A Framework for Spatial Database Explanations

by

Anique Tahir

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved May 2018 by the
Graduate Supervisory Committee:

Mohamed Sarwat, Chair
Sharon Hsiao
Ross Maciejewski

ARIZONA STATE UNIVERSITY

August 2018

ABSTRACT

In the last few years, there has been a tremendous increase in the use of big data. Most of this data is hard to understand because of its size and dimensions. The importance of this problem can be emphasized by the fact that Big Data Research and Development Initiative was announced by the United States administration in 2012 to address problems faced by the government. Various states and cities in the US gather spatial data about incidents like police calls for service.

When we query large amounts of data, it may lead to a lot of questions. For example, when we look at arithmetic relationships between queries in heterogeneous data, there are a lot of differences. How can we explain what factors account for these differences? If we define the observation as an arithmetic relationship between queries, this kind of problem can be solved by aggravation or intervention. Aggravation views the value of our observation for different set of tuples while intervention looks at the value of the observation after removing sets of tuples. We call the predicates which represent these tuples, explanations. Observations by themselves have limited importance. For example, if we observe a large number of taxi trips in a specific area, we might ask the question: Why are there so many trips here? Explanations attempt to answer these kinds of questions.

While aggravation and intervention are designed for non spatial data, we propose a new approach for explaining spatially heterogeneous data. Our approach expands on aggravation and intervention while using spatial partitioning/clustering to improve explanations for spatial data. Our proposed approach was evaluated against a real-world taxi dataset as well as a synthetic disease outbreak datasets. The approach was found to outperform aggravation in precision and recall while outperforming intervention in precision.

ACKNOWLEDGMENTS

I am extremely thankful to my advisor, Dr. Mohamed Sarwat, for his help in making this thesis possible. It has been a great learning experience working with him for over two years. I am grateful for his honest advice and guidance. In addition, I would also like to express my gratitude towards Dr. Ross Maciejewski and Dr. Sharon Hsiao for their insightful comments and critiques. Their feedback helped in improving the quality of the work. Last but not the least, I am grateful for the unconditional support provided by my family.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

Chapter 1

PROBLEM OVERVIEW

1.1 Introduction

When we analyze data, we might make some observations that pique our curiousity. We might want to explain trends or anomalies in data. Without an automated system, a data analyst has to go through several manual operations to find an explanation. If the size of the data is small, doing so might take a bit of time, but if we are dealing with a large amount of data, it can become a tedious process. Our system is designed to answer questions based on observations over a large amount of data. In the past few years there has been a lot of developments in the area of Big Data.

We can find several examples of decisions being made using big data analysis in everyday situations. Companies like Uber and Facebook handle large amounts of spatial data every day. This data can be used to improve service. UPS is saving millions of gallons of fuel per year by using Big Data Analytics. UPS uses On-Road Integrated Optimization and Navigation system(ORION) to determine the order of delivery, routes, and loading plans (InformationWeek, 2013).

Given the tremendous benefits of automated analysis, the motivation for this thesis was to create an automated system to help analysts answer questions about observations. The idea is to create a system which works on spatial data in general. This thesis comes up with a generic system to explain observations on spatial data.

The New York City Taxi and Limousine Commission (NYC TLC) regulates yellow cab taxis(NYC and Commission, 2016). It has licensed around 146,000 distinct

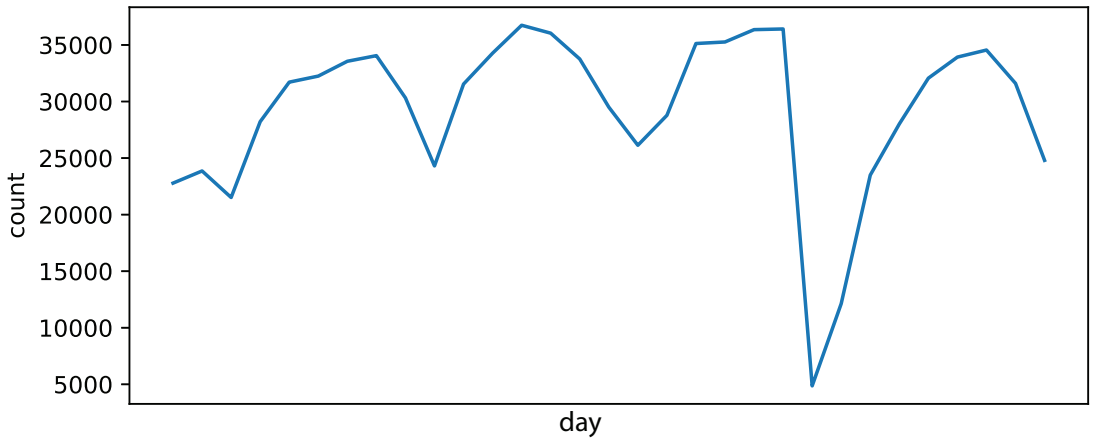

Figure 1.1: Average number of yellow cab trips against the day for January 2016

drivers. In order to view a demonstration of the database explanation system, we can use this dataset. The Yellow Cab data has several attributes including spatial attributes such as latitude, longitude and non spatial attributes such as tip amount and total amount for each trip. Fig. 1.1 shows the number of yellow cab trips against time for January 2016. The sharp decline in the number of trips in the last quarter of the month is pronounced. One might be inclined to look up the date when the number of trips crashed. In this paper, we look at different approaches to explain these types of observations. In fact, we also look at observations with a spatial dimension.

Whenever we analyze some data, we might be tempted to find out a reason for specific observations. While some data observations might be interesting, in order to make decisions based on these observations, one might find it useful to find an explanation. For example, someone looking to start a business might look at locations in which their specific category of business succeeds. It might be even more beneficial to find out why it prospers in that specific location.

This thesis describes a system that we have created to explain database observations. Our system intends to produce spatial explanations. There are two types of input to our system. One of the inputs is the dataset we want to analyze. The second

input is our observation. The observation can be in the form of an aggregate query. Our system uses one of several different solutions to find an explanation depending on what the user wants. The output of our system is an explanation for our system based on predicates.

The explanations produced by our system are formally defined later on in this document. However, it is important to introduce the basic idea so the reader can follow along. In our proposed system, explanations are parts of the data that have a significant effect on what we are observing. For example, if we are observing tips for taxi trips and removing a few tuples from the dataset has a significant effect on the tips, then the few tuples would be considered as an explanation for our observation. Another definition of explanations in our proposed solution involves looking at different parts of the data. For example, if we divide our data into a number of parts, the parts which deviate from the average value of tip percentage are considered as explanations because they introduce the largest differences.

In this thesis we define a taxonomy(Section 3.1) for observations and explanations. The observation we made in Fig.1.1 can be considered as a non spatial observation. Its explanation can either be non spatial or spatial. The first class in our taxonomy deals with non spatial explanations for non spatial observations(Section 3.1.1). The second class is related to spatial explanations for non spatial observations(Section 3.1.2) while the last class is about spatial explanations for spatial observations. Many geospatial datasets that we encounter contain time as one of the attributes. When we talk about spatial explanations, we do not take time into consideration. Instead, time is considered as a non-spatial attribute.

There are three main approaches to explanation that we study in this thesis. Each approach has been extended to satisfy our taxonomy. Each approach relies on the fact that the observation is an aggregated attribute in our data set while the explanation is

a predicate. **Aggravation**(Section 3.2.1) is based on the principle that if we consider only the tuples in our database satisfying our explanation predicate, the value of the observation on our modified dataset will be our measure of aggravation(Roy and Suciu, 2014; Meliou *et al.*, 2014).

In contrast, **Intervention**(Section 3.2.2) measures the influence of our explanation i.e. what will be the value of our observation when all the tuples satisfying our explanation predicate are removed(Roy and Suciu, 2014). We also extend Intervention to introduce **Hierarchical Intervention**(Section 4). This approach measures the value of intervention when the explanation consists of a cluster of spatial polygons in our explanation predicate.

Finally, **Salient Features**(Section 3.2.3) can be used to find explanations(Chirigati *et al.*, 2016). Each salient feature encapsulates a polygon where an attribute in the dataset is pronounced. The correlation between a salient feature of the observation and the salient feature of the explanation can give us possible explanations.

Each approach has its own set of advantages and disadvantages. While one approach might give us very specific explanations that show a textbook example of our observation, another might give us an explanation which is hard to see in the context of the observation but has a large overall impact on it. One of the objectives of this paper is to compare which approach is suitable considering its context.

We implemented(Section 4.4) these approaches, including hierarchical intervention using distributed data frameworks(Borthakur, 2007; Dean and Ghemawat, 2008; Shanahan and Dai, 2015; Zaharia *et al.*, 2016). We made optimizations in our implementation to make sure our approach is orders of magnitude faster than the naive approach. The implementation of salient features was used from the Data Polygamy framework (Chirigati *et al.*, 2016).

In order to compare each approach, we defined a few evaluation metrics. The **Intensity**(Section 4.3.1) metric measures how relevant each explanation is to the observation. To be more specific, it measures the value of the observation for the top explanations for each approach. The **Influence**(Section 4.3.2) metric measures the observation when the top explanation is removed from the data. We also compare the speed(Section 5.3) of our implementations of each approach.

1.2 Related Work

There has previously been some work related to database explanations. Most of this work revolves around the notion of causality. There is existing work in the field of Artificial Intelligence by Zhang and Zhang (2002) which expresses relationships between different attributes in a dataset as conditional probabilities. Based on the conditional probability each tuple is given a set of binary rankings.

The work by Meliou *et al.* (2010) is a survey which looks at causality from the perspective of a database problem. Traditionally, work on causality from the database perspective mainly deals with provenance i.e. events which occurred chronologically. This solution, however, only works with databases with timestamps. This paper also looks at the degree of responsibility which is defined as the number of tuples which have to be removed to change a binary observation. An example of a binary observation is winning and losing an election for instance. If a candidate wins by a high margin, each tuple has a lower degree of responsibility. On the other hand a close victory for a candidate increases each voters degree of responsibility.

The paper by Meliou *et al.* (2014) is a survey of work in causality and explanations in both the Artificial Intelligence and Database communities. The take away from this survey is that AI problems tend to have a bigger causal network while database problems tend to have more variables.

There has also been a lot of work on correlation which shares some common ground with the work on explanations. One interesting recent work on the subject is the Data Polygamy framework(Chirigati *et al.*, 2016). This framework is designed to find the correlation between a corpus of datasets. It uses the peaks and troughs of the data to calculate *salient features* (Dunn and Clark, 1986). The positive and negative correlation between these salient features can be used to decide whether dataset are related(Su *et al.*, 2014). The objective of our system is a bit different from finding corelations. There are a number of different factors which can effect observations. Correlation might be one of these attributes in certain cases but if we ignore other criteria such as selectivity then we might get results which do not have a significant impact. E.g. if two attributes are highly corelated in a certain spatial cluster but the selectivity of the cluster is small, it would lead to a low impact on the observation.

There has also been work related to why not explanations in databases. The work by (ten Cate *et al.*, 2015) looks at the question of why some tuples are missing from database results. This paper uses the assumption that the relationship between tuples is defined in the form of an ontology. The paper uses the relationship between the ontology for a schema and the ontology for an instance of the schema to judge whether an explanation exists. The ontologies that are used can be created manually or automatically. Using provenance (Cheney *et al.*, 2009) can help in creating ontologies.

Our solution mainly builds upon work by Roy and Suciu (2014) which outlines a formal approach to explain data. The main solutions outlined in that work are Aggravation and Intervention. The work by Roy and Suciu (2014) is designed to work with non spatial datasets. One of the main points of the paper is that their approach works on a dataset which can span several tables related by primary and foreign keys. While the approach outlined by Roy and Suciu (2014) is great for non spatial data, it does not translate well in the spatial domain. This approach develops on previous

work in causality and influence. It also resembles data mining concepts related to association rule mining(Agarwal *et al.*, 1994; Tan *et al.*, 2006). In association rule mining sets of attributes that occur together are assigned a support and confidence. The support measures the frequency of occurrence of a set of attributes while confidence measures how frequently an attribute occurs with another set of attributes. The work by Koperski and Han (1995) looks at association rule mining in a spatial context. Given a set of spatial relationships, it applies association rule mining to find relationships that frequently occur together. There is a difference between association rule mining and our prosposed approach. First of all, our proposed approach uses a user defined observation. Secondly, in our approach we are not looking at the associations but rather at the effect of removing or filtering pieces of data. A high association between predicates does not necessarily mean that removing them will significantly effect the observation.

Spatial Analysis is very popular in GeoScience and GeoInformatics. Regression techniques can be used to explain data(Dunn and Clark, 1986; Cleveland and Devlin, 1988). The idea behind regression techniques is to express an attribute that we are interested in as a dependent variable. This dependent variable can then be expressed in the form of parametric equation involving other attributes in the dataset as independent variables. An example of a loss function for this kind of regression is Ordinary Least Squares(OLS)(Dismuke and Lindrooth, 2006). The user of this system decides a dependent variable and a set of explanation variables. The resulting equation has coefficients assigned to each explanatory variable as well as a bias term. This results in a curve fitting problem. The curve fitting problem is solved using regression i.e. the coefficient terms and the bias are iteratively adjusted until the sum of squared error between the predicted curve and the ground truth results in a minimum value. Regression techniques are widely used for spatial data analysis.

However, they depend on predefined spatial partitioning and look at the data as a whole rather than looking at it from the perspective of a user defined observation compared to our solution.

Geographically Weighted Regression(GWR) expands on OLS regression for geospatial data(Brunsdon *et al.*, 1998; Charlton *et al.*, 2009). GWR tries to use the regression equation for each feature in the dataset. It uses the idea that spatially co-located points contribute more to each other by using a spatially aware kernel function. A kernel function like a Gaussian, for example, gives more weight to nearby points than to points which are far apart.

Besides work related to explanation, there has been a lot of research in the area of spatial correlation. Much of the work in this area extends from multiple old works by Getis(Getis, 1991; Ord and Getis, 1995; Getis and Ord, 1996; Getis and Griffith, 2002; Getis, 2007). The Getis Ord statistic (Ord and Getis, 1995) for example is useful in showing us areas with high local spatial associations. The Moran's I statistic is useful for measuring the spatial heterogeneity of the data(Assuncao and Reis, 1999; Zhang *et al.*, 2008). Moran's I is useful in hot spot analysis which can be viewed as a step in the way of finding explanations.

1.3 Contributions

In this paper we extend several approaches and compare them. There are three major contributions in this thesis:

- We extend aggravation and intervention for spatial explanations/observations. Aggravation and Intervention techniques in literature are designed for giving non spatial explanations for non spatial observation. When we look at spatially heterogeneous data, the spatial context has impact on the value of the explanations.

- We extend the use of salient features to give spatial explanations for simple observations based on attributes. Salient Features can be used to compare the correlation between attributes between multiple datasets. However, we re purpose the use of salient features for explanations. We use the salient features for attributes in the same dataset to form explanations.

- We introduce a new approach: Hierarchical Intervention. This approach uses spatial partitioning/clustering. We find that there is a difference between explanations for spatially heterogeneous data for different dimensions of spatial explanations. Therefore, we propose a new approach which uses a spatial hierarchy. This accounts for explanations for multiple dimensions of the data.

- We introduce a method to balance influence and intensity to give better explanations. Influence and Intensity measure the global and local impact of an explanation, respectively. We acknowledge that the user of our system can have a non binary preference for either. We construct our system in a way that it produces explanations as a linear relationship between influence and intensity.

- We introduce an automated system for giving explanations based on our findings. The system that we have defined has a lot of parameters. It requires observations, coefficients, arithmetic relationships and selectivities as parameters. We have designed a Web based User Interface to help a user with little knowledge of the system to get explanations for their observations.

One of the contributions of this thesis is to compare the different approaches. In order for us to compare different things, we need to do so on the basis of a common standard. The different solutions to the explanation problem are structurally very different from each other. As previously mentioned, some of them are originally not

designed to handle spatial data. We have designed a common taxonomy to compare all these different solutions. On top of designing this taxonomy, our extension of each approach is designed to make sure each solution adheres to the taxonomy. Even though, this still leaves room for differences between each solution, it gives us room for comparison.

We also define a number of evaluation metrics and an approach which uses them to come up with better explanations.

Chapter 2

PRELIMINARIES

In order to understand the system. The reader needs to understand a few concepts related to Data Mining and Databases. This chapter attempts to give a brief overview of the underlying concepts needed to understand the system.

2.1 Star Schema

Our system has the underlying assumption that the data that will be used to generate explanations is the Fact Table for a Star schema(Giovinazzo, 2000; Adamson, 2010). To understand the Fact Table, it is important to understand the structure of the Star Schema. A traditional relational database system contains a set of tables related by primary and foreign keys. For instance, we can use our running example of the NYC Taxi Trips dataset to illustrate a Star Schema. The trips and payment type can be represented in separate tables. One trip can have a single payment type while a payment type can be used in multiple trips. This is an example of a one-to-many relationship. In order to represent this data in a relational database, the *trips* table and the *payment_type* table need to have a primary and foreign key. Another way of storing this data without primary and foreign keys is to save all payment type information against all trips. Tables. 2.1 and 2.2 show some dummy data to illustrate our example for our running example of the NYC taxi data.

The illustrated example has a normalized schema(Beeri *et al.*, 1988). Table 2.2 can be considered as the central part of the schema because it contains the foreign key to the *payment_type* table.

PaymentTypeId	**Name**
1	Credit Card
2	Cash
3	No Charge

Table 2.1: Payment Type Table

pickup_lat	**pickup_lng**	**PaymentTypeId**	**tip_percentage**
34.4	-74.2	1	15.3
34.6	-74.1	1	10.2
34.6	-74.3	1	9.8
34.8	-74.6	2	11.2
34.6	-74.3	1	10.7
34.9	-74.1	3	0.0

Table 2.2: Trips Table

This type of schema where there is a central table consisting of facts while the remaining tables contain the meta data is called a star schema. The central table is called the Fact Table, whereas, the tables containing the meta data are called the Dimension Tables. In the case of the schema we just defined, Table. 2.1 is the dimension table while Table. 2.2 is the fact table.

2.2 Observations

Observations are features in the data that the user wants to explain. Observations are defined as arithmetic expressions over a set of aggregate queries. Let F be the fact table in our star schema dataset. In the course of this document, we will be using

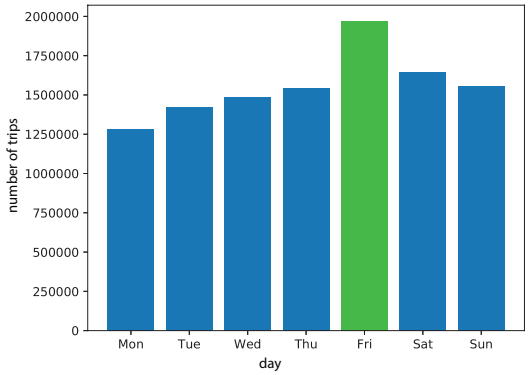

Figure 2.1: An histogram showing an example observation

relational algebra expressions defined by Elmasri and Navathe (2011) for aggregate expressions. Thus, the $\mathscr{F}$ symbol represents an aggregate function. An aggregate query is defined as:

$$_{A}\mathscr{F}_{B}(D), A \in F$$

B is an aggregate function. A is an attribute in our dataset. D is the fact table. Examples of an aggregate function include SUM, COUNT, and AVERAGE. We use SQL to construct an example for an observation. Queries 2.1 and 2.2 show examples of aggregate queries.

Observations made on data can also be represented on histograms. Fig. 2.1 shows an example of an observation. The green bar on the histogram represents an aggregate query where the day is Friday.

```
1 SELECT AVG(tip_percentage) FROM
2 FROM nyc_data
3 WHERE payment_type = 1
```

Query 2.1: Aggregate Query for average tip percentage with credit cards

```
SELECT AVG(tip_percentage) FROM
FROM nyc_data
WHERE payment_type = 2
```

Query 2.2: Aggregate Query for average tip percentage with cash

Using these aggregate queries we may form an observation based on the ratio of tip percentage with credit card against tip percentage with cash.

$$observation = \frac{\text{Query. 2.1}}{\text{Query. 2.2}}$$

2.3 Explanations

We represent explanations as a predicate. A predicate is a conditional statement which results in a boolean value. We go into more details for the formal definition of different kinds of explanations in Section 3.1. If we consider Queries 2.1 and 2.2 as an example. The explanation would be in the form of a predicate:

$$tip\_percentage = 15.3$$

Let D be our solution space. We can define our predicate to be the function P. Let X represent a set of attributes in our schema. Our explanation can now be formally defined as:

$$X|P(X) := \begin{cases} \text{true}, & \text{if } X \in D \\ \text{false}, & \text{otherwise} \end{cases} \tag{2.1}$$

Note that P is an open ended function. In the case of spatial explanations, P can take the form of a spatial function like $ST\_CONTAINS$ in PostGIS i.e. whether a polygon contains a point. In the non spatial context, P can represent functions like 'greater than', 'less than', etc.

2.4 K-Folds Cross Validation

K Folds cross validation is a technique for data evaluation(Kohavi *et al.*, 1995; Refaeilzadeh *et al.*, 2009). The data is divided into k parts. One of the parts is used as a training set and the remaining parts are used as test sets. The purpose of the training set is to model the data. Once we have a model, we can use it to classify and/or predict unseen data. The test set is used to evaluate how well the model was trained.

2.5 Precision and Recall

Precision and Recall are methods for evaluation(Olson and Delen, 2008; Powers, 2011). Whenever we classify data we may have true positives, false positive, true negative and false negatives. Precision is defined as:

$$precision = \frac{\text{true positives}}{\text{false positives} + \text{true positives}}$$

Recall is defined as:

$$recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

2.6 Distributed Processing Frameworks

Map Reduce(Dean and Ghemawat, 2008) is a framework for data processing which is designed for taking distributed and parallel computation into perspective. There are three main operations in a mapreduce process: map, shuffle and reduce. The map operation assigns a key to each element and performs any necessary transformations. The shuffle operation relocates the elements such that elements with the same key are nearby(since they are going to need each other in calculations). The reduce step performs a calculation on each element with the same key and returns the output.

Spark(Shanahan and Dai, 2015; Zaharia *et al.*, 2016) is a distributed and parallel processing framework. Spark uses a directed acyclic graph to perform calculations. Since the DAG created by Spark can have a lot of common nodes between tasks, the computational complexity of the operation is reduced compared to MapReduce.
Geospark (Yu *et al.*, 2015) is a framework for performing several spatial operations on data in Apache Spark. It also has a component which helps in data visualizations(Yu, 2018).

2.7 Front End Visualization Tools

React(Facebook, 2018b) is a front end framework which originated in Facebook. The React framework allows interfaces to be designed using components. Each component has properties and a state. A component can have subcomponents. This makes it simpler to design interfaces which show consistent data across components. Some of the charts included in the interface make use of the eCharts library(Baidu, 2018).
MapBox(Mapbox, 2018) is a library for displaying maps. The maps provided by mapbox consists of tiles and vectors. Each tile represents a cube of the map while vectors are shapes which represent roads, buildings, etc. Deck.gl(Uber, 2018) is a library for creating an overlay on top of the map. Examples of overlays include scatterplots, cartograms etc. Matplotlib was also used for static plots for evaluation(Hunter, 2007). We have used all of these tools in a GUI for our our framework. The details of the implementation can be found in Section 4.4

Chapter 3

SYSTEM ARCHITECTURE

3.1 Taxonomy

We define a taxonomy for observations and explanations. This taxonomy will help us standardize different approaches to explanation. It will also help us compare these approaches against each other. Our observation/explanation standard is based on the assumption that observations take the form of a query, whereas, explanations take the form of a predicate. Let D be the dataset that we are interested in. Let A_D be a set of attributes of D. Let V_a represent the set of values for in D for attribute a where $\{\neg\emptyset, \emptyset\} \subset V_a$. Then we can define our candidate explanation ϕ as

$$\phi \models \wedge_{i,j}(i = j)$$

where $i \in A_D, j \in V_i$

The observation is simply just an aggregate query on D. Each approach returns a set of candidate explanations and there is a value associated with each candidate explanation which measures its score. We define the scoring function for each approach in its respective section in the document.

3.1.1 Non spatial explanation for non spatial observations

Non spatial observations are aggregate queries on D. These queries can take any form. Non spatial explanations are candidate explanations based purely on non spatial attributes of the data. Let S be the spatial attributes in D. Our candidate explanation can be defined as

$$\phi \models \wedge_{i,j}(i = j)$$

where $i \in A_D, i \notin S, j \in V_i$

We can look at an example of non spatial explanations for non spatial observations by looking at the NYC TLC data for Yellow Cab. There are several attributes which are continuous e.g. tip amount, trip distance, and fare amount. A month of taxi trip data consists of 10 million tuples. This means that if we find all possible combinations of attribute values, it can exceed over a million. Instead each of the approaches we discuss tries to simplify how the explanations are calculated. For instance, salient features use crests and troughs in the value of an attribute over time. Fig. 1.1 shows the average number of taxi trips per day for one month. If we consider twenty thousand trips as an upper bound, we only get part of the curve which we call a salient feature(Section 3.2.3). We can use this salient feature for our explanations. The upper bound in the case of salient features is either user defined, or automatically generated. We discuss the details of the salient features implementation in Section 3.2.3

3.1.2 Spatial explanations for non spatial observations

Spatial explanations take the spatial attributes of the data into consideration. In contrast to non spatial explanations, the candidate explanations for spatial explanations contain polygons. Since we base our method on the assumption that important tuples are spatially co-located, points which fall inside these polyons form a candidate explanation. Let S be the spatial attributes of D. Let P be the set of all possible polygons from S. Let G be a function such that$G(s,t)$ is true when $(s,t) \in P$ and false otherwise, where s,t are the dimensions of a point in the Cartesian plane. Our candidate explanation can be defined as,

$$\phi \models \vee_{l,k} G(l,k)$$

where $k, l \in A_D$

To illustrate spatial explanation in action, we can use NYC TLC data again. Fig. 3.1 shows the explanation in terms of tip percentage where the observation is also the average tip percentage.

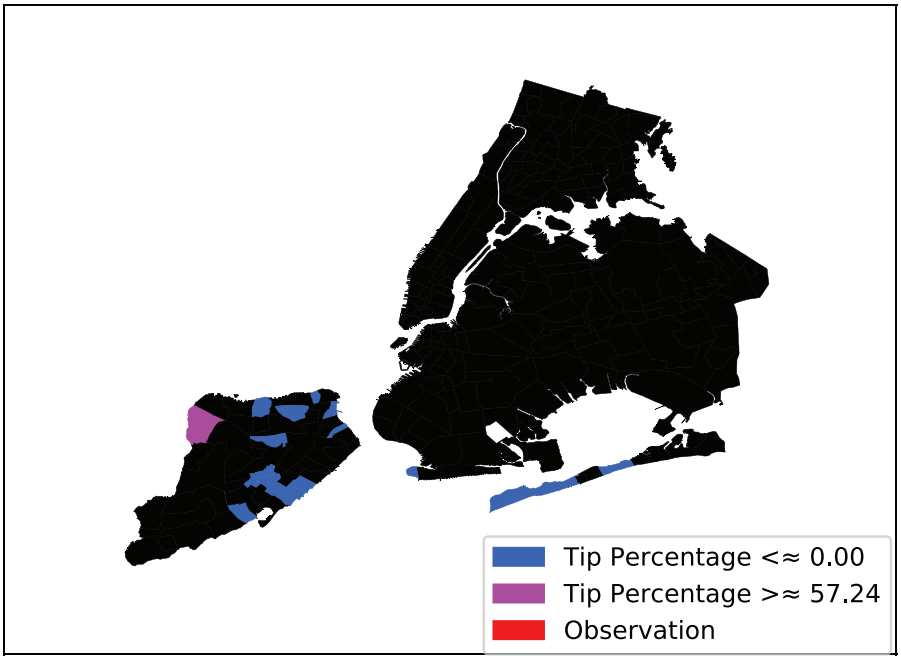

Figure 3.1: An example of spatial explanation

The polygons painted purple show polygons in the candidate explanation where the tip percentage is high, while polygons painted blue show candidate explanations where the tip percentage is low. It should be noted that P has a high number of permutations. It is up to the approach to decide which polygons to include in the candidate explanation. For instance, hierarchical intervention may choose polygons in a spatial proximity while aggravation may choose otherwise.

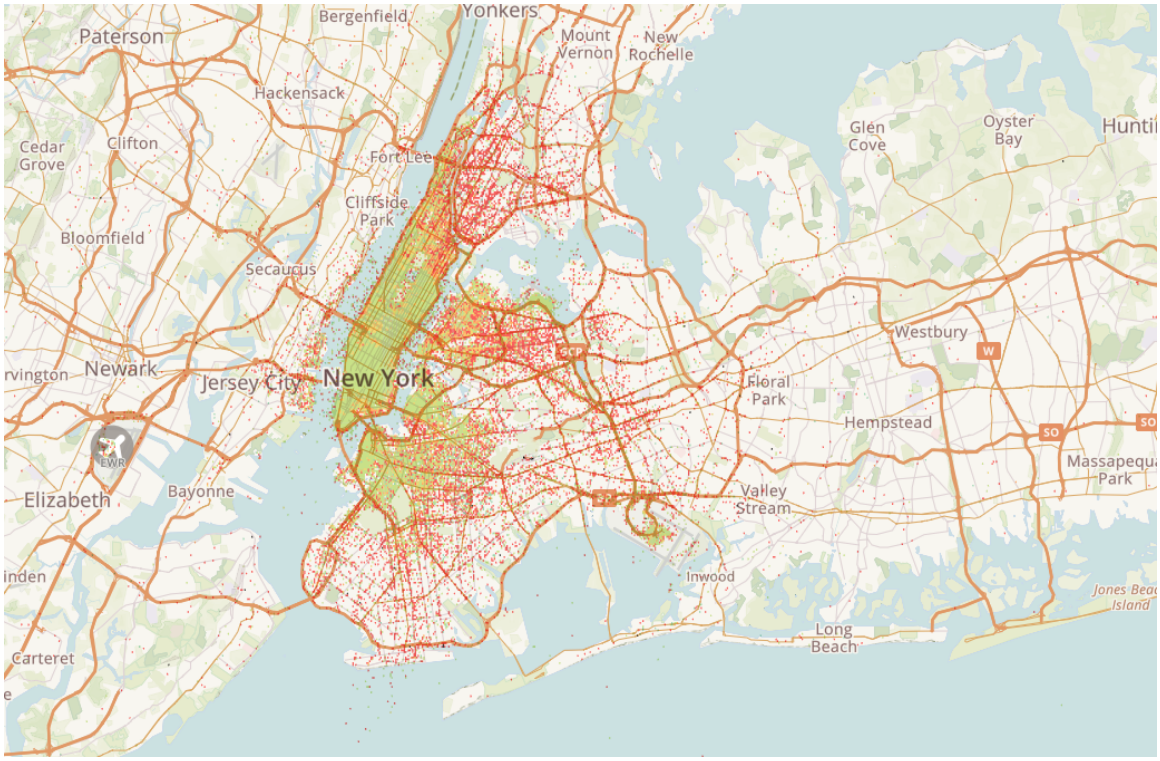

Figure 3.2: Heatmap for NYC trips for January 2016

Another interesting consideration when we are talking about spatial explanations is the density of the data. Fig. 3.2 shows the heatmap for tip percentage with respect to pickup coordinates. It is interesting to observe that the explanations provided in Fig. 3.1 are all areas with low density of data. One of the reasons for this is because each approach takes some liberty with our definition of the taxonomy. Even though, we defined P to contain all permutations of polygons, an approach may use limited polygons, such as neighborhoods or zones. The way the explanation is ranked also plays a large role.

3.1.3 Spatial explanations for spatial observations

In the last two classes of our taxonomy, we only looked at non spatial observations. In this class however, we will look at spatial observations. Spatial observations are

similar to non spatial observation. Both spatial and non spatial observations are represented by an aggregate query. The difference is that spatial observations have a polygon in the predicate. The spatial explanation in this class can be defined in the same way as in Section 3.1.2.

Fig. 3.3 shows an example of a spatial explanation for a spatial observation. The observation, in this case, is the average tip percentage when the pickup zone is LaGuardia airport. The candidate explanation displayed are also based on tip percentage, though they could have been any other attribute as well. Introducing a spatial predicate has a large observable impact on the explanation compared to the one in Section 3.1.2

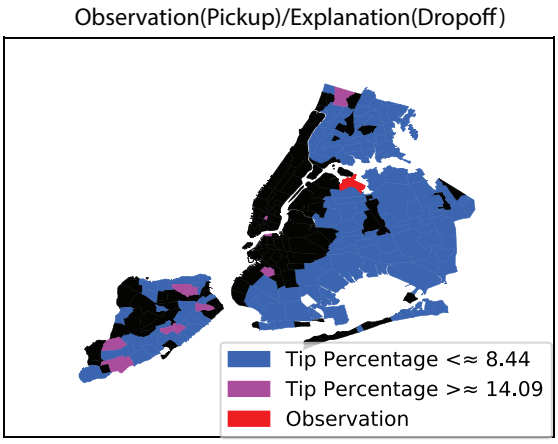

Figure 3.3: An example of spatial explanation for a spatial observation

There is a visible difference between the explanations in Fig. 3.1 and Fig. 3.3. This is because Fig. 3.1 shows the explanation for average tip percentage in general where the entire New York area is covered in the observation. Fig. 3.1 on the other hand shows the explanation in the context of the dropoff zone for the trip where the pickup zone is LaGuardia airport.

3.2 Studied Approaches

There are three main approaches that we use in this thesis. Aggravation, Intervention and Salient Features. We also introduce hierarchical intervention. This approach is an extension of intervention which takes spatial clusters into account. Originally, Aggravation and Intervention were used in a non spatial context in related works(Chirigati

et al., 2016). However, we extended these solutions to fit into our spatial taxonomy.

3.2.1 Aggravation

The first approach we will look at for explanation is aggravation. This approach tries to look at tuples which 'aggravate' the results. The main idea is to calculate the value of the observation for each candidate explanation. The candidate explanation with the highest value has the highest weight as the explanation.

Let q be our observation query(aggregate query). Let ϕ be our candidate explanation. Let D_ϕ be the dataset that satisfies our candidate explanation i.e. $D_\phi = \sigma_\phi(D)$. Let Q be a scalar function that takes in a dataset and gives the value obtained when q is applied to this data. The degree of candidate explanation, δ_{agg}, by aggravation can be simplified as $Q(D_\phi)$.

In order to have a degree of explanation which is high when we are closer to our desired explanation, we need to have a direction for our explanation. For example, if we want to observe high values of tip percentage in the NYC TLC data, our direction will be high. If we want to observe lower values of tip percentage, our direction will be low. If our direction is low, we are interested in the lower values. However, since we want our degree of candidate explanation to be higher when we have the desired candidate explanation, we can simply reassign its value.

$$\delta_{agg} := \begin{cases} Q(D_\phi), & \text{if } direction = high \\ -Q(\phi), & \text{otherwise} \end{cases} \tag{3.1}$$

In order to decrease the number of permutations for candidate explanations, we can bucket the values. Consider an attribute a in the dataset. Let V_a represent each value that a can takes up, where $V_a = \{v_1, v_2, ..., v_n\}$ and n is the number of tuples. Let the minimum value in V_a be represented by a_{min}.The mean value, μ, is defined as $\frac{\Sigma_i v_i}{n}$.

The standard deviation, σ, is defined as $\sqrt{\frac{1}{n-1}\Sigma_i(v_i - \mu)^2}$. Each value is assigned to a bucket. Let b_i be the bucket the value, v_i is assigned to. Then

$$b_i = \left\lfloor \frac{v_i - a_{min}}{\sigma} \right\rfloor$$

Now, instead of using all the permutations in V_a as part of our candidate explanations, we can rather use all the distinct values of buckets. The predicate represented by each bucket b_i is simply

$$a \geq (a_{min} + b_i \times \sigma) \wedge a < (a_{min} + b_i \times (\sigma + 1))$$

This decreases the number of permutions for candidate explanations by a factor of $\frac{1}{\sigma}$. As a hands on example of this approach consider the observation on the NYC TLC data represented in Query 3.1.

```
SELECT AVG(trip_distance) FROM
FROM nyc_data
```

Query 3.1: Aggregate Query for average tip percentage

We are interested in the candidate explanation in the context of the number of passengers. The top explanation for this observation gives $Q = 6.00$, $b_i = 3$. Since we know the bucket and σ. The explanation predicate turns out to be $passenger\_count \geq 3.97 \wedge passenger\_count < 5.30$. If you are a yellow cab driver, this explanation may tell you to be prepared for a long trip if you have four or five passengers.

Now that we have described the aggravation approach for the nonspatial case, we can extend it to handle spatial observations and explanations. We introduce a dataset of polygons P. The set P consists of distinct non overlapping polygons over our dataset D. Let s,t be the spatial attributes in D. Each tuple in P has two attributes:

polygon_id, and *polygon*. We create a new dataset

$$D' \leftarrow D \bowtie_{contains(P.polygon,(s,t))} P$$

Now we can use our spatial candidate explanations and observations defined in our taxonomy(Section 3.1) on out new dataset D'

Fig. 3.3 can be considered as an example of spatial aggravation where the spatial partitioning is based on taxi zones as dropoff locations. The observation is the tip percentage where the pickup zone is LaGuardia airport.

3.2.2 Intervention

Intervention is an approach inspired by the concept of influence. It builds on the aggravation approach(Section 3.2.1). Intervention tries to measure how much our observations would change had our explanation not been present. Let D be the dataset we are interested in. Let Q be a function which returns the value of our observation given a dataset. Keeping our taxonomy in context, this means Q returns the value of our aggregate observation query. Let ϕ be our candidate explanation. Let $\Delta_\phi \leftarrow \sigma_\phi(D)$. Let $D_\phi = D - \Delta_\phi$. The direction of our observation can either be *high* or *low* depending on whether we are interested in the greatest or least values of observation respectively. Our degree of candidate explanation by intervention, δ_{int}, can then be expressed as,

$$\delta_{int} := \begin{cases} -Q(D_\phi), & \text{if } direction = high \\ Q(D_\phi), & \text{otherwise} \end{cases} \tag{3.2}$$

We want the degree of candidate explanation by intervention to be higher the closer we are to the direction of the observation, therefore, we use the negative value when the direction is high. If the influence of the candidate explanation is high, it will

result in a low observation value once the candidate explanation is removed from the dataset.

Since intervention extends the idea presented by aggravation, it has similar issues when it comes to the number of permutations for candidate explanations. Similar to our approach in aggravation(Section 3.2.1), we can reduce the number of permutations by bucketing the attributes. We can extend intervention for spatial observations and explanations the same way we did for aggravation. The set P consists of distinct non overlapping polygons in our dataset D. Let s,t be the spatial attributes in D. Each tuple in P has two attributes: *polygon_id*, and *polygon*. We create a new dataset $D' \leftarrow D \bowtie_{contains(P.polygon,(s,t))} P$. Spatial observations can now be defined as aggregate queries with a predicate containing *P.polygon*. Candidate spatial explanations can be defined as candidate explanations containing *P.polygon* as emphasized in our taxonomy(Section 3.1).

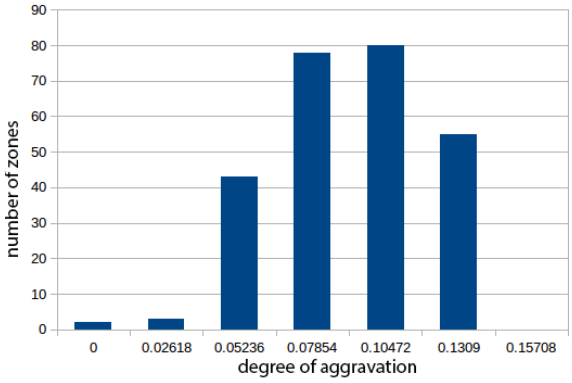

Figure 3.4: Number of Zones against Degree of Spatial Aggravation

In order to compare the differences between aggravation and intervention, it is helpful to look at their degrees of aggravation and intervention respectively. As an example, we look at the NYC TLC data where P is the set of taxi zones. Fig. 3.4 shows a histogram for the number of taxi zones which fall inside different range of the degree

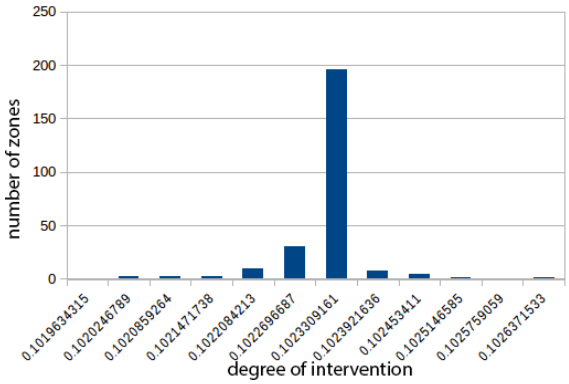

Figure 3.5: Number of zones against their degree of Spatial Intervention

of candidate explanation by aggravation. Fig. 3.5 shows a similar histogram for the number of zones between certain ranges for the degree of candidate explanation by intervention. The observation in both cases is the average tip percentage. The degrees of explanation and the number of zones represent the domain and range, respectively. It is immediately evident from these two histograms that the domain for explanation by aggravation is 150 times larger than intervention while the range for explanation by intervention is 2.5 times larger than aggravation in this instance of the problem. The reason for having a small domain for intervention is because the influence of removing one of the zones on the observation is very small, whereas, each zone may have a large difference in average tip percentage compared to other zones. We will look further into the implications of these differences when we evaluate these approaches(Section 5).

3.2.3 Salient Features

Another approach for finding explanations is the use of salient features. As the name suggests, salient features intend to highlight portions of data which stand out. One of the examples which illustrates this can be seen in Fig. 1.1. The figure shows the average number of taxi trips per day over the course of a month. There is a large

noticeable decline in the number of trips. The data representing this decline can be considered as a salient feature. Salient features like these tell a lot about the data. For example, if you were to look up the date illustrated in the example where the number of taxi trips drops, you would find that there was a travel ban because of extreme weather conditions.

Salient features are based on temporal or spatio temporal scalar functions. This approach does not work with non-temporal functions. We define two types of scalar functions: 1-D scalar functions and 2-D scalar functions. 1-D scalar functions map time to a scalar value. For example, the data presented in Fig. 1.1 maps time to the number of taxi trips. 2-D scalar functions map time and space to a scalar value. Fig. 3.1 shows an example of a 2-D scalar function for one time step. Since our taxonomy is concerned with only the spatial domain of this problem, we will be considering a 2-D scalar function where one time step covers the entire scope of our dataset, i.e. 1-D scalar function based on space. We go from 2-D to 1-D by only considering the entire time range as a single entity. It might be easier to understand this by considering a rectangle and a line. If a rectangle only has a unit width, it forms a line.

Now that we understand what salient features are, the natural question would be how to find salient features in a dataset. In order to calculate salient features, we define two threshold values θ^+ and θ^-. *Positive salient features* are the parts of our scalar function where its value is greater than θ^+ i.e. $f^{-1}([\theta^+, \infty))$. *Negative salient features* are the parts of our scalar function where its value is less than θ^- i.e. $f^{-1}((-\infty, \theta^-])$. The local maxima and minima are referred to as *critical points* i.e. $\nabla f = 0$

Our extension of this approach for spatial explanation stems from the fact that each feature can be represented as a predicate. For instance if we consider the function represented in Fig. 1.1. Let θ^- be 20,000. Then the negative salient feature can be

represented as a predicate based on time and the number of trips. In the case of a 2-D scalar function, the predicate would be based on the time, attribute, and a polygon. In case of a 1-D spatial scalar function, the predicate would consist of a polygon, and an attribute. In order to calculate an explanation, we measure the co-relation between scalar functions of different attributes in a dataset. Let f_1 be the scalar function for one attribute and f_2 be the scalar function of another attribute. Let F_1 and F_2 be the features of f_1 and f_2 respectively. We define F to be the set of features both f_1 and f_2 have in common, where $F = F_1 \cap F_2$. f_1 and f_2 are *feature related* at a point x if $x \in F$. f_1 and f_2 can be positively or negatively related. Let $F^+ = F_1^+ \cap F_2^+$ where F_1^+ and F_2^+ are positive features of f_1 and f_2 respectively. Let $F^- = F_1^- \cap F_2^-$ where F_1^- and F_2^- are negative features of f_1 and f_2 respectively. Then f_1 and f_2 are positively related at x if $x \in F^+$ or $x \in F^-$. f_1 and f_2 are negatively related at x if $x \notin F^+$ and $x \notin F^-$ and $x \in F$.

The value of relationship score, τ, can help us decide whether two attributes are co-related. If the attributes are related, then their features can be represented as candidate explanations. Let p be the set of positive relations in F. Let n be the set of negative relations in F. Then, $\tau = \frac{|p|-|n|}{|F|}$

Our system builds on the implementation of the salient features by Chirigati *et al.* (2016) in their work on the Data Polygamy framework. Calculating salient features consists of three main steps: Pre processing, aggregation, and indexing. Each step in this process is implemented as a map reduce operation.

In the preprocessing step, the spatial and temporal attributes of the dataset are used to select the data. The spatial and temporal attributes form the key in the MapReduce job while the average of our attributes forms the value. In the aggregation step, the scalar functions are generated for the data. Each attribute in our dataset is represented by a different scalar function. Several scalar functions are calculated for

each attribute for each spatio temporal resolution e.g. the spatio temporal resolution of 'hour' and 'neighborhood' has a different scalar function compared to that for 'day' and 'ZIP code'. The next step consists of indexing the data. Indexing consists of creating a graph out of the aggregated data. Each node in the graph represents a point in the spatiotemporal domain. It may be convenient to think of this as a 3-D graph where two axes represents space and another represents time. A node is connected to another node if they are adjacent to each other in space(e.g. two neighborhoods next to each other) and time. The value of each node is the average attribute value. Due to the nature of the data, some points in the graph might be missing. The indexing step linearly interpolates the missing points in the graph. The 2-D scalar functions we calculated are finally used to calculate whether each node in our graph is a negative salient feature or positive salient feature.

After the salient features are calculated we need to measure how well they explain the data. We do this by finding the relationship score between the observation scalar function and the scalar function for the remaining attributes in our dataset. The non spatial explanation consists of our threshold values while the spatial explanation consists of the polygons represented by the co-related salient features.

Chapter 4

SPATIAL EXPLANATIONS USING HIERARCHICAL INTERVENTION

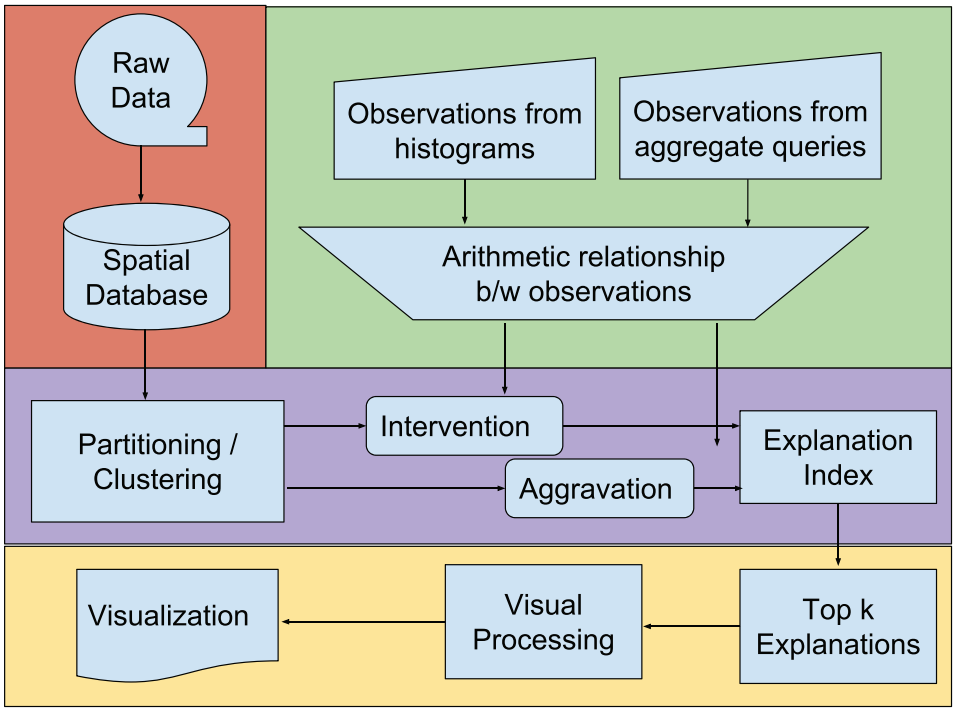

Figure 4.1: An outline for our system framework

4.1 Introduction

Hierarchical Intervention is an extension of the intervention approach. The idea behind hierarchical intervention is to improve our explanations by grouping together spatially colocated polygons in our candidate explanation. We divide the space into

a set of levels. The first level contains the entire space as one cluster, while the last level contains each zone or each point in each cluster.

If we look at a high-level overview of our method, it involves partitioning the data spatially into a hierarchy. The top level of the hierarchy consists of all the tuples in the data. The lowest level of the hierarchy contains each individual tuple. We use this hierarchy to perform aggravation and intervention. Each cluster in the hierarchy represents a spatial predicate. Finally, we compare all the clusters in every level of the hierarchy to rank explanations.

Algorithm 1 Algorithm for Hierarchical Intervention

```
1:  procedure EXPLAIN(tuples)
2:      input : tuples with spatial attribute
3:      output : ranked spatial explanations
4:      hierarchy ← Cluster(tuples)                          ▷ Step 1
5:      dag ← Create Dataframes from hierarchical levels     ▷ Step 2
6:      current_level ← Get last level from dag
7:      AggravationIntervention(current_level)
8:      current_level ← current_level − 1
9:      while current_level > 0 do
10:         AggravationIntervention(current_level)           ▷ Step 3
11:         ResolveOverlaps(current_level)                   ▷ Step 4
12:     end while
13:     explanations ← Rank(dag)                             ▷ Step 5
14:     return explantions
15: end procedure
```

Fig. 4.1 shows the system diagram for our solution which uses hierarchical interven-

tion. We take inputs in the form for aggregate queries. An arithmetic expression encapsulates the relationships between these queries. On the other hand, we have a spatial dataset. We use clustering/partitioning to create a hierarchy out of our spatial data. Depending on the hierarchy that we have created and our inputs, we perform aggravation and intervention. The results of aggravation and intervention are used to in a ranking system based on an explanation index. The explanation index measures the candidate explanations as a linear relationship between aggravation and intervention. How much each explanation approach is weighted in the explanation index is under the control of the data analyst. Finally, the top results are used for visualization. We have created a web-based GUI to display these kinds of explanations.

In order to get a better understanding of our spatial explanation approach, we will highlight our algorithm. We have designed our approach in the context of the programming paradigms provided by Apache Spark. The algorithm uses dataframes to store each level in the spatial hierarchy. The leaf nodes are populated using aggravation/intervention operations. The intervention/aggravation values of the non-leaf nodes are populated using the data from children nodes. Since our technique uses memoization in a directed acyclic graph, it can be categorized as a dynamic programming approach.

Fig. 4.2 shows our hierarchy in the form of a tree. Nodes with overlapping children are marked using a red dashed line. The children of these nodes represent clusters of tuples which have some common points of intersection. Since we want to build our graph bottom up, such overlaps present a concern. We do not want to use tuples multiple times in our calculations.

We represent each node in our hierarchy in a dataframe(Fig. 4.3). The dataframe contains the id of the node(*tid*), the id of the parent node(*parent*), intervention,

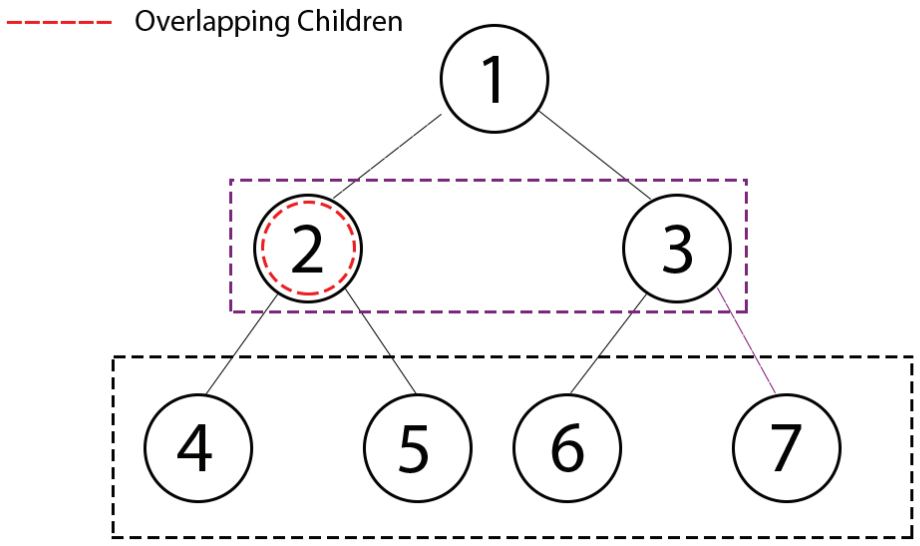

Figure 4.2: The hierarchy used by our algorithm

aggravation, aggregates of each attribute, and a column for intervention and aggravation for each aggregate query in our input. While dataframes look like traditional RDBMS tables, they have a few differences. Our solution exploits these differences to get more out of the system. In this case, we use the arbitrary number of columns in the dataframe to our advantage by representing the aggravation/intervention for each query in a separate column.

In order to build the DAG bottom up, we want to use the values of aggravation and intervention in the children nodes to build up. At the beginning of the algorithm, we keep a record of the aggregates for the entire dataset. As mentioned before, we also have aggregates for each node in our dataframe. We can use these two values to calculate aggravation or intervention. For example, if our aggregate query was SUM of an attribute and the node we are looking at consists of two child nodes, then the aggravation value would just be the sum of both the children. The intervention value

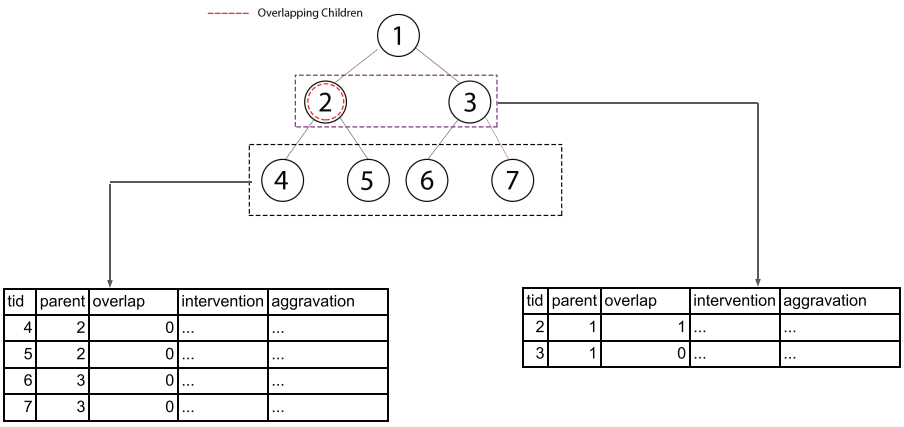

Figure 4.3: Dataframes represent each level in our hierarchy

would be the difference between the sum of the child nodes and the sum of the entire data. Using this approach we can build the explanations up using a left join on the parent and children dataframes.

Once we have built up our DAG one level, we still have tuples which contain incorrect data. These are the tuples with duplicates. Fig. 4.5 shows a representation of this scenario. In order to handle this case, we split our dataframe into two parts: one containing nodes with overlap bit, and one containing tuples without the overlap bit. The Dataframe with the tuples containing the overlap bit has the intervention and aggravation values recalculated. Finally, the split dataframes are merged together to give a final dataframe for the level in the hierarchy without and recounted values. This dataframe can be used for constructing the DAG up further iteratively till we reach the root node.

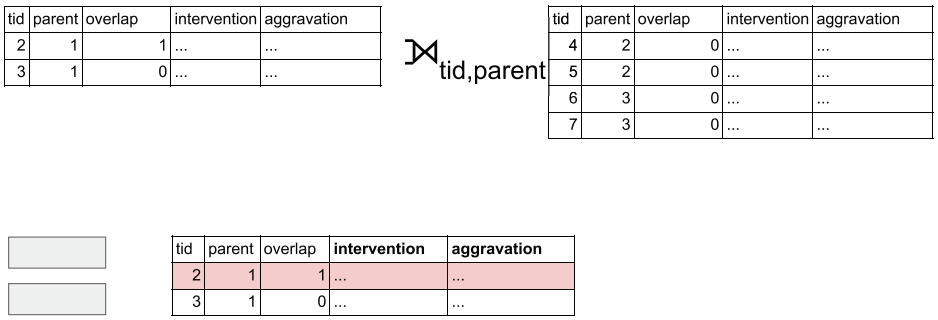

Figure 4.4: We use memoization to build DAG bottom up

4.2 Step1: Spatial Partitioning/Clustering

There are several approaches to cluster data. Each clustering approach has a large impact on the resulting explanations. In order to understand the issues with explanations using spatial clustering, we need to have an understanding of the clustering approaches.

4.2.1 R-Tree

An R-Tree is a data structure which is commonly used for spatial data indexing. The idea behind the R-Tree is similar to the idea behind a binary tree i.e. It is faster to query data if it is stored in the form of a hierarchy(Guttman, 1984). In an R Tree, this hierarchy exists in the form of Minimum Bounding Rectangles(MBR).

The top level of an R Tree consists of a set of MBRs which cover a large spatial area. Each MBR can now be subdivided into further MBRs which makes up the second

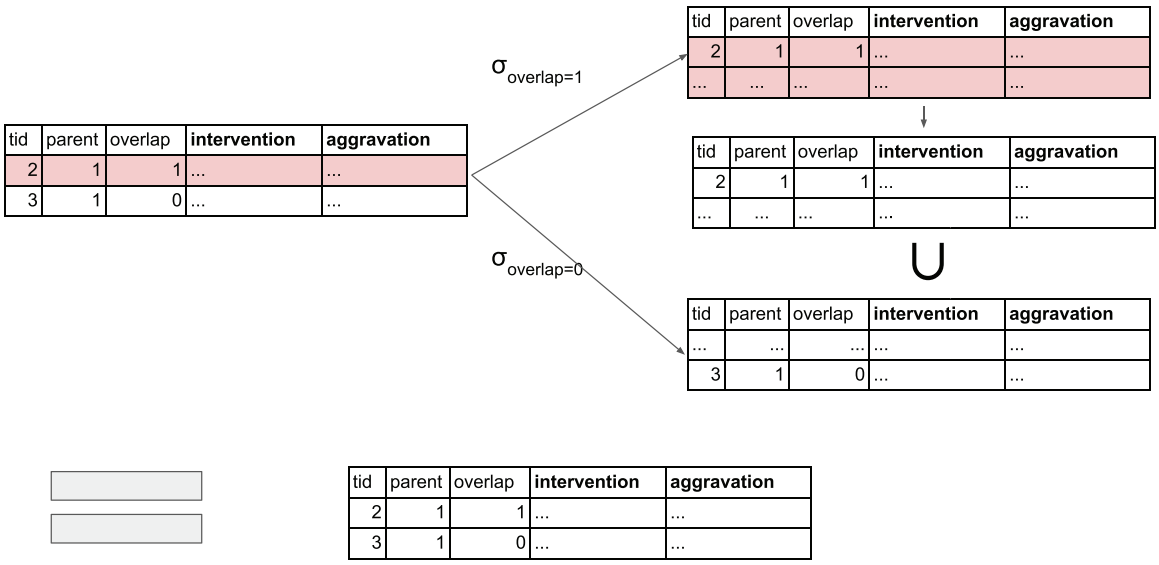

Figure 4.5: Recalculation is done for overlapping nodes

level of the tree and so forth. At the leaf nodes of the tree, each node consists of a single object in our underlying data.

There are three main considerations when building an R Tree: insertion, deletion, and search. In order to insert data, it is important to consider that the tree is balanced. A balanced tree results in faster search since the depth of the tree is reduced. The same thing needs to be taken into consideration with deletion. In order to insert an element, a heuristic is usually involved. The main objective when inserting into an R-Tree is to minimize the splitting and enlargement. If an element can be inserted into an existing MBR, then it is preferred. Otherwise, existing MBR's are either split or enlarged.

When elements from an R-Tree are deleted, the parent nodes are also taken into consideration. This is because deleting elements may result in a change in the parent MBR as well, unlike a B-Tree. B-Tree is an indexing structure inspired from a binary tree (Bayer and McCreight, 1970). An R-Tree maintains a minimum utilization constant into consideration when deleting. If a node in the R-Tree is below the minimum

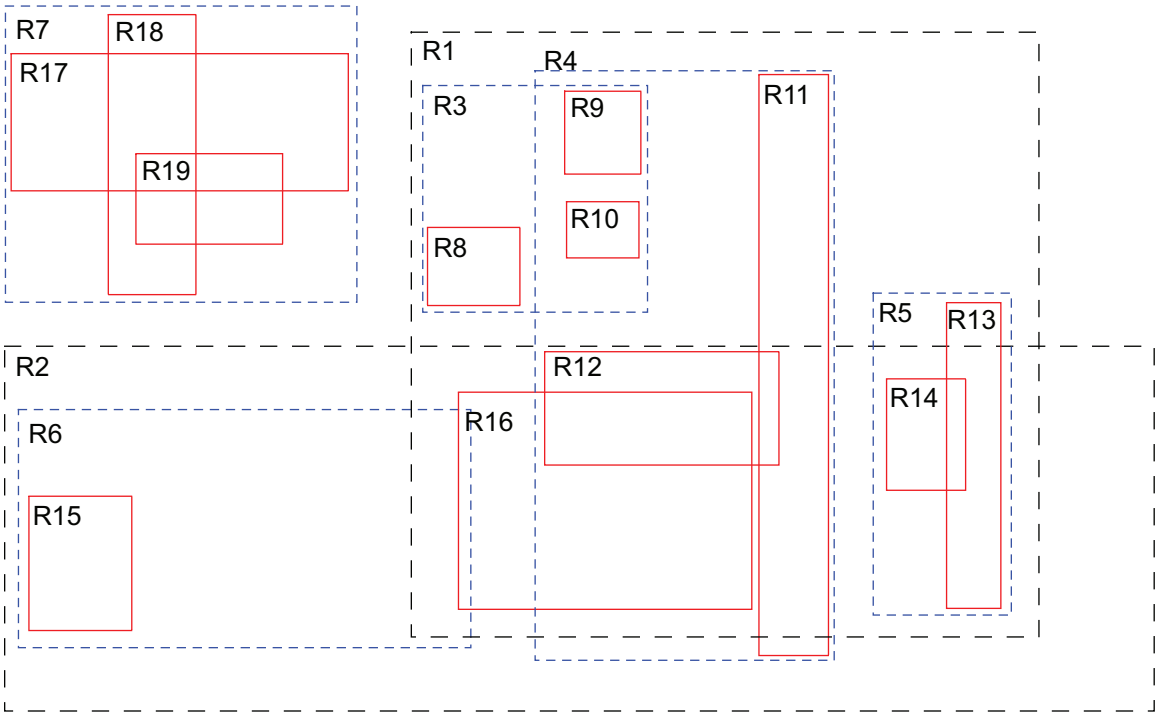

Figure 4.6: An illustration of R-Tree

utilization, the elements are reinserted.

Finally, the main purpose of constructing an R-Tree is to optimize search speed. If the branching factor of the R-Tree is λ and there are n elements that are indexed, then the search cost of the R-Tree is $\log_{\lambda} n$ assuming that the tree is balanced.

4.2.2 R-Tree*

R* Tree is a variation of R-Tree(Beckmann *et al.*, 1990). The objective of an R*-Tree is to minimize overlaps and coverage in an R-Tree. It also optimizes for margin and area. The idea behind R*-Tree which helps in achieving its objective is to use the perimeter of the MBR as a heuristic when splitting and creating R-Trees.

Unlike R-Trees, the insert operation of an R* Tree also incorporates deletion. When an element is added to an R-Tree, an existing MBR is either extended or split. How-

ever, ordering of the insertion in an R-Tree deforms existing MBRs and may lead to more coverage. The R* Tree solves this problem by removing elements from nodes and reinserting them. Splitting of MBRs is also based on perimeter. There are multiple ways to split an MBR, but if its split based on the least resulting perimeter as a heuristic, then this heuristic leads to better partitioning with less overlaps.
The search and delete operations of an R* Tree are the same as an R Tree.

4.2.3 K-means clustering/Voronoi partitioning

K means clustering is an algorithm designed to cluster a group of points (MacQueen *et al.*, 1967). As the name suggests, the user decides the number of clusters that he/she wants. K represents the number of clusters. The algorithm randomly selects k seed points. The seed points can also be selected using a heuristic to improve the clustering. The points closest to each of the seed points form clusters. The k means algorithm is iterative. Which means it does not end there. The centroids of the clusters are used as new seed points and the process is repeated until the centroids are constant. Algorithm 3 shows our version of the kmeans algorithm where the selectivity is used as a metric for judging the clusters.

4.2.4 Hierarchical Greedy Clustering

Hierarchical Greedy Clustering is a popular algorithm in data visualizations because of its speed (Agafonkin, 2016). The idea behind this algorithm is to randomly select points and create clusters around them.
Let D be our spatial dataset. Let P be a set of distinct polygons over our dataset D. We define a set of distinct clusters of polygons P_h. Let C be the set of centroids in P i.e. $\forall p_i \in P$, c_i is the centroid for p_i, where $c_i \in C$. We define a set of n levels, $L = \{l_1, l_2, ..., l_n\}$. We represent each centroid in C on a Cartesian plane where x_i is

the first dimension of the centroid c_i and y_i is the second dimension of the centroid c_i. Let x_{min}, x_{max} be the minimum and maximum value for x_i respectively. Let y_{min}, y_{max} be the minimum and maximum values of y_i respectively. There is a radius associated with each level in L. Let r_i be the radius for level l_i in L, We can define r_i in general as,

$$r_i = \frac{\sqrt{(y_{max} - y_{min})^2 + (x_{max} - x_{min})^2}}{2^{i-1}}$$

The set of clusters,G_i, for level l_i can be defined as $G_i = \{p_k | (x_k^2 + y_k^2 < r_i^2)\} \forall p_k \in P$. The algorithm for the implementation of this approach is covered in Section 4.4.1.

Greedy Hierarchical Clustering turned out to have a drawback when we use it for explanations. It selects seed points randomly. This means that in each level, the clusters can be highly imbalanced. We compared this approach to K means(Voronoi partitioning)(Hartigan and Wong, 1979; Aurenhammer and Klein, 2000), R Tree(Guttman, 1984) and R* Tree(Beckmann *et al.*, 1990).

The main idea behind the hierarchical intervention approach, regardless of the partitioning algorithm, is to help us in increasing the domain for our degree of explanation. Since intervention is a subset of this approach, the domain is at least as large as that for intervention. After we have partitioned our data, each partition can be used as a predicate for aggravation and/or intervention.

4.2.5 Step2: Hierarchical Dataframes Construction

In our implementation of the system, we use the programming paradigm provided by Apache Spark. This involves using dataframes for processing. Dataframes resemble tables in a traditional RDMS. Each cluster in our spatial hierarchy is represented as a node in a tree. Each level of the tree is encapsulated as a dataframe. Each node has an ID associated with it. A tuple in our dataframe contains the ID of the node, the ID of the parent node, a column representing the overlap bit, columns representing

the aggregates of the attributes, columns representing the aggravation/intervention for each query in our observation, and a column for the final values of aggravation/intervention after evaluating the arithmetic operation that represents our observation. The spatial hierarchy that we construct in step 1 of our algorithm is already a DAG since it takes the form of a tree. But for the next few steps, the DAG that we use is based on the dataframes. Instead of calculating aggravation and intervention for each tuple in our set of dataframes, we can use the values calculated in the leaf dataframe to build up the explanations.

4.2.6 Step3: Building up the DAG

Aggravation and Intervention are very expensive operations. In order to create a system for explanations which performs well, we need to reuse calculations. Thats the main reason we constructed our DAG in the first place. The intervention/aggravation values stored in one level of our DAG can be used one level above. We use the associative property of aggregate functions like sum and count to build our DAG bottom up. We use the group and join operations to implement this. The tuples at one level are grouped based on the parent id and left joined with the tuples at the parent level based on the node id and parent id of the child and parent dataframes respectively.

Our system is designed to work with arbitrary clustering algorithms. Some of the clustering algorithms like K-Means have no overlapping tuples in the clusters with others like R-Tree have partitions with overlaps. In our spatial hierarchy if a node has child nodes with overlapping points, the overlap bit for that node is set to *true*. Using our build up approach so far, these nodes contain inaccurate values of intervention and aggravation. Before building the DAG further up these values need to be fixed.

4.2.7 Step4: Overlap Resolution

Depending of the type of spatial partitioning, some nodes in our DAG have incorrect values of aggravation/intervention stored in the dataframe. These are nodes where the overlap bit is *true*. Since we are using dataframes, we cannot update individual tuples. Instead, we create two dataframes out of the original. One containing tuples where the overlap bit is true and one containing the rest of the tuples. The aggravation/intervention and aggregates for the first dataframe are recalculated. Finally both the dataframes are combined using union and the old dataframe is replaced with the new one in the DAG. Using this sequence of operations, we can reduce the number of calculations. Since our dataframe now contains accurate information, we can resume building the DAG further up.

4.3 Step5: Ranking Explanations

In order to get the most out of Hierarchical Intervention, we introduce two new metrics: Intensity and Influence.

4.3.1 Intensity

We define intensity as a metric which measures the standalone value of the explanation. The relevance metric borrows a lot from our definition of aggravation. It might be convenient to think of a web search engine when we are looking at the intensity metric. When we use a search engine, we provide a search term as a query. The search engine looks at all the pages in its database and returns the results in order of relevance. The top results in the search engine may not have a significant effect on the entire web if they were to be removed. However, the top result in the search engine has the highest relation to the data. For example, the top result for a

search engine which uses tf-idf might be a page containing the highest frequency of the search term(Robertson, 2004).

Let D be our dataset. Let ϕ be our candidate explanation. Let R be the function which maps our dataset to the value of our observation. Then we can define intensity as,

$$intensity = |R(\sigma(D)) - R(\sigma_{\phi}(D))|$$

4.3.2 Influence

We define influence as a metric which measures the value of the explanation compared to the entire dataset. The influence metric borrows from our definition of intervention. The influence metric measures how much the observation would be affected if we remove the data related to our explanation(the *influence* of our explanation on the observation). We can use the analogy of the search engine again here. One of the earliest algorithms used by Google to rank webpages used links to other pages(Brin and Page, 1998). The page which was linked the most on a variety of websites was ranked higher. If you remove a highly relevant page, many other pages might not exist. Influence uses the same principle.

Let D be our dataset. Let ϕ be our candidate explanation. Let R be the function which maps our dataset to the value of our observation. Then we can define relevance as,

$$influence = |R(\sigma(D)) - R(\sigma_{\neg\phi}(D))|$$

The greater the value of influence, the more its impact on the observation.

Our evaluation metrics suggest that influence and intensity explain data in different ways. Explanations with higher influence tend to give predicates which cover a larger area while explanations with higher intensity tend to give predicates which cover a small area. We can balance out these two metrics according to the preferences of the

observer. We define the *explanation index*, ϵ as:

$$\epsilon = \alpha \times influence + (1 - \alpha) \times intensity$$

α is the *explanation coefficient*. It is a variable whose value is decided by the user. Explanations given by Hierarchical Intervention can be ranked on the basis of the explanation index.

4.4 Implementation

We created an implementation of the our system as a proof of concept. Our system uses a web based GUI for visualization while the backend processing is done using Spark(Shanahan and Dai, 2015). Fig. 4.1 shows the archicture diagram of the entiresystem. We borrowed from the MapReduce implementation of Salient Features from the Data Polygamy Framework(Chirigati *et al.*, 2016). There are three main stages in our system pipeline: Preprocessing, User Input, and Explanation Visualization. The preprocessing stage is different for each approach to explanation. The user input stage can be divided into two further parts: The user observation, and the type of explanation the user is interested in. An example of the explanation approach would be intervention. The user observation can be one of the attributes the user wants to observe e.g. a mathematical expression spanning several aggregate queries. The aggravation approach that we use makes several optimizations to the naive approach one may come up with. A naive approach may consider each candidate explanation and calculate the degree of explanation by aggravation for each one. However, using such an approach would take a long time. If the attribute which we are looking into for explanation has m possible values, it would take $O(m)$ queries to come up with top explanations. If we have r rows in our dataset and each observation query has a linear time complexity, our total time complexity would be $O(rm)$. We can

reduce the time complexity by a large factor if we make each candidate explanation cover a larger range of the dataset.

As mentioned in Section 3.2.1, we use bucketing to reduce the number of permutations. We use the standard deviation, σ, of each attribute to separate values which are near the average from the others. We can get the minimum and standard deviation of each attribute over a single iteration of our table in $O(r)$ time. Since the bucket for each value of the attribute can be calculated in constant time, we create a new table containing the bucket value of the attribute in $O(r)$ time.

Since we bucketed our data, the number of candidate explanations have been reduced to $O(\binom{m}{\sigma})$. However, there is still room for improvement. Our observation is always an aggregate query. If we group on the explanation attributes, we can still get all of the corresponding values of the observation in one iteration of the data, making our time complexity $O(r)$.

Another component of our system is spatial data. Our system is designed to handle spatial observations and explanations. When given a spatial observation, our system filters out the data with the spatial predicate given in the observation. The remaining data is used just like any other explanation task. Spatial Explanations, on the other hand, require a bit more work. Given a set of polygons, doing a spatial join with the data is a very expensive process. Our system performs a spatial join once and assigns each tuple an ID based on which polygon it belongs to. The ID can now be used like a regular attribute for explanation.

Intervention is another approach that our system uses. Some of the work we did to optimize aggravation cannot be used to improve the time complexity for intervention. This is because intervention looks at the effect of removing data. While we could use a single iteration of the data for aggravation. We cannot do so for intervention because removing one portion of the data completely changes what the observation

looks like. We can still use bucketing to reduce the number of combinations.

4.4.1 Hierarchical Intervention

Hierarchical intervention is an improvement on top of the intervention approach. The main idea behind hierarchical intervention is to group spatially colocated points/polygons together in an attempt to improve explanations. We use a partitioning algorithm to create clusters. The base case of our algorithm consists of clusters with a single node each. This makes the results of intervention a subset of the results of hierarchical clustering. Algorithm 2 highlights the algorithm that we used for our implementation of greedy hierarchical clustering. Note that each level in the hierarchy is a set cover of all the nodes.

Greedy Hierarchical Clustering is an iterative algorithm. The base case(Algorithm. 2;line 27) is when we reach the lowest level in the hierarchy where we have individual points or zones. We start off with the highest level in the hierarchy(Algorithm. 2;line 4). We select random points and form clusters around them(Algorithm. 2;line 16). In the following iteration we reduce the size of the cluster by a constant factor(Algorithm. 2;line 8) and repeat until we reach the base case.

While greedy hierarchical clustering helps in increasing the domain for the degree of explanation by intervention, the random selection of seed points makes the rate of change of influence and intensity(Section 4.3) over the levels of hierarchy very volatile. Our point is illustrated by fig. 5.2. The red line represents influence while the blue line represents intensity. One would expect the curves to be logarithmic. However, they more closely resemble a sinusoidal function.

Algorithm 2 Algorithm for Hierarchical clustering

```
1:  procedure CLUSTER(max_radius, points)
2:      input : max_radius, points
3:      output : hierarchy_of_clusters
4:      level ← 0
5:      isBaseCase ← false
6:      while isBaseCase = false do
7:          level ← level + 1
8:          levelRadius ← max_radius / 2.0^level
9:          hierarchy ← ∅
10:         unusedPoints ← ∅
11:         for point ← points do
12:             unusedPoints ← unusedPoint ∪ point
13:         end for
14:         while unusedPoints.size > 0 do
15:             cluster ← ∅
16:             centroid ←random point from points
17:             for point ← unusedPoints do
18:                 if point inside levelRadius then
19:                     cluster ← cluster ∪ point
20:                     unusedPoint ← unusedPoints − point
21:                 end if
22:             end for
23:             hierarchy ← hierarchy ∪ cluster
24:         end while
```

25: $hierarchy\_of\_clusters \leftarrow hierarchy\_of\_clusters \cup \{level, hierarchy\}$

26: $isBaseCase \leftarrow true$

27: **for** $cluster \leftarrow heirarchy$ **do**

28: **if** $cluster.size > 1$ **then**

29: $isBaseCase \leftarrow false$

30: **end if**

31: **end for**

32: **end while**

33: **return** $hierarchy\_of\_clusters$

34: **end procedure**

In order to try to counter this problem, we used k means clustering. Algorithm 3 highlights our implementation of k-means clustering centered around our problem. The algorithm is designed to find the clusters based on a heuristic of giving more preference to balanced clusters i.e. each cluster has similar number of points.

Our algorithm for K-Means clustering is designed to handle data which is divided into groups of zones. The results of K-means clustering are dependent on the choice of seed tuples. In our algorithm, we allow a number of iterations of clustering to allow the best selection of seed tuples based on selectivity(Algorithm. 3; line 9). For each run, we select k seed tuples randomly. The Euclidean distance of each tuple in our dataset from the seed tuples is used to determine which cluster the tuple belongs to. When we have data divided into zones, each zone can represent a variable number of tuples. The Euclidean distance is simply chosen because we want each Voronoi partition to represent points closest to its seed. The centroid when considering each zone as a single entity is different from a centroid where a weight is assigned to each zone. Our algorithm takes this into account and calculates the centroid as an

Algorithm 3 Explanation Aware K means

```
1: procedure CLUSTER(points, k, number_of_runs)
2:     cluster_possibilities ← ∅
3:     for i ← 0, number_of_runs do
4:         cluster_possibilities ← cluster_possibilities ∪ ClusterOnce(points, k)
5:         i ← i + 1
6:     end for
7:     best_clustering ← ∅
8:     best_distance ← ∞
9:     for possibility ← cluster_possibilities do
10:        cluster ← GetCluster(possibility, points)
11:        k_distance ← GetKMeansDistance(cluster)
12:        if k_distance < best_distance then
13:            best_distance ← k_distance
14:            best_clustering ← cluster
15:        end if
16:    end for
17:    return best_clustering
18: end procedure
19: procedure GETKMEANSDISTANCE(clusters)
20:    max_distance ← −∞
21:    min_distance ← ∞
22:    centroids ← centroids of clusters
23:    for centroid ← centroids do
24:        related_cluster ← cluster from centroid
```

```
25:         cluster_weight ← GetClusterWeight(related_cluster)
26:         if cluster_weight > max_distance then
27:             max_distance ← cluster_weight
28:         end if
29:         if cluster_weight < min_distance then
30:             max_distance ← cluster_weight
31:         end if
32:     end for
33:     return |max_distance − min_distance|
34: end procedure
35: procedure GetClusterWeight(points)
36:     count ← 0
37:     for point ← points do
38:         count ← count + 1
39:     end for
40:     return count
41: end procedure
42: procedure GetCluster(centroids, points)
43:     clusters ← ∅
44:     for centroid ← centroids do
45:         clusters ← clusters ∪ {centroid, {}}
46:     end for
47:     for point ← points do
48:         closest_centroid ← nil
49:         for centroid ← centroids do
```

```
            if closest_centroid = nil then
                closest_centroid = centroid
            else if distance from centroid < distance from closest_centroid then
                closest_centroid ← centroid
            end if
        end for
        Add point to cluster where the centroid is closest_centroid
    end for
    return clusters
end procedure
procedure ClusterOnce(points, k)
    centroids ← ∅
    selected_points = ∅
    for i ← 1, k do
        random_point ← random point from points
        while random_point ∈ selected_points do
            random_point ← random point from points
        end while
        selected_points ← selected_points ∪ random_point
        centroids ← centroids ∪ random_point
        i ← i + 1
    end for
    while true do
        clusters ← GetCluster(centroids, points)
```

```
74:         new_centroids ← ∅
75:         for centroid ← centroids do
76:             cluster ← cluster for centroid
77:             sum_x ← 0
78:             sum_y ← 0
79:             num_values ← 0
80:             for point ← cluster do
81:                 sum_x ← sum_x + point.x
82:                 sum_y ← sum_y + point.y
83:                 num_values ← num_values + 1
84:             end for
85:             avg_point ← (sum_x/num_values, sum_y/num_values)
86:             closest_point ← nil
87:             for point ← cluster do
88:                 if closest_point = nil∨ distance of point from avg_point < dis-
    tance from closest_point then
89:                     closest_point ← point
90:                 end if
91:             end for
92:             new_centroids ← new_centroids ∪ closest_point
93:         end for
94:         centroids_same ← true
95:         for new_centroid ← new_centroids do
96:             centroid_found ← false
97:             for old_centroid ← centroids do
```

```
98:                 if old_centroid = new_centroid then
99:                     centroid_found ← true
100:                    break
101:                end if
102:            end for
103:            if centroid_found = false then
104:                centroid_same = false
105:                break
106:            end if
107:        end for
108:        if centroids_same then
109:            break
110:        else
111:            centroids ← new_centroids
112:        end if
113:    end while
114:    return centroids
115: end procedure
```

average based on the selectivity of each tuple in the cluster(Algorithm. 3; line 80). For each iteration we select new seed points as the tuples closest to the calculated centroids(Algorithm. 3; line 95). This process is repeated several times. The heuristic that we use to select the best seed tuples is mentioned in the $GetKMeansDistance$ procedure of our algorithm(Algorithm. 3; line 19). Here the heuristic is selected such that a small absolute difference between the selectivity of its clusters would be considered as a good clustering.

K-Means clustering gives a much better result than greedy hierarchical clustering for our problem. However, we also used other spatial partitioning approaches like R-Tree and R*-Tree for comparison. R*-Tree appears to be the partitioning algorithm which gives us the best results according to our experiments (Section 5). Fig. 5.1 shows the comparison between K-Means, R-Tree, and R*-Tree.

Once we have our hierarchy set up, we still have to use it to find an explanation. If we had n nodes, the number of clusters we have for our hierarchy is $n \log n$. This increases the number of combinations for explanations significantly. However, we can use some optimizations to save us some time complexity. Our approach uses memoization of explanations for each spatial node to calculate the value of the degree of explanation. We first calculate the statistics(sum, count) for each node individually and then subtract it from the statistics for the attribute we are observing. For example, let us consider a cluster, c, containing two spatial nodes a and b. We are interested in the average tip percentage for taxi trips. The degree of candidate explanation by intervention would be the average tip percentage when we remove a and b from our dataset. Our system would memoize the values for the sum and count of tip percentage for a, b as sum_a, $count_a$, sum_b and $count_b$. It will also calculate the sum and count for tip percentage for the entire dataset. When we need our system to find the degree of explanation for cluster c. The formula for average is simply $\frac{sum}{count}$. Since our system calculated all the stats in one query, it does not need to perform additional queries to find the degree. It can simply use $\frac{sum - sum_a - sum_b}{count - count_a - count_b}$ to calculate the average for intervention.

4.5 Interface

We created an interface for the system that we have proposed. This interface is web based. It consists of a backend server and a front end for visualizations. The Backend

is responsible for querying the Apache Spark daemon with relevant queries. The front end allows the user to define observations and visualize explanations. The front end interface makes use of Mapbox to display the map, Deck.gl to create a visualization overlay on the map. The overlay can be points or polygons. In order to handle state changes in the front end, we used React.js front end library. We also show histograms related to the data. For displaying these histograms, the Baidu eCharts library was used. Since the front end was programmed in ES6, webpack was used for transpilation(Webpack, 2018). Fig. 4.1 shows us a flow diagram for our system.

In order for the users to easily form observations, our frontend provides them with a series of histograms. Each bar on a histogram represents an aggregate query. Instead of writing queries, the user can easily use the bars in the histograms to be used as variables. These variables can then be used in an arithmetic expression representing the users observations. The user can also define custom variables in the form of aggregate SQL queries manually using the interface.

The back end of the system is an Apache Spark application which calls the Spark daemon for requests of aggravation, intervention, and hierarchical intervention. It must be noted, however, that there are some operations that the backend performs which are not executed as Spark tasks. There operations are: R-Tree and R*-Tree creation. The framework that we have designed requires us to parse and alter queries at multiple points in the pipeline. In order to avoid redesigning the wheel, we use the parser implementation provided by PrestoDB(Facebook, 2018a).

To allow the front end to communicate with the backend, there is a middleware application. The job of the middleware is to take the input from the front end and execute our backend application with the appropriate function and input parameters. Our middleware is designed using Node js(Tilkov and Vinoski, 2010; Cantelon *et al.*, 2017). It takes aggregate queries, selectivity range and the number of explanations

as input and sends it to the backend application. The results are then sent to the frontend.

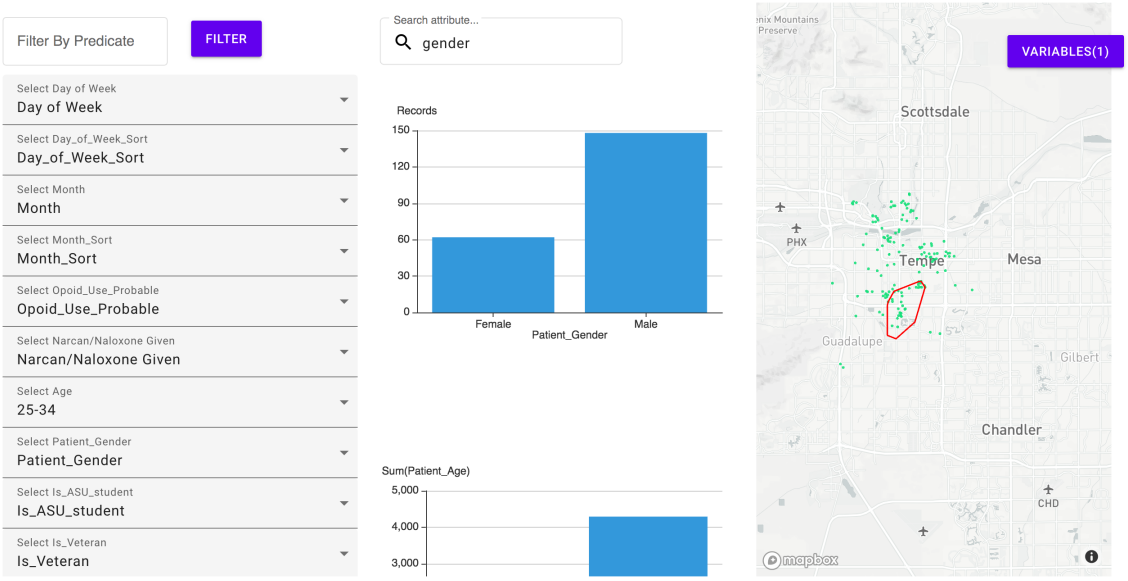

Figure 4.7: The web interface for our solution

Fig. 4.7 shows the interface that we have created for our solution. The users may add variables from the histograms or they may add them from their own sql queries. The interface allows the user to filter by attribute before viewing the histograms.

In order to give a better idea of how the GUI works, we will use the Tempe City Opioid Abuse Probable EMS Call data to show a use case (City of Tempe, 2018). This data consists of emergency calls related to opioid related cases in the city of Tempe. The location of the incidents and attributes about the subjects such as gender, veteran status, and whether the subject is a student, are recorded in this data.

Before starting the analysis. The GUI allows the user to filter the data based on its attributes. Fig. 4.8 shows the interface for prefiltering data.

Fig. 4.9 shows how the user can add bars on a histograms as observations. The GUI produces histograms based on the filtered data. In order to facilitate users who do

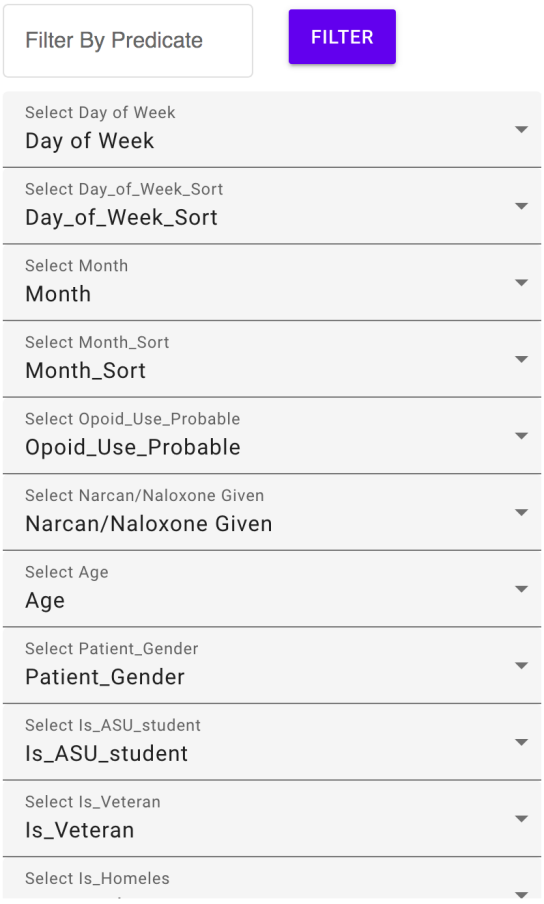

Figure 4.8: The filter interface for the GUI

not want to use a query language like SQL, the GUI allows them to add bars on the histograms as variables. Under the hood each bar is represented as an aggragate SQL query. The GUI also allows the user to add their own SQL if they want to. The user can also choose to view the scatterplot or the heatmap for the data represented by each bar.

Fig. 4.10 shows the user interface for specifying various parameters for the calculations. The user can specify the relationship between variables here.

Finally, Fig. 4.11 shows the map visualization of the explanations provided by the

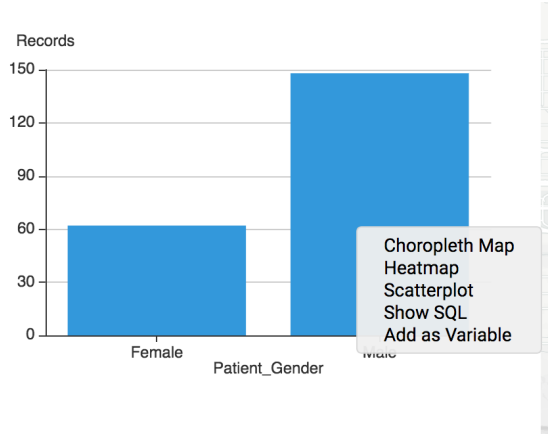

Figure 4.9: The histogram interface for the GUI

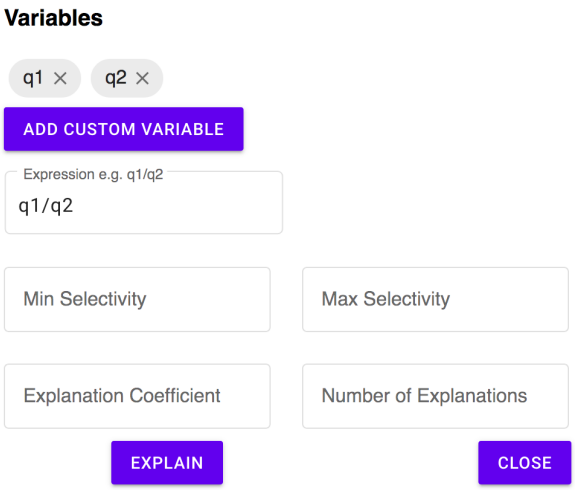

Figure 4.10: The parameters interface for the GUI

system.

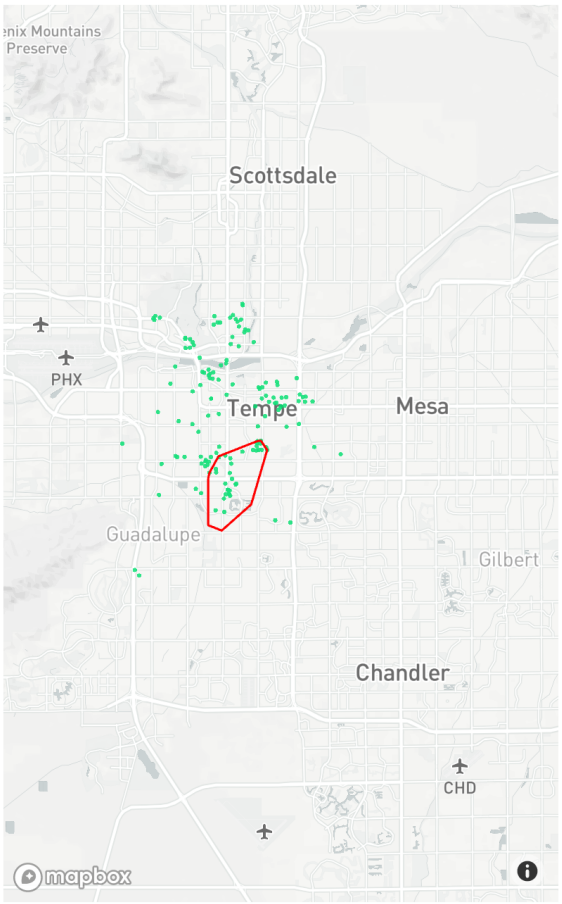

Figure 4.11: The map visualization interface for the GUI

Chapter 5

EXPERIMENTS/EVALUATION

The main contribution of this thesis is to introduce a new approach for explanations(Hierarchical Intervention). We evaluate this new approach in a number of different ways. We perform qualitative evaluation and speed/scalability evaluation.
In order to compare the different clustering techniques that we use in hierarchical intervention, we did a couple of experiments. Our solution relies on the fact that the higher the influence or intensity are a measure of the value of our explanations. We used a number of observations on the NYC TLC data to see how each clustering technique fares against the others.

Experimental Setup. In order to compare the different spatial partitioning techniques, we set up an environment for experiments. Since the quality of the explanations does not depend on the speed and scale of the system, these experiments were performed on a single node machine with an i7 6400 3.5 GHz CPU and 8GB of RAM. The dataset used for this experiment was the NYC TLC data for January 2016. This data contains 10 million trips.

We found that R-Tree and R* Tree partitioning is usually more stable than K Means or Greedy Clustering. The stability of the clustering technique is referring to its monotonicity of the explanation as a function of the number of clusters. Fig. 5.2 shows the influence against the number of clusters where the observation is the average passenger count. The influence and intensity are closer to a sinusoidal curve than a exponential or logarithmic curve. Fig. 5.1 shows another observation and a plot of the influence of R-Tree and R*-Tree as well. The influence as a function of the number of clusters is a lot more monotonic here. Furthermore, R* Tree has a higher

influence on average compared to other approaches making it a lot more suitable for explanations. The qualitative evaluation that we define is based on K-Folds cross validation. The reason for selecting these evaluations is to measure how the top explanations provided by each approach fair as stand alone parts of data as well as their effects on the entire dataset as a whole. Other than qualitative analysis we look at the speed and scalability of each approach for performance evaluation.

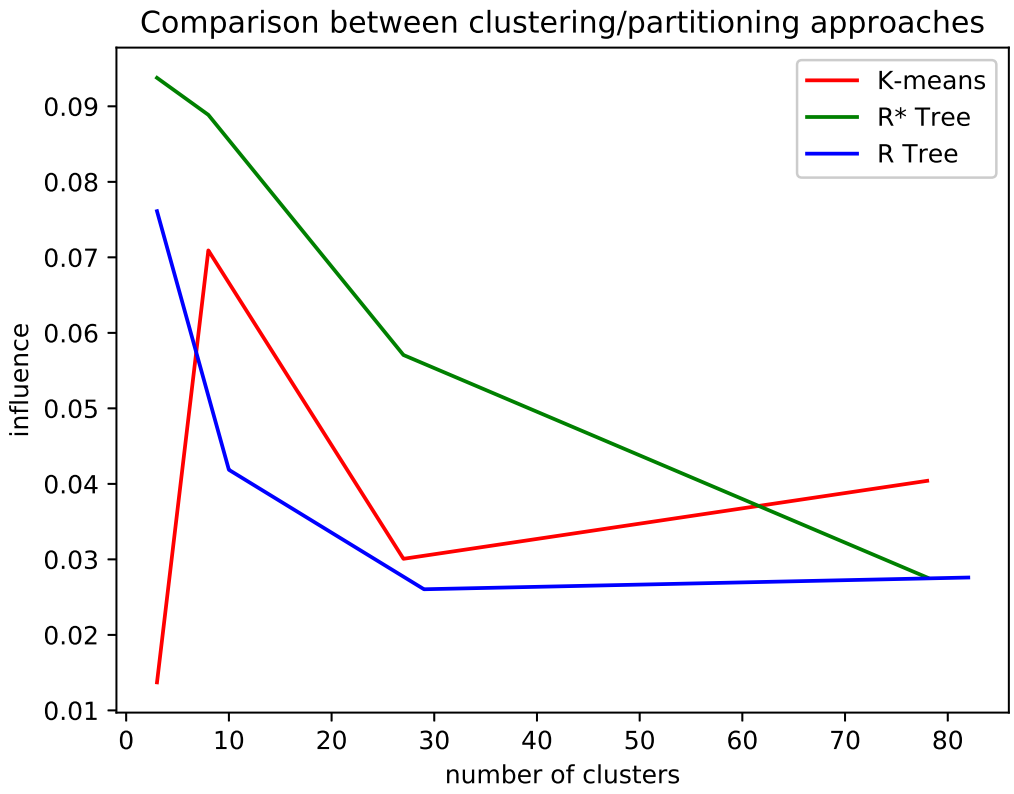

Figure 5.1: Comparison of Influence against number of clusters for K-Means, R-Tree and R*-Tree

Influence Comparison.

Fig. 5.3 shows the comparison between the influence of different approaches. Salient features and Aggravation have a very low influence compared to Intervention and

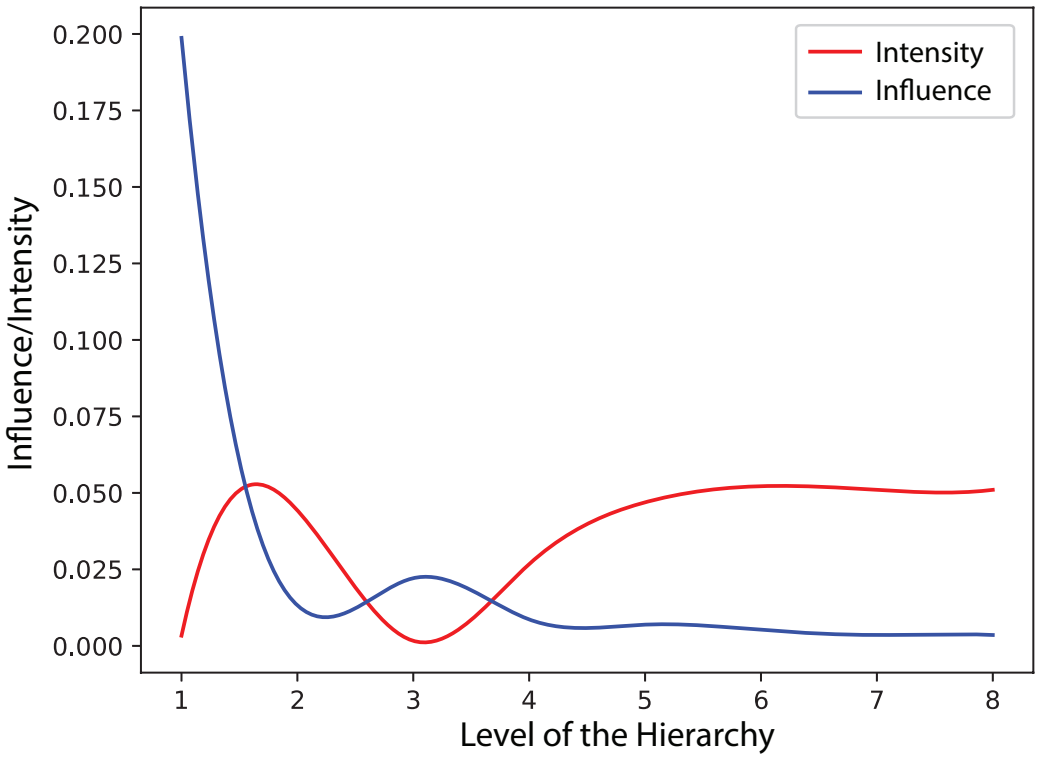

Figure 5.2: Comparison of Influence and Intensity against level of hierarchy with average passenger count as observation

hierarchical intervention. This is because the explanations provided by those approaches cover a very small portion of the entire dataset. This makes removing the associated data have a negligible impact on the entire data.

5.1 K-Folds cross validation

In order to qualitatively assess the solutions, we use K-Folds cross validation(Refaeilzadeh *et al.*, 2009). Since we do not have ground truth for our explanations, we have to make a compromise in terms of estimating how good our solution is. Our assumption here is that in general, a good explanation would stand out on

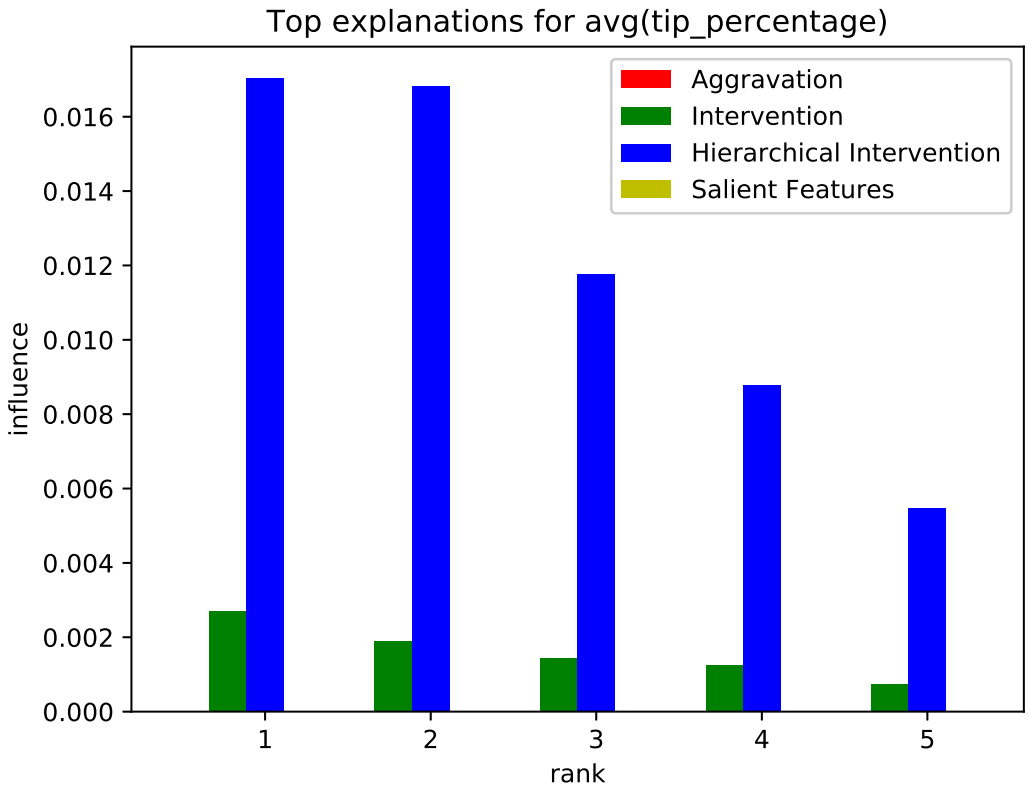

Figure 5.3: Influence for Spatial explanation for average tip percentage(Lower is better)

unseen data i.e. given a good explanation on part of the data, it would give similar results on unseen parts of the data. This is the same assumption that was made by Chirigati *et al.* (2016) in their evaluation of the Data Polygamy framework. We split the data into k parts. The data was divides by the month of the year. This is because there are specific observable patterns in each month, e.g. weekends have less density of trips than the weekdays for the dataset we evaluated. One of the parts is used to find the explanation(the training data). The rest of the data is the test data. We also calculate an explanation of the test data. The relative distance between the explanation index of the test data and the explanation index of the explanation provided by

the training data when applied to the test data is used as a measure of evaluation. The reason for using K-Folds evaluation in this way is because there is a lack of data with the ground truth values. We can see similar approaches for evaluation used in the works that our approach has been inspired from. The evaluation in the work by Roy and Suciu (2014) is less relaxed because they only look at the magnitude of intervention for evaluation, whereas, Chirigati *et al.* (2016) also use the NYC TLC data for evaluation using the assumption that the data for 2011 and 2012 have the same pattern. In order to measure the quality of the explanations provided by hierarchical intervention we compare it with intervention and aggravation.

There is a reason for applying the explanation provided by the training data to the test data. The observation can be anything. The size of the training and test data is different depending on k. The observation can depend on the size of the data. This adds a bias when the explanation index for the training data is compared with the explanation index of the test data. Thus, we must apply the explanation provided by the training data on the test data and compare it with the explanation when we apply the same approach to the test data.

Experimental Setup. We performed the evaluation on the first 5 months of data from the NYC yellow cab dataset from 2016. The data spanned 50 million trips. The k for our evaluation was 5. This value was chosen after a series of experiments. We wanted the size of each partition to be large enough to support our generalization assumption. If the solution space for evaluation is small, it leads to inconsistent results. Each month represented a partition for k-folds evaluation. According to our analysis, we found that the approach works very well with low values of alpha. Fig. 5.4 shows some of the results of our evaluation for a number of observations. The lower the value of the relative distance, the better the results.

The relative distance in the K Folds cross validation represents how much the ex-

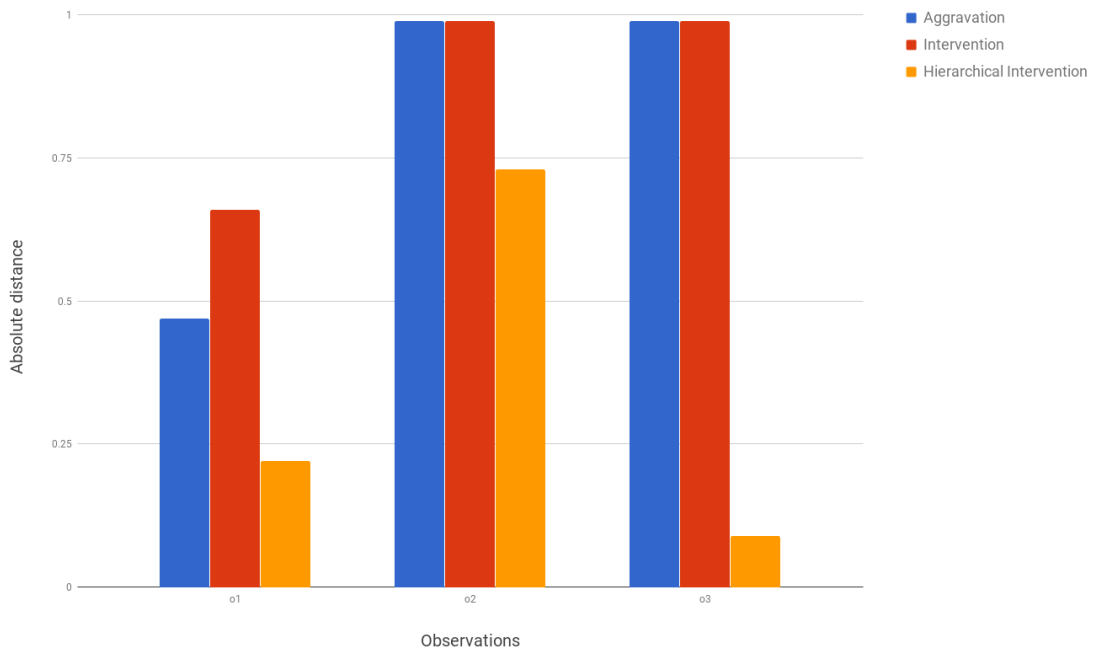

Figure 5.4: The result for k-folds evaluation shows good results when the approach is used in favor of influence(lower is better) for observations represented by Queries 5.1, 5.2, and 5.3

planation impact changes when we look at the explanations in the context of unseen data. A relative distance of zero represents that the explanation has the same impact on unseen data as if we had seen the data. A relative distance of one would represent that the top explanation for the unseen data has nothing in common with the explanation of seen data. Let e_{test} be the explanation index for the test set. Let e_{train} be the explanation index for the training set. Then,

$$relative\_distance = \frac{|e_{test} - e_{train}|}{|e_{test} + e_{train}|}$$

The relative distance metric was chosen because of its nature of showing a steep decline when two solutions do not match. It is also ideal because it caters to the

different types of observations that a user can specify. Since the observations in our system are defined as an arithmetic operation over aggregate queries, they do not have bounds on their values. Thus, it makes relative distance a useful metric for measuring similarity.

Here are the observations that we show in Fig. 5.4:

```
Select count(*) as q1 from data where payment_type=1;
Select count(*) as q2 from data where payment_type=2;
Observation = q1/q2
```

Query 5.1: o1 for Fig. 5.4

```
Select AVG(trip_distance) as q1 where passenger_count=4;
Observation = q1
```

Query 5.2: o2 for Fig. 5.4

```
Select count(*) as q1 from data where vendorid=1;
Select count(*) as q2 from data where vendorid=2;
Observation = q1/q2
```

Query 5.3: o3 for Fig. 5.4

5.2 Precision and Recall

Since it is very difficult to find datasets with the ground truth we used synthetic datasets to evaluate our solution(Maciejewski *et al.*, 2009). The synthetic dataset is fabricated using real data from Indiana Public Health Emergency Surveillance System. This data contains information about Patients in Indiana and their health issues over time and location. We can calculate precision and recall using this data(Powers, 2011). The data is synthesized in such a way that it shows similar patterns to the actual data. Synthetic data related to outbreaks of certain diseases is synthesized into

this data. In order to evaluate our system, we used the temporal component of the outbreak and the type of disease as our observation. To be specific, the synthetic data consisted of an outbreak of Gastrointestinal infection in July. Our observation was the ratio of the number of gastrointestinal infection incidents between June and July when there's an infection against the number of gastrointestinal infection incidents between June and July in synthesized data where there wasn't an outbreak. Precision and Recall was calculated by comparing the areas of the top ten explanations related to the ground truth with the approximate area covered by the outbreak data. We cannot use the individual points to calculate precision and recall because there is a lot of data unrelated to the outbreak which occurs in proximity. To put things into context, each instance of the synthesized data contains 2 million records. The data related to the outbreak consists of a few dozen records. Fig. 5.5 shows a heatmap of all the points in the synthesized data. The outbreak for this particular instance only affects a small part of Indiana to the North West just below Lake Michigan. Fig. 5.6 shows the points representing the outbreak as well as some of the top polygons returned using Hierarchical Intervention. While our system is designed to work fast on partitioning techniques with no overlaps like K-Means and Greedy Hierarchical Clustering. Some of the clustering apporaches that we use which give better results do have overlaps. Thus we used R*-Tree partitioning in our example. The candidate explanations provided by this partitioning techniques can have spatial overlaps.

Let E be the area represented by our polygons. Let O be the area represented by the Outbreak points. Then,

$$Precision = \frac{E \cap O}{E} = 0.31$$

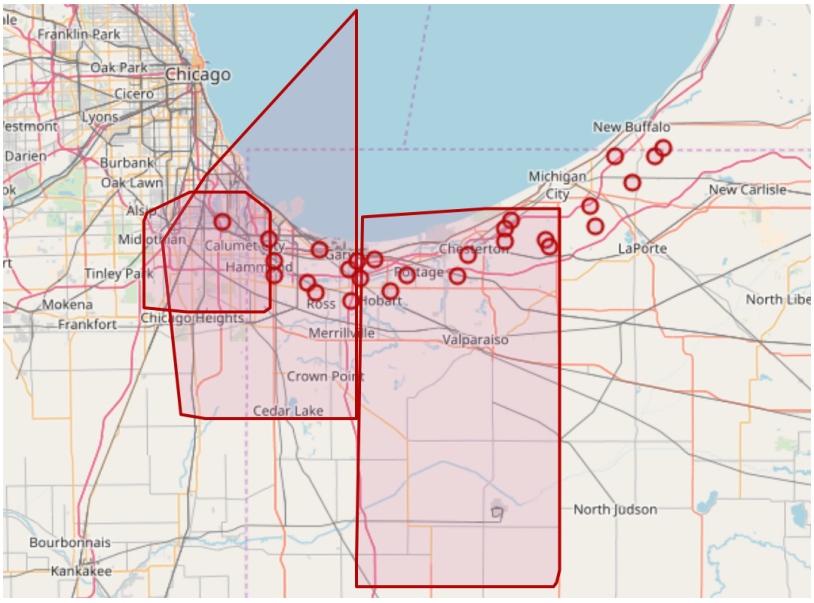

Figure 5.6: Synthetic Outbreak data. The circles show the ground truth while the polygons show the relevant explanations produced by our system

For calculating recall, we can use the actual outbreak points since it doesn't require false positives.

$$Recall = \frac{true\_positives}{false\_negatives} = 0.74$$

5.2.1 Comparison

We compared the precision and recall of Hierarchical Intervention with Aggravation and Intervention.

Aggravation. We used the same observation for aggravation that we did for Hierarchical Intervention. From the top ten explanations, only one of

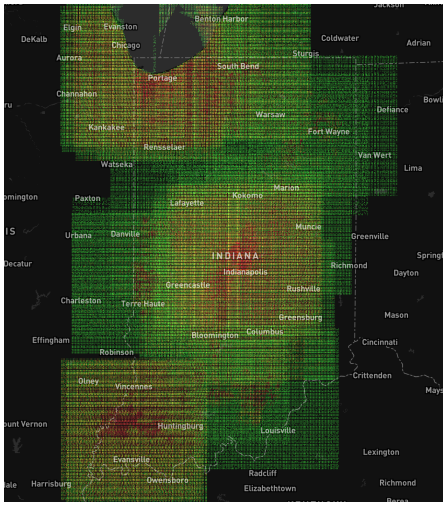

Figure 5.5: A heatmap showing sythesized data

the explanation was able to predict one of the 27 correct tuples in the ground truth.

$$Precision = 0.01$$

$$Recall = 0.04$$

Intervention. Finally, Intervention was used to find an explanation. Since there was only one hospital in the area of the outbreak, Intervention was able to provide that hospital as a relevant predicate in an explanation. This resulted in all the ground truth points being included in the explanation! However, since there are a large number of tuples related to this predicate, it resulted in a low precision.

$$Precision = 0.0003,\ Recall = 1$$

5.3 Speed and Scalability

The speed of each approach is simply the time taken to calculate the explanations. The time varies depending on the size of the data. For performance evaluation, we used the NYC TLC data for 2016. The data is divided into months. As we increase the number of months to be processed, the time taken for each approach increases almost linearly for aggravation, intervention and hierarchical intervention. For salient features, the time increases slightly exponentially. The results of our evaluation are displayed in Fig. 5.7

Experimental Setup. The speed of each approach is simply the time taken to calculate the explanations. Since all the approaches are implemented on top of distributed systems, the time varies depending on the number of nodes used. We experimented with calculating explanations for 1,3, and 5 months in a distributed environment. For the cluster, we used Google Compute Engine nodes(1 vCPU, 3.5GB RAM, 4 nodes). All the nodes in our cluster had the same specifications. The Aggravation,

Intervention and Hierarchical Intervention were tested on Apache Spark running on YARN while the Salient Features were generated with the Data Polygamy Framework running on YARN.

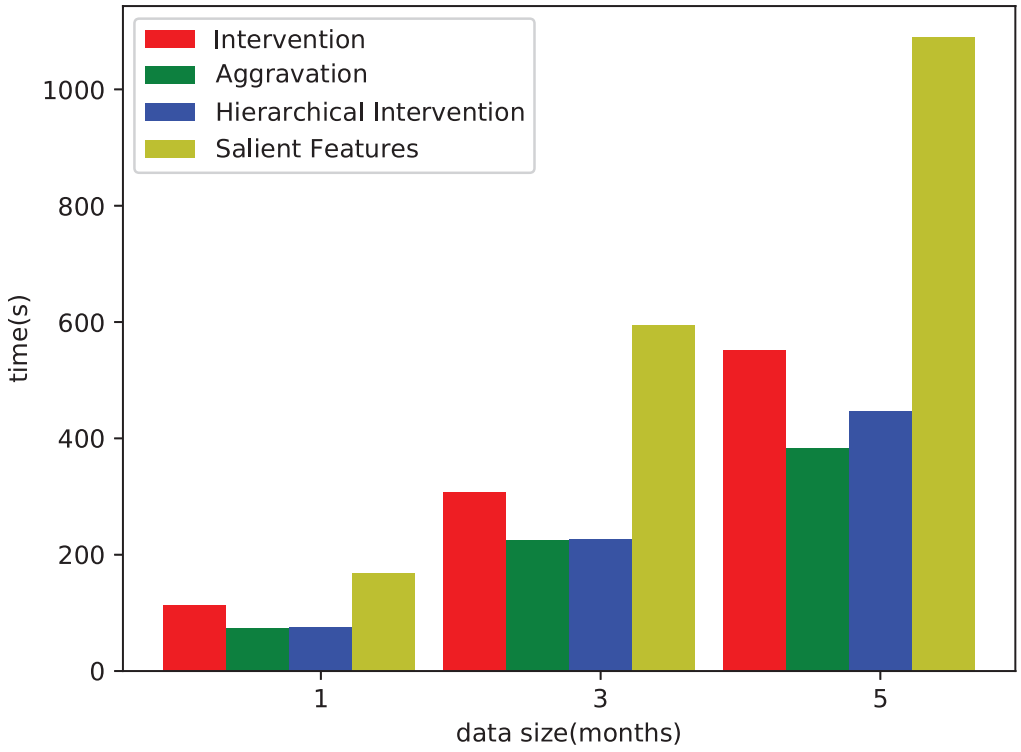

Figure 5.7: Time taken by each approach as we increase the size of data

The run times that were observed can be explained as follows. Aggravation takes the least amount of time of the compared approaches. Intervention costs more than aggravation because it includes the additional step of calculating statistics for the attributes as well as iterating the table. Hierarchical Intervention takes more time than aggravation but less time than intervention. This is because of several factors. One of the factors is that when we are using intervention and aggravation, we are using several attributes, on the other hand, hierarchical intervention only uses the

spatial attribute. On the other hand, we are creating a hierarchy. The in the case of this analysis we only used one query as the observation for the sake of comparison. However, multiple queries can affect the comparison as well. The time taken by Salient Features is also affected by a number of factors. First, the implementation of the salient features that we use is the one used in the Data Polygamy Framework (Chirigati *et al.*, 2016). This approach uses the MapReduce framework. The other approaches use the Spark framework. Since Spark makes use of an execution plan which is more efficient than MapReduce, it is a factor in the slower runtime of the Salient Features approach. Thus, when comparing salient features to the other approaches, this can be considered something similar to black box testing. The most expensive computation step in salient features is the interpolation step which doesn't occur in the other approaches.

Chapter 6

CONCLUSION/FUTURE WORK

In this thesis, we looked at different approaches for spatial explanations for arbitrary observations. We built an approach called Hierarchical Intervention. According to our evaluation, our approach outperforms aggravation and intervention in precision while it outperforms aggravation in recall. If we look at our approach from the context of a data analyst, it is designed to provide an advantage over traditional data analytics using relational database systems. The reason for this is because we use programming paradigms for distributed processing. Even though our implementation is designed to work on Apache Spark, the paradigms used may be useful for future distributed processing systems like GOLEM which have an even larger processing scope (Golem Factory GmbH, 2018). In addition to saving valuable time for data analyst, our system is also useful for reducing the scope of work for the analyst. The proposed system can be used as a search engine much like Google where the analyst uses observations instead of search terms and the system comes up with explanations instead of web pages. Instead of looking at the entire data, the analyst can look at a small subset represented by these explanations to find what they are looking for.

There are a number of improvements that can be made on top of our proposed approach that we weren't able to implement due to the lack of time or availability of datasets with the ground truth. One interesting idea is to use influence and intensity as parameters of perceptrons in a neural network(Grossberg, 1988; Widrow and Lehr, 1990). The neural network can be trained to explain specific datasets by using ground truth. Many spatial datasets that we find also have a temporal component. Our system does not handle the temporal aspect of the data. An improvement that can be

made to our approach is to use time range as a secondary component of our candidate explanations. We can build a temporal hierarchy in the same way we build a spatial hierarchy. The spatiotemporal hierarchy can be represented as a pyramid instead of a tree. If we extend our idea of the spatiotemporal system to a neural network, we can think of it from the perspective of a recurrent neural network(Chung *et al.*, 2016). It might also help to use Long Short Term Memory(LSTM) model to encapsulate the value of explanations at levels of the hierarchy which are far apart (Hochreiter and Schmidhuber, 1997). Since influence and intensity are expensive operations, the time complexity of the network should also be taken into account and heuristics should be used where necessary when using approaches with a lot of inputs.

REFERENCES

Adamson, C., *Star schema the complete reference* (McGraw Hill Professional, 2010).

Agafonkin, V., "Clustering millions of points on a map with Supercluster", https://blog.mapbox.com/clustering-millions-of-points-on-a-map-with-supercluster-272046ec5c97, accessed: 2018-04-27 (2016).

Agarwal, R., R. Srikant *et al.*, "Fast algorithms for mining association rules", in "Proc. of the 20th VLDB Conference", pp. 487–499 (1994).

Assuncao, R. M. and E. A. Reis, "A new proposal to adjust Moran's I for population density", Statistics in medicine **18**, 16, 2147–2162 (1999).

Aurenhammer, F. and R. Klein, "Voronoi diagrams", Handbook of computational geometry **5**, 201–290 (2000).

Baidu, "echarts", http://echarts.baidu.com/, accessed: 2018-03-28 (2018).

Bayer, R. and E. McCreight, "Organization and maintenance of large ordered indices", in "Proceedings of the 1970 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control", pp. 107–141 (ACM, 1970).

Beckmann, N., H.-P. Kriegel, R. Schneider and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles", in "Acm Sigmod Record", vol. 19, pp. 322–331 (Acm, 1990).

Beeri, C., P. A. Bernstein and N. Goodman, "A sophisticate's introduction to database normalization theory", in "Readings in Artificial Intelligence and Databases", pp. 468–479 (Elsevier, 1988).

Borthakur, D., "The hadoop distributed file system: Architecture and design", Hadoop Project Website **11**, 2007, 21 (2007).

Brin, S. and L. Page, "The anatomy of a large-scale hypertextual web search engine", Computer networks and ISDN systems **30**, 1-7, 107–117 (1998).

Brunsdon, C., S. Fotheringham and M. Charlton, "Geographically weighted regression", Journal of the Royal Statistical Society: Series D (The Statistician) **47**, 3, 431–443 (1998).

Cantelon, M., M. Harter, T. Holowaychuk and N. Rajlich, *Node. js in Action* (Manning Publications, 2017).

Charlton, M., S. Fotheringham and C. Brunsdon, "Geographically weighted regression", White paper. National Centre for Geocomputation. National University of Ireland Maynooth (2009).

Cheney, J., L. Chiticariu, W.-C. Tan *et al.*, "Provenance in databases: Why, how, and where", Foundations and Trends® in Databases **1**, 4, 379–474 (2009).

Chirigati, F., H. Doraiswamy, T. Damoulas and J. Freire, "Data polygamy: the many-many relationships among urban spatio-temporal data sets", in "Proceedings of the 2016 International Conference on Management of Data", pp. 1011–1025 (ACM, 2016).

Chung, J., S. Ahn and Y. Bengio, "Hierarchical multiscale recurrent neural networks", arXiv preprint arXiv:1609.01704 (2016).

City of Tempe, A., "Opioid Abuse Probable EMS Call", http://tempegov.maps.arcgis.com/apps/opsdashboard/index.html, accessed: 2018-05-02 (2018).

Cleveland, W. S. and S. J. Devlin, "Locally weighted regression: an approach to regression analysis by local fitting", Journal of the American statistical association **83**, 403, 596–610 (1988).

Dean, J. and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Communications of the ACM **51**, 1, 107–113 (2008).

Dismuke, C. and R. Lindrooth, "Ordinary least squares", Methods and Designs for Outcomes Research **93**, 93–104 (2006).

Dunn, O. J. and V. A. Clark, *Applied statistics: analysis of variance and regression* (John Wiley & Sons, Inc., 1986).

Elmasri, R. and S. Navathe, "Fundamentals of Database Systems sixth Edition Pearson Education", Reproduced with permission of the copyright owner. Further reproduction prohibited without permission (2011).

Facebook, "Prestodb", https://prestodb.io/, accessed: 2018-03-28 (2018a).

Facebook, "React - a javascript library for building user interfaces", https://reactjs.org/, accessed: 2018-03-28 (2018b).

Getis, A., "Spatial interaction and spatial autocorrelation: a cross-product approach", Environment and Planning A **23**, 9, 1269–1277 (1991).

Getis, A., "Reflections on spatial autocorrelation", Regional Science and Urban Economics **37**, 4, 491–496 (2007).

Getis, A. and D. A. Griffith, "Comparative spatial filtering in regression analysis", Geographical analysis **34**, 2, 130–140 (2002).

Getis, A. and J. K. Ord, "Local spatial statistics: an overview", Spatial analysis: modelling in a GIS environment **374**, 261–277 (1996).

Giovinazzo, W. A., *Object-oriented data warehouse design: building a star schema* (Prentice Hall PTR, 2000).

Golem Factory GmbH, "GOLEM", `https://golem.network/`, accessed: 2018-05-29 (2018).

Grossberg, S., "Nonlinear neural networks: Principles, mechanisms, and architectures", Neural networks **1**, 1, 17–61 (1988).

Guttman, A., *R-trees: A dynamic index structure for spatial searching*, vol. 14 (ACM, 1984).

Hartigan, J. A. and M. A. Wong, "Algorithm AS 136: A k-means clustering algorithm", Journal of the Royal Statistical Society. Series C (Applied Statistics) **28**, 1, 100–108 (1979).

Hochreiter, S. and J. Schmidhuber, "Long short-term memory", Neural computation **9**, 8, 1735–1780 (1997).

Hunter, J. D., "Matplotlib: A 2D graphics environment", Computing in science & engineering **9**, 3, 90–95 (2007).

InformationWeek, "Informationweek", https://www.informationweek.com/software/information-management/ups-nets-huge-fuel-savings-with-analytics/d/d-id/1112165, accessed: 2018-04-27 (2013).

Kohavi, R. *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection", in "Ijcai", vol. 14, pp. 1137–1145 (Montreal, Canada, 1995).

Koperski, K. and J. Han, "Discovery of spatial association rules in geographic information databases", in "International Symposium on Spatial Databases", pp. 47–66 (Springer, 1995).

Maciejewski, R., R. Hafen, S. Rudolph, G. Tebbetts, W. S. Cleveland, S. J. Grannis and D. S. Ebert, "Generating synthetic syndromic-surveillance data for evaluating visual-analytics techniques", IEEE Computer Graphics and Applications **29**, 3 (2009).

MacQueen, J. *et al.*, "Some methods for classification and analysis of multivariate observations", in "Proceedings of the fifth Berkeley symposium on mathematical statistics and probability", vol. 1, pp. 281–297 (Oakland, CA, USA, 1967).

Mapbox, "Mapbox", https://www.mapbox.com/, accessed: 2018-03-28 (2018).

Meliou, A., W. Gatterbauer, J. Y. Halpern, C. Koch, K. F. Moore and D. Suciu, "Causality in databases", IEEE Data Eng. Bull. **33**, EPFL-ARTICLE-165841, 59–67 (2010).

Meliou, A., S. Roy and D. Suciu, "Causality and explanations in databases", Proceedings of the VLDB Endowment **7**, 13, 1715–1716 (2014).

NYC and L. Commission, "TLC trip record data", `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`, accessed: 2018-04-29 (2016).

Olson, D. L. and D. Delen, *Advanced data mining techniques* (Springer Science & Business Media, 2008).

Ord, J. K. and A. Getis, "Local spatial autocorrelation statistics: distributional issues and an application", Geographical analysis **27**, 4, 286–306 (1995).

Powers, D. M., "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation", (2011).

Refaeilzadeh, P., L. Tang and H. Liu, "Cross-validation", in "Encyclopedia of database systems", pp. 532–538 (Springer, 2009).

Robertson, S., "Understanding inverse document frequency: on theoretical arguments for IDF", Journal of documentation **60**, 5, 503–520 (2004).

Roy, S. and D. Suciu, "A formal approach to finding explanations for database queries", in "Proceedings of the 2014 ACM SIGMOD international conference on Management of data", pp. 1579–1590 (ACM, 2014).

Shanahan, J. G. and L. Dai, "Large scale distributed data science using Apache Apark", in "Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining", pp. 2323–2324 (ACM, 2015).

Su, Y., G. Agrawal, J. Woodring, A. Biswas and H.-W. Shen, "Supporting correlation analysis on scientific datasets in parallel and distributed settings", in "Proceedings of the 23rd international symposium on High-performance parallel and distributed computing", pp. 191–202 (ACM, 2014).

Tan, P.-N. *et al.*, *Introduction to data mining* (Pearson Education India, 2006).

ten Cate, B., C. Civili, E. Sherkhonov and W.-C. Tan, "High-level why-not explanations using ontologies", in "Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems", pp. 31–43 (ACM, 2015).

Tilkov, S. and S. Vinoski, "Node. js: Using javascript to build high-performance network programs", IEEE Internet Computing **14**, 6, 80–83 (2010).

Uber, "deck.gl", https://uber.github.io/deck.gl/, accessed: 2018-03-28 (2018).

Webpack, "Webpack", https://webpack.js.org/, accessed: 2018-03-28 (2018).

Widrow, B. and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation", Proceedings of the IEEE **78**, 9, 1415–1442 (1990).

Yu, J., "Src: geospatial visual analytics belongs to database systems: the babylon approach", SIGSPATIAL Special **9**, 3, 2–3 (2018).

Yu, J., J. Wu and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data", in "Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems", p. 70 (ACM, 2015).

Zaharia, M., R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache Spark: a unified engine for big data processing", Communications of the ACM **59**, 11, 56–65 (2016).

Zhang, C., L. Luo, W. Xu and V. Ledwith, "Use of local Moran's I and GIS to identify pollution hotspots of Pb in urban soils of Galway, Ireland", Science of the total environment **398**, 1-3, 212–221 (2008).

Zhang, S. and C. Zhang, "Discovering causality in large databases", Applied Artificial Intelligence **16**, 5, 333–358 (2002).

APPENDIX A

NYC TLC SCHEMA

Attribute	**Type**
VendorID	int
tpep_pickup_datetime	date
tpep_dropoff_datetime	date
Passenger_count	int
Trip_distance	float
Pickup_longitude	float
Pickup_latitude	float
RateCodeID	int
Store_and_fwd_flag	bool
Dropoff_longitude	float
Dropoff_ latitude	float
Payment_type	int
Fare_amount	float
Extra	float
MTA_tax	float
Improvement_surcharge	float
Tip_amount	float
Tolls_amount	float
Total_amount	float

Table A.1: Trips Table

Attribute	**Type**
VendorID	int
Name	string

Table A.2: Vendors Table

Attribute	**Type**
RateCodeID	int
Name	string

Table A.3: Rate Code Table

Attribute	**Type**
Payment_type	int
Name	string

Table A.4: Payment Type Table