSH solves the decision version of the NN problem. The same can be done with kd trees but that would require complexity reduction overhead which increases the running time. We can use approximation parameter c to match the LSH result but it returns the results with very guarantee on the reality.

Charikar et. al. [47] uses the fact that LSH scheme leads to compact representation of objects hence estimating similarity between them from these compact sketches. The paper showed the rounding algorithms used in context of approximation algorithm can be done with LSH scheme for LP's and SDP's by finding alternatives to minwise independent permutation for estimation of set similarity and also estimate similarity between two vectors. And they also created scheme for n points distribution on metric

space, with distance is measured by Earth Mover Distance. It map distributions to metric space such that for distribution S and T .

$$EMD(S, T) \leq E_{h \in \mathcal{H}}[d(h(S), h(T))] \leq O(\log n \log \log n) \cdot EMD(S, T)$$

Haghani et. al. [48] maps multidimensional buckets of LSH to linearly ordered set of peers that derive requirements for search results of good quality and for limiting network access and maintaining indexed data jointly. They proposed two mapping techniques such that buckets storing similar values are present nearby and they possess load balancing due to predictable output distribution. These are robust and scalable solutions that create index using proposed mapping and algorithms that find K nearest neighbor and range queries.

Dasgupta et. al. [49] proposed a speed up method of Euclidean Locality sensitive hashing by using randomized hadamard transform in nonlinear nonlinear setting. If L is no. of hash functions are used k is the no of bits in d – dimensions they proposed a method reduce the complexity of LSH from $O(dkL)$ to $O(d \log d + kL)$. They introduced two new algorithms named ACHash and DHHash. ACHash is a modification of the fast Johnson–Lindenstrauss transform (FJLT) which obtain hash buckets by using rounding and thresholding steps. DHHash has two applications of Hadamard transform with collision probabilities as that of LSH. They have also shown results with the help of an analysis method performed over 4 large datasets with 20% query time improvement.

Boyi Xie et. al. [50] proposed a scheme for Semi Supervised agglomerative clustering. They calculated Hamming distance between hashed value of Kernelized Locality sensitive Hashing (KLSH) which decreases the computation time. They have used distance metric learning to get competitive precision and recall comparing to get k means. KLSH preserves neighborhood and provides a reasonable substitutes for exact inter instance distances with high probability and improves precision and recall.

Jeremy Buhler et. al. [51] introduced a new algorithm LSH – ALL – PAIRS it finds non-gapped local alignments in genomic sequence. Genomic DNA sequences are essentially compared to find conserved genome features. The algorithm uses randomized search technique of LSH which makes it sensitive and efficient in finding local

similarities as little as 63% identities in mammalian genomic sequences. However the algorithm guarantees to find only pairs that match exactly while the missed pairs are controlled by increasing the iteration. Theoretically the number of comparisons to be performed can be up to $\theta(lN^2)$ where l is the no. of hash functions. Practically for sequence comparisons combination of l and k is taken where k is the total no. of features.

Tanmoy Mondal et. al. [52] proposed a fast word retrieval technique which deals with heterogeneous document image collection. Gabor features of the word image are extracted which are used for generating hash table for fast retrieval of similar image from large dataset. Algorithm provides a sub linear time similarity search for a wide class of similarity functions. KLSH focused on unknown kernelized visual data and require no assumptions on input or data distribution. They were able to retrieve 50% of the relevant words when only 20% of the words were accessed.

Alexandr Andoni et. al. [53] were able to crank up a notch regarding LSH by introducing a new data structure for c approximate nearest neighbor problem for $\mathbb{R}^d$ they were able to achieve a query time $O_c(n^\rho + d \log n)$ and space complexity of $O_c(n^{1+\rho} + d \log n)$ where $\rho = \frac{7}{8c^2} + O\left(\frac{1}{c^3}\right) + o_c(1)$. This data structure surpasses the lower bound complexity of LSH. The idea that was used in this approach was to hash points so that collision probability of points that are closer to each other becomes much higher than for those which are far apart. Query results are retrieved from the bucket on which the query point falls.

Wang, Shuhui, et al. [36] Due to the fact of SSL semi supervised learning not being able to incorporate multiple information sources and is not suitable for learning on real world dataset due to heavy computation and uncontrolled unlabeled data caused because of no sample selection on unlabeled data. So they proposed a method that imposed LASSO regularization on the kernel coefficients for avoiding over fitting and conditional expectations called scalable semi supervised multiple kernel learning (S^3 MKL). For reducing risk of unlabeled data, they used multi – kernel locality sensitive hashing (MKLSH). Hence a combination of S^3 MKL and MKLSH was presented by them. They were able to prove that this method is suitable for annotation of image and personalized re-ranking.

Hu Xin et. al. [54] explored the contemporary nature of static and dynamic clustering and used clustering ensemble concept to propose DUET method. It improved existing ensemble algorithm to reconcile differences between base malware and clustering by incorporating cluster quality measure. They improved scalability of behavior based clustering by adopting LSH. DUET method unifies static and dynamic clustering results systematically by building a framework. They proved that DUET improved coverage by 20 – 40% with highest precision of the two algorithms.

V.Tomar et.al. [55] Based on manifold learning based technique that are used for feature space transformation and semi supervised learning they proposed an approach to conquer the immense computational requirements of this technique. They applied LSH technique to this method that uses cosine correlation based distance measure. Due to LSH they are able to achieve complexity reduced by a factor of 9 without affecting speech recognition performance. In the paper they populated the affinity matrices in space transformation by using a fast LSH algorithm. They have used Correlation preserving discriminant analysis CDPA that utilizes cosine based distance measure for characterizing domain relationships against feature vectors. Hence they used E2LSH for hashing. They implemented LSH within CDPA framework to achieve computational gains without affecting ASR performance.

Debing Zhang et. al. [56] tried to focus on Nearest neighbor problem and its importance and the solution that are generally provided to fix it is creating a data structure like *kd* - tree, *k* means trees. This paper also proposes a method that combines the advantages of both hashing method's fast computation and an effective data structure. This method easily accelerates the searching procedure. It takes hierarchical *k* means tree and extend its each node into a structure which contains accurate representations and approximate representations in hamming space. Hence it approximated and accelerated the traditional search strategy by introducing the hamming space computation. It is liable to consume $O(1)$ time complexity in modern CPU architecture.

Kristen Grauman et. al. [57] in hashing algorithms we try to learn the binary projections of the data that is hashed. This binary projection is a powerful tool to index large collection according to their content. And we can search data efficiently using hash

tables. They proposed several supervised and unsupervised techniques to generate binary code. Based on spectral analysis and kernelized random projections they try to integrate boosting, metric learning and neural networks into hash key construction. These methods make scalable retrieval for variety of robust similarity measures. These methods were able to provide crucial scalability for usual search problems. The diversity of approaching wide range techniques makes this procedure very much popular.

Yadong Mu et .al. [58] discussed about, usually locality sensitive hashing support metric data but human perception compares reasonably with non – metric distances, so considering this theory they proposed LSH technique that support non metric data. They embed data into an implicit kernel Krein space and then hashed to obtain binary bits and to ensure collision probability reflects the non – metric distance of feature space they used norm keeping property of p – stable function. The symmetric non – metric distances are efficiently interpreted by Krein space theory. It captures information contained in negative singular values by deriving two step LSH function.

Maxim Raginsk et. al. [59] talks about the problem of matching binary codes of vectors that are similar in original space map. For this they introduced simple distribution free, random projection based encoding scheme that relates the hamming distance between binary code two vectors with the value of shift invariant kernels between the vectors. This method makes kernel bandwidth a free parameter, hence providing flexibility to adapt to target neighborhood size. This makes sure that a fraction of neighbors are mapped to the strings for each query. This strategy increases recall for low hamming radius but sacrifice some precision.

Junfeng He et. al. [60] focuses on large scale data of general formats with any kernel function and develop a new hashing algorithm to create codes for it. These kernel functions can be on graphs, sequences, vectors, sets and so on and can be explicitly represented and optimized and applied to compute hash codes for general format samples. They made algorithm scalable to huge data and reduce time and space complexity for indexing and searching by incorporating efficient techniques. This method handles diverse types of similarities (including both feature similarity and semantic similarity), so can be varied according to the task requirement. This paper creates

compact hash codes which can be scaled to huge dataset even consisting samples in millions and that for general type of data for any kernel.

Chenxia Wu et. al. [34] studied under the regularized learning based hashing framework about the semi supervised hashing methods. To capture the relationship among the data points they introduced non – linear hash function which makes the dimensionality of matrix much smaller than the dimensionality of one that use linear hash function and at the same time make it independent of dimensionality of data space. In this paper they proposed bootstrap sequential projection learning method based semi supervised non – linear hashing algorithm that deals with the error accumulated while converting the real value embeddings into binary code by correcting them holistically without any extra computational overhead. It takes into account all the previously learned bits. They retained the projection vector through various iterations as the process of learning with the help of regression matrix that was created with the help of anchors. These anchors are the centers of clusters. Bootstrap learning method is liable to achieve best performance in comparison to other learning methods.

In our work we try to find nearest neighbor in Euclidian distance with the help of kernel function for given dataset of images. The images we use to work on are in the form of extracted feature which is performed using different feature set.

We define nearest neighbor problem as

“given a set P of n points in a d -dimensional space, build a data structure that, given a query point q , reports any point within a given distance r to the query (if one exists).”

In Content based systems (such as recommender system) similarity search plays an important role, as recommender system is all about providing items similar to the users taste. For this a variety of data structures have been proposed for indexing and searching data points in a low dimensional space, but these algorithms becomes less effective in high dimensional space due to their space and time complexity which grows exponentially. This phenomenon is known as “Curse of Dimensionality”. To tackle this problem recent studies focus on approximate approaches that try to remove high dimensional problem. We try to find data points that are at distance $(1 + \epsilon) * r$ from query. As mentioned earlier it is a good practice to specify items with high dimensionality to provide a good recommendation to users. Hence a well known and recent technique called locality sensitive hashing is applied to many applications. It was first proposed by Indyk et.al. for binary hamming space $\{0,1\}^d$ then it was extended for use in Euclidean space by Datar et.al. One limitation of regular LSH is they require vector representation of data explicitly. This limitation is addressed by kernel functions. Kernel functions are capable of capturing similarity between data points. These kernel functions are able to make a framework to process the underlying embeddings of data implicitly. This framework is provided by kernel based methods particularly known as Kernel based LSH In our work we first analyze the previously build kernel function

based locality sensitive hashing which is the building block of our research work. Then we create multiple kernels from the same dataset and try to combine them with equal weights treating each kernel equally. As this approach is naïve, so we implement a learning method that iteratively learn the best combination for kernel functions. Then finally we will use that multi kernel combination to create hash codes for the data and store them in a hash table and then for each query that arrives we compute the hash code for each query and fetch the results from the hash table.

So our problem statement could be stated as:

To create a searching technique for high dimensional data (images) on the basis of similarity by using multiple kernel functions and combining them efficiently by learning the best weight of each kernel function.

Bootstrap and Kernel Based LSH

4.1 Analysis of Kernelized Locality Sensitive Hashing

KLSH is the basic algorithm that synthesizes large multi kernel algorithms. It works with high dimensional dataset very easily and uses a kernel function to create a kernel matrix from the given data. From this kernel matrix we obtain the hash keys for a particular given length. To explain this procedure verbosely we introduce certain technical terms and notations mentioned in table 1.

Symbols	Meanings
X	Matrix of data points of size $d \times n$, where d is dimensionality and n is no. of images.
$P1$	subset of X containing r elements, $r < n$.
$\ker(x_i, x_j)$	An RBF kernel function that defines similarity level between x_i and x_j .
h_i	Hash function for i^{th} bit.
$\phi(x)$	feature set for input x .
K	Kernel matrix made from kernel function.
b	Total no. of hash bits.
$d(x_i, x_j)$	Euclidian distance
U	vector containing cluster centers
G	vector of length m containing mAP of i^{th} kernel g_i where $i=1,2,\dots,m$
θ	Standard deviation
Z	regression matrix
W	Eigen projection matrix
H	hamming matrix

Table 1. Description of notations that are used

Our data points are represented as a feature matrix. Let $X = \mathbb{R}^d$ be a $d \times n$ matrix containing features for n data points. The algorithm efficiently computes the kernel

matrix K from a kernel function $\ker(x_i, x_j)$, for sampled data points that are taken from the original dataset X . Kernel function used is a Gaussian kernel function which is essentially designed to process high dimensional data. The details about this kernel function are explained in next section. The kernel matrix that has been produced acts as a similarity measure for high dimensional data points. Then LSH projects all the data points to hash key $(h_1, h_b, \dots, h_b)$ which is a low dimensional binary space where b is the length of the hash keys. These hash keys are being produced using kernel matrix K . The procedure for creating hash keys is for a data value goes through certain processing. All the images (data value) i.e. the complete set X are picked up one by one through an iterative process. The KLSH function [29] first of all creates a set $p1$, contains r random data points from the dataset of n data points. $p1$ subset of $X = \{x_1, x_2, \dots, x_n\}$. These r data points are then applied on kernel function $\ker(x_i, x_j)$ to produce a kernel matrix K of normalized form where $i, j = 1, 2, \dots, r$. $\ker(x_i, x_j)$ calculates the inner product of x_i, x_j and maps data points x_i , into a functional space. This $\ker(x_i, x_j)$ maps data point x_i through a nonlinear feature mapping function $\phi(x_i)$. This function satisfies the condition $\phi^T(x_i)\phi(x_i) = \ker(x_i, x_j)$.

After that it generates b random vectors relating to the subset $p1$ as $\{e_{p1}^1, e_{p1}^2, \dots, e_{p1}^b\}$ where each random vector e^l where $l = 1, 2, \dots, b$ is calculated as a vector containing t random indices from range $[1, r]$.

Calculate vector $W_y = K^{-1/2} e_y^{p1}$. We will have $W_y = \{w_y^1, w_y^2, \dots, w_y^r\}$ for $y = 1, 2, \dots, b$ Where $K^{-1/2}$ is calculated as svd of K . gives U, V, S matrices then replacing $K = UVU^T$ and $K^{-1/2} = UV^{-1/2}U^T$ Then calculate the hash function:

$$h_y(\phi(x)) = \text{sign}(\sum_{j=1}^r w_y^j \ker(x_i, x_j)) \text{ where } j = 1, 2, \dots, r.$$

When query $q \in \mathbb{R}^d$ arrives to find s similar results or nearest neighbor, this query is also projected on b hash keys using above procedure and then using hash table we find s most approximate nearest neighbor. The details of this procedure are explained in algorithm 1. The trick for making a better implementation of KLSH is by making $r \rightarrow n$ i.e. the sample set is same as the original dataset. The matrix produced is of order $n \times n$ called the gram matrix. This would make the approximation exactly equal to the query.

Algorithm 1: Outlines the procedure of implementing KLSH**Input:**

A database with n images : $\{x_i : i=1,2,\dots,n\}$, $x_i \in \mathbb{R}^d$

Length of the hash key: b

A kernel function $k(\cdot, \cdot)$

Steps:

Select r random data points $p1=\{x_1, x_2, \dots, x_r\}$

Calculate distance matrix D as $d_{i,j} = \min d(x_i, x_j)$ for $j = 1 \dots p, j \neq i$

Calculate standard deviation σ as $\sigma = \sqrt{\frac{\sum (x - \bar{x})^2}{n}}$

Calculate kernel matrix K as $K = [ker(x_i, x_j)]_{p \times p}$

$$\text{Where } ker(i, j) = \exp(-(d(x_i, x_j))^2 / \sigma^2)$$

For l=1,...,b repeat

Form e^l by selecting t indices at random from $[1, \dots, p]$ as $e^l = e_1^l, e_2^l, \dots, e_t^l$.

Form w^l as $= K^{-1/2} \times e^l$ where $\text{svd}(K)=U.V.S$ and $K^{-1/2} = UV^{-1/2}U^T$.

i.e. w^l will be a vector of length p.

Construct hash function as $h_k(\phi(x)) = \text{sign}(\sum_{j=1}^p w_j^k k(x, x_j))$.

End for

Hence the way to achieve high precision make $r \rightarrow n$. but this would make the calculation more complex and hence increase the complexity. This implies a tradeoff has to be done between precision and complexity. So the value of r must be decided appropriately. The analysis of this algorithm has also been done by using various theorems [35] that show kernel similarity is well approximated by $\frac{p}{tb} h_i^T h_j$ the error decreases as the number of bits b increases.

4.2 Multi kernel Locality Sensitive Hashing

The kernel matrix that has been created by algorithm 1 using a particular feature set from r data samples is also created for other feature set on the same data samples. So in this way we have few no. kernel matrices and now we create the hashing bits for each data item using all these kernel matrices so as to include all variety of features in the bits. Let us say if we have m feature set, we will be able to create m kernel matrices according to

algorithm 1. We assign a few no. of hashing bits say b_l to each kernel matrices K_l where $l = 1, 2, \dots, m$ and calculate the value of each bit by creating random vector e^r where $r = 1, 2, \dots, b_l$ and from each vector we create a weight vector w_r using the SVD of the l^{th} kernel matrices.

Algorithm 2: MK-LSH

Input:

A database with n images: $\{x_i : i=1, 2, \dots, n\}$, $x_i \in \mathbb{R}^d$

Bit allocation vector $[b_1, b_2, \dots, b_n]$

A set of m kernel function $\{k_l | l=1, \dots, m\}$

Steps:

$r=0$;

Select r random data points $p1 = \{x_1, x_2, \dots, x_r\}$

For $l=1, \dots, m$

Calculate kernel matrix K_l as $K_l = [ker(x_i, x_j)]_{p \times p}$

For $r=1, \dots, b_l$ **repeat**

Form e^r by selecting t indices at random from $[1, \dots, p]$ as

$$e^r = e_1^r, e_2^r, \dots, e_t^r,$$

Form w^r as $= K_l^{-1/2} \times e^r$ where $svd(K_l) = U.V.S$ and

$$K_l^{-1/2} = UV^{-1/2}U^T$$

i.e. w^r will be a vector of length p

Construct hash function as $h_k(\phi(x)) = sign(\sum_{j=1}^p w_j^k k_l(x, x_j))$

End for

End for

Output: a set of b hash function $\mathcal{H} = \{h_k | k = 1, \dots, b\}$.

Afterwards we calculate the hashing bits of that particular data item using sign function. The details of this procedure are given in Algorithm 2 and figure 1. The total no. of bits b is equal to the sum of all bits allocated to m kernel matrices i.e. $b = \sum_{l=1}^m b_l$. The value b_l that is chosen defines the configuration of the multiple kernels used.

The values b_l learned from algorithm 3 are then applied on MK-LSH algorithm [36].

4.3 Bootstrap sequential projection multi – kernel locality sensitive hashing

There are various methods to combine multiple kernels as we discussed earlier. To make multiple kernel matrices, we use different feature sets to define input matrix X and apply RBF kernel functions. The different feature set used are mentioned in further section. We propose our technique for combining multiple kernels in a more efficient way. Bootstrap sequential projection multi – kernel locality sensitive hashing (BSPMKLSH). This algorithm creates a difference in the error condition as compared to the former boosting approach. BMKLSH judges all the previous bits [35] separately by using boosting technique [38]. But the latter judges all previous bits holistically i.e. error will occur with either positive pairs with large hamming distance or negative pairs with small hamming distance. αk and βk are threshold to determine ‘large’ and ‘small’ hamming distance after ‘ k ’ projections. Our objective is to learn the optimal allocation of bit size to the kernel function.

4.3.1 Bootstrap sequential projection Learning

A clustering method is firstly performed on the similarity distance of all inputs to obtain cluster centers:

$$U = \{u_i \in \mathbb{R}^d\}_i^r \quad (7)$$

That act as anchors. Then the anchor graph defines the truncated similarities defined by Z_{ij} , between r anchors and n data points.

$$Z_{ij} = \begin{cases} \frac{\exp(-\frac{d^2(u_i, g_j)}{\theta})}{\sum_{i' \in r} \exp(-\frac{d^2(u_{i'}, g_j)}{\theta})}, & \text{if } i' \in r \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

It forms a Z matrix that measures the underlying similarities between raw samples and their corresponding data points. $Z(x)$ is x ’s corresponding column of regression matrix Z . By introducing hash functions using anchors creates a non linear hashing function that derives a projection matrix W . A relaxation is introduced through removing binary constraints to the following objective function:

$$\max_W \frac{1}{2} \text{tr}\{W^T Z S Z^T W\} \quad (9)$$

To maximize the information provided by each bit a regularizer is added.

$$W^T Z Z^T W \quad (10)$$

After applying the relaxation we can solve the projection problem by directly obtaining the eigen values of

$$Q = Z S Z^T + \lambda Z Z^T \quad (11)$$

Hence the error condition formulated as after k projections:

$$\begin{aligned} \frac{k}{2} - \sum_{t=1}^k \text{sgn}(w_i^T z(g_i) z(g_j)^T w_t) &> \alpha k, (g_i g_j) \in P \\ \frac{k}{2} - \sum_{t=1}^k \text{sgn}(w_i^T z(g_i) z(g_j)^T w_t) &< \beta k, (g_i g_j) \in N \end{aligned} \quad (12)$$

We represent the error conditions with matrix H^k with data points:

$$H_{ij}^k = \sum_{t=1}^k \text{sgn}(w_i^T z(g_i) z(g_j)^T w_t) \quad (13)$$

In matrix form

$$H^k = \sum_{t=1}^k \text{sgn}(Z^T w_t w_t^T Z) \quad (14)$$

H^k can be calculated recursively as:

$$H^k = H^{k-1} + \text{sgn}(Z^T w_t w_t^T Z) \quad (15)$$

S is a label matrix that defines the relationship between the elements x_i, x_j . In this case we use the similarity degree between these two elements as relationship.

$$S_{ij} = \begin{cases} 1 : \text{if } (g_i, g_j) \in P \\ -1 : \text{if } (g_i, g_j) \in N \\ 0 : \text{otherwise} \end{cases} \quad (16)$$

S_{ij}^1 represent a logical relationship of pair (x_i, x_j) :

$$S_{ij}^1 = \begin{cases} 1 : d^2(g_i, g_j) - \alpha k < 0 \\ -1 : d^2(g_i, g_j) - \beta k > 0 \\ 0 : \text{otherwise} \end{cases} \quad (17)$$

Error condition can be rewritten wrt S_{ij}^1 and H^k

$$\begin{aligned} H_{ij}^k - \alpha k < 0, S_{ij}^1 > 0 \\ H_{ij}^k - \beta k > 0, S_{ij}^1 < 0 \end{aligned} \quad (18)$$

The updating rule for S is

$$S^{k+1} = S^1 + \Delta S^k \quad (19)$$

ΔS^k is increased weight matrix which is determined by:

$$\Delta S_{ij}^k = \begin{cases} \frac{\alpha k - H_{ij}^k}{2k} : H_{ij}^k - \alpha k < 0, S_{ij}^1 > 0 \\ \frac{\beta k - H_{ij}^k}{2k} : H_{ij}^k - \beta k > 0, S_{ij}^1 < 0 \\ 0 : \text{otherwise} \end{cases} \quad (20)$$

After extracting k^{th} projections from Q^k , to minimize redundancy in bits, Z is updated as

$$Z = (I - w_k w_k^T) Z \quad (21)$$

Algorithm 3: BSPMKLSH

Input:

mAP of all kernel functions from algorithm 1 in the form of vector G ,

l = no. of projection iterations required, constant λ

Initialize:

$$H^0 = 0, C^1 = ZZ^T$$

Steps:

Calculate S^1 using equation 17

Calculate Z using equation 8

For $k=1, \dots, l$

$$Q^k = Z S^k Z^T + \lambda C^k$$

Form e as eigen vectors of Q^k set $w_k = e$

$$H^k = H^{k-1} + \text{sgn}(Z^T w_k w_k^T Z)$$

Calculate ΔS^k according to equation 20

$$S^{k+1} = S^1 + \Delta S^k$$

$$U^k = I - w_k w_k^T$$

$$C^{k+1} = U^k C^k U^{kT}$$

Form $W = \{w_1, w_2, \dots, w_l\}$

End For

$$\text{Calculate } b_i = \frac{\sum_l w_l^i}{\sum_{f=1}^m \sum_l w_l^f} \times b.$$

Output: b_i

We recursively calculate the update of ZZ^T to reduce computational effort. Since both C^k and U^k are with the size of $r \times r$, we can efficiently compute the residual. The detailed procedure is summarized in algorithm 3.

Bootstrap learning scheme possess complexity in calculating eigen projection matrix of $O(l.m^2)$ for m kernel functions and l projection rounds. The cost of calculating hamming matrix is $O(m^2)$. The values of b_l learned from algorithm 3 are applied in algorithm 2. To ensure the effectiveness of this algorithm we perform experiment and compare its performance with the rest of the algorithms. The details of this experiment are discussed in next chapter.

5.1. Test Bed

The dataset that we have used to examine our proposed algorithm, called Flickr [37] (known as MIRFLICKR-25000) has been widely used for scalable image retrieval in content based searching. This dataset contains 25,000 images maintained in groups, each having one query image. We implemented our algorithm in matlab language environment on a machine with configuration of 2gb ram intel i3 2.0 GHz processor.

5.2 Experimental Setup

We present our result in both complexity wise and in terms of percentage of data items searched with hashing function. We set parameter ρ to control the fractions of nearest neighbor to be linearly scanned using LSH. Next we calculate performance metric using mean average precision (mAP). Precision value is ratio of relevant examples over total retrieved examples. mAP is the mean of AP of all the queries.

5.2.1 Image Feature extraction

We use 5 different feature extraction methods that are commonly used for describing image.

GABOR filter is a linear filter that is used for edge detection. They are appropriate for texture representation and orientation as their texture representation and discrimination is similar to that of humans.

GIST features [62] provide a statistical summary of spatial envelop representation of the scene to find global image feature.

Color histogram represents image's color distribution in various color spaces. It provides a summary of how data is distributed in an image.

Edge direction histogram converts 2D image into a set of curves and is more compact than pixels. The discontinuities of intensity function of an image are localized.

Local binary pattern labels the pixels with binary number which is determined by threshold the neighborhood of each pixel.

5.2.2 Kernel function

Each image is described by the above mentioned 5 features. Then we build kernel function for each of these 5 features. For this we adopt RBF kernel. This Gaussian kernel function is one kind of radial basis kernel function.

$$ker(x, x') = \exp(-\gamma^{-1}d(x, x'))$$

$d(x, x')$ is the L2 distance. This function can also be written as:

$$ker(x, x') = \exp - \frac{\|x - y\|^2}{2\sigma^2}$$

In this function σ plays a significant role while evaluating performance and is tuned according to problem. It's over estimated value will make look exponential as linear and non linear effect of high dimensional projection will be faded. Whereas the underestimated value will make function lack in regularization and noise in training data highly affects decision boundary.

5.2.3 Experimental Procedure

We firstly started with analyzing of single kernel based LSH and based on that we derived kernel matrix of random data set based on each feature set. The representation of such kind on kernel matrix is represented in Fig 1.

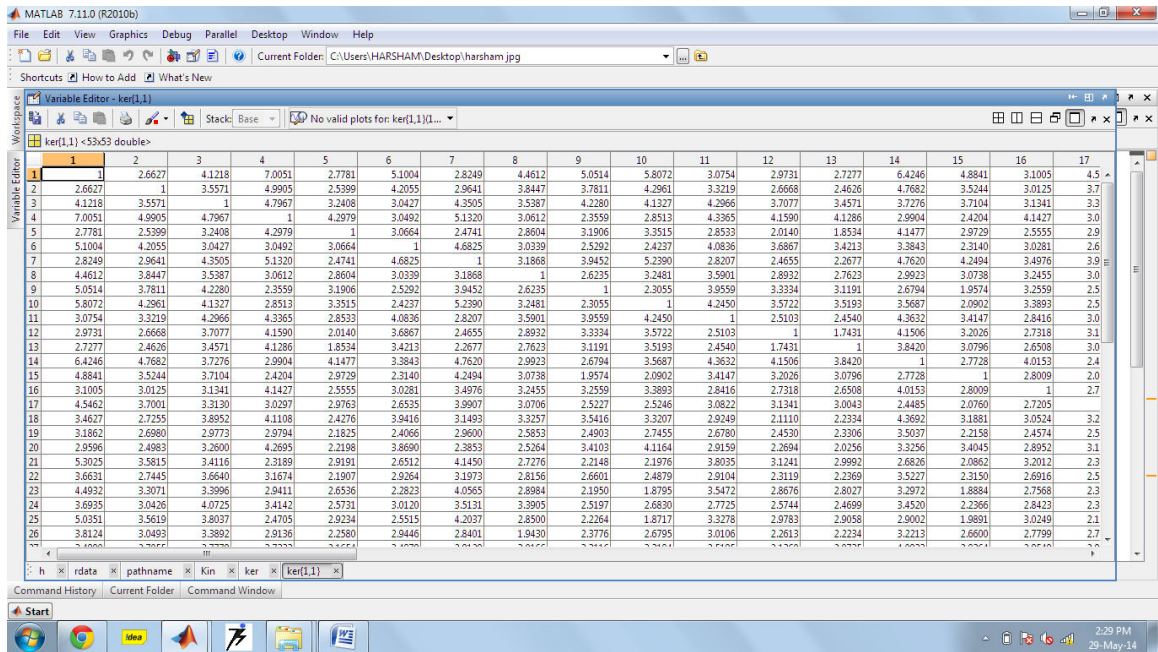


Fig 1. Representation of a 58 x 58 kernel matrix based on single feature set

In the same manner we created a kernel matrix for each feature set and derived the mean Average Precision for each kernel function separately.

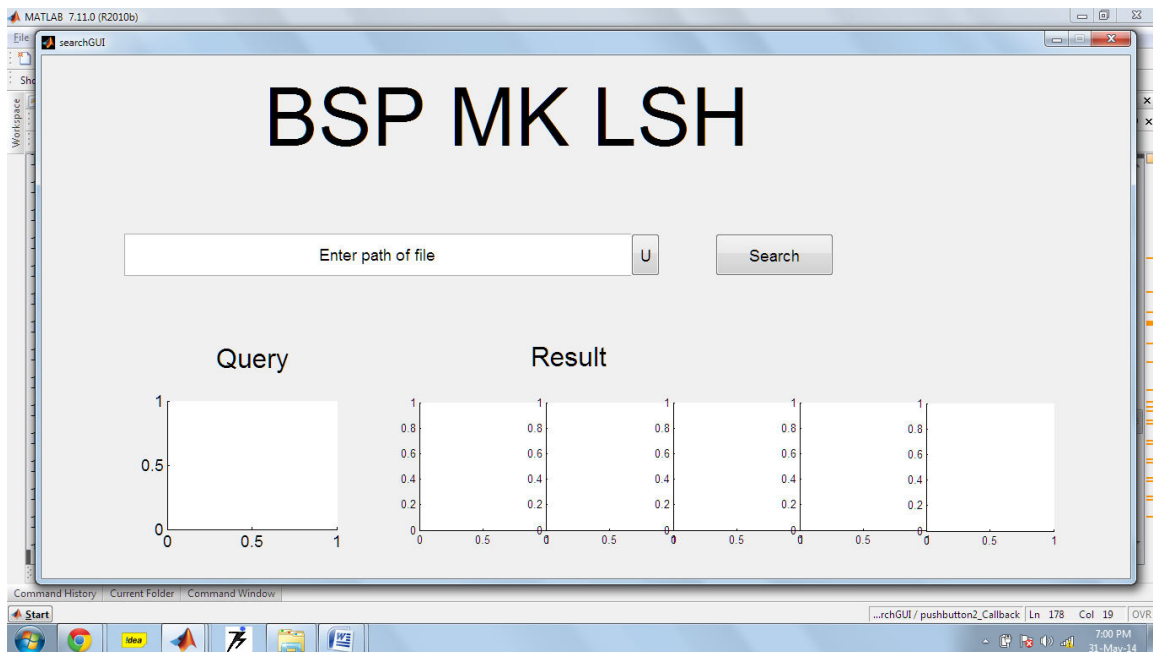


Fig 2. GUI of Bootstrap Sequential Projection Multi kernel Locality Sensitive Hashing

This Mean value precision is utilized to learn the hash bit size of each feature set based on our novel technique of bootstrap sequential projection learning. The GUI of our query processor is shown in fig 2.

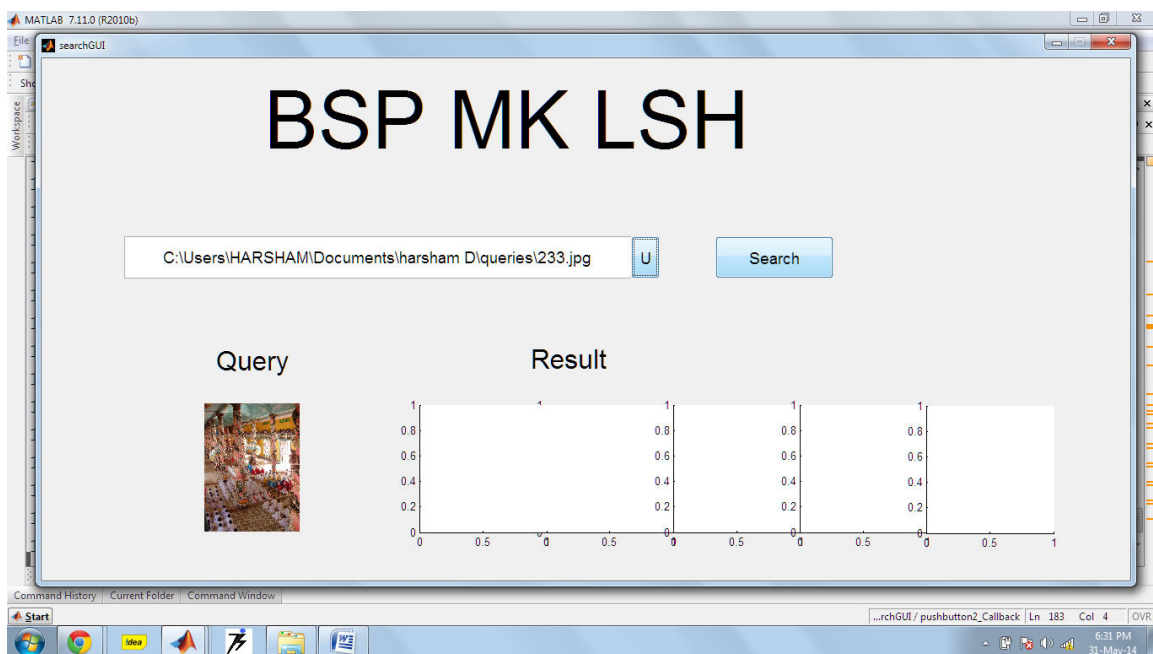


Fig 3. Query extraction and processing

This GUI takes the query as shown in fig 3 and processes it to find its hashing bits according to different kernel functions and evaluate the results. The learned values were then applied on multi kernel LSH algorithm to give efficient results as shown in fig 4.

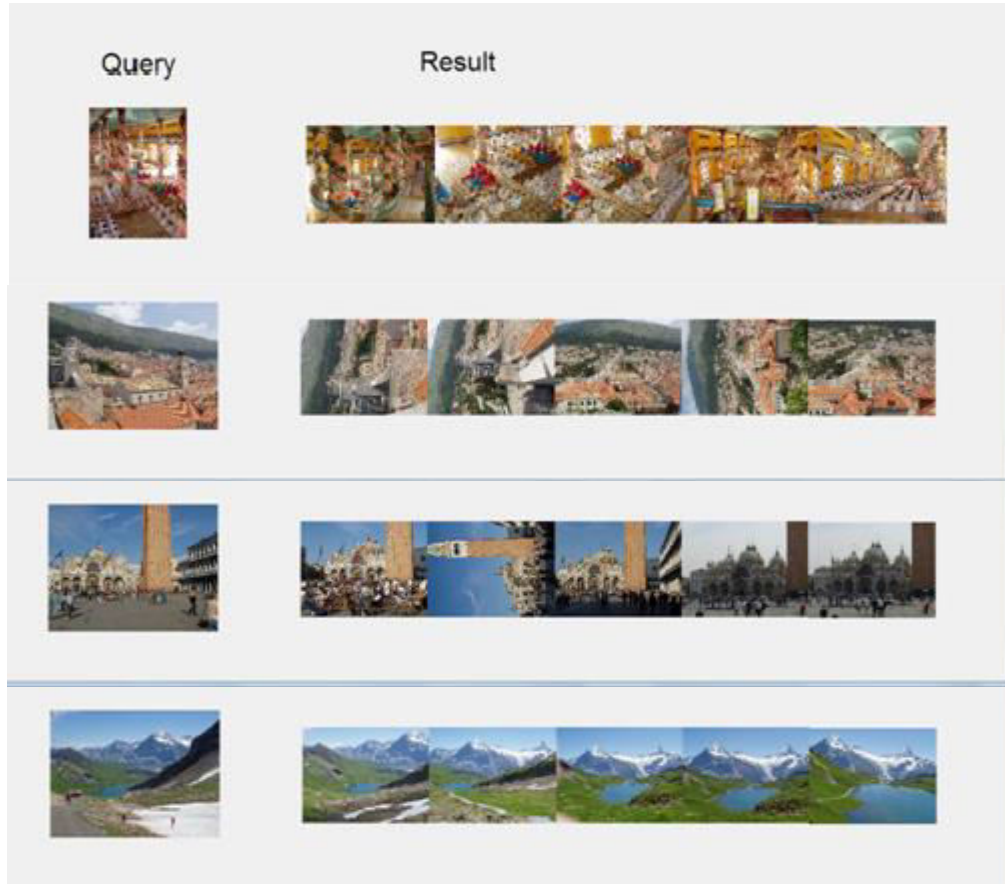


Fig 4. Results of the queries

5.3 Algorithm comparison

As learning approach has only been implemented in BMK-LSH and BSP-MK-LSH, so it would only be fair to compare only these two algorithms. We compare the two algorithms in table 2. l is the number of rounds of projection.

Algorithm	Learning Complexity	Searching Complexity
BMK-LSH	$Tmdn^{1/1+\epsilon}$	m.d.b.
BSP-MK-LSH	$ldn^{1/1+\epsilon}$	m.d.b.

Table 2. Algorithms complexity comparisons

5.4 Experimental Result

We perform experiments in terms of top-n precision where $n = 1; 2; 3; 4$. We fix parameters as $\rho = 0.1$; $b = 100$; $p = 100$; $l = 20$; $t = 30$. Parameter ρ is basically used to show experimental result in the form of database items searched instead of measuring search time. It is not difficult to understand that with increasing ρ value more image examples will be fetched and inspected and also increases the calculations of kernel matrix but after a certain point retrieval of relevant images halt and only resistance is added. As mentioned earlier in section 3 values of b and p represents the hashing bit size and size of subset used for computing kernel matrix respectively which and when they are increased our average precision also increases. Also the values of b and p needs to be same which is clearly visible in the step 8 of algorithm 3 which carries the multiplication of matrices of size $p \times p$ and $b \times 1$. The value of t indices represents the size of vector e^l which is not very sensitive to the algorithm though increasing t would not always increase performance it could also degrade the performance. Next is the value of l which represents the iteration rounds. Through our experiments we have found that l obtains a saturation value after which increasing l introduces no effect.

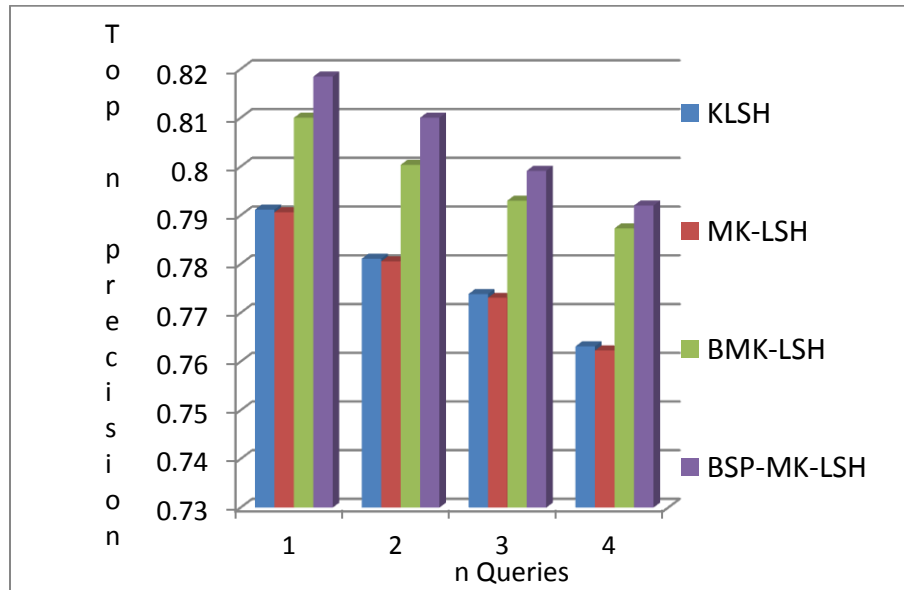


Fig 5. mAP of n query for BSP MK LSH

The algorithm 2 uses a regularizer. The effect of adding a regularizer prevent over fitting of models/features by penalizing them with extreme parameter value. They fine tune the complexity of the model by using augmented error function with cross validation. The data sets used in complex models produces leveling-off of validation as complexity of the models increases. The training data set error decreases while validation data set error remain constant. A second factor introduced by regularizer weighs the penalty against more complex models with an increasing variance in data errors providing increasing penalty as complexity increases. As a result produces fine judgment of size of feature set used than produced from learning by boosting iterations.

Algorithm	mean	std
Simple- KLSH	0.16902	± 0.00100
MK-LSH	0.16761	± 0.00086
BMK-LSH	0.20460	± 0.00400
BSP-MK-LSH	0.21139	± 0.00362

Table 3. mAP and standard deviations of different algorithms

Conclusion

In our thesis we have worked on the basic block of kernel based LSH called Kernelized Locality Sensitive hashing. We analyze this procedure thoroughly and also implement a multi kernel based LSH which does the same work as KLSH but w.r.t. various feature data set. We have proposed a novel method of combining multiple kernels together by learning the optimal combination of the hashing bits. We have done a literature survey to find a good method of high dimensional feature analysis and were able to find a powerful method that combines the multi kernel by assigning them weight that are learned by a boosting technique. Then we introduced a learning algorithm, Bootstrap Sequential Projection that learns the optimal weight of bits for each kernel. Bootstrapping is successfully applied to the hashing bit size learning. This learning algorithm increases its efficiency with increasing number of kernel functions. We have conducted a thorough experiment to evaluate the performance of our proposed algorithm, which concludes this algorithm to be a better approach than boosting multi-kernel LSH.

In Future work we will try to implement our techniques in other domains like e-commerce, textual context etc. This procedure can also be used to learn the compact binary code representation of high dimensional data. Including more and more feature set would make it more versatile. We can also try to embed textual features of the text related to the images and increase its scope of information learning.

References

- [1] John L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509_517, 1975.
- [2] Hannan Samet. *Foundations of Multidimensional and Metric Data Structures*. Elsevier, 2006.
- [3] Roger Weber, Hans J. Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proceedings of the 24th Int. Conf. Very Large Data Bases (VLDB)*, 1998.
- [4] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, “Recommending and Evaluating Choices in a Virtual Community of Use,” *Proc. Conf. Human Factors in Computing Systems*, 1995.
- [5] G. Linden, B. Smith, and J. York, “Amazon.com Recommendations: Item-to-Item Collaborative Filtering,” *IEEE Internet Computing*, Jan./Feb. 2003.
- [6] B.N. Miller, I. Albert, S.K. Lam, J.A. Konstan, and J. Riedl, “MovieLens Unplugged: Experiences with an Occasionally Connected Recommender System,” *Proc. Int’l Conf. Intelligent User Interfaces*, 2003.
- [7] D. Billsus, C.A. Brunk, C. Evans, B. Gladish, and M. Pazzani, “Adaptive Interfaces for Ubiquitous Web Access,” *Comm. ACM*, vol. 45, no. 5, pp. 34-38, 2002.
- [8] W. Wade, “A Grocery Cart that Holds Bread, Butter, and Preferences,” *New York Times*, Jan. 16, 2003.
- [9] M. Balabanovic and Y. Shoham, “Fab: Content-Based, Collaborative Recommendation,” *Comm. ACM*, vol. 40, no. 3, pp. 66-72, 1997.
- [10] P. Brusilovsky and E. Millán, “User models for adaptive hypermedia and adaptive educational systems,” *The adaptive web*, 2007, pp. 3–53

- [11] N. Belkin and B. Croft, "Information Filtering and Information Retrieval," *Comm. ACM*, vol. 35, no. 12, pp. 29-37 1992.
- [12] F. Ricci, L. Rokach, B. Shapira, and K.B. P., "Recommender systems handbook," *Recommender Systems Handbook*, 2011, pp. 1–35.
- [13] A. Gunawardana and G. Shani, "A survey of accuracy evaluation metrics of recommendation tasks," *The Journal of Machine Learning Research*, vol. 10, 2009, pp. 2935–2962.
- [14] J.S. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, July 1998.
- [15] M. Pazzani and D. Billsus, "Learning and Revising User Profiles: The Identification of Interesting Web Sites," *Machine Learning*, vol. 27, pp. 313- 331, 1997
- [16] J.J. Rocchio, "Relevance Feedback in Information Retrieval," *SMART Retrieval System—Experiments in Automatic Document Processing*, G. Salton, ed., chapter 4, Prentice Hall, 1971.
- [17] G. Salton, *Automatic Text Processing*. Addison-Wesley, 1989
- [18] G. Adomavicius and A. Tuzhilin, "Multidimensional Recommender Systems: A Data Warehousing Approach," *Proc. Second Int'l Workshop Electronic Commerce WELCOM '01*), 2001b.
- [19] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.
- [20] N. Good, J.B. Schafer, J.A. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl, "Combining collaborative filtering with personal agents for better recommendations," *Proceedings of the National Conference on Artificial Intelligence*, JOHN WILEY & SONS LTD, 1999, pp. 439–446
- [21] N. Littlestone and M. Warmuth, "The Weighted Majority Algorithm," *Information and Computation*, vol. 108, no. 2, pp. 212- 261, 1994.

- [22] M. Pazzani, “A Framework for Collaborative, Content-Based, and Demographic Filtering, *Artificial Intelligence Rev.*, pp. 393-408, Dec. 1999.
- [23] Y. Zhang, J. Callan, and T. Minka, “Novelty and Redundancy Detection in Adaptive Filtering,” *Proc. 25th Ann. Int’l ACM SIGIR Conf.*, pp. 81-88, 2002.
- [24] P. Indyk and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.
- [25] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.
- [26] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD Conference*, pages 563–576, 2009
- [27] Q. Lv, W. Josephson, Z. Wang, M. S. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*. Vienna, Austria, 2007.
- [28] J. He, S.-F. Chang, R. Radhakrishnan, and C. Bauer. Compact hashing with joint optimization of search accuracy and time. In *CVPR*, pages 753–760, 2011.
- [29] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.
- [30] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, pages 1509–1517, 2009.
- [31] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *KDD*, pages 1129–1138, 2010.
- [32] S. C. H. Hoi and M. R. Lyu. A multimodal and multilevel ranking scheme for large-scale video retrieval. *IEEE Transactions on Multimedia*, 10(4):607–619, 2008.
- [33] J. Zhuang, T. Mei, S. C. H. Hoi, X.-S. Hua, and S. Li. Modeling social strength in social media community via kernel-based learning. In *ACM Multimedia*, pages 113–122, 2011. 64

- [34] C. Wu, Jianke Zhu, Deng Cai, Chun Chen, and Jiajun Bu, "Semi-supervised Nonlinear Hashing Using Bootstrap Sequential Projection Learning", Knowledge and Data Engineering, IEEE Transactions on (Volume:25 , Issue: 6), 10,1380-1393, june 13
- [35]H. Xia, Pengcheng Wu, S. C.H. Hoi and R. Jin, "Boosting Multi-Kernel Locality-Sensitive Hashing for Scalable Image Retrieval", SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.2012 ACM 978-1-4503-1472-5/12/08
- [36] S. Wang, S. Jiang, Q. Huang, and Q. Tian. S3mkl: scalable semi-supervised multiple kernel learning for image data mining. In ACM Multimedia, pages 163–172, 2010.
- [37] M. J. Huiskes and M. S. Lew, "The mir flickr retrieval evaluation," in MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval. New York, NY, USA: ACM, 2008.
- [38] J. Wang, S. Kumar, and S.-F. Chang. "Sequential Projection Learning for Hashing with Compact Codes". In Proceedings of the 27th International Conference on Machine Learning (ICML), 2010.
- [39] Satuluri, Venu, and Srinivasan Parthasarathy. "Bayesian locality sensitive hashing for fast similarity search." Proceedings of the VLDB Endowment 5.5 (2012): 430-441.
- [40] Gan, Junhao, et al. "Locality-sensitive hashing scheme based on dynamic collision counting." Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012.
- [41] Bahmani, Bahman, Ashish Goel, and Rajendra Shinde. "Efficient distributed locality sensitive hashing." Proceedings of the 21st ACM international conference on Information and knowledge management. ACM, 2012.
- [42] Sundaram, Narayanan, et al. "Streaming similarity search over one billion tweets using parallel locality-sensitive hashing." Proceedings of the VLDB Endowment 6.14 (2013): 1930-1941.
- [43] Zhang, Ruijie, Fushan Wei, and Bicheng Li. "E2LSH based multiple kernel approach for object detection." Neurocomputing 124 (2014): 105-110.

- [44] Xue, Guangtao, et al. "A topology-aware hierarchical structured overlay network based on locality sensitive hashing scheme." Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks. ACM, 2007.
- [45] Joly,Alexis, and Olivier Buisson. "A posteriori multi-probe locality sensitive hashing." Proceedings of the 16th ACM international conference on Multimedia. ACM, 2008.
- [46] Datar, Mayur, et al. "Locality-sensitive hashing scheme based on p-stable distributions." Proceedings of the twentieth annual symposium on Computational geometry. ACM, 2004.
- [47] Charikar, Moses S. "Similarity estimation techniques from rounding algorithms."Proceedings of the thirty-fourth annual ACM symposium on Theory of computing. ACM, 2002.
- [48] Haghani, Parisa, Sebastian Michel, and Karl Aberer. "Distributed similarity search in high dimensions using locality sensitive hashing." Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. ACM, 2009.
- [49] Dasgupta, Anirban, Ravi Kumar, and Tamas Sarlos. "Fast locality-sensitive hashing." Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011.
- [50] Xie, Boyi, and Shuheng Zheng. "Kernelized Locality-Sensitive Hashing for Semi-Supervised Agglomerative Clustering." arXiv preprint arXiv:1301.3575 (2013).
- [51] Buhler, Jeremy. "Efficient large-scale sequence comparison by locality-sensitive hashing." Bioinformatics 17.5 (2001): 419-428.
- [52] Mondal, Tanmoy, et al. "A fast word retrieval technique based on kernelized locality sensitive hashing." Document Analysis and Recognition (ICDAR), 2013 12th International Conference on. IEEE, 2013.
- [53] Andoni, Alexandr, et al. "Beyond locality-sensitive hashing." arXiv preprint arXiv:1306.1547 (2013).

- [54] Hu, Xin, and Kang G. Shin. "DUET: integration of dynamic and static analyses for malware clustering with cluster ensembles." Proceedings of the 29th Annual Computer Security Applications Conference. ACM, 2013.
- [55] Tomar, Vikrant Singh, and Richard C. Rose. "Locality Sensitive Hashing for Fast Computation of Correlational Manifold Learning based Feature space Transformations," Proc. Interspeech. 2013.
- [56] Zhang, Debing, et al. "A unified approximate nearest neighbor search scheme by combining data structure and hashing." Proceedings of the Twenty-Third international joint conference on Artificial Intelligence. AAAI Press, 2013.
- [57] Grauman, Kristen, and Rob Fergus. "Learning binary hash codes for large-scale image search." Machine Learning for Computer Vision. Springer Berlin Heidelberg, 2013. 49-87.
- [58] Mu, Yadong, and Shuicheng Yan. "Non-Metric Locality-Sensitive Hashing." AAAI. 2010.
- [59] Raginsky, Maxim, and Svetlana Lazebnik. "Locality-sensitive binary codes from shift-invariant kernels." NIPS. Vol. 22. 2009.
- [60] He, Junfeng, Wei Liu, and Shih-Fu Chang. "Scalable similarity search with optimized kernel hashing." Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2010.
- [61] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. OSDI '04.
- [62] Aude Oliva, Antonio Torralba." Modeling the shape of the scene: a holistic representation of the spatial envelope" International Journal of Computer Vision, Vol. 42(3): 145-175, 2001

List of Publications

- [1] Harsham Mehta and Deepak Garg, “Bootstrap Sequential Projection Multi Kernel Locality Sensitive Hashing,” 3rd International Conference on Advances in Computing, Communications and Informatics (ICACCI-2014), IEEE, 2014 [Accepted]