Ziqi Tan U88387934

Boston University
CS 562- Advanced Database Systems

Take Home Final Exam – Spring 2020
Return by: 5:00PM EST, Thursday, May 7, 2020 on Gradescope

Instructions

- This exam is OPEN books and notes.
- You have **48 hours** to complete it. There are 4 Problems.
- Make sure that you answer all the questions.
- Use the Gradescope and return your answer by 5:00PM EST, Thursday, May 7, 2020.

Good Luck!

Problem 1

1) Dynamic Time Warping (DTW) is a technique to find an optimal alignment between two time series and is also used to define the similarity (or distance) between time series. Subsequence matching is used to find a subsequence that best matches a query sequence: Given a large time series $X=[x_1, x_2, \ldots, x_n]$ and a query $Q[q_1, q_2, \ldots, q_m]$, we need to find a subsequence $X[x_i,\ldots,x_j]$ from X that has the smallest DTW distance to Q. Provide an algorithm to find the best subsequence matching efficiently. What is the cost of your algorithm?

2) Using the algorithm that you discussed in the previous question, compute the subsequence similarity between the time series X=[1,3,3,2,1,3,2,4] and Q=[3,1,1] and report the best subsequence from X that has the smallest DTW distance to Q. (If there are more than one, you just need to report one).

3) Give an approach to index a large set of strings for similarity queries assuming the similarity function is the Edit distance. Namely, how we can organize the set S of strings so when we have a new query string Q, we can find the string in S that is the most similar to Q under the Edit distance.

Problem

1.1 It is similar to the original DTW.

Only two differences:

① When we initialize the first row, we don't accumulate the distance.

② When we find a path backward, we start from the minimum distance cost in the last row of the cost matrix.

Algorithm:

$X = [x_1, x_2, \dots, x_N]$ $Q = [q_1, q_2, \dots, q_n]$

Cost matrix $C(M, N)$ is a M by N matrix.

$$C(1, n) = |x_n - q_1| \quad (L_1 \text{ distance}, n = 1, 2, \dots, N)$$

$$C(m, 1) = \sum_{k=1}^{m} |x_1 - q_m| \quad (m = 1, 2, \dots, M)$$

for $C(m, n)$ where $m > 1$ and $n > 1$:

$$C(m, n) = |x_n - q_m| + \min\{ C(m-1, n), C(m, n-1), C(m-1, n-1) \}$$

$$j^* = \arg\min_j C(M, j) \quad \text{s.t. } j = 1, 2, \dots, N$$

We walk on the cost matrix backward from $C(M, j^*)$. The next coordinate should be the $\min(C(M-1, n), C(M, n-1), C(M-1, n-1))$. Then we update m and n and walk backward until $m = 0$. Now the index n is what we want. Now Let n be i^*.

Return (i^*, j^*).

$X[i^*, j^*]$ is the subsequence we want.

Therefore, the time complexity is $O(MN)$ where M is the

1.2

	X	1	3	3	2	1	3	2	4
Q									
3		2	0	0	1	2	0	1	1
1		2	2	2	1	1	2	1	4
1		2	4	4	2	1	3	2	4

Report sequence (3, 2, 1).

1.3 Use M-tree to organize the set S of string. The metric distance adopts the Edit Distance.

Randomly choose strings as the root node and then build the M-tree.

Most similar string query Q:

perform KNN search (best first algorithm) on the M-tree like what we do in the R-tree. In this case, $k=1$ to find the most similar string.

Problem 2

1) FastMap is a dimensionality reduction technique that maps points from a high dimensional Euclidian space into a lower dimensional space. What is the problem of the FastMap when the space is not Euclidean (for example when the space is a non-vector space)? Explain what can be the issue in this case. Also, propose an approach to solve the problem.

2) Consider the following distance matrix (pairwise distances) between 5 objects. Apply FastMap on these objects to get the projection to 1-d space (one dimension). Explain every step.

	O1	O2	O3	O4	O5
O1	0	2	1	3	4
O2	2	0	1	3	3
O3	1	1	0	3	3
O4	3	3	3	0	1
O5	4	3	3	1	0

3) Explain which property of the *metric distance* we use in the search algorithm of the **M-tree** and how we use it.

Problem 3

1) Explain how SVD is used to index documents and retrieve the most similar documents to user queries. What is the difference (if there is any) between LSI and SVD ?

2) In typical large document databases, the number of unique (different) words (terms) that appear in all the documents, which is called dictionary, can be very large. What is the challenge of using SVD in that case?
Can you propose a technique that combines another dimensionality reduction technique with SVD in order to address this problem? What are the advantages and disadvantages of your approach? Explain.

For a specific example, assume that you have a database of 10 million documents and a dictionary of 50000 words. How you will apply this technique to this example.

3) Assume a dataset of 4 documents and 6 terms with the following term-doc matrix.

D =

```
1     2     0     0
1     1     0     1
3     4     0     0
1     1     1     0
0     1     2     3
0     0     2     3
```

Problem 2

(1) What is the problem?

Notice one of the steps of the fastmap algorithm. We need to use cosine law to project the high-dimension objects to a lower dimension. Cosine law is the law in Euclidean space. If we step out of this space, we cannot use cosine law.

How to solve?

Reference: Learning Euclidean Embeddings for Indexing and classification

Vassilis Athitsos, Joni Alon, Stan Sclaroff, George Kollios

Method: Euclidean embeddings provide an alternative for indexing non-Euclidean spaces. Using embeddings, we associate each object with a Euclidean vector, so that distances between the mappings of those objects.

Eg: Bourgain embedding, FastMap

Problem 2 (2)

choose two pivot object:

Randomly pick object b.

Heuristic epochs:

find the object c that is most farthest from b

find the object a that is most farthest from c

if b is c:

break

return (object 1, object 5) as the pivot

project objects on line (object 1, object 5) by cosine law:

dist (object 1, object 5) = 4

cosine law: $\frac{2^2+4^2-3^2}{2\times 4} = \frac{11}{8}$

$$\frac{1^2+4^2-3^2}{2\times 4} = 1$$

$$\frac{3^2+4^2-1^2}{8} = 3$$

projection: $[0, \frac{11}{8}, 1, 3, 4]$

Problem 2

(3) We use triangle inequality of the metric distance in the search algorithm of the M-tree. We use it to save unnecessary distance compulations when performing KNN search with a priority queue PR.

We dynamically set up a lower bound distance:

$$d_{min}(T(O_r)) = \max\{d(O_r, Q) - r(O_r), 0\}$$

No object in $T(O_r)$ can have a distance from Q less than $d(Q_r, Q) - r(O_r)$. Thus, we are able to use the bound to prune.

with SVD decomposition, D = U *S* V_T:

U =

-0.32	0.19	0.06	0.54	0.08	0.74
-0.26	-0.02	0.53	-0.50	0.61	0.13
-0.73	0.45	0.05	-0.11	-0.36	-0.32
-0.24	0.03	-0.83	-0.24	0.40	0.09
-0.38	-0.58	0.03	0.50	0.28	-0.42
-0.26	-0.64	-0.03	-0.36	-0.48	0.37

6 x 2

S =

2 x 2

6.07	0	0	0	
0	4.96	0	0	
0	0	1.0	0	
0	0	0	0.67	
0	0	0	0	
0	0	0	0	

V =

2 x 4

-0.50	0.31	-0.08	-0.80
-0.74	0.32	0.07	0.58
-0.25	-0.48	-0.83	0.05
-0.36	-0.74	0.54	-0.11

and 1/6.07 = 0.16, 1 / 4.96 = 0.20

i) Given the above SVD decomposition, compute the mapping of the 5 documents to a 2-dimensional LSI space.

ii) Given the following query: Q = [1 1 0 0 0 0]$_T$, find the best 2 matches between the query and the documents using the LSI approach.

Problem 3

(1) Difference between SVD and LSI:

The process of LSI:

① Construct a term-document matrix.

② Perform SVD on the matrix.

③ Use the S, V, D to identify the concepts contained in the text.

④ LSI may perform dimensional reduction.

SVD is one of the steps of LSI.

How SVD is used?

① Build the term-doc matrix A.

$A = USV^T$

S holds the singular value on its diagonal.

V holds the eigenvectors.

U holds the left eigenvectors.

② We can perform dimensionality reduction by taking first k eigenvectors and eigenvectors.

$U \leftarrow U_k$, $S \leftarrow S_k$, $V^T \leftarrow V_k^T$

③ V_T^k holds information features of N documents.

④ Map the query to a feature space.

$$q = q^T U_k S_k^{-1}$$

⑤ Use cosine similarity on q and V_k^T. Find the similar documents in the result.

Problem 3 (2)

The challenge is that the matrix may be too large to perform SVD.

We can use Random Projection to reduce the dimensionality then apply SVD to reduce dimensionality farther.

The original matrix may be a sparse matrix because many words do not appear very often. Thus, random projection can reduce the dimensionality to save great amount computation.

Example: 50000 words and 10 million documents.

Select k random n-dimensional vectors where $k = \varepsilon^{-2} \ln M$.

Project the original points into the k vectors.

The result k-dimensional space approximately preserves the distances with high probability

Problem 3

(i) $k = 2$

$$U_k = \begin{bmatrix} -0.32 & 0.19 \\ -0.26 & -0.02 \\ -0.73 & 0.45 \\ -0.24 & 0.03 \\ -0.38 & -0.58 \\ -0.26 & -0.64 \end{bmatrix} \qquad S_k = \begin{bmatrix} 6.07 & 0 \\ 0 & 4.96 \end{bmatrix}$$

$$V_k^T = \begin{bmatrix} -0.50 & -0.74 & -0.25 & -0.36 \\ 0.31 & 0.32 & -0.48 & -0.74 \end{bmatrix}$$

$$U_k S_k V_k^T = \begin{bmatrix} 1.297 & 1.783 & 0.035 & 0.006 \\ 0.774 & 1.157 & 0.464 & 0.670 \\ 2.944 & 4.040 & 0.047 & -0.037 \\ 0.799 & 1.159 & 0.310 & 0.437 \\ 0.275 & 0.800 & 2.014 & 3.019 \\ -0.199 & 0.145 & 1.984 & 2.991 \end{bmatrix}$$

(ii) $q^T = [1\ 1\ 0\ 0\ 0\ 0]$

$$\underset{1\times 6}{q} = \underset{1\times 6}{q^T} \underset{6\times 2}{U_k} \underset{2\times 2}{S_k^{-1}} = [-0.0928 \quad 0.034]$$

Use cosine similarity on V_k^T and q.

Result: doc 1 97.9% doc 2 99.8% top 2 match

doc 3 12.7% doc 4 10.1%

Problem 4

1) What are the differences and what the similarities between Pig and Hive.

2) Provide a Map Reduce algorithm to compute the product of a matrix M [m x n] and a matrix N [n x s]. You need to provide the map and reduce functions.

   The product of two matrices P = M * N is defined as:

$$P[i,j] = \sum_{k=1}^{n} M[i,k] * N[k,j]$$

   You can make your own assumptions about the storage and structure of the matrices M and N (for example, if they are dense or sparse, etc).

   Is your algorithm a minimal Map Reduce algorithm? If yes, show why it is. If not, discuss if it is possible to get minimal Map Reduce algorithm for this problem.

3) Spark uses *lazy evaluation* to compute transformations to RDD's. That is, it does not apply transformations to RDDs until it is required to do so. What is the benefit for Spark to do that? Explain.

Problem 4

(1) Similarities:

Both Hive and Pig are data warehouse built on the top of Apache Hadoop.

They both use a SQL-Like Language and translate the SQL-Like queries into MapReduce jobs.

Differences:

	Hive	Pig
Language	HQL	Pig-Latin
Developed by	Facebook	Yahoo
Data	Structured	Structured / Semi-structured

(2) Assume

$$AB = C$$

where A is a $m \times n$ matrix and B is a $n \times m$ matrix. C is a $m \times m$ matrix.

Idea: parallize the computation of each element in matrix C.

Let M be the number of rows of A or the number of columns of B.

Data preprocess: < key, value >

key: (matrix*, ith row, jth column)

value: the element of matrix* in the ith row and jth column

Emit all elements both in A and B.

eg:

$\langle (A, 1, 1), a_{11} \rangle$	$\langle (B, 1, 1), b_{11} \rangle$
$\langle (A, 1, 2), a_{12} \rangle$	$\langle (B, 1, 2), b_{12} \rangle$
⋮	⋮
$\langle (A, 1, n), a_{1n} \rangle$	$\langle (B, 1, m), b_{1m} \rangle$
$\langle (A, 2, 1), a_{21} \rangle$	$\langle (B, 2, 1), b_{21} \rangle$
$\langle (A, 2, 2), a_{22} \rangle$	$\langle (B, 2, 2), b_{22} \rangle$
⋮	⋮
$\langle (A, m, n), a_{mn} \rangle$	$\langle (B, n, m), b_{nm} \rangle$

Mapper 1 (<(matrix*, i, j), mat_{ij}):

Let M be the number of rows of A or the number of columns of B.

if matrix* is A:

for k in range [1, M]:

emit (<(i, k), (mat_{ij}, j)>)

if matrix* is B:

for k in range [1, M]:

emit (<(k, j), (mat_{ij}, i)>)

Reducer 1 (<(x, y), [(v_1, k), (v_2, k), (v_3, k) ...])

// Compute mat_{ij}

// $C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$ M

// There are 2M key value pair in the list.

// k varies from 1 to M.

// V is either A_{ik} or B_{kj}.

produce <(x, y), C_{xy}> where C_{xy} is the xth row and yth column element in the result matrix

This is a minimal algorithm.

1. Every machine use only $O(n)$ space. In each reducer, it stores $2n$ elements where n is the number of columns of matrix A.

2. Every machine sends and receives at most $O(n)$ words of information. In each reducer, it receives $O(2n)$ words of infomation where n is the number of columns of matrix A.
However, if we consider the overall network traffic, this algorithm will not be a minimal Map Reduce.
Overall transfer : $O(mns/t)$ where m is the number of rows of matrix A and t is the number of machines.
We are not able to make it minimal.

3. Constant round : 1 round is enough to get every element of the result matrix.

4. Every machine performs only $O(T_{seq}/t)$ amount of computation. In this case, we parallize the computation of each element of the result matrix.

Example: reference : Emory University Course 554

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \times \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Mapper input	Mapper output
$<(A,1,1),1> \rightarrow$	$((1,1),(1,①)),((1,2),(1,①))$
$<(A,1,2),2> \rightarrow$	$((1,1),(2,②)),((1,2),(2,②))$
$<(A,1,3),3> \rightarrow$	$((1,1),(3,③)),((1,2),(3,③))$
⋮	⋮
$<(B,1,1),6> \rightarrow$	$((1,1),(6,①)),((2,1),(6,①))$
$<(B,2,1),5> \rightarrow$	$((1,1),(5,②)),((2,1),(5,②))$
$<(B,3,1),4> \rightarrow$	$((1,1),(4,③)),((2,1),(4,③))$
⋮	⋮

Reduce 1 : (1,1)

$1 \times 6 + 2 \times 5 + 3 \times 4 = C_{11}$

① ① ② ② ③ ③

(3)

Benefits:

1. Increase manageability.
2. Save computation and reduce complexities.
3. Provide optimization by reducing the number of queries.