NOVEL GEOSPATIAL DATA SCIENCE TECHNIQUES FOR INTERDISCIPLINARY APPLICATIONS

by

ARPAN MAN SAINJU

ZHE JIANG, COMMITTEE CHAIR
SUSAN VRBSKY
XIAOYAN HONG
LINA PU
YU GAN

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2021

Copyright Arpan Man Sainju 2021
ALL RIGHTS RESERVED

ABSTRACT

With the advancement of GPS and remote sensing technologies, an enormous amount of geospatial data are being collected from various domains. Examples include crime locations, road temporally detailed networks, earth observation imagery, and GPS trajectories. Geospatial data science studies computational techniques to discover interesting, previously unknown, but potentially useful patterns from large spatial datasets. It is important for various applications. Crime hotspot detection helps law enforcement departments to create effective strategies to allocate police resources and to prevent crimes. Earth observation imagery classification plays a crucial role in flood extent mapping and water resource management. Big companies like UPS use truck GPS trajectories data to find efficient routes that can ultimately minimize the delivery time and reduce carbon footprint. However, geospatial data science poses several computational challenges. First, the spatial data volume is rapidly growing. For example, NASA collects around 12TB of earth observation imagery per day. Second, spatial data exhibits spatial dependency which imply nearby samples are not statistically independent. Third, different spatial patterns of interest may exist in different spatial scales. Finally, there can be limited observations. For example, sometimes it can be difficult or even impossible to get the complete observation of spatial features in an area due the presence of obstacles (e.g., clouds).

My thesis investigates novel geospatial data science techniques to address some of these challenges. I propose novel parallel spatial colocation mining algorithms on GPUs to address the challenge of large data volume. Similarly, I propose a deep learning framework to automatically map the road safety features from streetview imagery that captures spatial dependency. Next, I propose a novel approach to address the challenge of limited

observation based on the physics-aware spatial structural constraint. Finally, I propose a novel spatial structured model called hidden Markov contour tree (HMCT), a contour tree structure, to capture directed spatial dependency on flow directions between all locations on a 3D surface.

DEDICATION

I would like to dedicate this work to my parents Manoj Man Sainju and Sita Sainju.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor, Dr. Zhe Jiang, for his continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study. I am also thankful to my committee members Dr. Lina Pu, Dr. Susan Vrbsky, Dr. Xiaoyan Hong, and Dr. Yu Gan, for their valuable suggestions. I want to thank my labmates in the spatial data science lab, including Wenchong He, Stephen Kirby, Sultan Ansari, and Mostafijur Rahman. Finally, I would like to thank my family: my father Manoj Man Sainju, my mother Sita Sainju, my sister Priyanka Sainju, and my wife Sweta Mahaju, for supporting me in my Ph.D. journey.

CONTENTS

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGMENTS v

LIST OF TABLES x

LIST OF FIGURES xi

CHAPTER 1 INTRODUCTION 1

1.1 Geospatial Data Science 1

1.2 Societal Applications 2

1.3 Challenges 3

1.3.1 Spatial Dependency 3

1.3.2 Large Data Volumes 4

1.3.3 Multiple Spatial Scales 4

1.3.4 Limited Observations 5

1.4 Summary of Contributions 5

CHAPTER 2 ADDRESSING THE LARGE DATA VOLUME CHALLENGE: GRID-BASED COLOCATION MINING ALGORITHMS ON GPU FOR BIG SPATIAL EVENT DATA: A SUMMARY OF RESULTS 8

2.1 Introduction 8

2.2 Problem Statement 11

2.2.1 Basic Concepts 11

2.2.2 Problem Definition 12

2.3 Approach 13

2.3.1 Algorithm Overview 14
2.3.2 Cell-Aggregate-Based Upper Bound Filter 15
2.3.3 Refinement Algorithms 19
2.4 Evaluation 23
2.4.1 Results on Synthetic Data 24
2.4.2 Results on Real World Dataset 27
2.5 Discussion 28
2.6 Summary 28
CHAPTER 3 ADDRESSING SPATIAL DEPENDENCY CHALLENGE: MAPPING ROAD SAFETY FEATURES FROM STREETVIEW IMAGERY: A DEEP LEARNING APPROACH 30
3.1 Introduction 30
3.2 Related Works 33
3.2.1 Transportation Safety 33
3.2.2 Deep Learning for Spatio-Temporal Data 34
3.3 Problem Description 34
3.4 Approach 36
3.4.1 Extract Image Feature with CNN 36
3.4.2 Model Spatial Linear Pattern with LSTM 39
3.4.3 Multi-label Classification 41
3.5 Experimental Evaluation 41
3.5.1 Dataset Description 43
3.5.2 Hyperparameter Settings 44
3.5.3 Classification Performance 45
3.5.4 Case Study 52
3.6 Summary 54

CHAPTER 4 ADDRESSING LIMITED OBSERVATION CHALLENGE: SPATIAL CLASSIFICATION WITH LIMITED OBSERVATIONS BASED ON PHYSICS-AWARE STRUCTURAL CONSTRAINT 55

4.1 Introduction 55

4.2 Problem Statement 58

4.2.1 Preliminaries 58

4.2.2 Problem Definition 59

4.3 Approach 59

4.3.1 Physics-Aware Structural Constraint 60

4.3.2 Model Probabilistic Formulation 61

4.3.3 Model Parameter Learning and Class Inference 63

4.3.4 How the Model Address Limited Observations? 65

4.4 Experimental Evaluation 66

4.4.1 Experiment Setup 66

4.4.2 Classification Performance Evaluation 68

4.4.3 Computational Performance Evaluation 71

4.5 Summary 72

CHAPTER 5 ADDRESSING SPATIAL DEPENDENCY CHALLENGE: a HIDDEN MARKOV CONTOUR TREE MODEL FOR SPATIAL STRUCTURAL PREDICTION 74

5.1 Introduction 74

5.2 Problem Statement 77

5.2.1 Preliminaries 77

5.2.2 Formal problem definition 78

5.3 Proposed Approach 79

5.3.1 Modeling Spatial Contour Structure 79

5.3.2 Hidden Markov Contour Tree (HMCT) 80

5.3.3 HMCT Learning and Inference 82
5.3.4 Computational Performance Tuning 87
5.3.5 Using HMCT for Post-processing 89
5.4 Evaluation 94
5.4.1 Classification Performance Comparison 95
5.4.2 Computational Performance Comparison 99
5.4.3 Evaluation of HCMT as Post Processing 103
5.5 Summary 109
CHAPTER 6 CONCLUSION AND FUTURE WORKS 110
REFERENCES 112

LIST OF TABLES

2.1 Comparison of filter and refinement on synthetic data (time in secs) 26

2.2 Comparison of filter and refinement on real dataset (time in secs) 26

3.1 Class distribution 44

3.2 Classification on Oxford dataset 47

3.3 Classification on Tuscaloosa dataset 47

4.1 Class transition probability and prior probability 63

4.2 Dataset description 67

4.3 Comparison on Mathew, Kinston flood data 68

4.4 Comparison on Mathew, Grimesland flood data 68

4.5 Time costs analysis of my model (seconds) 72

5.1 Class transition probability and prior probability 82

5.2 Classification on real dataset in Greenville, NC 96

5.3 Classification on real data in Grimesland, NC 98

5.4 Time costs of individual components (seconds) 102

5.5 HMCT as post-processing on Greenville dataset 103

5.6 HMCT as post-processing on Grimesland dataset 103

LIST OF FIGURES

1.1 Spatial data types 1

2.1 A problem example with inputs and outputs. (a) Input spatial features and instances; (b) Candidate and prevalent colocation patterns, instance tables . 13

2.2 Grid-aggregate based Upper Bound Filter: (a) A regular grid (b) An execution trace of upper bound filter 18

2.3 Illustrative execution trace for grid-based refinement 21

2.4 Illustrative execution trace for hash-join-based refinement 23

2.5 Experiment setup (a) Experiment design (b) Synthetic dataset (clumpiness of 1) (c) Synthetic dataset (clumpiness of 2). N_1 and N_2 are noise. 23

2.6 Results on synthetic datasets: (a) Effect of the number of instances with clumpiness as 1 (b) Effect of the number of instances with clumpiness as 20 . 25

2.7 Effect of clumpiness with the number of instances as 250k in synthetic dataset 26

2.8 Results on real world dataset: (a) Pruning ratio versus prevalence thresholds (b) Speedup versus prevalence thresholds 27

3.1 Three common classes of road safety features on Google Streetview Imagery. (a) Rumble strips (b) Concrete barrier (c) Metal crash barrier 31

3.2 Overall framework of the deep learning approach 37

3.3 LSTM unit 39

3.4 Training and validation performance of CNN-sharedLSTM over 50 epochs . 45

3.5 Training and validation performance of CNN-separateLSTM for rumble strips over 50 epochs 45

3.6 Training and validation performance of CNN-separateLSTM for metal crash

barriers over 50 epochs 45

3.7 Training and validation performance of CNN-separateLSTM for concrete barriers over 50 epochs 46

3.8 Prediction maps on Oxford dataset for (a) Rumble strips, (b) Metal crash barriers and (c) Concrete barriers respectively 48

3.9 Prediction maps on Tuscaloosa dataset for (a) Rumble strips, (b) Metal crash barriers and (c) Concrete barriers respectively 49

3.10 Classification map for rumble strips using CNN-separateLSTM model 51

3.11 Classification map for metal crash barriers using CNN-separateLSTM model 52

3.12 Classification map for concrete barriers using CNN-separateLSTM model . . 53

4.1 An illustration problem example (dry in green, flood in orange) 60

4.2 Illustration of partial order class dependency 61

4.3 Illustration of model structure 62

4.4 Illustration of message propagation in split tree 64

4.5 Sensitivity of my model to initial parameters π and ρ 70

4.6 Parameter iterations and convergence in my model 71

4.7 Computational performance of out model on varying test region sizes 72

5.1 Example of raster framework and flow dependency 78

5.2 Illustration of hidden Markov contour tree 80

5.3 Illustration of message propagation leaves to root 85

5.4 Illustration of message propagation leaves to root 85

5.5 An illustrative example of the discriminative version of HMCT 90

5.6 Sensitivity of HMCT to initial parameters π and ρ 100

5.7 Parameter iterations and convergence in HMCT 100

5.8 Computational performance of HMCT-C versus HMCT-UC on varying: (a) Test region sizes (b) Number of iterations (c) Maximum node in-degree . . . 101

5.9 Sensitivity of HMCT-PP+DT to π and ρ 104

5.10 Sensitivity of HMCT-PP+MLC to π and ρ 104

5.11 Sensitivity of HMCT-PP+RF to π and ρ 105

5.12 Sensitivity of HMCT-PP+GBM to π and ρ 105

5.13 Sensitivity of HMCT-PP+U-Net to π and ρ 106

5.14 Results on Greenville flood mapping (flood in brown, dry in green) 106

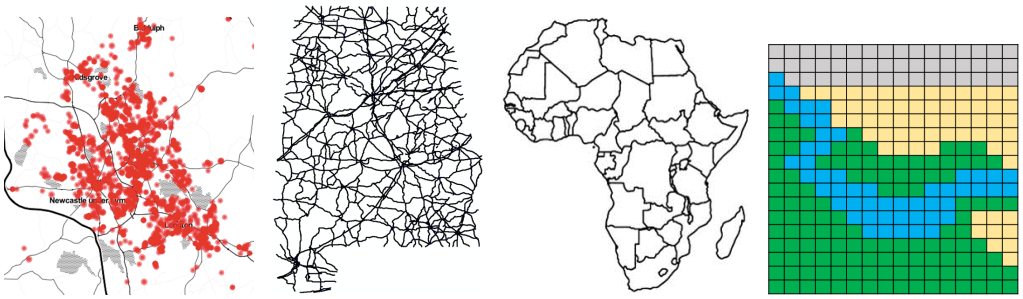
5.15 Results on Grimesland flood mapping (flood in brown, dry in green) 107

CHAPTER 1

INTRODUCTION

1.1 Geospatial Data Science

Spatial data are geo-referenced data, that have location information associated with it. Spatial data can be represented using two different models: the object model and the field model. The object model represents spatial data as discrete, identifiable entities such as points, lines, and polygons. Points can be used to represent an object or event on a map. Examples are cities in a map and locations of crimes (Figure 1.1(a)). Similarly, lines can represent the road segments (Figure 1.1(b)). Polygons can represent country boundaries (Figure 1.1(c)).The field model represents spatial data as raster data (Figure 1.1(d)). Raster data are structured as a grid of cells and each cell stores non-spatial attribute values.



(a) Points [69] (b) Lines [23] (c) Polygons [94] (d) Rasters

Figure 1.1: Spatial data types

Geospatial data science studies the process of discovering interesting, previously unknown, but potentially useful patterns from large spatial datasets. Spatial data can be used to extract important spatial patterns such as spatial colocation patterns, spatial hotspots, predictive models, spatial outliers, spatial clusters, and spatial change patterns. Given a set

of spatial features and their instances, colocation mining aims to find subsets of features whose instances are frequently located together. Examples of colocation patterns include symbiotic relationships between species such as Nile Crocodiles and Egyptian Plover, as well as environmental factors and diseases (e.g., air pollution and lung cancer). Colocation mining is important in many application that aims to find associations between spatial features. Similarly, given a set of spatial features and their instances, spatial hotspot detection aims to find the areas that have higher concentration of instances of a target spatial feature compared to its surroundings. Spatial hotspot analysis can be beneficial to identify areas with a high number of crimes. Given a spatial raster framework along with explanatory features layer, as well as a set of training samples with class labels outside the framework, spatial prediction aims to train a model to predict a class layer with the objective of minimizing the classification errors. Spatial prediction models can be used to identify flood extent based on remote sensing imagery. Based on the type of spatial data and the choice of output patterns, we can determine an appropriate geospatial data science algorithm.

1.2 Societal Applications

Organizations such as Transportation Institute, USGS, NASA, Police Department can benefit from geospatial data science by using it to identify patterns that can help in important decision-making processes. Moreover, geospatial data science techniques can be used to address grand societal challenges such as crime management, disease control, disaster management, and infrastructure management. In public safety, police departments use spatial hotspot detection tools to analyze crime patterns. These tools can help develop crime prevention strategies. In public health, colocation mining can be used to identify the association between diseases and environmental factors. For example, the West Nile virus, some bird species (crows and jays), and mosquitoes have a strong spatial association. West Nile virus is transmitted to birds through a bite of infected mosquitoes. Iden-

tifying such colocation patterns can help the Centers for Disease Control and Prevention (CDC) to develop an effective plan to control the spread of infectious diseases. In disaster management, the Federal Emergency Management Agency (FEMA) can use spatial classification models to enhance the situational awareness for disaster management. For example, during a major flood event such as during Hurricane Harvey floods in 2017, first responders needed to know where floodwater was in order to plan rescue effort. The traditional approach of sending a field crew on the ground to record high-water marks, or visually inspecting earth observation imagery can be time-consuming and expensive. Spatial predictive models can be used to quickly generate flood maps, which can be cost-effective and ultimately help save lives. Similarly, in transportation, mapping the pattern of roadside safety features along the major highway networks can help transportation engineers to better maintain roadside infrastructures and identify the areas where new infrastructures should be implemented.

Geospatial data science techniques can provide efficient and cost-effective solutions to solve major societal challenges. Geospatial data science techniques have an interdisciplinary nature and require an understanding of underlying physics or theories in their application domain. Moreover, domain experts are required to validate the spatial patterns. For example, interpreting the association between crimes in crime colocation patterns requires collaboration with domain experts on crime analysis.

1.3 Challenges

Geospatial data science poses several computational challenges that are not well-addressed by traditional data science techniques.

1.3.1 Spatial Dependency

Tobler's first law of geography states that "Everything is related to everything else, but near things are more related than distant things" [96]. For example, similar kinds of trees or vegetation tend to cluster together. People with similar kinds of occupations and back-

grounds tend to live in the close neighborhood. In spatial statistics, such phenomenon is refered to as spatial autocorrelation, which voilates most of the existing statistical assumption of an independent and identical distribution (i.i.d.). Ignoring spatial dependency and assuming i.i.d. when analyzing spatial data may produce inaccurate and inconsistent results. Furthermore, as spatial data are embedded in continuous space, many classical data mining techniques assume discrete data (e.g. transaction in association rule mining) may not be effective (e.g. breaking neighboring locations into different transactions).

1.3.2 Large Data Volumes

Another challenge is the large data volume. For example, satellite based earth observation data volume is increasing at the rate of several Terabytes per day [99]. The size of spatial data is rapidly increasing because of recent advancements in smart phones, GPS, and remote sensing technologies. In some cases, the input data may not be large, but the size of the intermediate results in pattern mining can be enormous. For example, mining colocation patterns from big spatial data require us to generate the candidate colocation patterns, which can be exponential to the number of spatial features. Moreover, in order to determine if a candidate colocation pattern is prevalent, we need to generate its pattern instances. This is computationally expensive due to checking spatial neighborhood relationships between different instances, particularly when the number of instances is large and clumpy (e.g., many instances are within the same spatial neighborhoods).

1.3.3 Multiple Spatial Scales

Spatial scale refers to the extent of an area at which a spatial pattern occurs. Different spatial objects may exhibit different spatial scale. In roadside safety features mapping problem, the spatial scales of road safety features may vary across different class categories. For example, concrete barriers are often very long (e.g., miles). In contrast, metal crash barriers are much shorter (e.g., hundred meters). Ignoring spatial scale may produce inaccurate and inconsistent results.

1.3.4 Limited Observations

In practice, sometimes it can be difficult or even impossible to get the complete observation of spatial features in the area under study due to the presence of some obstacles (e.g., clouds). For example, to extract flood patterns using earth observation images, there can be limited observations on samples in the raster framework due to only a subset of sensors being deployed in certain regions or most observations are obscured by clouds. In other words, only a subset of samples has complete explanatory feature values, making it hard to predict classes for all locations. Moreover, among the sample pixels with complete explanatory feature values, their feature values may contain rich noise and obstacles. For example, high-resolution earth imagery often has noise such as shadows.

1.4 Summary of Contributions

To address the above challenges, this thesis makes the following contributions.

- **Chapter 2** discusses a novel parallel spatial colocation pattern mining algorithms on GPUs to address the challenges of spatial large data volume. Given a set of spatial features and their instances across different locations, spatial neighborhood distance threshold and minimum intrestingness measure threshold, spatial colocation pattern mining aims to extract all colocation patterns whose interestingness measure value is equal or above the interestingness measure threshold. To improve computational efficiency, I introduce a filter and refine based strategy. My algorithms include a novel cell-aggregate based upper bound filter and two parallel implementations of refinement algorithms. The proposed cell-aggregate-based filter computes upper bounds of the interest measure of colocation patterns based on aggregated counts of event instances in neighborhood cells. The upper bound can be computed in parallel on GPU without generating pattern instances and is also insensitive to pattern clumpiness (the average number of overlaying colocation instances for a given colocation instance). I also provide a theoretical analysis of the correctness and completeness of

proposed algorithms. I conducted extensive experimental evaluations on both real-world and synthetic data with various parameter settings. Experimental results show that proposed GPU algorithms are promising with over 30 times speedup on up to millions of instances.

- **Chapter 3** introduces a deep learning framework to automatically map the road safety features from streetview imagery. Accurate maps of road safety features are an important component of safety management systems for federal or state transportation agencies, helping traffic engineers to identify locations where new infrastructure should be implemented. In current practice, mapping road safety features is largely done manually (e.g., observations on the road or visual interpretation of streetview imagery), which is both expensive and time-consuming. Mapping the road safety features based on streetview imagery poses several unique challenges. First, streetview images are not independent and identically distributed along the road network. In contrast, the safety feature types of consecutive images along the same road network path often resemble each other (also called the spatial dependency). Second, the spatial scales of road safety features may vary across different class categories. For example, concrete barriers are often very long (e.g., miles). In contrast, metal crash barriers are much shorter (e.g., hundred meters). Third, individual images may be imperfect due to some noise or obstacles. For example, a safety feature can be blocked by a large vehicle and thus become invisible in an image. To address these challenges, I introduce my recently proposed deep learning-based model on both convolution and recurrent units. I use convolutional neural network (CNN) model to extract semantic features from individual images. I also use a recurrent neural network, Long Short-Term Memory (LSTM), to model spatial sequential structure on extracted features from consecutive images along a road network path (spatial dependency). The integration of CNN and LSTM enables my deep learning model to utilize not only the content of individual images but also the geographic context be-

tween images. Evaluations on real-world streetview images collected from highways in Alabama show that my approaches outperform several baseline methods in classification performance.

- **Chapter 4** discusses a novel spatial classification models when there are only limited observations with an incomplete spatial coverage. In flood extent classification problems, sometimes we may have only a limited number of ground sensors being deployed in the flooded area, or only a small amount of data from remote sensors are free of obscurity, such as clouds. For example, a drone used during a flood disaster can only collect spectral images covering limited areas. Most of the existing spatial classification methods assume that incomplete features are rare in observations. Thus, they cannot adequately be applied to our problem, where the vast majority of samples have missing features (limited feature observations). To address the limitations, I introduce spatial classification model based on physics-aware structural constraints. My approach assumes that a spatial contextual feature (e.g., geographic terrain from elevation) is observed for all sample locations, based on which we established spatial structural constraints. I evaluated the proposed model on two real-world hydrological datasets. Results show that my approach significantly outperformed several baseline methods in classification performance, and that the proposed solution is computationally efficient on a large data volume.

- **Chapter 5** discusses hidden Markov contour tree (HMCT), which uses a contour tree structure to capture directed spatial dependency of flow directions between all locations on terrains. I develop efficient algorithms for model parameter learning and class inference. I also extend the HMCT model into a post-processor that can enhance the classified results from other models based on the topography constraints. Evaluations on real-world data show that my HMCT algorithms are scalable to large data sizes, and can achieve higher classification performance over existing methods for hydrological applications such as flood mapping.

CHAPTER 2

ADDRESSING THE LARGE DATA VOLUME CHALLENGE: GRID-BASED COLOCATION MINING ALGORITHMS ON GPU FOR BIG SPATIAL EVENT DATA: A SUMMARY OF RESULTS

2.1 Introduction

The focus of this chapter is on extracting colocation patterns from large geographic event datasets. Given a set of spatial features and their instances, colocation mining seeks to uncover subsets of spatial features whose instances are commonly situated together. Examples of colocation patterns include symbiotic relationships between species such as anemone and clownfish, as well as environmental factors and disease events (e.g., air pollution and lung cancer).

Societal applications: Colocation mining is useful in many applications that seek to discover relationships between different spatial events or factors. For example, in the field of public safety, law enforcement authorities are looking for connections between different types of crime events and possible crime generators. Ecologists study the interactions and spatial distributions of distinct species by analyzing their common geographical footprints. Similarly, identifying links between human disease and potential environmental factors is a major challenge in public health. Colocation patterns are used in climate science to identify links between the incidence of various climate extreme events. In a location-based service, colocation patterns aid in the identification of travelers who have similar favorite destinations, allowing for more effective tour selection.

Challenges: Mining colocation patterns from big spatial event data poses several computational challenges. First, in order to evaluate if a candidate colocation pattern is prevalent,

we need to generate its instances. This is computationally expensive due to checking spatial neighborhood relationships between different instances, particularly when the number of instances is large and instances are clumpy (e.g., many instances are within the same spatial neighborhoods). Second, the number of candidate colocation patterns are exponential to the number of spatial features. Evaluating a large number of candidate patterns can be computationally prohibitive. Finally, the distribution of event instances in the space may be uneven, making it hard to design parallel data structure and algorithms.

Related work: In the literature, colocation pattern mining has been extensively studied. Some of the early works includes spatial association rule mining [52, 62] and colocation patterns based on event-centric model [35]. Various algorithms have been proposed to efficiently identify colocation patterns, including Apriori generator and multi-resolution upper bound filter [35], partial join [120] and joinless approach [119], iCPI tree based colocation mining algorithms [15]. There are also works on identifying regional [61, 106, 58] or zonal [19] colocation patterns, and statistically significant colocation patterns [10, 11, 12], top-K prevalent colocation patterns [118] or prevalent patterns without thresholding [36]. Existing algorithms are mostly sequential, and can be insufficient when the number of event instances is very large (e.g., several millions). Recently, parallel colocation mining algorithms have been proposed based on the Map-reduce framework [117] to handle a large data volume. However, these algorithms need a large number of nodes to scale up, which is economically expensive, and their reducer nodes have a bottleneck of aggregating all instances of the same colocation patterns. Another work proposes a GPU based parallel colocation mining algorithm [2] using iCPI tree [3, 116, 4] and the joinless approach, but this method assumes that the number of neighbors for each instance is within a small constant (e.g., 32), and thus can be inefficient when instances are dense and unevenly distributed.

To address limitations of related work, I introduce a novel GPU colocation mining algorithms based on a grid index, including a cell-aggregate-based upper bound filter and two

refinement algorithms [83] [80]. Cell-aggregate-based filter is easier to implement on GPU and is also insensitive to pattern *clumpiness* (the average number of overlaying colocation instances for a given colocation instance) compared with the existing multi-resolution filter [35]. I use a GPU platform due to its better energy efficiency and pricing compared to Map-reduce based clouds.

Contributions: I make the following contributions in this chapter: (1) I provide the design and implementaion of parallel colocation mining algorithms on GPU, including a novel cell-aggregate based upper bound filter that is easier to implement on GPU and also insensitive to pattern clumpiness (i.e., number of colocation instances within the same neighborhood), two parallel refinement algorithms based on prefix-based HashJoin and grid search; (2) I provide theoretical analysis of the correctness and completeness of proposed algorithms; (3) I conduct extensive experimental evaluations on both real world and synthetic data with various parameter settings. Preliminary results show that proposed GPU algorithms are promising with over 30 times speedup on up to millions of instances.

Scope and outline: I focus on spatial colocation patterns defined by the event-centric models [35]. Other colocation definitions such as Voronoi diagram based are beyond the scope of this chapter. I also assume the underlying space is Euclidean space. Also, in this chapter, I am only concerned with the comparison of computational performance of various colocation mining algorithms.

The outline of the chapter is as follows. Section 2.2 reviews basic concepts and the definition of the colocation mining problem. Section 2.3 introduces the GPU colocation pattern mining algorithms, and analyzes the theoretical properties of algorithm correctness and completeness. Section 2.4 summarizes the experimental evaluation of algorithms on both synthetic datasets and a real world dataset. Section 2.5 discusses memory bottleneck issues in my approach and concludes the chapter.

2.2 Problem Statement

2.2.1 Basic Concepts

This subsection reviews some basic concepts based on which the colocation mining problem can be defined. More details on the concepts are in [35].

Spatial feature and instances: A *spatial feature* is a categorical attribute such as a crime event type (e.g., assault, drunk driving). For each spatial feature, there can be multiple *feature instances* at the same or different point locations (e.g., multiple instances of the same crime type "assault"). In the example of Figure 2.1(a), there are three spatial features (A, B and C). For spatial feature A, there are three instances (A_1, A_2, and A_3). Two feature instances are *spatial neighbors* if their spatial distance is smaller than a threshold. Two or more instances form a *clique* if every pair of instances are spatial neighbors.

Spatial colocation pattern: If the set of instances in a clique are from different feature types, then this set of instances is called a *colocation (pattern) instance*, and the corresponding set of features is a *colocation pattern*. The *cardinality* or *size* of a colocation pattern is the number of features involved. For example, in Figure 2.1(a), (A_1, B_1, C_1) is an instance of colocation pattern (A, B, C) with a size or cardinality of 3. If we put all the instances of a colocation pattern as different rows of a table, the table is called an *instance table*. For example, in Figure 2.1(b), the instance table of colocation pattern (A, B) has three row instances, as shown in the third table of the bottom panel. A spatial colocation pattern is *prevalent* (significant) if its feature instances are *frequently* located within the same neighborhood cliques. In order to quantify the prevalence or frequency, an interestingness measure called participation index has been proposed [35].

The *participation ratio* of a spatial feature within a candidate colocation pattern is the ratio of the number of unique feature instances that participate in colocation instances to the total number of feature instances. For example, in Figure 2.1, the participation ratio of B in candidate colocation pattern $\{A, B\}$ is $\frac{2}{3}$ since only B_1 and B_2 participate into colocation instances ($\{A_1, B_1\}$, $\{A_3, B_2\}$). The *participation index* (PI) of a candidate

colocation pattern is the minimum of participation ratios among all member features. For example, the participation index of the candidate colocation pattern $\{A, B\}$ in Figure 2.1 is the minimum of $\frac{3}{3}$ and $\frac{2}{3}$, and is thus $\frac{2}{3}$. We use "candidate colocation patterns" to refer to those whose participation index values are undecided. PI is used as the measure of prevalence of a colocation pattern because it follows apriori property. It basically implies that if the pattern is not prevalent then any of its superset pattern will also not be prevalent which can be exploited for computational efficiency.

2.2.2 Problem Definition

I now introduce the formal definition of colocation mining problem [35].

Given:

• A set of spatial features and their instances

• Spatial neighborhood distance threshold

• Minimum threshold of participation index: θ

Find:

• All colocation patterns whose participation index are above or equal to θ

Objective:

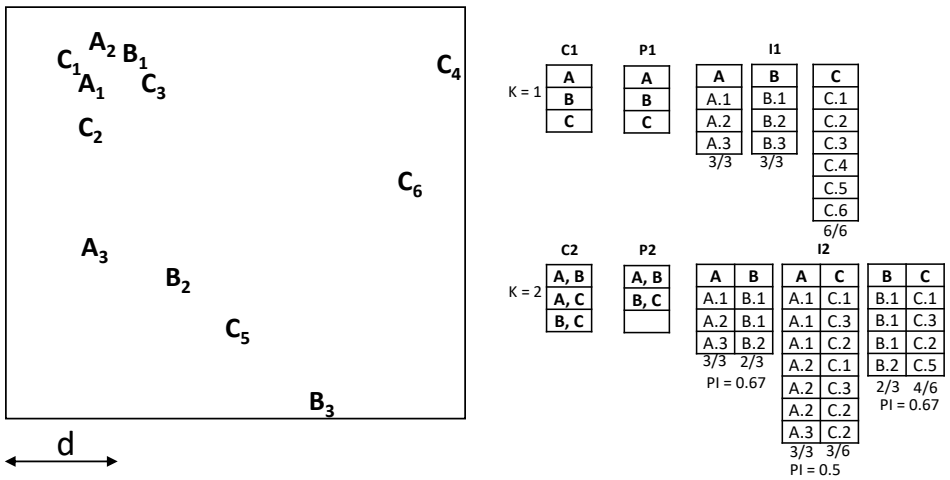
• Minimize computational time cost

Constraint:

• Spatial neighborhood relationships are defined in Euclidean space

Figure 2.1 provides a problem example. The input data contains 12 instances of 3 spatial features A, B, and C. The neighborhood distance threshold is d. The prevalence threshold is 0.6. The output prevalent colocation patterns include $\{A, B\}$ (participation index 0.67) and $\{B, C\}$ (participation index 0.67). Colocation mining is similar to association rule mining in market basket analysis [1], but is different in that there are no given "transactions" in continuous space. Generation colocation instance tables ("transactions") is the most computationally intensive part.

My baseline approach [35] uses the filter and refine strategy for colocation pattern min-



(a) (b)

Figure 2.1: A problem example with inputs and outputs. (a) Input spatial features and instances; (b) Candidate and prevalent colocation patterns, instance tables

ing. They proposed multiresolution filter that generates the coarse instance table for each candidate colocation pattern without performing any computationally expensive distance computation. The coarse instance table contains the entire actual as well as some false instances of the candidate colocation pattern. We can use this coarse instance table to compute the coarse participation index. Coarse participation index is the upper bound of the actual participation index for a given candidate colocation pattern. Hence, if the upper bound coarse participation index is lower than the prevalence threshold, we do not have to generate the computationally intensive instance table to calculate the actual participation index.

2.3 Approach

This section introduces proposed GPU colocation mining algorithm. I start with the overview of algorithm structure, and then describe the main part for parallel algorithms implemented in GPU. Next, I prove the correctness and completeness of the algorithm.

2.3.1 Algorithm Overview

The overall structure of the proposed algorithm is similar to the one proposed by Huang et al. in 2004 [35]. I design a novel upper-bound filter based on aggregated counts of feature instances in grid cells. Compared with the multi-resolution filter [35], the upper bound filter is easier to parallelize on GPU and does not rely on the assumption that colocation instances are clumpy into a small number of cells.

The overall structure of my algorithm is shown in Algorithm 1. The algorithm identifies all prevalent colocation patterns iteratively. Candidate colocation patterns and their instance tables of cardinality $k+1$ are generated, based on prevalent patterns and their instance tables of cardinality k. Each candidate pattern of cardinality $k+1$ is then evaluated based on the participation index computed from its instance table. For cardinality $k = 1$, prevalent colocation patterns simply consist of the set of input features, and their instance tables are the instance lists for each feature (step 1 in Algorithm 1). Step 2 generates candidate patterns C_{k+1} of size $k+1$ based on prevalent patterns P_k using Apriori property [1] (i.e., a candidate pattern of size $k+1$ cannot be prevalent and thus needs not to be generated if any subset pattern of size k is not prevalent). Step 4 builds a grid index on spatial point instances with the cell size equal to the distance threshold. Step 5 counts the number of instances for each feature in every cell. This will be used in the upper bound filter. Step 7 starts the iteration. As long as the set of candidate patterns C_{k+1} is not empty, the algorithm evaluate each candidate pattern $c \in C_{k+1}$. When evaluate a candidate pattern, the algorithm first computes an upper bound of its participation index in parallel using GPU kernels based on the grid index (step 9-10). If the upper bound is below the threshold, the candidate pattern is pruned out. Otherwise, the algorithm runs into a refinement phase, generating the pattern instance table $I_{k+1}.c$ and computing the participation index PI. I design two different parallel refinement algorithms to speed up instance table generation: one using the grid to rule out unnecessary joins, the other using prefix-based hash joins (steps 13 to 16). After all prevalent patterns of cardinality $k+1$ are identified, the al-

gorithm go to the next iteration (steps 19-22). Figure 2.1(b) illustrates the execution trace for $k = 1$ and $k = 2$.

Algorithm 1 Parallel-Colocation-Miner

Input: A set of spatial features F
Input: Instances of each spatial features $I[F]$
Input: Neighborhood distance threshold d
Input: Minimum prevalence threshold θ
Output: All prevalent colocation patterns P

1: Initialize $P \leftarrow \emptyset$, $k \leftarrow 1$, $C_k \leftarrow F$, $P_k \leftarrow F$
2: Initialize $C_{k+1} \leftarrow$ aprioriGen(P_k, $k+1$)
3: Initialize $P_{k+1} \leftarrow \emptyset$
4: Initialize instance tables I_k ($k = 1$) by feature instances
5: Overlay a regular grid with cell size $d \times d$ (total N cells)
6: Compute $CountMap[N \times |F|]$ in one round instance scanning
7: **while** $|C_{k+1}| > 0$ **do**
8: **for each** $c \in C_{k+1}$ **do**
9: Initialize $PCountMap[N \times |c|] \leftarrow 0$
10: $Upperbound =$ParallelCellAggregateFilter($CountMap$,$PCountMap$,c)
11: **if** $Upperbound \geq \theta$ **then**
12: $BitMap \leftarrow 0$ //initialize bitmap for instances of each feature
13: **if** Hash Join Refinement **then**
14: $(I_{k+1}.c, PI) \leftarrow$ ParallelHashJoinRefine(I_k, c)
15: **else if** Grid Search Refinement **then**
16: $(I_{k+1}.c, PI) \leftarrow$ ParallelGridSearchRefine($I_k, CInstances, c, BitMap$)
17: **if** $PI \geq \theta$ **then**
18: $P_{k+1} = P_{k+1} \cup c$
19: $P \leftarrow P \cup P_{k+1}$ //add prevalent patterns to results
20: $k \leftarrow k+1$; $C_k \leftarrow C_{k+1}$; $P_k \leftarrow P_{k+1}$, $I_k \leftarrow I_{k+1}$ //prepare next iteration
21: $C_{k+1} \leftarrow$ aprioriGen(P_k, $k+1$)
22: $P_{k+1} \leftarrow \emptyset$
23: **return** P

2.3.2 Cell-Aggregate-Based Upper Bound Filter

The cell aggregate based upper bound filter first overlays a regular grid with its cell size equal to the distance threshold (shown in Figure 2.2), and then computes an upper bound of participation index based on aggregated counts of feature instances in cells. This filter is different from the multi-resolution filter [35] in that the computation of upper bound is not based on generating coarse scale colocation instance tables. There are two main

advantages of cell aggregate based filter on GPU: first, it is easily parallelizable and can leverage the large number of GPU cores; second, its performance does not rely on the assumption that pattern instances are clumpy into a small number of cells, which is required by the multi-resolution filter.

To introduce cell aggregate based filter, I define a key concept of **quadruplet**. A quadruplet of a cell is a set of four cells, including the cell itself as well as its neighbors on the right, bottom, and right bottom. For a cell that is located on the right and bottom boundary of the grid, not all four cells exist and its quadruplet is defined empty (these cells will still be covered by other quadruplets). For example, in Figure 2.2, the quadruplet of cell 0 includes cells $(0, 1, 4, 5)$, while the quadruplet of cell 15 is an empty set.

Based on the concept of quadruplet, we can check all potential colocation instances by examining all quadruplets. When examining a quadruplet, the filter computes the aggregated count of instances for every feature in the candidate pattern. If the aggregated count for any feature is zero, then there cannot exist colocation instances in the quadruplet. Otherwise, we pretend that all these feature instances participate into colocation pattern instances. This tends to overestimate the participating instances of a colocation pattern (an "upper bound"), but avoids expensive spatial join operations.

Algorithm 2 shows details of the cell aggregate based filter. The algorithms have three main variables, including $CountMap$, $PCountMap$, and $QuadrupletAggregate$. $CountMap$ records the true aggregated instance count for each feature in every cell. $PCountMap$ records the instance count for each feature in every cell that potentially participates in the candidate colocation pattern. $QuadrupletAggregate$ is a local array for each cell to record the aggregated count of instances within the quadruplet for each pattern feature. Specifically, steps 1 to 11 computes potential number of participating instances for each feature in each cell in parallel. A kernel thread is allocated to each cell. For a specific cell i, the kernel first gets the quadruplet (step 2). Step 3 initializes a local array $QuadrupletAggregate$ with zero values. Steps 4 to 8 compute the aggregated count of instances for each pat-

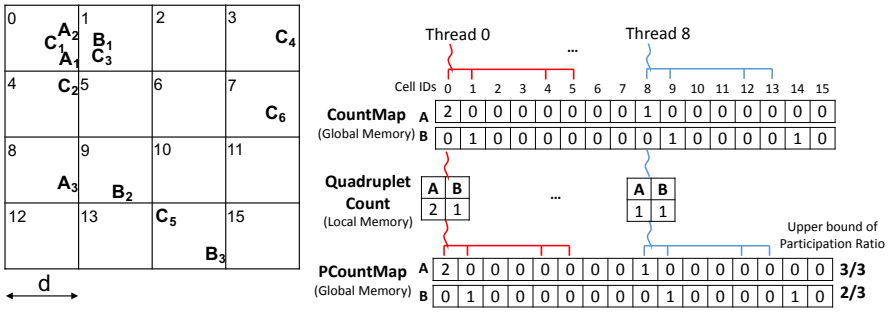
Algorithm 2 ParallelCellAggregateFilter

Input: $CountMap$, feature instance count in cells
Input: $PCountMap$, participating feature instance count in cells
Input: PR, participation ratio
Input: Candidate colocation pattern c
Output: Upper bound of participation index, $upperBound$

1: **for each** cell i **do in parallel**
2: $QuadrupletCells$ = GETQUADRUPLET(cell i)
3: $QuadrupletAggregate[|c|] \leftarrow 0$
4: **for each** feature $f \in c$ **do**
5: **for each** cell $j \in QuadrupletCells$ **do**
6: $QuadrupletAggregate[f] \leftarrow QuadrupletAggregate[f] + CountMap[j][f]$
7: **if** $QuadrupletAggregate[f] == 0$ **then**
8: **finish** the parallel thread for cell i //no pattern instance in the quadruplet
9: **for each** feature $f \in c$ **do**
10: **for each** cell $j \in QuadrupletCells$ **do**
11: $PCountMap[j][f] \leftarrow CountMap[j][f]$ //participating instance count
12: **for each** feature $f \in c$ **do**
13: $PR[f]$ = PARALLELSUM($CountMap[\ ][f]$)$/|I_1.f|$
14: $upperBound$ = MIN(PR)
15: **return** $upperBound$

tern feature ($QuadrupletAggregate[f]$). If aggregated count of any pattern feature is zero, then there cannot be any candidate pattern instance in the quadruplet and thus the parallel kernel thread terminates (step 8). Otherwise, all feature instances in the quadruplet can potentially participate into colocation pattern instances. Steps 9 to 11 record the potential participating instances from each cell in the quadruplet. This is done by copying instance counts from $CountMap$ to $PCountMap$ for the 4 cells in the quadruplet. It is worth noting that duplicated-counting on the same cell is avoided since different GPU kernel threads may over-write the count for a cell with the same value. Finally, steps 12 to 14 compute the upper bound of participation index based on counts of potential participating instances in $PCountMap$. I use built in GPU library to compute the total counts of distinct participating instances in step 13.

Figure 2.2 provides an illustrative execution trace of Algorithm 2. Figure 2.2(a) shows the input spatial instances overlaid with a regular grid. The distance threshold is d. Assume



(a) (b)

Figure 2.2: Grid-aggregate based Upper Bound Filter: (a) A regular grid (b) An execution trace of upper bound filter

that the candidate colocation pattern is (A, B). Figure 2.2(b) shows how the filter works. The $CountMap$ array stores the number of instances for feature A and B in each cell. A GPU thread is assigned to each cell to compute the counts of feature instances within its quadruplet. For example, the leftmost GPU thread is assigned to cell 0. The aggregated instance count for this quadruplet $((0, 1, 4, 5))$ is shown by the leftmost $QuadrupletCount$ array, with 2 instances for A and 1 instance for B. Since instances from both features exist, the number potential participating instances in these four cells ($PCountMap$) are copied from corresponding cell values in $CountMap$, as shown by the fork branches close to the bottom. In contrast, the quadruplet of cell 1 $((1, 2, 5, 6))$ does not contain instances of A, and thus cannot contain colocation pattern instances.

Lemma 2.3.1. *The participation index of a colocation pattern in the cell-aggregate-based filter is an upper bound of the true participation index value.*

Proof. The proof is based on the following fact. We create an upper bound to the true number of neighboring points in neighboring cells (quadruplet) by assuming that all pairs of points of neighboring cells are within the distance threshold, which coincides with the cell size. Of course, some of them will not, but it is impossible for points not within neighboring cells to be neighboring with respect to the distance threshold. □

Theorem 2.3.2. *The cell aggregate based upper bound filter is correct and complete.*

Proof. The proof is based on Lemma 2.3.1. The algorithm is complete (it does not mistakenly prune out any prevalent pattern) due to the upper bound property. The algorithm is correct since it computes the exact participation index of a candidate pattern if it passes the upper bound filter. □

2.3.3 Refinement Algorithms

The goal of the refinement phase is to generate the instance table of a candidate colocation pattern, and to compute participation index. Generating colocation instance tables is the main computational bottleneck, and thus is done in GPU. As shown in Algorithm 1, we have two options for refinement algorithms, a geometric approach based on grid search called ParallelGridSearchRefine and a combinatorics approach based on prefix-based hash join called ParallelHashJoinRefine, similar to sequential algorithms discussed in Huang et al. [35]. I now introduce the two algorithms below.

Algorithm 3 *ParallelGridSearchRefine*

Input: I_k, instance table of patterns of size k
Input: $CellInstances$, feature instances for each cell
Input: $BitMap$, bitmap for participating instances from different features
Output: $I_{k+1}.c$, instance table of colocation c (size $k+1$) if prevalent
Output: PI, participation index of pattern c

1: $//I_k.(c[1..k])$ is instance table of sub-pattern of c with first k features
2:
3: Initialize $I_{k+1}.c \leftarrow \emptyset$
4: **for each** row instance $rIns \in I_k.(c[1..k])$ **do in parallel**
5: get *neighborhood* cells of first feature instance in $rIns$
6: **for each** cell i in neighborhood **do**
7: **for each** instance ins of feature type $c[k+1]$ in cell i **do**
8: **if** ins is neighbor of all feature instances in $rIns$ **then**
9: Create new row instance of c, $rInsC =< rIns, ins >$
10: $I_{k+1}.c = I_{k+1}.c \cup rInsC$ //add new instance into c's instance table
11: **for each** feature $f \in c$ **do**
12: $BitMap[f][rInsC[f]] = \text{true}$
13: **for each** feature $f \in c$ **do**
14: $PR[f] = \text{ParallelSum}(BitMap[\ ][f])/|I_1.f|$
15: $PI = \text{Min}(PR)$
16: **return** $I_{k+1}.c$, PI

Geometric approach: The geometric approach generate an instance table of a size $k+1$ pattern based on the instance table of a size k pattern. For example, when generating the instance table of pattern (A, B, C), it starts from the instance table of (A, B) and joins each row of the table with instances of the last feature type C. In order to reduce redundant computation, I utilize the grid index and only check the instances of the last feature type within neighboring cells. Algorithm 3 provides details of the grid-based refinement algorithm. Step 2 is kernel assignment. Each kernel thread first finds out all neighboring cells of the size k row instance $rIns$ (step 3). Then, for each neighboring cell, the kernel thread finds out every instance ins of the last feature type in the cell. It joins ins with size k instance $rIns$ to create a size $k+1$ pattern instance $rInsC$ if they are spatial neighbors (steps 4 to 7). The new pattern instance $rInsC$ is inserted into the final instance table $I_{k+1}.c$, and a bitmap is updated to mark the instances that participate into the colocation pattern (steps 8 to 10). Finally, the participation ratio and participation index are computed (steps 11 to 13).

Figure 2.3 shows an example. The input data is the same as Figure 2.1. Assume that the candidate pattern is (A, B, C). A kernel thread is assigned to each row instance of table (A, B). Thread 1 is assigned to instance (A_1, B_1), and it scans all neighboring cells of A_1 (cells $0, 1, 4, 5$). Based on the cell to instance index, the kernel thread checks all instances of feature C (C_1, C_3, C_2) in these cells, and conducts a spatial join operation. The final output size $k+1$ instances from this thread are (A_1, B_1, C_1), (A_1, B_1, C_3), and (A_1, B_1, C_2). One issue in GPU implementation is that we need to allocate memory for an output instance table, and specify the specific memory location to which each kernel thread writes its results. For example, in the output instance table of pattern (A, B, C) in Figure 2.3, the first kernel thread generates 3 row instances, so the second kernel thread has to start with the $4th$ row when writing its instances. It is hard to predetermine the total required memory and enforcing memory coalesce when threads are writing results. Thus, I use a two-run strategy in which in the first run I can calculate the exact size of output in-

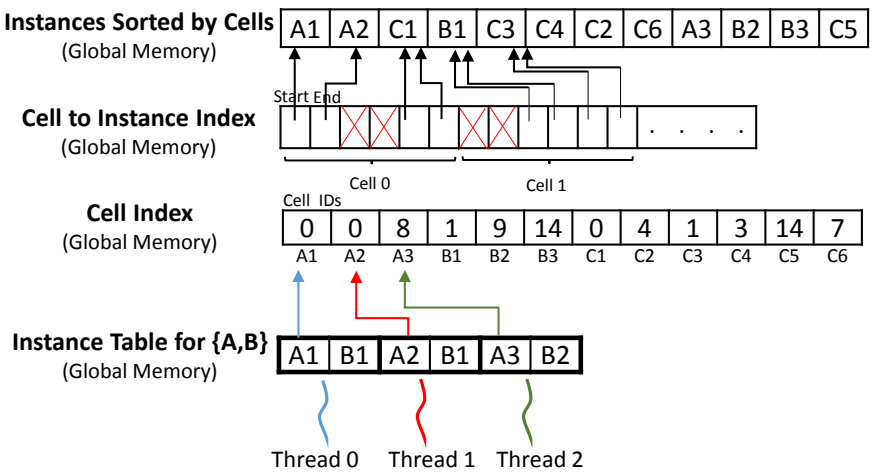


Figure 2.3: Illustrative execution trace for grid-based refinement

stance table as well as slot counts of the number of row instances generated by each kernel thread. In the second run, I allocate memory for output instance table, and use the slot counts to guide which row a kernel thread needs to start from when writing results. Similar to the grid-based refinement, I use two-run strategy to allocate memory and enforce memory coalesce.

Algorithm 4 *ParallelHashJoinRefine*

Input: I_k, instance table of patterns of size k
Input: $BitMap$, bitmap for instances of different features
Output: $I_{k+1}.c$, instance table of colocation c if prevalent
Output: PI, participation index of pattern c

1: $//I_k.c1$ and $I_k.c2$ instance tables of $c1 = c[1..k]$ and $c2 = c[1..k-1, k+1]$
2: **for each** row instance $rIns1$ in $I_k.c1$ **do in parallel**
3: **for each** row instance $rIns2$ in $I_k.c2$ starting with $rIns1[1]$ **do**
4: **if** $rIns1$ and $rIns2$ forms an instance of c **then**
5: Create new instance $rInsC$ by merging $rIns1$ and $rIns2$
6: $I_{k+1}.c \leftarrow I_{k+1}.c \cup rInsC$
7: **for each** feature $f \in c$ **do**
8: $BitMap[f].[rInsC[f]] = \text{true}$
9: **for each** feature $f \in c$ **do**
10: $PR[f] = \text{ParallelSum}(BitMap[\,][f])/|I_1.f|$
11: $PI = \text{Min}(PR)$
12: **return** $I_{k+1}.c$, PI

Prefix-based hash Join based refinement: Another option is to generate size $k+1$ instance

table by a combinatorics approach. For example, we can generate instance table of pattern (A, B, C) by joining rows in instance tables of (A, B) and (A, C). The join condition is that the first k instances from the two tables should be the same, and the last instances from two tables should be spatial neighbors. For example, when joining a row (A_1, B_1) with another row (A_1, C_1), we check that the first instance is the same (A_1), and the last instances B_1 and C_1 are spatial neighbors. So these two rows are joined to form a new row instance (A_1, B_1, C_1). In sequential implementations [35], the join process can be done efficiently through sort-merge join. However, for GPU algorithm, sort merge is difficult due to the order, dependency and multi-attribute keys. I choose to use hash join instead. A prefix-based hash index is built on the second table based on instances of the first spatial feature. Details are shown in Algorithm 4. A kernel thread is allocated to each row in the first size k instance table $I_k.c1$ (step 2). The kernel thread then scans all rows in the second size k instance table $I_k.c2$ that has the same first feature instance. For example, if the row in the first table is (A_1, B_1), then the thread only scans rows starting with A_1 in the instance table of (A, C). If the two rows satisfy the join condition (sharing the same first k instances, and having last instances as neighbors), a size $k + 1$ instance is created and inserted into output size $k + 1$ table (steps 5 to 8). Finally, the participation index is computed (steps 9 to 11). It is worth noting that when generating instance tables of size $k = 2$ patterns, I use the grid-based method since hash-index cannot be created in that case.
An illustrative execution trace is shown in Figure 2.4. The raw input data is still the same. Each kernel thread is allocated to a row in instance table (A, B). For example, thread 1 works on pattern instance (A_1, B_1), and scans instance table (A, C). Based on the hash index on A instances, the thread only needs to check (A_1, C_1), (A_1, C_3) and (A_1, C_2). It turns out that B_1 is a neighbor for all C_1, C_2 and C_3. So these instances are inserted to the final output instance table (A, B, C).

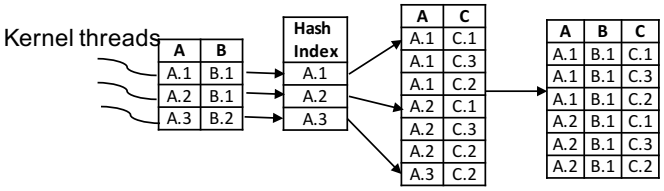


Figure 2.4: Illustrative execution trace for hash-join-based refinement

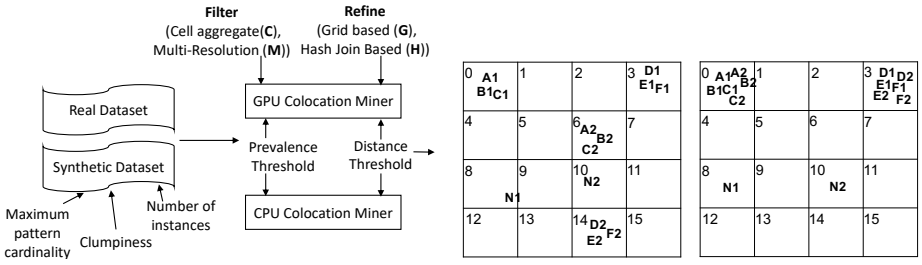


Figure 2.5: Experiment setup (a) Experiment design (b) Synthetic dataset (clumpiness of 1) (c) Synthetic dataset (clumpiness of 2). N_1 and N_2 are noise.

2.4 Evaluation

The goals of evaluation are to:

- Evaluate the speedup of GPU colocation algorithms against a CPU algorithm.
- Compare cell-aggregate-based filter with multi-resolution filter on GPU.
- Compare grid-based refinement with hash-join-based refinement on GPU.
- Test the sensitivity of GPU algorithms to different factors.

Experiment Setup: As shown in Figure 2.5, I implemented four GPU colocation mining algorithms with two filter options (**M** for multi-resolution filter and **C** for cell-aggregate based filter) and two refinement options (**G** for grid-based and **H** for hash-join based). I also implemented a CPU colocation mining algorithms by Huang et al. [35] (multi-resolution filter, grid-based instance table generation for size $k = 2$, and sort-merge based instance table generation for size $k > 2$). I only compared computational performance since all methods produce the same patterns. For each experiment, I measured the time cost of one run for CPU algorithm, and averaged time cost of 10 runs for GPU algorithms. Algorithms were implemented in C++ and CUDA, and run on a Dell workstation with In-

tel(R) Xeon(R) CPU E5-2687w v4 @ 3.00GHz, 64GB main memory, and a Nvidia Quadro K6000 GPU with 2880 cores and 12GB memory.

Dataset description: The real dataset contains 13 crime types and 165,000 crime event instances from Seattle in 2012 [21]. The synthetic data is generated similarly to [35]. Figure 2.5(b-c) provide illustrative examples. I first chose a study area size of 10000×10000, a neighborhood distance threshold (also the size of a grid cell) of 10, a maximal pattern cardinality of 5, and the number of maximal colocation patterns as 2. The total number of features was 12 (5×2 plus 2 additional noise features). I then generated a *number of instances* for each maximal colocation pattern. Their locations were randomly distributed to different cells according to the **clumpiness** (i.e., the number of overlaying colocation instances within the same neighborhood, higher clumpiness means larger instance tables). In the experiments, I varied the number of instances and clumpiness to test sensitivity.

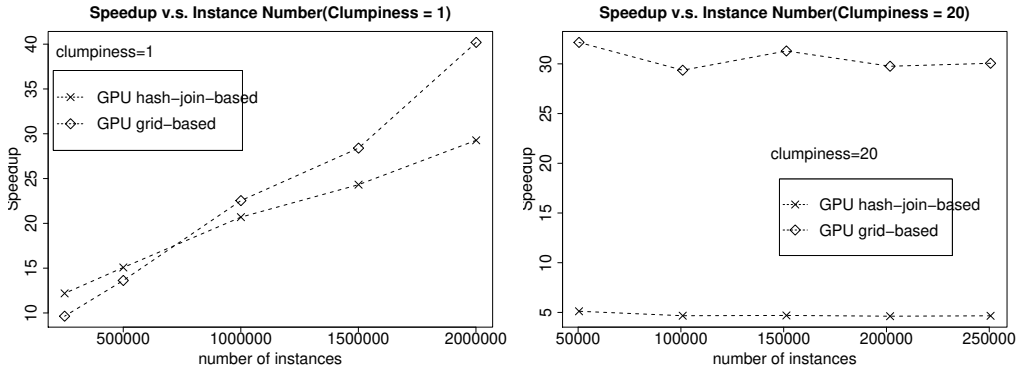
Evaluation metric: I used the speedup of the GPU algorithms over the CPU algorithm on computational time.

2.4.1 Results on Synthetic Data

Effect of the number of instances

I conducted this experiment with two different parameter settings. For both settings, the minimum participation index threshold was 0.5. In the first setting, I set the clumpiness to 1 (very low clumpiness), and varied the number of feature instances as 250,000, 500,000, 1,000,000, 1,500,000 and 2,000,000. Results are summarized in Figure 2.6(a). GPU algorithms in the plot are based on grid-based filtering. We can see that the speedup of both GPU algorithms increases with the number of feature instances. The grid-based refinement gradually becomes superior over the hash join based refinement in GPU algorithms as the number of instances increases. The reason can be that the cell-instance index in grid-based refinement is done once and for all, while the prefix-based hash index in hash-join based refinement needs to be created repeatedly for each new instance table. The comparison of two approaches with 250,000 instances (the first two points in the curve) may be less con-

clusive since the running time for both approaches is too small (far below one second). In the second setting, I set the clumpiness value as 20, and varied the number of feature instances as 50,000, 100,000, 150,000, 200,000, and 250,000. The number of feature instances were set smaller in this setting due to the fact that given the same number of feature instances, a higher clumpiness value results in far more colocation pattern instances (see Figure 2.5(b) versus (c)) but we only have limited memory. The results are summarized in Figure 2.6(b). We can see that the grid based refinement is persistently better than hash-join based refinement (around 30 versus 5). The reason is that when the clumpiness is high, there are a large number of pattern instances being formed combinatorially. Many of them share the same prefix (i.e., first feature instance). Thus, each GPU kernel thread in prefix-based hash-join refinement was loaded with heavy computation when doing the join operation, impacting the parallel performance. In contrast, in the grid-based refinement, each GPU kernel thread only scans a limited number of instances within neighboring cells.



(a) (b)

Figure 2.6: Results on synthetic datasets: (a) Effect of the number of instances with clumpiness as 1 (b) Effect of the number of instances with clumpiness as 20

Effect of Clumpiness

I set the number of instances to 250k, and the prevalence threshold to 0.5. Grid-based filtering was used for GPU algorithms. I varied the clumpiness value as 1, 5, 10, 15, and 20.

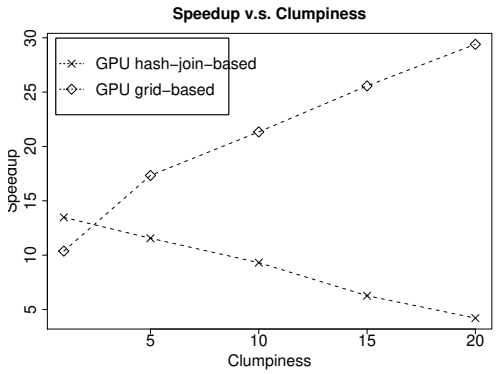


Figure 2.7: Effect of clumpiness with the number of instances as 250k in synthetic dataset

Result in Figure 2.7 confirm with my analysis above that when clumpiness is higher, the performance of grid-based refinement gets better while the performance of hash-join based refinement gets worse.

Table 2.1: Comparison of filter and refinement on synthetic data (time in secs)

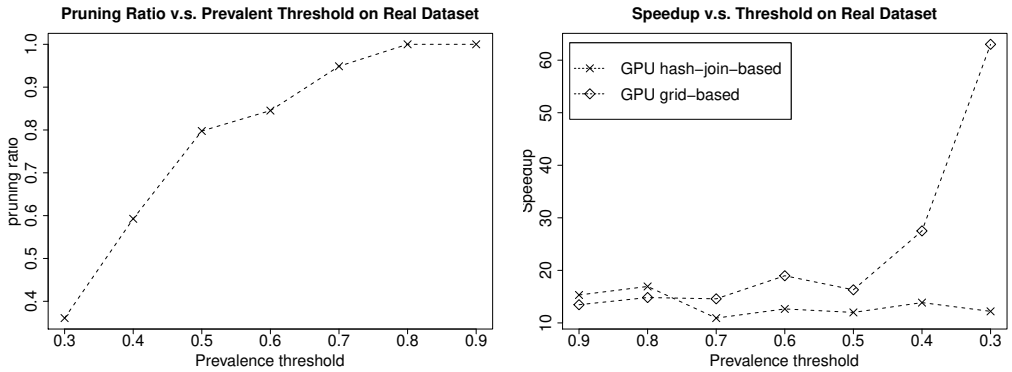
Clumpiness	Approaches	Filter Time	Refine Time	Total Time	Speedup
1	CPU Baseline	15.3	18.8	34.1	-
	GPU-Filter:M, Refine:H	0.8	1.1	1.9	17.9x
	GPU-Filter:C, Refine:G	**0.2**	**0.7**	**0.9**	**37.9x**
	GPU-Filter:C, Refine:H	**0.2**	**1.0**	**1.2**	**28.4x**
20	CPU Baseline	0.9	407.5	408.4	-
	GPU-Filter:M, Refine:H	**0.1**	97.3	97.4	4.2x
	GPU-Filter:C, Refine:G	**0.1**	**13.8**	**13.9**	**29.4x**
	GPU-Filter:C, Refine:H	**0.1**	96.9	97	4.2x

Comparison on filter and refinement

I also compared the computational time of filter and refinement phases of GPU algorithms in the above experiments for the cases with the largest number of instances. Details are

Table 2.2: Comparison of filter and refinement on real dataset (time in secs)

Approaches	Filter Time	Refine Time	Total Time	Speedup
CPU Baseline	6.5	340.9	347.4	-
GPU-Filter:M, Refine:H	0.2	26.	26.8	13x
GPU-Filter:C, Refine:G	**0.5**	**5.0**	**5.5**	**63.2x**
GPU-Filter:C, Refine:H	**0.5**	26.9	27.4	12.7x



(a) (b)

Figure 2.8: Results on real world dataset: (a) Pruning ratio versus prevalence thresholds (b) Speedup versus prevalence thresholds

summarized in Table 2.1. When clumpiness is 1, the grid-based filter is much faster than the multi-resolution filter in GPU algorithms (0.2 second versus 0.8 second), making the overall GPU speedup better (37.9 and 28.4 times versus 17.9 times). The reason is that a low clumpiness significantly impacts the multi-resolution filter (coarse scale instance tables cannot be much smaller than true instance tables), while the grid-based filter was less sensitive to clumpiness (more robust) since the time cost of grid-based filtering does not depend on instance distribution. When clumpiness is 20, the refinement phase becomes the bottleneck. The grid-based refinement has a significantly higher speedup than the hash-join refinement (29.4 times versus 4.2 times).

2.4.2 Results on Real World Dataset

Effect of minimum participation index threshold

I fixed the distance threshold as 10 meters and varied the prevalence thresholds from 0.3 to 0.9 (I did not chose thresholds lower than 0.3, because there would be too many instance tables, exceeding memory capacity). The clumpiness of the real dataset was high due to a large density of points. Results are summarized in Figure 2.8. As we can see, as the prevalence threshold gets higher, the pruning ratio (candidate patterns being pruned out) gets improved (Figure 2.8(a)). The GPU algorithm with grid based refinement is much better than the GPU algorithm with hash join based refinement. This is consistent

with the results on synthetic datasets when the clumpiness is high.

Comparison of filter and refinement

I also compared the detailed computational time in the filter and refinement phases. The distance threshold was 10 meters, and the prevalent threshold was 0.3. Results are summarized in Table 2.2. Due to a high clumpiness, the refinement phase is the bottleneck, and the grid-based refinement is better than the hash-join based refinement (63.2 times overall speedup versus 12.7 times overall speedup).

2.5 Discussion

Preliminary results show that GPU algorithms are promising for the colocation mining problem with above 30 times speedup. One limitation of the my GPU algorithm is memory bottleneck. My algorithm generates instance tables of candidate colocation patterns in GPU. When spatial points are dense (clumpy) and the number of points is large (e.g., millions), such instance tables can reach gigabytes in size. In my implementation, I generate one instance table each time in GPU global memory, and transfer the results to the host in pinned memory. Results show that the time cost of memory copy was significantly smaller (due to pinned memory) than the time cost of GPU computation. In case that the GPU global memory is insufficient for a very large instance table, we can slice it into smaller pieces, compute one piece each time, and transfer results to the host memory. We need to store all relevant instance tables of cardinality k in host memory when computing instance tables of cardinality $k + 1$. Thus, the host memory size can also be a bottleneck. This may be less a concern in future when the main memory price gets lower.

2.6 Summary

This chapter investigates GPU colocation mining algorithms. I introduce cell-aggregate-based upper bound filter, which is easier to implement on GPU and less sensitive to data

clumpiness compared with the existing multi-resolution filter. I also introduce two recent GPU implementations of refinement algorithms. I provide theoretical analysis on the correctness and completeness of proposed algorithms. Preliminary results on both real-world data and synthetic data on various parameter settings show that GPU algorithms are promising.

CHAPTER 3

ADDRESSING SPATIAL DEPENDENCY CHALLENGE: MAPPING ROAD SAFETY FEATURES FROM STREETVIEW IMAGERY: A DEEP LEARNING APPROACH

3.1 Introduction

Every year, around 6 million car accidents occur in the U.S. on average [51]. Traffic safety has long been an important societal issue. In order to avoid or mitigate vehicle crashes, traffic engineers place roadside barriers to prevent out of control vehicles from diverting off the roads and hitting the roadside hazards. Such road safety features can also prevent vehicles from crossing into the path of other vehicles. During winter season, vehicles can become more difficult to control on icy and slippery road surface, particularly when the vehicle speed is high. Barriers on the roadside can act as a safety precaution in such cases. Other safety features such as rumble strips help alert inattentive drivers who are deviating from their lanes. Figure 3.1 shows three different common type of road safety features, rumble strip, concrete barrier, and metal crash barrier. Federal, state and local governments spent several hundred billion dollars each year on transportation infrastructure development and maintenance [68]. Mapping safety features along road networks can play a crucial role in managing and maintaining road safety infrastructures. Traffic engineers can use the detailed safety feature map to identify locations where new safety infrastructure should be invested.

In current practice, mapping road safety features are mostly done manually by well-trained traffic engineers driving through road networks or visually interpreting streetview images. A **streetview image** is a geo-referenced image taken at a specific location on the ground. One common example is Google Streetview Imagery collected by vehicles equipped with

Figure 3.1: Three common classes of road safety features on Google Streetview Imagery. (a) Rumble strips (b) Concrete barrier (c) Metal crash barrier

GPS and cameras driving along streets on road networks. However, such a manual process is both expensive and time-consuming. Given the large amount of information to collect, the cost of these approaches quickly become prohibitive.

The focus of this chapter is to develop a deep learning algorithm that can automatically map road safety features from streetview imagery. The results can be used by the transportation agencies in management and maintenance of road safety infrastructures, as well as planning the investment on new infrastructures. Specifically, we can utilize a small set of manually labeled imagery (whose road safety features are visually inspected) to learn a classification model. Then the model can be used to classify safety feature types on a large number of unlabeled imagery along the road network.

However, mapping the road safety features based on streetview imagery poses several unique challenges. First, streetview images are not independent and identically distributed along a road network. In contrast, the safety feature types of consecutive images along a same road network path often resemble each other (also called the spatial dependency). Second, the spatial scales of road safety features may vary across different class categories. For example, concrete barrier is often very long (e.g., miles). In contrast, metal crash barriers are much shorter (e.g., hundred meters). Third, individual images may be imperfect due to some noise or obstacles. For example, a safety feature can be blocked by a large vehicle and thus become invisible in an image.

To address these challenges, I introduce my recently proposed deep learning model based on both convolutional and recurrent units [84]. I use covolutional neural network (CNN) model to extract semantic features from individual images. I also use a recurrent neural network, Long Short-Term Memory (LSTM), to model spatial sequential structure on extracted features from consecutive images along a road network path (spatial dependency). The integration of CNN and LSTM enables my deep learning model to utilize not only the content of individual images but also the geographic context between images. Evaluations on real world streetview images collected from highways in Alabama show that my approaches outperform several baseline methods in classification performance.
In summary, the contributions of this chapter are listed below:

- I propose to use integrated deep learning models that combine CNN and LSTM. The integrated models can utilize not only the content of individual images but also the spatial sequential structure between images.
- I compare my approaches with several baseline methods on two real world streetview imagery datasets collected in Alabama. I achieve 3 and 5 percent improvement in F-score over the best baseline method on two different test datasets.
- I perform a case study of mapping road safety features with 69,500 streetview imageries over all the major highways in the entire state of Alabama.

The outline of the chapter is as follows. Section 3.2 discusses some of the related works. Section 3.3 formally defines the problem. Section 3.4 introduces the approaches. Section 3.5 summarizes the results of the experimental evaluation on two real world datasets as well as discusses the case study of mapping road safety features over all the major highways in Alabama. Section 3.6 concludes the chapter.

3.2 Related Works

In this section, I briefly review the relevant research on transportation safety and deep learning techniques for spatial and spatiotemporal data.

3.2.1 Transportation Safety

Related work in transportation safety often focuses on analyzing the protective effect of different road safety features (e.g., roadside barriers) [49, 78, 129]. For example, studies in [8, 37, 31] quantify the protective effect of barriers with regards to motorcyclist injury. Work in [17] analyzes the performance of roadside barriers related to vehicle size and type. [100] studies how to increase the effectiveness of the roadside barriers in safety protection. For example, studies found that concrete barriers can hold high-energy truck crash, but can also cause more fatalities. Some recent work focuses on developing energy absorbing barrier [86]. Beside the protective effect, other studies on roadside barriers focus on the impact on mitigating near-road air pollution [97]. The study on effect of solid barriers on dispersion of roadway emissions in [87, 33] shows that roadside barriers is one of the most practical mitigation methods. There are also works that analyze spatial patterns from traffic accident event locations such as network hotspots and colocation patterns [75, 83, 80, 61]. [40] proposes efficient algorithms to identify primary corridors from cyclists' GPS trajectories on urban road networks to study riding behaviors for safety issues. [64] shows techniques to detect coarse scale hotspots of road failure events through geo-tagged tweets from social media.

Recently, researchers have used Google Streetview imagery along the road network for traffic sign detection for roadway inventory management [6, 7, 98]. Other works use streetview imagery to estimate the demographic makeup of neighborhoods [27], to assess street-level greenness in an urban area [56]. To the best of my knowledge, there is little research on utilizing streetview imagery to automatically map road safety features.

3.2.2 Deep Learning for Spatio-Temporal Data

The work in this chapter generally fall into the area of spatial classification or prediction [39, 45, 46, 47, 41, 43, 110, 48, 42] in spatiotemporal data mining [88, 44]. In recent years, deep learning techniques have shown great growth in this area. One common approach is to integrate deep convolutional neural networks (CNN) with recurrent neural networks such as Long Short-Term Memory (LSTM). The CNN component can be used to model spatial dependency structure in one temporal snapshot, while the LSTM component can be used to model temporal dynamics between different snapshots. For example, [126] uses fully convolutional networks with LSTM to estimate vehicle count maps based on city cameras. [114] uses CNN-LSTM model together with multi-view learning to predict taxi demand. [109] uses a one-dimensional CNN to capture spatial features of traffic flow and two LSTM models to capture the short-term variability and periodicities of traffic flow. [113] addresses the traffic prediction problem with a new spatiotemporal model. It uses a flow gating mechanism to learn the dynamic similarity between locations, and uses a periodically shifted attention mechanism to handle long-term periodic temporal shifting. [122, 123] researches on better traffic accident prediction to improve transportation and public safety. In these existing works, LSTM model is often used to model temporal dynamics between multiple spatial snapshots. The difference from my work introduced in this chapter is that I use LSTM to capture linear spatial sequential structure between consecutive images along a road network path. There is also other work based on spatiotemporal residual networks for traffic flow prediction [125].

3.3 Problem Description

In this section, we discuss some basic concepts and describe the problem.

Road network: A road network is a network whose nodes are road intersections, and whose edges are road segments. At the same time, a road network is also a spatial network whose nodes are spatial points and whose edges are spatial line strings. In other words, a road

network has both graph properties and geometric properties.

Streetview imagery: Streetview imagery is a sequence of geo-referenced images whose locations are embedded on road network edges (in the form of line strings). The imagery is collected through driving a vehicle equipped with GPS and camera, so that each image can be geo-referenced based on the GPS time stamp. In this chapter, we used Google Streetview API to select imagery at a regular spatial interval of 20 meters.

Road safety feature: A road safety feature is defined as the measure or infrastructure placed on a road to improve safety. We consider three most common safety features: rumble strips, concrete barrier and metal crash barrier. Figure 3.1 shows examples of the three safety features from Google Streetview imagery.

- *Rumble Strips:* Rumble strips (Figure 3.1(a)) are milled grooves or rows of raised pavement markers placed perpendicular to the direction of travel, or a continuous sinusoidal pattern milled longitudinal to the direction of travel. It creates a vibration and rumbling sound transmitted through the wheels into the vehicle interior which can alert the drivers who have drifted from their lanes.

- *Concrete barrier:* Concrete barrier (Figure 3.1(b)) is a rigid barrier. It is easy to maintain. This type of barrier is often used on roads where traffic in opposing direction is flowing in close proximity due to lack of space.

- *Metal crash barrier:* Metal crash barrier (Figure 3.1(c)), also known as guardrails, is usually made from steel beams or rails. Metal barrier helps absorb the energy from a crashed vehicle and reduce damages to vehicle and occupants. It can also act as a good visual guide during night time for the driver to maintain their lane position.

Problem Definition: Given a road network with geo-referenced streetview imagery sampled at an equal distance interval, as well as a small collection of labeled imagery sequences (each image has three binary class labels corresponding to the existence of rumble strips, concrete barrier, and metal crash barrier respectively), the road safety feature mapping

problem aims to learn a classification model that can predict the labels for all unlabeled images on the road network. Since each image may contain multiple types of road safety features at the same time, this problem is a multi-label classification problem.

3.4 Approach

In this section, I introduce a deep learning approaches to map road safety features from streetview imagery . Figure 3.2 illustrates the overall framework of the introduced model. The bottom component shows the data collection process. I sample a number of spatial points along road network edges at an equal distance interval (e.g., 20 meters), and then use Google Streetview API to download geo-referenced imagery at those point locations. I fixed the distance interval of 20 meters because the average length of some road safety features such as metal crash barrier is only a few hundred meters. If we select a higher distance interval, there may not exist enough images for short extent barrier such as metal crash barrier. Although lower distance interval can provide fine-grained dataset, it increases the number of streetview images to be downloaded which incurs extra cost. The middle component of the model is based on retrained CNN model to extract low dimensional features from individual images. The last component is LSTM layer. In contrast to existing works, the LSTM layer in my model does not capture temporal dynamics between different spatial snapshots, but represents spatial sequential pattern between consecutive imagery along road network edges.

3.4.1 Extract Image Feature with CNN

Convolutional Neural Network (CNN) was developed mainly for image classification. CNN introduces the concept of parameter sharing which allows the model to learn less number of parameters in comparison to regular neural network. Similar to regular neural networks, CNN also consists of a sequence of layers. I briefly describe each layer in CNN below. Input Layer holds the raw pixel color values (RGB) of the images. Usually, the pixel values are normalized to stabilize the learning process and dramatically reduce the number of

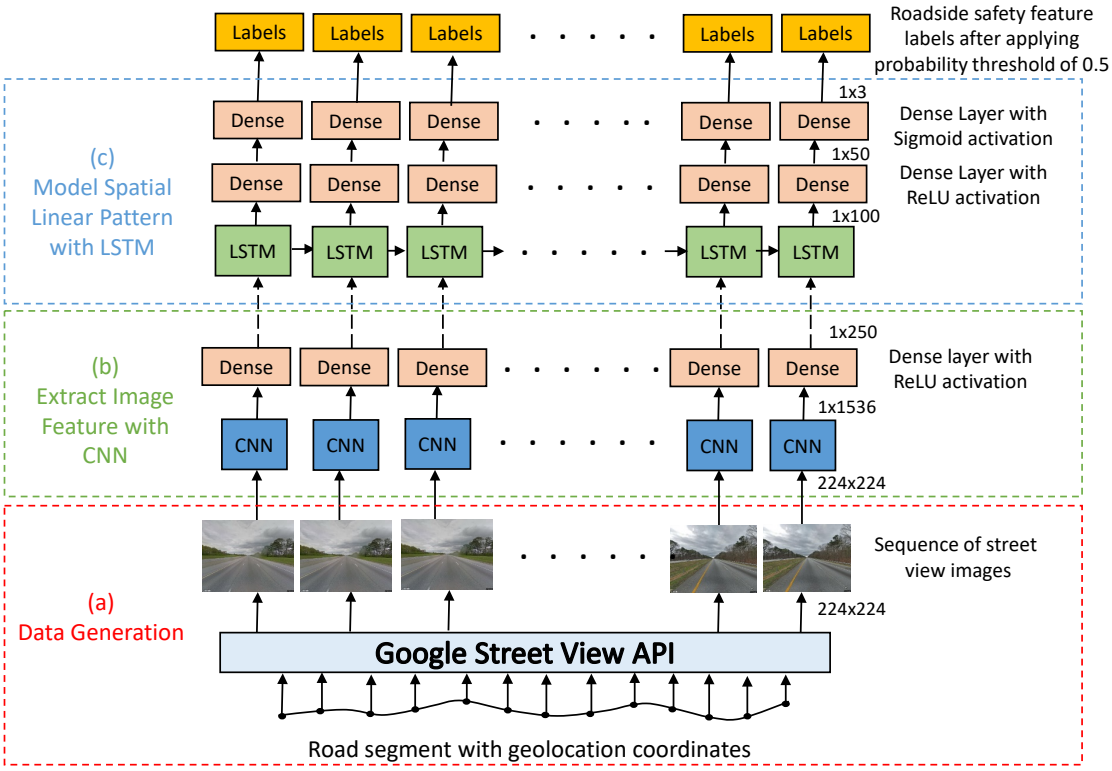


Figure 3.2: Overall framework of the deep learning approach

training epochs required to train deep learning models. Convolutional layer transforms the input using convolution operation. A convolution operation is element-wise multiplication of a pixel and its neighborhood pixels color value (RGB) by a matrix. It is also known as convolution filter. Different filters are used to convolve around all the pixels in an image. Filters like horizontal and vertical edge detecting filter can extract the linear feature from the image. Other complicated filters such as sobel filters can extract non-linear edges. In CNNs, filters are not defined, they are learned during the training process. By stacking layers of convolutions on top of each other, we can get more abstract and in-depth information from a CNN. ReLU stands for Rectified Linear Unit, which is a type of activation function commonly used in neural networks. Activation functions are applied to introduce non-linear properties to the network. The function returns 0 if it receives any negative input. However, for any positive value x, the function returns the same value back. So, it can be written as $f(x) = max(0, x)$. ReLU activation function is computationally

less expensive as there is no complicated math, which can reduce the model training time. The function of pooling layer is to reduce the spatial size of the input. It is also known as downsampling layer. Pooling layer can reduce the number of parameters and computation in the network. It applies a filter (usually of size 2x2) to the input volume. Pooling filters can be based on different operations such as max, min or average. The most common one is max filter which extracts the max value from the filter region. Fully connected (Dense) layer takes an input volume (output of activation function) and outputs a N-dimensional vector. Similar to regular neural networks, neurons in a fully connected layer have full connections to all activation in the previous layer.

I use the current state-of-the-art Inception-ResnetV2 [92] model to extract features from individual streetview images. I use the Keras implementation of Incpetion-ResnetV2 pretrained on ImageNet [24] dataset with 1000 classes. Inception-ResnetV2 combines the idea of residual connections to inception architecture. In residual connection, each layer feeds into the next layer and directly into the layers about few hops away. Residual connections are important for very deep architecture. When deeper networks starts converging, the accuracy can saturate at a point and eventually degrade. Residual connections are designed to overcome this degrading problem. As the Inception-v4 network is very deep with around 200 layers, combining Inception architecture with residual connections can be beneficial.

One issue is that the existing CNN models are not pre-trained on the road safety feature classes. To address this issue, I re-train the Inception-ResnetV2 model. Specifically, I first remove the original final dense layer with the softmax function on 1000 classes. Since we only have 3 classes, i.e., rumble strips, concrete barrier, and metal crash barrier, I replace the previous dense layer with software max function on 1000 classes with a dense layer with sigmoid function on three classes. The reason why I use sigmoid function instead of softmax function is that the class labels are non-exclusive. In other words, each image may contain multiple road safety features (e.g., both ramble strips and metal crash barriers).

I also add an additional dense (fully connected) layer with 250 nodes before the final sigmoid layer (and after the last average pooling layer with 1,536 nodes). The reason is to reduce the dimension of extracted image features (250 versus 1536). This is important because the extracted image features from CNN is fed as an input to LSTM cells and images are being classified into a smaller number of classes than the original CNN model. As shown in Figure 3.2, I retrain the CNN model using the streetview dataset. The input to the CNN model is a set of 224x224 streetview images. After retraining, I extract the output of dense layer with 250 nodes to get a feature vector of 250 dimensions for each image in the sequence. I then create a set of feature sequences to feed into the LSTM model.

3.4.2 Model Spatial Linear Pattern with LSTM

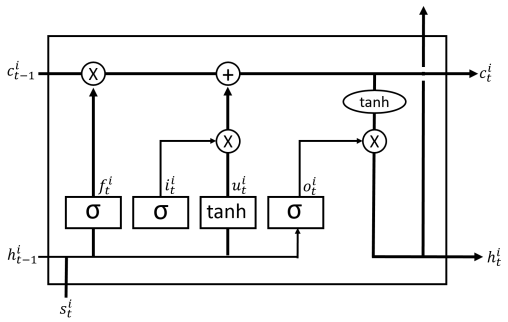


Figure 3.3: LSTM unit

In order to model spatial linear (sequential) structure along a road network path, I used the LSTM model on a sequence of image features extracted by the CNN model. LSTM is a type of recurrent neural networks that uses gating functions to avoid the exploding and vanishing gradient issues. The gate function can help a model to memorize the state of previous units in a sequence. LSTM models a sequential structure by maintaining a sequence of memory cells (c_t with t as the spatial location index). In each spatial location t, LSTM takes an input feature s^i_t, hidden state h_{t-1} and cell state c_{t-1}. Figure 3.3 shows a LSTM unit with a cell state(c_t) and three different gates: input gate, output gate and forget gate. The forget gate (f_t) decides how much information from a previous cell unit is ignored before coming to the next cell. The input gate (i_t) decides how much contribu-

tion an input feature vector makes to the current cell state. Finally, the output gate (o_t) decides what the current LSTM unit is going to output (current cell state c_t and current hidden state h_t) based on the cell state. The LSTM transaction equations are as follows,

$$\begin{aligned} f_t^i &= \sigma(W_f h_{t-1}^i + U_f s_t^i + b_f) \\ i_t^i &= \sigma(W_i h_{t-1}^i + U_i s_t^i + b_i) \\ o_t^i &= \sigma(W_o h_{t-1}^i + U_o s_t^i + b_o) \\ u_t^i &= tanh W_u h_{t-1}^i + U_u x_t^i + b_u) \\ c_t^i &= f_t * c_{t-1}^i + i_t^i * u_t \\ h_t^i &= o_t * tanh(c_t^i) \end{aligned} \tag{3.1}$$

where σ denotes the sigmoid activation function, $tanh$ is hyperbolic tangent function and $*$ denotes element-wise product. W and U denote model parameters. As Figure 3.2(c) shows, the LSTM model consists of 4 hidden layers. The first layer is an LSTM layer with an output dimension of 100 units. The second layer is a 20% dropout layer. The third layer is a dense layer with 50 nodes.

Such recurrent structure is well-suitable to capture the spatial dependency across consecutive images. According to the first law of geography: "everything is related to everything else, but near things are more related than distant things." For example, concrete barriers are often very long spanning over several miles. Metal crash barriers, in contrast, have a shorter spatial scale within a few hundred meters. LSTM has the strength of learning both short range and long range spatial dependency. The scale of spatial sequential structure can be controlled by the forget gates learned in the LSTM model. For example, for long rumble strips, the forget gates of individual cells would be largely open (receiving memory states from previous images far away due to longer spatial dependency). On the other hands, for shorter metal crash barriers that may cover only a few images, the forget gates in LSTM cells for corresponding images should be largely closed without receiving infor-

mation from images far away.

3.4.3 Multi-label Classification

This problem is a multi-label classification problem because each image may contain multiple types of road safety features at the same time. To model multi-label classification, I introduce two approaches. First approach involves training a shared CNN-LSTM model for all three class label together. Second approach involves training separate CNN-LSTM models for each class label independently.

In first approach, the last layer in the model is a sigmoid transformation layer with 3 nodes, corresponding to the three independent class labels (rumble strips, concrete barrier, and metal crash barrier). This is different from the common softmax layer whose output node values sum into one because class labels are assumed to be exclusive to each other. I use the binary cross-entropy loss. To get the final output labels for each image, I use a threshold of 0.5 on the sigmoid outputs.

In second approach, I transform the multi-label problem into multiple single-label problems. I learn three independent models corresponding to each class label. For instance, the independent model for metal crash barriers will only classify the image based on presence of metal crash barrier. Likewise, independent models for concrete barrier and rumble strips. In this design decision, the last layer is a sigmoid transformation layer with only 1 node, corresponding to either one of the three independent class labels. I use the binary cross-entropy loss. However, it would also be fine to use softmax activation function with categorical cross-entropy loss.

3.5 Experimental Evaluation

In this section, I compared my recently proposed method with baseline methods on two real world datasets in classification performance. Experiments were conducted on a Dell workstation with Intel(R) Xeon(R) CPU E5-2687w v4@3.00GHz, 64GB main memory, and a Nvidia Quadro K6000 GPU with 2880 cores and 12GB memory. I used Keras with Ten-

sorflow as backend to run the deep learning models. Candidate classification methods included:

- **CNN only**: I used Inception-ResnetV2 CNN model on streetview images with three classes: rumble strips, concrete barriers and metal crash barriers. I added one more dense layer with 250 nodes and a *ReLU* activation function before the final sigmoid layer with 3 nodes.

- **CNN-DT**: I extracted output of second last layer (with 250 nodes) from our CNN only model (Inception-ResnetV2) as feature vectors and fed it into a Decision Tree (DT) model. I used the scikit-learn package in Python.

- **CNN-RF**: I extracted output of second last layer (with 250 nodes) from our CNN only model (Inception-ResnetV2) as feature vectors and fed it into a Random Forest (RF) model. I used the scikit-learn package in Python.

- **CNN-sharedLSTM**: This is our proposed model to address the issue of multi-label classification using shared CNN-LSTM model for all three class labels together.

- **CNN-separateLSTM**: This is our proposed model to address the issue of multi-label classification using separate single-label independent CNN-LSTM models corresponding to each class label.

Unless specified otherwise, I used default parameters in open source tools in baseline methods.

Evaluation Metrics: To evaluate the candidate classification methods, I used precision, recall and F-score. I computed the precision, recall and F-score for all the class labels. Finally, I computed the weighted average F-score for all candidate classification methods. To calculate the weighted average, I used equation 3.2,

$$Avg.F = F_{RS}\frac{RS_n}{RS_n + MCB_n + CB_n} + F_{MCB}\frac{MCB_n}{RS_n + MCB_n + CB_n} + F_{CB}\frac{CB_n}{RS_n + MCB_n + CB_n} \tag{3.2}$$

where F_{RS}, F_{MCB}, and F_{CB} refers to F-score of rumble strips, metal crash barriers and concrete barriers class labels respectively. Similarly, RS_n, MCB_n, and CB_n refers to the number of image containing class labels: rumble strips, metal crash barriers and concrete barriers respectively.

3.5.1 Dataset Description

To evaluate the performance of the proposed models, first I randomly selected 3,745 labeled isolated streetview images across the state of Alabama for pre-training. I used this dataset to pre-train the CNN in all baseline and proposed methods. Next, I selected different road segments within I-20 highway in Alabama to extract spatially continuous streetview images for training, validation, and test datasets. The images in extracted datasets are different from the images in pre-training dataset. For training and validation, I selected road segment from 337'08.4"N 852'28.2"W to 335'06.5"N 863'40.6"W in I-20 East near Oxford, Alabama. I selected two test datasets. Our first test dataset was based on the road segment from 335'09.6"N 852'32.1"W to 336'56.4"N 851'25.2"W in I-20 West near Oxford, Alabama, which is closer to the training and validation datasets. Our second dataset was based on the road segment from 338'09.1"N 878'05.6"W to 331'13.2"N 879'59.3"W in I-20 West near Tuscaloosa, Alabama, which is far from the training and validation datasets. The first test dataset is refered to as Oxford dataset and second test dataset is refered to as Tuscaloosa dataset in rest of the chapter. I used same training and validation dataset for both of the test datasets.

I divided the road segments into an equally distanced set of geolocation coordinates. I set the distance interval of 20 meters. I then used Google Street View API to download the streetview images respective to each coordinate. I have three safety features classes: rumble strips (**RS**), metal crash barriers (**MCB**), and concrete barriers (**CB**). Table 3.1

shows the number of images and class distribution for training, validation and test datasets.

Table 3.1: Class distribution

	Number of Images	Rumble Strips	Metal Crash Barriers	Concrete Barriers
Pre-Training Set	3745	1882	1149	1632
Training Set	983	868	324	352
Validation Set	594	493	96	224
Oxford dataset	950	857	279	354
Tuscaloosa dataset	1350	879	403	784

3.5.2 Hyperparameter Settings

For the proposed models, there are several design decision to be made including input vector length in LSTM, dimension of the hidden state of LSTM, learning rate, dropout, optimization function and training batch size. To train the proposed CNN-LSTM models, I used the input vector length of 50 spatially continuous images on each sequence. I used a sliding window of 1 on training and validation datasets to create training and validation sequences for our proposed CNN-LSTM models. I generated 883 training and 544 validation sequences. I set the input vector length in LSTM as 50, and the dimension of the hidden state of LSTM as 100. The input vector length as 50 refers to streetview images covering 1000 meters (20 meters separation between images). For learning rate, I first started with high learning rate of 10^{-3} and observed oscillating training and validation loss curves. I then gradually decreased the learning rate and obtained more stable curves with the learning rate of 10^{-6} for CNN and CNN-LSTMs models. Next, I varied the dropout value from 0.2 to 0.6. I observed that with the higher value of dropout the convergence of training and validation loss was slower and required higher number of epochs. So, I tuned the dropout to the optimal value of 0.2. I trained CNN and CNN-LSTMs models using Adam optimizer. The training batch size for CNN model was 32 and that for CNN-LSTMs were 1.

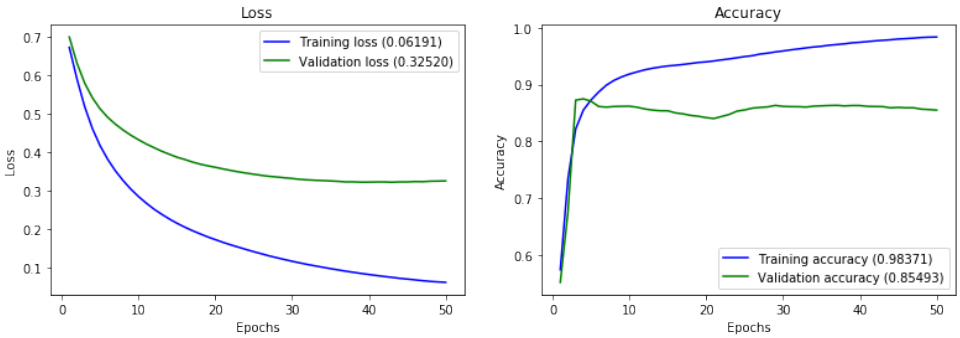


Figure 3.4: Training and validation performance of CNN-sharedLSTM over 50 epochs

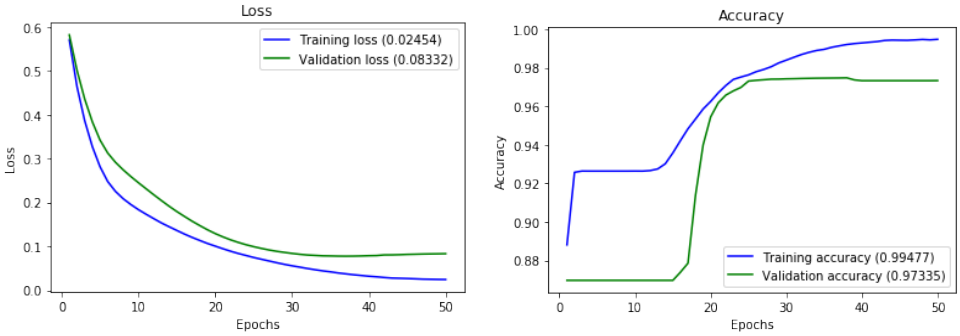


Figure 3.5: Training and validation performance of CNN-separateLSTM for rumble strips over 50 epochs

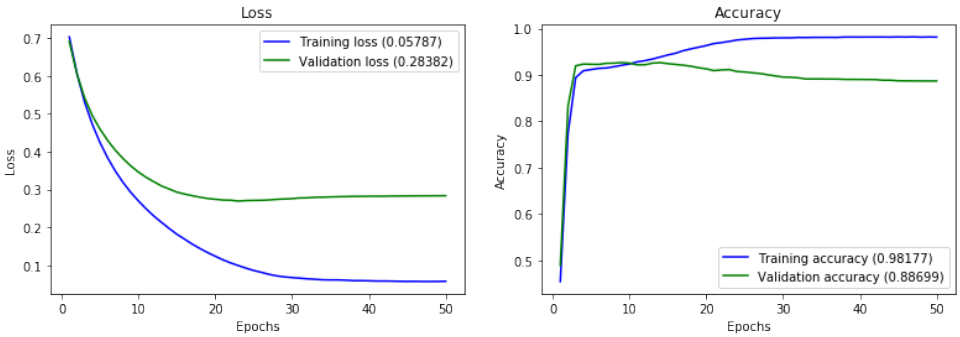


Figure 3.6: Training and validation performance of CNN-separateLSTM for metal crash barriers over 50 epochs

3.5.3 Classification Performance

Figure 3.4 shows the training performance of CNN-sharedLSTM model. Similarly, Figure 3.5, 3.6, and 3.7 shows the training performance of CNN-separateLSTM model for rumble strips, metal crash barriers, and concrete barriers class labels respectively. The

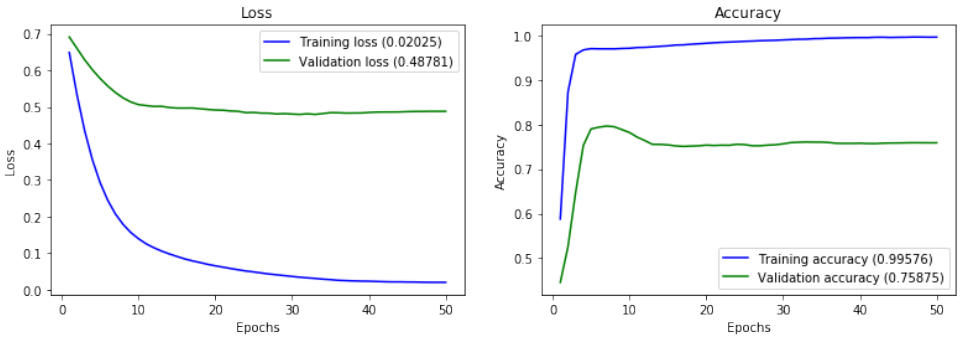


Figure 3.7: Training and validation performance of CNN-separateLSTM for concrete barriers over 50 epochs

training and validation loss for CNN-sharedLSTM are 0.06 and 0.32 respectively. In case of CNN-separateLSTM, the training and validation loss for rumble strips class label are 0.02 and 0.08 respectively. We can observe that the gap between training and validation loss for rumble strips in CNN-separateLSTM model is lower than that of CNN-sharedLSTM model. The training and validation loss in CNN-separateLSTM for metal crash barriers class label are 0.05 and 0.28 respectively, which also shows similar trend. It is because the CNN-separateLSTM model can better learn the spatial scale of rumble strips and metal crash barrier separately. Rumble stirps have very high spatial scale in comparison to the metal crash barriers. Furthermore, rumble strips and metal crash barriers are mostly easy to identify in the streetview images. Finally, the training and validation loss for concrete barrier class label are 0.02 and 0.48 respectively. As shown in in Figure 3.7, the gap between training and validation curve is high and indicates over-fitting. It is likely because the concrete barriers are very diverse and hard to learn. I observe the texture of concrete barriers in some regions are very similar to the texture of the roads.

I compare different candidate methods on precision, recall, and F-score. To obtain the predicted class label, I set the probability threshold of 0.5. A road safety feature probability value above the threshold indicates presence of the road safety feature in the image. Results are summarized in Table 3.2 and 3.3 for Oxford and Tuscaloosa datasets respectively.

Table 3.2: Classification on Oxford dataset

Classifiers	Class	Precision	Recall	F	Avg. F
CNN-DT	RS	0.95	0.83	0.89	0.85
	MCB	0.79	0.82	0.80	
	CB	0.81	0.76	0.78	
CNN-RF	RS	0.94	0.87	0.90	0.87
	MCB	0.89	0.86	0.87	
	CB	0.78	0.82	0.80	
CNN only	RS	0.94	0.95	0.95	0.89
	MCB	0.91	0.78	0.84	
	CB	0.74	0.85	0.79	
CNN-sharedLSTM	RS	0.94	0.98	0.96	0.91
	MCB	0.90	0.83	0.86	
	CB	0.74	0.91	0.82	
CNN-separateLSTM	RS	0.95	0.96	0.96	0.92
	MCB	0.93	0.83	0.88	
	CB	0.77	0.93	0.84	

Table 3.3: Classification on Tuscaloosa dataset

Classifiers	Class	Precision	Recall	F	Avg. F
CNN-DT	RS	0.85	0.84	0.85	0.75
	MCB	0.50	0.86	0.63	
	CB	0.94	0.54	0.69	
CNN-RF	RS	0.88	0.87	0.88	0.77
	MCB	0.58	0.77	0.66	
	CB	0.93	0.58	0.71	
CNN only	RS	0.87	0.97	0.92	0.78
	MCB	0.78	0.73	0.75	
	CB	0.89	0.49	0.63	
CNN-sharedLSTM	RS	0.85	0.98	0.91	0.79
	MCB	0.68	0.84	0.75	
	CB	0.93	0.53	0.68	
CNN-separateLSTM	RS	0.88	0.96	0.92	0.83
	MCB	0.74	0.80	0.77	
	CB	0.96	0.61	0.76	

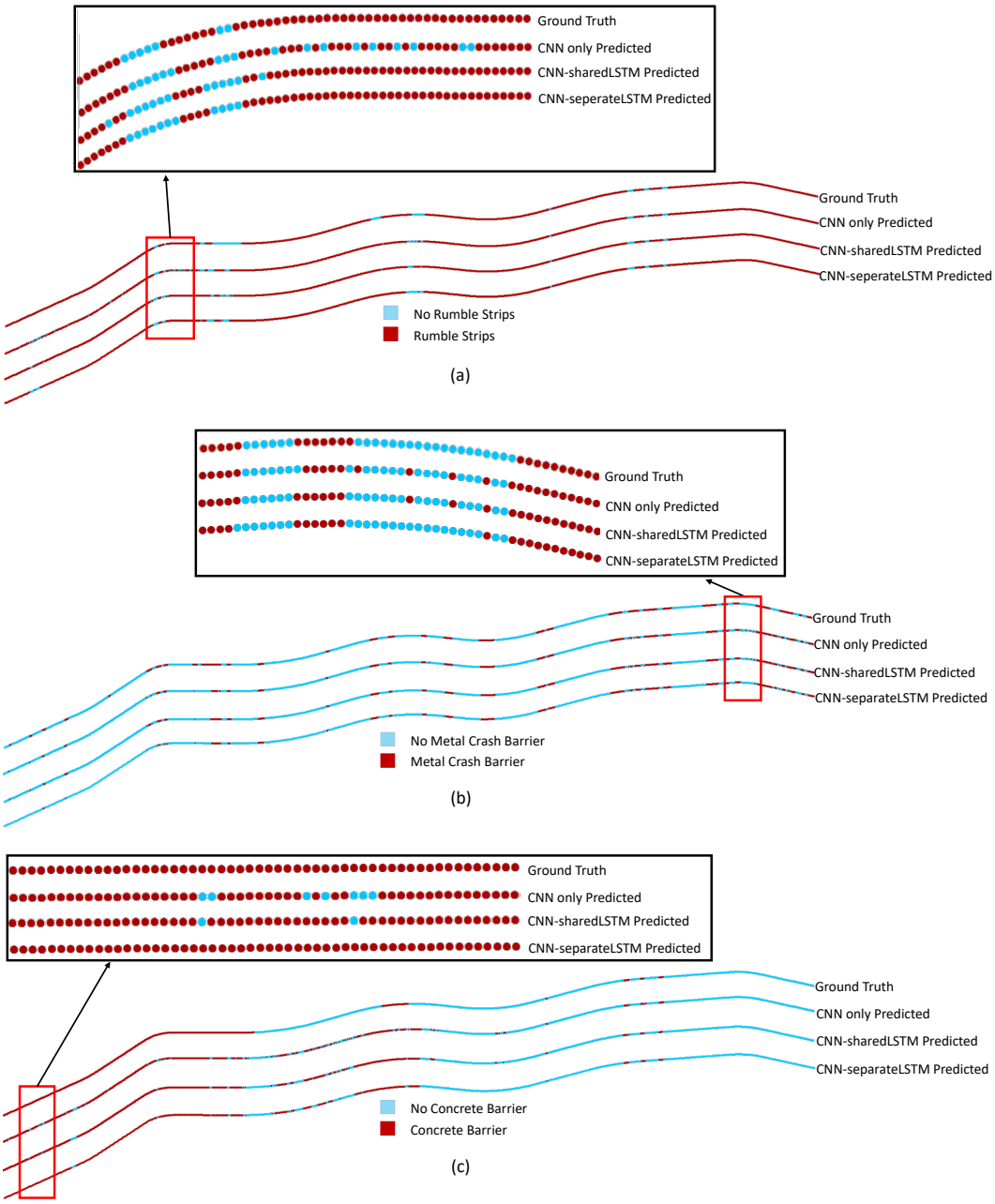


Figure 3.8: Prediction maps on Oxford dataset for (a) Rumble strips, (b) Metal crash barriers and (c) Concrete barriers respectively

For Oxford dataset in Table 3.2, the average F-score of CNN-DT, CNN-RF and CNN only models are 0.85, 0.87 and 0.89 respectively. we can observe the average F-score for CNN with DT and RF is lower than CNN only. It is because CNN-DT and CNN-RF takes the output of 2nd last layer (with 250 output dimension) of CNN only model as the features

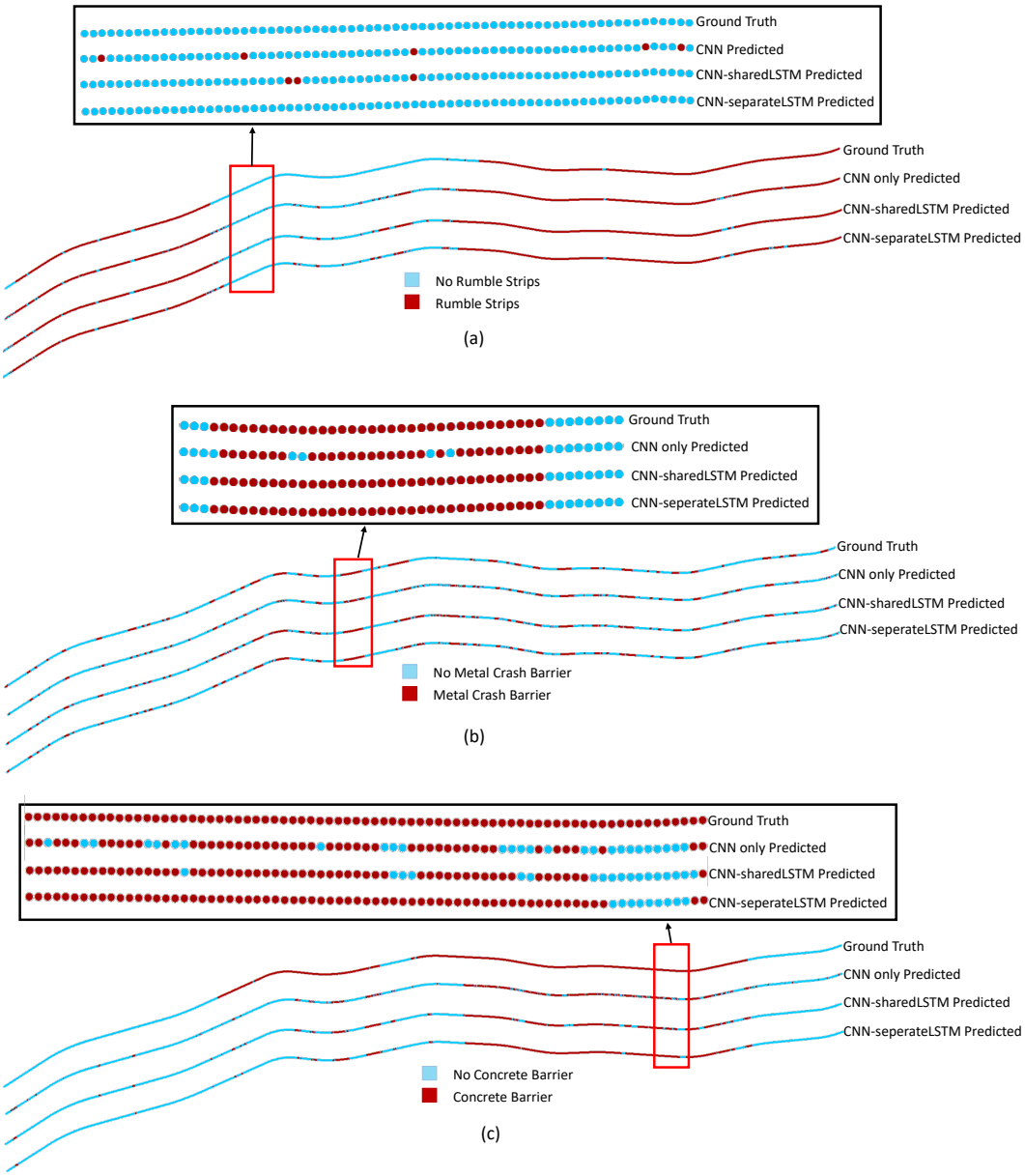


Figure 3.9: Prediction maps on Tuscaloosa dataset for (a) Rumble strips, (b) Metal crash barriers and (c) Concrete barriers respectively

and fits the models. But the last layer in CNN only model, a dense layer with 3 nodes, have extra learnable parameters which can help CNN only model train better. Next, we can also observe that both of the proposed models: CNN-sharedLSTM and CNN-separateLSTM perform better than CNN only model with average F-score of 0.91 and 0.92 respectively. CNN only model may fail to correctly identify some inbetween images in the test image

sequence. Our proposed models can correct these errors by incorporating spatial dependency in the learning process using LSTM network. Also, we can observe that our CNN-separateLSTM performs better than CNN-sharedLSTM. It is because in case of CNN-separateLSTM, the independent models for each class label can better learn the spatial scale of different class labels separately. For example, metal crash barrier have an average length of few hundred meters wheres other road safety features such as rumble strips may have average length of few kilometers. Furthermore, for CNN-separateLSTM, we can observe very high F-score of 0.96 on rumble strips class which is consistent with the learning curve in Figure 3.5 with small gap between the training and validation loss. In case of metal crash barriers, I observe F-score of 0.88 which is also consistent with the learning curve shown in Figure 3.6 with medium gap between training and validation loss. Finally, for concrete barrier, I observe lowest F-score of 0.84, also consistent with the learning curve shown in Figure 3.7 with high gap between training and validation loss.

For Tuscaloosa dataset in Table 3.3, we can also observe similar trends as discussed above. But, the overall performance of all the candidate methods is lower. The average F-score of CNN-DT, CNN-RF, CNN only, CNN-sharedLSTM, and CNN-separateLSTM are 0.75, 0.77, 0.78, 0.79, and 0.83 respectively. It is likely because the location of images in Tuscaloosa dataset lies far from the training and validation region. So, the training and validation images might not be very similar to images in Tuscaloosa dataset. It can be easily improve by introducing some representative images closer to Tuscaloosa dataset in training and validation data.

I also visualize the predicted class label on map based on all the predicted results summarized in Table 3.2 and Table 3.3. Figure 3.8 shows the ground truth and prediction maps for three road safety feature classes based on CNN only, CNN-sharedLSTM and CNN-separateLSTM models on Oxford dataset. Similarly, Figure 3.9 shows ground truth and prediction maps on Tuscaloosa dataset. We can observe that different road safety features have different spatial scale. From Figures 3.8.(a) and 3.9.(a), we can observe that

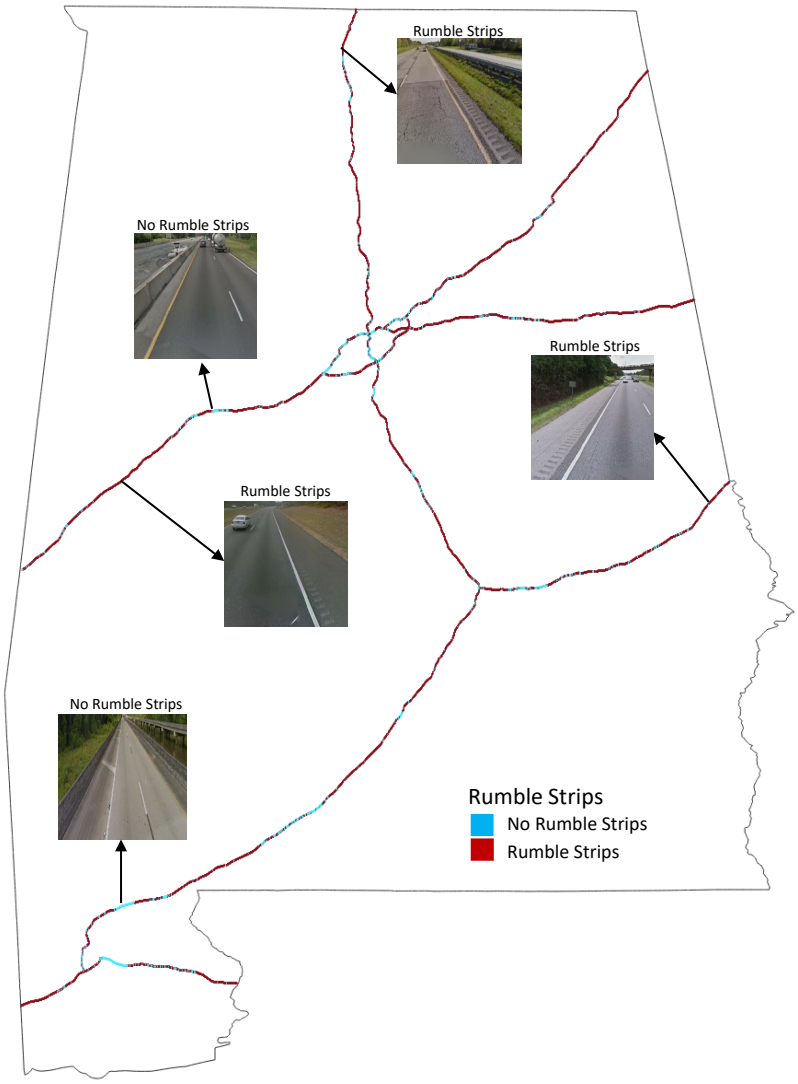


Figure 3.10: Classification map for rumble strips using CNN-separateLSTM model

ground truth spatial scale of rumble strips are usually very long. From Figures 3.8.(b) and 3.9.(b), we can observe that ground truth spatial scale of metal crash barriers are usually very short. And finally, from Figures 3.8.(c) and 3.9.(c), we can observe that ground truth spatial scale of concrete barriers can vary from short to long. Next, in all the prediction maps, we can observe CNN only based predictions contain some isolated errors. But, CNN-sharedLSTM model was able to correct some of those isolated errors as highlighted in zoom-in sub-visualizations. However, we can see that CNN-separateLSTM model to be more accurate, which is consistent with the summarized result in Table 3.2 and 3.3. The

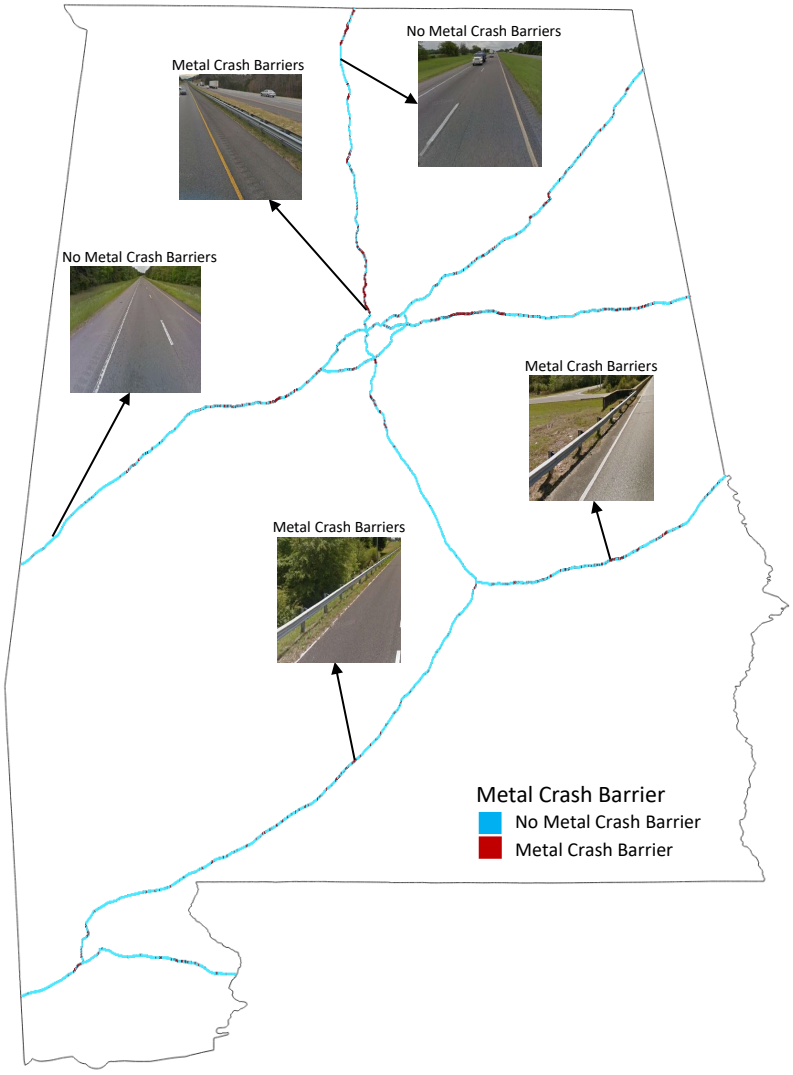


Figure 3.11: Classification map for metal crash barriers using CNN-separateLSTM model

CNN-sharedLSTM and CNN-separateLSTM models are able to generate more accurate map than CNN only model due to the incorporation of spatial sequential structure in the learning process.

3.5.4 Case Study

I downloaded 69,500 streetview images of all the major highways in the entire state of Alabama. The major highways include I-10, I-20, I-59, I-65, I-85 and I-459 within Alabama. I then classified the road safety features in those images using our CNN-separateLSTM model. The map for each road safety features based on the classification results is shown

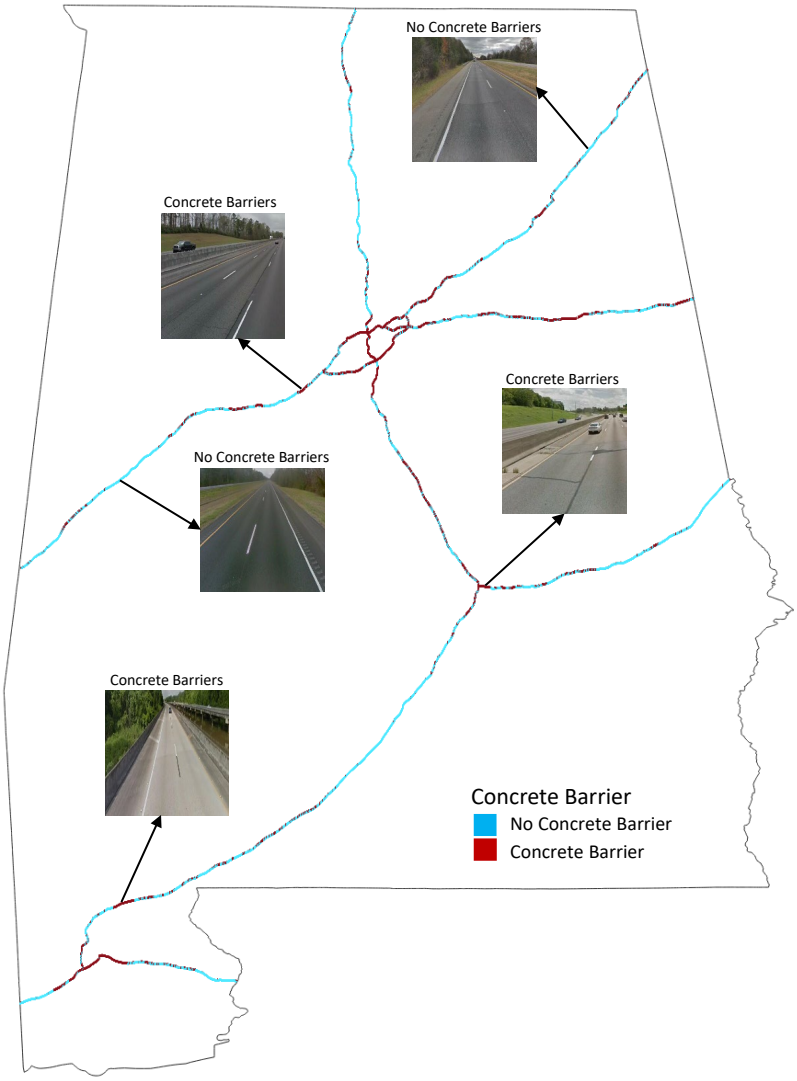


Figure 3.12: Classification map for concrete barriers using CNN-separateLSTM model

in figures 3.10, 3.11 and 3.12. I randomly checked few data points to verify the classification result and found that most of the images were accurately classified.

According to the predicted maps, 54,890 streetview images were classified to have rumble strips which is around 1200 km out of 1390 km in major highways in Alabama. 12,841 streetview images were classified to have metal crash barriers which implies around 257 km out of 1390 km. Finally, 26,997 streetview images were classified to have concrete barriers which implies around 540 km out of 1390 km. I observe that the spatial scale of the predicted map for rumble strips are very long which is consistent with the spatial scale

observed in the ground truth for test datasets. Similarly, from the predicted map in Figure 3.11, I observe that the metal crash barriers are evenly distributed throughout the major highways in Alabama and have short spatial scale. It is also consistent with the test datasets. Finally, from the predicted map in Figure 3.12, I observe that the spatial scale of concrete barriers can vary. I further observe that the long chain concrete barriers are usually located near city areas. Short concrete barriers are usually placed in the bridges. Also, I notice that the bridges with the concrete barriers usually do not contain rumble strips.

3.6 Summary

This chapter introduces two different CNN-LSTM based spatial classification models for mapping safety features along road networks. My CNN-LSTM models can capture spatial linear structure between consecutive images along a road network path. Results on real world Google Streetview images collected in Alabama show that my models outperform several baseline methods. Furthermore, through experimental evaluation, I found out that the separate CNN-LSTM models for independent class labels performed better than shared CNN-LSTM model.

CHAPTER 4

ADDRESSING LIMITED OBSERVATION CHALLENGE: SPATIAL CLASSIFICATION WITH LIMITED OBSERVATIONS BASED ON PHYSICS-AWARE STRUCTURAL CONSTRAINT

4.1 Introduction

Given a spatial raster framework with explanatory feature layers, a spatial contextual layer (e.g., a potential field), as well as a set of training samples with class labels outside the framework, the spatial classification problem [39] aims to learn a model that can predict a class layer. I particularly focus on spatial classification with limited feature observations, i.e., only limited pixel locations in the raster framework have explanatory feature data available. For example, in earth imagery classification, the explanatory feature layers are spectral bands of earth imagery pixels; the spatial contextual layer can be elevation, and the target class layer consists of pixel classes (e.g., flood or dry). In the example, it often happens that the elevation values are available for all pixels in the framework, but only limited pixel locations have spectral data (e.g., a drone or aerial plane could not cover the entire region due to limited time during a flood disaster).

The problem is important in many societal applications such as flood extent mapping for disaster response and national water forecasting. Flood extent mapping plays a crucial role in addressing grand societal challenges such as disaster management, national water forecasting, as well as energy and food security [44, 111]. For example, during Hurricane Harvey floods in 2017, first responders needed to know where floodwater was in order to plan rescue efforts. In national water forecasting, detailed flood extent maps can be used to calibrate and validate the NOAA National Water Model [66]. In current practice, flood

extent maps are mostly generated by flood forecasting models, whose accuracy is often unsatisfactory in a high spatial resolution [22, 60]. Other ways to generate flood maps involve sending a field crew on the ground to mark down the floodwater extent on a map, but the process is both expensive and time-consuming. A promising alternative is to utilize observation data from groundwater sensors and remote sensors on aerial planes or drones. However, sensor observations may have limited spatial coverage due to only a subset of sensors being deployed at certain regions, making the problem in spatial classification with limited features. For example, a drone during a flood disaster can only collect spectral images in limited areas. Note that though I use flood mapping application as an example, the problem can potentially be applied to other broad applications such as water quality monitoring [112] along river networks.

The problem poses several unique challenges that are not well addressed by traditional classification techniques. First, there are limited feature observations on samples in the raster framework due to only a subset of sensors being deployed in certain regions. In other words, only a subset of samples has complete explanatory feature values, making it hard to predict classes for all samples. Second, among the sample pixels with complete explanatory feature values, their feature values may contain rich noise and obstacles. For example, high-resolution earth imagery often has noise: clouds and shadows. Third, the explanatory features of image pixels can be insufficient to distinguish classes (also called class confusion) due to heterogeneity. For instance, pixels of tree canopies overlaying flood water have the same spectral features with trees in dry areas, yet their classes are different. Finally, the number of pixel locations can be very large for high-resolution data (e.g., hundreds of millions of locations in one city), requiring scalable algorithms.

Over the years, various techniques have been developed to address missing feature observations (or incomplete data) in classification [26]. Existing methods can be categorized into data cleaning or imputation, utilizing classification models that allow for missing feature values, and modeling missing features as hidden variables with the EM (Expectation-

Maximization) algorithm. Data cleaning will remove samples that miss significant feature values. Data imputation focuses on filling in missing values either by statistical methods [57] (e.g., mean feature values from observed samples) or by predicted feature values using a regression model based on observed samples [85, 13, 121, 14, 79]. A different approach focuses on classification models and algorithms that allow for missing feature values in learning and prediction without data imputation. For example, a decision tree model allows for samples with missing features in learning and classification [73, 72, 107]. During training, the weight is one for an observed feature value but is lower than one if the feature value is missing (the weight for different possible values are based on their frequencies in completely observed samples). During classification, a decision tree can explore all possible tree traversal paths for samples with missing features and select the final class prediction with the highest probability. Similarly, there are some other models or algorithms that can be extended to allow for missing feature values, such as neural network ensembles [38], support vector machine [20, 89, 71], etc. The last category is to model missing feature values as hidden variables and use the EM (Expectation-Maximization) algorithm for effective learning and inference [59, 29, 30, 108]. Specifically, the joint distribution of all samples' features (both observed and missing features) can be represented by a mixture model with fixed by yet unknown parameters. In the EM algorithm, I can use initialized parameters and observed features to estimate the posterior distribution of hidden variables (missing features), and then further update the parameters for the next iteration. However, all these existing methods above assume that incomplete feature observations only happen on a small subset of samples, and thus cannot adequately be applied to my problem where the vast majority of samples have missing features (limited feature observations).

To fill the gap, I introduce my recently proposed approach that incorporates physics-aware structural constraints into model representation [82]. My approach assumes that a spatial contextual feature is observed for all sample locations, and establishes spatial structural

constraints from the spatial contextual feature map. I design efficient algorithms for model parameter learning and class inference and conduct experimental evaluations to validate the effectiveness and efficiency of the proposed approach against existing works. More specifically, I make the following contributions:

- I introduce a new approach that utilizes physics-aware spatial structural constraints to handle limited feature observations in spatial classification.
- I design efficient algorithms for model parameter learning and class inference.
- I evaluate the proposed model on two real-world hydrological datasets. Results show that the proposed approach significantly outperforms several baseline methods in classification accuracy, and the proposed solution is computationally efficient on a large data volume.

4.2 Problem Statement

4.2.1 Preliminaries

Here I define several basic concepts to define the problem formally.

A *spatial raster framework* is a tessellation of a two dimensional plane into a regular grid of N cells. Spatial neighborhood relationship exists between cells based on cell adjacency. The framework can contain m non-spatial explanatory feature layers (e.g., spectral bands in earth imagery), one potential field layer (e.g., elevation), and one class layer (e.g., *flood*, *dry*).

Each cell in a raster framework is a *spatial data sample*, noted as $\mathbf{s_n} = (\mathbf{x}_n, \phi_n, y_n)$, where $n \in \mathbb{N}, 1 \leq n \leq N$, $\mathbf{x}_n \in \mathbb{R}^{m \times 1}$ is a vector of m non-spatial explanatory feature values with each element corresponding to one feature layer, $\phi_n \in \mathbb{R}$ is a cell's potential field value, and $y_n \in \{0, 1\}$ is a binary class label.

A raster framework with all samples is noted as $\mathcal{F} = \{\mathbf{s_n} | n \in \mathbb{N}, 1 \leq n \leq N\}$, non-spatial explanatory feature matrix of all samples are noted as $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$, the potential field

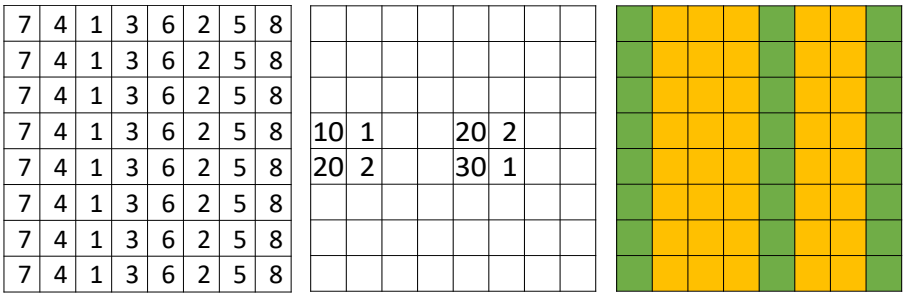
vector is noted as $\mathbf{\Phi} = [\phi_1, ..., \phi_N]^T$, and the class vector is noted as $\mathbf{Y} = [y_1, ..., y_N]^T$.
In a raster framework, it may happen that only a limited number of samples have non-spatial explanatory features being observed. I define $\mathcal{O}$ as the corresponding set of these sample indices. Samples with complete explanatory features are noted as $\{\mathbf{x}_n | n \in \mathcal{O}\}$. The corresponding sample feature matrix is noted as $\mathbf{X_o}$.

4.2.2 Problem Definition

Given a raster framework with the explanatory features of a limited number samples in the framework $\mathbf{X}_o$, the potential field layer of all samples in the framework $\mathbf{\Phi} = [\phi_1, ..., \phi_N]^T$, and a set of training samples with class labels outside the framework, the spatial classification problem aims to learn a classifier f to predict the class layer $\mathbf{Y} = f(\mathbf{X}_o, \mathbf{\Phi})$. For example, in earth imagery classification, the explanatory feature layers are spectral bands of earth imagery pixels; the spatial contextual layer can be elevation, and the target class layer consists of pixel classes (e.g., flood or dry). In the example, it often happens that the elevation values are available for all pixels in the framework (elevation values do not change over time and thus can be collected at once), but only limited pixel locations have spectral data (e.g., a drone or aerial plane could not cover the entire region due to limited time during a flood disaster). Figure 4.1 shows a toy example of a raster framework that consists of sixty-four samples with a one-dimensional explanatory feature and a full potential field layer. There are only eight samples with observed explanatory features (four non-empty cells in Figure 4.1(b)). The goal is to learn a model that can predict the class layer in Figure 4.1(c).

4.3 Approach

In this section, I introduce my recently proposed approach. I start with physics-aware structural constraints and then introduce my probabilistic model and its learning and inference. I will introduce my approach in the context of flood mapping application example, but the proposed method can be potentially generalized to other applications such as ma-



(a) Spatial potential field (elevation, $\mathbf{\Phi}$)

(b) Non-spatial feature ($\mathbf{X}_o$)

(c) True classes of all samples ($\mathbf{Y}$)

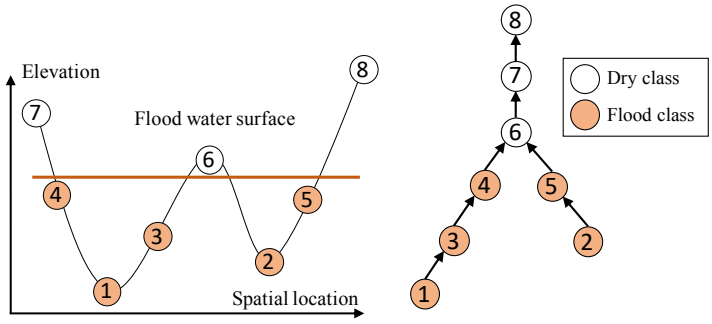
Figure 4.1: An illustration problem example (dry in green, flood in orange)

terial science and biochemistry.

4.3.1 Physics-Aware Structural Constraint

The main idea of my proposed approach is to establish a spatial dependency structure of sample class labels based on the physical constraint from the spatial potential field layers (e.g., water flow directions based on elevation). Figure 4.2 is an illustrative example. Figure 4.2(a) is the elevation values of eight pixels in one dimensional space (e.g., pixels on a row in Figure 4.1). Due to gravity, water flows from high locations to nearby lower locations. If location 4 is flooded, locations 1 and 3 must also be flooded. Such a dependency structure can be established based on the topology of the potential field surface (e.g., elevation). Figure 4.2(b) shows a directed tree structure that captures the flow dependency structure. If any node is *flood*, then all sub-tree nodes must be *flood* due to gravity. The structure is also called *split tree* in topology [18, 25], where a node represents a vertex on a mesh surface (spatial potential field), and the edges show the topological relationships between vertices. We can efficiently construct the tree structure from a potential field map following the topological order of pixels based on the union-find operator (the time complexity is $O(N \log N)$) [18]. The main idea is to sort all the pixels with an increasing order of elevation (this is also the topological order of nodes in the tree). Then, starting from the pixel with the lowest elevation, nodes were added to the rears of tree branches through a visited spatial neighbor. For example, nodes 1 and 2 were first added to the tree as leaf

nodes (none of their neighbors was in the tree yet), and then node 3 was added to the rear of the branch corresponding to its visited neighbor (node 1). Similarly, nodes 4 and 5 were added to two branches respectively. Node 6 were added to merge two branches since it is the neighbor of both node 3 and node 2. In order to efficiently maintain the set of nodes in a current branch and to quickly reach the rear end, I use efficient union-find operation on disjoint sets based on path compression. The details can be found in the split tree construction in [18]. It is worth to note that though the illustrative example in Figure 4.2 is in one-dimensional space for simplicity, the structure can be applicable to two-dimensional space in general cases. For example, I can create a single tree structure for the entire elevation map in Figure 4.1(a).



(a) Eight consecutive sample locations in one dimensional space

(b) Partial order constraint in a reverse tree

Figure 4.2: Illustration of partial order class dependency

4.3.2 Model Probabilistic Formulation

Now, I introduce my approach that integrates physics-aware structural constraint into the probabilistic model formulation to handle limited samples with observed features. Figure 4.3 illustrates the overall model structure. It consists of two layers: a hidden class layer with unknown sample classes (y_n) and an observation layer with limited sample feature vectors ($\mathbf{x}_n$). Each node corresponds to a spatial data sample (raster cell). Edge directions show a probabilistic conditional dependence structure based on physical constraint.

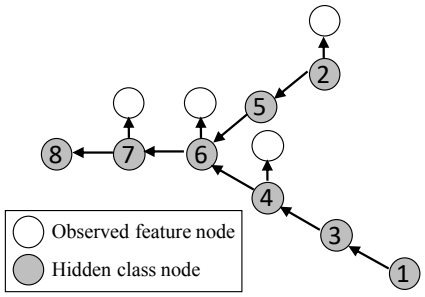


Figure 4.3: Illustration of model structure

The joint distribution of all observed samples' features and classes can be expressed as Equation 5.1, where $\mathcal{P}_n$ is the set of parent samples of the nth sample in the dependency tree, and $y_{k\in\mathcal{P}_n} \equiv \{y_k | k \in \mathcal{P}_n\}$ is the set of class nodes corresponding to parents of the nth sample. For a leaf node n, $\mathcal{P}_n = \emptyset$, and $P(y_n|y_{k\in\mathcal{P}_n}) = P(y_n)$.

$$P(X_o, \mathbf{Y}) = P(X_o|\mathbf{Y})P(\mathbf{Y}) = \prod_{n\in\mathcal{O}} P(\mathbf{x}_n|y_n) \prod_{n=1}^{N} P(y_n|y_{k\in\mathcal{P}_n}) \tag{4.1}$$

The conditional probability of sample feature vector given its class can be assumed i.i.d. Gaussian for simplicity, as shown in Equation 5.2, where $\boldsymbol{\mu}_{y_n}$ and $\boldsymbol{\Sigma}_{y_n}$ are the mean and covariance matrix of feature vector $\mathbf{x}_n$ for class y_n ($y_n = 0, 1$). It is worth noting that $P(\mathbf{x}_n|y_n)$ could be more general than i.i.d. Gaussian.

$$P(\mathbf{x}_n|y_n) \sim \mathcal{N}(\boldsymbol{\mu}_{y_n}, \boldsymbol{\Sigma}_{y_n}) \tag{4.2}$$

Class transitional probability follows the partial order constraint. For example, due to gravity, if any parent's class is *dry*, the child's class must be *dry*; if all parents' classes are *flood*, then the child has a high probability of being *flood*. Consider *flood* as the positive class (class value 1) and *dry* as the negative class (class value 0), the transitional probability is actually conditioned on the product of parent classes $y_{\mathcal{P}_n} \equiv \prod_{k\in\mathcal{P}_n} y_k$. Table 4.1 shows two parameters for class transitional probability (ρ) and class prior probability (π).

Table 4.1: Class transition probability and prior probability

$P(y_n \mid y_{\mathcal{P}_n})$	$y_{\mathcal{P}_n}=0$	$y_{\mathcal{P}_n}=1$
$y_n=0$	1	$1-\rho$
$y_n=1$	0	ρ

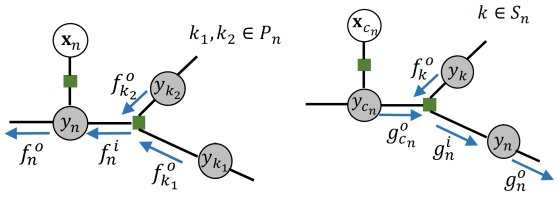
	$P(y_n)$
$y_n=0$	$1-\pi$
$y_n=1$	π

4.3.3 Model Parameter Learning and Class Inference

My model parameters include the mean and covariance matrix of sample features in each class, the prior probability of leaf node classes, and class transition probability for non-leaf nodes. I denote the entire set of parameters as $\mathbf{\Theta} = \{\rho, \pi, \boldsymbol{\mu}_c, \mathbf{\Sigma}_c | c = 0, 1\}$. Learning the set of parameters poses two major challenges: first, Equation 5.1 both unknown parameters and hidden class variables $\mathbf{Y} = [y_1, ..., y_N]^T$ that are non-i.i.d.; second, the number of samples (N) can be huge (e.g., millions of pixels).

To address these challenges, I propose to use the expectation-maximization (EM) algorithm together with message (belief) propagation. The main idea of the EM algorithm is to first initialize a parameter setting, and compute the posterior expectation of log-likelihood (Equation 5.1) on hidden class variables (E-step). The posterior expectation is a function of unknown parameters, and thus can be maximized by updating the parameter values (M-step). The two steps can be done iteratively until the parameter values converge. One remaining issue is the calculation of posterior expectation of log-likelihood on hidden class variables. The requires to compute the marginal posterior distribution of $P(y_n, y_{k \in \mathcal{P}_n} | \mathbf{O}, \mathbf{\Theta_0})$ and $P(y_n)$ for each node n. This is very challenging due to its high dimensionality of $\mathbf{Y}$. To address this challenge, I use message propagation. Message propagation is based on the sum and product algorithm [53, 76]. Propagation of message along nodes in a graph (or tree) is equivalent to marginalizing out node variables in the overall joint distribution in Equation 5.1.

The message passing process is illustrated in Figure 4.4. Specifically, forward message propagation from leaves to root is based on Equation 4.3 and Equation 4.4, where $f_n^i(y_n)$ and $f_n^o(y_n)$ are the incoming message into and outgoing message from a hidden class node



(a) From leaves to root (b) From root to leaves

Figure 4.4: Illustration of message propagation in split tree

y_n respectively. Backward message propagation from root to leaves also follows a recursive process, as shown in Equation 4.5 and Equation 4.6, where $g^i_n(y_n)$ and $g^o_n(y_n)$ are the incoming and outgoing messages for class node y_n respectively. For those samples without feature vector x_n, the outgoing forward and backward messages are the same with incoming forward and backward messages respectively, because we do not consider the feature probability for those samples.

$$f^i_n(y_n) = \begin{cases} P(y_n) & \text{if } y_n \text{ is leaf} \\ \sum_{y_{k\in\mathcal{P}_n}} P(y_n|y_{k\in\mathcal{P}_n}) \prod_{k\in\mathcal{P}_n} f^o_k(y_k) & \text{otherwise} \end{cases} \tag{4.3}$$

$$f^o_n(y_n) = \begin{cases} f^i_n(y_n)P(\mathbf{x}_n|y_n) & \text{if } n \in \mathcal{O} \\ f^i_n(y_n) & \text{otherwise} \end{cases} \tag{4.4}$$

$$g^i_n(y_n) = \begin{cases} 1 & \text{if } y_n \text{ is root} \\ \sum_{y_{c_n}, y_{k\in\mathcal{S}_n}} g^o_{c_n} P(y_{c_n}|y_n) \prod_{k\in\mathcal{S}_n} f^o_k(y_k) & \text{otherwise} \end{cases} \tag{4.5}$$

$$g^o_n(y_n) = \begin{cases} g^i_n(y_n)P(\mathbf{x}_n|y_n) & \text{if } n \in \mathcal{O} \\ g^i_n(y_n) & \text{otherwise} \end{cases} \tag{4.6}$$

After both forward and backward message propagation, we can compute the marginal posterior distribution of hidden class variables based on the equations below, where P' is unnormalized marginal distribution.

$$P'(y_n|\mathbf{X}_o, \mathbf{\Theta_0}) = \begin{cases} f^i_n(y_n)g^i_n(y_n)P(\mathbf{x}_n|y_n) & \text{if } n \in \mathcal{O} \\ f^i_n(y_n)g^i_n(y_n) & \text{otherwise} \end{cases} \tag{4.7}$$

$$P'(y_n, y_{k\in\mathcal{P}_n}|\mathbf{X}_o, \mathbf{\Theta_0}) = \prod_{k\in\mathcal{P}_n} f_k^o(y_k) g_n^o(y_n) \tag{4.8}$$

$$P(y_n|\mathbf{X}_o, \mathbf{\Theta_0}) \leftarrow \frac{P'(y_n|\mathbf{X}_o, \mathbf{\Theta_0})}{\sum\limits_{y_n} P'(y_n|\mathbf{X}_o, \mathbf{\Theta_0})} \tag{4.9}$$

$$P(y_n, y_{k\in\mathcal{P}_n}|\mathbf{X}_o, \mathbf{\Theta_0}) \leftarrow \frac{P'(y_n, y_{k\in\mathcal{P}_n}|\mathbf{X}_o, \mathbf{\Theta_0})}{\sum\limits_{y_n, y_{k\in\mathcal{P}_n}} P'(y_n, y_{k\in\mathcal{P}_n}|\mathbf{X}_o, \mathbf{\Theta_0})} \tag{4.10}$$

After the computation of marginal posterior distribution, we can update model parameters by maximizing the posterior expectation of log-likelihood (the maximization or M step in EM), as shown by equations below.

$$\rho = \frac{\sum\limits_{n|\mathcal{P}_n\neq\emptyset} \sum\limits_{y_n} \sum\limits_{y_{\mathcal{P}_n}} y_{\mathcal{P}_n}(1-y_n) P(y_n, y_{\mathcal{P}_n}|\mathbf{X}, \mathbf{\Theta_0})}{\sum\limits_{n|\mathcal{P}_n\neq\emptyset} \sum\limits_{y_n} \sum\limits_{y_{\mathcal{P}_n}} y_{\mathcal{P}_n} P(y_n, y_{\mathcal{P}_n}|\mathbf{X}, \mathbf{\Theta_0})} \tag{4.11}$$

$$\pi = \frac{\sum\limits_{n|\mathcal{P}_n=\emptyset} \sum\limits_{y_n} y_n P(y_n|\mathbf{X}, \mathbf{\Theta_0})}{\sum\limits_{n|\mathcal{P}_n=\emptyset} \sum\limits_{y_n} P(y_n|\mathbf{X}, \mathbf{\Theta_0})} \tag{4.12}$$

$$\mu_c = \frac{\sum\limits_{n\in\mathcal{O}} \mathbf{x}_n P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}{\sum\limits_{n\in\mathcal{O}} P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}, c = 0, 1 \tag{4.13}$$

$$\Sigma_c = \frac{\sum\limits_{n\in\mathcal{O}} (\mathbf{x}_n - \boldsymbol{\mu}_c)(\mathbf{x}_n - \boldsymbol{\mu}_c)^T P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}{\sum\limits_{n\in\mathcal{O}} P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}, c = 0, 1 \tag{4.14}$$

After learning model parameters, we can infer hidden class variables by maximizing the overall probability. A naive approach that enumerates all combinations of class assignments is infeasible due to the exponential cost. I use a dynamic programming-based method called *max-sum* [74]. The process is similar to the sum and product algorithm above. The main difference is that instead of using sum operation, we need to use max operation in message propagation, and also memorize the optimal variable values.

4.3.4 How the Model Address Limited Observations?

The main intuition behind how my model handles limited observations is that the model can capture physical constraints between sample classes. The spatial structural constraints

are derived from the potential field layer that is fully observed on the entire raster framework, regardless of whether non-spatial features are available or not. The topological structure in a split tree is consistent with the physical law of water flow directions on a topographic surface based on gravity. In this sense, even though many samples in the raster framework do not have non-spatial explanatory features observed, we can still infer their classes based on information from the pixels in the upstream or downstream locations. Another potential question is how my model can effectively learn parameters given very limited observations. This question can be answered from the perspective of how model learning works. The major task of model learning is to effectively update parameters of $P(\mathbf{x}_n|y_n)$ for observed pixels in the test region, so that we can infer the posterior class probabilities on these pixels and further infer hidden classes on other pixels. As long as the training samples could give the model a reasonable initial estimate of posterior class probabilities on the observed pixels (e.g., truly dry pixels having a higher probability of being dry), the update of parameters should be effective. This is because that parameter updates are largely weighted average of the sample mean and covariance matrices on fully observed pixels. The weights are the posterior class probability of observed samples.

4.4 Experimental Evaluation

4.4.1 Experiment Setup

In this section, I compared my proposed approach with baseline methods in related works on real-world datasets. Evaluation candidate methods are listed below. Note that I did not include data imputation methods (e.g., filling in mean feature values) due to its low capability of handling very limited observations. Unless specified otherwise, I used default parameters in open source tools for baseline methods. Experiments were conducted on a Dell workstation with Intel(R) Xeon(R) CPU E5-2687w v4 @ 3.00GHz, 64GB main memory, and Windows 10.

- **Label propagation with structure (LP-Structure)**: In the implementation of this baseline method, I used the maximum likelihood classifier (MLC) and GBM respectively to pre-classify fully observed samples and then ran label propagation [102] on the topography tree structure. I named them as **LP-Structure-MLC** and **LP-Structure-GBM**. The initial classifiers were from R packages.

- **EM with i.i.d. assumption (EM-i.i.d.)**: In the implementation of this baseline method [28], I treated missing features and unknown classes as latent variables and used the EM algorithm assuming that sample features follow i.i.d. Gaussian distribution in each class. Moreover, I assumed RGB (red, green, blue) features and elevation features are uncorrelated.

- **EM with structure (EM-Structure)**: This is my proposed approach. I treated unknown classes as latent variables and used the EM algorithm assuming that samples follow the topography tree dependency structure. The codes were implemented in C++.

Data Description: The real-world datasets were collected from Kinston North Carolina and Grimesland North Carolina in Hurricane Matthew 2016. I used aerial imageries from NOAA National Geodetic Survey [65] with red, green, blue bands in a 2-meter spatial resolution and digital elevation map from the University of North Carolina Libraries [67]. The test region size was 1743 by 1349 in Kinston and 2757 by 3853 in Grimesland. The number of observation samples was 31,168 in Kinston and 237,312 in Grimesland. The number of training and testing samples (pixels) are listed in Table 4.2.

Table 4.2: Dataset description

Dataset	Training Set		Testing Set	
	Dry	Flood	Dry	Flood
Matthew, Kinston	5,000	5,000	48,071	47,967
Matthew, Grimesland	5,000	5,000	75,670	59,405

Evaluation Metric: For classification performance evaluation, I used precision, recall, and

F-score. For computational performance evaluation, I measured the computational time costs in seconds.

4.4.2 Classification Performance Evaluation

Table 4.3: Comparison on Mathew, Kinston flood data

Classifiers	Class	Prec.	Recall	F	Avg. F
LP-Structure-GBM	Dry	0.91	0.56	0.69	0.74
	Flood	0.68	0.94	0.79	
LP-Structure-MLC	Dry	0.86	0.55	0.67	0.72
	Flood	0.67	0.91	0.77	
EM-i.i.d.	Dry	1.00	0.39	0.56	0.66
	Flood	0.62	1.00	0.76	
EM-Structure	Dry	0.94	0.99	0.96	0.96
	Flood	0.99	0.94	0.96	

Table 4.4: Comparison on Mathew, Grimesland flood data

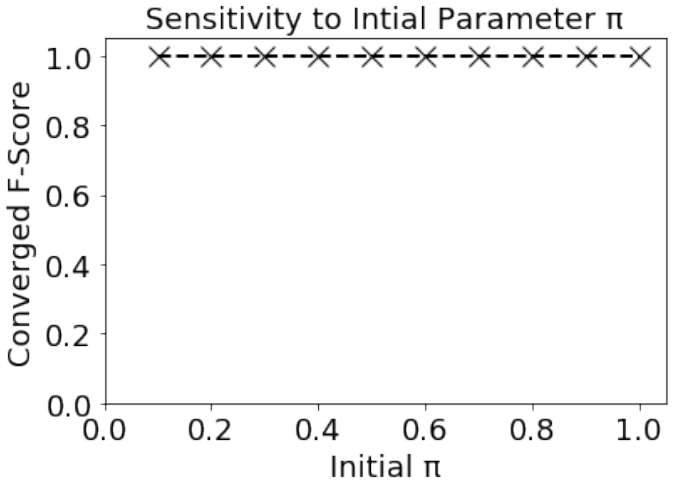
Classifiers	Class	Prec.	Recall	F	Avg. F
LP-Structure-GBM	Dry	0.81	0.60	0.69	0.70
	Flood	0.61	0.82	0.70	
LP-Structure-MLC	Dry	0.90	0.75	0.82	0.81
	Flood	0.73	0.90	0.81	
EM-i.i.d.	Dry	0.83	0.74	0.78	0.77
	Flood	0.71	0.80	0.75	
EM-Structure	Dry	0.99	0.96	0.97	0.97
	Flood	0.95	0.99	0.97	

I first compared different methods on their precision, recall, and F-score on the two real-world datasets. The results were summarized in Table 4.3 and Table 4.4 respectively. On the Kinston dataset, EM algorithm with the i.i.d. assumption performed the worst with an average F-score of 0.66. The reason was that this method was not able to utilize the spatial structural constraint between sample classes. Its training process only updated the parameter of Gaussian feature distribution in each class. When predicting the classes of samples with only elevation feature, the method used only the learned Gaussian distribution of elevation feature on each class without considering spatial structure based on elevation values. On the same dataset, label propagation after pre-classification with the GBM model and the maximum likelihood classifier slightly outperformed the EM algorithm with the i.i.d. assumption. The main reason was that label propagation on the topography tree

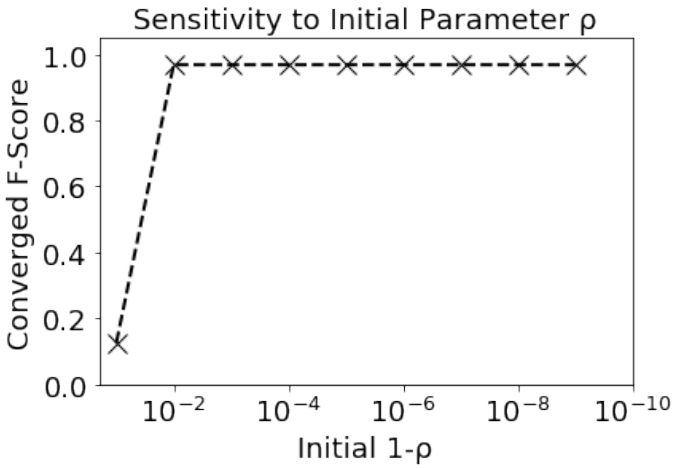
(split tree) structure utilized the physical constraint between sample classes when inferring the classes of unobserved samples without RGB features. However, label propagation still showed significant errors, particularly in the low recall on the dry class. Through analyzing the predicted map, I observed that the label propagation algorithm was very sensitive to the pre-classified class labels on the observed samples in the test region. Errors in the pre-classification phase may propagation into unobserved samples (those without RGB feature values). In label propagation methods, once the errors were propagated to unobserved samples, they were hard to be reverted. This was different from the EM algorithm, which could update the probabilities in iterations. I did not report the results of label propagation on a grid graph structure (only considering spatial neighborhood structure without physics-aware constraint) due to poor results. My model based on the EM algorithm assuming structural dependency between class labels performed the best with an average F-score of 0.96. The main reason was that my model could leverage the physical constraint to infer unobserved samples, and also could effectively update sample probabilities during iterations with the EM algorithm. In my model, I used training samples to initialize the parameters of the Gaussian distribution of sample features in each class. Based on the reasonable initial parameters, I can have a reasonable estimation of the posterior class probabilities of all samples in the test region. Based on the posterior class probabilities, the distribution parameters could be further updated. The representative training samples helped make sure that parameter iterations would converge in the right path.

Similar results were observed on the Grimesland dataset. In the label propagation method, pre-classification based on GBM performed worse than pre-classification based on MLC. The reason may be due to overfitting of GBM compared with MLC when predicting initial labels on the fully observed samples. The EM algorithm with the i.i.d. assumption performed slightly better on this dataset. The reason might be that the final prediction of classes of the unobserved samples (with only elevation feature but without RGB features) was based on a slightly better fitted normal distribution. My model showed the best per-

formance with an F-score of 0.97.



(a)



(b)

Figure 4.5: Sensitivity of my model to initial parameters π and ρ

The effect of model initial parameters

I analyzed the sensitivity of my proposed model on different initial parameter settings. The parameters of $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ were estimated from training data, but parameters ρ and π were from user input. Since ρ captured the transitional probability of a sample being flood given its parents were all flood, its value should be very high (close to 1) due to spatial autocorrelation. π is the initial class prior probabilities for samples without parent nodes

(local lowest location). We could set it close to 0.5. I tested the sensitivity of my model to different initial values of ρ and π on the Kinston dataset. I first fixed ρ as 0.999 and varied the value of π from 0.1 to 0.9. Then, I fixed π as 0.3 and varied the value of ρ from 0.9. The results were shown in Figure 4.5. We can see that the model was generally not very sensitive to the initial parameter values. For parameter ρ, as long as $1 - \rho$ was smaller than 0.01 (ρ greater than 0.99), the converged F-score was good. For parameter π, the results were consistently good for my model with an initial π between 0.1 to 0.9. The main reason was that π influenced only a small number of samples at the local lowest locations on the elevation map.

The parameter iterations of my model were shown in Figure 4.6. The model converged fast with only 20 iterations.

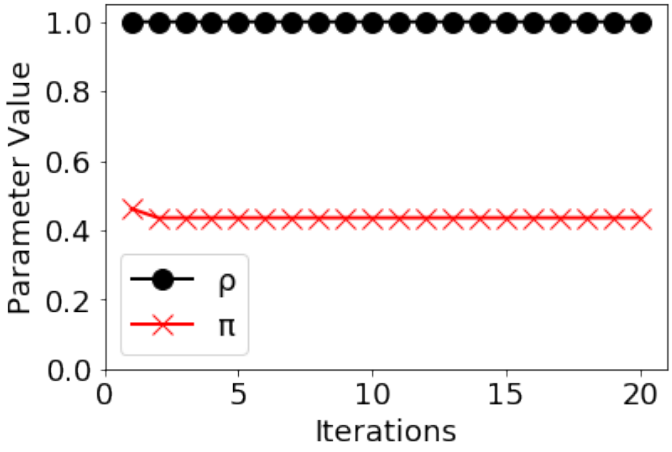


Figure 4.6: Parameter iterations and convergence in my model

4.4.3 Computational Performance Evaluation

I also evaluated the computational performance of my model learning and inference on different input data sizes. I used the Grimesland dataset to test the effect of different test region sizes. I varied the region size from around 2 million pixels to over 10 million pixels. The computational time costs of my model was shown in Figure 4.7. It can be seen that the time cost grows almost linearly with the size of the test region. This was because my learning and inference algorithms involve tree traversal operations with a linear time

complexity on the tree size (the number of pixels on the test region). The model was computationally efficient. It could classify around 10 million pixels in around 2 minutes.

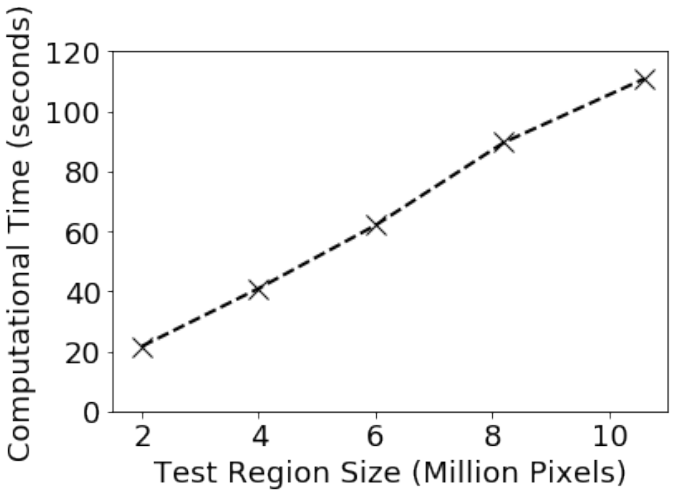


Figure 4.7: Computational performance of out model on varying test region sizes

I further analyzed the time costs of different components in my model, including split tree construction, model parameter learning, and class inference. I analyzed the results on both datasets (same as the settings in Table 4.3 and Table 4.4. Results showed that tree construction and class inference took less time than parameter learning. This was because the learning involves multiple iterations of message propagation (tree traversal operations).

Table 4.5: Time costs analysis of my model (seconds)

	Kinston	Grimesland
Tree construction	3.2	8.39
Parameter learning	25.74	86.79
Class inference	3.8	15.62
Total time	32.74	110.80

4.5 Summary

In this chapter, I address the problem of spatial classification with limited feature observations. The problem is important in many applications where only a subset of sensors are deployed at certain regions or partial responses are collected in field surveys. Existing re-

search on incomplete or missing data has limitations in assuming that incomplete feature observations only happen on a small subset of samples, and thus cannot solve problems whereby the vast majority of samples have missing feature observations. To address this issue, I introduce my new approach that incorporates physics-aware structural constraints into model representation. I develop efficient algorithms for model parameter learning and class inference. Evaluations on real-world hydrological applications show that my approach significantly outperforms several baseline methods in classification accuracy, and the proposed solution is computationally efficient on a large data volume.

CHAPTER 5

ADDRESSING SPATIAL DEPENDENCY CHALLENGE: A HIDDEN MARKOV CONTOUR TREE MODEL FOR SPATIAL STRUCTURAL PREDICTION

5.1 Introduction

Spatial structured models are predictive models that capture dependency structure between samples based on their locations. Given data samples in a spatial raster framework with explanatory feature layers and a potential field layer, as well as an independent set of training samples with class labels, the spatial structured learning problem aims to learn a model that can predict sample classes in the same framework [39]. For example, in flood extent mapping from earth imagery, data samples are imagery pixels in a regular grid, the explanatory feature layers are spectral bands, and the potential field can be elevation that controls water flow directions. The goal is to predict the classes (flood or dry) of pixels based on not only the spectral features but also the implicit flow directions on an elevation surface.

The problem is important in many societal applications such as flood extent mapping for disaster response and national water forecasting.Flood extent mapping plays a crucial role in addressing grand societal challenges such as disaster management, national water forecasting, as well as energy and food security. For example, during Hurricane Harvey floods in 2017, first responders needed to know where flood water was in order to plan rescue efforts. In national water forecasting, detailed flood extent maps can be used to calibrate and validate the NOAA National Water Model [66], which can forecast the flow of over 2.7 million rivers and streams through the entire continental U.S. [22]. In current practice, flood extent maps are mostly generated by flood forecasting models, whose accuracy is of-

ten unsatisfactory in high spatial details [22]. Other ways to generate flood maps involve sending a field crew on the ground to record high-water marks, or visually interpreting earth observation imagery [16]. However, the process is both expensive and time consuming. With the large amount of high-resolution earth imagery being collected from satellites (e.g., DigitalGlobe, Planet Labs), aerial planes (e.g., NOAA National Geodetic Survey), and unmanned aerial vehicles, the cost of manually labeling flood extent becomes prohibitive. Note that though I use flood mapping as a motivation example, the problem can potentially be applied to many other applications such as water quality monitoring and air pollution mapping in which pollutants are transmitted following flow directions [54], protein structure learning in biochemistry [32], and geometric shape analysis in computer graphics [70].

However, the problem poses several unique challenges that are not well addressed in traditional classification problems. First, implicit spatial dependency structure exists between pixel locations. For example, due to gravity, flood water tends to flow from one location to nearby lower locations. Such dependency structure is complex, following contour patterns on a 3D surface. Second, data contains rich noise and obstacles. For example, high-resolution earth imagery often has noise, clouds and shadows. In addition, the spectral features of image pixels can be insufficient to distinguish classes (also called class confusion) due to heterogeneity. For instance, pixels of tree canopies overlaying flood water have the same spectral features with those trees in dry areas, yet their classes are different. Finally, the problem is also computationally challenging due to the cost of modeling complex spatial structure on a large data volume (e.g., millions of sample locations).

Existing spatial structured models often assume that the dependency structure is based on spatial proximity or spatial network topology. Models based on spatial proximity assume that adjacent or nearby locations have stronger dependency. The assumption comes from the first law of geography [96], "everything is related to everything else, but near things are more related than distant things". Examples include Markov random field [5] , condi-

tional random field [55], spatial network embedding [103, 115, 127, 104, 105], and convolutional neural networks [114]. Other works incorporate spatial structures by spatial regularization on loss function (to penalize difference between nearby locations) together with efficient optimization [91, 50, 90]. Models based on spatial network topology assume that dependency between sample locations only follows an underlying spatial network structure. These models are useful for samples that are only located along spatial networks (e.g., traffic accidents along road networks). Examples of such models include spatial network Kriging [5, 9] and spatial network autoregressive models [101]. In summary, related works often capture undirected spatial dependency structure based on distance or network topology. Complex dependency structure that is both directed and prevalent at all locations in the continuous space are largely unexplored. Recently, a geographical hidden Markov tree (HMT) model [111] has been proposed, which captures directed spatial dependency based on flow directions across all locations, but it is largely motivated by a one dimensional spatial view and does not consider complex spatial structures such as contours on a 3D surface.

To fill the gap, I introduce my recently proposed approach, a novel spatial structure model called hidden Markov contour tree (HMCT) [42] [81]. It is a probabilistic graphical model that generalizes the common hidden Markov model (HMM) from a total order sequence to a partial order polytree. Specifically, the hidden class layer contains nodes (pixels) in a contour tree structure to reflect flow directions between all locations on a 3D surface. I also introduced efficient learning algorithms based on contour tree node collapsing and value pre-aggregation. Preliminary results showed that HMCT significantly outperforms several baseline methods on real-world flood mapping datasets and the proposed model is scalable to a large data volume [42]. In this chapter, I further extend my preliminary results with the following additional contributions.

- I introduce an extension of the HMCT model [42] from generative to discriminative so that the model can be used as a post-processor (HMCT-PP) [81]. I analyze the

theoretical properties of the extended model.

- I conduct evaluations on real-world flood mapping datasets. Results show that HMCT-PP can significantly enhance the results from baseline methods.
- I also conduct sensitivity analysis of the proposed approaches to initial parameters. I also interpret the results through visualization and analysis.

5.2 Problem Statement

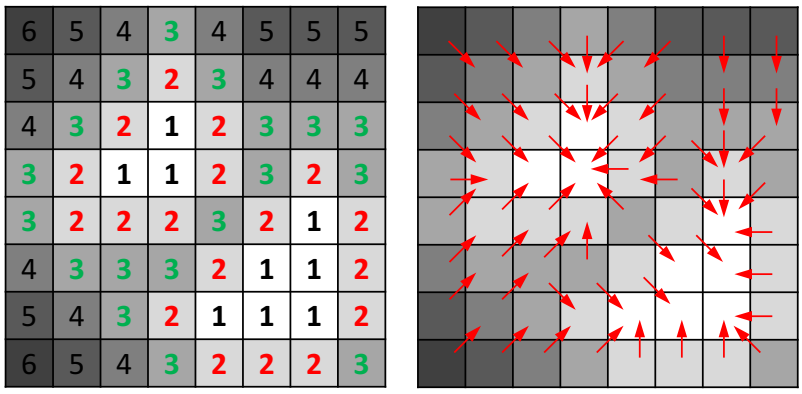
5.2.1 Preliminaries

Definition 1. *A* spatial raster framework *is a tessellation of a two dimensional plane into a regular grid of N cells. Spatial neighborhood relationship exists between cells based on cell adjacency. The framework can contain m non-spatial explanatory feature layers (e.g., spectral bands in earth imagery), one potential field layer (e.g., elevation), and one class layer (e.g.,* flood*,* dry*).*

Definition 2. *Each cell in a raster framework is a* spatial data sample*, noted as $\mathbf{s_n} = (\mathbf{x}_n, \phi_n, y_n)$, where $n \in \mathbb{N}, 1 \leq n \leq N$, $\mathbf{x}_n \in \mathbb{R}^{m \times 1}$ is a vector of m non-spatial explanatory feature values with each element corresponding to one feature layer, $\phi_n \in \mathbb{R}$ is a cell's potential field value, and $y_n \in \{0, 1\}$ is a binary class label.*

A raster framework with all samples is noted as $\mathcal{F} = \{\mathbf{s_n} | n \in \mathbb{N}, 1 \leq n \leq N\}$, non-spatial explanatory features of all samples are noted as $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$, the potential field layer is noted as $\mathbf{\Phi} = [\phi_1, ..., \phi_N]^T$, and the class layer is noted as $\mathbf{Y} = [y_1, ..., y_N]^T$.

Definition 3. *A* contour *in a raster framework is a set of contiguous samples whose potential field values are equal. For example, in Figure 5.1(a), there are two contours for the elevation value* 1*. Note that my definition of contour here is discrete. Original definition based on a continuous field in topology can be found in [63, 25].*



(a) Elevation values with contours

(b) Flow directions

Figure 5.1: Example of raster framework and flow dependency

Definition 4. *Spatial flow dependency exists between cells following the gradient on the potential field layer. Formally, a* flow dependency $\mathbf{s}_i \rightsquigarrow \mathbf{s}_j$ *exists if and only if there exist a sequence of neighboring (adjacent) cells* $< \mathbf{s}_i, \mathbf{s}_{p_1}, \mathbf{s}_{p_2}, ..., \mathbf{s}_{p_l}, \mathbf{s}_j >$ *such that* $\phi_i \leq \phi_{p_1}$, $\phi_{p_l} \leq \phi_j$, *and* $\phi_{p_k} \leq \phi_{p_{k+1}}$ *for any* $1 \leq k \leq l-1$. *For example, due to gravity, flood water can flow from cells with elevation* 3 *to neighboring cells with elevation* 2 *in Figure 5.1(a).*

5.2.2 Formal problem definition

I now formally define the spatial structured learning problem.

Input:

- Spatial raster framework $\mathcal{F} = \{\mathbf{s}_n | n \in \mathbb{N}, 1 \leq n \leq N\}$
- Explanatory features of samples $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$
- Spatial potential field values of samples: $\mathbf{\Phi} = [\phi_1, ..., \phi_N]^T$
- Training samples $\{\mathbf{s}_k | k \in \textit{training set}\}$ outside $\mathcal{F}$

Output: A spatial structured model $f : \mathbf{Y} = f(\mathbf{X})$

Objective: minimize classification errors

Constraint:

- Explanatory feature layers contain noise and obstacles
- Sample classes follow partial order flow dependency
- Sample class is binary, $y_n \in \{0, 1\}$

5.3 Proposed Approach

The section introduces a novel spatial structured model called hidden Markov contour tree together with effective and efficient learning algorithms.

5.3.1 Modeling Spatial Contour Structure

My goal is to explore a data structure to represent flow dependency between pixel locations on a potential field (e.g., 3D elevation surface). Such dependency structure can be represented by *contour tree* [18, 25]. A contour tree is a polytree (directed acyclic graph whose underlying undirected graph is a tree) that captures the evolution of contours (level sets) of a potential field. Figure 5.2(b) shows an example based on the elevation surface in Figure 5.1. Assume, we increase an elevation threshold from 1 to 6. For threshold 1, two separate contours appear, corresponding to the two leaf nodes on the bottom of Figure 5.2(b). For threshold 2, the two contours both grow bigger but remain separated. For threshold 3, the two contours merge into one (corresponding to the central node with number 3), and a new separate contour appears based on the bottom right pixel in Figure 5.1(a) (corresponding to the tree node 3 on a side branch). For threshold 4, a contour of elevation 3 splits into three. Then, each contour grows separately as the threshold further increases. Contour tree is naturally a good representation for flow direction dependency between locations on an elevation surface. For example, in Figure 5.2(b), if the central node with elevation 3 is in the *flood* class, all its parent nodes (the ones with elevations 1 and 2 below it) must be in the *flood* class, since flood water flows from a high elevation to nearby lower elevations.

Existing research on contour tree often focuses on the critical nodes (such as local minimum and maximum, saddle point) [63, 25]. Tree construction algorithms often require an input elevation surface to be represented as a mesh with unique values [18] . In order to run the algorithms on an elevation surface with duplicated values, we can use perturbation to enforce an arbitrary order on locations with an equal elevation. This makes the tree unnecessarily large, as shown in Figure 5.2(a). To address this limitation, we introduce a

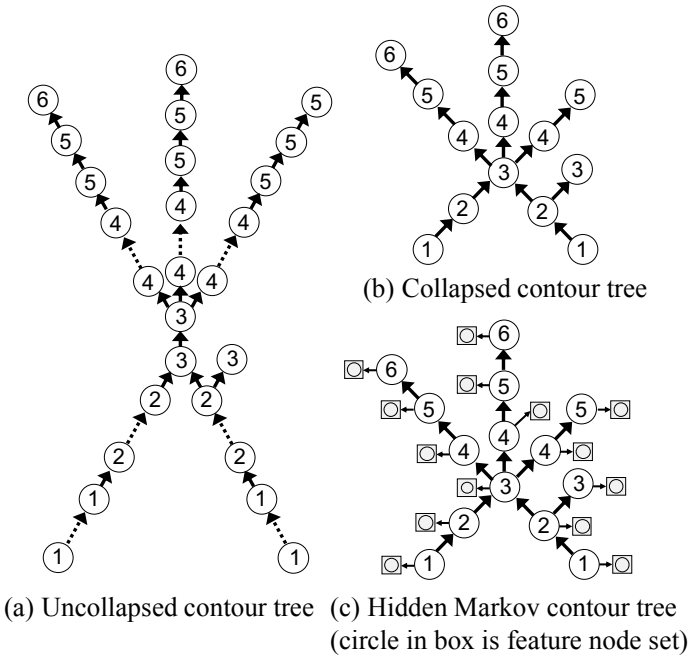


Figure 5.2: Illustration of hidden Markov contour tree

recently proposed collapsed contour tree, which starts from a contour tree from existing algorithm but then collapses nodes in the same contour. Figure 5.2(a-b) shows an example. For instance, two connected nodes with elevation 3 in Figure 5.2(a) are merged into one in Figure 5.2(b). Node collapsing can dramatically reduce the size of a contour tree, potentially reducing the cost of learning and inference algorithms. Algorithm 5 shows the detailed steps for node collapsing. The main idea is to use breadth-first search to find each collapsable contour component, and then to collapse tree nodes within that component. The time complexity is $O(N)$ where N is the total number of nodes in the uncollapsed contour tree.

5.3.2 Hidden Markov Contour Tree (HMCT)

I introduce my recently proposed a hidden Markov contour tree (HMCT), a probabilistic graphical model that generalizes the common hidden Markov model from a total order sequence to a partial order polytree [42] [81]. A HMCT model consists of two layers: a hidden class layer in the form of a collapsed contour tree, and an observation feature layer. Each contour tree node in the hidden class layer represents a same unknown hidden class

Algorithm 5 Collapse Contour Tree

Input:

• An uncollapsed contour tree

Output:

• A collapsed contour tree

Initialize all nodes as *unvisited*
for each tree node $\mathbf{s_n}$ by topological order **do**
  if $\mathbf{s_n}$ is *unvisited* **then**
    Mark $\mathbf{s_n}$ as *visited*
    Breadth First Search (BFS) from $\mathbf{s_n}$ to its contour
    Collapse BFS traversed nodes, mark them *visited*
return the collapsed contour tree

shared over all pixels in the contour. Each contour tree edge represents the class transitional probability between pixels from two contours based on flow dependency. A hidden class node in contour tree is connected to a set of observed feature nodes (circle in boxes in Figure 5.2(c)), which correspond to the explanatory feature vectors of all pixels in that contour.

The joint distribution of all samples' features and classes can be formulated as Equation 5.1, where $\mathcal{P}_n$ is the set of parent samples of the nth sample in the dependency tree ($\mathcal{P}_n = \emptyset$ for a leaf node), and $y_{k\in\mathcal{P}_n} \equiv \{y_k | k \in \mathcal{P}_n\}$ is the set of parent classes of node n. $\mathbf{x}_n = \{\mathbf{x}_{n_i} | 1 \leq i \leq N_n\}$ is the set of feature nodes corresponding to hidden class node n (i.e., $\mathbf{x}_{n_i}$ is the feature vector for a pixel n_i in the nth contour). Note that my notation of $\mathbf{x}_n$ here represents a set, and should not be confused with notations in Section 5.2.

$$P(\mathbf{X}, \mathbf{Y}) = P(\mathbf{X}|\mathbf{Y})P(\mathbf{Y}) = \prod_{n=1}^{N}\prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n) \prod_{n=1}^{N} P(y_n|y_{k\in\mathcal{P}_n}) \tag{5.1}$$

For simplicity, observed feature nodes are assumed conditionally independent given their hidden class node, following an i.i.d. Gaussian distribution, as shown in Equation 5.2, where $\boldsymbol{\mu}_{y_n}$ and $\boldsymbol{\Sigma}_{y_n}$ are the mean and covariance matrix of feature vector $\mathbf{x}_{n_i}$ for class y_n ($y_n = 0, 1$).

$$P(\mathbf{x}_{n_i}|y_n) \sim \mathcal{N}(\boldsymbol{\mu}_{y_n}, \boldsymbol{\Sigma}_{y_n}) \tag{5.2}$$

Class transitional probability follows the partial order flow dependency constraint. For example, due to gravity, if any parent's class is *dry*, the child's class must be *dry*; if all parents' classes are *flood*, then the child has a high probability of being *flood* due to spatial autocorrelation. Consider *flood* as the positive class (class 1) and *dry* as the negative class (class 0), the product of all parents' classes is $y_{\mathcal{P}_n} \equiv \prod_{k \in \mathcal{P}_n} y_k$. Class transitional probability can be modeled as Table 5.1, where ρ is a parameter close to 1. The prior class probability of a node without parents is also shown on the right of Table 5.1, where π is another parameter.

Table 5.1: Class transition probability and prior probability

$P(y_n \mid y_{\mathcal{P}_n})$	$y_{\mathcal{P}_n} = 0$	$y_{\mathcal{P}_n} = 1$
$y_n = 0$	1	$1 - \rho$
$y_n = 1$	0	ρ

	$P(y_n)$
$y_n = 0$	$1 - \pi$
$y_n = 1$	π

5.3.3 HMCT Learning and Inference

The parameters of hidden Markov contour tree include the mean and covariance matrix of sample features in each class, prior probability of leaf node classes, and class transition probability for non-leaf nodes. We denote the entire set of parameters as $\mathbf{\Theta} = \{\rho, \pi, \boldsymbol{\mu}_c, \mathbf{\Sigma}_c | c = 0, 1\}$. Learning the set of parameters poses two major challenges: first, there exist unknown hidden class variables $\mathbf{Y} = [y_1, ..., y_N]^T$, which are non-i.i.d.; second, the number of node variables can be large.

To address these challenges, I propose to use the expectation-maximization (EM) algorithm and message (belief) propagation. My EM-based approach has the following major steps:

(a) Initialize parameter set $\mathbf{\Theta_0}$

(b) Compute posterior distribution of hidden classes:

$P(\mathbf{Y}|\mathbf{X}, \mathbf{\Theta_0})$

(c) Compute posterior expectation of log likelihood:

$LL(\mathbf{\Theta}) = \mathbb{E}_{\mathbf{Y}|\mathbf{X},\mathbf{\Theta_0}} \log P(\mathbf{X}, \mathbf{Y}|\mathbf{\Theta})$

(d) Update parameters:

$\mathbf{\Theta_0} \leftarrow_{\mathbf{\Theta}} LL(\mathbf{\Theta})$

Return $\mathbf{\Theta_0}$ if it's converged, otherwise goto (b)

Algorithm 6 shows the details. First, I initialize parameters either with random values within reasonable range (for ρ and π) or with initial estimates based on training samples (e.g., the mean and covariance of features in each class for $\boldsymbol{\mu}_c$ and $\mathbf{\Sigma}_c$). Then I conduct a breadth-first search on the tree from any start node as a root. After this, the algorithm starts the iteration till parameters converge. In each iteration, it propagates messages first from leaves to root (steps 6-7) and then from root to leaves (steps 8-9). Marginal posterior distribution of node classes are then computed (steps 10-11). Based on this, the algorithm updates parameters (step 12). Message propagation is based on the sum and product algorithm [53, 76]. Propagation of message along nodes in a graph (or tree) is equivalent to marginalizing out node variables in the overall joint distribution in Equation 5.1.
Figure 5.3 illustrates the forward message propagation process from leaves to root (denoted as f). Each node can have only one outgoing message (denoted as $f^o_n(y_n)$, but can have multiple incoming messages (incoming messages from a child c and the parent side are denoted as $f^i_{n\leftarrow child\ c}(y_n)$ and $f^i_{n\leftarrow parent}(y_n)$ respectively). As illustrated in Figure 5.3(a), computing $f^i_{n\leftarrow child\ c}(y_n)$ involves integration (sum) over the product of outgoing messages from the child c and c's other parents, together with the conditional probability between c and c's all parents, as specified in (3). The outgoing message from node n has two cases: to its child c or to its parents. In the first case, we first need to compute another incoming message from parents $f^i_{n\leftarrow parent}(y_n)$ based on (4). Then the outgoing message to a child c_0 can be computed based on (5). These are also illustrated in Figure 5.3(a). In the second case, we only need to compute outgoing message to parent based on (6) (also illustrated in Figure 5.3(b)). Note that we denote $P(\mathbf{x}_n|y_n) \equiv \prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n)$ in these equations for brevity.

Algorithm 6 EM Algorithm for Hidden Markov Tree

Input:

• $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]^T$: cell sample feature matrix

• $\mathcal{T}$: a contour tree for spatial dependency

• ϵ: parameter convergence threshold

Output:

• $\mathbf{\Theta} = \{\rho, \pi, \boldsymbol{\mu}_c, \mathbf{\Sigma}_c | c = 0, 1\}$: set of model parameters

Initialize $\mathbf{\Theta_0}$, $\mathbf{\Theta}$

Find a node n_0 in $\mathcal{T}$ as the root

Do a breadth-first search (BFS) on $\mathcal{T}$ from root n_0

while $\|\mathbf{\Theta_0} - \mathbf{\Theta}\|_\infty > \epsilon$ **do**

  $\mathbf{\Theta_0} \leftarrow \mathbf{\Theta}$

  for each y_n from leaves to root in BFS **do**

    Compute messages $f_n^i(y_n), f_n^o(y_n)$ by (3)-(6)

  for each y_n from root to leaves in BFS **do**

    Compute messages $g_n^i(y_n), g_n^o(y_n)$ by (7)-(12)

  for each $y_n, 1 \leq n \leq N$ **do**

    // Compute marginal distributions:

   $P(y_n|\mathbf{X}, \mathbf{\Theta_0}), P(y_n, y_{k\in\mathcal{P}_n}|\mathbf{X}, \mathbf{\Theta_0})$

  Update $\mathbf{\Theta}$ based on marginal distributions:

  $\mathbf{\Theta} \leftarrow_{\mathbf{\Theta}} \mathbb{E}_{\mathbf{Y}|\mathbf{X},\mathbf{\Theta_0}} \log P(\mathbf{X}, \mathbf{Y}|\mathbf{\Theta})$ by (13)-(16)

return Θ

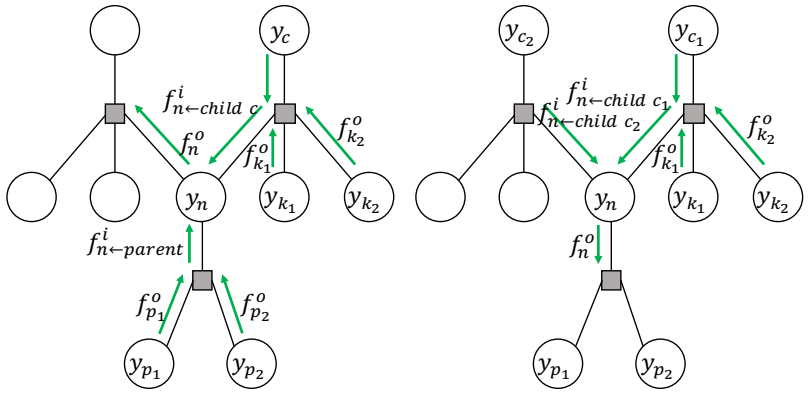
$$f^i_{n\leftarrow child\ c}(y_n) = \sum_{y_c, y_{\{k\in\mathcal{P}_c, k\neq n\}}} P(y_c|y_{k\in\mathcal{P}_c}) f^o_c(y_c) \prod_{k\in\mathcal{P}_c, k\neq n} f^o_k(y_k) \tag{5.3}$$

$$f^i_{n\leftarrow parent}(y_n) = \begin{cases} P(y_n) & \text{if } n \text{ no parent} \\ \sum_{y_{k\in\mathcal{P}_n}} P(y_n|y_{k\in\mathcal{P}_n}) \prod_{k\in\mathcal{P}_n} f^o_k(y_k) & \text{otherwise} \end{cases} \tag{5.4}$$

$$f^o_n(y_n) = P(\mathbf{x}_n|y_n) f^i_{n\leftarrow parent}(y_n) \prod_{c\in\mathcal{C}_n, c\neq c_0} f^i_{n\leftarrow child\ c}(y_n) \tag{5.5}$$

$$f^o_n(y_n) = P(\mathbf{x}_n|y_n) \prod_{c\in\mathcal{C}_n} f^i_{n\leftarrow child\ c}(y_n) \tag{5.6}$$

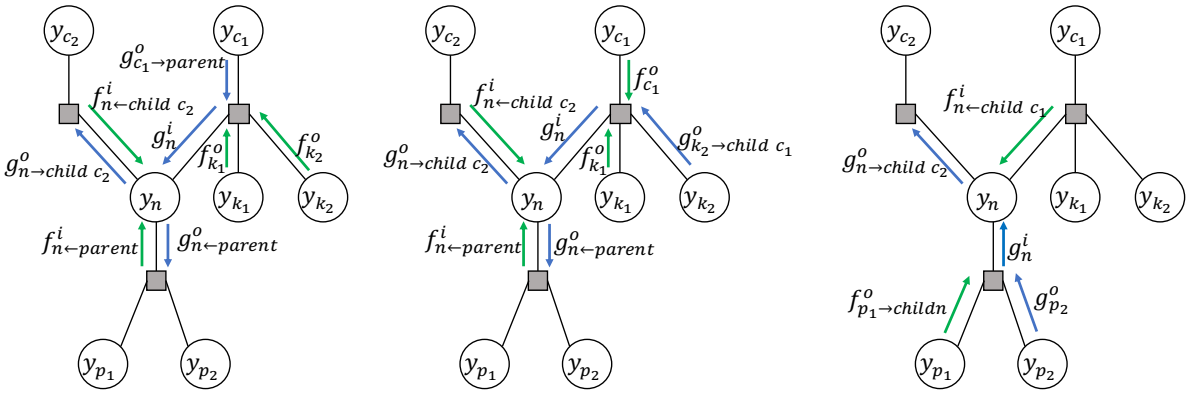
Backward message propagation from root to leaves also follows a recursive process. Each node only has one incoming message but several outgoing messages. There are three cases as shown in Figure 5.4(a-c): incoming message $g_n^i(y_n)$ can be from a child, a child's parent, or a parent. In the first two cases, $g_n^i(y_n)$ is computed from outgoing messages on the child



(a) Case 1: f out to child c_1 (b) Case 2: f out to parent

Figure 5.3: Illustration of message propagation leaves to root

side based on (7) and (8) respectively. The outgoing messages to another child or parent can be computed based on (9) and (10) for both of the first two cases. In the third case, $g^i_n(y_n)$ is computed based on outgoing messages from the parent side, as specified in (11). The outgoing message to a child can be computed based on (12). Since in the third case, the incoming message is from the parent side, we do not need to compute outgoing message to parent $g^o_{n\rightarrow parent}(y_n)$.



(a) Case 1: g from child (b) Case 2: g from a child's parent (c) Case 3: g from parent

Figure 5.4: Illustration of message propagation leaves to root

$$g^i_n(y_n) = \sum_{y_{c_0}, y_{p\in\mathcal{P}_{c_0}}, p\neq n} g^o_{c_0\rightarrow parent}(y_{c_0}) P(y_{c_0}|y_{p\in\mathcal{P}_{c_0}}) \cdot \prod_{p\in\mathcal{P}_{c_0}, p\neq n} f^o_p(y_p) \tag{5.7}$$

$$g_n^i(y_n) = \sum_{y_{c_0}, y_{\{p \in \mathcal{P}_{c_0}, p \neq n\}}} g^o_{p_{c_0} \to child\ c_0}(y_{p_{c_0}}) f^o_{c_0}(y_{c_0}) P(y_{c_0}|y_{k \in \mathcal{P}_{c_0}}) \prod_{p \in \mathcal{P}_{c_0}, p \neq p_{c_0}, p \neq n} f^o_p(y_p) \tag{5.8}$$

$$g^o_{n \to child\ c}(y_n) = g^i_n(y_n) P(\mathbf{x}_n|y_n) f^i_{n \leftarrow parent}(y_n) \prod_{c' \in \mathcal{C}_n, c' \neq c_0, c' \neq c} f^i_{n \leftarrow child\ c'}(y_n) \tag{5.9}$$

$$g^o_{n \to parent}(y_n) = g^i_n(y_n) P(\mathbf{x}_n|y_n) \prod_{c \in \mathcal{C}_n, c \neq c_0} f^i_{n \leftarrow child\ c}(y_n) \tag{5.10}$$

$$g^i_n(y_n) = \sum_{y_{k \in \mathcal{P}_n}} P(y_n|y_{k \in \mathcal{P}_n}) g^o_{p_0 \to n}(y_{p_0}) \prod_{p \in \mathcal{P}_n} f^o_p(y_p) \tag{5.11}$$

$$g^o_{n \to child\ c}(y_n) = g^i_n(y_n) P(\mathbf{x}_n|y_n) \prod_{c' \in \mathcal{C}_n, c' \neq c} f^i_{n \leftarrow child\ c'}(y_n) \tag{5.12}$$

After both forward and backward message propagation, we can compute marginal posterior distribution of hidden class variables. The unnormalized marginal posterior distribution of the class of a leaf node (without parents), as well as the class of a non-leaf node (with parents) can be computed by multiplying all messages coming into these nodes. Results can be normalized by dividing them over the total sum of all variable configurations. After computing the marginal posterior distribution, we can update model parameters by maximizing the posterior expectation of log likelihood (the maximization or M step in EM). Taking the marginal posterior distributions computed above together with the prior and transitional probabilities in Table 5.1 into the posterior expectation, we can get the following parameter update formulas (the M step in EM).

$$\rho = \frac{\sum_{n|\mathcal{P}_n \neq \emptyset} \sum_{y_n} \sum_{y_{\mathcal{P}_n}} y_{\mathcal{P}_n} y_n P(y_n, y_{\mathcal{P}_n}|\mathbf{X}, \mathbf{\Theta_0})}{\sum_{n|\mathcal{P}_n \neq \emptyset} \sum_{y_n} \sum_{y_{\mathcal{P}_n}} y_{\mathcal{P}_n} P(y_n, y_{\mathcal{P}_n}|\mathbf{X}, \mathbf{\Theta_0})} \tag{5.13}$$

$$\pi = \frac{\sum_{n|\mathcal{P}_n = \emptyset} \sum_{y_n} y_n P(y_n|\mathbf{X}, \mathbf{\Theta_0})}{\sum_{n|\mathcal{P}_n = \emptyset} \sum_{y_n} P(y_n|\mathbf{X}, \mathbf{\Theta_0})} \tag{5.14}$$

$$\boldsymbol{\mu}_c = \frac{\sum_{n=1}^{N} \sum_{i=1}^{N_n} \mathbf{x}_{n_i} P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}{\sum_{n=1}^{N} \sum_{i=1}^{N_n} P(y_n = c|\mathbf{X}, \mathbf{\Theta_0})}, c = 0, 1 \tag{5.15}$$

$$\boldsymbol{\Sigma}_c = \frac{\sum_{n=1}^{N}\sum_{i=1}^{N_n}(\mathbf{x}_{n_i}-\boldsymbol{\mu}_c)(\mathbf{x}_{n_i}-\boldsymbol{\mu}_c)^T P(y_n=c|\mathbf{X},\boldsymbol{\Theta_0})}{\sum_{n=1}^{N}\sum_{i=1}^{N_n} P(y_n=c|\mathbf{X},\boldsymbol{\Theta_0})}, c=0,1 \tag{5.16}$$

Class inference: After learning model parameters, we can infer hidden class variables by maximizing the overall probability. A naive approach that enumerate all combinations of class assignment is infeasible due to the exponential cost. I use a dynamic programming based method called *max-sum* [74]. The process is similar to the sum and product algorithm above. The main difference is that instead of using sum operation, we need to use max operation in message propagation, and also memorize the optimal variable values.

5.3.4 Computational Performance Tuning

Time complexity of Algorithm 6: The cost includes contour tree traversal, message calculation, marginal probability calculation, and parameter update. Tree traversal cost is $O(N_T)$ where N_T is the number of collapsed contour tree nodes. Message calculation for the nth contour tree node is $O(N_n 2^d)$, where N_n is the number of feature nodes for tree node n, and d is the in-degree. The exponential term 2^d comes from the sum of products on all parent combinations. The term N_n comes from the aggregation over individual feature nodes in $P(\mathbf{x}_n|y_n) \equiv \prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n)$ as well as in updating $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$. Thus, the cost per iteration is $O(N2^d)$ where N is the total number of feature nodes or pixels ($N \gg N_T$). The overall cost is $O(N2^dI)$ where I is the number of iterations.

From the analysis above, we can see that the total cost is linear to the total number of feature nodes N instead of the number of collapsed contour tree nodes N_T, due to the need of aggregating over individual feature nodes. To improve computational efficiency, I **pre-aggregate** feature node values for each contour tree node (class node) once, and later to use the pre-aggregated results in message calculation. In this way, the remaining cost after pre-aggregation is $O(N_T)$ instead of $O(N)$. Pre-aggregation can be done during contour tree node collapsing.

The next question is how to do the pre-aggregation correctly (returning the same final results). In fact, there are only three parts involving aggregation over feature nodes: calcu-

lating $P(\mathbf{x}_n|y_n)$ in Equation 5.17, updating $\boldsymbol{\mu}_c$ in Equation 5.15, and updating $\boldsymbol{\Sigma}_c$ in Equation 5.16. My goal here is to compute these three parts based on pre-aggregated feature nodes over each class node ($\mathbf{x}_{n_i}$ with $1 \leq n_i \leq N_n$). The specific process is described in Theorem 5.3.1.

$$\begin{aligned}\log P(\mathbf{x}_n|y_n) = \log \prod_{i=1}^{N_n} P(\mathbf{x}_{n_i}|y_n) = -\frac{N_n}{2}(m\log(2\pi) + \log|\Sigma_{y_n}|) \\ -\frac{1}{2}\sum_{i=1}^{N_n}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)^T\Sigma_{y_n}^{-1}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)\end{aligned} \tag{5.17}$$

Theorem 5.3.1. *Assume* $A_n = \sum_{i=1}^{N_n}\mathbf{x}_{n_i}\mathbf{x}_{n_i}^T$, $B_n = \sum_{i=1}^{N_n}\mathbf{x}_{n_i}$, $C_n = N_n$. *We can calculate Equation 5.15, Equation 5.16 and Equation 5.17 based on pre-aggregated results* A_n, B_n *and* C_n *on each class node* n.

Proof. Equation 5.18, Equation 5.19, and Equation 5.20 show how to calculate Equation 5.15, Equation 5.16 and Equation 5.17 based on pre-aggregated values A_n, B_n and C_n. Note that $\langle\cdot,\cdot\rangle$ is inner product between two matrices.

$$\sum_{i=1}^{N_n}\mathbf{x}_{n_i} = B_n \tag{5.18}$$

$$\sum_{i=1}^{N_n}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)(\mathbf{x}_{n_i} - \boldsymbol{\mu}_c)^T = A_n - B_n\boldsymbol{\mu}_c^T - (B_n\boldsymbol{\mu}_c^T)^T + C_n\boldsymbol{\mu}_c\boldsymbol{\mu}_c^T \tag{5.19}$$

$$\begin{aligned}\sum_{i=1}^{N_n}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_{y_n})^T\Sigma_{y_n}^{-1}(\mathbf{x}_{n_i} - \boldsymbol{\mu}_{y_n}) = \langle\Sigma_{y_n}^{-1}, A_n - B_n\boldsymbol{\mu}_{y_n}^T - (B_n\boldsymbol{\mu}_{y_n}^T)^T \\ + N_n\boldsymbol{\mu}_{y_n}\boldsymbol{\mu}_{y_n}^T\rangle\end{aligned} \tag{5.20}$$

□

Time complexity of refined algorithm: Pre-aggregation part (i.e., computing A_n, B_n, and C_n for all contour tree node n) is $O(N)$, but this can be done for only once during contour tree construction and node collapsing. After pre-aggregation, the time complexity for learning and inference part is reduced from $O(2^dN \cdot I)$ to $O(2^dN_T \cdot I)$ where d is node in-degree. Since N_T is the number of contour tree nodes after node collapsing, it is significantly smaller than the total number of pixels N.

5.3.5 Using HMCT for Post-processing

The original hidden Markov contour tree model is a generative model based on the joint probability distribution of both sample classes and features. Sample feature distribution is conditioned on hidden class nodes (i.e., $P(\mathbf{x}_n|y_n)$). Sometimes in practice, we are given initial class probabilities of test samples predicted based on their features (i.e., $P(y_n|\mathbf{x}_n)$) from an existing classifier (without knowing about details in their feature values or training data). Such initial class probabilities can be erroneous due to feature noise and obstacles. In this case, I still hope to incorporate the similar structural dependency between class labels like before but cannot directly apply the HMCT model. Thus, I extend the HMCT model into a post-processing version called **HMCT-PP**. The input of HMCT-PP include initial class probabilities predicted by an existing classifier based on sample features as well as the dependency tree structure between samples, the output is refined class probability after incorporating the structural dependency between classes in the tree. Since HMCT-PP directly models sample classes conditioned on features, the model becomes discriminative. I use the uncollapsed version of contour tree in HMCT-PP because the initial class probability is provided for each individual sample pixel.

Model Structure

An example of the new model architecture (HMCT-PP) is shown in Figure 5.5. In the figure, white nodes represent hidden classes of pixels and gray nodes represent the corresponding feature values. I can see that the same flow dependency structure exists between hidden class nodes (except that the nodes are uncollapsed), but arrow directions between feature nodes and hidden class nodes are reversed to make the model discriminative. In other words, the hidden class layer is now conditioned on the feature layer. The arrows from a feature node to its corresponding hidden class note indicates the class probability of a sample (pixel) based on relevant feature values. Such a design of the new model structure is to allow the model to input initial class probabilities based on feature values

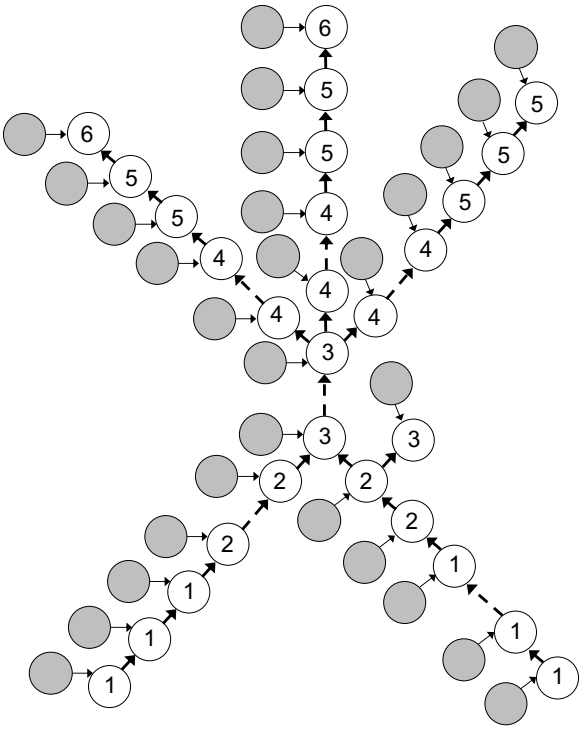


Figure 5.5: An illustrative example of the discriminative version of HMCT

($P(y_n|\mathbf{x}_n)$). Note that in this example, the hidden class node y_n of a sample is conditioned on only its own features $\mathbf{x}_n$ for simplicity. The idea can be extended to allow initial input class probabilities of samples based on spatial contextual features such as those from the U-Net model. The arrows between class nodes in the hidden class layer enforce structural dependency between sample classes. The reason why I do not use *collapsed* contour tree in the hidden class layer is that every individual sample may have a unique initial class probability based on relevant feature values in the model input. Even though two samples are on the same contour segment (i.e., their class nodes could have been collapsed in a collapsed contour tree), their initial class probabilities may differ. Using the uncollapsed contour tree in the hidden class layer of HMCT-PP provides flexibility to allow such input cases. It is worth noting that though the model has feature nodes of all samples in the overall structure, the model may not need the actual feature values in the input for class inference, since the initial class probability $P(y_n|\mathbf{x}_n)$ implicitly contains feature information.

Probabilistic Formulation

I now introduce the probabilistic formulation of the revised discriminative model. Based on the conditional independence assumption captured by the model tree structure in Figure 5.5, the overall conditional probability of all sample classes given all sample features $P(\mathbf{Y}|\mathbf{X})$ can be decomposed into local factors as shown in Equation 5.21, where $P(y_n|y_{\mathcal{P}_n}, \mathbf{X})$ represents the conditional class probability of a sample given its relevant features and parents' classes. For a sample without parents (i.e., leaf nodes in the contour tree), $y_{\mathcal{P}_n} = \emptyset$. If the input initial class probabilities of samples are produced by an i.i.d. classifier (i.e., the initial class probability of a sample is based on their its own feature), then we can assume $P(y_n|y_{\mathcal{P}_n}, \mathbf{X}) = P(y_n|y_{\mathcal{P}_n}, \mathbf{x}_n)$. Otherwise, if the input initial class probabilities of samples are produced by a spatial contextual classifier, we can assume that $P(y_n|y_{\mathcal{P}_n}, \mathbf{X}) = P(y_n|y_{\mathcal{P}_n}, \mathbf{x}_{S(n)})$, where $\mathbf{x}_{S(n)}$ is the spatial contextual features for sample n. For example, for the U-Net model, the spatial context for a pixel is all the pixels in the input image chunk. However, I still need to estimate the local factors, which I will discuss next.

$$P(\mathbf{Y}|\mathbf{X}) = \prod_n P(y_n|y_{\mathcal{P}_n}, \mathbf{X}) \tag{5.21}$$

I first discuss how to estimate the local factors in the case that the input class probabilities of pixels are provided by an i.i.d. classifier, i.e., the input class probability of each pixel is based on its own feature ($P(y_n|\mathbf{X}) = P(y_n|\mathbf{x}_n)$). Many i.i.d. classifiers such as decision tree, random forest, and gradient boosted model fall into this category. This also corresponds to the example of model structure in Figure 5.5. The corresponding estimation of local factors in Equation 5.21 is shown in Equation 5.22. The deductions in the second row and the third row are based on Bayes' theorem and the product rule. In the third row, the local factor $\prod_n P(y_n|y_{\mathcal{P}_n}, \mathbf{x}_n)$ is expressed by $P(y_n|\mathbf{x}_n)$, $P(y_{\mathcal{P}_n}|y_n)$ and $P(y_{\mathcal{P}_n}|\mathbf{x}_n)$. In the fourth row, I make an approximation that $P(y_{\mathcal{P}_n}|\mathbf{x}_n) = P(y_{\mathcal{P}_n}$, i.e., the classes of parent nodes $y_{\mathcal{P}_n}$ are independent from the feature of the current node $\mathbf{x}_n$. This ap-

proximation is reasonable based on the model structure in Figure 5.5. In the fifth row, I use Bayes' theorem again to express $P(y_{\mathcal{P}_n}|y_n)$ by $P(y_n|y_{\mathcal{P}_n})$ and $P(y_n)$. Finally, I cancel out the term $P(y_{\mathcal{P}_n})$ in both numerator and denominator and get the expression in the last row. In this expression, the local factor is expressed by the class probability of a sample conditioned on its feature $P(y_n|\mathbf{x}_n)$, class transitional probability from physical constraints $P(y_n|y_{\mathcal{P}_n})$ and the marginal class probability of a sample $P(y_n)$. Among these factors, $P(y_n|\mathbf{x}_n)$ is given as inputs from an existing classifier, e.g., decision tree, random forest or gradient boosted tree. $P(y_n|y_{\mathcal{P}_n})$ can be computed by a hyper-parameter ρ as in the original HMCT model. Based on the physical constraint, we can set ρ close to 1 (e.g., $\rho = 0.99999$). (Note that for leaf nodes without parents, the factor $P(y_n|y_{\mathcal{P}_n})$ in the numerator cancels out with factor $P(y_n)$ in the denominator.) For marginal class probability $P(y_n)$, it is hard to estimate for all individual samples. We can make a further approximation that $P(y_n)$ is specified by a fixed parameter for all samples, i.e., $P(y_n = 1) = \pi$, where π can be set as 0.5. In the final expression, the term $P(y_n|\mathbf{x}_n)$ incorporates the local class probability based on a sample's feature itself, and the term $\frac{P(y_n|y_{\mathcal{P}_n})}{P(y_n)}$ incorporates the effect of adding structural dependency on a node's class probability. In this way, the class label of one node is determined not only by the local feature information itself but also the class labels at other locations based on the physical constrains.

$$
\begin{aligned}
&\prod_n P(y_n|y_{\mathcal{P}_n}, \mathbf{x}_n) \\
&= \prod_n \frac{P(y_n, y_{\mathcal{P}_n}|\mathbf{x}_n)}{P(y_{\mathcal{P}_n}|\mathbf{x}_n)} \\
&= \prod_n \frac{P(y_n|\mathbf{x}_n)P(y_{\mathcal{P}_n}|y_n)}{P(y_{\mathcal{P}_n}|\mathbf{x}_n)} \\
&\approx \prod_n \frac{P(y_n|\mathbf{x}_n)P(y_{\mathcal{P}_n}|y_n)}{P(y_{\mathcal{P}_n})} \\
&= \prod_n \frac{P(y_n|\mathbf{x}_n)P(y_{\mathcal{P}_n})P(y_n|y_{\mathcal{P}_n})}{P(y_{\mathcal{P}_n})P(y_n)} \\
&= \prod_n \frac{P(y_n|\mathbf{x}_n)P(y_n|y_{\mathcal{P}_n})}{P(y_n)}
\end{aligned}
\tag{5.22}
$$

I now discuss the case that the input class probabilities of pixels are provided by a spatial contextual classifier such as the deep learning model U-Net [77]. In this case, $P(y_n|\mathbf{X}) = P(y_n|\mathbf{x}_{S(n)})$, where $S(n)$ is the set of contextual pixels being used in predicting the class $P(y_n)$. In general, we can follow the same deduction as in the case above by replacing $\mathbf{x}_n$ by $\mathbf{x}_{S(n)}$. The only difference is that the approximation in the fourth row is less accurate in theory now since the parents' classes may be dependent on the current sample's feature in a spatial contextual classifier. But the final expression is still very intuitive. The term $P(y_n|\mathbf{x}_{S(n)})$ reflects the initial class probability. The term $\frac{P(y_n|y_{\mathcal{P}_n})}{P(y_n)}$ reflects the effect of adding structural dependency on a node's class probability.

Class Inference

The class inference can be done based on the probabilistic formulation of the joint conditional distribution $P(\mathbf{Y}|\mathbf{X})$ as expressed in Equation 5.22. The problem is non-trivial due to the combinatorics of all possible classes of tree nodes. We use the same message propagation method called *max-sum* [74] as in HMCT. The main idea is to use dynamic programming to only memorize the current maximum class assignment along a tree traversal order. The only difference from HMCT is that the messages are defined differently due to the change of the terms in the local factors. For class inference, I use fixed hyper-parameters without parameter learning since the model is post-processing on the outputs of an existing classifier. As discussed in the probabilistic formulation section, there are only two hyper-parameters in the revised model HMCT-PP: class transitional probability ρ and approximated marginal class probability π. For the hyper-parameter ρ, we can easily set a value that is close to 1 based on the physical constraint. The hyper-parameter π ($P(y_n = 1) = \pi$) is less straightforward. Since the term appears in the denominator of the term $\frac{P(y_n|y_{\mathcal{P}_n})}{P(y_n)}$, a lower value of π enhances the chance of nodes being inferred as class 1 (flood). In practice, the value of π can be selected based on a rough estimation of the flood extent over the area (π should be small if the flood class is largely under-estimated

in the original classifier). For example, in the flood mapping application as tested in this chapter, the flood class is often under-estimated by many classifiers because pixels of tree canopies overlaying flood water are often mistakenly classified into the dry (non-flood) class. Thus, we should use a generally smaller π (e.g., 0.1). In the case of lacking such prior knowledge, we can choose to set $\pi = 0.5$ (no preference for either class). I test the sensitivity of the class inference algorithm to the two hyper-parameters in the evaluation.

5.4 Evaluation

In this section, I compared my proposed method with baseline methods in classification performance on two real world datasets. We also evaluated the computational scalability of my method, particularly on the effect of tree node collapsing. Experiments were conducted on Intel(R) Xeon(R) CPU E5-2687w v4 @ 3.00GHz, 64GB main memory, and Windows 10. Unless specified otherwise, we used default parameters in open source tools for baseline methods. Candidate classification methods include:

- **Non-spatial classifiers with raw features**: I tested decision tree (**DT**), random forest (**RF**), maximum likelihood classifier (**MLC**), and gradient boosted tree (**GBM**) in R packages on **raw** features (red, green, blue spectral bands).
- **Non-spatial classifiers with additional potential field feature (elev.)**: I tested **DT**, **RF** and **MLC** here.
- **Non-spatial classifier with post-processing label propagation (LP):** I tested **DT**, **RF** and **MLC** based on label propagation with 4-neighborhood [128].
- **Markov random field (MRF):** I used open source implementation [95] based on the graph cut method [93].
- **Deep learning:** I used U-Net [77] implemented in Python and Tensorflow (source codes [124]). I used a batch size of 16, a dropout rate of 0.2, and a learning rate of

0.0001. I trained the model in 50 epochs. The model uses double convolution layers with batch-normalization. The downsample path contains five pooling layers. I used a loss function based on the dice coefficient and the Adam optimizer.

- **Hidden Markov Tree (HMT)**: I used HMT [111] in C++.
- **Hidden Markov Contour Tree (HMCT):** This is my proposed method [42]. Both collapsed (**HMCT-C**) and uncollapsed (**HMCT-UC**) versions were implemented in C++.
- **Hidden Markov Contour Tree - Post Processing (HMCT-PP):** This is my TKDE extension on discriminative HMCT model as post-processing implemented in C++ [81].

Dataset description: I used two flood mapping datasets from the cities of Greensville and Grimesland in North Carolina during Hurricane Mathew in 2016. Explanatory features were red, green, blue bands in aerial imagery from NOAA National Geodetic Survey [65]. The potential field was digital elevation map from the University of North Carolina Libraries [67]. All data were resampled into 2 meter by 2 meter resolution. The number of training and test samples were 5000 per class (flood, dry) in both datasets. The test region size was 1856 by 3149 in Greensville and 2757 by 3853 in Grimesland. Training samples in Greensville were drawn beyond but close to the test region. Training samples in Grimesland dataset were drawn far away from the test region to evaluate model generalizability. Note that for the deep learning method U-Net, I had to provide extra training set in form of contiguous segmented flood imagery. I used 240 image blocks (each of a size of 224 by 224) for training, and 60 blocks for validation.

5.4.1 Classification Performance Comparison

Results on the Greensville dataset were summarized in Table 5.2. Decision tree, random forest, gradient boosted tree, and maximum likelihood classifier achieved overall F-scores between 0.69 and 0.72 on raw features. Adding the elevation feature improved the overall

Table 5.2: Classification on real dataset in Greenville, NC

Classifiers	Class	Precision	Recall	F	Avg. F
DT+Raw	Dry	0.65	0.87	0.74	0.69
	Flood	0.80	0.53	0.64	
RF+Raw	Dry	0.64	0.97	0.77	0.69
	Flood	0.94	0.45	0.61	
GBM+Raw	Dry	0.72	0.73	0.72	0.72
	Flood	0.73	0.72	0.72	
MLC+Raw	Dry	0.72	0.88	0.79	0.77
	Flood	0.85	0.66	0.74	
DT+elev.	Dry	0.76	1.00	0.87	0.84
	Flood	1.00	0.70	0.82	
RF+elev.	Dry	0.74	1.00	0.85	0.81
	Flood	1.00	0.64	0.78	
MLC+elev.	Dry	0.70	0.98	0.82	0.78
	Flood	0.97	0.59	0.73	
DT+LP	Dry	0.64	0.96	0.77	0.69
	Flood	0.93	0.46	0.61	
RF+LP	Dry	0.64	1.00	0.78	0.69
	Flood	1.00	0.43	0.60	
MLC+LP	Dry	0.72	0.96	0.82	0.79
	Flood	0.94	0.63	0.75	
MRF	Dry	0.71	0.98	0.82	0.78
	Flood	0.96	0.60	0.74	
U-Net	Dry	0.87	0.79	0.83	0.84
	Flood	0.81	0.90	0.85	
HMT	Dry	0.60	0.99	0.75	0.63
	Flood	0.98	0.34	0.50	
HMCT-UC	Dry	0.96	0.95	0.96	0.96
	Flood	0.95	0.96	0.96	
HMCT-C	Dry	0.97	0.96	0.96	0.96
	Flood	0.96	0.97	0.96	

F-score dramatically. The improvement was due to the relatively lower elevations among *flood* class pixels. However, the recall of the flood class was still mostly below 0.7 even after adding the elevation feature. The reason is that the right elevation threshold in models differ between training samples and test samples. Post-processing based on label propagation (LP) slightly improved performance of non-spatial classifiers (MLC) due to the removal of salt-and-pepper noise errors. Similar results were found in Markov random field. The deep learning method (U-Net) outperformed non-spatial classifiers (with an F-score of 0.84) due to its capability to learn complex textures from aerial images. HMT performed poorly with very low precision for the dry class and low recall for the flood class, meaning that it mis-classified a significant amount of flood class samples into the dry class. Further investigation showed that many large scale obstacles (tree canopies with dry class features) in the flood area biased the learning and inference of HMT model. In contrast, HMCT models performed significantly better. The reason was that HMCT model flow dependency of locations on two sides of the river in separate tree branches (conditionally independent from each other), mitigating the impact of feature obstacles on model learning and inference.

Results on the Grimesland dataset were summarized in Table 5.3. In this dataset, training samples were drawn far away from the test region to evaluate model generalizability. We can see that most non-spatial classifiers performed poorly except for GBM and MLC (overall F-score 0.73 and 0.76). Adding elevation features improved DT and RF but degraded MLC. This is because training area was overall higher than the entire test region. Thus, elevation thresholds learned from training area were not generalizable to test area. Post-processing also had mixed effect on non-spatial classifiers, because it sometimes mistakenly smoothed out corrected classified samples. U-Net outperformed the other traditional non-spatial models due to its capability of learning complex image textures. HMT again performed poorly due to the same reason as in Greensville dataset. HMCT outperformed others with an overall F-score of 0.94. HMCT showed better generalizability be-

Table 5.3: Classification on real data in Grimesland, NC

Classifiers	Class	Precision	Recall	F	Avg. F
DT+Raw	Dry	0.53	0.92	0.67	0.47
	Flood	0.69	0.17	0.27	
RF+Raw	Dry	0.54	0.98	0.70	0.50
	Flood	0.90	0.18	0.30	
GBM+Raw	Dry	0.69	0.84	0.76	0.73
	Flood	0.79	0.62	0.70	
MLC+Raw	Dry	0.70	0.91	0.79	0.76
	Flood	0.87	0.61	0.72	
DT+elev.	Dry	0.79	0.55	0.65	0.70
	Flood	0.66	0.85	0.72	
RF+elev.	Dry	1.00	0.27	0.43	0.58
	Flood	0.58	1.00	0.73	
MLC+elev.	Dry	0.54	0.88	0.67	0.52
	Flood	0.69	0.26	0.38	
DT+LP	Dry	0.52	0.98	0.67	0.43
	Flood	0.83	0.09	0.17	
RF+LP	Dry	0.53	1.00	0.69	0.45
	Flood	0.96	0.11	0.21	
MLC+LP	Dry	0.71	0.96	0.81	0.77
	Flood	0.93	0.60	0.73	
MRF	Dry	0.70	0.97	0.81	0.76
	Flood	0.96	0.57	0.72	
U-Net	Dry	0.81	0.90	0.85	0.84
	Flood	0.89	0.77	0.83	
HMT	Dry	0.57	0.96	0.72	0.56
	Flood	0.89	0.27	0.41	
HMCT-UC	Dry	0.97	0.92	0.95	0.95
	Flood	0.93	0.97	0.95	
HMCT-C	Dry	0.99	0.94	0.96	0.96
	Flood	0.94	0.99	0.97	

cause its flow dependency was based on relative elevation values in test region. In HMCT, training samples were only used for providing a reasonably well initial model parameters $\{\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c | c = 1, 2\}$.

Sensitivity of HMCT to initial parameters: I conducted sensitivity of the HMCT (with node collapsing) model to different initial parameter values on prior class probability π and class transitional probability ρ (the parameters of $\{\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c | c = 1, 2\}$ were initialized based on maximum likelihood estimation on the training set). Due to space limit, I only showed results on the Greensville dataset. First, I fixed initial $\rho = 0.99$ and varied initial π from 0.1 to 0.9. Results of converged values of ρ together with their corresponding final F-scores were shown in Figure 5.6(a-b). It can be seen that the HMCT model was quite stable with different initial π values. Similarly, I fixed initial $\pi = 0.5$, and varied initial ρ from 0.2, 0.3, to 0.99. Results in Figure 5.6(c-d) showed the same trend. In practice, we can select an initial π value around 0.5 and a relatively high initial ρ value such as 0.9 (because *flood* pixels' neighbor is more likely to be *flood* due to spatial autocorrelation).

Parameter iterations and convergence in HMCT: Here I fixed the initial $\pi = 0.5$ and initial $\rho = 0.99$, and measured the parameter iterations and convergence of HMCT (with node collapsing) on the Greensville dataset. The convergence threshold was set 0.001%. The values of π and ρ at each iteration were summarized in Figure 5.7 (I omitted $\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c$ due to the large number of variables. The parameters converged after 14 iterations. The converged value of ρ was close to 1 as expected.

5.4.2 Computational Performance Comparison

I evaluated the computational performance of the proposed HMCT model. I particularly compared the time costs of HMCT-UC (without node collapsing) and HMCT-C to understand the effect of node collapsing and pre-aggregation. The time cost of HMT should be very close to HMCT-UC since they both have the same number of tree nodes and same set of node operations in learning and inference. Thus, I did not include the cost of HMT to avoid redundancy. For HMCT-C, I further controlled the maximum in-degree (the number

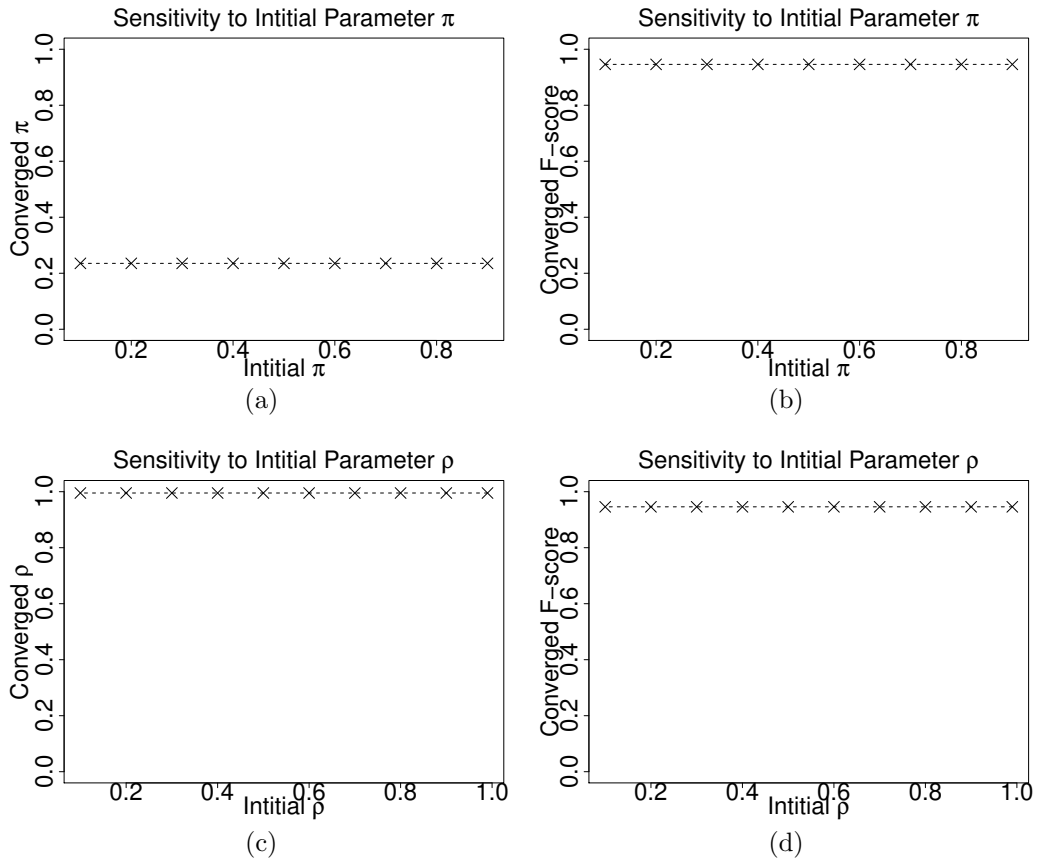


Figure 5.6: Sensitivity of HMCT to initial parameters π and ρ

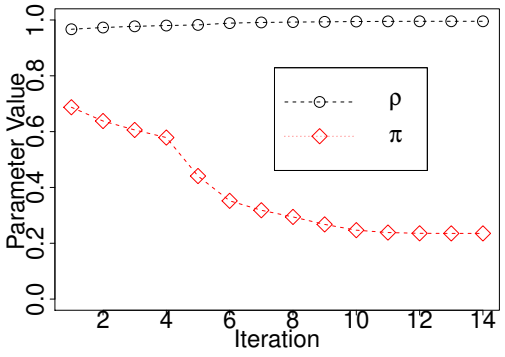


Figure 5.7: Parameter iterations and convergence in HMCT

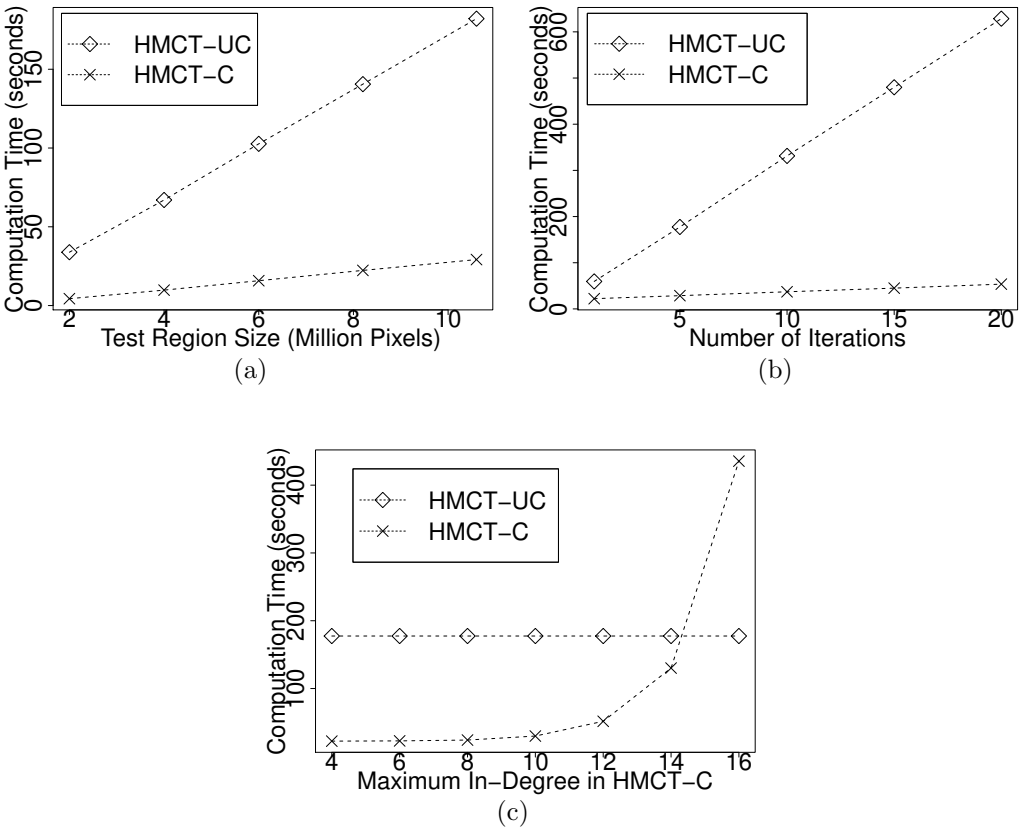


Figure 5.8: Computational performance of HMCT-C versus HMCT-UC on varying:(a) Test region sizes (b) Number of iterations (c) Maximum node in-degree

of parents) for each tree node. This was done simply by stopping the collapsing operations on a node when its in-degree reached the maximum. I compared the two methods on different input data sizes, different number of iterations in parameter learning, and different settings of maximal in-degree per node (in HMCT-C only).

Results were summarized in Figure 5.8. I fixed input parameters $\rho = 0.99$ and $\pi = 0.3$. First, I chose the number of iterations as 5 and the maximum allowed in-degree in HMCT-C as 5. I varied the size of the test region from around 2 million pixels to around 10.6 million pixels. Results in Figure 5.8(a) showed that the time costs of both HMCT-UC and HMCT-C increased almost linearly with the test region size. The rate of increase in HMCT-C was smaller. This was consistent with the $O(N_T)$ term in my cost model. Note that N_T is much smaller in HMCT-C due to node collapsing. HMCT-C ran very fast, cost-

Table 5.4: Time costs of individual components (seconds)

	HMCT-UC	HMCT-C	HMCT-C
Max in-degree	N/A	4	16
Tree construction	13.47	13.47	13.47
Node Collapsing	N/A	6.64	6.64
Pre-aggregation	N/A	0.28	0.28
Parameter learning	152.51	1.80	394.01
Class inference	10.71	0.21	21.09
Total time	176.69	22.41	435.49

ing less than 30 seconds on over 10 million pixels. In the second experiment, I chose a test region with around 10.6 million pixels, and set the maximum allowed in-degree in HMCT-C as 10. I varied the number of iterations in parameter learning from 1 to 20. Results in Figure 5.8(b) showed that the costs of both methods increased almost linearly with the number of iterations. This is consistent with my cost model. In the third experiment, I chose a test region with around 10.6 million pixels, and set the number of iterations as 5. I varied the maximum allowed in-degree in HMCT-C from 4 to 16 (I did not choose values smaller than 4 because the maximum in-degree of uncollapsed tree nodes is already 4 due to the four-neighborhood I used). Results in Figure 5.8(c) showed that as the maximum allowed in-degree increases, the cost of HMCT-C grew exponentially, exceeding the cost of HMCT-UC when the maximum in-degree was 16. This is consistent with the 2^d term in my cost model.

Table 5.4 showed detailed time costs for individual components in HMCT algorithms. I chose an input data with a test region of 10.6 million pixels, and the number of iterations as 5, and a max in-degrees of 4 and 16 for HMCT respectively. From the table, we can see that when the max in-degree is 4, learning and inference in HMCT-C were significantly faster than HMCT-UC due to node collapsing. However, when the max in-degree is 16, the costs of HMCT-C was almost twice as that of HMCT-UC, due to the exponential cost in terms of in-degree (2^d). In practice, we can set the maximum allowed in-degree as 4.

I also measured the computational cost of U-Net model on one NVIDIA Tesla P100 GPU.

The training set contains 240 image chunks with an equal size of 224 by 224 pixels. The validation set contain 60 image chunks. The total time of model training is 208 seconds for 50 epochs. The time costs of predicting classes on test images are 2.44 seconds for the Grimesland dataset (204 image chunks) and 1.04 seconds for the Greensville dataset (112 image chunks).

Table 5.5: HMCT as post-processing on Greenville dataset

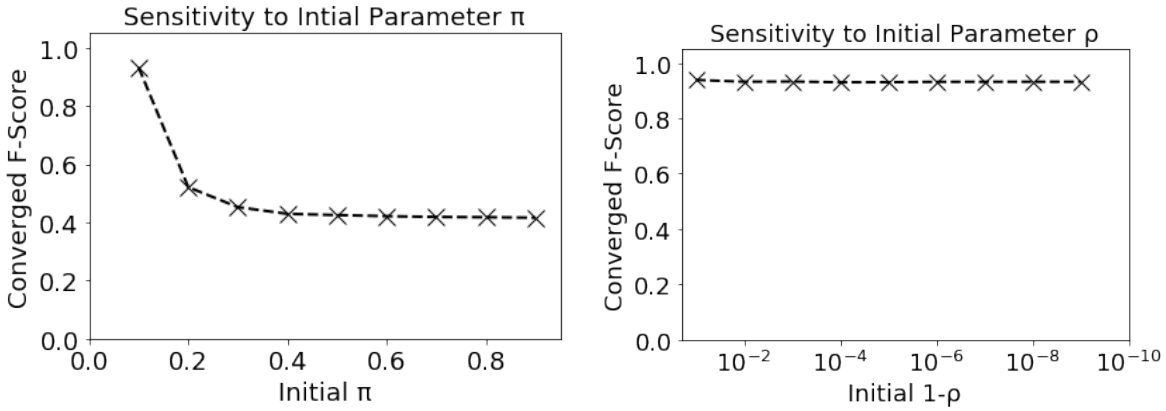
Classifiers	Class	Precision	Recall	F	Avg. F
HMCT-PP+DT	Dry	0.97	0.90	0.94	0.94
	Flood	0.91	0.98	0.94	
HMCT-PP+MLC	Dry	0.95	0.91	0.93	0.93
	Flood	0.92	0.95	0.94	
HMCT-PP+RF	Dry	0.85	0.99	0.91	0.91
	Flood	0.99	0.82	0.90	
HMCT-PP+GBM	Dry	0.90	0.96	0.93	0.93
	Flood	0.96	0.89	0.93	
HMCT-PP+U-Net	Dry	0.98	0.98	0.99	0.99
	Flood	0.99	0.98	0.99	

Table 5.6: HMCT as post-processing on Grimesland dataset

Classifiers	Class	Precision	Recall	F	Avg. F
HMCT-PP+DT	Dry	0.86	0.98	0.92	0.91
	Flood	0.98	0.84	0.90	
HMCT-PP+MLC	Dry	0.88	0.99	0.93	0.93
	Flood	0.97	0.87	0.92	
HMCT-PP+RF	Dry	0.88	0.99	0.93	0.93
	Flood	0.99	0.86	0.92	
HMCT-PP+GBM	Dry	0.95	0.96	0.96	0.96
	Flood	0.95	0.95	0.96	
HMCT-PP+U-Net	Dry	0.95	0.98	0.96	0.96
	Flood	0.98	0.94	0.96	

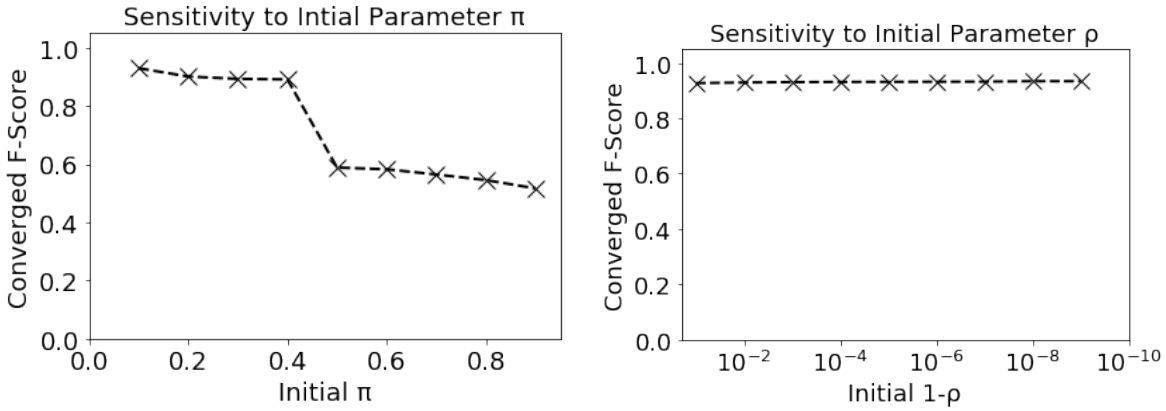
5.4.3 Evaluation of HCMT as Post Processing

Classification performance: I now evaluate the HMCT as a post-processor on top of class probability predictions from five existing methods: decision tree, random forest, maximum likelihood classifier, gradient boosted tree and U-Net. Note that I did not run HMCT-PP on top of HMCT itself because HMCT class inference is based on the maximum likeli-



(a) Fix $\rho = 1 - 10^{-9}$ (b) Fix $\pi = 0.1$

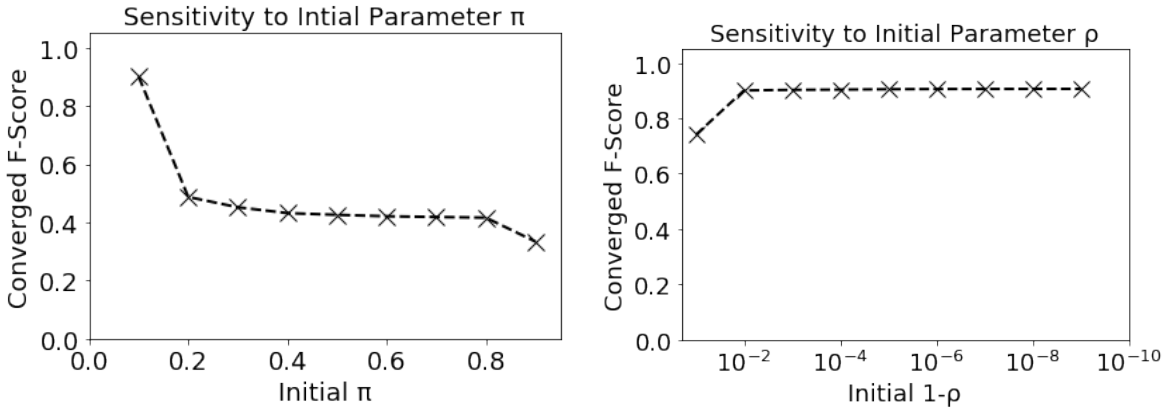
Figure 5.9: Sensitivity of HMCT-PP+DT to π and ρ



(a) Fix $\rho = 1 - 10^{-9}$ (b) Fix $\pi = 0.1$

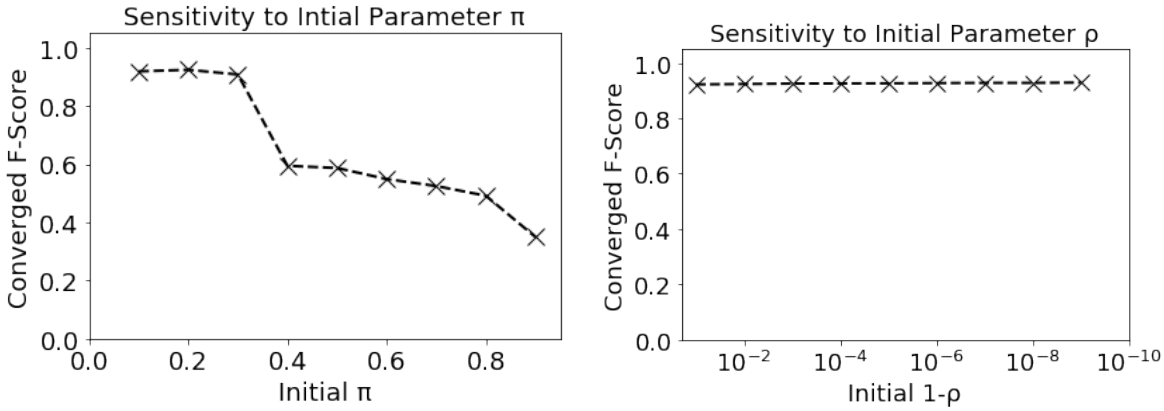
Figure 5.10: Sensitivity of HMCT-PP+MLC to π and ρ

hood method instead of drawing a threshold (e.g., 0.5) on class probabilities of individual pixels. In other words, the inference algorithm of HMCT only tells which class assignment of all pixels has the highest overall likelihood without providing the class probabilities of individual pixels. Results on the two datasets were summarized in Table 5.5 and Table 5.6 respectively. For initial parameters, I set $\rho = 1 - 10^{-9}$ and chose π based on the situation. If the pre-classified flood extent is largely under-estimated, then I set π to a low value (e.g., $\pi = 0.1$ for DT, MLC, GBM) to enlarge the extent. If the pre-classified flood extent is moderate, then I set a moderate value of π (e.g., $\pi = 0.4$ for U-Net). From the two tables, I can find that HMCT-PP significantly enhanced the performance of base



(a) Fix $\rho = 1 - 10^{-9}$ (b) Fix $\pi = 0.1$

Figure 5.11: Sensitivity of HMCT-PP+RF to π and ρ

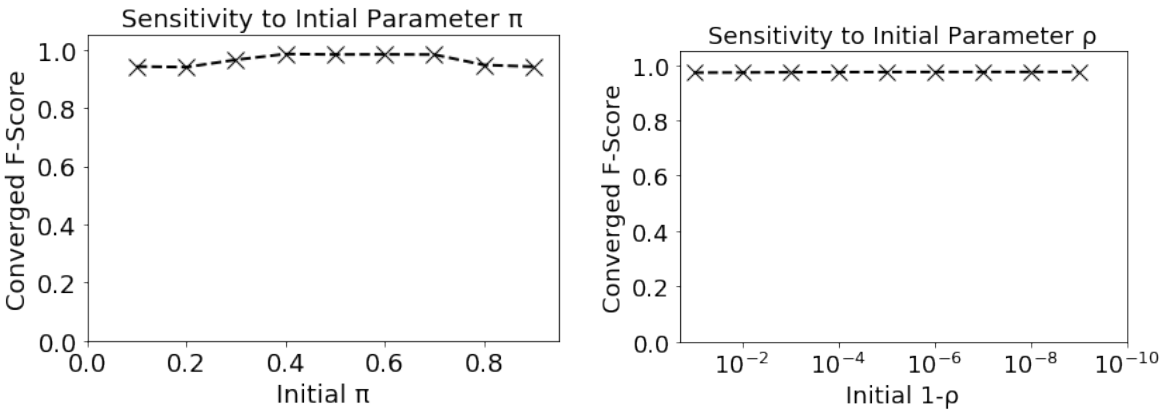


(a) Fix $\rho = 1 - 10^{-9}$ (b) Fix $\pi = 0.1$

Figure 5.12: Sensitivity of HMCT-PP+GBM to π and ρ

classifiers, improving F-cores from below 0.8 to above 0.9. Among all the base methods, HMCT-PP with U-Net achieved the best performance with an F-score of 0.99 and 0.96 in two datasets. The reason for the significant improvement is that HMCT can utilize the high-confident predictions of flood areas from base methods and leverage the class dependency structure from topography constraint to fill the gap of missing flood extent or fine tune the correct flood extent boundary.

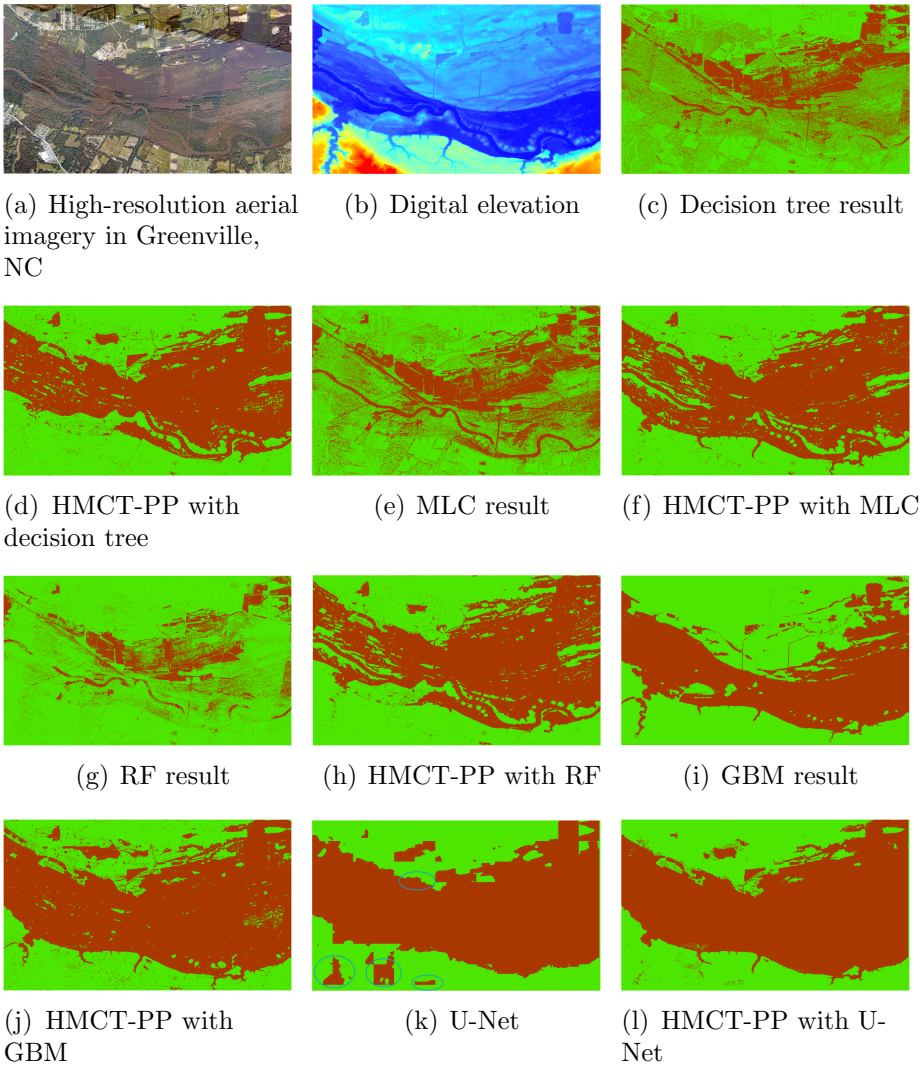
One important observation is that HMCT-PP with U-Net outperforms HMCT-C itself. The reason is that HMCT assumes that explanatory feature values of samples in each class follows an i.i.d. Gaussian distribution. This assumption can be over simplistic for



(a) Fix $\rho = 1 - 10^{-9}$

(b) Fixed $\pi = 0.4$

Figure 5.13: Sensitivity of HMCT-PP+U-Net to π and ρ



(a) High-resolution aerial imagery in Greenville, NC

(b) Digital elevation

(c) Decision tree result

(d) HMCT-PP with decision tree

(e) MLC result

(f) HMCT-PP with MLC

(g) RF result

(h) HMCT-PP with RF

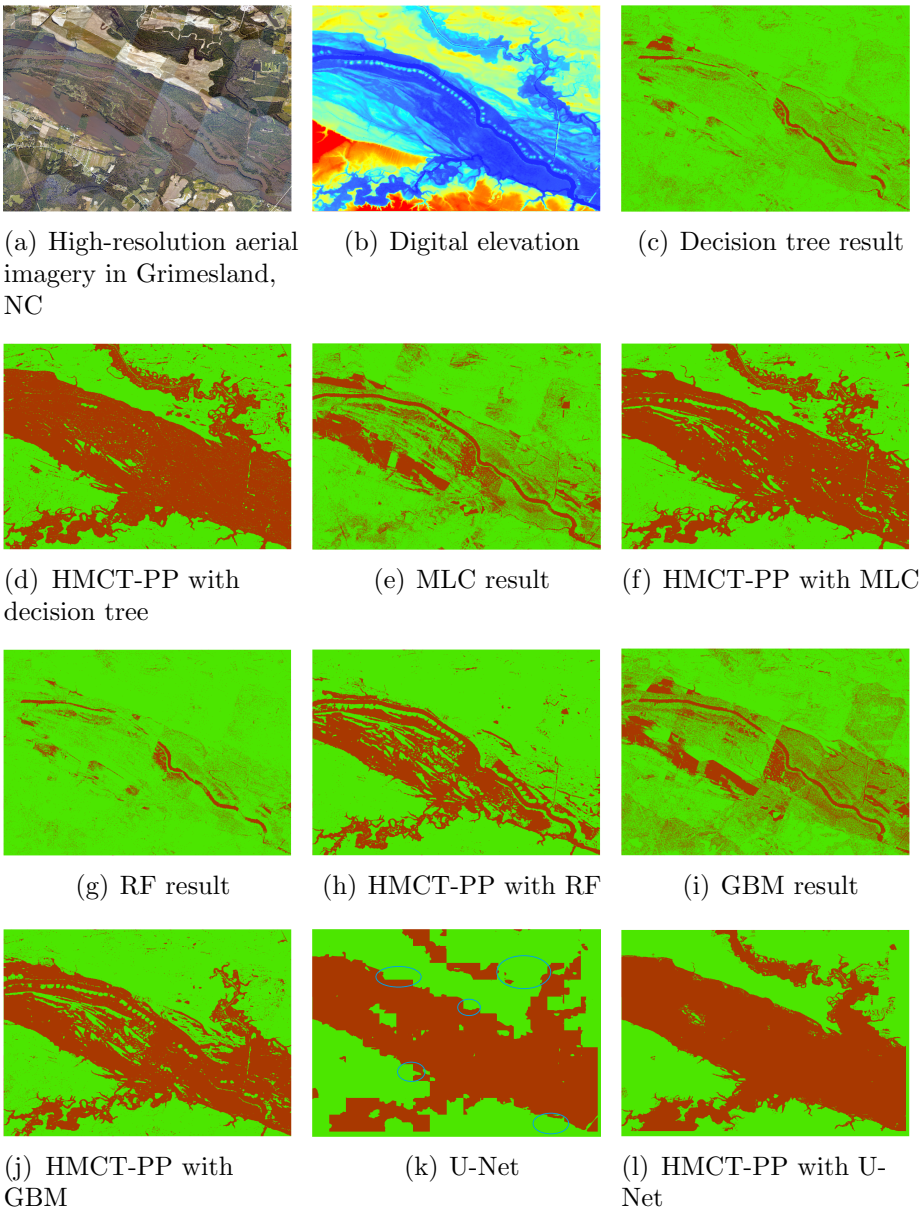
(i) GBM result

(j) HMCT-PP with GBM

(k) U-Net

(l) HMCT-PP with U-Net

Figure 5.14: Results on Greenville flood mapping (flood in brown, dry in green)



(a) High-resolution aerial imagery in Grimesland, NC

(b) Digital elevation

(c) Decision tree result

(d) HMCT-PP with decision tree

(e) MLC result

(f) HMCT-PP with MLC

(g) RF result

(h) HMCT-PP with RF

(i) GBM result

(j) HMCT-PP with GBM

(k) U-Net

(l) HMCT-PP with U-Net

Figure 5.15: Results on Grimesland flood mapping (flood in brown, dry in green)

the real-world datasets. The main advantage of HMCT is that it captures spatial structural dependency between different class nodes in the contour tree. In other words, HMCT models $P(y_n|y_{P_n}$ in a sophisticated contour tree structure but makes a simplistic assumption on $P(\mathbf{x}_n|y_n)$ (i.e., i.i.d. Gaussian). In contrast, HMCT-PP with U-Net combines the strength of HMCT in modeling $P(y_n|y_{P_n}$ in a contour tree structure and the strength of U-Net in learning $P(y_n|\mathbf{X})$ with spatial contexts of feature X. Thus, I observed that HMCT-

PP with U-Net has an average F-score of 0.99, higher than the average F-score of 0.96 in HMCT-C.

Sensitivity to initial parameters: I conducted sensitivity analysis of the HMCT-PP on the five base models to different values of input parameters ρ and π. I only showed results on the Grimesland dataset. When evaluate the sensitivity to π, I fixed the parameter $\rho = 1 - 10^{-10}$, and varied the value of π from 0.1 to 0.9. The results on five base methods are shown in the left sub-figures in Figure 5.9, Figure 5.10, Figure 5.11, Figure 5.12, and Figure 5.13 respectively. From the results, we can find that the initial value of π made a significant effect on the final classification performance. For HMCT-PP on DL and RF, only $\pi = 0.1$ is effective. The reason is that the initial classified flood map has very low recall (under-estimation). Thus, we need to set π small enough to compensate for the errors. For HMCT-PP with MLC and GBM, the range of effective π values are wider (0.1 to 0.4 for MLC, and 0.1 to 0.3 for GBM). The reason is likely that the under-estimation in initial flood map is not very significant, and thus we do not necessary set a very small value of π to compensate it. For U-Net model, we can find that the performance is not very sensitivity to π ($\pi = 0.4$ showed the best performance). The reason is that the initial classification in U-Net is mostly confident and correct (F-score is 0.84), and errors are largely on the flood extent boundary. From the analysis, values of π can be chosen based on an initial guess on whether flood is under-estimated or not.

Visualized maps: The above analysis is further confirmed by the visualization of classified flood extent maps from different approaches shown in Figure 5.14 and Figure 5.15 respectively. The left column of sub-figures show the flood maps from base classifiers of DT, MLC, RF, GBM, and U-Net respectively, where brown color represents flood and green color represents dry. From the results, we can see that all methods except U-Net shows a significant under-estimation of flood extent due to the tree canopies in the area. This confirms the low recalls in Table 5.2 and Table 5.3. The post-processed flood maps by HMCT-PP filled a significant portion of the gap. We can also find that the classified

flood maps from U-Net successfully identified the flood areas, but the flood extent boundaries were too coarse. The reason is that U-Net classified the image patch-wise, and very often an entire patch was predicted as flood. The results of HMCT-PP with U-Net showed better flood extent boundaries based on topography.

5.5 Summary

In this chapter, I introduce hidden Markov contour tree (HMCT), a spatial structured model that can capture flow dependency on a potential field surface. I discuss efficient algorithms for model parameter learning and class inference. I also introduce an extention to the HMCT model into a post-processor that can enhance the classified results from other models based on the topography constraints. Evaluations on real world data show that my HMCT algorithms are scalable to large data sizes, and can achieve higher classification performance over existing methods for hydrological applications such as flood mapping.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

This thesis investigates novel geospatial data science techniques to address the computational challenges of geospatial data. The study of spatial data has the potential to transform society. Research on geospatial data science is vital because of its central role in critical societal challenges such as climate change, disaster management, disease control, and crime management. Scientists use geospatial data science techniques on spatial data to study global water resources, deforestation, and climate change. However, geospatial data science poses several computational challenges that are not well addressed by traditional data science techniques, such as spatial dependency, large data volume, and incomplete observation. The thesis introduces novel parallel spatial data mining algorithms on GPUs (e.g., GPU colocation mining) to address the computational challenges of a large spatial data volume. Experiment results on the real-world and synthetic datasets showed that my GPU-based colocation mining algorithms achieved a significant performance gain over the sequential algorithms. The thesis also introduces a deep learning framework that captures the spatial dependency (i.e., sequential structure) in the road networks and applied the framework to mapping road safety features from street view imagery. Evaluations on real-world street-view images collected from highways in Alabama show that my approach outperformed several baseline methods in classification performance. Next, the thesis introduces a novel approach based on the physics-aware spatial structural constraint (e.g., water flow direction on geographic terrains) to address the challenge of limited observations with incomplete spatial coverage. The proposed techniques have been applied to the application of flood extent mapping from remote sensing imagery. Results show that

my approach significantly outperformed several baseline methods in classification performance and that the proposed solution is computationally efficient on a large data volume. Finally, the thesis introduces a spatial structure model called a hidden Markov contour tree (HMCT) to capture the spatial flow dependency on a potential field surface. Evaluations on real-world flood mapping datasets show that HMCT outperforms multiple baseline methods in classification performance, and the HMCT can also effectively enhance the results of other baseline methods. Computational experiments also show that HMCT is scalable to large data sizes (e.g., classifying millions of samples in seconds).

I plan to explore several directions in my future work. First, I plan to extend my works on colocation mining. As the number of spatial features increases, we can have an exponential number of candidate patterns. Sometimes, the study's goal may not be to extract all the interesting patterns from the spatial dataset. But simply to identify if some set of candidate patterns are interesting or not. To address this, I plan to explore query-based colocation mining in spatial relational databases. The main idea of query-based colocation mining is to quickly check if a candidate pattern is interesting or not without extracting all the interesting patterns in the dataset. Another extension to my current colocation mining approach would be developing a novel filtering technique with a tighter upper bound than our proposed parallel 4-cell parallel cell aggregate-based filter. The major drawback of my current filtering technique is that it is a counting-based filter. Such filters only work best when the clumpiness of the dataset is very high. In contrast, when the data points are well separated we may need a different filtering technique to provide a tighter upper bound participation index. Finally, I plan to collaborate with researchers in the biological domain to identify novel symbiotic relationships from ecological data. Symbiosis is widely acknowledged as a key driver of evolution. Identifying novel symbiotic relationships can help researchers better understand the evolutionary process.

Next, I would like to explore the extension to our flood mapping framework. During a flooding event, the area under the study can be flooded due to floodwater from several dif-

ferent sources. In my previous works, we assume sample feature distribution in each class to be i.i.d. Gaussian. The actual sample feature distribution in each class can be multi-modal in real-world datasets, violating our assumption on a single-modal Gaussian distribution. To account for this observation, I plan to extend our model to allow for multi-modal sample feature distribution in each class. The multi-modal extension will be more robust than the existing single-modal approach.

I plan to continue my work on interdisciplinary problems that are both societally significant and are of national priority. In particular, I plan to explore the use of spatial-temporal data for understanding human mobility and disease transmission dynamics. Spatial-temporal data are the data that relate to both space and time. Mining spatial-temporal data can help us discover the evolutionary history of human settlements, identify weather patterns, predict natural disasters such as earthquakes and hurricanes, and determine global warming trends [34]. The major challenge of spatial-temporal data is the non-identical distribution of data samples across the entire space and time [34]. Existing data mining techniques assume that the data are independently and identically distributed and fail to handle spatial-temporal data.

Finally, I would also like to research developing spatial prediction models for traffic safety. One of the interesting areas in traffic safety is identifying locations that are susceptible to accidents. Real-time prediction of accident-prone zones can help drivers take extra caution while driving through those areas and eventually help in saving lives. Many factors can help us to predict the accident-prone zones in the road networks. The valuable data for such prediction tasks can come from heterogeneous sources such as weather information, road quality, visibility, traffic flow, and other sensor data from smart vehicles. Utilizing all the data together can potentially improve predictive performance.

REFERENCES

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[2] Witold Andrzejewski and Pawel Boinski. Gpu-accelerated collocation pattern discovery. In *East European Conference on Advances in Databases and Information Systems*, pages 302–315. Springer, 2013.

[3] Witold Andrzejewski and Pawel Boinski. A parallel algorithm for building icpi-trees. In *East European Conference on Advances in Databases and Information Systems*, pages 276–289. Springer, 2014.

[4] Witold Andrzejewski and Pawel Boinski. Parallel gpu-based plane-sweep algorithm for construction of icpi-trees. *Journal of Database Management (JDM)*, 26(3):1–20, 2015.

[5] Luc Anselin. *Spatial econometrics: methods and models*, volume 4. Springer Science & Business Media, 2013.

[6] Vahid Balali, Elizabeth Depwe, and Mani Golparvar-Fard. Multi-class traffic sign detection and classification using google street view images. In *Transportation Research Board 94th Annual Meeting, Transportation Research Board, Washington, DC*, 2015.

[7] Vahid Balali, Armin Ashouri Rad, and Mani Golparvar-Fard. Detection, classification, and mapping of us traffic signs using google street view images for roadway inventory management. *Visualization in Engineering*, 3(1):15, 2015.

[8] Michael R Bambach, RJ Mitchell, and Raphael H Grzebieta. The protective effect of roadside barriers for motorcyclists. *Traffic injury prevention*, 14(7):756–765, 2013.

[9] Sudipto Banerjee, Bradley P Carlin, and Alan E Gelfand. *Hierarchical modeling and analysis for spatial data*. Chapman and Hall/CRC, 2014.

[10] Sajib Barua and Jörg Sander. Sscp: mining statistically significant co-location patterns. In *International Symposium on Spatial and Temporal Databases*, pages 2–20. Springer, 2011.

[11] Sajib Barua and Jörg Sander. Mining statistically significant co-location and segregation patterns. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1185–1199, 2014.

[12] Sajib Barua and Jörg Sander. Statistically significant co-location pattern mining. 2015.

[13] Gustavo EAPA Batista, Maria Carolina Monard, et al. A study of k-nearest neighbour as an imputation method. *HIS*, 87(251-260):48, 2002.

[14] Yoshua Bengio and Francois Gingras. Recurrent neural networks for missing or asynchronous data. In *Advances in neural information processing systems*, pages 395–401, 1996.

[15] Pawel Boinski and Maciej Zakrzewicz. Collocation pattern mining in a limited memory environment using materialized icpi-tree. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 279–290. Springer, 2012.

[16] PA Brivio, R Colombo, M Maggi, and R Tomasoni. Integration of remote sensing data and gis for accurate mapping of flooded areas. *International Journal of Remote Sensing*, 23(3):429–441, 2002.

[17] James E Bryden and Jan S Fortuniewicz. Traffic barrier performance related to vehicle size and type. *Transportation Research Record*, 1065:69–78, 1986.

[18] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.

[19] Mete Celik, James M Kang, and Shashi Shekhar. Zonal co-location pattern discovery with dynamic parameters. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 433–438. IEEE, 2007.

[20] Gal Chechik, Geremy Heitz, Gal Elidan, Pieter Abbeel, and Daphne Koller. Max-margin classification of incomplete data. In *Advances in Neural Information Processing Systems*, pages 233–240, 2007.

[21] City of Seattle, Department of Information Technology, Seattle Police Department. Seattle Police Department 911 Incident Response. https://data.seattle.gov/Public-Safety/Seattle-Police-Department-911-Incident-Response/3k2p-39jp.

[22] Don Cline. Integrated water resources science and services: an integrated and adaptive roadmap for operational implementation. Technical report, National Oceanic and Atmospheric Administration, 2009.

[23] data.gov. Road Segments in Alabama. https://catalog.data.gov/dataset/tiger-line-shapefile-2015-state-alabama-primary-and-secondary-roads-state-based-shapefile.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[25] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[26] Pedro J García-Laencina, José-Luis Sancho-Gómez, and Aníbal R Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, 2010.

[27] Timnit Gebru, Jonathan Krause, Yilun Wang, Duyun Chen, Jia Deng, Erez Lieberman Aiden, and Li Fei-Fei. Using deep learning and google street view to estimate the demographic makeup of neighborhoods across the united states. *Proceedings of the National Academy of Sciences*, 114(50):13108–13113, 2017.

[28] Z Ghahramani and MI Jordan. Learning from incomplete data (tech. rep. no. aim-1509), 1994.

[29] Zoubin Ghahramani and Michael I Jordan. Supervised learning from incomplete data via an em approach. In *Advances in neural information processing systems*, pages 120–127, 1994.

[30] Zoubin Ghahramani and Michael I Jordan. Learning from incomplete data. 1995.

[31] Raphael Grzebieta, Mike Bambach, and Andrew McIntosh. Motorcyclist impacts into roadside barriers: Is the european crash test standard comprehensive enough? *Transportation Research Record*, 2377(1):84–91, 2013.

[32] David Günther, Roberto A Boto, Juila Contreras-Garcia, Jean-Philip Piquemal, and Julien Tierny. Characterizing molecular interactions in chemical systems. *IEEE transactions on visualization and computer graphics*, 20(12):2476–2485, 2014.

[33] Gayle SW Hagler, Wei Tang, Matthew J Freeman, David K Heist, Steven G Perry, and Alan F Vette. Model evaluation of roadside barrier impact on near-road air pollution. *Atmospheric Environment*, 45(15):2522–2530, 2011.

[34] Ali Hamdi, Khaled Shaban, Abdelkarim Erradi, Amr Mohamed, Shakila Khan Rumi, and Flora D Salim. Spatiotemporal data mining: a survey on challenges and open problems. *Artificial Intelligence Review*, pages 1–48, 2021.

[35] Yan Huang, Shashi Shekhar, and Hui Xiong. Discovering colocation patterns from spatial data sets: a general approach. *IEEE Transactions on Knowledge and data engineering*, 16(12):1472–1485, 2004.

[36] Yan Huang, Hui Xiong, Shashi Shekhar, and Jian Pei. Mining confident co-location rules without a support threshold. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 497–501. ACM, 2003.

[37] Hussein H Jama, Raphael H Grzebieta, Rena Friswell, and Andrew S McIntosh. Characteristics of fatal motorcycle crashes into roadside safety barriers in australia and new zealand. *Accident Analysis & Prevention*, 43(3):652–660, 2011.

[38] Kai Jiang, Haixia Chen, and Senmiao Yuan. Classification for incomplete data using classifier ensembles. In *2005 International Conference on Neural Networks and Brain*, volume 1, pages 559–563. IEEE, 2005.

[39] Zhe Jiang. A survey on spatial prediction methods. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[40] Zhe Jiang, Michael Evans, Dev Oliver, and Shashi Shekhar. Identifying k primary corridors from urban bicycle gps trajectories on a road network. *Information Systems*, 57:142–159, 2016.

[41] Zhe Jiang, Yan Li, Shashi Shekhar, Lian Rampi, and Joseph Knight. Spatial ensemble learning for heterogeneous geographic data with class ambiguity: A summary of results. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 23. ACM, 2017.

[42] Zhe Jiang and Arpan Man Sainju. Hidden markov contour tree: A spatial structured model for hydrological applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 804–813. ACM, 2019.

[43] Zhe Jiang, Arpan Man Sainju, Yan Li, Shashi Shekhar, and Joseph Knight. Spatial ensemble learning for heterogeneous geographic data with class ambiguity. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(4):43, 2019.

[44] Zhe Jiang and Shashi Shekhar. *Spatial big data science*. Springer, 2017.

[45] Zhe Jiang, Shashi Shekhar, Pradeep Mohan, Joseph Knight, and Jennifer Corcoran. Learning spatial decision tree for geographical classification: a summary of results. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 390–393. ACM, 2012.

[46] Zhe Jiang, Shashi Shekhar, Xun Zhou, Joseph Knight, and Jennifer Corcoran. Focal-test-based spatial decision tree learning: A summary of results. In *2013 IEEE 13th International Conference on Data Mining*, pages 320–329. IEEE, 2013.

[47] Zhe Jiang, Shashi Shekhar, Xun Zhou, Joseph Knight, and Jennifer Corcoran. Focal-test-based spatial decision tree learning. *IEEE Transactions on Knowledge and Data Engineering*, 27(6):1547–1559, 2014.

[48] Zhe Jiang, Miao Xie, and Arpan Man Sainju. Geographical hidden markov tree. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[49] Hawzheen Karim, Rolf Magnusson, and Mats Wiklund. Assessment of injury rates associated with road barrier collision. *Procedia-social and behavioral sciences*, 48:52–63, 2012.

[50] Anuj Karpatne, Ankush Khandelwal, Shyam Boriah, and Vipin Kumar. Predictive learning in the presence of heterogeneity and limited training data. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 253–261, 2014.

[51] Driver Knowledge. Accident statistics, 2013.

[52] Krzysztof Koperski and Jiawei Han. Discovery of spatial association rules in geographic information databases. In *International Symposium on Spatial Databases*, pages 47–66. Springer, 1995.

[53] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[54] Alexander Kuhn, Wito Engelke, Markus Flatken, Hans-Christian Hege, and Ingrid Hotz. Topology-based analysis for multimodal atmospheric data of volcano eruptions. In *Topological Methods in Data Analysis and Visualization*, pages 35–50. Springer, 2015.

[55] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[56] Xiaojiang Li, Chuanrong Zhang, Weidong Li, Robert Ricard, Qingyan Meng, and Weixing Zhang. Assessing street-level urban greenery using google street view and a modified green view index. *Urban Forestry & Urban Greening*, 14(3):675–685, 2015.

[57] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[58] Bozhong Liu, Ling Chen, Chunyang Liu, Chengqi Zhang, and Weidong Qiu. Rcp mining: Towards the summarization of spatial co-location patterns. In *International Symposium on Spatial and Temporal Databases*, pages 451–469. Springer, 2015.

[59] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007.

[60] Venkatesh Merwade, Francisco Olivera, Mazdak Arabi, and Scott Edleman. Uncertainty in flood inundation mapping: current issues and future directions. *Journal of Hydrologic Engineering*, 13(7):608–620, 2008.

[61] Pradeep Mohan, Shashi Shekhar, James A Shine, James P Rogers, Zhe Jiang, and Nicole Wayant. A neighborhood graph based approach to regional co-location pattern discovery: A summary of results. In *Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 122–132. ACM, 2011.

[62] Yasuhiko Morimoto. Mining frequent neighboring class sets in spatial databases. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 353–358. ACM, 2001.

[63] James R Munkres. *Topology*. Pearson, January 2000.

[64] Aibek Musaev, Zhe Jiang, Steven Jones, Pezhman Sheinidashtegol, and Mirbek Dzhumaliev. Detection of damage and failure events of road infrastructure using social media. In *International Conference on Web Services*, pages 134–148. Springer, 2018.

[65] National Oceanic and Atmospheric Administration. Data and imagery from noaa's national geodetic survey. https://www.ngs.noaa.gov.

[66] National Oceanic and Atmospheric Administration. National Water Model: Improving NOAA's Water Prediction Services. http://water.noaa.gov/documents/wrn-national-water-model.pdf, 2018.

[67] NCSU Libraries. LIDAR Based Elevation Data for North Carolina. https://www.lib.ncsu.edu/gis/elevation, 2018.

[68] Congressional Budget Office. Public spending on transportation and water infrastructure, 1956 to 2017, 2018.

[69] onlinejournalismblog. Crime Map. https://onlinejournalismblog.com/2015/08/24/when-to-use-maps-in-data-visualisation-a-great-big-guide/.

[70] Valerio Pascucci, Xavier Tricoche, Hans Hagen, and Julien Tierny. *Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications*. Springer Science & Business Media, 2010.

[71] Kristiaan Pelckmans, Jos De Brabanter, Johan AK Suykens, and Bart De Moor. Handling missing values in support vector machine classifiers. *Neural Networks*, 18(5-6):684–692, 2005.

[72] J Ross Quinlan. Unknown attribute values in induction. In *Proceedings of the sixth international workshop on Machine learning*, pages 164–168. Elsevier, 1989.

[73] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[74] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[75] Benjamin Romano and Zhe Jiang. Visualizing traffic accident hotspots based on spatial-temporal network kernel density estimation. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 98. ACM, 2017.

[76] O Ronen, JR Rohlicek, and M Ostendorf. Parameter estimation of dependence tree models using the em algorithm. *IEEE Signal Processing Letters*, 2(8):157–159, 1995.

[77] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[78] Carlos Roque and João Lourenço Cardoso. Observations on the relationship between european standards for safety barrier impact severity and the degree of injury sustained. *IATSS research*, 37(1):21–29, 2013.

[79] Donald B Rubin. *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons, 2004.

[80] Arpan Man Sainju, Danial Aghajarian, Zhe Jiang, and Sushil K Prasad. Parallel grid-based colocation mining algorithms on gpus for big spatial event data. *IEEE Transactions on Big Data*, 2018.

[81] Arpan Man Sainju, Wenchong He, and Zhe Jiang. A hidden markov contour tree model for spatial structured prediction. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[82] Arpan Man Sainju, Wenchong He, Zhe Jiang, and Da Yan. Spatial classification with limited observations based on physics-aware structural constraint. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 898–905, 2020.

[83] Arpan Man Sainju and Zhe Jiang. Grid-based colocation mining algorithms on gpu for big spatial event data: A summary of results. In *International Symposium on Spatial and Temporal Databases*, pages 263–280. Springer, 2017.

[84] Arpan Man Sainju and Zhe Jiang. Mapping road safety features from streetview imagery: A deep learning approach. In *ACM Transactions on Data Science*. ACM, 2019.

[85] Joseph L Schafer. *Analysis of incomplete multivariate data*. Chapman and Hall/CRC, 1997.

[86] Jennifer D Schmidt, Ronald K Faller, Dean L Sicking, John D Reid, Karla A Lechtenberg, Robert W Bielenberg, Scott K Rosenbaugh, Jim C Holloway, et al. Development of a new energy-absorbing roadside/median barrier system with restorable elastomer cartridges. Technical report, Nebraska. Dept. of Roads, 2013.

[87] Nico Schulte, Michelle Snyder, Vlad Isakov, David Heist, and Akula Venkatram. Effects of solid barriers on dispersion of roadway emissions. *Atmospheric environment*, 97:286–295, 2014.

[88] Shashi Shekhar, Zhe Jiang, Reem Ali, Emre Eftelioglu, Xun Tang, Venkata Gunturi, and Xun Zhou. Spatiotemporal data mining: a computational perspective. *ISPRS International Journal of Geo-Information*, 4(4):2306–2338, 2015.

[89] Alexander J Smola, SVN Vishwanathan, and Thomas Hofmann. Kernel methods for missing variables. In *AISTATS*. Citeseer, 2005.

[90] Jonathan Stoeckel and Glenn Fung. SVM feature selection for classification of SPECT images of alzheimer's disease using spatial information. In *ICDM*, pages 410–417, 2005.

[91] Karthik Subbian and Arindam Banerjee. Climate multi-model regression using spatial smoothing. In *SIAM International Conference on Data Mining (SDM)*, pages 324–332, 2013.

[92] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[93] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields. In *European conference on computer vision*, pages 16–29. Springer, 2006.

[94] templategallary. Countries Boundaries in Africa. http://www.vinotique.com/post_coloring-africa-map-outline_69794/.

[95] The Middlebury Computer Vision Pages. C++ Source Code of MRF. http://vision.middlebury.edu/MRF/code/, 2018.

[96] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970.

[97] Zheming Tong, Richard W Baldauf, Vlad Isakov, Parikshit Deshmukh, and K Max Zhang. Roadside vegetation barrier designs to mitigate near-road air pollution impacts. *Science of the Total Environment*, 541:920–927, 2016.

[98] Victor JD Tsai, Jyun-Han Chen, and Hsun-Sheng Huang. Traffic sign inventory from google street view images. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 41:243–246, 2016.

[99] Ranga Raju Vatsavai, Auroop Ganguly, Varun Chandola, Anthony Stefanidis, Scott Klasky, and Shashi Shekhar. Spatiotemporal data mining in the era of big spatial data: algorithms and applications. In *Proceedings of the 1st ACM SIGSPATIAL international workshop on analytics for big geospatial data*, pages 1–10. ACM, 2012.

[100] Carlos M Vieira, Henrique A Almeida, Irene S Ferreira, Joel O Vasco, Paulo J Bártolo, Rui B Ruben, and Sérgio P Santos. Development of an impact absorber for roadside barriers. In *Proceedings of the 7th LS-DYNA Forum*, 2008.

[101] Philip A Viton. Notes on spatial econometric models. *City and regional planning*, 870(03):9–10, 2010.

[102] Fei Wang and Changshui Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2007.

[103] Hongjian Wang and Zhenhui Li. Region representation learning via mobility flow. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 237–246. ACM, 2017.

[104] Pengyang Wang, Yanjie Fu, Hui Xiong, and Xiaolin Li. Adversarial substructured representation learning for mobile user profiling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 130–138, 2019.

[105] Pengyang Wang, Yanjie Fu, Jiawei Zhang, Xiaolin Li, and Dan Lin. Learning urban community structures: A collective embedding perspective with periodic spatial-temporal mobility graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(6):1–28, 2018.

[106] Song Wang, Yan Huang, and Xiaoyang Sean Wang. Regional co-locations of arbitrary shapes. In *International Symposium on Spatial and Temporal Databases*, pages 19–37. Springer, 2013.

[107] Geoffrey I Webb. The problem of missing values in decision tree grafting. In *Australian Joint Conference on Artificial Intelligence*, pages 273–283. Springer, 1998.

[108] David Williams, Xuejun Liao, Ya Xue, Lawrence Carin, and Balaji Krishnapuram. On classification with incomplete data. *IEEE transactions on pattern analysis and machine intelligence*, 29(3):427–436, 2007.

[109] Yuankai Wu and Huachun Tan. Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework. *arXiv preprint arXiv:1612.01022*, 2016.

[110] Miao Xie, Zhe Jiang, and Arpan Man Sainju. Geographical hidden markov tree for flood extent mapping. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2545–2554. ACM, 2018.

[111] Miao Xie, Zhe Jiang, and Arpan Man Sainju. Geographical hidden markov tree for flood extent mapping. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2545–2554. ACM, 2018.

[112] Xiaoying Yang and Wei Jin. Gis-based spatial regression and prediction of water quality in river networks: A case study in iowa. *Journal of Environmental Management*, 91(10):1943–1951, 2010.

[113] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, Yanwei Yu, and Zhenhui Li. Modeling spatial-temporal dynamics for traffic prediction. *arXiv preprint arXiv:1803.01254*, 2018.

[114] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Zhenhui Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[115] Zijun Yao, Yanjie Fu, Bin Liu, Wangsu Hu, and Hui Xiong. Representing urban functions through zone embedding with human mobility patterns. In *IJCAI*, pages 3919–3925, 2018.

[116] Jin Soung Yoo and Douglas Boulware. Incremental and parallel spatial association mining. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 75–76. IEEE, 2014.

[117] Jin Soung Yoo, Douglas Boulware, and David Kimmey. A parallel spatial co-location mining algorithm based on mapreduce. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 25–31. IEEE, 2014.

[118] Jin Soung Yoo and Mark Bow. Mining top-k closed co-location patterns. In *Spatial Data Mining and Geographical Knowledge Services (ICSDM), 2011 IEEE International Conference on*, pages 100–105. IEEE, 2011.

[119] Jin Soung Yoo and Shashi Shekhar. A joinless approach for mining spatial colocation patterns. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1323–1337, 2006.

[120] Jin Soung Yoo, Shashi Shekhar, John Smith, and Julius P Kumquat. A partial join approach for mining co-location patterns. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 241–249. ACM, 2004.

[121] Song-Yee Yoon and Soo-Young Lee. Training algorithm with incomplete data for feed-forward neural networks. *Neural Processing Letters*, 10(3):171–179, 1999.

[122] Zhuoning Yuan, Xun Zhou, and Tianbao Yang. Hetero-convlstm: A deep learning approach to traffic accident prediction on heterogeneous spatio-temporal data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 984–992. ACM, 2018.

[123] Zhuoning Yuan, Xun Zhou, Tianbao Yang, James Tamerius, and Ricardo Mantilla. Predicting traffic accidents through heterogeneous urban data: A case study. In *Proceedings of the 6th International Workshop on Urban Computing (UrbComp 2017), Halifax, NS, Canada*, volume 14, 2017.

[124] ZFTurbo. ZF_UNET_224 Pretrained Model. https://github.com/ZFTurbo/ZF_UNET_224_Pretrained_Model, 2018.

[125] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, Xiuwen Yi, and Tianrui Li. Predicting citywide crowd flows using deep spatio-temporal residual networks. *Artificial Intelligence*, 259:147–166, 2018.

[126] Shanghang Zhang, Guanhang Wu, Joao P. Costeira, and Jose M. F. Moura. Fcn-rlstm: Deep spatio-temporal neural networks for vehicle counting in city cameras. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[127] Yunchao Zhang, Yanjie Fu, Pengyang Wang, Xiaolin Li, and Yu Zheng. Unifying inter-region autocorrelation and intra-region structures for spatial embedding via collective adversarial learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1700–1708, 2019.

[128] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[129] Yaotian Zou, Andrew P Tarko, Erdong Chen, and Mario A Romero. Effectiveness of cable barriers, guardrails, and concrete barrier walls in reducing the risk of injury. *Accident Analysis & Prevention*, 72:55–65, 2014.