Hashing Hyperplane Queries to Near Points with Applications to Large-Scale Active Learning
Prateek Jain, Sudheendra Vijayanarasimhan, and Kristen Grauman
Microsoft Research Lab, Bangalore, INDIA, 2University of Texas, Austin, TX, USA

Motivation
For large-scale active learning, want to repeatedly query annotators to label the most uncertain examples in a massive pool of unlabeled data.
Margin-based selection criterion for SVMs [Tong & Koller, 2000] selects points nearest to current decision boundary:

\[ x = \text{argmin}_{x \in \mathcal{D}} |w^T x| \]

10% Accounting for all costs
10% g in AUROC
5% Improvement
Selection + labeling time
15%
Learning curves
Selection time
AUROC
g scale
10% 5%
Hence, can return a point for which \(|v_i - v_j|\) is small, we want collisions to be probable for vectors perpendicular to hyperplane (assuming normalized data).

Unlikely to split x and w
\( h(x) \neq h(w) \)
Likely to split x and w
\( h(x) = h(w) \)

Main Idea: Sub-linear Time Active Selection
Idea: Define two hash function families that are locality-sensitive for the nearest neighbor to a hyperplane query search problem. The two variants offer trade-offs in error bounds versus computational cost.

First Solution: Hyperplane Hash
Intuition: To retrieve those points for which \(|w^T x| \) is small, we want collisions to be probable for vectors perpendicular to hyperplane (assuming normalized data).

\[ h_i(z) = \begin{cases} h_u(a, z), & \text{if } z \text{ is a database point vector}, \\ h_v(z, -z), & \text{if } z \text{ is a query hyperplane vector}. \end{cases} \]

where \( h_u(a, b) = \text{sign}(|a^T b|), \text{sign}(|b^T a|) \), is a two-bit hash, and \( u, v \sim \mathcal{N}(0, I) \).

\[ h(x) = \frac{1}{2} \left( 1 - \frac{1}{\sqrt{\epsilon}} \right) \parallel x \parallel^2 \]

\[ 
\text{Probability of collision between } w \text{ and } x \text{ is given by } 
Pr(h_i(w) = h_i(x)) = \frac{1}{2} \left( 1 - \frac{1}{\sqrt{\epsilon}} \right) \parallel x \parallel^2 
\]

\[ 
\text{Hence, can return a point for which } |v_i - v_j| < \epsilon \text{ in sub-linear time } O(N^2). 
\]

Second Solution: Embedded Hyperplane Hash
Intuition: Design Euclidean embedding after which minimizing distance is equivalent to minimizing \(|w^T x|\), making existing approx. NH methods applicable.

Using k-bit hash keys for each point \( p \):

\[ h_i(p) = h_i^[1](p), h_i^[2](p), \ldots, h_i^[k](p) \]

Provide a query \( q \), search over examples in the \( l \) buckets to which \( q \) hashes.

- Use \( l = N^ \frac{1}{l} \) hash tables for \( N \) points, where \( \frac{k}{l} = 1 \).
- A \((1-\epsilon)\)-approximate solution is retrieved in time \( O(N^2)\).

Definition 1. LSH functions [Gionis, Indyk, & Motwani, 1999]

Let \( h_i \) denote a random choice of a hash function from the family \( \mathcal{H} \). The family \( \mathcal{H} \) is called \((r, r(1 - \epsilon), p_1, p_2)\)-sensitive for \( d, \) when, for any \( a, b \in S \):

- If \( p \in B(q, r(1 - \epsilon)) \) then \( Pr[h_i(q) = h_i(p)] \leq p_1 \).
- If \( p \in B(q, r) \) then \( Pr[h_i(q) = h_i(p)] \leq p_2 \).

\[ 
\text{Compute } k \text{-bit hash keys for each point } p: 
( h_i^[1](p), h_i^[2](p), \ldots, h_i^[k](p) ) \]

Given a query \( q \), search over examples in the \( l \) buckets to which \( q \) hashes.

- Use \( l = N^ \frac{1}{l} \) hash tables for \( N \) points, where \( \frac{k}{l} = 1 \).
- A \((1-\epsilon)\)-approximate solution is retrieved in time \( O(N^2)\).

Definition 2. Hyperplane Hash (H-Hash) Functions

We define H-Hash function family \( \mathcal{H} \) as:

\[ h_i(z) = \begin{cases} h_u(a, z), & \text{if } z \text{ is a database point vector}, \\ h_v(z, -z), & \text{if } z \text{ is a query hyperplane vector}. \end{cases} \]

where \( h_u(a, b) = \text{sign}(|a^T b|), \text{sign}(|b^T a|) \), is a two-bit hash, and \( u, v \sim \mathcal{N}(0, I) \).

\[ h(x) = \frac{1}{2} \left( 1 - \frac{1}{\sqrt{\epsilon}} \right) \parallel x \parallel^2 \]

\[ 
\text{Probability of collision between } w \text{ and } x \text{ is given by } 
Pr(h_i(w) = h_i(x)) = \frac{1}{2} \left( 1 - \frac{1}{\sqrt{\epsilon}} \right) \parallel x \parallel^2 
\]

\[ 
\text{Hence, can return a point for which } |v_i - v_j| < \epsilon \text{ in sub-linear time } O(N^2). 
\]

Definition 3. Embedded Hyperplane Hash (EH-Hash) Functions

We define EH-Hash function family \( \mathcal{E} \) as:

\[ h_i(z) = \begin{cases} h_u(V(z)), & \text{if } z \text{ is a database point vector}, \\ h_v(V(z)), & \text{if } z \text{ is a query hyperplane vector}, \end{cases} \]

where \( V(a) = \text{vec}(aa^T) = [a_1, a_2, \ldots, a_n, a_2, a_3, \ldots, a_n] \) gives the embedding, and \( h_u(b) = \text{sign}(|b^T u|), \text{sign}(|u^T b|) \) with \( u \in \mathbb{R}^d \) sampled from \( N(0, I) \).

Embedding inspired by [Basri et al., 2009]; we give LSH bounds for \( (k_u - \pi/2)^2 \).

- Since \( |V(x) - V(w)|^T = 2 + 2|x^T w| \), distance between embeddings of \( x \) and \( w \) proportional to desired distance, so standard LSH function \( h_u(b) \) applicable.
- Probability of collision between \( w \) and \( x \) is given by

\[ 
Pr(h_i(w) = h_i(x)) = \frac{1}{2} \left( 1 - \frac{1}{\sqrt{\epsilon}} \right) \parallel x \parallel^2 
\]

and we have \( p_t = \frac{1}{2} \left( 1 - \frac{1}{\sqrt{\epsilon}} \right) \parallel x \parallel^2 \).

Hence, sub-linear time search with about twice the \( p_t \) guaranteed by H-Hash.

- Issue: \( V(a) \) is \( d \)-dimensional, higher hashing overhead.
- Solution: Compute \( h_u(b) V(a) \) approximately using randomized sampling:

Lemma 4. Sampling to Approximate Inner Product
Let \( v \in \mathbb{R}^d \), define \( p_t = \frac{1}{2} \left( 1 - \frac{1}{\sqrt{\epsilon}} \right) \parallel x \parallel^2 \). Construct \( v \in \mathbb{R}^d \) such that the \( i \)-th element is \( v_i \) with probability \( p_t \) and is 0 otherwise. Select \( t \) such elements using sampling with replacement. Then, for any \( y \in \mathbb{R}^d \), \( \epsilon > 0 \), \( c \geq 1 \), \( t \geq \frac{1}{\epsilon} \),

\[ 
Pr[|\langle v, y \rangle | \leq \epsilon |\langle v, y \rangle |^2 ] < 1 - c/t \]

Trade-off: EH-Hash has faster pre-processing, but EH-Hash has stronger bounds.

Experimental Results
- Goal: Show that proposed algorithms can select examples nearly as well as the exhaustive approach, but with substantially greater efficiency.
- Counting for both selection and labeling time, our approach performs better than either random selection or exhaustive active selection.
- In future work, we plan to explore extensions for non-linear kernels.