Topic Ranger : a tool for topic exploration and analysis of spatio-temporal documents

Lu, You

2019

Lu, Y. (2019). Topic Ranger : a tool for topic exploration and analysis of spatio-temporal documents. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/106850>

<https://doi.org/10.32657/10220/49683>

NANYANG TECHNOLOGICAL UNIVERSITY

**Topic Ranger: A Tool for Topic Exploration
and Analysis of Spatio-Temporal
Documents**

A Master Thesis

Submitted to the School of Computer Science and Engineering
of Nanyang Technological University

by

Lu You

In Partial Fulfillment for the Degree of Master of Engineering

Supervisor: Cong Gao

January 2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

January 23 2019

.....
Date

吕尤

.....
Lu You

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

January 23 2019

.....
Date

.....
Professor Cong Gao

Authorship Attribution Statement

This thesis contains material from 1 paper published in the following peer-reviewed journal where I was the first and/or corresponding author.

Chapter 3 is published as W.-J. Hsu, Y. Lu, and Z. Q. Lee, "Accelerating topic exploration of multidimensional documents," in 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), May 2017, pp. 1520-1527.

The contributions of the co-authors are as follows:

- Prof. Hsu provided the initial project direction and prepared the manuscript drafts.
- I edited part of the manuscript drafts. The manuscript was revised by Prof. Hsu and Dr. Lee.
- I analyzed the time complexity in theory with Prof. Hsu and Dr. Lee.

January 14 2019

.....
Date

吕尤

.....
Lu You

Acknowledgement

I would like to thank my supervisor Professor Cong Gao for his kind and patient guidance for this research work. He is always supporting me whenever I meet problems in my research work. Without his help, the research and the thesis could not be successfully completed.

I would also like to thank Professor Hsu Wen Jing for his helpful suggestions for the research. He always helps me to follow the right direction through the research work and thesis writing with patience and kindness.

I am grateful to Dr. Zhao Kaiqi for providing the help to my implementation and experiment. I finished the experiment and result analysis thanks to his kind help and useful suggestions.

Finally, I am also grateful to the researchers and experts in data indexing structure and topic modeling study. I completed the research work based on their study and research.

Abstract

With the wide-spread usage of social media such as Facebook and Twitter, large amount of data with both spatial and temporal information has become available. Topic modelling has been a useful tool to uncover latent information from such data. This thesis considers a specific type of topic model computational problem called *topic-range queries*, where the topic model of interest is restricted to the data records that fall within a dynamically specified geographic region and time period. To achieve this purpose, one naive approach is to directly apply a range query to retrieve the data items falling within the specified spatio-temporal range, then derive the topic model from the retrieved data by using a known algorithm such as LDA (Latent Dirichlet Allocation). When dealing with large volume of data, however, the two-step naive approach could each incur substantial amount of time. Novel algorithms for expediting the topic-range queries have been designed, including the fast topic combining algorithm FSS (Fast Set Sampling) which indexes the dataset with a tree, and pre-compute the topic model of the subset of data associated with each node of the tree. To answer a topic-range query, the tree nodes covered by the range query are identified, and the pre-computed topic models associated with these tree nodes are merged to produce an approximate result. Compared to the naive approach, this approximation of topic model substantially can reduce runtime. In the original design of the FSS algorithm, Cube trees are used as the indexing structure to support spatio-temporal range queries. In the literature, however, Range Trees offer a better worst-case query time guarantee for a range query. This master thesis thus

considers a new combination of Range Trees and FSS (called Topic Ranger) to support the topic-range queries. The thesis presents the design, implementation of several versions of Topic Ranger for trade-offs between execution time and memory space. It also documents the experiments and comparisons of the execution time and the quality of the resulting approximate topic models against that of the original FSS scheme.

Contents

Acknowledgement	i
Abstract	ii
List of Figures	vii
1 Introduction	1
1.1 Background and Motivation	1
1.2 Problem Description	2
1.3 Scope of Work	4
1.4 Synopsis	6
2 Literature Review	7
2.1 Data Indexing	7
2.1.1 KD Tree/ KDB Tree	8
2.1.2 Quadtree/Cube Tree	9
2.1.3 Ball Tree	10
2.1.4 Cover Tree	12
2.1.5 Range Tree	14
2.2 Topic Modeling	22
2.2.1 Definition and Purpose of Topic Modeling	22
2.2.2 Relevant Research of Topic Modeling	24
2.3 Topic Model Evaluation	31

2.3.1	Perplexity	31
2.3.2	Left-to-right Algorithm	32
2.3.3	Distances and Divergence	33
2.4	Fast Set Sampling Algorithm	36
2.5	Summary	39
3	Topic Ranger: Methodology	40
3.1	Structure of Topic Ranger	40
3.2	Algorithms and Improvements	47
3.2.1	Range Tree Construction Algorithm	48
3.2.2	FSS Topic Model Combining Algorithm	52
3.2.3	Parallel Pre-Computing	54
3.2.4	Removal of Tree On Secondary Dimension	55
3.3	Summary	55
4	Experiments and Result Analysis	57
4.1	Experiments	57
4.2	Comparisons of Various Incarnations of Topic Ranger	60
4.2.1	Parameter: Branching Factor	61
4.2.2	Parameter: Leaf Node Size	72
4.2.3	Conclusion of Topic Ranger with Different Parameters	81
4.3	Comparison Between Topic Ranger and FSS with Cube Tree	81
4.3.1	Storage Size of Data	83
4.3.2	Runtime Cost	84
4.3.3	Topic Model Quality	90
4.3.4	Conclusion of Comparison between Topic Ranger and FSS with Cube Tree	96
4.4	Summary	97

5	Summary and Future Work	99
5.1	Summary of the Research	99
5.2	Future Directions	100
	Appendices	102
A	An Example of Tree Construction of Topic Ranger	103
B	Topic-Range Query Set of Chapter 5	115
C	Topic-Range Query Set of Chapter 6	117
D	Figures - Comparison of Query Runtime Cost: CT, RTS and RT	119
E	Figures - Comparison of Query Runtime Cost: CT and RTS	124
F	Figures - Comparison of FSS Runtime Cost	129
G	Figures - Comparison of OLDA Runtime Cost	134
H	Figures - Comparison of Runtime Cost of All Algorithms	139
I	Figures - Comparison of Topic Model Perplexity of All Algorithms	144
J	Figures - Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models	149
K	Figures - Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures	154
	References	159

List of Figures

1.1	An example of topic models of a document (Blei, 2012)	3
1.2	Designed framework structure of the newly designed algorithm "Topic Ranger"	5
2.1	Left: Corresponding binary tree. Right: KD tree. (Couto & Napoli, 1998)	8
2.2	Left: Getting a set of octants with a cube by subdividing recursively. Right: Corresponding tree. (Wikipedia, 2010)	9
2.3	Example of a 2D ball tree for each level [25]	11
2.4	An example of cover tree. The first branch of the cover tree is shown in (a), the second branch is shown in (b), and the third branch is shown in (c). (Kibriya & Frank, 2007)	13
2.5	An example of 1D range tree and querying with 1D tree. (Eastman, 2012)	15
2.6	Bottom: 2-layer range tree for 2D range query set. (Qiao, Gan & Tao, 2016)	16
2.7	An example of pointers between sorted arrays (Suri, 2002)	19
2.8	pLSA graphic topic model. (Blei, Ng & Jordan, 2003)	23
2.9	Topic models	25
2.10	A subtree under the topic "Computer" of a hLDA model. (Chen, Zhu, Lu & Liu, 2018)	29
3.1	Canonical set for a range query (Mount, 2001)	41

3.2	An example of converting Canonical Set of a range query into a heap . . .	42
3.3	An example of 3-branching hierarchical tree for spatio-temporal dataset	45
3.4	An example of 2-branching hierarchical tree with sub range trees as leaf nodes	47
3.5	An example of 2-branching hierarchical tree with sub range trees with 3 types of nodes	49
4.1	Comparison of Topic Ranger range tree file size	62
4.2	Comparison of RT Query Time Cost	65
4.3	Comparison of RTS Query Time Cost	66
4.4	Comparison of RT Topic Model Approximation Runtime Cost	67
4.5	Comparison of RTS Topic Model Approximation Runtime Cost	68
4.6	Comparison of Topic Ranger Topic Model Perplexity	70
4.7	Comparison of the Bhattacharyya Distance from the Topic Ranger Topic Model to the LDA Topic Model	71
4.8	Comparison of Topic Ranger Range Tree File Size	72
4.9	Comparison of RT Query Time Cost	75
4.10	Comparison of RTS Query Time Cost	76
4.11	Comparison of RT Topic Model Approximation Runtime Cost	77
4.12	Comparison of RTS Topic Model Approximation Runtime Cost	78
4.13	Comparison of Topic Ranger Topic Model Perplexity	79
4.14	Comparison of the Bhattacharyya Distance from the Topic Ranger Topic Model to the LDA Topic Model	80
4.15	Comparison of Query Runtime Cost (Query 1): Upper: Comparison of CT, RTS and RT. Lower: Comparison of CT and RTS	84
4.16	Comparison of FSS Runtime Cost	85
4.17	Comparison of OLDA Runtime Cost	87

4.18	Comparison of Runtime Cost of All Algorithms	89
4.19	Comparison of Topic Model Perplexity of All Algorithms	91
4.20	Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models	94
4.21	Comparison of FSS Runtime Cost	95
A.1	An example of 2-branching hierarchical tree with sub range trees as leaf nodes for the toy dataset	113
D.1	Comparison of Query Runtime Cost: CT, RTS and RT - Query 1-4 . . .	120
D.2	Comparison of Query Runtime Cost: CT, RTS and RT - Query 5-8 . . .	121
D.3	Comparison of Query Runtime Cost: CT, RTS and RT - Query 9-12 . . .	122
D.4	Comparison of Query Runtime Cost: CT, RTS and RT - Query 13-15 . .	123
E.1	Comparison of Query Runtime Cost: CT and RTS - Query 1-4	125
E.2	Comparison of Query Runtime Cost: CT and RTS - Query 5-8	126
E.3	Comparison of Query Runtime Cost: CT and RTS - Query 9-12	127
E.4	Comparison of Query Runtime Cost: CT and RTS - Query 13-15	128
F.1	Comparison of FSS Runtime Cost: Query 1-4	130
F.2	Comparison of FSS Runtime Cost: Query 5-8	131
F.3	Comparison of FSS Runtime Cost: Query 9-12	132
F.4	Comparison of FSS Runtime Cost: Query 13-15	133
G.1	Comparison of OLDA Runtime Cost: Query 1-4	135
G.2	Comparison of OLDA Runtime Cost: Query 5-8	136
G.3	Comparison of OLDA Runtime Costs: Query 9-12	137
G.4	Comparison of OLDA Runtime Cost: Query 13-15	138
H.1	Comparison of Runtime Cost of All Algorithms: Query 1-4	140

H.2	Comparison of Runtime Cost of All Algorithms: Query 5-8	141
H.3	Comparison of Runtime Cost of All Algorithms: Query 9-12	142
H.4	Comparison of Runtime Cost of All Algorithms: Query 13-15	143
I.1	Comparison of Topic Model Perplexity of All Algorithms: Query 1-4 . . .	145
I.2	Comparison of Topic Model Perplexity of All Algorithms: Query 5-8 . . .	146
I.3	Comparison of Topic Model Perplexity of All Algorithms: Query 9-12 . .	147
I.4	Comparison of Topic Model Perplexity of All Algorithms: Query 13-15 .	148
J.1	Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 1-4	150
J.2	Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 5-8	151
J.3	Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 9-12	152
J.4	Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 13-15	153
K.1	Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 1-4 .	155
K.2	Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 5-8 .	156
K.3	Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 9-12 .	157
K.4	Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 13-15	158

Chapter 1

Introduction

1.1 Background and Motivation

With the wide spread development of social media, the social media users have generated huge amount of data. In 2018, for instance, 473,400 tweets on average were sent every minute, which accumulates into a huge volume of data [5].

The social media data often contain information about the time and location at which they were generated. With the huge spatio-temporal dataset from social media, data analysis has become an important approach to extract the valuable latent information. Topic modeling is one of the methods of data analysis, which aims to find the main theme of the social media data [6]. This thesis considers a special type of topic model computational problem called topic-range queries, where the topic model of interest is restricted to the data records that fall within a dynamically specified geographic region and time period. Topic-range queries can be applied to track the topics people talked about, mine the patterns of the topic trends, and predict the changes of the topic content, etc. [7, 8, 9]. All these purposes have an important position in social science and data analysis [10, 11, 12].

One brute force method to support topic-range query is to retrieve all the data items within the region and time period, and then compute the topic model based on the queried data items. However, this naive approach incurs high runtime and memory cost

with large datasets. Computing the topic model for a topic-range query with efficiency becomes the goal of many research proposals. An algorithm named Fast Set Sampling algorithm (FSS) has been proposed for topic-range query [15]. The original FSS algorithm indexes the whole corpus with a tree, and pre-compute the topic models every node of the tree. To answer a topic-range query, we retrieve all the nodes covered by the range query, and combine the topic model of smaller nodes to the largest node by re-sampling the data items in the smaller node with reference to the topic model of the largest node. Compared to directly computing topic models, this approach provides an approximate topic model with a lower runtime cost.

In the original design of the FSS algorithm, cube trees are used as the indexing structure to support spatio-temporal range queries. In the literature, however, Range Trees [32, 30, 31, 33] offer a better worst-case query time guarantee for a range query. This master thesis thus considers a new combination of Range Trees and FSS (called Topic Ranger) to support the topic-range queries. The thesis presents the design, implementation of several versions of Topic Ranger for trade-offs between execution time and memory space. It also documents the experiments and comparisons of the execution time and the quality of the resulting approximate topic models against that of the original FSS scheme.

1.2 Problem Description

In this section, we further explain the problem to be solved and relevant terminologies.

- Topic model is a statistical model describing the topics of a document collection [6, 13]. While using topic modeling, we assume that one document contains many different topics at the same time. Each of the topics is actually a probabilistic distribution over words. An example showing the structure of the topic model is presented in Fig. 1.1:

Figure 1.1: An example of topic models of a document (Blei, 2012)

The purpose of topic modeling is to find out the abstract topics in the dataset. By comparing the topic models of different datasets, latent patterns and information may be uncovered for further applications.

- Spatio-temporal Data: This refers to the data records that contain the users' geographic information and the timestamp of posting.
- Spatio-temporal Range Query: For such queries, the geographic ranges of data records will be specified in the form of latitude and longitude values. Meanwhile, the time period of query should also be defined.
- Exact Point Query: A query operation that retrieves all the data records meeting exactly the query input conditions.
- Exact Range Query: A query operation that retrieves all the data records falling within the specified boundaries of a query.

- **Data Indexing Structure Supporting Exact Point Query and Exact Range Query:** Without checking the data items one by one, the data indexing data structure enables returning the query result of an exact point query or exact range query.
- **Topic-Range Query:** To compute the topic model, we can apply range query to the dataset to retrieve the data records, and directly compute topic model using the Latent Dirichlet Allocation algorithm (LDA). However, we want to solve the problem with less runtime cost, and still generate topic models with high accuracy. Therefore, the goal is to get an improved algorithm to generate the topic model with good quality of the range query, and its runtime should be lower than LDA.

1.3 Scope of Work

Fig. 1.2 shows a framework for supporting the topic-range queries. In Fig. 1.2, we firstly cleanse the raw data to get cleansed data. Then we use data indexing structure to store and index data, and we pre-train the topic models for subsets of data. With the topic models, we apply range queries to get the query result, and try to extract the topic model of the query result. Based on the topic model of range query result, we can visualize the topic models or perform further statistical analysis.

Figure 1.2: Designed framework structure of the newly designed algorithm "Topic Ranger"

This thesis presents the design and implementation of the main components of the framework in Fig. 1.2. It also documents the evaluation of the quality of the topic models computed. The correctness of range query will be examined, and the quality of the topic models will be evaluated and compared to the topic model generated by the original implementation of FSS with cube tree.

1.4 Synopsis

The rest of the thesis is organized as follows. In Chapter 2, we review the literature on topic modeling researches, data structures and topic model evaluation metrics. We also review the details of the FSS algorithm. We then present our design and implementation details of Topic Ranger algorithm in Chapter 3. Chapter 4 presents the experiment details. Chapter 5 shows the results and analysis of the experiments using Topic Ranger with different parameters. Chapter 6 compares the experiment results of Topic Ranger and FSS with cube tree. Finally, we present a conclusion together with future directions in Chapter 7.

Chapter 2

Literature Review

This chapter reviews the literature about the existing approaches for topic modeling and range queries. The first section is about data indexing for range query, the second section is about topic modeling, the third section talks about topic model evaluation, and the fourth section is about FSS algorithm, which performs fast topic modeling by pre-training and combining.

For each part of study, a detailed description will be stated, and relevant research will be reviewed. Different approaches and their variants will be compared and discussed. New schemes for data indexing and topic modeling will be created by using the ideas of selected approaches, and the selected evaluation method will be used in the evaluation process.

2.1 Data Indexing

The data indexing algorithms will be discussed and compared in this section. The details of the algorithm will be stated. The aspects compared include time complexity of data indexing, space complexity for storage, point query time complexity and range query time complexity. Due to large amount of existing literature in data indexing, only the data structures that support both point query and range query of multi-dimensional dataset will be reviewed.

2.1.1 KD Tree/ KDB Tree

KD tree is a binary tree used in k -dimensional data indexing [16, 17]. To keep the height balance of the sub trees, a hyperplane is used to divided the data points in the parent node orthogonal space into two sets, each with the same or nearly the same number of data points. An example of 2D KD tree is shown in Fig. 2.1:

Figure 2.1: Left: Corresponding binary tree. Right: KD tree. (Couto & Napoli, 1998)

In Fig. 2.1, each leaf node represents one data item in the dataset. The non-leaf nodes record the position of the splitting hyperplanes. In the KD tree construction, all the data items with the median value are assigned to the one of the two child nodes, and the two child nodes are separated by the median. The dataset is recursively divided into two balanced datasets on each dimension. Besides the static data structure construction algorithm, dynamic insertion algorithms are also designed for KD tree for insertion and deletion.

KDB trees use the same algorithm of construction and insertion operation [17, 18]. The difference between KD tree and KDB tree is that all the leaf nodes of KDB tree are pointers to the data items, while leaf nodes of KD tree are the stored data items. The leaf node pointers are linked together as a linked list, which makes sequential tracing operation and fast range query operation available.

For query operations, both exact point query and exact range query are available with KD tree. The point querying search process is the same as binary tree. However, the time cost of range querying progress depends on the number of cells covered by the range.

2.1.2 Quadtree/Cube Tree

Both Quadtree and cube tree (also named as Octree) are constructed by dividing the range of dataset into rectangles [20]. Quadtree is for 2D dataset and cube tree is for 3D dataset. Therefore, each parent node in Quadtree and cube tree has 4 and 8 child nodes respectively. For the datasets of social media, if the geographic tag is added, then the dataset can be seen as spatio-temporal dataset with three dimensions. As such, cube tree will be more suitable and meaningful than Quadtree, especially in spatio-temporal analysis. An example of cube tree is shown below in Fig. 2.2:

Figure 2.2: Left: Getting a set of octants with a cube by subdividing recursively. Right: Corresponding tree. (Wikipedia, 2010)

As shown in the left-hand side of Fig.2.2, cube tree construction is a decomposition progress. Before constructions, terminating conditions including the minimum number of points contained by a leaf node, the minimum length of cube's edge, and the maximum number of decomposition iterations should be defined. In each iteration, the cube covering the range of dataset will be divided into 8 smaller cubes. This process continues until one of the limitations has been reached.

An exact range query might not be available with cube tree unless this is a full cube tree. The terminating conditions might terminate the splitting of the sub trees when multiple points are still stored together in the lowest level cells. Cube tree is often used in searching for K nearest neighbors [22].

2.1.3 Ball Tree

Ball tree, which is known as a binary version of metric tree, is used in space partitioning for multi-dimensional dataset [23, 24]. Different from KD trees and KDB trees, a ball tree uses hypersphere to divide the data points instead of hyperplane, and the hypersphere balls in the same level are allowed to overlap with each other. The name "Ball tree" comes from the hypersphere used for data partitioning.

For each of the nodes of a ball tree, it defines a hypersphere which covers a subset of dataset. The root node refers to a hypersphere that covers the whole dataset, while each of the leaf nodes refers to a single data point. Each interior node contains two hypersphere balls which split the parent node's dataset into two disjoint subsets. For any data points that belong to the intersection part of two hypersphere balls, they are assigned to either one of the hypersphere ball.

Metric tree, also named as M-tree, is a multi-branching version of ball tree [25]. By using larger branching factor, it has a shorter tree height, which implies that the query time used for a metric tree is shorter than a ball tree for the same dataset. An example of ball tree is shown in Fig. 2.3:

Figure 2.3: Example of a 2D ball tree for each level [25]

From Fig.2.3, for each of the level, the subset of data points contains half of the points of its parent. Therefore, the number of subsets for each level is twice that of the upper level.

To construct a ball tree, an algorithm named k-d construction algorithm is used, which is similar to the construction of KD trees. For each of the iterations, based on

the current dataset to be divided, the dimension for the largest spread c and the central point p are selected along the dimension c . Two subsets L and R are the two child nodes of the current dataset, and they are constructed by adding data points into the subsets. For those points to the left of the point p , they belong to the subset L , and it is similar to the subset R . Finally, the distance from the furthest point to the point p is taken as the diameter of the hypersphere for the subset. The time complexity is $\mathcal{O}(n(\log n)^2)$ for construction, where n denotes the number of data items [23, 24].

For data querying, a query ball is provided, and all the leaf nodes containing single data points that fall inside the query ball will be retrieved as the query result. A recursive search algorithm is applied to search down from the root of the ball tree. For each of the level, the nodes that are not contained inside of the query ball are pruned. Here since the distance between the centers and radius lengths of the hypersphere balls are used, the query process will be very efficient.

Beside querying with pruning, querying with branch and bound is also applicable to ball trees. Such a querying method is often used in a case of counting problem: Given a query point q , a number of m points nearest to q should be queried, and a smallest hypersphere ball B containing all the m points should be generated. The m nearest points are selected through a priority queue of the distance to q . Based on the m data points, the center and the radius of B will be calculated.

2.1.4 Cover Tree

The cover tree is a data structure designed for nearest-neighbor search [26, 27, 24, 28]. It is similar to KD tree, which constructs the tree by covering the data points in hierarchical levels. The cover tree is usually used to solve such a problem: With n data points in a dataset S within a space D , a query point $q \in D$ is given, and a $p \in S$ will be searched as the data point nearest to q .

To solve such a problem, the cover tree divides the dataset hierarchically from the root node to the leaf nodes. For each of the level, an integer scale i is set as the index for each of the tree level in a descending order from the root node. Let C_i denote the set nodes at level i . We have the three properties for the cover tree below:

- Nesting: $C_i \subseteq C_{i-1}$
- Covering: For each of the data point p that belongs to the set of C_{i-1} , there exists exactly one point $q \in C_i$ such that a distance $d(p, q)$ between p and q is less or equal to 2^i .
- Separation: For all the pairs of $p \in C_i$ and $q \in C_i$, the distance $d(p, q)$ in between is greater than 2^i .

Based on the three properties, the cover tree is constructed. Similar to ball trees, a cover tree also bounds the subsets with hypersphere. An example of a cover tree is shown in Fig. 2.4:

Figure 2.4: An example of cover tree. The first branch of the cover tree is shown in (a), the second branch is shown in (b), and the third branch is shown in (c). (Kibriya & Frank, 2007)

From Fig. 2.4, we can find that the root node refers to a hypersphere ball that covers all the data points within the dataset. If the diameter of a parent node's hypersphere ball is d , then its child nodes will have hypersphere balls with diameter $0.5d$ [24, 29]. One child node's hypersphere will share the same center of its parent node's hypersphere, while the other child nodes are using data points outside of constructed $0.5d$ -size hypersphere balls as centers.

Cover trees are efficient in nearest-neighbor searching. Both single query and batch query are supported with the cover tree data structure [26]. Take $B_S(p, r)$ as a hypersphere ball with r as the radius and p as the center of the ball. A parameter named c denotes the expansion constant, so that $c \geq 2$. And for all data points p , $|B_S(p, 2r)| \leq c|B_S(p, r)|$ for $r > 0$. The choosing of expansion constant is affected by the dimension of the dataset. If the dataset is k dimensional, then $c \geq 2^k$. In the same time, cover trees occupy small space with a space complexity of $\mathcal{O}(n)$, where n refers to the size of the dataset.

2.1.5 Range Tree

The range tree is a balanced binary tree which can be used to index multi-dimensional dataset [30, 31]. Both point queries and exact range queries are supported by the range tree because the range tree has exact leaf nodes for each value on each dimension of the dataset. For multi-dimensional dataset, the range tree is built as a binary tree on the first dimension. From each node in the previous dimension, a sub tree on the next dimension is built. One example of 1D range tree and one example of a two-dimensional tree are shown in Fig. 2.5:

Figure 2.5: An example of 1D range tree and querying with 1D tree. (Eastman, 2012)

As Fig. 2.5 shows, a 1D range tree construction can be regarded as a binary search tree. The 1D range tree stores all the data items into its leaf nodes, and its internal tree nodes are all assigned with the largest value of the tree under its left child node. A 1D range tree has a time complexity same as binary search tree. However, with the increasing number of dimension, the time complexity of range tree construction goes up:

Figure 2.6: Bottom: 2-layer range tree for 2D range query set. (Qiao, Gan & Tao, 2016)

As Fig. 2.6 shows, among all the dimensions, one dimension needs to be chosen to construct the basic 1D range tree, and for each internal node of the 1D range tree, a sub range tree indexing data items in the second dimension will be constructed under the range of the first dimension node. The number of levels of subtrees depends on the

dataset dimension, and in each level, the number of sub trees depends on the number of nodes in the last level. For example, as shown in the bottom figure in Fig. 2.6, to construct a 2D range tree, the x -dimension is selected for constructing a 1D range tree. For nodes in the 1D range tree, the second layer of subtrees are constructed to describe the ranges existing in the y -dimension with specific x values. This fact makes the time complexity to increase with the dimension.

With a range tree, both exact point queries and exact range queries are supported. The 1D range query is the simplest query to be explained. As shown in the top figure of Fig. 2.5, the structure of the 1D range tree is similar to a binary search tree. For the point query, we query the data point as what we do to a binary search tree: We start from the root of the range tree, compare the query input to the root. If the internal tree node is exactly the input of query, then we directly return the largest leaf node of its left subtree.

For the range query $[x_1, x_2]$, as shown in the bottom figure of Fig. 2.5, the 1D range tree will search the values of both x_1 and x_2 till the leaf node level. If we are searching x_1 and x_2 in the range tree, there will be two search paths: v_{split} to x_1 and v_{split} to x_2 , where v_{split} is the last internal tree node on the common path of the search paths of x_1 and x_2 . For all the tree nodes on the search path v_{split} to x_1 , if the value of the tree node v is larger than x_1 , all the leaf nodes of the tree under v 's right child node will be added to the query result. If v is a leaf node and larger than x_1 , then it will be added to the query result. The contrast operation is applied to the tree nodes on the search path from v_{split} to x_2 . The collection of nodes with a set of subtrees is named as *Canonical Set* [67, 68]. The result of exact range query is expressed in Canonical Set form. In the bottom figure of Fig. 2.5, the canonical set of the range query $[x_1, x_2]$ contains 6 trees under the internal tree nodes of the two search paths. Since there are only two search paths in the range tree, and there are at most $\mathcal{O}(\log n)$ tree nodes on each search path,

the time complexity of range query is $\mathcal{O}(\log n)$ for a 1D range tree. If there are s data items falling in the query range in total, then the time complexity of reporting the query result will be $\mathcal{O}(s)$ if the size of query result is s . Therefore, the final time complexity of range query is $\mathcal{O}(\log n + s)$.

For a multi-dimensional range tree, the subtrees on the other dimensions are also involved in the exact range query. The multi-dimensional range tree will search the query recursively from the 1D range tree to the subtrees on the last dimension when a query comes. From Fig. 2.5, to query the range $[(x = 5, y = 3), (x = 15, y = 11)]$, we firstly query the range $[5, 15]$ on the 1D range tree of x axis. Then for all the internal tree nodes on the search paths of $[5, 15]$, we check whether there are subtrees of y axis attached to them. For these subtrees of y axis under the internal tree nodes, we query the range $[3, 11]$ of y axis. The leaf nodes of all the subtrees on the other dimensions are always falling in the range under the subtree roots in the previous dimension. Therefore, in this example, all the y axis subtree leaf nodes falling in the range $[3, 11]$ must be in the range $[(x = 5, y = 3), (x = 15, y = 11)]$. Therefore, all these leaf nodes are added to the query result dataset. Similar to 1D range query, for the first dimension, the query time complexity is $\mathcal{O}(\log n)$, and the number of subtrees in the Canonical Set of $[5, 15]$ is in $\mathcal{O}(\log n)$. Therefore, for the second dimension, there will be $\mathcal{O}(\log n)$ subtrees being queried. The query time complexity is $\mathcal{O}(\log^2 n)$, and the Canonical Set size of all the subtrees is in $\mathcal{O}(\log^2 n)$. Therefore, finally for a k -dimensional dataset, the query time complexity is $\mathcal{O}(\log^k n)$. The time complexity of reporting the query result dataset is s when the size of query result is s . Therefore, the final time complexity of range query is $\mathcal{O}(\log^k n + s)$.

To speed up the query progress, a technique named *fractional cascading* is used [32, 33]. Fractional cascading is important in solving iterative search problems, especially geometric retrieval problems, which is one of the main problems in my research [32]. To

explain what is fractional cascading, we need to firstly give an example on fractional cascading: When we have two sorted arrays $A1$ and $A2$ with $A2 \subseteq A1$, and we want perform a range query on both $A1$ and $A2$, a naive query search can take $\mathcal{O}(2\log n + s)$, where n is the size of $A1$ and s is the size of query result. However, since we have $A2 \subseteq A1$, we can decrease the query time cost, that is we construct pointers from $A1[i]$ to $A2[j]$ for any i and j within the array index, where $A2[j]$ is the smallest element in $A2$ that $A1[i] \leq A2[j]$. An example of these pointers are shown below in Fig. 2.7:

Figure 2.7: An example of pointers between sorted arrays (Suri, 2002)

From Fig. 2.7, with these pointers, for querying $A1$ of the range $[x_1, x_1]$, we firstly query x_1 to find the smallest $A1[i]$ so that $A1[i] \geq x_1$, and the time complexity is $\mathcal{O}(\log n)$. Then we add s_1 elements larger than $A1[i]$ and smaller or equal to x_2 . The total time complexity is $\mathcal{O}(\log n + s_1)$. After we do the range query on $A1$, we follow $A1[i]$'s pointer to $A2$ to find $A2[j]$, and directly start the querying from $A2[j]$. Since we have $A2 \subseteq A1$, starting the query search from $A2[j]$ will never miss any query result elements in $A2$. The time complexity of finding $A2[j]$ using pointer is $\mathcal{O}(1)$, and reporting s_2 query result elements in $A2$ takes $\mathcal{O}(s_2)$ time complexity, where $s_1 + s_2 = s$. Therefore, the final time complexity of range querying the two sorted array is $\mathcal{O}(\log n + s + 1)$ instead of $\mathcal{O}(2\log n + s)$. The method of using pointers across sorted arrays to speedup range query is named as *fractional cascading*.

To achieve this goal in multi-dimensional range trees, instead of construct the subtrees of data items for the last dimension, we construct sorted arrays to store these data items, and we construct pointers from the subtrees of the last dimension to the elements of the sorted array. If we have k dimensions for the dataset, we use T as the notation of a subtree on the $k - 1$ th dimension, T_l and T_r as the left and right subtrees of T on the $k - 1$ th dimension. We use the data items in T to construct sorted array A by sorting the data items of T on k th dimension, and we use the data items in T_l and T_r to construct sorted arrays A_l and A_r . Obviously, we have $A_l \subseteq A$ and $A_r \subseteq A$. Therefore, we can construct two pointers for each data items in A . One is to elements in A_l , the other is to elements in A_r . The pointers are still pointing to the smallest sorted array elements which are larger or equal to the data items.

When a multi-dimensional range query comes, we firstly search the range tree from the first dimension tree to $k - 1$ th dimension subtrees in an iterative order. Then for the k th dimension, we do not need to check all the sorted arrays attached to the internal tree nodes on the two search paths. We only need to perform searching to the sorted array A of the internal node v_{split} . Then for all the sorted arrays on the search paths from v_{split} to x_1 and x_2 , we only need to access them with a time complexity $\mathcal{O}(1)$, for they are the descendant nodes of v_{split} . Therefore, the total time complexity for range querying a multi-dimensional range tree with fractional cascading is $\mathcal{O}(\log^{k-1}n + s)$.

2.1.5.1 Comparison between Data Structures

The time complexity for construction, space complexity, time complexity of range query will be compared. Let k denote the dimension of data, n the number of data items, c the expansion constant used in the cover tree, s the number of data items that should be retrieved as the results of the query. Table 2.1 shows a comparison of the few major indexing schemes reviewed:

Table 2.1: Comparison between KD Tree, Cube Tree and Range Tree

Issues to Be Compared	KD Tree	Cube tree	Ball tree	Cover tree	Range Tree
Balanced or not	Balanced	No guarantees to be balanced [22]	Balanced	Can be balanced through re-balancing [27]	Balanced
Multi-dimensional dataset supported or not	Supported	3D (2D for Quadtree), can be extended to multi-dimensional data structure as a variant	Supported	Supported	Supported
Exact range query supported or not	Supported	Not supported	Supported	Supported	Supported
Time Complexity of Construction (Worst case)	$\mathcal{O}(kn \log n)$	Time complexity is largely depending on the number of levels of cube tree	$\mathcal{O}(n(\log n)^2)$	$\mathcal{O}(c^6 n \log n)$	$\mathcal{O}(n \log^{k-1} n)$
Space Complexity	$\mathcal{O}(n)$	Largely depends on the type of the cube tree and the limitation of maximum number of levels. If the maximum number of levels is fixed, then the data storage will be a constant C .	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n(\frac{\log n}{\log \log n})^{k-1})$
Time Complexity of Range Query	Range query: $\mathcal{O}(n^{1-1/k} + s)$	Non-exact range query: $\mathcal{O}(\log n)$ [34]	For pruning operations: $\mathcal{O}(\log n + s)$	Query: $\mathcal{O}(c^{12} \log n + s)$ Batch Query: $\mathcal{O}(c^{16} n + s)$	Range Query: $\mathcal{O}(\log^{k-1} n + s)$
Nearest-neighbor search supported or not:	Supported	Supported	Supported	Supported	Not supported

Although cube tree is not used for exact range query, it can be used for nearest-neighbor searches. Constructing a KD tree is faster than constructing a range tree, a ball tree or a cover tree. However, range querying with a KD tree is less efficient than with a range tree. If both exact range queries and nearest-neighbor searches are to be performed on the dataset, KD tree might be a better choice with more general query functions.

2.2 Topic Modeling

2.2.1 Definition and Purpose of Topic Modeling

According to Blei, Ng & Jordan (2003), in LDA model, each document in the dataset can be regarded as a bag of words. To generate a document, firstly an abstract topic is chosen as one of the topics of the document. For the chosen topic, a set of words are assigned to the topic, and one of the word is chosen to be put into the document. By repeating the selection operations, many words are chosen and a document is formed with these words. The topics of a document is presented by probability distribution, and the words of a topic are also presented by probability distribution. Another example of topic model is pLSA (Probabilistic Latent Semantic Analysis): For each document, a topic is selected and assigned to this document. Therefore, based on the idea of pLSA, whenever a word is added to generate a document, a word under the chosen topic is selected following the multinomial distribution of vocabularies. Now we take the words as w , the topic assigned to word w as z , and the topic distribution over one document d as θ_d , a graphic model of pLSA is shown in Fig. 2.8:

Figure 2.8: pLSA graphic topic model. (Blei, Ng & Jordan, 2003)

In Fig. 2.8, the outer plate represents the document d . For the document, and the inner plate shows that topic is a mixture of vocabularies. By following distribution, θ_d a topic assignment z_n is chosen for the n th word. By following z_n , a word is finally chosen as w_n . In the view of LDA, the document generation progress is "choosing a topic based the topic distribution and choosing a word based on the word distribution". In reality, the documents are known, and the vocabularies are easy to be retrieved with document collection. The unknown part is the latent topics hidden behind the document collection. Therefore, topic modeling algorithms are designed to discover these abstract topics.

The purpose of topic modeling is to describe the mixture of topics talked about in a dataset. In my research, the social media data change with time, location and other attributes. The changes of these attributes are reflected by the contents of data, and topic models are able to explicitly describe the changes in the data. With topic models trained with dataset of different time periods and geographical regions, we may be able to extract the patterns of changes. An example is uncovering the relationship between the topic models and time, which reveals the speed of changing a main topic and the patterns of changes of topic. Another possible direction might be finding the models describing the latent relationship between the "popularity" of a specific topic and the

distance from the center of the area that the topic is talked about. In summary, there is still large potential in topic modeling analysis, and more research problems relevant to social science and data mining are involving topic modeling.

Although multiple topic modeling algorithms are available for the research, some algorithms appear to be more suitable. The requirements of applied topic modeling algorithm are listed below:

- Nonparametric algorithm: A parametric topic modeling algorithm needs setting of the number of topics. However, the topic number setting is done by the data analysts, which might not be the exact situation. Using a nonparametric topic modeling algorithm eliminates the need to specify the number of topics. The algorithm should adaptively choose the number of topics according to the data.
- Online training algorithm: Although currently the dataset used in the research is static, the dynamic dataset might be used as an extension. The fast topic model combining algorithm also combines the smaller dataset to the bigger dataset. Therefore, the idea of online training algorithm should be taken into consideration for fast modeling process.

2.2.2 Relevant Research of Topic Modeling

Six topic modeling algorithms will be briefly introduced, and five of them will be compared against each other. The relevant issues involved with topic modeling framework and inference like MCMC (Markov Chain Monte Carlo) and Gibbs Sampling will be reviewed. After introduction of framework, with the comparison of algorithms in the next section, one of the algorithms named LDA will be chosen as the topic modeling algorithm used in my research. An overview of the algorithms to be reviewed is shown in Fig. 2.9:

Figure 2.9: Topic models

In Fig. 2.9, the first topic modeling algorithm is LSA (Latent Semantic Analysis), which is the first algorithm that originated the idea "topic" and systematically estimated the topic by constructing a document-term matrix and by use of SVD (Singular Value Decomposition) to downsize the matrix [35]. Based on LSA, pLSA introduces probabilistic method [36]. It estimates the probability of the co-occurrence of a pair of word and document, and from the probability $P(w, d)$ the probability of topics are estimated.

After pLSA, a new algorithm LDA (Latent Dirichlet Allocation) is designed and soon becomes the most popular algorithm [13, 37]. LDA is a mathematical generative model, which involves Bayesian theorem. The main difference between LDA and pLSA is that LDA uses Bayesian theorem. Another difference is that LDA uses Dirichlet Process to model the word distribution and topic distribution, which simplifies the inference process. For pLSA, EM (Expectation-Maximization) method is used for inference, which takes a number of iterations with maximizing the difference between likelihood of θ in each iteration. For LDA, the inference will be simpler with Bayesian theorem, if we use MCMC and Gibbs Sampling instead of VB (Variational Bayesian Method) [38].

Based on LDA, there are many variants created for improvements in the modeling quality or execution efficiency. In Pachinko Allocation, a DAG (Direct Acyclic Graph) is created by taking words w as leaf nodes and put topics to nodes at higher levels [39]. This allows to model the relations among different topics. The other variant reviewed for the research is hLDA (hierarchical Latent Dirichlet Allocation) [13]. This variant provide non-parametric Bayesian inference which does not require a pre-setting of the number of topics. To see the differences and relationships between the topic modeling methods, Table 2.2 is created for the comparison of graphic models of pLSA and LDA:

Table 2.2: Comparison between pLSA and LDA Graphic Model

Topic Modeling Methods	Probabilistic Graphic Model
pLSA	<p>(a) pLSA Graphic Topic Model: To generate a document, a topic assignment z_n is randomly selected following the topic distribution, and the word w_n is randomly selected as the nth word of the document by following the word distribution. (Blei, Ng & Jordan, 2003)</p>
LDA	<p>(b) LDA Graphic Topic Model: There are two parameters α and β, which are Dirichlet prior of topic distribution over document and Dirichlet prior of word distribution over topic respectively (Blei, Ng & Jordan, 2003)</p>

From Table 2.2, it can be summarized that both pLSA and LDA are generating a document through one topic distribution and one word distribution. However, pLSA constructs a more simple graphic model, while LDA introduces two parameters α and β as the Dirichlet priors for the two distributions. These parameters make the usage of Dirichlet Distribution possible. Because for Dirichlet Distribution, the posterior distribution is in the same family of the prior distribution, it becomes easy to perform sampling for each iteration with using the posterior probability resulted from last iteration as the prior probability for the next iteration. After comparing pLSA and LDA, the three variants of LDA are also compared in Table 2.3:

Table 2.3: Comparison of Pachinko Allocation and hLDA

Topic Modeling Methods	Probabilistic Graphic Model
Pachinko Allocation	<p>A Direct Acyclic Graph of Pachinko Allocation: All the words are regarded as the leaf nodes, while the topics are the parent nodes of higher levels. (Li & McCallum, 2006)</p> <p>(a) LDA (b) Four-Level PAM</p> <p>Graphic Topic Model Comparison between LDA Model and Four-Level Pachinko Allocation Model. (Li & McCallum, 2006)</p>
hLDA	<p>Graphical topic model of hLDA with a nested Chinese Restaurant Process Prior. (Blei, Griffiths, Jordan & Tenenbaum, 2003)</p>

From Table 2.3, we can figure out that for all the three variants of LDA, they are using different method to enhance the performance. Pachinko-allocation and hLDA are both focusing on the logic of topic hierarchy in real case. For example, the nested Chinese Restaurant Process used in hLDA refers to a topic model structure which is very close to the real case [13, 42, 43]. By taking the root node as the door of a Chinese restaurant, and its child nodes as tables, we may take each document as the customer, and figure out the path it follows. Based on the path chosen, the probability distribution of each node within the path will be inferred from the Gibbs Sampling result of the dataset. A sample of the subtree of the hLDA topic model is shown in Fig. 2.10:

Figure 2.10: A subtree under the topic "Computer" of a hLDA model. (Chen, Zhu, Lu & Liu, 2018)

Table 2.4 shows a comparison of the reviewed algorithms, and the features of each topic modeling method:

Table 2.4: Comparison between Topic Modeling Algorithms

Issues to Be Compared	PLSA	LDA	Pachinko Allocation	hLDA
Brief Introduction	LSA is the algorithm which set up the concept of "abstract topics". Its variation PLSA is an early topic modeling algorithm with improving LSA by introducing probabilistic theory.	LDA is a general form of PLSA, which is one of the most popular topic modeling algorithms. Bayesian theorem is involved in LDA for inference.	Pachinko Allocation is a variation of LDA, which provides a direct acyclic graph of topics and words.	hLDA is a variation of LDA which introduces nonparametric inference.
Probabilistic Distribution Involved	Multinomial distribution	Dirichlet distribution	Dirichlet distribution. For an improved non-parametric Bayesian prior version of Pachinko Allocation, hierarchical Dirichlet Process is used [38].	Nested Chinese Restaurant Process [39]. Hierarchical Dirichlet Process [40, 41]
Pros	N.A.	Mathematically generative compared to PLSA. Modeling the prior explicitly. Getting rid of EM method. Using MCMC or Gibbs sampling method, which are simpler.	Enhancing the flexibility and expressive ability [42].	Removing the limitation in prior of topic distribution by using nonparametric Bayesian inference. Representing topics in a hierarchical structure.
Cons	Using EM method. Implicit prior.	Have to set the number of topics in advance (less flexibility). Sequential algorithm.	Sequential algorithm.	Complex topic model structure for inference.

From Table 2.4, LDA is a classical topic modeling algorithm with variations of different improvements. Although LDA is frequently used in topic modeling, the selection of optimal number of topics becomes an important problem [44]. Since it is difficult to select the number of topics, nonparametric Bayesian inference variation of LDA is important in further research. The parallel algorithm is also needed to handle the huge dataset. Therefore, the two directions of improvement will be taken into consideration. Besides these algorithms mentioned, dynamic version of LDA is also used to train incoming data items and combine the topic models with existing topic models [14]. The dynamic version is named as Online-LDA.

After comparison of multiple topic modeling algorithms, the method revealed in LDA in my research, and hLDA might be also involved to have a non-parametric version of the framework. The algorithm LDA will be further studied and improved with these features to be applied as a more suitable algorithm in the data analyzing process.

2.3 Topic Model Evaluation

To measure how close the topic modeling result is to the actual topic models, the topic model derived by an algorithm should be evaluated. There are multiple metrics for evaluating a topic model. One of the most popular evaluation metrics is perplexity. Based on perplexity, an algorithm named "left-to-right" is also used for topic model evaluation. Besides perplexity and left-to-right, among different distances and divergence for measuring the difference, Bhattacharyya distance is also an important evaluation metric for measuring the difference of two probabilistic distributions, which is suitable for calculating the difference of topic models.

2.3.1 Perplexity

Perplexity is one of the most suitable measurements, for it is applied to measure the quality of the probability distribution. With the perplexity, different probabilistic mod-

els may be uniformly compared [39, 60]. In the case of topic modeling, the results of topic modeling are probability distributions of words over topics and probability distributions of topics over documents. Therefore, perplexity is frequently used in topic model evaluation.

The main idea of calculating perplexity of a probability distribution is to sample n words $w_1, \dots, w_n$, and predict the next word w_{n+1} with the sample set. To minimize the uncertainty of prediction, the concept of entropy in data mining is used [59, 61]. The formula of entropy is:

$$Entropy = H(\mathbf{W}) = - \sum_{i=1}^n P(w_i) \log_2(P(w_i)) \quad (2.1)$$

The higher the entropy is, the higher the uncertainty of the prediction is, and the worse the probability model is when it is used to describe the topic model. If we take the entropy as exponent and the value "2" as the base, then the formula will be:

$$Perplexity = 2^{H(\mathbf{W})} = 2 - \sum_{i=1}^n P(w_i) \log_2(P(w_i)) \quad (2.2)$$

Thus, the perplexity should be minimized to achieve best quality of topic model. Since the topic mixture and word occurrence are represented in the probability form in LDA and its variants, perplexity can be directly applied for evaluation.

2.3.2 Left-to-right Algorithm

Although the perplexity is widely used for topic model evaluation, we still have a method for further evaluation of topic model, which is called "left-to-right" algorithm [45, 46].

The algorithm is used to evaluate the topic model generated from training dataset with the testing set as the benchmark. We take the topic t distributed over the words with the probability as ϕ_t , the $\Phi = \phi_1, \dots, \phi_T$ as the topic probability distribution vector, the tokens as w , and the document collection as m . To measure how well the topic model

of training dataset describes the testing dataset, we take the training set of data as W' , and the testing set as W . Then the probability used for measuring the topic model will be:

$$P(W|W') = \int d\Phi d\alpha d\mathbf{m} P(W|\Phi, \alpha\mathbf{m}) P(\Phi, \alpha\mathbf{m}|W'), \quad (2.3)$$

where α is the Dirichlet prior.

From the formula, we can briefly take the probability $P(W|W')$ as estimating the value of $P(W|\Phi, \alpha\mathbf{m})$ with the sampling of training dataset or with point estimate. Since LDA topic provides the topic probabilistic distribution Φ of the training dataset, we can retrieve the probability $P(W|\Phi, \alpha\mathbf{m})$ by sampling the words in W' under the topic model and topic assignment of words of training dataset. The formula 2.4 to describe how left-to-right algorithm estimates the probability $P(W|\Phi, \alpha\mathbf{m})$ is below:

$$P(\mathbf{w}|\Phi, \alpha\mathbf{m}) = \prod_n P(w_n|\Phi, \alpha\mathbf{m}) = \prod_n \sum_{z \leq n} P(w_n, z \leq n|\Phi, \alpha\mathbf{m}), \quad (2.4)$$

where n refers to the word position in the document. The algorithm samples testing dataset items with the training dataset topic model as the reference, and calculate the probability.

2.3.3 Distances and Divergence

Since the topic model is described in the format of probability distribution, the difference between two topic models can be measured by different metrics including distances and divergence. To select a suitable statistical metric, we need to review their definitions and features.

In statistics, we use different metrics to examine the difference between different datasets. For example, in data clustering, we have so many different metrics for measuring the distances between data clusters [50, 51, 52, 53]. Distances like Euclidean distance

(also named as l_2 distance) can be used for calculating the similarity of data and clustering the data. Furthermore, Manhattan distance (also named as l_1 distance) is usually used to describe the geometry distance in an environment of city blocks, while Chebyshev distance (also names as l_∞ distance) can provide better performance in clustering convergence for situations like medical image clustering [56]. Besides, Minkowski distance is the generalized distance of Manhattan distance and Euclidean distance, which is also widely used for data clustering [50, 51, 53, 55].

Although there are so many forms of distance used for the data clustering, when it comes to the situation of probabilistic distribution, we still need to think of new approaches for measuring the difference. The first metric widely used for the probabilistic distribution similarity calculation is Kullback-Leibler divergence [57, 58]. Kullback-Leibler divergence can measure the similarity of both discrete and continuous probabilistic distributions. The formula for calculating the Kullback-Leibler divergence of probabilistic distribution P and Q is

$$D_{KL}(P||Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right) \quad (2.5)$$

for the discrete probabilistic distribution. And for the continuous probabilistic distribution, the definition of Kullback-Leibler divergence is

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx \quad (2.6)$$

Although Kullback-Leibler divergence measures every probability in the distribution, a serious problem makes it not suitable for evaluating every topic model: Since the probability values are taken as the denominator in the calculation, there must not be any probability with the value 0 in the two probabilistic distributions. However, it is possible that θ or ϕ probabilistic distribution contains 0 value for topic models, though the

possibility is very low. Therefore, to guarantee that there are no mistakes in evaluation of any topic model, Kullback-Leibler divergence is not used for topic model evaluation in our research.

To avoid the problem of 0 denominator, we turn to another statistical metric Mahalanobis distance, which can be used to measure the distance from an observed point to a dataset with known mean and covariance [62, 63, 64]. It can also be used to measure two different probabilistic distribution vector without the consideration of 0 values [65, 66]. However, Mahalanobis distance requires the two probabilistic distribution vector to share the same covariance matrix, which is a limitation.

To get rid of the limitation of Mahalanobis distance, we finally choose to use Bhattacharyya distance, which is more generalized than Mahalanobis distance, and still can avoid the problem of 0 value of probability [47, 48, 49]. The Bhattacharyya distance is defined based on the Bhattacharyya coefficient. Similar to Kullback-Leibler divergence, Bhattacharyya coefficient and Bhattacharyya distance can also measure both discrete and continuous probabilistic distribution [72, 73]. Since the topic model is also presented in the form of probabilistic distribution, Bhattacharyya distance has been used to measure the similarity of topic models [75, 76]. For the discrete probabilistic distributions p and q , the definition of Bhattacharyya coefficient is

$$BC(p, q) = \sum_{x \in \mathbf{X}} \sqrt{p(x)q(x)} \quad (2.7)$$

For the continuous probabilistic distributions, the definition of Bhattacharyya coefficient is

$$BC(p, q) = \int \sqrt{p(x)q(x)} dx \quad (2.8)$$

Bhattacharyya distance is calculated based on the value of Bhattacharyya coefficient

$$D_B(p, q) = -\ln(BC(p, q)) \quad (2.9)$$

Since Bhattacharyya distance gets fewer limitations compared to Mahalanobis distance and Kullback-Leibler divergence, and it is suitable for measuring the difference of probabilistic distributions, we decide to take it as one of the metrics for evaluation. The difference from the Topic Ranger generate model to LDA model will be measured by Bhattacharyya distance.

2.4 Fast Set Sampling Algorithm

To enhance the time efficiency in topic modeling, a topic model combining algorithm is designed for topic model querying problems [15]. The algorithm is named as "Fast Set Sampling algorithm (FSS)", for the algorithm solves the topic model query problem with a fast sampling method. The algorithm is specially designed for the spatio-temporal corpus and the topic modeling for quickly retrieving the topic model of spatio-temporal range query result, which is exactly the situation for the social media data.

The main idea of using the FSS algorithm to solve topic-range query is that we solve the topic-range query by combining pre-computed topic models of data cubes instead of directly computing the topic model for the query result dataset. The FSS algorithm contains 3 steps to solve the topic-range query:

- Topic model pre-computing: We index the whole corpus with the cube tree. Each tree node of the cube tree is a data cube. We use the data items within the data cube to compute LDA topic models and store them.
- Topic-range querying: We set a threshold $\tau \in (0, 1)$ for the topic-range query. When a topic-range query comes, we take the query range as a cube, and we select all the tree nodes intersected with the query cube. We check whether the tree nodes

are covered by the query range over the ratio τ . For all the tree nodes covered by the query range over the ratio τ , we put them into a heap. For those tree nodes covered by the query range less than τ , we collect all the queried data items from these tree nodes into a list. We name the list as "remaining list".

- Topic model approximation: We pop out the largest tree node from the heap, and take its topic model as the based topic model $Z1$. Then we pop out the second largest tree node in the heap. We use $Z1$ as a reference to re-sample the word tokens in the smaller tree node. Here, different from OLDA, we do not re-sample all the tokens in the documents simultaneously. Instead, we only re-sample the word tokens under the same topic at the same time. This helps to save runtime cost in approximating the topic model. After re-sampling, we finally have the count of topic assignment of a topic k to a word v . Then we combine the topic models of the two tree nodes by summing up the count of topic assignment of k to v of the two tree nodes. The new distribution of the topic assignment is the approximate topic model of the two tree nodes.

After combining the topic model of the second largest tree node to the topic model of the largest tree node, we repeat the re-sampling to the next largest tree node in the heap. We perform the re-sampling until the heap is empty. Then for the remaining list, we use the approximate topic model as a reference to re-sample its remaining list. After adjusting the approximate topic model with the re-sampling result of the remaining list, we finally have the approximate topic model of the query range. It is taken as the query result of the topic-range query.

Although the FSS algorithm provides an approximate topic model, it can greatly decrease the runtime cost in answering the topic-range query. To apply FSS algorithm, a cube tree is built for a spatio-temporal dataset, and apply LDA algorithm for topic

modeling of data to each cell of the cube tree. Let W denote the number of words, K denote the number of topics and I denote the number of sampling iterations. The time complexity of training topic models for a dataset is $\mathcal{O}(WKI)$. The main concept of the FSS algorithm is that if two adjacent cells are already trained for topic models, then the topic model of the bigger cell consisting of the two cells can be trained by combining the topic models of the two cells. Let M denote the number of documents in the cell, and V denote the vocabulary size of the cell. In previous Online-LDA methods, by re-training the smaller cell the topic models are combined, which take $\mathcal{O}(W_2KI)$, where W_2 is the number of words in the smaller cell. The combining algorithm further improved the time complexity to $\mathcal{O}(K(K + |M_2| + |V_2|)I)$ with collapsed Gibbs Sampling.

Beside the high efficiency in topic model combining, a bounded error is also compared to online LDA algorithm. The error bound of combining two datasets W_1 and W_2 is $\sqrt{2 \frac{|W_1 \cap W_2| + |W_2| - 1}{|W_1| + |W_2|}}$. From the error bound, we can figure out that the smaller the intersection between W_1 and W_2 is, the smaller the error bound is. A smaller size of W_2 gives a smaller error bound, too.

To evaluate the quality of the combined topic models, the evaluation metric "left-to-right algorithm" is used to estimate a perplexity [45, 46]:

$$Perplexity(W_{test}) = \exp\left(-\frac{\log P(W_{test}|\Phi_{train})}{|W_{test}|}\right) \quad (2.10)$$

By testing FSS algorithm on Twitter dataset and Meetup corpus, the runtime cost for range query topic modeling is proved to be decreased by FSS algorithm, while the perplexities of the topic models are within the accuracy threshold. FSS algorithm provides topic models with good quality while deducting the time cost, which enables fast topic-range query of spatio-temporal dataset.

The algorithm provides a new direction of time efficiency improvement with combining topic models pre-trained for cube tree cells. However, it is hard to perform exact range

query with cube tree, i.e., it is difficult to get the exact queried sub-dataset for an arbitrary range query. Consequently, the topic models used for combining the pre-computed models may not exactly cover the queried range, and the combined topic modeling result might not be accurate for general range queries. The total size of combined cells might also varies in different data structures, and causes different qualities of resulting topic models. Therefore, it is worthy for us to design variations of FSS algorithm with other data structures, and provide more evaluation metrics for topic models.

2.5 Summary

In this chapter, to design an algorithm for answering the topic-range queries, we have reviewed the FSS algorithm, which is the base of our algorithm. We have also reviewed the data indexing structures, the topic modeling algorithms and the topic model evaluation metrics. In order to support the exact range queries over a multi-dimensional dataset, we have selected the range tree as the data indexing structure. For the topic modeling algorithms, we have chosen LDA as the algorithm used for topic modeling, for it is one of the most widely used topic modeling algorithms. We have also selected LDA and Online-LDA as benchmarks to compare with our design in topic-range queries. Among the topic model evaluation metrics, we have decided to use perplexity to evaluate the quality of topic models. We also use the range query result to directly calculate a LDA topic model, and use Bhattacharyya distance to measure the difference between the LDA topic model and the topic model computed with Topic Ranger.

Chapter 3

Topic Ranger: Methodology

Based on the design and implementation of FSS algorithm, a new design named as "Topic Ranger" has been developed. The new design uses another data structure to combine with FSS algorithm, which makes the cells and data items used for topic combination different from the previous design. The new design also makes the implementation more flexible for data storage, pre-computing of topic models and topic model range query. In this section, we firstly describe the structure of Topic Ranger, and then we show the improvements and the algorithm of Topic Ranger.

3.1 Structure of Topic Ranger

Among all the reviewed data structures, we decided to choose range tree as the data structure to combine with FSS algorithm. The range tree can provide these benefits:

- Range tree supports multi-dimensional dataset. Therefore, the spatio-temporal social media dataset can be well organized with range trees.
- Range tree supports exact range query. Different from the cube tree, which might be affected by the depth, the range tree can always retrieve the results of exact query without checking the data points in a cell one by one.

- Range tree is flexible. Although we usually build the range tree with 2 branches, we can adjust the branching factor to any integer for better performance or for smaller time or memory costs. This is quite different from the cube tree, which only divide the cells into 8 child cells for each iteration.

Based on these features of range tree, we decided to use range tree for the variation of FSS algorithm, and we tested the memory size and time requirement of our range tree implementation. In the range tree, each node has a subtree including its descendants. The collection of nodes with a set of subtrees is named as *Canonical Set*, which has been briefly mentioned in Chapter 2. The result of exact range query is expressed in Canonical Set form [67, 68]. An example of Canonical Set is shown below in Fig. 3.1:

Figure 3.1: Canonical set for a range query (Mount, 2001)

When we query the range $(2, 23)$ on the 1D range tree shown in Fig. 3.1, we search down from the root to the leaf nodes of the 1D range tree, and return all the leaf nodes with value between 2 and 23. The query result naturally forms a set of five subtrees in the range tree. The five subtrees are marked with shadowed circles in Fig. 3.1, which cover the result of the range query $(2, 23)$ together with their child nodes. The set of the five subtrees is called the Canonical Set of the result of range query $(2, 23)$.

Similar to topic modeling process in the original FSS algorithm, the topic models are pre-trained from the lower levels to the root of the range tree. Therefore, similar to FSS algorithm, the topic model of queried result can be retrieved by combining the topic models of each of the subtree root nodes in the Canonical Set. For example, when we perform a range query to a 1D range tree, we can retrieve the Canonical Set of the query, and put the root of the Canonical Set subtrees into a heap as shown in Fig. 3.2:

Figure 3.2: An example of converting Canonical Set of a range query into a heap

As shown in Fig. 3.2, similar to the covered cube tree nodes in the FSS algorithm, we can also put the covered subtrees in the Canonical Set of a range query into the heap, pop them out by their size, and re-sample the data items in the smaller subtrees to update the topic model of the largest subtree. Therefore, with the Canonical Set, the FSS algorithm can be applied to the range tree in the same way as the cube tree.

Beside the possibility of applying the FSS algorithm to the range tree, the time complexity of using the range tree for topic model approximation is also guaranteed in theory. Canonical Set has a property that at each level of the range tree, the number of subtree root nodes never exceeds 2. Every two subtrees sharing the same parent node will be represented by their parent node, which means that in each level only the two end of the query range might not have adjacent node sharing the same parent. In conclusion, the number of root nodes in a Canonical Set in a range tree is $\mathcal{O}(\log n)$, where n denotes the number of leaf nodes, and $\log n$ denotes the height of the range tree. For the topic model combining process, similar to combining smaller cube tree cell topic models to the bigger cube tree cells, the combining process is from the lower level to the higher level of the range tree.

To make sure that the range tree supports exact query, we construct the range tree with only a single data item in each leaf node. Let n denote the number of documents in a dataset, then the number of leaf nodes in the range tree of the first dimension will be n . Assume that the number of words in each document is limited by $\mathcal{O}(w)$, the dimension of dataset is d , and k topics are to be generated from a subset of data. Then the exact range query time cost is in $\mathcal{O}(\lg^d n)$ for searching certain values in a multi-dimensional range tree. The pre-training time cost of topic models will be $\mathcal{O}(wkI \cdot n \lg^d n)$ [70]. With using LDA algorithm for topic modeling of w words and k topics, the time complexity is $\mathcal{O}(wkI)$, where I is the number of iterations in Gibbs Sampling. Since there are n documents, for each level of the range tree, the topic modeling takes time in $\mathcal{O}(wkI \cdot n)$,

and the maximum height of a 1D range tree is $\mathcal{O}(\log n)$. Therefore, for a 1D range tree, the topic modeling pre-training time complexity is $\mathcal{O}(wkI \cdot n \log n)$. For a d -dimensional range tree, as in exact range query, $\mathcal{O}(\lg^d n)$ levels are transversed. Therefore, the pre-training process of topic modeling for the nodes in a d -dimensional range tree is $\mathcal{O}(wkI \cdot n \lg^d n)$. Based on the queried result and pre-trained topic models, the topic model of queried dataset will be retrieved with the combining algorithm. The time complexity is upper-bounded by $\mathcal{O}(\sum_{i=1}^x (k_i + m_i + v_i) k_i I) = \mathcal{O}((klg^d n + M + Vlg^d n)kI)$, where x denotes the number of root nodes in the Canonical Set, m denotes the number of documents and v denotes vocabulary size.

With the usage of canonical set of range tree, we can easily complete the same process done to the cube tree in FSS algorithm. However, since the subtrees for other dimensions of the dataset are built on each internal nodes of the range tree, the memory size will be larger than expected. And also, although the time complexity of tree construction is only $\mathcal{O}(n \log^{k-1} n)$, the actual time cost is very large for construct one range tree. Therefore, we decide to build a hierarchical range tree with only the range of the data items and pre-computed topic models, and taking small binary range trees as leaf nodes for fast querying of each queried data item. In this way, the runtime and memory cost for tree construction and pre-computing of topic models will be decreased.

Since the binary range tree leaf node is very small and built in the common way, we can modify the construction of hierarchical tree for exploration. Of course, we can build the hierarchical tree in the classic way by sorting and splitting the dataset by only one dimension in the hierarchical tree, and ignore the subtree on other dimensions of the hierarchical tree to save the space. However, this structure cannot efficiently divide the data space into multi-dimensional ranges. Constructing the hierarchical tree by sorting and splitting the dataset by only one dimension is not effective for pre-computing of topic models, because all the leaf node range trees will be only constructed based on a

subset of a small range of the first dimension and large boundaries of other dimensions. The leaf node will not contain a small multi-dimensional range for both temporal and spatial dimensions, which is not effective for further range queries. Furthermore, even if we locate the canonical set in the hierarchical tree, the data items in the canonical set are still not highly covered by the multi-dimensional range query because of the large boundary of multiple dimensions. In this case, we need to think of another way to build the hierarchical tree, so that the leaf nodes can contain only small ranges of all the dimensions. A design to achieve this goal is that we still build only one hierarchical tree without any subtrees on other dimensions, so that we can save memory space. However, we alternately sort the dataset by different dimensions, and perform the division of the dataset for the next level child nodes. A simple example is shown in Fig. 3.3:

Figure 3.3: An example of 3-branching hierarchical tree for spatio-temporal dataset

As the hierarchical tree structure shown in Fig. 3.3, for a spatio-temporal dataset, we choose to sort the data items by their timestamp, sending location's latitude and longitude alternately for each level. After three times splitting, the subset of corpus will be limited within a three-dimensional range, which is much smaller than the boundary of the whole corpus. If we take the root node as level 0, then similar to the cube tree, when the branching factor is 2, the hierarchical range tree generates 8 child nodes at the $3n$ level for n is a non-negative integer. Each of the child node at level $3n$ takes $1/8$ data items of the ancestor node at level $3(n-1)$, whose data items take half of the range of the corpus in each dimension. However, by changing the branching factor, we can have more choices. We can divide the nodes into 27, 64, 125 and even more child nodes in the $3n$ level, which provides more nodes for topic model combining. Furthermore, the $3n+1$ and $3n+2$ level nodes can also be used for topic model combining, which provides more nodes for topic combining of possible range queries.

Based on the structure of hierarchical tree, the size of the leaf node range tree can also be controlled. Smaller leaf nodes can provide higher accuracy for the queried topic models, for the levels of the hierarchical tree are more, and the nodes are more divided. With smaller leaf nodes, the number of data items involved in topic combining can be smaller and more closed to the query result size, which is also an important issue. The hierarchical tree structure together with the leaf node size controlling can help to adjust the performance in runtime and memory space cost, as well as the quality of topic models.

By separating the range tree into hierarchical tree and leaf node sub range tree, the structure of Topic Ranger is constructed. An example of the Topic Ranger range tree structure is shown in Fig. 3.4:

Figure 3.4: An example of 2-branching hierarchical tree with sub range trees as leaf nodes

As the structure shown in Fig. 3.4, the hierarchical tree nodes briefly divide the dataset into subsets, and range trees are built to store the data items at leaf nodes. The hierarchical tree nodes only stores the topic models and the boundary (maximum and minimum values) of the data items for each dimension, while the leaf node sub range trees stores the data items and take charge of exact range queries.

After constructing the range tree, for each of the hierarchical tree node and leaf node, the topic models are pre-computed. The branching factor defines the number of child nodes, and the size of leaf nodes defines how deep we train the topic models for the tree of dataset.

3.2 Algorithms and Improvements

In this section, the range tree construction algorithm will be shown. The hierarchical tree nodes and the leaf node range tree will be implemented separately. Some modifications and their results will also be discussed in this section.

3.2.1 Range Tree Construction Algorithm

Although the description of the range tree structure seems complex, the algorithm used for constructing the range tree is quite simple and direct. In the description of the structure of Topic Ranger section, we separate the whole data structure into the hierarchical tree with only data ranges and pre-computed models, and leaf node sub range trees. The hierarchical tree node contains the boundary of data subset as well as LDA topic models, while the leaf node contains the data items and takes charge of query processing.

To make sure that the topic models of leaf nodes can be stored for quickly accessed, we want to make the leaf node range trees more efficient. Therefore, we extract the data record ID together with its geometric information as well as the timestamp as new points, whose values are the data record IDs, and the attributes are timestamps, latitudes and longitudes. We only use these points to build real range trees on leaf, and store the original data item list with text contents on another type of nodes. Each time we need to retrieve the exact text content of the data record, we use the data record ID to retrieve the content from the new type of nodes. Furthermore, we add one more type of nodes to store the topic model of sub range tree, to pass the range query to sub range tree, and to return the query result from sub range tree to its parent hierarchical nodes. We name the hierarchical node as "Range Tree Hierarchical Node (RH)", the leaf node sub range tree as "Range Tree (RT)", and the intermediate node placed at the leaf level between RH and RT as "Range Tree Full Node (RF)". Table 3.1 summarizes the data and functions provided by these three nodes:

Table 3.1: Data and Functions Providing of Three Node Types

Tree Node Type	Data Storage	Function
Range Tree Hierarchical Node (RH)	<ul style="list-style-type: none"> - Boundary of subset of data - Topic model of the subset of data - A list of child RH nodes or one RF node 	<ul style="list-style-type: none"> - Return the query result from its child nodes - Return the topic model of the whole subset of data for combining and sampling
Range Tree Full Node (RF)	<ul style="list-style-type: none"> - Boundary of subset of data - The root of range tree (RT) - Original data record list with text content 	<ul style="list-style-type: none"> - Return the query result from RT of the root it contains - Return the topic model of the whole subset of data for combining and sampling
Range Tree (RT)	<ul style="list-style-type: none"> - The point list of the data record ID and its time and geometric attributes - A real range tree to index the points 	<ul style="list-style-type: none"> - Return the IDs of the query result according to the range query

From Table 3.1, we know that the RH nodes occupy the main body of the hierarchical tree for quick access of topic models for topic combining, the RF nodes are placed at the leaf node level as intermediate nodes, and the RT nodes only take charge of range query. An example of these three types of nodes is shown in Fig. 3.5:

Figure 3.5: An example of 2-branching hierarchical tree with sub range trees with 3 types of nodes

From Fig. 3.5, the RH nodes can provide topic models when they are in the canonical set, while the RT nodes provides remaining queried data items for further sampling to adjust the final queried topic model, and provide the exact result of range queries. The RF nodes are only used to transfer the range query and query result between RH and RT. To build the three types of nodes, we use Algorithm 1:

Algorithm 1: Topic Ranger Range Tree Construction

input : Branching factor b , Leaf node size l , dimension d , Data item set S
output: Topic Ranger Range Tree for FSS Topic Combining:
 $\text{Range}_{\text{HierarchicalNode}}(S[i], b, l, d)$

- 1 create new empty RH node $root$;
- 2 initialize $root$'s b and l ;
- 3 update time and region boundary of $root$ with S ;
- 4 **if** $|S| < l$ **then**
 - 5 extract data record IDs and generate points with only IDs with time and location attributes from S ;
 - 6 construct RT to store and index points;
 - 7 store the root of RT to a new RF node;
 - 8 store S in RF node;
 - 9 $root.childNodeList.Add(RF)$;
- 10 **else**
 - 11 sort S by dimension d ;
 - 12 split S to b subsets;
 - 13 **if** $d \leq 1$ **then**
 - 14 $d = d + 1$;
 - 15 **else**
 - 16 $d = 0$;
 - 17 **end**
 - 18 **for** $inti = 0; i < b; i++$ **do**
 - 19 $RH = \text{new Range}_{\text{HierarchicalNode}}(S[i], b, l, d)$
 - 20 $root.childNodeList.Add(RH)$;
 - 21 **end**
- 22 **end**

The main idea of Algorithm 1 is to build a hierarchical tree indexing the leaf node range trees for Topic Ranger. We firstly construct a new RH node as the *root* of the hierarchical tree (line 1), and setup the branching factor b and leaf node size l for the hierarchical (line 2). When we preprocess the corpus, we find the the boundaries of the time and region of the data items. We assign these boundaries to *root*, which are taken as the time and region boundaries of the data items under *root*.

After setting up the parameters of *root*, we input the corpus S . If the size of S is smaller than the leaf node size l , then we recognize that we have reached the leaf node level of the hierarchical tree, and directly construct a new range tree under *root* (line 4-9). To save memory space, we do not build leaf node range tree directly with the timestamp, location and Tweet content. We use an ID of the Tweet instead of the content of the Tweet, and store the Tweet ID together with its timestamp and location as a point (line 5). We generate a list of points for the whole corpus S , then we use the list of the points to construct a 3D range tree RT (line 6). We store the root node of RT to a new *RF* node as well as the corpus S (line 7 and 8). Finally we attach the RF node to *root* as its child (line 9).

If the size of S is larger than the leaf node size l , then we need to further construct one more level for the hierarchical tree under *root* (line 10-21). Since we want to index the corpus using the three dimensions of timestamp, latitude and longitude alternatively, we firstly sort the corpus S by the input dimension d (line 11), and split the corpus S into b subsets equally (line 12). Since we set the branching factor of the hierarchical tree to b , *root* should have b child nodes. We adjust the current data indexing dimension d to the next dimension $d + 1$ or 0 (line 13-17). Then we input a subset of S , the branching factor b , the leaf node size l and the new dimension for sorting and sorting d for a new RH node construction (line 19). We repeat the new RH node construction for b times, and add these b new RH nodes to the child node list of *root*. In such a recursive method, Algorithm 1 can build a hierarchical tree from its root to the leaf nodes.

Here, we need to highlight one issue: Although the dataset S is passed down from *root* RH node to the leaf nodes, S is only stored in RF nodes, and indexed in RT range trees. The RH nodes contains a set of RH nodes or one RF node as its child node.

To make the hierarchical tree structure clearer, we have an example for the tree construction. The example is dumped into Appendix A.

3.2.2 FSS Topic Model Combining Algorithm

Based on the range tree construction, for each of the RH node, we retrieve the text content of the dataset from its child nodes, and perform topic model pre-computing on it. The pre-computed topic models are stored at each of the RH nodes. The structure of range tree and the pre-computed topic models are stored in a file named "rtree.big".

When a range query comes, Topic Ranger firstly search the canonical set of the hierarchical range tree. Here we set a threshold τ for the RH nodes. For each RH node in the canonical set, once more than τ of the RH node dataset falls in the queried range, we take the RH node into a heap h .

After examining all the RH nodes in the canonical set, we use the h heap to apply FSS algorithm for topic combining. We take out the topic models of the RH nodes from large size to small, and sample the data items in the small RH node with the topic model of large RH node as reference [15]. After sampling, we get a combined topic model for all these RH nodes in the canonical set. Then for those RH nodes with queried data items less than τ , we directly retrieve the exact queried data records from them, and perform Online-LDA on these data records with the combined topic model as reference. The final topic model is returned as the queried topic model.

An pseudo algorithm is shown in Algorithm 2:

Algorithm 2: Topic-Range Query Result Approximation

```
input : Topic-range query  $Q$ , Topic Ranger with indexed range tree  $TR$ ,  
        threshold  $\tau$   
output: Topic model  $M$   
1  $RH\_list = Topic\_Ranger.root.query(Q);$   
2 initialize heap  $h$ ;  
3 initialize data item list  $remaining$ ;  
4 for  $int\ i = 0; i < RH\_list.count; i++$  do  
5    $queried\_size = RH\_list[i].query(Q).Count;$   
6   if  $queried\_size/RH\_list[i] \geq \tau$  then  
7      $heap.push(RH\_list[i]);$   
8   else  
9      $remaining.Add(RH\_list[i].query(Q));$   
10  end  
11 end  
12  $model = heap.pop.model;$   
13 for  $inti = 0; i < heap.size; i++$  do  
14    $RH\ node = heap.pop;$   
15   update  $model$  by re-sampling  $node$  with the reference of  $model$ ;  
16 end  
17 adjust  $model$  by Online LDA training on  $remaining$ ;  
18 return  $model$ ;
```

Similar to the FSS algorithm, Algorithm 2 also use a heap h to store the tree nodes covered by the topic-range query Q , and we set a value τ as the threshold to check whether we want to add the covered tree node to the heap. We firstly gather all the hierarchical tree nodes covered by the topic-range query, and put these RH nodes into a list RH_list (line 1). Then we initialize an empty heap h and an empty data item list $remaining$ (line 2 and 3). Then we check through the list of the RH nodes covered by the query range. We count the number of data items $queried_size$ in the RH node that are falling in the query range (line 5), and calculate the ratio $queried_size/|RH_list[i]|$, where $|RH_list[i]|$ is the number of data items within the boundaries of the i th RH node in the RH_list . If the ratio is larger to equal to τ , we add the i th RH node in the RH_list to the heap h (line 7). If the ratio is less than τ , we directly add the queried data items

in the i th RH node to the *remaining* data item list (line 9).

After we complete the construction of heap h and data item list *remaining*, we pop out the largest RH node from the heap h (line 12). We take the topic model of the largest RH node as the *model* to be adjusted. Then we pop out the left RH nodes from h one by one in a decreasing order by their size, and perform re-sampling on the RH nodes with *model* as the reference (line 15). By summing up the word token counts of each topic assignment, *model* is combined with the topic models of other RH nodes. Finally, in line 17, we adjust *model* with the *remaining* data item list using the OLDA algorithm. The adjusted *model* is returned as the approximate topic model of the topic-range query Q (line 18).

3.2.3 Parallel Pre-Computing

Based on the range tree built with Algorithm 1, we need to perform LDA topic modeling to all the RH nodes. In the original implementation of FSS, the LDA training process is done sequentially. Here we use multiple threads for topic pre-computing. Each thread holds one RH node and does the Gibbs sampling on its dataset.

We tested the multi-threading version of topic model pre-computing on a 1-million-record dataset of Twitter data with branching factor 2 and leaf node size 500. Tree construction together with sequential topic training takes 9.7769 minutes, while tree construction together with multi-threading topic training takes 9.2032 minutes. For a 3-million-record dataset, tree construction with sequential topic training takes 99.4502 minutes, while the tree construction together with multi-threading topic training takes only 39.6841 minutes, which is a great enhancement.

3.2.4 Removal of Tree On Secondary Dimension

After we designed Algorithm 1, we tested the algorithm on a dataset with 1 million data records of Twitter data. Unfortunately, the memory space is very large because of the subtrees on other dimensions. We decide to remove these sub trees to save memory space. Although this will sacrifice the time cost in range query, it greatly decreases the storage of data. An in-memory range tree with subtrees on next dimensions takes up to 9.7 GB, while an in-memory range tree without subtrees take only 4.2 GB.

Although removing subtrees provides a great improvement in memory size of the data structure, the time cost of query search is sacrificed. For the training set range tree of an 1-million-record dataset of Twitter data, we set the branching factor to 2 and leaf node size to 500. For the same range query "30N, 90W" to "40N, 85W", "1/1/ 2015 06:00-10:00", the query time for range tree with subtrees is only 42.3358ms on average, while the range tree without subtrees takes up to 128.0073ms on average. Since FSS algorithm for topic combination also searches the canonical nodes and remaining data items, the FSS combining time cost also gets different values. The range with subtrees takes only 5209.2979 ms for topic combining, while the range tree without subtrees takes 6339.0292ms. The runtime cost is very large if we remove the subtrees. Therefore, in the experiment, we implement two versions of Topic Ranger for comparison. One version of Topic Ranger uses full range tree, and the other version uses range tree without subtrees on other dimensions.

3.3 Summary

In this chapter, we have described the Topic Ranger algorithm. We combine the FSS algorithm and the range tree together. Since the memory space cost of the range tree is huge, we use a hierarchical tree to split the dataset into small subsets, and index the subsets into the range trees. We name the range trees as RT, the node containing the

range of RT as RF node (Range Tree Full Node), and the upper hierarchical tree nodes as RH node (Range Tree Hierarchical Node). For each RH, it contains the range of its child RH node's dataset. After data indexing, we pre-compute the topic models for the datasets of all RH nodes. By checking the range of RH node's dataset, only the RT covered by the query will be used for combining the topic models. By combining the topic models with the FSS algorithm, we compute the approximation of the topic-range query result.

Chapter 4

Experiments and Result Analysis

4.1 Experiments

In Section 4.1, we explain the experiment conditions of the experiments. The hardware details, the dataset, the training set and testing set are stated. The data format and the result format are presented.

For the experiment of Topic Ranger with different parameters, we use Dell Windows 7 with Intel[®] Xeon[®] E5-2680 v2 @ 2.80 GHz with 20 cores with 64 GB RAM. The dataset we use is Twitter 2015 dataset located in the United States. We take a 1-million-record subset for experiment. A training set and a testing set are generated randomly following 5-fold cross-validation [71, 74].

For the experiment of comparison between Topic Ranger and Fast Set Sampling (FSS) with cube tree, we use Intel[®] Xeon[®] E7-8890 v3 @ 2.50 GHz with 4 processors and 72 cores with 512 GB RAM. We take a 25-million-record subset of Twitter 2015 dataset located in the United States for experiment. A training set and a testing set are generated randomly following 5-fold cross-validation.

For the tweets in the dataset, all the stop words have been removed, and the words have been stemmed. The format of one Tweet data record is: "yyyy-mm-dd hh:mm:ss (Tab) Word,word,word,...,word (Tab) Latitude (Tab) Longitude"

An example is: "2015-01-01 00:00:11 people,dozen,geo,phones 40,812268 -73.935039"

These Tweet records are stored in Range Tree Full (RF) nodes, and attached with a generated ID. The ID is the index of the Tweet record within the list of local RF node. In Range Tree (RT) node, the real range trees at the leaf nodes, the points have 3-dimensional vector as attributes, and the IDs of Tweets are the value of the points.

For example, the example Tweet is the data record with index 6 in the list of the local RF node, then the point is: "(635556672110000000, 40,812268, -73.935039):6"

Here, "635556672110000000" is the ticks of "2015-01-01 00:00:11".

In the pre-processing stage, the hierarchical tree and the leaf node range trees are constructed, and the topic models describing the tweets within the range of the tree nodes are computed. After training, the probabilistic distribution of the topic models as well as the points and the Tweet list of the tree node will be stored. The information of a hierarchical tree node stored into the file is:

- Leaf node size
- Branching factor
- Dimension for data sorting
- Dimension for data sorting in the child nodes
- Geographical boundaries of Tweets of the tree node
- Time boundaries of Tweets of the tree node
- Topic model describing the Tweets of the tree node
- Number of Tweets that can be retrieved from all the descendant nodes
- Child hierarchical nodes, or the RF node

The information of a RF node is also stored. The file includes the information below:

- Geographical boundaries of Tweets of the child range tree of RF node
- Time boundaries of Tweets of the child range tree of RF node
- Number of Tweets stored in RF node
- Content of Tweets
- Topic model describing the Tweets of the tree node

The structure of range trees under RF nodes are not stored, for the full range tree is difficult to store into the file. When the number of Tweets is small, the range trees can be constructed easily.

For each range query, the format of query contains the south-west point and north-east point of the queried region. The time range is input in the format "mm/dd/yyyy hh:mm:ss".

For example, a range query can be:

30,-90

40,-85

01/01/2015 06:00:00

01/01/2015 10:00:00

Then for every point falling in the query, their values are returned to RF nodes. If the parent Range Tree Hierarchical (RH) nodes need to add the text content of queried data records to remaining set, then the corresponding RF nodes return the Tweets according the the values of points as indices of the Tweet list.

After Topic Ranger combines the topic models, the perplexity of the result topic model is computed with the queried result from testing set and "left-to-right algorithm" [45, 46]. Meanwhile, an Latent Dirichlet Allocation (LDA) model and an Online-LDA (OLDA) modal are also trained for the queried result. The perplexities are also computed

for OLDA and LDA models. Beside the perplexity, we also measure the Bhattacharyya distance from Topic Ranger model to LDA model, and OLDA model to LDA model [47]. The output of one range query test is recorded as the format below:

"Query ID, South, West, North, East, Start Time, End Time, Training Set Size, Testing Set Size, Query Time Train, Query Time Test, FSS Time, OLDA Time, LDA Time, FSS Perplexity, OLDA Perplexity, LDA Perplexity, B distance FSS to LDA, B distance OLDA to LDA "

We mainly focus on and record the runtime cost, perplexity and Bhattacharyya distance. For all the experiments, the stopping condition of the Gibbs Sampling is set to "maximum iterations not exceeding 100" as well as "topic model convergence less than 0.0001". The sampling iteration terminates when any of the two stopping conditions become true. Each topic-range query is repeated for 30 times and we compare the query result with box plots.

4.2 Comparisons of Various Incarnations of Topic Ranger

In Section 4.2, we compare Topic Rangers with different parameters. The parameters to be adjusted are the branching factor and leaf node size. The comparisons are conducted in three aspects: data storage size, runtime cost and quality of approximate topic models. In the comparison of the quality of approximate topic models, the range queries are changed in latitude, longitude and time to observe the differences. To adjust the size of the topic-range query, we take " δ -latitude" as the region query ranging from South to North in latitude, and " δ -longitude" as the region query ranging from West to East in

longitude. Both of them are ranging from 5, 10, to 15. For the time span of the topic-range query, the length of queried time range changes from 4 hours to 6 and 8 hours. With a fixed location and timestamp of the region range query, we gradually enlarge the size of the query in all the dimensions and apply them to the implementations of Topic Ranger.

4.2.1 Parameter: Branching Factor

4.2.1.1 Storage Size of Data

Since we have a wide range of the parameter values to choose, and the experiment can take a very long time for Topic Ranger with different parameters, we decide to use 1-million-record dataset for comparison of Topic Ranger with different parameters. For 1-million-record dataset, we generate the training and testing set. Based on the same training and testing set, we set the leaf node size to 500, and gradually increase the branching factor from 2 to 6. The size of the range tree storage file "rtree.big" for different branching factors is ranging from 0.74GB to 2.1 GB. The size of "rtree.big" is not very large, which is because that the file only stores the tweets within each leaf node of the hierarchical tree. When the program reads the file to construct the range tree, the in-memory size differs according to the parameter setting of Topic Ranger. Topic Ranger using full range tree has a much larger in-memory size than Topic Ranger using range tree without subtrees on other dimensions. We refer the Topic Ranger using full range tree as "RTS", and Topic Ranger using range tree without subtrees on other dimensions as "RT". The comparison is listed in Table 4.1:

Table 4.1: Comparison of Topic Ranger Range Tree File Size and In-Memory Size

Branching Factor	File Size (GB)	In-Memory Size of RT (GB)	In-Memory Size of RTS (GB)	Ratio of In-Memory Space Size of RT over RTS
2	2.10	4.2	9.7	43.2990%
3	1.42	3.0	8.5	35.2941%
4	1.07	2.4	7.0	34.2857%
5	0.922	2.2	7.1	30.9859%
6	0.927	2.1	5.9	35.5932%

From Table 4.1, we can also visualize the changes of data storage size in Fig 4.1:

Comparison of Memory Size

Figure 4.1: Comparison of Topic Ranger range tree file size

From Fig. 4.1, we can find that in general, the more branches the range trees have, the smaller size the data storage file has. This is because the number of leaf nodes and topic models are decreased when the branching factor and leaf node size increases. For Topic

Ranger with any branching factor, using range tree without subtrees on other dimensions can save much in-memory space than using full range tree. From the ratio of in-memory space size of RT over RTS shown in Table 4.1, RT only takes nearly 1/3 of the memory space of RTS.

4.2.1.2 Runtime Cost

Runtime Cost for Topic Ranger Preprocessing

For the preprocessing stage of Topic Ranger, we include both the range tree construction and topic model pre-computing. Since in the preprocessing stage, we do not need to apply topic-range queries, we can use range tree without subtrees on other dimensions, which uses smaller memory space and constructs less trees. The time for file writing is also counted in. The comparison of runtime costs for preprocessing of 1-million-record training dataset are listed below in Table 4.2:

Table 4.2: Comparison of Topic Ranger Preprocessing Time Cost

Branching Factor	Time Cost (minute)
2	9.2032
3	6.0802
4	4.6643
5	4.0575
6	4.1446

From Table 4.2, similar to the file size of "rtree.big", the preprocessing time also decreases with increasing the branching factor and leaf node size in general.

Runtime Cost for Re-Constructing Range Tree with Preprocessed File

RTS and RT construct different structures of range trees from the same dataset. Therefore, it is important to measure the time used to re-construct the hierarchical tree

and range tree from the file for RTS and RT. The details of re-construction runtime are shown below in Table 4.3:

Table 4.3: Comparison of Re-Constructing Range Tree from Preprocessed File

Branching Factor	RT Runtime Cost for Re-Construction (second)	RTS Runtime Cost for Re-Construction (second)
2	80.627	120.684
3	55.871	88.789
4	15.77	42.971
5	14.021	44.093
6	14.237	37.031

From Table 4.3, both RT and RTS get a lower runtime cost in re-constructing the tree structure from the file while the branching factor is increasing. RTS always costs a much higher runtime cost than RT, for it needs to construct more subtrees on other dimensions.

Runtime Cost for Querying Data Records

To examine the runtime cost for query and topic combining, we decide to generate range queries with different sizes in latitude, longitude and time period. The list of topic-range queries are shown in Appendix B. Since RTS always has a much lower runtime cost than RT, we separate the runtime costs of the two implementations into two groups for comparison.

For the topic-range queries, we divide them into three groups by their time period of query. For each group, we visualize the results by 3D scattering. The x axis presents the size of latitude, while the y axis presents the size of longitude. The z axis present the runtime costs. The query time costs of RT and RTS are shown below in Fig. 4.2 and Fig. 4.3:

● Branching factor 2
● 3
● 4
● 5
● 6

(a) Time Period: 4 Hours

● Branching factor 2
● 3
● 4
● 5
● 6

(b) Time Period: 6 Hours

● Branching factor 2
● 3
● 4
● 5
● 6

(c) Time Period: 8 Hours

Figure 4.2: Comparison of RT Query Time Cost

Figure 4.3: Comparison of RTS Query Time Cost

From Fig. 4.2 and 4.3, we can get some information about the branching factor: For RT, the implementation with branching factor 2 and 3 performs the best for the topic-range queries. Similarly, RTS with branching factor 2 and 3 also has the lowest query time cost, except the topic-range queries (δ -latitude = 5, δ -longitude = 15, 8 hours) and (δ -latitude = 15, δ -longitude = 15, 8 hours), for which RTS with branching factor 4 performs the best.

Runtime Cost for Topic Model Approximation

Besides query time for training and test dataset, we also test the runtime cost of FSS topic model combining. Similar to the measurement of query time cost, we show the details of the topic model approximation runtime cost of RT and RTS in Fig. 4.4 and 4.5:

(a) Time Period: 4 Hours

(b) Time Period: 6 Hours

(c) Time Period: 8 Hours

Figure 4.4: Comparison of RT Topic Model Approximation Runtime Cost

Figure 4.5: Comparison of RTS Topic Model Approximation Runtime Cost

From Fig. 4.4 and 4.5, we find that FSS runtime cost comparison does not follow with query time cost comparison. Although RT with branching factor 2 and 3 performs the best for many topic-range queries in the aspect of query time cost, RT with branching factor 6 performs better than branching factor 2 and 3 for (δ -latitude = 5, δ -longitude = 10, 6 hours), (δ -latitude = 5, δ -longitude = 15, 8 hours) and (δ -latitude = 10, δ -longitude = 10, 8 hours). For RTS, RTS with branching factor 2 performs the best for most of the topic-range queries. For topic-range queries with time period length of 6 hours and 8 hours, RTS with branching factor 2 requires the lowest runtime cost for 16 queries, except (δ -latitude = 15, δ -longitude = 10, 6 hours) and (δ -latitude = 15, δ -longitude = 10, 8 hours), for which RTS with branching factor 4 performs the best in runtime cost. The runtime cost of topic model approximation might be defined by multiple latent factors.

4.2.1.3 Topic Ranger Result Quality

Based on the comparison of runtime cost, we want to compare the quality of topic models. For the topic model evaluation, we use two metrics: the perplexity and the Bhattacharyya distance.

Perplexity

Still for the same parameter settings of the range queries, we apply experiment and generate average value of perplexity based on the content of query result from testing dataset. Since the subtrees of the range tree only affect the runtime cost, RTS and RT get very close approximate topic model. The perplexity of Topic Ranger with different branching factors are visualized in Fig. 4.6:

Figure 4.6: Comparison of Topic Ranger Topic Model Perplexity

From Fig. 4.6, for all the three groups of topic-range queries, branching factor 6 performs the best for the small δ -longitude topic-range queries. When the value of δ -longitude goes up, branching factor 5 performs the best among all the implementation. For all the queries, the best 2 implementations are always using branching factor 5 and branching factor 6. However, from the results, no trend can be identified according to the branching factor.

Bhattacharyya Distance

Similar to the perplexity, the Bhattacharyya distance is also not affected by the implementation of range tree. Here, we measure the Bhattacharyya distance from Topic Ranger topic model to the LDA topic model, for the LDA algorithm is used to directly calculate the topic model of the query result dataset. If we take LDA topic model as the ground truth of topic-range query result, then the shorter the Bhattacharyya distance is, the more accurate the Topic Ranger topic model is. The details are shown in Fig. 4.7:

Figure 4.7: Comparison of the Bhattacharyya Distance from the Topic Ranger Topic Model to the LDA Topic Model

Unfortunately, from Fig. 4.7, the comparison result of Bhattacharyya distance is random in view of branching factor. The Bhattacharyya distance might be affected by multiple latent parameters. Branching factor is not the only parameter that affects the Bhattacharyya distance.

4.2.2 Parameter: Leaf Node Size

4.2.2.1 Storage Size of Data

Similar to the experiment of branching factor, we adjust the leaf node size with a fixed branching factor. Here, since branching factor 2 is easy to be used for the comparison with the cube tree, we fix the branching factor to 2, and set the leaf node size to 500, 2000, 5000 and 10000. The memory space comparison is shown below in Table 4.4:

Table 4.4: Comparison of Topic Ranger Range Tree File Size and In-Memory Size

Leaf Node Size	File Size (GB)	In-Memory Size of RT (GB)	In-Memory Size of RTS (GB)	Ratio of In-Memroy Space Size of RT over RTS
500	2.10	4.2	9.7	43.2990%
2000	1.74	3.5	11.1	31.5315%
5000	1.56	3.2	12.0	26.6667%
10000	1.39	2.9	12.8	22.6563%

The visualization of Table 4.4 is shown below in Fig 4.8:

Figure 4.8: Comparison of Topic Ranger Range Tree File Size

From Table 4.4 and Fig. 4.8, the leaf node affects the in-memory space of Topic Ranger in an interesting way: By increasing the leaf node size, the memory space of RT decreases, while the memory space of RTS increases. This is mainly caused by the subtrees on the other dimensions. Even the total sizes of datasets are the same, the total size of a smaller number of larger full range trees is still larger than a larger number of full smaller range trees.

4.2.2.2 Runtime Cost

Runtime Cost for Topic Ranger Preprocessing

Similar to the experiment of branching factor, we only store one file for both RT and RTS. Therefore, we still use RT to train the pre-computed topic models, for RT does not cost more time and space to generate subtrees. The details of the runtime cost for Topic Ranger is shown below in Table 4.5:

Table 4.5: Comparison of Topic Ranger Preprocessing Time Cost

Leaf Node Size	Time Cost (minute)
500	9.2032
2000	8.1473
5000	7.8607
10000	6.7382

From Table 4.5, the larger the leaf node size is, the lower the pre-processing time cost is. This is because larger leaf node size makes the depth of the hierarchical tree smaller. Therefore, fewer topic models will be pre-computed if the leaf node size is larger.

Runtime Cost for Re-Constructing Range Tree with Preprocessed File

For the experiment for leaf node size comparison, we also measure the re-construction runtime which is used for both RT and RTS. The details of the runtime cost are shown in Table 4.6:

Table 4.6: Comparison of Re-Constructing Range Tree from Preprocessed File

Leaf Node Size	RT Runtime Cost for Re-Construction (second)	RTS Runtime Cost for Re-Construction (second)
500	80.627	120.684
2000	58.64	240.308
5000	59.194	196.123
10000	59.05	284.523

From Table 4.6, different from the experiment of branching factor comparison, the re-construction time cost is not monotonic increasing or decreasing. Adjusting the leaf node size makes the runtime cost of re-construction more difficult to be predicted. This is because that with the larger leaf node size, the larger full range tree takes more time to construct, while the number of the full range trees are smaller. The two factors affect the total runtime cost together.

Runtime Cost for Querying Data Records

For the query runtime cost, topic model approximation runtime cost, perplexity and Bhattacharyya distance measurement, we use the same set of topic-range queries as we used in the experiment of branching factor. Here, we still compare the query time cost of RT and RTS separately, for the query time cost of the two implementations are very different. The query time cost results are shown in Fig. 4.9 and 4.10:

Figure 4.9: Comparison of RT Query Time Cost

Figure 4.10: Comparison of RTS Query Time Cost

From Fig. 4.9 and 4.10, for RT and RTS, the implementation with the largest leaf node size 10000 takes the smallest query time cost. Only for the topic-range query (δ -latitude = 15, δ -longitude = 15, 8 hours), RT with leaf node size 5000 performs the best.

For the topic-range query (δ -latitude = 15, δ -longitude = 15, 6 hours), RTS with leaf node size 5000 performs the best. Topic Ranger with larger leaf node size takes smaller runtime cost for query, for the number of the nodes involved in the query is smaller.

Runtime Cost for Topic Model Approximation

For the same set of topic-range queries, we measure the topic model approximation runtime cost using the FSS algorithm. The result is shown below in Fig. 4.11 and 4.12:

Figure 4.11: Comparison of RT Topic Model Approximation Runtime Cost

Figure 4.12: Comparison of RTS Topic Model Approximation Runtime Cost

From 4.11 and 4.12, for RT, the largest leaf node size 10000 performs the best for most of the topic-range queries with 4-hour time period. For topic-range queries with 6-hour time period, RT with leaf node size 10000 performs the best for smaller region size, while RT with leaf node size 500 performs the best for larger regions size. When the time period of the topic-range query has been enlarged to 8 hours, RT with leaf node 500 performs the best for most of the topic-range queries. Compared to RT, the comparison result of RTS is more consistent. RTS with leaf node size 500 performs the best in all the topic-range queries, except the query (δ -latitude = 15, δ -longitude = 5, 4 hours), for which RT with leaf node size 10000 performs the best in runtime cost.

4.2.2.3 Topic Ranger Result Quality

Perplexity

Similar to the experiment of branching factor, for the perplexity of topic model, RT and RTS generates very close topic models. We show the comparison results in Fig. 4.13:

Figure 4.13: Comparison of Topic Ranger Topic Model Perplexity

From Fig. 4.13, when δ -longitude is 5, the three leaf node sizes provide close perplexities to each other. While δ -longitude is gradually increasing, the differences between the perplexities increase. Topic Ranger with leaf node size 2000 performs the best in

perplexity for topic-range queries with larger δ -longitude. Besides, Topic Ranger with leaf node size 500 can have the highest perplexity than other implementations.

Bhattacharyya Distance

For the leaf node size comparison, we also want to compare the Bhattacharyya distance. The comparison results are shown in Fig. 4.14:

Figure 4.14: Comparison of the Bhattacharyya Distance from the Topic Ranger Topic Model to the LDA Topic Model

Unfortunately, from Fig. 4.14, Bhattacharyya distance is still random when we adjust the leaf node size. There should be more latent factors that affect the Bhattacharyya distance.

4.2.3 Conclusion of Topic Ranger with Different Parameters

From the result of experiments, we find that the parameter setting of Topic Ranger is more likely affecting the pre-processing time cost, data storage size and query time cost. Both the memory space and pre-processing time are monotonic with the branching factor and leaf node size. However, the effect of the branching factor over query time cost and the FSS algorithm runtime cost is not stable, while the leaf node size affects the runtime costs more consistently.

In view of perplexity, Topic Ranger with branching factor 5 and 6 provides the lowest perplexity. Topic Ranger with leaf node size 2000 also provides the lowest perplexity for most of the queries. However, the Bhattacharyya distance is not as consistent as perplexity. The Bhattacharyya distance is random for both experiment sets. There might be more latent factors affecting the Bhattacharyya distance.

Although we have several findings for the experiment, the findings might not be true for all the datasets and all the topic-range queries. This is because that the topic-range queries are not generated by random, and the dataset is very small. More generalized experiments are still needed to further study the parameters of Topic Ranger.

4.3 Comparison Between Topic Ranger and FSS with Cube Tree

In Section 4.3, we want to compare Topic Ranger with FSS with cube tree. Topic Ranger with branching factor 2 is dividing the data cube into halves from each dimension, which is similar to the cube tree. Therefore, to make a standard for both implementations to

compare, we still use "branching factor 2" as the parameter setting for Topic Ranger, and compare the results to FSS algorithm with cube tree.

Unlike Topic Ranger, the implementation of FSS with cube tree does not use the parameter "branching factor" or "leaf node size". It takes "accuracy threshold" as a parameter, which defines the levels of cube tree nodes to re-train the whole dataset within the node. It also has a parameter "memory threshold", which defines the maximum number of topic models the cube tree should have. The number of nodes constructed by the cube tree is always smaller than the memory threshold. Since the parameter of FSS with cube tree is so different from Topic Ranger, we decide to set the accuracy threshold to 1, so that both FSS with cube tree and Topic Ranger trains a pre-computing topic model for each level of the tree. Furthermore, we count the number of pre-computed topic models in Topic Ranger, and set the memory threshold of FSS algorithm to the same value, so that FSS with cube tree has the number of pre-computed tree nodes near to Topic Ranger.

In Section 4.3, we firstly compare the memory space cost FSS with cube tree and Topic Ranger. Then we compare the query time cost of FSS with cub tree and Topic Ranger, as well as the runtime cost of the FSS algorithm and the OLDA algorithm. We also compare their in the aspect of the perplexity and Bhattacharyya distance to the direct computed LDA topic model.

For the comparison between Topic Ranger and FSS with cube tree, we use a much larger dataset than the dataset used in Section 4.2, so that we are able to get a more generalized conclusion in the comparison of FSS with different data indexing structures. For a 25-million-record dataset, we generate a training dataset and a testing dataset by using 5 cross-validation. Since the dataset is much larger than 1-million-record dataset, we raise the leaf node size to 2000 to decrease the number of tree nodes for pre-computation. With branching factor 2 and leaf node size 2000, for the training dataset, Topic Ranger

finally trains 32767 topic models for 32767 nodes. Therefore, we set the memory threshold of FSS with cube tree to 32770, so that the cube tree will construct child nodes fewer than 32770, and very close to 32767. Finally the cube tree constructs and pre-computes 32766 topic models for 32766 nodes, which is very close to Topic Ranger.

Similar to the topic-range queries in Section 4.2, we adjust the size of the query range, and apply these topic-range queries to both FSS with cube tree and Topic Ranger. However, to make the experiment more general, we randomly generate the southwest corner of the query region and the start time of the query time span. Then we use them to generate the northeast corner and end time with a fixed region size and time length. We apply multiple randomly generated topic-range queries to the training dataset, and remove those topic-range queries without query results. We have finally gathered 15 queries from the random queries for further testing and analysis. We order the topic-range queries by their size of result dataset from small to large. The full details of all the randomly generated queries are listed in Appendix C.

4.3.1 Storage Size of Data

Similar to the comparison in Section 4.2, we compare both the file size and in-memory space size of FSS with cube tree, Topic Ranger using range tree without subtrees on other dimensions, and Topic Ranger using full range tree. For convenience, we refer FSS with cube tree as "CT", Topic Ranger using range tree without subtrees as "RT", and Topic Ranger using full range tree as "RTS". The file size and in-memory space size are shown below in Table 4.7:

Table 4.7: Comparison in File Size and In-Memory Size of Different FSS Implementations

Type of Implementation	File Size (GB)	In-Memory Size (GB)
CT	28.6	58.8
RT	66.1	127.9
RTS	66.1	311.2

From Table 4.7, RTS takes the most in-memory space, while CT takes the least in-memory space. Since we use the same format to store the information of hierarchical tree and topic model for RT and RTS, they have the same size of data storage file, which is larger than the file generated by CT.

4.3.2 Runtime Cost

4.3.2.1 Runtime Cost in Queries

Since the comparison results are very consistent for all the topic queries, we only show one figure as Fig. 4.15, and the other figures are shown in Appendix D and Appendix E:

Figure 4.15: Comparison of Query Runtime Cost (Query 1): Upper: Comparison of CT, RTS and RT. Lower: Comparison of CT and RTS

From Fig. 4.15, we find that FSS with cube tree always has a much better query time cost than Topic Ranger. The main reason is that Topic Ranger needs to combine the query results from child nodes. The query runtime cost of RT is much higher than both RTS and CT, this is because subtrees can speedup querying process.

4.3.2.2 Topic-Range Query Approximation Runtime Cost

Although RT and RTS get much higher query time cost than CT, RTS can get a lower topic combination runtime cost than CT for some range queries. For convenience, we refer the runtime cost of CT, RTS and RT as $T(\text{CT})$, $T(\text{RTS})$ and $T(\text{RT})$. For the 15 queries we tested, there are four possible results: (1) $T(\text{CT}) < T(\text{RTS}) < T(\text{RT})$; (2) $T(\text{RTS}) < T(\text{CT}) < T(\text{RT})$; (3) $T(\text{RTS}) < T(\text{RT}) < T(\text{CT})$; (4) $T(\text{CT}) \approx T(\text{RTS}) < T(\text{RT})$. Here, for the fourth result type, we put the query result into this type when the box plot of $T(\text{RTS})$ is fully covered by $T(\text{CT})$ or vice versa. All the figures are inside Appendix F. The examples of topic combination runtime cost are shown in Fig. 4.16:

Figure 4.16: Comparison of FSS Runtime Cost

From Fig. 4.16, the runtime cost of topic-range query approximation of CT, RTS and RT varies for different queries. The distribution of the four types of runtime cost comparisons of topic-range query approximation within 15 queries is shown in Table 4.8:

Table 4.8: Comparison in runtime cost of topic-range query result approximation

Type of Comparison Result	Number of Queries	Indices of Queries
(1) $T(\text{CT}) < T(\text{RTS}) < T(\text{RT})$	7	1, 4, 5, 8, 9, 10, 14
(2) $T(\text{RTS}) < T(\text{CT}) < T(\text{RT})$	3	2, 7, 12
(3) $T(\text{RTS}) < T(\text{RT}) < T(\text{CT})$	2	13, 15
(4) $T(\text{CT}) \approx T(\text{RTS}) < T(\text{RT}) < T(\text{CT})$	3	3, 6, 11

From Table 4.8, CT has a lower runtime cost than RTS and RT for 7 topic-range queries. These 7 queries covers the query result size from small to large. However, the other three types of comparison results also cover different size of query result. The different types of results are mainly caused by the different data structure. For different topic-range queries, Topic Ranger and FSS with cube tree put different numbers of nodes into the heap for further fast sampling, which makes the runtime cost varies.

4.3.2.3 Comparison of OLDA Runtime Cost

Although we mainly study and explore the idea of the FSS algorithm, we also want to test the runtime cost of the OLDA algorithm. The comparison results of OLDA runtime cost of the three implementations can be grouped into three types: (1) $T(\text{CT}) < T(\text{RTS}) < T(\text{RT})$; (2) $T(\text{RTS}) \leq T(\text{RT}) < T(\text{CT})$; (3) $T(\text{CT}) \approx T(\text{RTS}) < T(\text{RT})$. We put all the figures in Appendix G, and the examples of the three types of runtime cost comparisons are shown in Fig. 4.17:

Figure 4.17: Comparison of OLDA Runtime Cost

The distribution of the three types of OLDA runtime cost comparisons is shown in Table 4.9:

Table 4.9: Comparison in runtime cost of topic-range query result approximation

Type of Comparison Result	Number of Queries	Indices of Queries
(1) $T(CT) < T(RTS) < T(RT)$	4	1, 8, 9, 10
(2) $T(RTS) \leq T(RT) < T(CT)$	7	6, 7, 11, 12, 13, 14, 15
(3) $T(CT) \approx T(RTS) < T(RT)$	4	2, 3, 4, 5

From Table 4.9, CT has a lower runtime cost than RTS and RT in OLDA for only 4 queries, while RTS costs a lower OLDA runtime cost than CT for 7 queries. RTS has a better performance in OLDA runtime cost than CT from Query 11 to Query 15, which are all queries with large size of result.

4.3.2.4 Comparison of Runtime Cost of All Algorithms

For all the algorithms used for answering the topic-range query, we want to compare their runtime cost and check whether the FSS algorithm helps to save time. We use the notation $T(\text{CT})$, $T(\text{RTS})$ and $T(\text{RT})$ to refer the runtime cost of FSS with cube tree, FSS with full range tree and FSS using range tree without subtrees on other dimensions. Similarly, $T(\text{CT OLDA})$, $T(\text{RTS OLDA})$ and $T(\text{RT OLDA})$ refer to the runtime cost of OLDA with corresponding data indexing structures, while $T(\text{CT LDA})$, $T(\text{RTS LDA})$ and $T(\text{RT LDA})$ refer to the runtime cost of LDA with corresponding data indexing structures. We generate figures to visualize the runtime cost of all the three algorithms. There are generally four types of the comparison results: (1) $T(\text{RT})$ and $T(\text{RT OLDA})$ takes the highest runtime cost; (2) $T(\text{CT LDA})$, $T(\text{RTS LDA})$ and $T(\text{RT LDA})$ takes the highest runtime cost; (3) $T(\text{CT OLDA})$ takes the highest runtime cost and (4) The OLDA and LDA runtime costs are higher, as well as $T(\text{RT})$ and/or $T(\text{CT OLDA})$. The figures visualizing the comparison results are put to Appendix H. The visualization examples are shown in Fig. 4.18:

(a) $T(RT)$ and $T(RT\ OLDA)$ takes the highest runtime cost

(b) $T(CT\ LDA)$, $T(RTS\ LDA)$ and $T(RT\ LDA)$ takes the highest runtime cost

(c) $T(CT\ OLDA)$ takes the highest runtime cost

(d) The OLDA and LDA runtime costs are higher, as well as $T(RT)$ and/or $T(CT\ OLDA)$

Figure 4.18: Comparison of Runtime Cost of All Algorithms

From Fig. 4.18, we can find that the FSS algorithm takes lower runtime cost than OLDA and LDA. For the four types of comparison results shown in Fig. 4.18, the distribution of the queries for these types of comparison results is shown in Table 4.10:

Table 4.10: Comparison in Runtime Cost of All Algorithms

Type of Comparison Result	Number of Queries	Indices of Queries
(1) $T(\text{RT})$ and $T(\text{RT OLDA})$ takes the highest runtime cost	4	1, 2, 3, 4
(2) $T(\text{CT LDA})$, $T(\text{RTS LDA})$ and $T(\text{RT LDA})$ takes the highest runtime cost	2	9, 10
(3) $T(\text{CT OLDA})$ takes the highest runtime cost	7	6, 7, 11, 12, 13, 14, 15
(4) The OLDA and LDA runtime costs are higher, as well as $T(\text{RT})$ and/or $T(\text{CT OLDA})$	2	5, 8

From Table 4.10, we can find that more queries with small result size are grouped to the first type, while the third type includes more queries with large result dataset size.

4.3.3 Topic Model Quality

4.3.3.1 Perplexity

Although the FSS algorithm provides a lower runtime cost by answering the topic-range query with approximation, we need to compare the quality of the approximate topic models. Perplexity is an unsupervised method to compare the topic model quality. For the perplexity of topic models, we compare all the algorithms together. We refer the perplexity of Topic Ranger with CT, RTS and RT as $P(\text{CT})$, $P(\text{RTS})$ and $P(\text{RT})$. We also use $P(\text{CT OLDA})$, $P(\text{RTS OLDA})$ and $P(\text{RT OLDA})$ to represent the perplexity of OLDA with the three data indexing structures, and $P(\text{CT LDA})$, $P(\text{RTS LDA})$ and $P(\text{RT LDA})$ as the perplexity of LDA with the three data indexing structures.

From the comparison, the LDA topic models are very close to each other, no matter what data indexing structure we use. This is because the LDA algorithm only uses the whole query result dataset to compute topic model, which is not related to the data indexing structure. For the other two algorithms, in view of data indexing structure

comparison, we find that there are three types of comparison results. These comparison results are: (1) $P(\text{CT}) < P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) < P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$; (2) $P(\text{CT}) > P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) > P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$; (3) $P(\text{CT}) > P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) < P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$. Beside these three types, for very small size of query result, the distribution of the perplexity of the algorithms and data indexing structures can have very large variance. All the figures comparing the perplexity are put to Appendix I. The perplexity comparison is shown in Fig. 4.19:

(a) $P(\text{CT}) < P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) < P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$

(b) $P(\text{CT}) > P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) > P(\text{RTS OLDA})$

(c) $P(\text{CT}) > P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) < P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$

(d) Very small query result size

Figure 4.19: Comparison of Topic Model Perplexity of All Algorithms

For the perplexity of topic model, the comparison result distribution over the 15 queries is more extreme. The details is shown in Table 4.11:

Table 4.11: Comparison in Topic Model Perplexity of All Algorithms - In View of Data Indexing Structures

Type of Comparison Result	Number of Queries	Indices of Queries
(1) $P(\text{CT}) < P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) < P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$	2	8, 12
(2) $P(\text{CT}) > P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) > P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$	11	2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 15
(3) $P(\text{CT}) > P(\text{RTS}) \approx P(\text{RT})$ and $P(\text{CT OLDA}) < P(\text{RTS OLDA}) \approx P(\text{RT OLDA})$	1	14
Very small query result size	1	1

From Fig. 4.19 and Table 4.11, RTS and RT are more likely to compute approximate topic model with a lower perplexity than CT. RTS OLDA and RT OLDA are also computing topic models with lower perplexity than CT OLDA. When we use the topic model as the reference to generate a dataset, the lower the perplexity of topic model is, the more the generated dataset is close to the range query result dataset. Therefore, the range tree is more likely to provide a better topic model than the cube tree.

In terms of the algorithms, there are also different types of the comparison results. For convenience, we name the perplexity of the three algorithms' topic models as $P(\text{FSS})$, $P(\text{OLDA})$ and $P(\text{LDA})$. There are mainly three types of the comparison results: (1) $P(\text{FSS}) < P(\text{OLDA}) < P(\text{LDA})$; (2) $P(\text{OLDA}) < P(\text{FSS}) < P(\text{LDA})$; (3) $P(\text{FSS}) \approx P(\text{OLDA}) < P(\text{LDA})$. The perplexities of the topic models with very small query result size are very random, so Query 1 is not in any of the three types. The details of distribution of the three types are shown below in Table 4.12:

Table 4.12: Comparison in Topic Model Perplexity of All Algorithms - In View of Algorithms

Type of Comparison Result	Number of Queries	Indices of Queries
(1) $P(\text{FSS}) < P(\text{OLDA}) < P(\text{LDA})$	8	2, 3, 4, 5, 6, 11, 13, 15
(2) $P(\text{OLDA}) < P(\text{FSS}) < P(\text{LDA})$	2	7, 14
(3) $P(\text{FSS}) \approx P(\text{OLDA}) < P(\text{LDA})$	4	8, 9, 10, 12
Very small query result size	1	1

From Table 4.12, the FSS algorithm computes topic models with the lowest perplexity for 8 queries, while the OLDA algorithm computes topic models with the lowest perplexity for 2 queries. For all the queries, the LDA algorithm always computes topic models with the highest perplexity.

4.3.3.2 Bhattacharyya Distance

Although perplexity is taken as a useful unsupervised evaluation metric for topic model, we also want to measure how close the FSS/OLDA trained topic model is from LDA topic model. LDA topic model is the topic model directly computed from the range query result, which means that it can be taken as the ground truth of the topic-range query. Still, we use $B(\text{CT})$, $B(\text{RTS})$ and $B(\text{RT})$ to refer the Bhattacharyya distance to LDA topic models of FSS with cube tree, FSS with full range tree and FSS using range tree without subtrees on other dimension. Similarly, we use $B(\text{CT OLDA})$, $B(\text{RTS OLDA})$ and $B(\text{RT OLDA})$ for the OLDA algorithm with different data indexing structures. For the OLDA algorithm, the Bhattacharyya distances generated by implementations using different data indexing structures is not easy to be grouped. The box plots are highly covered with each other. Beside the other figures in Appendix J, we select some examples of the comparisons. These examples are shown in Fig. 4.20:

Figure 4.20: Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models

Since grouping the comparison results of Bhattacharyya distance of OLDA with different data indexing structures is difficult, we mainly focus on comparing other aspects of the Bhattacharyya distance result. For the FSS algorithm, in view of data indexing structure, the Bhattacharyya distance comparison results can be grouped into three types: (1) $B(CT) < B(RTS) \approx B(RT)$; (2) $B(CT) > B(RTS) \approx B(RT)$; (3) No group. We put all the figures to Appendix K. The examples are shown in Fig. 4.21:

Figure 4.21: Comparison of FSS Runtime Cost

From Fig. 4.21, OLDA topic models have shorter Bhattacharyya distances than FSS topic models to LDA, which means that OLDA generates a closer topic model to LDA for range query. The data indexing structure can affect the Bhattacharyya distance for different range queries. The distribution of the Bhattacharyya distance comparison results is shown below in Table 4.13:

Table 4.13: Comparison in Bhattacharyya Distance

Type of Comparison Result	Number of Queries	Indices of Queries
(1) $B(CT) < B(RTS) \approx B(RT)$	7	3, 4, 5, 8, 9, 10, 14
(2) $B(CT) > B(RTS) \approx B(RT)$	6	6, 7, 11, 12, 13, 15
No group	2	1, 2

From Table 4.13, the comparison results of Bhattacharyya distance are random. It is difficult to generalize the features of the topic-range queries in each comparison result type. We can only summarize that the OLDA algorithm is more likely to generate topic models with lower Bhattacharyya distance to LDA topic models than the FSS algorithm, for the computing progress of OLDA is closer to LDA. Approximating the topic model by combining pre-computed topic models can increase the Bhattacharyya distance from the approximate topic model to LDA topic model.

4.3.4 Conclusion of Comparison between Topic Ranger and FSS with Cube Tree

For a 2-branching Topic Ranger and a FSS with cube tree model with the same number of nodes and pre-computed topic models, we compare the memory space, runtime cost, topic model perplexity and Bhattacharyya distance to LDA topic models. For the data memory space, both RT and RTS needs a larger memory size to store the data than CT. For the topic-range queries, RT and RTS also needs more time to query the result dataset than CT. However, for the runtime cost of the FSS algorithm, it is possible that RTS provides a lower runtime cost than CT. For the runtime cost of the OLDA algorithm, RTS OLDA is even more likely to complete the topic model approximation with a lower time cost than CT OLDA.

Beside the comparison in memory space and runtime cost, the perplexity and Bhattacharyya distance are also compared. For the perplexity of the topic models, RTS and RT generates the lowest perplexity among all the implementations for most of the topic-range queries, which means that we are more likely to generate a dataset close to the query result by referring to the topic models computed by RTS and RT. However, when we are using the FSS algorithm to compute approximate topic models, the Bhattacharyya distance comparison is not consistent in view of data indexing structure. Even if we use

the OLDA to compute the approximate topic models, the Bhattacharyya distance comparison results are still random in view of data indexing structures. The only conclusion about the Bhattacharyya distance is that: To shorten the Bhattacharyya distance from the approximate topic model to the LDA topic model, the OLDA algorithm is a better algorithm to use than the FSS algorithm.

4.4 Summary

In this chapter, we have stated the experiment details, presented the experiment results, compared the results of different algorithms and discussed the possible reasons. We use a processed dataset with 1-million Tweets to generate the training and testing set. We firstly use the training set to pre-calculate topic models use different algorithms, and then we generate random topic-range queries and calculate the approximated topic model of the query results. Finally, we use the query results of the testing set to evaluate the topic model.

In Section 4.2, we compare the approximated topic models using Topic Ranger with different parameters. We implement two versions of Topic Ranger: Topic Ranger using full range tree (RTS) and Topic Ranger using range tree without subtrees on other dimensions (RT). A full range tree provides a much smaller querying runtime cost than a range tree without subtrees on other dimensions. However, it takes much more memory space. Therefore, RTS always provides a smaller querying runtime cost than RT with a much larger memory space cost.

Besides, there are two more parameters we have considered: branching factor and leaf node size of the hierarchical tree. For the branching factor, we choose the number of branches from 2 to 6. We find that the more branches the hierarchical tree has, the smaller memory space and less preprocessing time Topic Ranger takes. However, for the querying runtime cost and the topic model approximation runtime cost, there is no

trend can be identified according to the branching factor. For the perplexity of the topic models, hierarchical trees with more branches (5 and 6) has better perplexity, but there is also no trend can be found. The Bhattacharyya distance is random in the view of branching factor. These results infers that the results are not only influenced by the branching factor, but also other latent factors.

For the leaf node size, we choose the size 500, 2000, 5000 and 10000. We find that the larger leaf node size the hierarchical tree has, the smaller memory space and less preprocessing time Topic Ranger takes. For the querying runtime cost, large leaf node size (10000) provides better performance. However, for the topic model approximation runtime cost, leaf node size 500 performs the best for most of the topic-range queries. For the perplexity of the topic models, Topic Ranger with leaf node size 2000 calculates the topic models with the best perplexity. The Bhattacharyya distance is still random in the view of branching factor, which means that the Bhattacharyya distance might be affected by multiple latent factors.

In Section 4.3, we compare RT and RTS with the FSS algorithm with Cube Tree (CT). CT has the best memory space cost and querying runtime cost among the three. However, for the topic model approximation runtime cost, RTS has smaller runtime cost than CT for some of the topic-range queries. We also compare the FSS algorithm with LDA and Online-LDA, and find that the FSS algorithm takes lower runtime cost than Online-LDA and LDA. For the perplexity of the topic models, the FSS algorithm provides the best perplexity in most of the topic-range queries. Among the 15 random topic-range queries, the FSS algorithm has worse perplexity than Online-LDA in only two queries (No. 7 and 14). However, for the Bhattacharyya distance, Online-LDA is more likely to generate topic models with lower Bhattacharyya distance to LDA topic model than the FSS algorithm. This result infers that combining topic models might cause a larger difference from the approximate topic model to the directly computed LDA topic model.

Chapter 5

Summary and Future Work

5.1 Summary of the Research

This thesis has documented the design and implementation of Topic Ranger to support fast and accurate topic-range queries. We have done the literature review on multiple data indexing structures for a better performance of topic-range query. We have also reviewed the topic modeling algorithms and evaluation metrics of topic models, so that we can compare the quality of topic models generated by different algorithms.

Based on the literature review, we have finally designed a new tool called "Topic Ranger" for supporting the topic-range queries. We have changed the data indexing structure from cube tree to hierarchical range tree, and performed tests on Topic Ranger with different parameter settings. With the comparison of runtime cost, data storage size, perplexity of topic model, and Bhattacharyya distance from generated model to LDA topic model, we have found that the parameter setting does affect the runtime cost, memory cost and topic model quality. The higher the branching factor and leaf node size are, the less preprocessing time and data storage size Topic Ranger incurs. Although the parameter setting has an intuitive impact on the preprocessing time cost and data storage size, the runtime cost of topic model querying process and the topic model quality are found to be affected by both the parameter setting and range query size in the experiments.

To compare the performance of Topic Ranger and FSS with Cube Tree, which is the original implementation of FSS algorithm, we have also measured the runtime cost as well as the topic model quality. Although we have generated two versions of Topic Ranger, RT (Topic Ranger using range tree without subtrees on other dimensions) and RTS (Topic Ranger using full range tree), both of them cannot beat FSS with cube tree in query time cost and memory space cost. However, for some of the topic range queries, it is possible that RTS (or even RT) incurs a lower runtime cost in topic modeling than FSS with cube tree.

For the perplexity of topic model, there is a difference between the topic model of Topic Ranger and FSS with cube tree in the general case. Topic Ranger can provide topic models with a lower perplexity than FSS with cube tree for most of the queries. However, for Bhattacharyya distance comparison to LDA (Latent Dirichlet Allocation) topic model, the topic models of FSS generate random Bhattacharyya distances. It is difficult to find a consistent phenomenon in Bhattacharyya distance in view of data indexing structures.

5.2 Future Directions

We would suggest the following issues for further research:

- Study on Topic Ranger parameter settings: Although we have done experiments to study this issue, we still need to get more conclusive results with larger datasets and random topic range queries. The latent factors that affect the Bhattacharyya distance also need to be found.
- Balance between query time cost and data storage size: To decrease the data storage size, we remove all the sub-trees on higher dimensions, which makes the query time cost much higher. A Topic Ranger with both small size of data storage and quick query speed should be valuable and with high potential.

- Balance between the time cost and memory space cost: With a smaller value of branching factor and leaf node size, we can have a set of nodes with more details in the data indexing structure. However, we do not need such a detailed Topic Ranger for every corpus and for every range query. Quantifying the branching factor and leaf node size with the information of corpus and size of range queries is an interesting problem, which is also meaningful in enhancing efficiency of Topic Ranger.
- Relationship between topic model quality and other possible factors: From Topic Rangers and FSS with cube tree with different setting, we did not find a best algorithm that excels under all evaluation metrics for all the types of range queries. The quality of the topic model is not directly relevant to the data indexing structure of the algorithm. It might be caused by the size of covered nodes, the size of the biggest covered nodes, the number of the data records being included in the covered nodes, or even some more complex parameters.

We look forward to further improvements in the algorithms for topic-range queries.

Appendices

Appendix A

An Example of Tree Construction of Topic Ranger

For a toy dataset with 32 Tweet records, we want to build a hierarchical tree indexing small range trees, so we firstly construct an empty RH node as the root of the hierarchical tree. The toy dataset contains Tweets records below:

```
2015-01-01 00:00:01 Grier,wish,follow,u 33.745058 -117.911645
2015-01-01 00:00:01 YEAR 39.05447 -77.065632
2015-01-01 00:00:02 mph,Temperature,Humidity 38.679722 -90.463056
2015-01-01 00:00:03 love 36.472223 -94.109471
2015-01-01 00:00:03 woke 42.840735 -83.788655
2015-01-01 00:00:06 girls,Youre,sweaty 38.964343 -94.781334
2015-01-01 00:00:06 employees 40.525679 -75.500143
2015-01-01 00:00:07 up,breakfast,brother 40.02093 -86.006587
2015-01-01 00:00:06 jarred,promote 33.789318 -84.536674
2015-01-01 00:00:06 fault 40.632396 -74.107769
2015-01-01 00:00:09 squidney,sydviciouss 38.931457 -77.129937
2015-01-01 00:00:11 taxi,ride 40.137019 -74.240181
2015-01-01 00:00:11 see,and,disappearing,meant,be,my 42.349632 -71.101461
2015-01-01 00:00:02 Time,party,Chun,want,involved 32.908693 -96.764122
```

2015-01-01 00:00:11 tired,get 43.675232 -70.321317
2015-01-01 00:00:11 real,drinking 36.833716 -76.299604
2015-01-01 00:00:03 Humidity,Barometer 28.058333 -82.320556
2015-01-01 00:00:03 meet,are,world,are,SURE 33.620889 -117.939024
2015-01-01 00:00:05 beautiful,Finest 32.7271 -117.197676
2015-01-01 00:00:16 tonight,out 39.985139 -76.017759
2015-01-01 00:00:17 Year,love,on,2015,bless 18.334839 -64.91843
2015-01-01 00:00:21 Job,Giraffe 41.126485 -73.71402
2015-01-01 00:00:22 baby,cross,right 39.125244 -84.5226
2015-01-01 00:00:13 tomorrow,the,2015,out,been 32.95404 -97.064544
2015-01-01 00:00:22 more,spend 39.394241 -76.581589
2015-01-01 00:00:14 sick 33.820855 -86.048699
2015-01-01 00:00:13 ever,thought,you,you,crawl,take,hour,nap 39.385706 -84.488837
2015-01-01 00:00:13 bursting,fanfiction 33.49238 -112.016631
2015-01-01 00:00:15 Limestone,Barometer,today,58 34.754722 -87.122778
2015-01-01 00:00:15 decimal,logical,an 40.593732 -74.105752
2015-01-01 00:00:24 weeks,god,you 35.790052 -86.413224
2015-01-01 00:00:24 Company,Part 38.646991 -90.224967

For the hierarchical tree, we set the branching factor to 2, and the leaf node size to 5. The order of dimensions we use to sort and split the dataset is in the order "timestamp, latitude, longitude". Since the size of the toy dataset is larger than leaf node size 5, we need to split the toy dataset into 2 subsets and input them to two child nodes of the root RH node. Therefore, we only store the boundaries of the toy dataset in the root RH node, then we sort the toy dataset by the timestamp, then we split it into two subset equally:

Subset 1:

1/1/2015 0:00:01 Grier,wish,follow,u 33.745058 -117.911645
 1/1/2015 0:00:01 YEAR 39.05447 -77.065632
 1/1/2015 0:00:02 mph,Temperature,Humidity 38.679722 -90.463056
 1/1/2015 0:00:02 Time,party,Chun,want,involved 32.908693 -96.764122
 1/1/2015 0:00:03 love 36.472223 -94.109471
 1/1/2015 0:00:03 woke 42.840735 -83.788655
 1/1/2015 0:00:03 Humidity,Barometer 28.058333 -82.320556
 1/1/2015 0:00:03 meet,are,world,are,SURE 33.620889 -117.939024
 1/1/2015 0:00:05 beautiful,Finest 32.7271 -117.197676
 1/1/2015 0:00:06 girls,Youre,sweaty 38.964343 -94.781334
 1/1/2015 0:00:06 employees 40.525679 -75.500143
 1/1/2015 0:00:06 jarred,promote 33.789318 -84.536674
 1/1/2015 0:00:06 fault 40.632396 -74.107769
 1/1/2015 0:00:07 up,breakfast,brother 40.02093 -86.006587
 1/1/2015 0:00:09 squidney,sydvicious 38.931457 -77.129937
 1/1/2015 0:00:11 taxi,ride 40.137019 -74.240181

Subset 2:

1/1/2015 0:00:11 see,and,disappearing,meant,be,my 42.349632 -71.101461
 1/1/2015 0:00:11 tired,get 43.675232 -70.321317
 1/1/2015 0:00:11 real,drinking 36.833716 -76.299604
 1/1/2015 0:00:13 tomorrow,the,2015,out,been 32.95404 -97.064544

1/1/2015 0:00:13 ever,thought,you,you,crawl,take,hour,nap 39.385706 -84.488837
1/1/2015 0:00:13 bursting,fanfiction 33.49238 -112.016631
1/1/2015 0:00:14 sick 33.820855 -86.048699
1/1/2015 0:00:15 Limestone,Barometer,today,58 34.754722 -87.122778
1/1/2015 0:00:15 decimal,logical,an 40.593732 -74.105752
1/1/2015 0:00:16 tonight,out 39.985139 -76.017759
1/1/2015 0:00:17 Year,love,on,2015,bless 18.334839 -64.91843
1/1/2015 0:00:21 Job,Giraffe 41.126485 -73.71402
1/1/2015 0:00:22 baby,cross,right 39.125244 -84.5226
1/1/2015 0:00:22 more,spend 39.394241 -76.581589
1/1/2015 0:00:24 weeks,god,you 35.790052 -86.413224
1/1/2015 0:00:24 Company,Part 38.646991 -90.224967

Since both Subset 1 and 2 have 16 Tweet records, which is larger than leaf node size 5, we need to split them to construct one more level of RH nodes. But for this time, we use "latitude" to sort and split the dataset, instead of "timestamp":

Subset 3:

1/1/2015 0:00:03 Humidity,Barometer 28.058333 -82.320556
1/1/2015 0:00:05 beautiful,Finest 32.7271 -117.197676
1/1/2015 0:00:02 Time,party,Chun,want,involved 32.908693 -96.764122
1/1/2015 0:00:03 meet,are,world,are,SURE 33.620889 -117.939024
1/1/2015 0:00:01 Grier,wish,follow,u 33.745058 -117.911645
1/1/2015 0:00:06 jarred,promote 33.789318 -84.536674

1/1/2015 0:00:03 love 36.472223 -94.109471

1/1/2015 0:00:02 mph, Temperature, Humidity 38.679722 -90.463056

Subset 4:

1/1/2015 0:00:09 squidney, sydvicious 38.931457 -77.129937

1/1/2015 0:00:06 girls, Youre, sweaty 38.964343 -94.781334

1/1/2015 0:00:01 YEAR 39.05447 -77.065632

1/1/2015 0:00:07 up, breakfast, brother 40.02093 -86.006587

1/1/2015 0:00:11 taxi, ride 40.137019 -74.240181

1/1/2015 0:00:06 employees 40.525679 -75.500143

1/1/2015 0:00:06 fault 40.632396 -74.107769

1/1/2015 0:00:03 woke 42.840735 -83.788655

Subset 5:

1/1/2015 0:00:17 Year, love, on, 2015, bless 18.334839 -64.91843

1/1/2015 0:00:13 tomorrow, the, 2015, out, been 32.95404 -97.064544

1/1/2015 0:00:13 bursting, fanfiction 33.49238 -112.016631

1/1/2015 0:00:14 sick 33.820855 -86.048699

1/1/2015 0:00:15 Limestone, Barometer, today, 58 34.754722 -87.122778

1/1/2015 0:00:24 weeks, god, you 35.790052 -86.413224

1/1/2015 0:00:11 real, drinking 36.833716 -76.299604

1/1/2015 0:00:24 Company, Part 38.646991 -90.224967

Subset 6:

1/1/2015 0:00:22 baby,cross,right 39.125244 -84.5226
1/1/2015 0:00:13 ever,thought,you,you,crawl,take,hour,nap 39.385706 -84.488837
1/1/2015 0:00:22 more,spend 39.394241 -76.581589
1/1/2015 0:00:16 tonight,out 39.985139 -76.017759
1/1/2015 0:00:15 decimal,logical,an 40.593732 -74.105752
1/1/2015 0:00:21 Job,Giraffe 41.126485 -73.71402
1/1/2015 0:00:11 see,and,disappearing,meant,be,my 42.349632 -71.101461
1/1/2015 0:00:11 tired,get 43.675232 -70.321317

Still, Subset 3-6 have Tweet records more than leaf node size 5. So we split the four subsets into eight smaller subsets. In this time of splitting, we sort the data by *longitude*, so the subsets are:

Subset 7:

1/1/2015 0:00:03 meet,are,world,are,SURE 33.620889 -117.939024
1/1/2015 0:00:01 Grier,wish,follow,u 33.745058 -117.911645
1/1/2015 0:00:05 beautiful,Finest 32.7271 -117.197676
1/1/2015 0:00:02 Time,party,Chun,want,involved 32.908693 -96.764122

Subset 8:

1/1/2015 0:00:03 love 36.472223 -94.109471
1/1/2015 0:00:02 mph,Temperature,Humidity 38.679722 -90.463056

1/1/2015 0:00:06 jarred,promote 33.789318 -84.536674

1/1/2015 0:00:03 Humidity,Barometer 28.058333 -82.320556

Subset 9:

1/1/2015 0:00:06 girls,Youre,sweaty 38.964343 -94.781334

1/1/2015 0:00:07 up,breakfast,brother 40.02093 -86.006587

1/1/2015 0:00:03 woke 42.840735 -83.788655

1/1/2015 0:00:09 squidney,sydvicious 38.931457 -77.129937

Subset 10:

1/1/2015 0:00:01 YEAR 39.05447 -77.065632

1/1/2015 0:00:06 employees 40.525679 -75.500143

1/1/2015 0:00:11 taxi,ride 40.137019 -74.240181

1/1/2015 0:00:06 fault 40.632396 -74.107769

Subset 11:

1/1/2015 0:00:13 bursting,fanfiction 33.49238 -112.016631

1/1/2015 0:00:13 tomorrow,the,2015,out,been 32.95404 -97.064544

1/1/2015 0:00:24 Company,Part 38.646991 -90.224967

1/1/2015 0:00:15 Limestone,Barometer,today,58 34.754722 -87.122778

Subset 12:

1/1/2015 0:00:24 weeks,god,you 35.790052 -86.413224
1/1/2015 0:00:14 sick 33.820855 -86.048699
1/1/2015 0:00:11 real,drinking 36.833716 -76.299604
1/1/2015 0:00:17 Year,love,on,2015,bless 18.334839 -64.91843

Subset 13:

1/1/2015 0:00:22 baby,cross,right 39.125244 -84.5226
1/1/2015 0:00:13 ever,thought,you,you,crawl,take,hour,nap 39.385706 -84.488837
1/1/2015 0:00:22 more,spend 39.394241 -76.581589
1/1/2015 0:00:16 tonight,out 39.985139 -76.017759

Subset 14:

1/1/2015 0:00:15 decimal,logical,an 40.593732 -74.105752
1/1/2015 0:00:21 Job,Giraffe 41.126485 -73.71402
1/1/2015 0:00:11 see,and,disappearing,meant,be,my 42.349632 -71.101461
1/1/2015 0:00:11 tired,get 43.675232 -70.321317

When we split the subsets till they each only contains 4 Tweet records, we can finally construct eight RF nodes to store Subset 7-14. To save the memory space in construction of range trees, we replace the Tweets record content by their indices in the subset, and store the information of timestamp, latitude, longitude and index as a Point. The structure of a Point is: (timestamp, latitude, longitude): index. Here, we do not directly use the time format of timestamp. Instead, we use ticks of the timestamp, which can

be presented in the format of long number. For example, the ticks of the timestamp "1/1/2015 0:00:01" is 635556672010000000. So the Point used to construct the eight range trees are listed below:

Point in Range Tree 1:

(635556672030000000, 33.620889, -117.939024): 0
(635556672010000000, 33.745058, -117.911645): 1
(635556672050000000, 32.7271, -117.197676):2
(635556672020000000, 32.908693, -96.764122):3

Point in Range Tree 2:

(635556672030000000, 36.472223, -94.109471):0
(635556672020000000, 38.679722, -90.463056):1
(635556672060000000, 33.789318, -84.536674):2
(635556672030000000, 28.058333, -82.320556):3

Point in Range Tree 3:

(635556672060000000, 38.964343, -94.781334):0
(635556672070000000, 40.02093, -86.006587):1
(635556672030000000, 42.840735, -83.788655):2
(635556672090000000, 38.931457, -77.129937):3

Point in Range Tree 3:

(635556672010000000, 39.05447, -77.065632):0
 (635556672060000000, 40.525679, -75.500143):1
 (635556672110000000, 40.137019, -74.240181):2
 (635556672060000000, 40.632396, -74.107769):3

Point in Range Tree 4:

(635556672130000000, 33.49238, -112.016631):0
 (635556672130000000, 32.95404, -97.064544):1
 (635556672240000000, 38.646991, -90.224967):2
 (635556672150000000, 34.754722, -87.122778):3

Point in Range Tree 5:

(635556672240000000, 35.790052, -86.413224):0
 (635556672140000000, 33.820855, -86.048699):1
 (635556672110000000, 36.833716, -76.299604):2
 (635556672170000000, 18.334839, -64.91843):3

Point in Range Tree 6:

(635556672220000000, 39.125244, -84.5226):0
 (635556672130000000, 39.385706, -84.488837):1
 (635556672220000000, 39.394241, -76.581589):2
 (635556672160000000, 39.985139, -76.017759):3

Point in Range Tree 7:

(635556672150000000, 40.593732, -74.105752):0

(635556672210000000, 41.126485, -73.71402):1

(635556672110000000, 42.349632, -71.101461):2

(635556672110000000, 43.675232, -70.321317):3

The final structure of the hierarchical range tree of the toy dataset is shown in Fig.

A.1:

Figure A.1: An example of 2-branching hierarchical tree with sub range trees as leaf nodes for the toy dataset

As shown in Fig. A.1, the toy dataset is passed from the root of the hierarchical tree to the leaf nodes to construct small range trees. Since the Point is a data structure with

a 3D vector and a value, the range trees are constructed as three-dimensional range trees. Whenever we pass a topic-range query to the hierarchical tree, we pass the topic-range query to the root of the hierarchical tree, check the intersection of the query range with the boundaries of the RH nodes, and only let the intersected RH nodes pass down the query range to their child nodes till the leaf nodes and the range trees.

Appendix B

Topic-Range Query Set of Chapter 5

Table B.1: Topic-Range Query Set of Chapter 5

South	West	North	East	Start Time	End Time	Latitude (De- gree)	Longitude (Degree)	Time Period (Hour)	Query Result Size
25	-110	40	-105	1/1/2015 8:00	1/1/2015 12:00	15	5	4	2482
25	-110	40	-105	1/1/2015 6:00	1/1/2015 12:00	15	5	6	3633
25	-110	40	-105	1/1/2015 4:00	1/1/2015 12:00	15	5	8	4603
25	-110	40	-100	1/1/2015 8:00	1/1/2015 12:00	15	10	4	5684
25	-110	40	-100	1/1/2015 6:00	1/1/2015 12:00	15	10	6	8436
25	-110	40	-100	1/1/2015 4:00	1/1/2015 12:00	15	10	8	11179
25	-110	40	-95	1/1/2015 8:00	1/1/2015 12:00	15	15	4	36430
25	-110	40	-95	1/1/2015 6:00	1/1/2015 12:00	15	15	6	54660
25	-110	40	-95	1/1/2015 4:00	1/1/2015 12:00	15	15	8	73924
30	-110	40	-105	1/1/2015 8:00	1/1/2015 12:00	10	5	4	2482
30	-110	40	-105	1/1/2015 6:00	1/1/2015 12:00	10	5	6	3633
30	-110	40	-105	1/1/2015 4:00	1/1/2015 12:00	10	5	8	4603
30	-110	40	-100	1/1/2015 8:00	1/1/2015 12:00	10	10	4	5600
30	-110	40	-100	1/1/2015 6:00	1/1/2015 12:00	10	10	6	8318
30	-110	40	-100	1/1/2015 4:00	1/1/2015 12:00	10	10	8	11007
30	-110	40	-95	1/1/2015 8:00	1/1/2015 12:00	10	15	4	24929
30	-110	40	-95	1/1/2015 6:00	1/1/2015 12:00	10	15	6	37558
30	-110	40	-95	1/1/2015 4:00	1/1/2015 12:00	10	15	8	50975
35	-110	40	-105	1/1/2015 8:00	1/1/2015 12:00	5	5	4	1771
35	-110	40	-105	1/1/2015 6:00	1/1/2015 12:00	5	5	6	2632
35	-110	40	-105	1/1/2015 4:00	1/1/2015 12:00	5	5	8	3280
35	-110	40	-100	1/1/2015 8:00	1/1/2015 12:00	5	10	4	3682
35	-110	40	-100	1/1/2015 6:00	1/1/2015 12:00	5	10	6	5513
35	-110	40	-100	1/1/2015 4:00	1/1/2015 12:00	5	10	8	7289
35	-110	40	-95	1/1/2015 8:00	1/1/2015 12:00	5	15	4	7306
35	-110	40	-95	1/1/2015 6:00	1/1/2015 12:00	5	15	6	10963
35	-110	40	-95	1/1/2015 4:00	1/1/2015 12:00	5	15	8	14693

Appendix C

Topic-Range Query Set of Chapter 6

Table C.1: Topic-Range Query Set of Chapter 6

Query In-dex	South	West	North	East	Start Time	End Time	Latitude (De-gree)	Longitude (De-gree)	Time Period (Day)	Query Result Size
1	46.78983711	-68.42187113	61.78983711	-53.42187113	1/14/2015 10:14	1/19/2015 10:14	15	15	5	46
2	22.7319293	-127.1135793	32.7319293	-117.1135793	1/7/2015 00:10	1/12/2015 00:10	10	10	5	4771
3	46.99755332	-101.3778755	56.99755332	-91.37787545	1/8/2015 07:44	1/13/2015 7:44	10	10	5	5470
4	46.99755332	-101.3778755	61.99755332	-86.37787545	1/8/2015 07:44	1/13/2015 7:44	15	15	5	6243
5	34.52670388	-105.7956379	44.52670388	-95.79563792	1/14/2015 5:47	1/19/2015 5:47	10	10	5	146641
6	34.52670388	-105.7956379	49.52670388	-90.79563792	1/14/2015 5:47	1/19/2015 5:47	15	15	5	315083
7	21.0256495	-103.1693811	31.0256495	-93.16938109	1/10/2015 02:45	1/15/2015 2:45	10	10	5	340568
8	41.12128984	-83.65405363	61.12128984	-73.65405363	1/1/2015 01:04	1/6/2015 01:04	20	10	5	438942
9	41.12128984	-83.65405363	66.12128984	-68.65405363	1/1/2015 01:04	1/6/2015 01:04	25	15	5	699493
10	41.12128984	-83.65405363	71.12128984	-63.65405363	1/1/2015 01:04	1/6/2015 01:04	30	15	5	700102
11	34.52670388	-105.7956379	54.52670388	-85.79563792	1/14/2015 5:47	1/19/2015 5:47	20	20	5	729156
12	22.7319293	-127.1135793	37.7319293	-112.1135793	1/7/2015 00:10	1/12/2015 00:10	15	15	5	734543
13	21.0256495	-103.1693811	35.0256495	-88.16938109	1/10/2015 02:45	1/15/2015 2:45	14	15	5	818858
14	22.7319293	-127.1135793	42.7319293	-107.1135793	1/7/2015 00:10	1/12/2015 00:10	20	20	5	991289
15	21.0256495	-103.1693811	41.0256495	-83.16938109	1/10/2015 02:45	1/15/2015 2:45	20	20	5	1597805

Appendix D

Figures - Comparison of Query Runtime Cost: CT, RTS and RT

Figure D.1: Comparison of Query Runtime Cost: CT, RTS and RT - Query 1-4

Figure D.2: Comparison of Query Runtime Cost: CT, RTS and RT - Query 5-8

Figure D.3: Comparison of Query Runtime Cost: CT, RTS and RT - Query 9-12

Figure D.4: Comparison of Query Runtime Cost: CT, RTS and RT - Query 13-15

Appendix E

Figures - Comparison of Query Runtime Cost: CT and RTS

Figure E.1: Comparison of Query Runtime Cost: CT and RTS - Query 1-4

Figure E.2: Comparison of Query Runtime Cost: CT and RTS - Query 5-8

Figure E.3: Comparison of Query Runtime Cost: CT and RTS - Query 9-12

Figure E.4: Comparison of Query Runtime Cost: CT and RTS - Query 13-15

Appendix F

Figures - Comparison of FSS Runtime Cost

Figure F.1: Comparison of FFS Runtime Cost: Query 1-4

Figure F.2: Comparison of FFS Runtime Cost: Query 5-8

Figure F.3: Comparison of FFS Runtime Cost: Query 9-12

Figure F.4: Comparison of FFS Runtime Cost: Query 13-15

Appendix G

Figures - Comparison of OLDA Runtime Cost

Figure G.1: Comparison of OLDA Runtime Cost: Query 1-4

Figure G.2: Comparison of OLDA Runtime Cost: Query 5-8

Figure G.3: Comparison of OLDA Runtime Costs: Query 9-12

Figure G.4: Comparison of OLDA Runtime Cost: Query 13-15

Appendix H

Figures - Comparison of Runtime Cost of All Algorithms

Figure H.1: Comparison of Runtime Cost of All Algorithms: Query 1-4

Figure H.2: Comparison of Runtime Cost of All Algorithms: Query 5-8

Figure H.3: Comparison of Runtime Cost of All Algorithms: Query 9-12

Figure H.4: Comparison of Runtime Cost of All Algorithms: Query 13-15

Appendix I

Figures - Comparison of Topic Model Perplexity of All Algorithms

Figure I.1: Comparison of Topic Model Perplexity of All Algorithms: Query 1-4

Figure I.2: Comparison of Topic Model Perplexity of All Algorithms: Query 5-8

Figure I.3: Comparison of Topic Model Perplexity of All Algorithms: Query 9-12

Figure I.4: Comparison of Topic Model Perplexity of All Algorithms: Query 13-15

Appendix J

Figures - Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models

Figure J.1: Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 1-4

Figure J.2: Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 5-8

Figure J.3: Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 9-12

Figure J.4: Comparison of Bhattacharyya Distance from OLDA Topic Models to LDA Topic Models: Query 13-15

Appendix K

**Figures - Comparison of
Bhattacharyya Distance from FSS
and OLDA Topic Models to LDA
Topic Models In View of Data
Indexing Structures**

Figure K.1: Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 1-4

Figure K.2: Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 5-8

Figure K.3: Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 9-12

Figure K.4: Comparison of Bhattacharyya Distance from FSS and OLDA Topic Models to LDA Topic Models In View of Data Indexing Structures: Query 13-15

References

- [1] J. A. Obar and S. Wildman, “Social media definition and the governance challenge: An introduction to the special issue,” *Telecommunications Policy*, vol. 39, 08 2015.
- [2] M. Riese. (2016) The definitive history of social media. [Online]. Available: <https://www.dailydot.com/debug/history-of-social-media/>
- [3] A. Perrin, *Social Media Usage: 2005-2015: 65% of Adults Now Use Social Networking Sites—a Nearly Tenfold Jump in the Past Decade*. Pew Research Trust, 2015. [Online]. Available: <https://books.google.com.sg/books?id=OupAnQAACAAJ>
- [4] WE ARE SOCIAL UK. (2018) Global digital report 2018. [Online]. Available: <https://wearesocial.com/uk/blog/2018/01/global-digital-report-2018>
- [5] DOMO. (2018) Data never sleeps 6.0. [Online]. Available: <https://www.domo.com/learn/data-never-sleeps-6>
- [6] D. M. Blei, “Probabilistic topic models,” *Commun. ACM*, vol. 55, no. 4, pp. 77–84, Apr. 2012. [Online]. Available: <http://doi.acm.org/10.1145/2133806.2133826>
- [7] L. Bolelli, c. Ertekin, and C. L. Giles, “Topic and trend detection in text collections using latent dirichlet allocation,” in *Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval*, ser. ECIR ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 776–780. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00958-7_84

- [8] J. H. Lau, N. Collier, and T. Baldwin, “On-line trend analysis with topic models: #twitter trends detection topic model online,” in *COLING*, 2012.
- [9] Y. Tian, P. S. Kochhar, E.-P. Lim, F. Zhu, and D. Lo, “Predicting best answerers for new questions: An approach leveraging topic modeling and collaborative voting,” in *Social Informatics*, A. Nadamoto, A. Jatowt, A. Wierzbicki, and J. L. Leidner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 55–68.
- [10] D. Ramage, E. Rosen, J. Chuang, C. Manning, and D. Mcfarland, “Topic modeling for the social sciences,” 01 2009.
- [11] N. Sukhija, M. Tatineni, N. Brown, M. V. Moer, P. Rodriguez, and S. Callicott, “Topic modeling and visualization for big data in social sciences,” in *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCCom/IoP/SmartWorld)*, July 2016, pp. 1198–1205.
- [12] R. Wesslen, “Computer-assisted text analysis for social science: Topic models and beyond,” *CoRR*, vol. abs/1803.11045, 2018. [Online]. Available: <http://arxiv.org/abs/1803.11045>
- [13] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003. [Online]. Available: <http://dl.acm.org/citation.cfm?id=944919.944937>
- [14] M. D. Hoffman, D. M. Blei, and F. Bach, “Online learning for latent dirichlet allocation,” in *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’10. USA: Curran Associates

REFERENCES

- Inc., 2010, pp. 856–864. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2997189.2997285>
- [15] K. Zhao, L. Chen, and G. Cong, “Topic exploration in spatio-temporal document collections,” in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD ’16. New York, NY, USA: ACM, 2016, pp. 985–998. [Online]. Available: <http://doi.acm.org/10.1145/2882903.2882921>
- [16] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975. [Online]. Available: <http://doi.acm.org/10.1145/361002.361007>
- [17] I. Wald and V. Havran, “On building fast kd-trees for ray tracing, and on doing that in $O(n \log n)$,” *2006 IEEE Symposium on Interactive Ray Tracing*, pp. 61–69, 2006.
- [18] J. T. Robinson, “The k-d-b-tree: A search structure for large multidimensional dynamic indexes,” in *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’81. New York, NY, USA: ACM, 1981, pp. 10–18. [Online]. Available: <http://doi.acm.org/10.1145/582318.582321>
- [19] D. D. Couto and J. Napoli. (1998) kd trees. [Online]. Available: <http://groups.csail.mit.edu/graphics/classes/6.838/S98/meetings/m13/kd.html>
- [20] T. K. Chan, I. Gargantini, and T. R. S. Walsh, “Conversion and integration of boundary representations with octrees,” in *Proceedings of Graphics Interface ’89*, ser. GI ’89. Toronto, Ontario, Canada: Canadian Man-Computer Communications Society, 1989, pp. 203–210.
- [21] Wikipedia. (2018) Octree. [Online]. Available: <https://en.wikipedia.org/wiki/Octree>

REFERENCES

- [22] M. Kamburelis. (2006) Castle game engine documentation. [Online]. Available: https://castle-engine.io/vrml_engine_doc/output/xsl/vrml_engine.pdf
- [23] S. M. Omohundrol and S. M. Omohundro, “Five balltree construction algorithms,” 1989.
- [24] A. M. Kibriya and E. Frank, “An empirical comparison of exact nearest neighbour algorithms,” in *Knowledge Discovery in Databases: PKDD 2007*, J. N. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenič, and A. Skowron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 140–151.
- [25] J. Vanderplas, A. Connolly, Ž. Ivezić, and A. Gray, “Introduction to astroml: Machine learning for astrophysics,” in *Conference on Intelligent Data Understanding (CIDU)*, oct. 2012, pp. 47–54.
- [26] A. Beygelzimer, S. Kakade, and J. Langford, “Cover trees for nearest neighbor,” in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML ’06. New York, NY, USA: ACM, 2006, pp. 97–104. [Online]. Available: <http://doi.acm.org/10.1145/1143844.1143857>
- [27] M. Izbicki and C. Shelton, “Fast direct triangulation of 3d surfaces-computational complexity over cover trees,” in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1162–1170. [Online]. Available: <http://proceedings.mlr.press/v37/izbicki15.html>
- [28] Y. Wang, A. Meliou, and G. Miklau, “Rc-index: Diversifying answers to range queries,” *Proc. VLDB Endow.*, vol. 11, no. 7, pp. 773–786, Mar. 2018. [Online]. Available: <https://doi.org/10.14778/3192965.3192969>

REFERENCES

- [29] R. D. Kumar and K. Ramareddy, “Article: a simple region descriptor based on object area per scan line,” *International Journal of Computer Applications*, vol. 3, no. 7, pp. 24–27, June 2010, published By Foundation of Computer Science.
- [30] B. Chazelle, “Lower bounds for orthogonal range searching: I. the reporting case,” *J. ACM*, vol. 37, no. 2, pp. 200–212, Apr. 1990. [Online]. Available: <http://doi.acm.org/10.1145/77600.77614>
- [31] —, “Lower bounds for orthogonal range searching: Part ii. the arithmetic model,” *J. ACM*, vol. 37, no. 3, pp. 439–463, Jul. 1990. [Online]. Available: <http://doi.acm.org/10.1145/79147.79149>
- [32] B. Chazelle and L. J. Guibas, “Fractional cascading: I. a data structuring technique,” *Algorithmica*, vol. 1, no. 1, pp. 133–162, Nov 1986. [Online]. Available: <https://doi.org/10.1007/BF01840440>
- [33] Q. Shi and J. F. JaJa, “Fast fractional cascading and its applications,” vol. UMIACS-TR-2003-71, 2003/08/01/ 2003. [Online]. Available: <http://drum.lib.umd.edu/handle/1903/1296>
- [34] D. Ryppl and Z. Bittnar, “Triangulation of 3d surfaces: From parametric to discrete surfaces,” in *Proceedings of the Third International Conference on Engineering Computational Technology*, ser. ICECT’03. Edinburgh, UK, UK: Civil-Comp press, 2002, pp. 33–34. [Online]. Available: <http://dl.acm.org/citation.cfm?id=870555.870568>
- [35] T. K. Landauer, P. W. Foltz, and D. Laham, “An introduction to latent semantic analysis,” *Discourse Processes*, vol. 25, no. 2-3, pp. 259–284, 1998. [Online]. Available: <https://doi.org/10.1080/01638539809545028>

REFERENCES

- [36] T. Hofmann, “Learning the similarity of documents: An information-geometric approach to document retrieval and categorization,” 2000.
- [37] M. Girolami and A. Kaban, “On an equivalence between plsi and lda,” 2003.
- [38] I. Yildirim. (August 2012) Bayesian inference: Gibbs sampling. [Online]. Available: <http://www.mit.edu/~ilkery/papers/GibbsSampling.pdf>
- [39] P. F. Brown, V. J. D. Pietra, R. L. Mercer, S. A. D. Pietra, and J. C. Lai, “An estimate of an upper bound for the entropy of english,” *Computational Linguistics*, vol. 18, no. 1, 1992. [Online]. Available: <http://aclweb.org/anthology/J92-1002>
- [40] W. Li and A. McCallum, “Pachinko allocation: Dag-structured mixture models of topic correlations,” in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML ’06. New York, NY, USA: ACM, 2006, pp. 577–584. [Online]. Available: <http://doi.acm.org/10.1145/1143844.1143917>
- [41] D. M. Blei, M. I. Jordan, T. L. Griffiths, and J. B. Tenenbaum, “Hierarchical topic models and the nested chinese restaurant process,” in *Proceedings of the 16th International Conference on Neural Information Processing Systems*, ser. NIPS’03. Cambridge, MA, USA: MIT Press, 2003, pp. 17–24. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2981345.2981348>
- [42] D. M. Blei, T. L. Griffiths, and M. I. Jordan, “The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies,” *J. ACM*, vol. 57, no. 2, pp. 7:1–7:30, Feb. 2010. [Online]. Available: <http://doi.acm.org/10.1145/1667053.1667056>
- [43] J. Chen, J. Zhu, J. Lu, and S. Liu, “Scalable training of hierarchical topic models,” *Proc. VLDB Endow.*, vol. 11, no. 7, pp. 826–839, Mar. 2018. [Online]. Available: <https://doi.org/10.14778/3192965.3192972>

REFERENCES

- [44] B. Wang, Y. Liu, Z. Liu, M. Li, and M. Qi, "Topic selection in latent dirichlet allocation," in *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Aug 2014, pp. 756–760.
- [45] H. M. Wallach, "Structured topic models for language," 2008.
- [46] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno, "Evaluation methods for topic models," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 1105–1112. [Online]. Available: <http://doi.acm.org/10.1145/1553374.1553515>
- [47] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society*, vol. 35, pp. 99–109, 1943.
- [48] T. Kailath, "Kailath, t.: The divergence and bhattacharyya distance measures in signal selection. iee trans. on comm. tech. 15(1), 52-60," *Communication Technology, IEEE Transactions on*, vol. 15, pp. 52 – 60, 03 1967.
- [49] M. S. Khalid and M. B. Malik, "Biased nature of bhattacharyya coefficient in correlation of gray-scale objects," in *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.*, Sept 2005, pp. 209–214.
- [50] A. Singh, A. Yadav, and A. Rana, "Article: K-means with three different distance metrics," *International Journal of Computer Applications*, vol. 67, no. 10, pp. 13–17, April 2013, full text available.
- [51] S. Pandit and S. Gupta, "A comparative study on distance measuring approaches for clustering," *International Journal of Research in Computer Science*, vol. 2, p. 29, 12 2011.

REFERENCES

- [52] D. Bora and D. Anil Kumar Gupta, “Effect of different distance measures on the performance of k-means algorithm: An experimental study in matlab,” vol. 5, 05 2014.
- [53] A. S. Shirkhorshidi, S. Aghabozorgi, and T. Y. Wah, “A comparison study on similarity and dissimilarity measures in clustering continuous data,” *PLOS ONE*, vol. 10, no. 12, pp. 1–20, 12 2015. [Online]. Available: <https://doi.org/10.1371/journal.pone.0144059>
- [54] P. E. BLACK, “Manhattan distance” ”dictionary of algorithms and data structures.” [Online]. Available: <https://ci.nii.ac.jp/naid/20001171731/en/>
- [55] Q. Zhan, “An improved k-means algorithm based on structure features,” *JSW*, vol. 12, pp. 62–81, 2017.
- [56] A. Mousa and Y. Yusof, “An improved chebyshev distance metric for clustering medical images,” vol. 1691, 12 2015, p. 040020.
- [57] S. Kullback and R. A. Leibler, “On information and sufficiency,” *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.
- [58] L. B. Heilprin, “Information theory and statistics. solomon kullback. wiley, new york; chapman and hall, london, 1959. xvii + 395 pp. illus.” *Science*, vol. 131, no. 3404, pp. 917–918, 1960. [Online]. Available: <http://science.sciencemag.org/content/131/3404/917.3>
- [59] C. E. Shannon, “A mathematical theory of communication,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, no. 1, pp. 3–55, Jan. 2001. [Online]. Available: <http://doi.acm.org/10.1145/584091.584093>

REFERENCES

- [60] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker, “Perplexitya measure of the difficulty of speech recognition tasks,” *The Journal of the Acoustical Society of America*, vol. 62, no. S1, pp. S63–S63, 1977. [Online]. Available: <https://doi.org/10.1121/1.2016299>
- [61] R. M. Gray, *Entropy and Information Theory*. Berlin, Heidelberg: Springer-Verlag, 1990.
- [62] P. C. Mahalanobis, “On the generalised distance in statistics,” in *Proceedings National Institute of Science, India*, vol. 2, no. 1, Apr. 1936, pp. 49–55. [Online]. Available: <http://ir.isical.ac.in/dspace/handle/1/1268>
- [63] R. D. Maesschalck, D. Jouan-Rimbaud, and D. Massart, “The mahalanobis distance,” *Chemometrics and Intelligent Laboratory Systems*, vol. 50, no. 1, pp. 1 – 18, 2000. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0169743999000477>
- [64] G. Martos, A. Muñoz, and J. González, “On the generalization of the mahalanobis distance,” in *Proceedings, Part I, of the 18th Iberoamerican Congress on Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications - Volume 8258*, ser. CIARP 2013. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 125–132. [Online]. Available: https://doi.org/10.1007/978-3-642-41822-8_16
- [65] A. R. de Leon and K. C. Carrière, “A generalized mahalanobis distance for mixed data,” *J. Multivar. Anal.*, vol. 92, no. 1, pp. 174–185, Jan. 2005. [Online]. Available: <http://dx.doi.org/10.1016/j.jmva.2003.08.006>
- [66] G. Gallego, C. Cuevas, R. Mohedano, and N. Garca, “On the mahalanobis distance classification criterion for multidimensional normal distributions,” *IEEE Transactions on Signal Processing*, vol. 61, no. 17, pp. 4387–4396, Sept 2013.

REFERENCES

- [67] *More Geometric Data Structures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 219–241. [Online]. Available: https://doi.org/10.1007/978-3-540-77974-2_10
- [68] H. Yuan and M. J. Atallah, “Data structures for range minimum queries in multi-dimensional arrays,” in *SODA*, 2010.
- [69] D. M. Mount. (2001) Lecture 20: Range trees. [Online]. Available: <http://www.cs.umd.edu/~meesh/cmsc420/ContentBook/FormalNotes/MountNotes/lecture20-rangetrees.pdf>
- [70] W.-J. Hsu, Y. Lu, and Z. Q. Lee, “Accelerating topic exploration of multi-dimensional documents,” in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2017, pp. 1520–1527.
- [71] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1643031.1643047>
- [72] J. Chung, P. Kannappan, C. T. Ng, and P. Sahoo, “Measures of distance between probability distributions,” *Journal of Mathematical Analysis and Applications*, vol. 138, pp. 280–292, 02 1989.
- [73] K. A. Lee, C. H. You, H. Li, T. Kinnunen, and K. C. Sim, “Using discrete probabilities with bhattacharyya measure for svm-based speaker verification,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 861–870, May 2011.

REFERENCES

- [74] D. PA and J. Kittler, “Pattern recognition: A statistical approach,” 01 1982.
- [75] V. Andrei and O. Arandjelovic, “Complex temporal topic evolution modelling using the kullback-leibler divergence and the bhattacharyya distance,” *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 2016, 12 2016.
- [76] J. Varadarajan and J. Odobez, “Topic models for scene analysis and abnormality detection,” in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, Sep. 2009, pp. 1338–1345.
- [77] S. Suri. (October 9 2002) Range searching. [Online]. Available: <https://www.cs.ucsb.edu/~suri/cs235/RangeSearching.pdf>