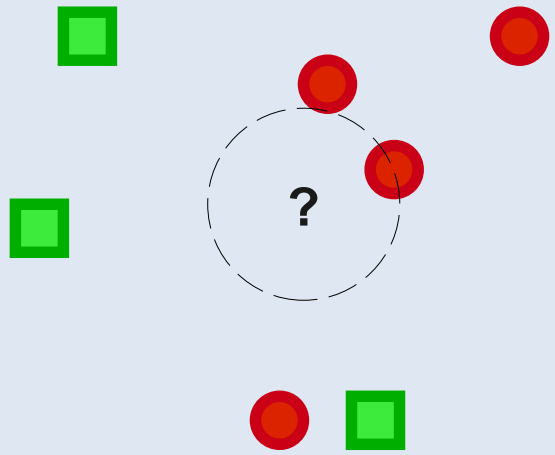
Locality-Sensitive Hashing

CS 395T: Visual Recognition and Search

Marc Alban

Nearest Neighbor

- Given a query any point q, return the point closest to q.
- Useful for finding similar objects in a database.
- Brute force linear search is not practical for massive databases.

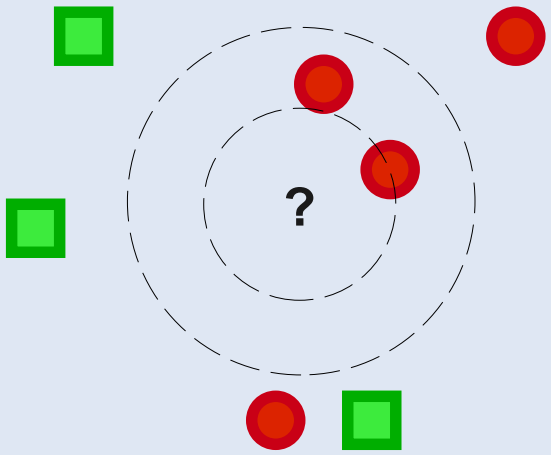


The "Curse of Dimensionality"

- For $d < 10 \text{ to } 20$, data structures exist that require sublinear time and near linear space to perform a NN search.
- Time or space requirements grow exponentially in the dimension.
- The dimensionality of images or documents is usually in the order of several hundred or more.
  - Brute force linear search is the best we can do.

(r, ϵ)-Nearest Neighbor

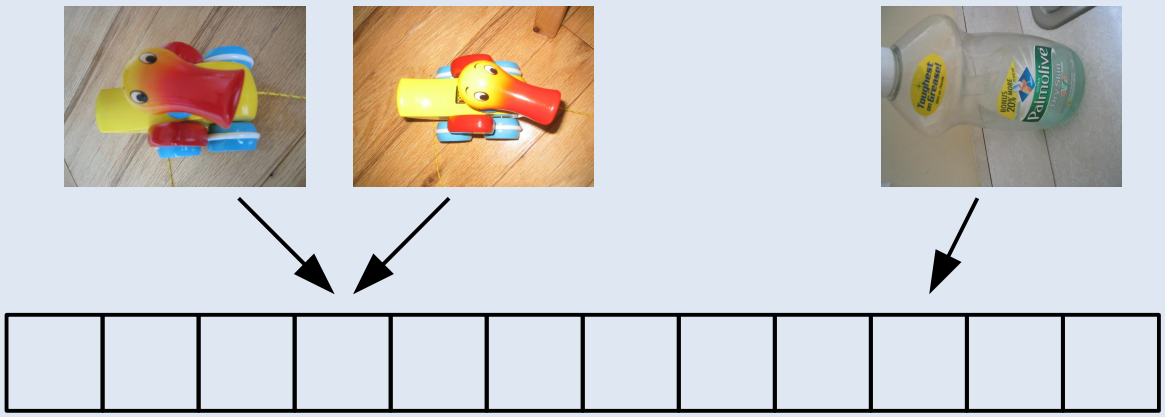
- An approximate nearest neighbor should suffice in most cases.
- **Definition:** If for any query point q, there exists a point p such that $||q - p|| \leq r$, w.h.p return p' such that $||q - p'|| \leq (1 + \epsilon)\, r$.



Locality-sensative Hash Families

Definition: A *LSH family*, $\mathcal{H}(c, r, P_1, P_2)$, has the following properties for any $q, p \in S$:

1. If $||p - q|| \leq r$ then $Pr_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
2. If $||p - q|| \geq cr$ then $Pr_{\mathcal{H}}[h(p) = h(p)] \leq P_2$



Hamming Space

- **Definition:** *Hamming space* is the set of all 2^N binary strings of length N.
- **Definition:** The *Hamming distance* between two equal length binary strings is the number of positions for which the bits are different.

$$\|1011101, 1001001\|_H = 2$$

$$\|1110101, 1111101\|_H = 1$$

Hamming Space

- Let a hashing family be defined as $h_i(p) = p_i$ where p_i is the i^{th} bit of p.

$$Pr_{\mathcal{H}}\left[h\left(p\right) \neq h\left(q\right)\right] = \frac{\|p,q\|_H}{d}$$

$$Pr_{\mathcal{H}}\left[h\left(p\right) = h\left(q\right)\right] = 1 - \frac{\|p,q\|_H}{d}$$

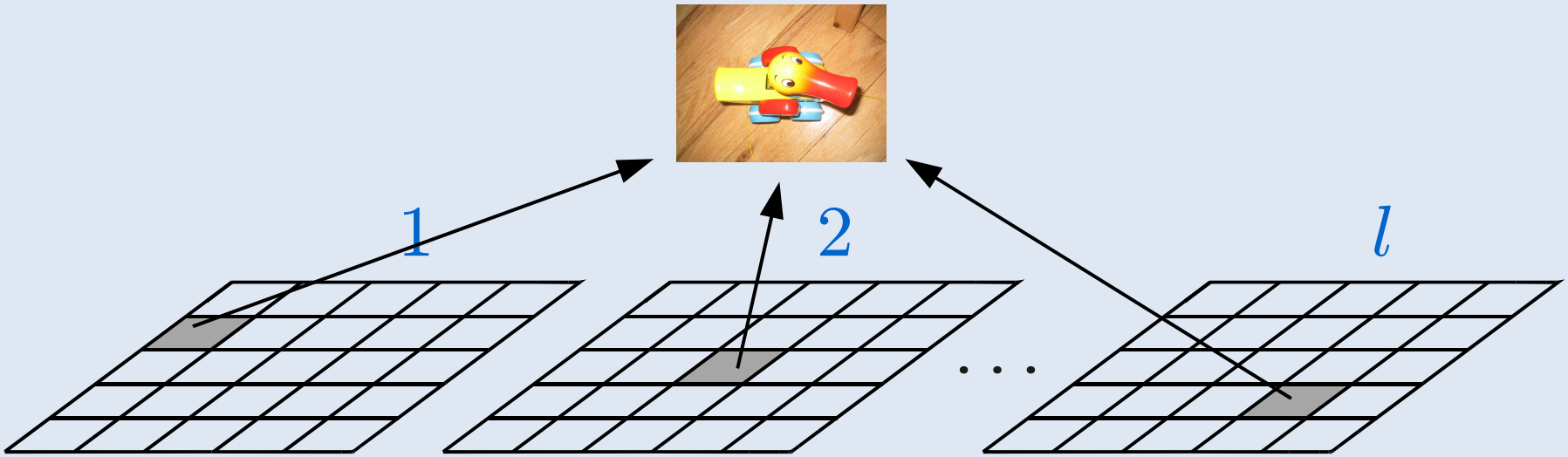
Clearly, this family is locality sensative.

k-bit LSH Functions

- A k-bit locality-sensitive hash function (LSHF) is defined as: $g(p) = [h_1(p), h_2(p), \ldots, h_k(p)]^{\mathrm{T}}$
  - Each h_i is chosen randomly from $\mathcal{H}$.
  - Each h_i results in a single bit.
- Pr(similar points collide) $\geq 1 - \left(1 - \frac{1}{P_1}\right)^k$
- Pr(dissimilar points collide) $\leq P_2^k$

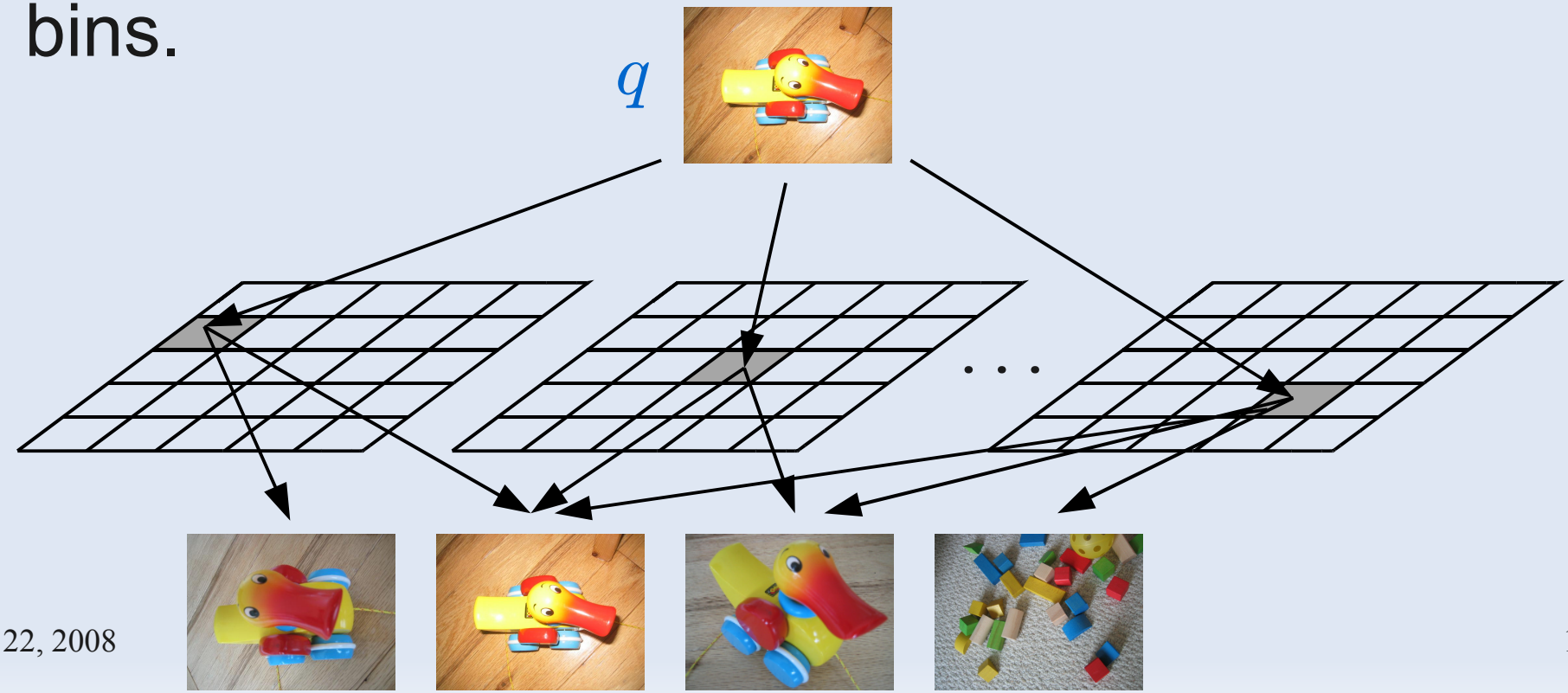
LSH Preprocessing

- Each training example is entered into l hash tables indexed by independantly constructed $g_1, \ldots, g_l$.
- Preprocessing Space: $O(lN)$



LSH Querying

- For each hash table i, $1 \leq i \leq l$
  - Return the bin indexed by $g_i(q)$
- Perform a linear search on the union of the bins.



Parameter Selection

- Suppose we want to search at most B examples. Then setting

$$k = \log_{1/P_2}\left(\frac{N}{B}\right), l = \left(\frac{N}{B}\right)^{\frac{\log(1/P_1)}{\log(1/P_2)}}$$

ensures that it will succeed with high probability.

Experiment 1

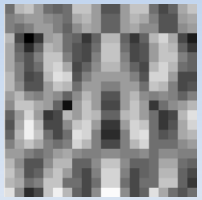
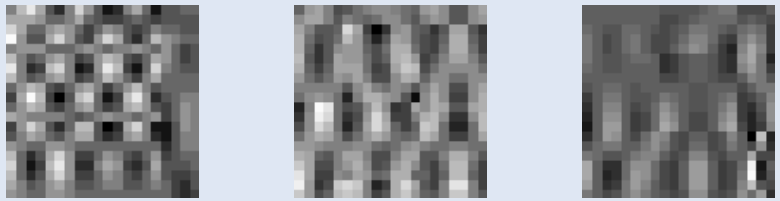
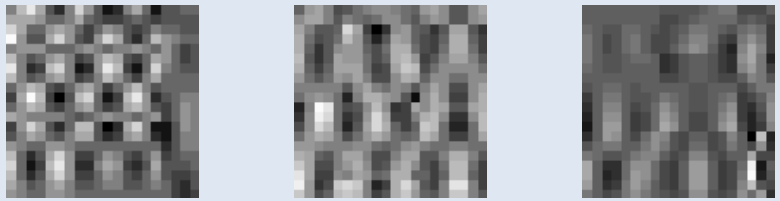
- Compare LSH accuracy and performance to exact NN search. Examine the influence of:
  - k, the number of hash bits.
  - l, the number of hash tables.
  - B, the maximum search length.
- Dataset
  - 59500 20x20 patches taken from motorcycle images.
  - Represented as 400-dimensional column vectors

Hash Function

- Convert the feature vectors into binary strings and use the Hamming hash functions.
- Given a vector $x \in \mathbb{N}^d$ we can create a unary representation for each element x_i.
- $Unary_C(x_i)$ = x_i 1's followed by $(C - x_i)$ 0's, where C is the max coordinate for all points.
- $u(x) = Unary_C(x_1), \ldots, Unary_C(x_d)$
- Note that for any two points p, q :

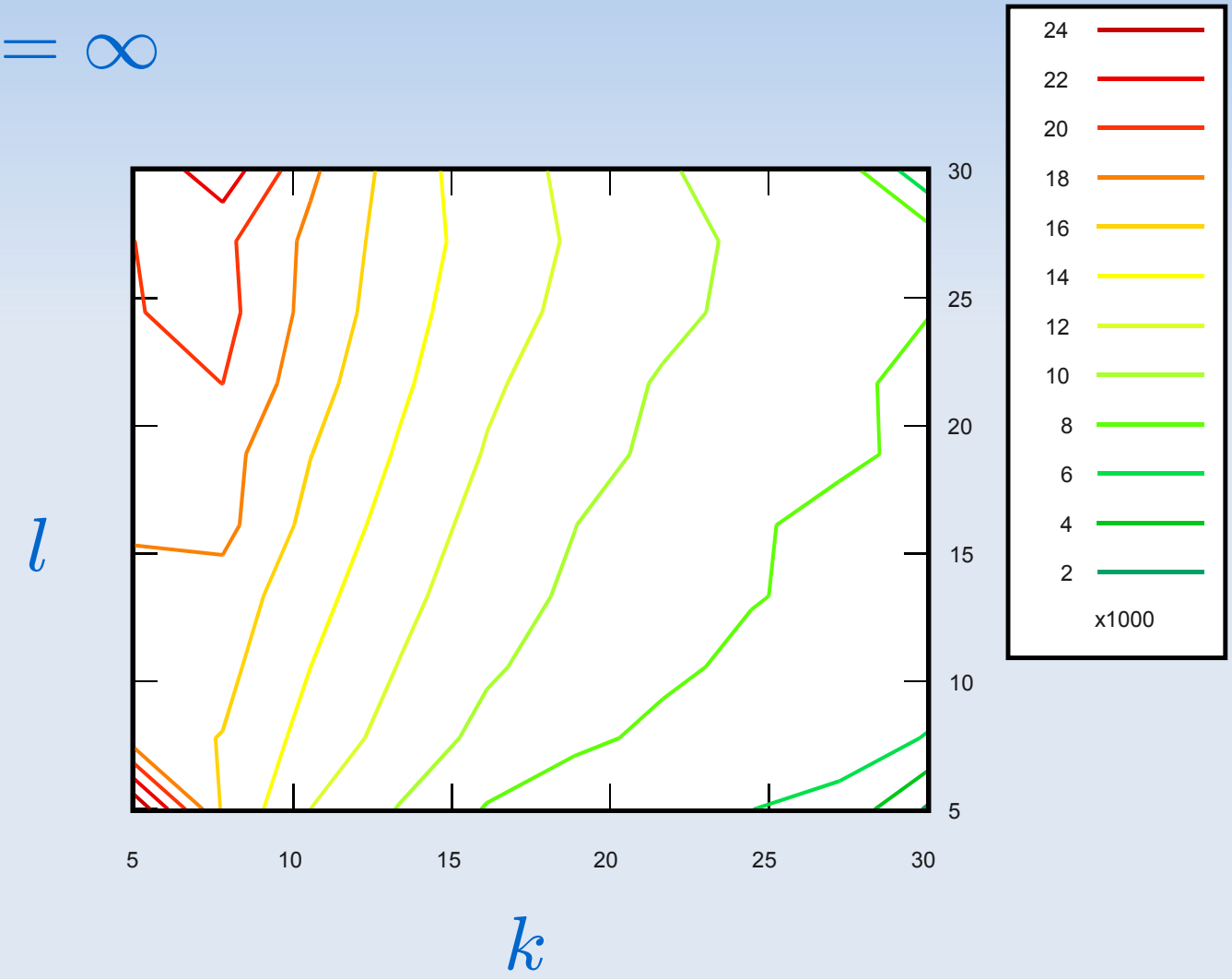
$$\|p, q\| = \|u(p), u(q)\|_H$$

Example Query

- $l = 20,\ k = 24,\ B = \infty$
- Query = 
- Examples searched: 7,722 of 59,500
- Result = 
- Actual NNs = 

Average Search Length

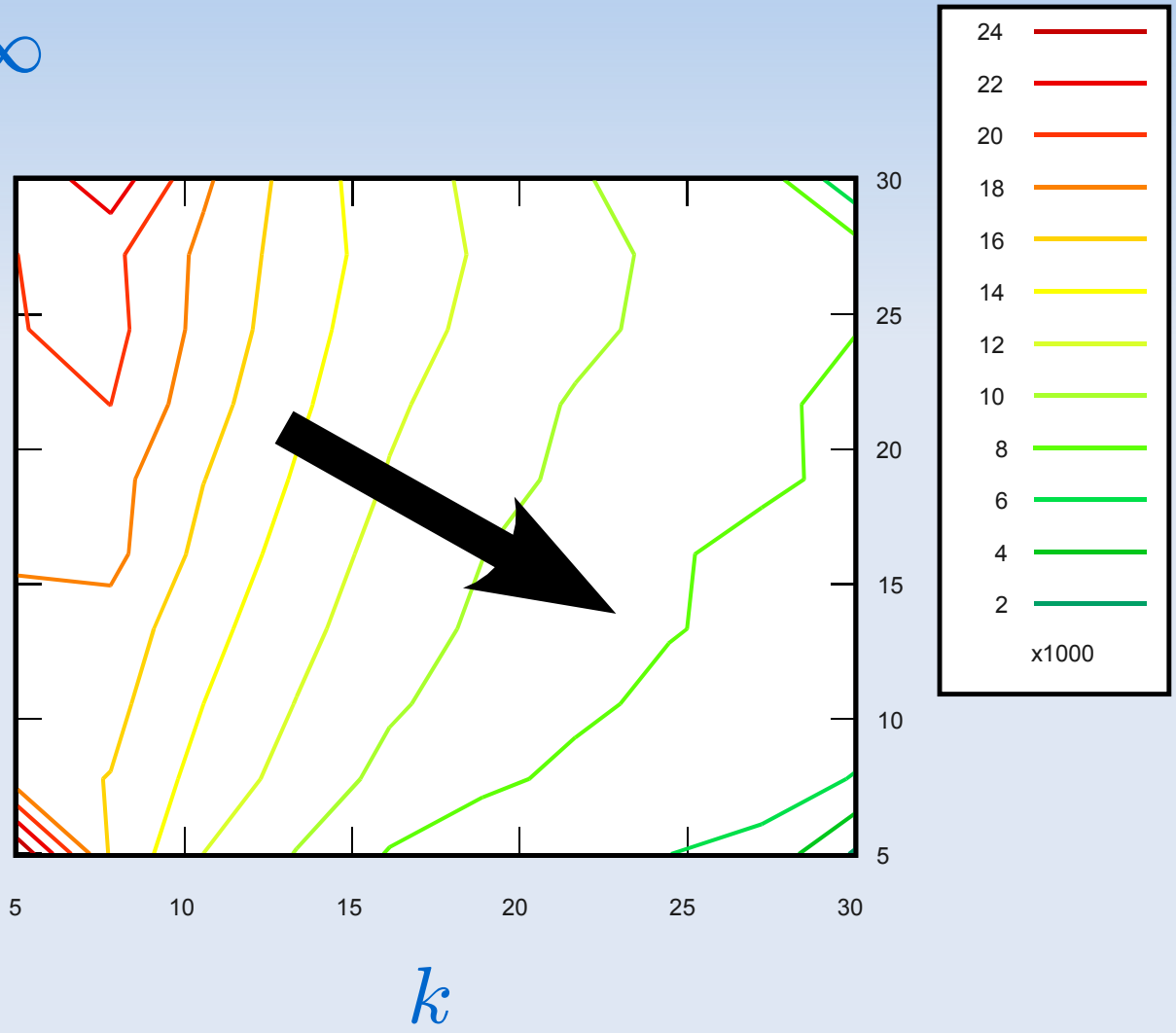
- Let $B = \infty$



Average Search Length

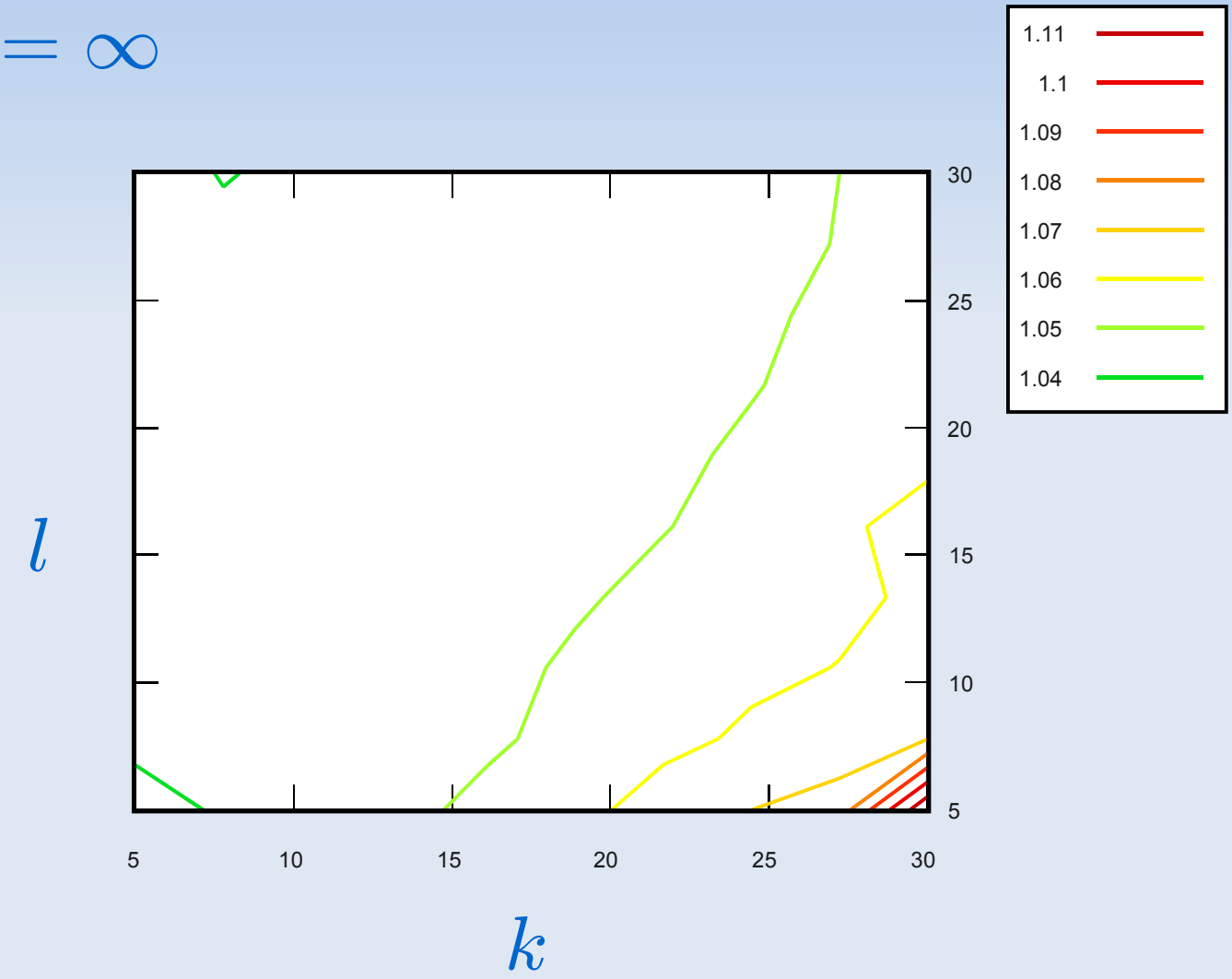
- Let $B = \infty$

- More hash bits, (k), result in shorter searches.
- More hash tables (l), result in longer searches.



Average Approximation Error

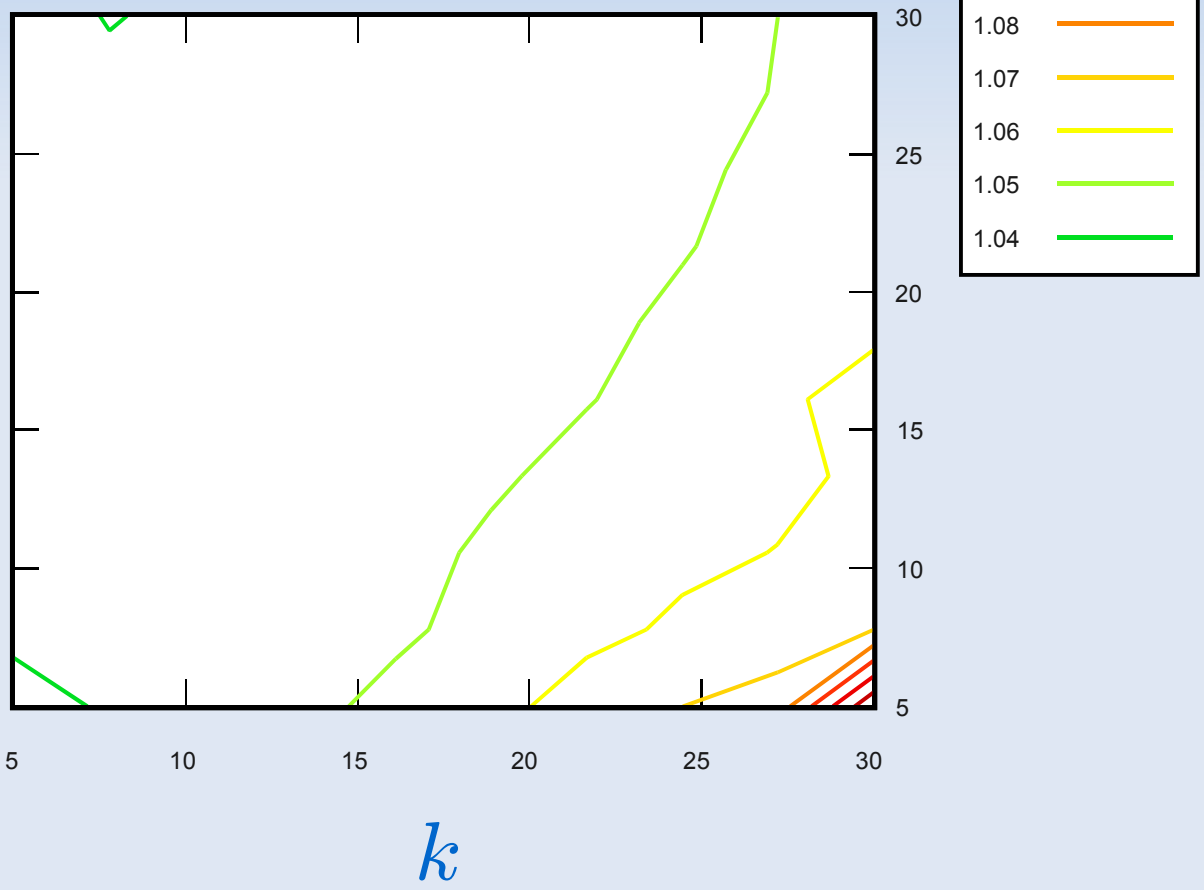
- Let $B = \infty$



Average Approximation Error

- Let $B = \infty$

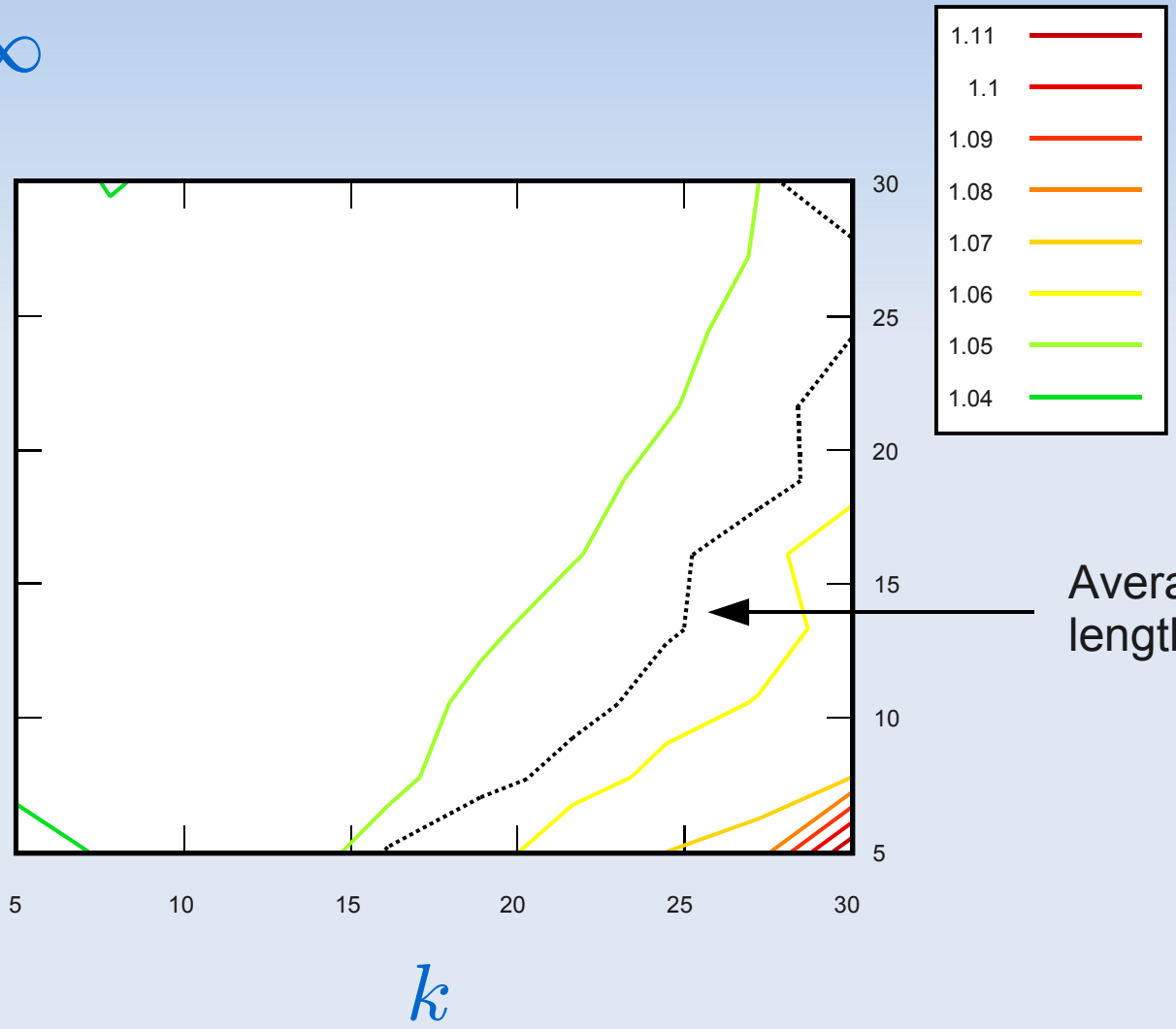
- Over hashing can result in too few candidates to return a good approximation.
- Over hashing can cause algorithm to fail.



Average Approximation Error

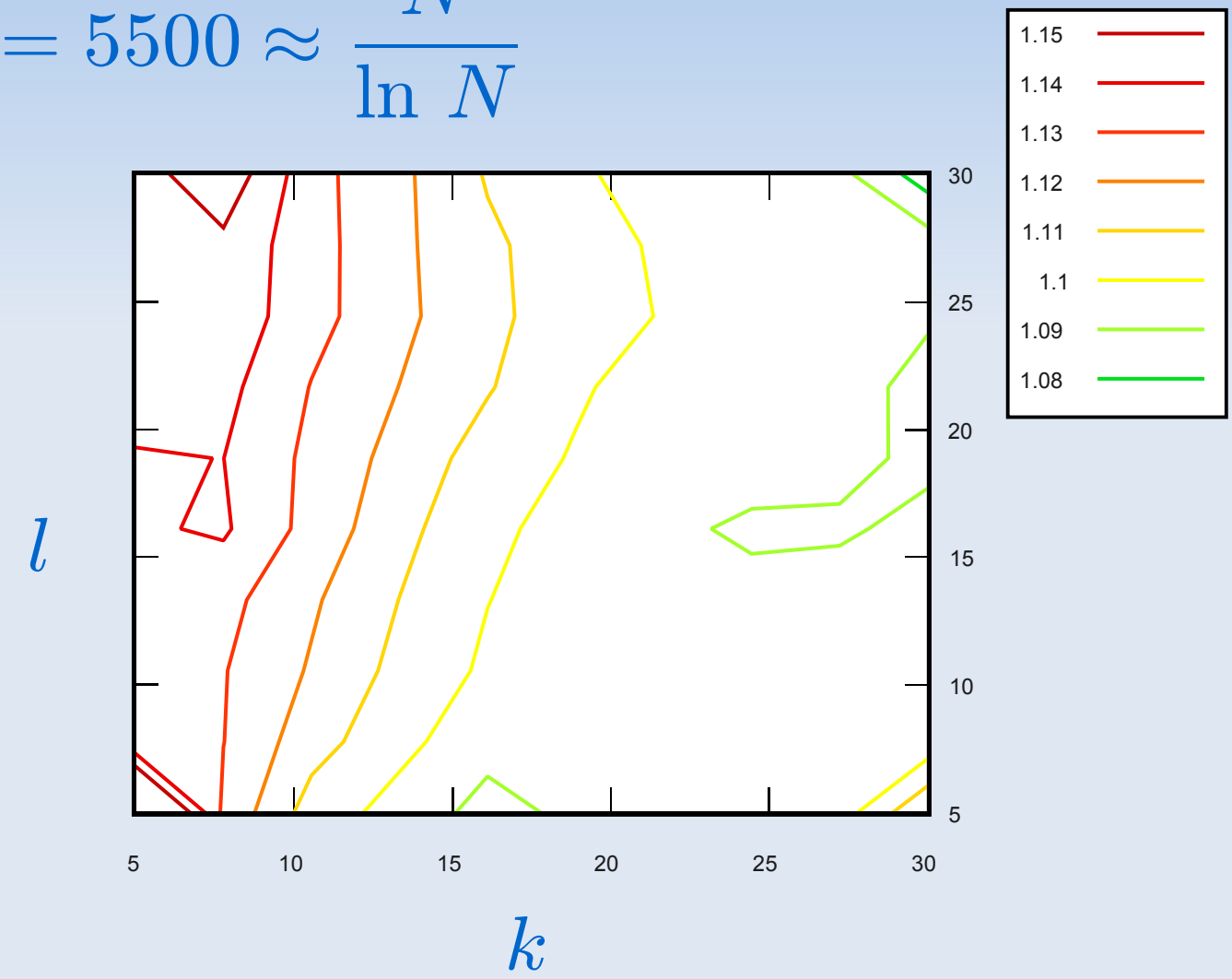
- Let $B = \infty$

- Over hashing can result in too few candidates to return a good approximation.
- Over hashing can cause algorithm to fail.



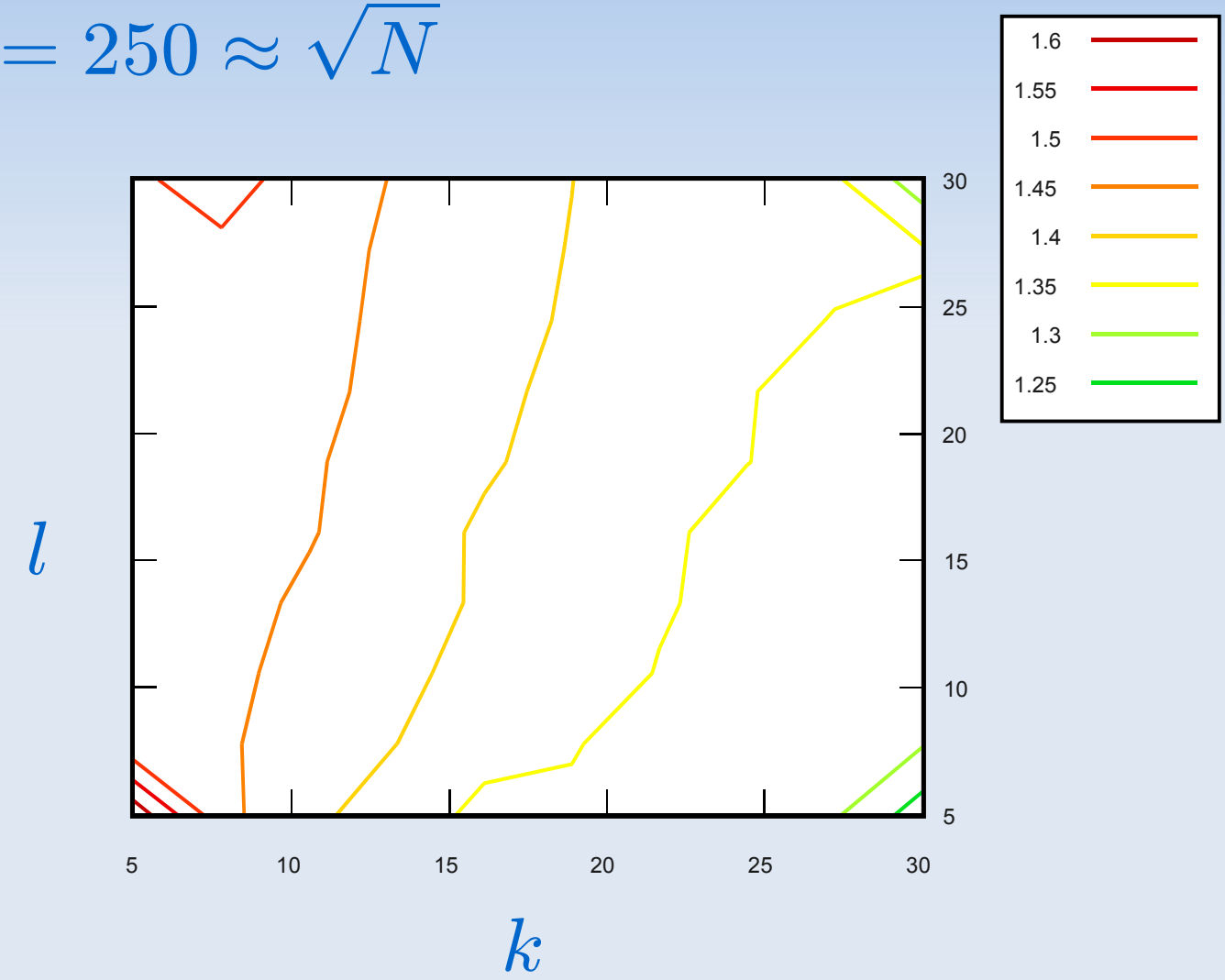
Average Approximation Error

- Let $B = 5500 \approx \dfrac{N}{\ln N}$



Average Approximation Error

- Let $B = 250 \approx \sqrt{N}$



Experiment 2

- Examine the effect of the approximation on the subjective quality of the results.
- Dataset
  - D. Nistér and H. Stewénius. *Scalable recognition with a vocabulary tree*
  - 2550 sets of 4 images represented as document-term matrix of the visual words.

Experiment 2: Issues

- LSH requires a vector representation.
- Not clear how to easily convert a bag of words representation into a vector one.
  - A binary vector where the presence of each word is a bit does not provide a good distance measure.
  - Each image has roughly the same number of different words from any other image.
  - Boostmap?

Conclusions

- Approximate Nearest Neighbors is neccessary for very large high dimensional datasets.
- LSH is a simple approach to aNN.
- LSH requires a vector representation.
- Clear relationship between search length and approximation error.

Tools

- Octave (MATLAB)
- LSH Matlab Toolbox - http://www.cs.brown.edu/~gregory/code/lsh/
- Python
- Gnuplot

References

- 'Fast Pose Estimation with Parameter Senative Hashing' – Shakhnarovich et al.
- 'Similarity Search in High Dimensions via Hashing' – Gionis et al.
- 'Object Recognition Using Locality-Sensitive Hashing of Shape Contexts' - Andrea Frome and Jitendra Malik
- 'Nearest neighbors in high-dimensional spaces', Handbook of Discrete and Computational Geometry – Piotr Indyk
- Algorithms for Nearest Neighbor Search - http://simsearch.yury.name/tutorial.html
- LSH Matlab Toolbox - http://www.cs.brown.edu/~gregory/code/lsh/