# Randomness Efficient Feature Hashing for Sparse Binary Data

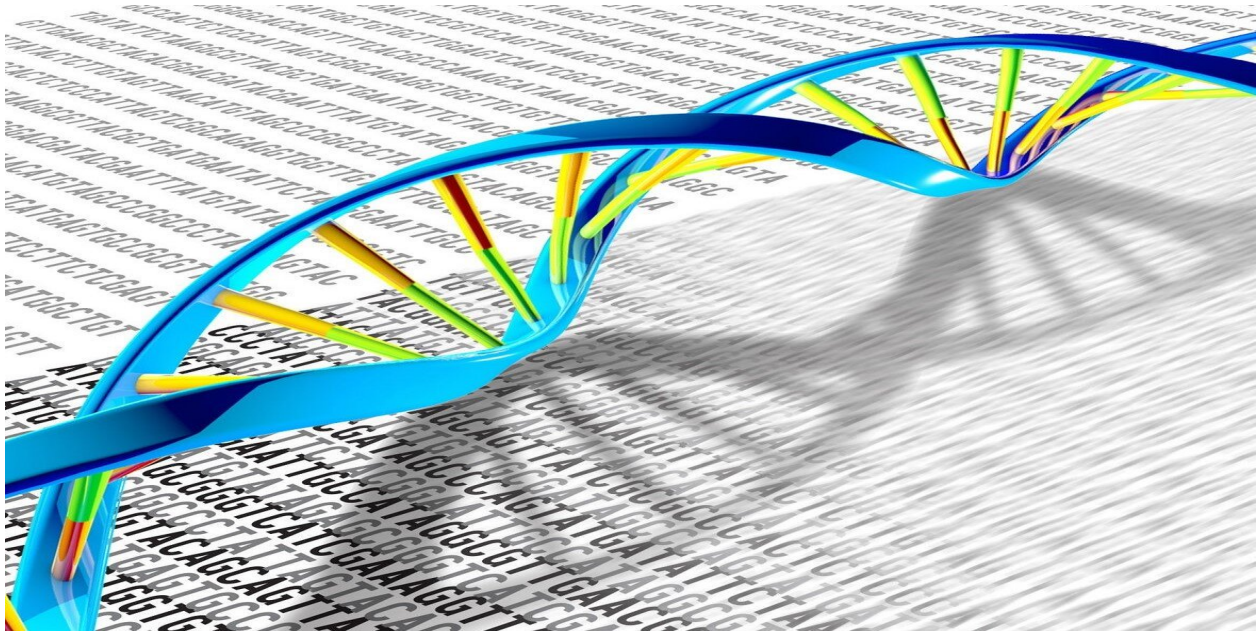
Karthik Revanuru

CRED

November 18, 2020

Joint work with Rameshwar Pratap, Ravi Anirudh and Raghav Kulkarni

# High dimensional data



- Text, Geneome Data
  - bag-of-words representation
- Images
  - most pixels off when converting to black and white
  - Fourier spectrum of most real world images is sparse
- Interaction Matrices
  - user x item matrix in a recommendation system

# Popular similarity/distance measures

- Inner Product
  - Number of common neighbors in social network
- Cosine Similarity
  - Text relevance
- Jaccard Similarity
  - User similarity in recommendation systems
- Euclidean Distance
  - Clustering
- Hamming Distance
  - Error correction

# Similarity preserving dimensionality reductions

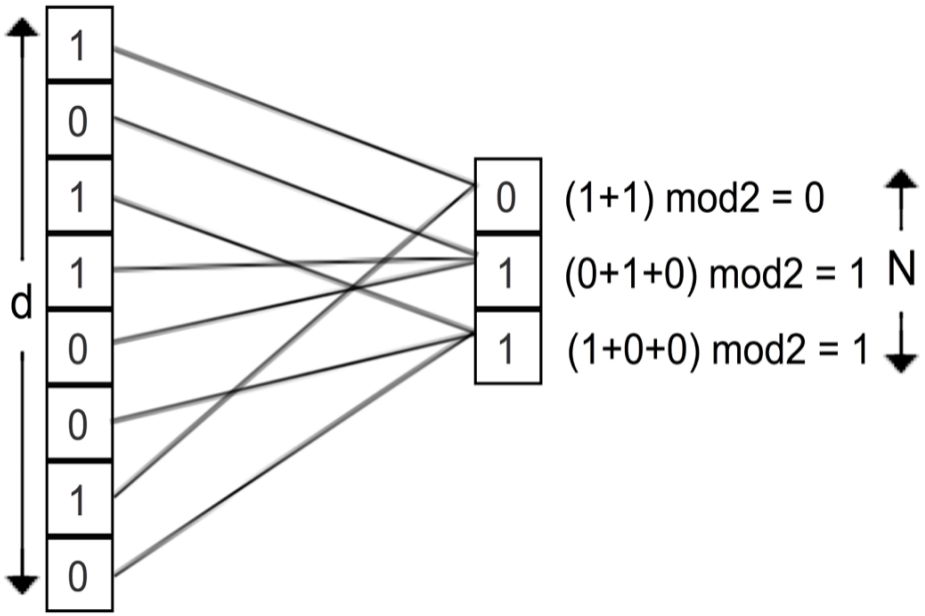
Why similarity preserving dimensionality reductions are useful?

- Typically similarity sub-routines are called multiple times
- Compression $\rightarrow$ Efficient running time
- Compression $\rightarrow$ Efficient storage space
- Also serves as a regularization by pruning unimportant information

# Our focus: Sparse Binary Data

- Text, Genome Sequencing
  - Bag of words
- Images
  - Black and white
- Interaction Matrices
  - User x item interaction

# BCS – Efficient dimensionality reduction for sparse binary data




A simple and efficient dimensionality reduction for sparse binary data

- Binary to binary
- Preserves multiple similarity measures

# BCS Applications

- Document Clustering
- Recommendation systems
- Compressing Social Networks
- Extreme Classification

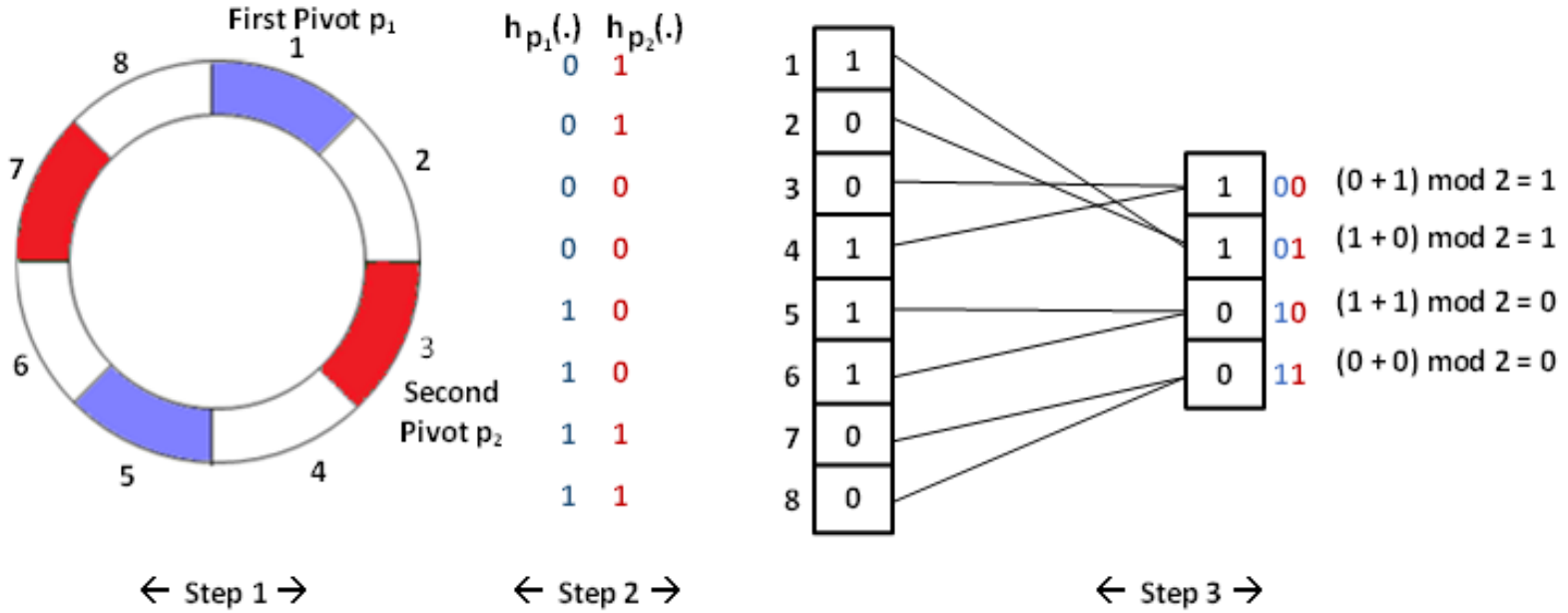
# Limitation of BCS

## Our focus

Randomness efficient compression algorithm

- **Why randomness is important?**
  - Most feature hashing algorithms are randomised
  - Random bit generation is expensive
  - It's a key resource like CPU and Memory
- BCS requires randomness which is linear in data dimension $O(dlogN)$
- Becomes expensive as data dimension $d$ becomes large
- We reduce randomness from linear in $d$ to logarithmic in $d$.

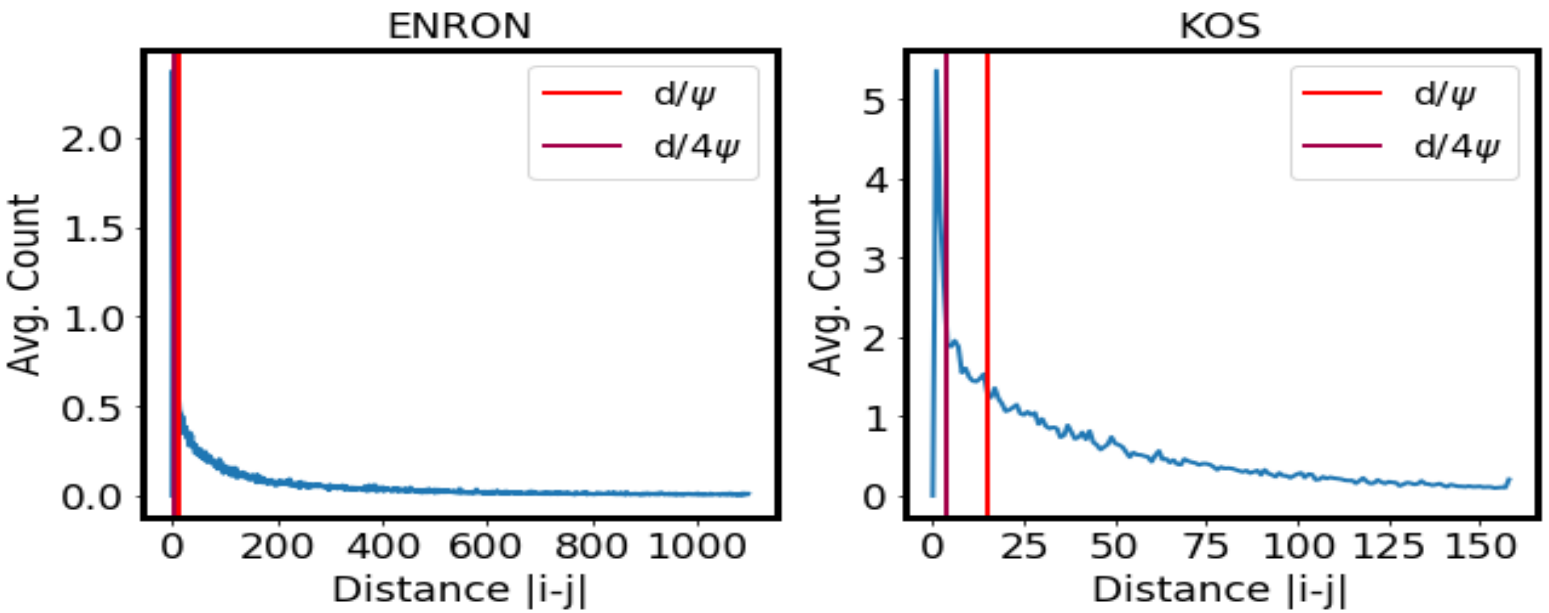
# Our result – PivotHash




- Sample Pivots
- Cyclic distance and generate hash function
- Parity of bits fallen in same bucket
- Generate only $O(\log d)$ pivots to determine the mapping

# Well Spreadness assumption

- **Well spreadness:** distance between non zero indices $= \Omega(d/\psi)$
- Natural assumption on sparse binary dataset
- We give empirical validation.



# Theoretical guarantee of PivotHash

PivotHash works for **well-spread data**

- If $k = O(\psi \log \psi)$, which implies $N < 2k$ then sketch obtained via Pivothash approximates all the similarity measure in the same sketch.
- The number of random bits required by Pivot Hash to satisfy the above guarantee is $O(\psi \log \psi \log d)$.

N - Dimension of the compressed data
$\psi$ - Upper bound on the number of 1's in binary vectors
$k$ - Number of sampled pivots for PivotHash
$d$ - Dimension of dataset

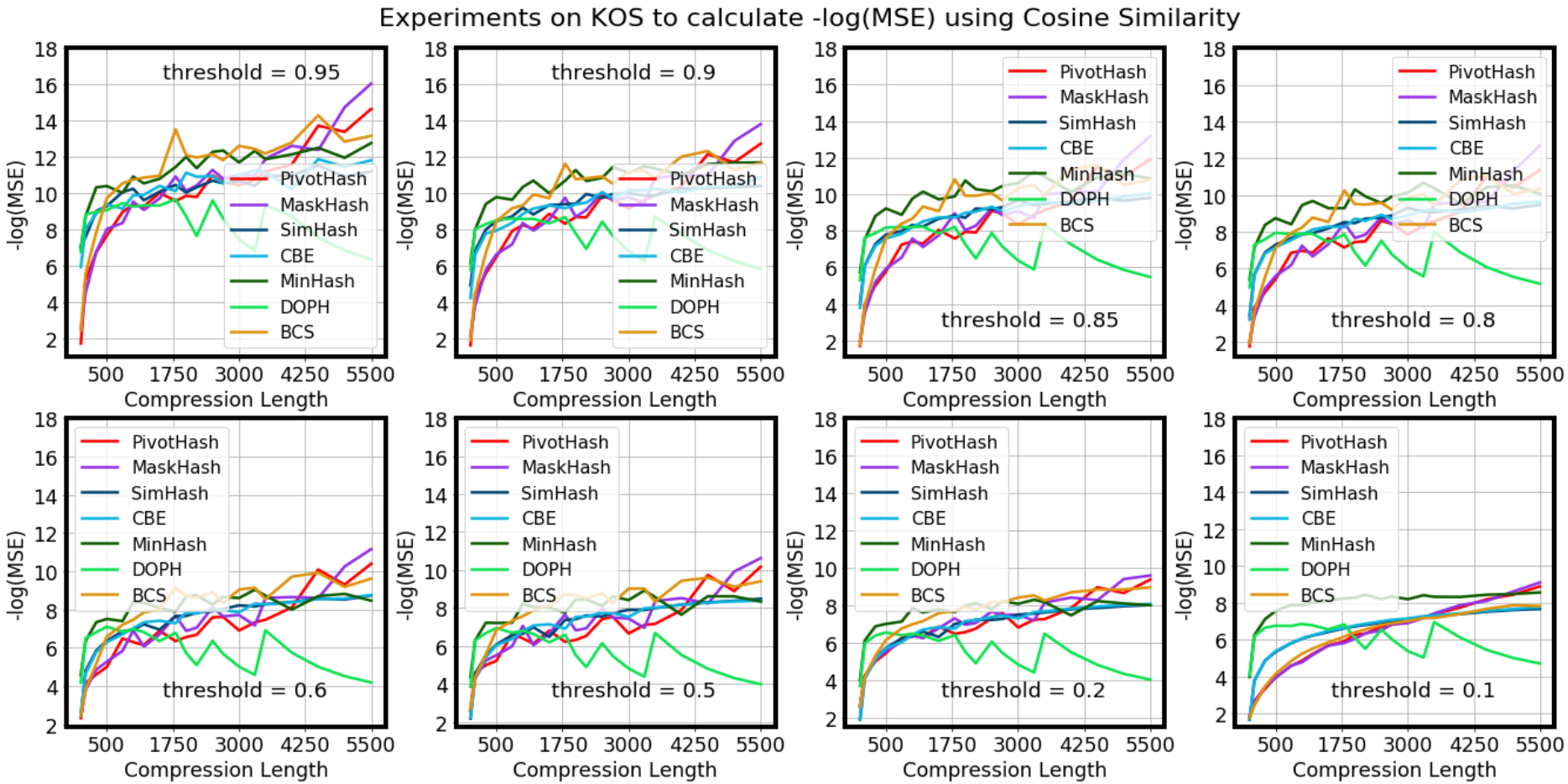
# Theoretical Comparison with Baselines

| Algorithm | No of random bits | Compression time |
|---|---|---|
| **This work** | $O(\psi \log \psi \log d)$ | $O(\psi \log \psi \log d + \psi)$ |
| BCS | $O(d \log \mathrm{N})$ | $O(d \log \mathrm{N} + \psi)$ |
| DOPH | $O(d \log d)$ | $O(d \log d + \psi + \mathrm{N})$ |
| CBE | $O(d)$ | $O(d \log d)$ |
| Simhash | $O(d\mathrm{N})$ | $O((d + \psi)\mathrm{N})$ |
| Minhash | $O((d \log d)\mathrm{N})$ | $O((d \log d + \psi)\mathrm{N})$ |

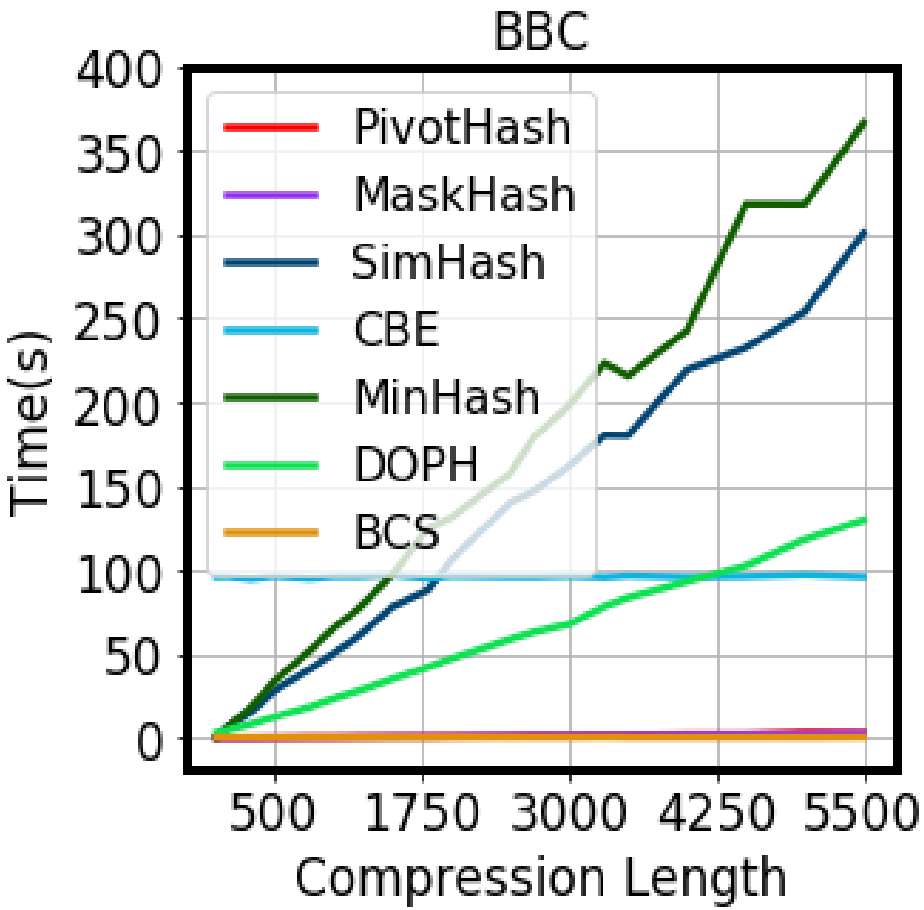
# Experimental Results

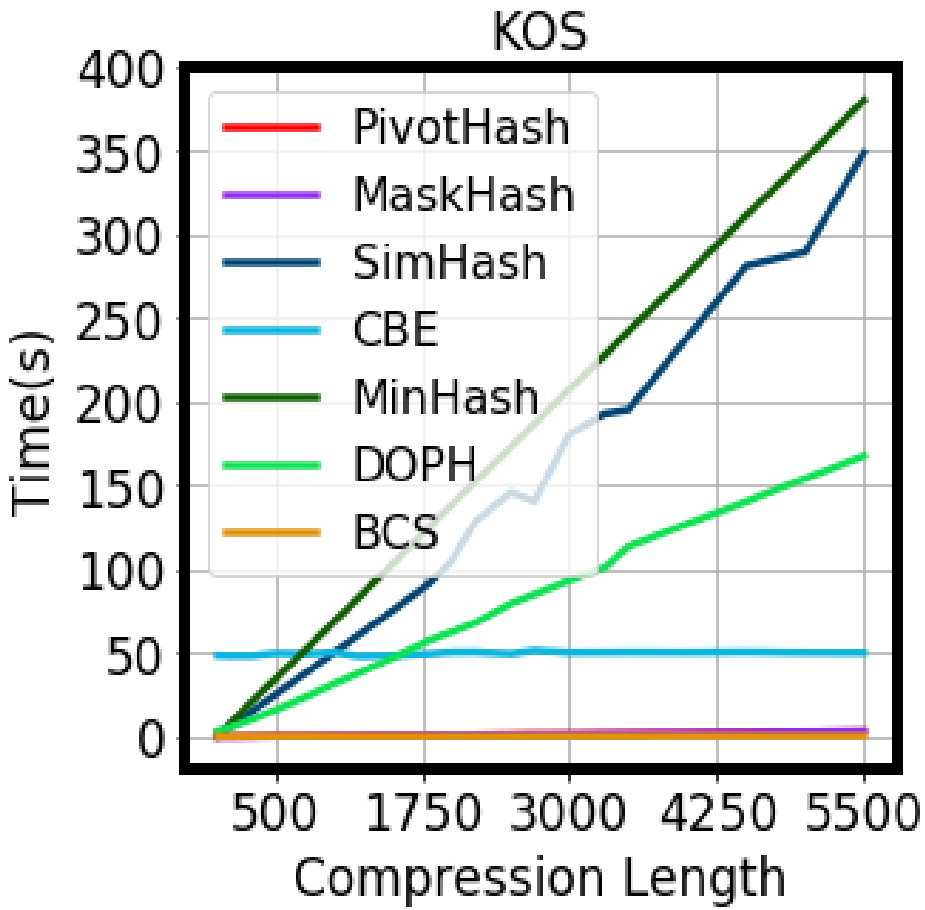
- All pair similarity and bucketize $\in \{0.95, 0.9, 0.85, 0.8, 0.6, 0.5, 0.2, 0.1\}$
- Compress vectors using Pivothash and competing algorithms.
- Compute difference of similarity before and after compression in each bucket.
- Calculate mean and take negative logarithm
- Four public datasets are used. (BBC, KOS, ENRON and NYTimes)

# Experimental Results – MSE Plot



# Experimental Results – Time Plot



# Advantages of PivotHash – Summary

- Randomness efficient (logarithmic in dimension)
- Compact Sketch
- Approximates multiple Similarities
- Can be efficiently adapted in distributed setting
- Almost similar empirical performance to BCS

# *Thank You*