Data-Dependent Hashing for Approximate Nearest Neighbor Searches

Matthew Kangas

U of M - Morris

15th April 2017

Spell Checking

- “Aple”?

Spell Checking

- “Aple”, “Spple”, “Apble”, or “Aplpe”?

Spell Checking

- "Apple" could be misspelled as "Aple", "Spple", "Apble", or "Aplpe"?

Spell Checking

- "Apple" could be misspelled as "Aple", "Spple", "Apble", or "Aplpe"?
- These are all close, but not exact

Spell Checking

- "Apple" could be misspelled as "Aple", "Spple", "Apble", or "Aplpe"?
- These are all close, but not exact
- Many possible dictionary words, very few are plausible

Spell Checking

- "Apple" could be misspelled as "Aple", "Spple", "Apble", or "Aplpe"?
- These are all close, but not exact
- Many possible dictionary words, very few are plausible
- Goal is to find the nearest neighbor to the misspelled word

- Searching for an exact match in a dictionary doesn't work, since it won't match exactly

- Searching for an exact match in a dictionary doesn't work, since it won't match exactly
- Use data-dependent hashing to structure the dictionary to find close neighbors

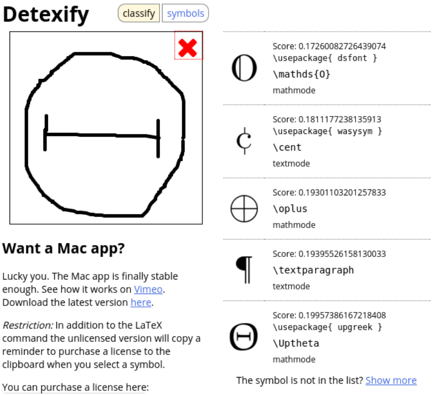
Outline

Outline for section 1

1. Background
   - Nearest Neighbor and Approximate Nearest Neighbor
   - Hashing
   - Locality Sensitive Hashing
2. Data-Dependent Hashing
   - Spherical LSH
   - Preprocessing
   - Querying
3. Conclusion

Nearest Neighbor Applications

- Spell Checking
- Fingerprint Matching
- Character Matching



What is a nearest neighbor search? (*NNS*)

- Used for finding close, but maybe not exact matches

What is a nearest neighbor search? (*NNS*)

- Used for finding close, but maybe not exact matches
- Requires an established set of possible matches

What is a nearest neighbor search? (*NNS*)

- Used for finding close, but maybe not exact matches
- Requires an established set of possible matches
  - Dictionary or Collection of faces

What is a nearest neighbor search? (*NNS*)

- Used for finding close, but maybe not exact matches
- Requires an established set of possible matches
  - Dictionary or Collection of faces
- Takes a query item, and finds the closes matching item in the set

What is a nearest neighbor search? (*NNS*)

- Used for finding close, but maybe not exact matches
- Requires an established set of possible matches
  - Dictionary or Collection of faces
- Takes a query item, and finds the closes matching item in the set
- This is very computationally expensive on large sets

Approximate Nearest Neighbor (*ANN*)

- Variant of the nearest neighbor search

Approximate Nearest Neighbor (*ANN*)

- Variant of the nearest neighbor search
- Returns an item that is within set bounds instead of the absolute closest

Approximate Nearest Neighbor (*ANN*)

- Variant of the nearest neighbor search
- Returns an item that is within set bounds instead of the absolute closest
- Implementations are considerably faster than NNS implementations

Approximate Nearest Neighbor (*ANN*)

- Variant of the nearest neighbor search
- Returns an item that is within set bounds instead of the absolute closest
- Implementations are considerably faster than NNS implementations
- Data-Dependent Hashing (*DDH*) is an improvement on Locality Sensitive Hashing (*LSH*), two methods of solving ANN

Hashing

- Commonly used in both organizing data and verifying equality

Hashing

- Commonly used in both organizing data and verifying equality
- If two files return the same result given the same hash function, they are most likely the same

Hashing

- Commonly used in both organizing data and verifying equality
- If two files return the same result given the same hash function, they are most likely the same
- However if two distinct files are hashed and return the same result, this is called a *collision*

Hashing

- Commonly used in both organizing data and verifying equality
- If two files return the same result given the same hash function, they are most likely the same
- However if two distinct files are hashed and return the same result, this is called a *collision*
- Hash("Apple") = 9f6290f4436e5a2351f12e03b6433c3c

Hashing

- Commonly used in both organizing data and verifying equality
- If two files return the same result given the same hash function, they are most likely the same
- However if two distinct files are hashed and return the same result, this is called a *collision*
- Hash("Apple") = 9f6290f4436e5a2351f12e03b6433c3c
- Hash("Aple") = 674dc064ecd744c1d85d2b471cca818b

Hash Tables

Table 1: Possible Words Hash Table

Key	Value
9f6290f4436e5a2351f12e03b6433c3c	Apple
674dc064ecd744c1d85d2b471cca818b	Apble
c935d187f0b998ef720390f85014ed1e	Dog
6d5c6fcfde82eb131e35fb1cf1cd8143	Cog
b81a30c12698563b79179ec37d43629b	Approximate
e498749f3c42246d50b15c81c101d988	Application

Locality Sensitive Hashing

- Family of hash functions defined on a data space

Locality Sensitive Hashing

- Family of hash functions defined on a data space
- Build to have specific properties distinct from conventional hashing

Locality Sensitive Hashing

- Family of hash functions defined on a data space
- Build to have specific properties distinct from conventional hashing
- Similar inputs ideally collide

Locality Sensitive Hashing

- Family of hash functions defined on a data space
- Build to have specific properties distinct from conventional hashing
- Similar inputs ideally collide
- Dissimilar inputs ideally do not collide

Locality Sensitive Hashing

- Family of hash functions defined on a data space
- Build to have specific properties distinct from conventional hashing
- Similar inputs ideally collide
- Dissimilar inputs ideally do not collide
- *Apple* and *Applw* have a higher chance of collision in their outputs than *Apple* and *Brown*

Locality Sensitive Hashing

- Family of hash functions defined on a data space
- Build to have specific properties distinct from conventional hashing
- Similar inputs ideally collide
- Dissimilar inputs ideally do not collide
- *Apple* and *Applw* have a higher chance of collision in their outputs than *Apple* and *Brown*
- Allows for data to be loosely categorized based on hash results

Locality Sensitive Hashing Continued

- Set of information is preprocessed into a data structure using set of hashes

Locality Sensitive Hashing Continued

- Set of information is preprocessed into a data structure using set of hashes
- Stored in hash tables each tied with a different hash function

Locality Sensitive Hashing Continued

- Set of information is preprocessed into a data structure using set of hashes
- Stored in hash tables each tied with a different hash function
- To query, item is passed through each hash, items are retrieved from tables based on hash results

Locality Sensitive Hashing Continued

- Set of information is preprocessed into a data structure using set of hashes
- Stored in hash tables each tied with a different hash function
- To query, item is passed through each hash, items are retrieved from tables based on hash results
- This reduces the number of items that need to be compared directly

Preprocessing

Table 2: LSH Hash Tables

Hash #1	
Key	**Value**
AP	Apple
AB	Able
PL	Play
TH	This
AI	Airplane
FL	Flight
BA	Banana
GR	Green
BR	Brown

Hash #2	
Key	Value
A_P	Apple
A_L	Able
P_A	Play
T_I	This
A_R	Airplane
F_I	Flight
B_N	Banana
G_E	Green
B_O	Brown

Hash #3	
Key	Value
P_L	Apple
B_E	Able
L_Y	Play
H_S	This
I_L	Airplane
L_G	Flight
A_A	Banana
R_E	Green
R_W	Brown

Querying

Table 3: LSH Hash Tables

	Hash #1			Hash #2			Hash #3	
	Key	**Value**		**Key**	**Value**		**Key**	**Value**
→	AP	Apple		A_P	Apple	→	_P_L	Apple
	AB	Able		A_L	Able		_B_E	Able
	PL	Play		P_A	Play		_L_Y	Play
	TH	This		T_I	This		_H_S	This
	AR	Arbor	→	A_B	Arbor		_R_O	Arbor
	FL	Flight		F_I	Flight		_L_G	Flight
	BA	Banana		B_N	Banana		_A_A	Banana
	GR	Green		G_E	Green		_R_E	Green
	BR	Brown		B_O	Brown		_R_W	Brown

Outline for section 2

Overview

- Faster and More efficient ANN search

Overview

- Faster and More efficient ANN search
- Created by Andoni et al.[1]

Spherical LSH

- Version of a locality sensitive hash

Spherical LSH

- Version of a locality sensitive hash
- Partitions a d-dimensional set

Spherical LSH

- Version of a locality sensitive hash
- Partitions a d-dimensional set
- Data set is projected onto the surface of a d-dimension sphere or radius 1

Spherical LSH

- Version of a locality sensitive hash
- Partitions a d-dimensional set
- Data set is projected onto the surface of a d-dimension sphere or radius 1
- Items are chosen randomly to be the center of a partition

Spherical LSH

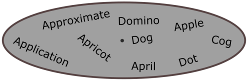
- Version of a locality sensitive hash
- Partitions a d-dimensional set
- Data set is projected onto the surface of a d-dimension sphere or radius 1
- Items are chosen randomly to be the center of a partition
- When complete every partition will be a group of relatively close items

Building the data structure

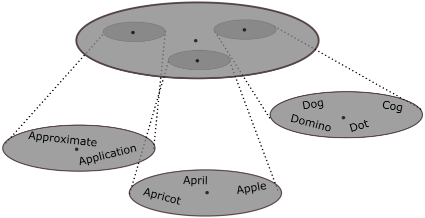
Building the data structure

- Data set is recursively partitioned with spherical hash
  - Each partition's contents are partitioned if the radius of the partition is over a set size

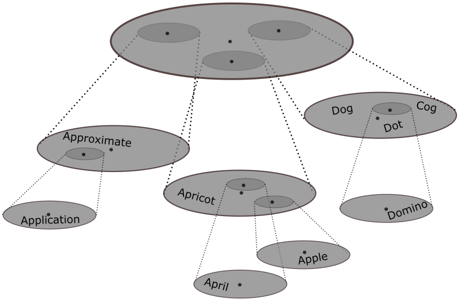
Building the Data Structure



Building the Data Structure



Building the Data Structure



Building the Data Structure

- Data set must be at most dimension $log(n) \cdot log(log(n))$

Building the Data Structure

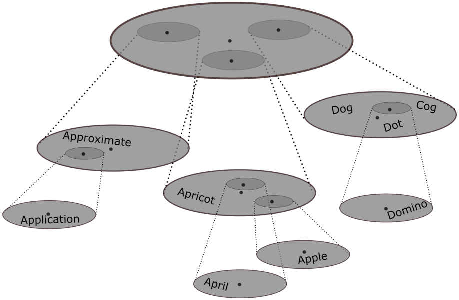
- Data set must be at most dimension $log(n) \cdot log(log(n))$
- Methods exist to insure this

Building the Data Structure

- Data set must be at most dimension $log(n) \cdot log(log(n))$
- Methods exist to insure this
- Most cases have minimal distortion

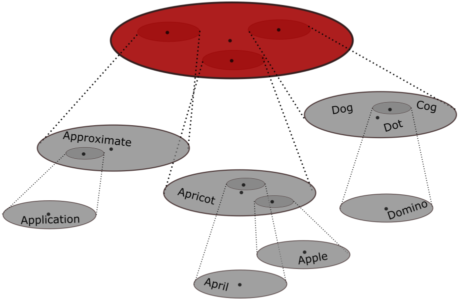
Querying the data structure

- "Apble" is the misspelled word being queried



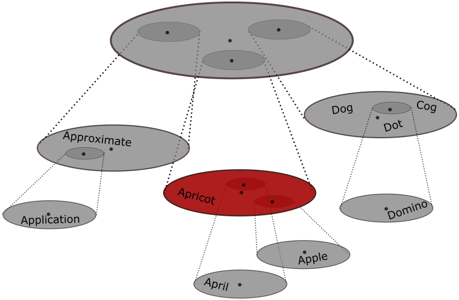
Querying the data structure

- Our word is not close to the center, so the recursive path is taken



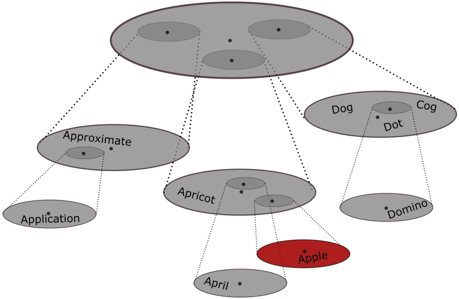
Querying the data structure

- "Apble" is not found in the subpartition, so the query continues



Querying the data structure

- Here the point is found to be close enough to the center of the partition to be returned



Outline for section 3

Preprocessing Time

Data-dependent Hashing

- $O(n^{1+o_c(1)})$
- $n =$ number of items in set
- $o_c(1)$ small constant that approaches 0

Locality Sensitive Hashing

- $O(kt \cdot n^{1+P})$
- $n =$ number of items in set
- $P =$ constant less than 1
- $k =$ complexity of the hash function
- $t =$ time to run each hash function

Search Efficiency

Data-dependent Hashing

- $O(n^{o_c(1)})$
- $n =$ number of items in set
- $o_c(1)$ small constant that approaches 0

Locality Sensitive Hashing

- $O(n^P(kt + d))$
- $n =$ number of items in set
- $P =$ constant less than 1
- $k =$ complexity of the hash function
- $t =$ time to run each hash function
- $d =$ data set dimension

Space Requirements

Data-dependent Hashing

- $O(n^{1+o_c(1)})$
- $n =$ number of items in set
- $o_c(1)$ small constant that approaches 0

Locality Sensitive Hashing

- $O(n^{1+P})$
- $n =$ number of items in set
- $P =$ constant less than 1

References

[1] ANDONI, A., AND RAZENSHTEYN, I.
Optimal data-dependent hashing for approximate near neighbors.
In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2015), STOC '15, ACM, pp. 793–801.

[2] INDYK, P., AND MOTWANI, R.
Approximate nearest neighbors: Towards removing the curse of dimensionality.
In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1998), STOC '98, ACM, pp. 604–613.

[3] WIKIPEDIA.
Locality-sensitive hashing — Wikipedia, the free encyclopedia, 2017.
[Online; accessed 29-March-2017].

[4] WIKIPEDIA.
Nearest neighbor search — Wikipedia, the free encyclopedia, 2017.
[Online; accessed 29-March-2017].

Questions?