THESIS

ADAPTIVE SPATIOTEMPORAL DATA INTEGRATION USING DISTRIBUTED QUERY RELAXATION OVER HETEROGENEOUS OBSERVATIONAL DATASETS

Submitted by

Saptashwa Mitra

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2018

Master's Committee:

Advisor: Sangmi Lee Pallickara

Shrideep Pallickara
Kaigang Li

ABSTRACT

ADAPTIVE SPATIOTEMPORAL DATA INTEGRATION USING DISTRIBUTED QUERY RELAXATION OVER HETEROGENEOUS OBSERVATIONAL DATASETS

Combining data from disparate sources enhances the opportunity to explore different aspects of the phenomena under consideration. However, there are several challenges in doing so effectively that include *inter alia*, the heterogeneity in data representation and format, collection patterns, and integration of foreign data attributes in a ready-to-use condition. In this study, we propose a scalable query-oriented data integration framework that provides estimations for spatiotemporally aligned data points. We have designed Confluence, a distributed data integration framework that dynamically generates accurate interpolations for the targeted spatiotemporal scopes along with an estimate of the uncertainty involved with such estimation. Confluence orchestrates computations to evaluate spatial and temporal query joins and to interpolate values. Our methodology facilitates distributed query evaluations with a dynamic relaxation of query constraints. Query evaluations are locality-aware and we leverage model-based dynamic parameter selection to provide accurate estimation for data points. We have included empirical benchmarks that profile the suitability of our approach in terms of accuracy, latency, and throughput at scale.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents, Sumita and Subrata Mitra, for being my pillars of support and strength throughout my life. I thank them for showering me with unconditional love and always believing in me and pushing me to do better. Thank you, for all the sacrifices you have made, both in your life and your career to ensure that I always had the best opportunities in life.

I would also like to thank my advisor Dr. Sangmi Lee Pallickara for her guidance. Research was (and is still) not easy for me and I thank her for being patient with me, providing valuable advice and for her continued support throughout my career here at CSU.

I thank my committee members, Dr. Shrideep Pallickara and Dr. Kaigang Li, for promptly agreeing to be on my Thesis committee. Thank you for accommodating my Thesis defense on a such a short notice and for your valuable inputs. My research has been supported by funding from the US National Science Foundations Advanced Cyberinfrastructure (ACI-1553685), so I am grateful to them for their financial support.

Finally, I would like to thank all the faculty and staff here at the Computer Science Department for being so helpful and making life at the university pleasant.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The rapid growth of geo-sensors and remote sensing technologies have allowed geoscientists to explore numerous and complex attributes to understand phenomena that evolve spatiotemporally. The collected data is often integrated with other datasets from different data sources to enhance exploration options. A starting point for such explorations is to use queries to retrieve relevant portions from disparate datasets. The results from these queries can then be fused to create a custom dataset.

Challenges in accomplishing this stem from data volumes, the characteristics of the dataset, and the data collection processed. Data generated by these sensors are voluminous and are produced at high velocities. The heterogeneity of the collected data including their resolutions, representations, and encoding format posed challenges in storing, retrieving, processing, and analyzing them. The data gathering process involves diverse sensing equipment and variations in the data collection patterns.

Existing frameworks for spatial data management [1–8] provide scalable solutions for geospatial data storing and query over the multiple datasets. While these systems are highly effective from the scalable storage system standpoint, their spatial join algorithms rely on traditional query evaluations that return intersecting polygon pairs that do not address the nature of mismatching between the locations of samples and timestamps. Once the data is retrieved, users perform data interpolations to estimate values at the precise matching location and time. This often causes repetitive data retrievals because most interpolation algorithms are iterative entailing access to the surrounding area and/or temporal scopes. Furthermore, users often explore the parameters to achieve more accurate estimations.

To address these challenges, we propose a framework for accomplishing effective, accurate, and scalable data integration operations. In particular, our framework is independent of the underlying data model and supports: (1) a distributed data hosting environment for the heterogeneous

datasets, (2) approximate join operations, (3) feature interpolations in case of spatiotemporal misalignment with uncertainty estimate, and (4) automatic interpolation parameter selection schemes.

1.1 Research Questions

Enabling scalable data integration operations over a distributed geospatial data storage must address challenges stemming from dataset sizes, variations in the density of data availability, and latency requirements during approximate query evaluation. The following research questions to guide our study in this paper:

- **RQ1:** How can we support query-based integration of diverse geospatial datasets that are not spatially and/or temporally aligned?
- **RQ2** How can the system cope with different data representation models such as raster and vector?
- **RQ3** How can we effectively orchestrate our data integration algorithm while ensuring support for interactive applications?
- **RQ4** How can the system select parameters of the interpolation algorithms that ensure the highest accuracy?

1.2 Overview of Our Approach

Our framework, Confluence, facilitates query-driven scalable and interactive spatiotemporal data integration. Confluence dynamically generates estimations at the aligned data point. Data alignments are performed based on the specified key indexing attributes such as spatial coordinates and temporal ranges. The integration operations support vector and raster data representation models. Raster data is made up of pixels (spatiotemporal extent) and each pixel has an associated value. Vector data model represents the data space using points, lines, and polygons. Our system automatically transforms values between models by means of aggregating data points or applying representative data values. The results are delivered with the associated uncertainty information.

Based on the user′s query (e.g. spatial coverage), the system performs distributed query relaxation that allows query constraints to be relaxed to cover neighboring spatiotemporal scopes. The maximum range of relaxation is configurable in the system. To reduce data movements within the storage cluster, the system schedules the computations on nodes that host a majority of the data that match the user′s query. The system provides separate indexing schemes for boundary regions of blocks that need to be transferred to perform integration and it is used for query evaluation and caching.

This study proposes a dynamic relaxation model, an adaptive interpolation scheme to cope with the repetitive exploration of the parameter spaces for interpolation algorithms, such as Inverse Distance Weighing (IDW) [9]. Confluence trains model and estimates the best parameters for a given target spatiotemporal region. Our experiments used an artificial neural network to train the dynamic relaxation model for IDW, and it showed, depending on the feature being interpolated, an improvement in accuracy by $9\% \sim 31.25\%$ with the estimation compared to using static parameters.

We validate our ideas in the context of the Galileo system for spatiotemporal data. Galileo is a cloud-compatible hierarchical distributed hash table (DHT) implementation for voluminous and multidimensional spatiotemporal observations [10]. The key organizing principle in Galileo is the use of geohashes whose precision can be controlled to collocate data from contiguous geographical extents.

1.3 Paper Contributions

Confluence targets interactive data integration that allows the analysis to import relevant portions of voluminous auxiliary dataset interactively. Our query-based data integration process performs dynamic alignments across datasets and also generates interpolated values in situations where data is unavailable. The dynamically generated estimations are ready-to-use as input to interactive analytics including visualizations. Specific contributions of Confluence include:

- A rapid, query-driven approach to data integration with dynamic, run-time estimation of values.
- A model-based approach to select the parameters that achieve the most accurate estimation per data points.
- Geospatial data integration across the spatial data representation models.
- Data representation neutral integration of datasets with uncertainty measures.

1.4 Paper Organization

The remainder of this paper is organized as follows. In section 2, we discuss some related work in this field along with the standard spatiotemporal join operation and introduce the Galileo distributed storage system, where we will stage our datasets. In section 3, we discuss the concept of relaxed data integration and the different scenarios encountered while doing so based on the type of datasets involved. Section 4 deals with the methodology used in our data integration operation, followed by a discussion of the system architecture and process flow in section 5. In section 6, we evaluate the performance of the Confluence system for various scenarios involving relaxed data integration, along with a case study showing the applications of integrated data followed by a conclusion in Section 7.

Chapter 2

Background and Related Work

2.1 Related Work

2.1.1 Spatiotemporal Data Analysis

With data being collected at an unprecedented rate, the number of big data available for analysis has seen a drastic increase in the past few years. The recent popularity in the area of Big Spatial Data has seen an increase in the number of such spatiotemporal datasets, along with technologies designed specifically for spatiotemporal data management, data processing, and spatial analysis (such as spatial query, visualization etc) [11].

The improvements in sensor technologies have led to increased analytics being carried out on Big Spatial Data. Along with distributed analytics on such large spatial datasets, the integration of multiple spatial datasets in a distributed systems setting has also been explored in previous research. There are several systems that successfully handle storage and range queries on spatial data. However, performing spatiotemporal integration has been lesser explored [1–7], since it involves dealing with the heterogeneity of the participating datasets (variety), along with the significant differences in the coverage, accuracy, and timeliness of the records in the datasets. Also, most of these cases deal simply with spatial data integration where all records from participating datasets that overlap spatially are returned in the output. Hence, an even smaller subset of the work actually deals with the spatiotemporal integration of the participating datasets that is defined by *finding pairs of points within x Euclidean distance and t temporal distance* [7].

A common theme among the research mentioned above is that the spatiotemporal data is first housed in a distributed storage system, such as Hadoop DFS [1] and the analytics jobs are designed using existing methodologies such as Hadoop Map-Reduce or Apache Spark that can be executed on top of it. In order to reduce the data access time on these distributed systems, all such research create some sort of *Spatial Indexing structure* on top of the storage systems. These indices help

accelerate the process of data access by acting as a metadata that can be used to reduce the number of blocks accessed and hence reduce disk I/O. However, it must be noted that the creation of the spatial indices is a separate process that has to be completed before any spatiotemporal join queries can be fired. The creation of this index over the already-stored dataset is time-consuming and is a one-time operation, assuming no updates are made to the filesystem. In case of updates or insertion of new blocks in the dataset, these indices need to be updated using a separate batch job, which could be an overhead. This reduces the throughput of such systems. Our work focuses on creation and maintenance of a spatiotemporal index that creates and updates itself with every update to the dataset and hence we do not have to separately run services to generate/ update it before a data integration operation.

Integration of observational data streams in the context of peer-to-peer grids have been explored in [12, 13] while those in multimedia settings have been explored in [14]. Often these integrations have been performed at the publish/subscribe system [15, 16] underpinning them.

Efforts have also explored the use of queries to launch analytic tasks [17, 18]; these queries performed targeted analytics and are not designed to support general analytic operations. They do not support integration of multiple datasets either. Budgaga et al have explored the use of parameter space sampling and the use of ensemble methods to construct models over spatiotemporal phenomena [19]. Unlike our effort, this approach does not entail data integration. The Synopsis [20] system constructs sketches of spatiotemporal data that can be subsequently used to construct synthetic datasets for particular portions of the feature space. This is synergistic with our methodology and can be used to support interpolation operations as a future work. Our methodology can also interoperate with spatiotemporal data storage systems [10, 21]. Efforts to integrate data from diverse data sources have explored the use of metadata as the basis for integration [22]. However, metadata focused efforts do not perform uncertainty quantification.

2.1.2 Spatiotemporal Interpolation

The calculation of a point estimate for the value of an object of interest from a set of claims has also been covered in various research. One such case is the change of support problem, which involves interpolation of spatial data features at points different from those at which they have been observed [23, 24].

Related works on interpolating values of unknown locations based on values from related surrounding regions have been covered in works such as [25]. The focus of this work has been to use the inverse-distance weighing interpolator (IDW), with cross-validation as a method of predicting the unknown value of a parameter. The underlying assumption with this methodology is that the value of a parameter at a point is influenced by the parameter values observed in its neighborhood and the degree of this influence decreases the further away that observation is from the point of interpolation. This was followed by a subsequent jackknife resampling was then used to reduce bias of the predictions and estimate their uncertainty. While IDW assumes the influence of the neighboring points on the interpolated value as a function of their distance from the point of interpolation, there are methods that take other factors into consideration as well. For example, Kriging [26], is another interpolation method that determines the influence of the neighboring points by creating statistical models that track the relationship among the neighboring observations. However, unlike IDW, Kriging is a multistep operation and could be very time consuming if the number of observations involved is large.

Since at the interpolation point, we do not know that value of the actual parameter, several research has attempted to quantify the uncertainty/ error involved in the interpolated value in the form of a trust score. [27] [28] dealt not only with finding out the point value of an object of interest from a given set of claims but also with quantifying a confidence interval for the estimate. The confidence interval is a measure of the accuracy of the estimated value achieved through the iterative process. The confidence in the predicted feature value is inversely proportional to the size of the confidence interval that is returned. Here, for the interpolation, in an iterative manner, the interpolated value is computed based on the trustfulness score of the information

sources (neighboring observations), the correctness of which is then used to update the trust value of the sources and so on.

2.2 Spatiotemporal Data Integration

A Spatiotemporal Data Join is one of the most general form of the data integration operation. It involves two participating spatiotemporal datasets to be joined based on the overlapping spatiotemporal attributes. This operation is denoted by the method:

```
STJoin (target_data, source_data, spatial_coverage, temporal_range)
```

where we have two spatiotemporal datasets, *target_data* and *source_data*. The target_data is the dataset for which records from the foreign_data dataset needs to be imported for integration. In a distributed setting, the foreign_data is the dataset whose records need to be fetched, if necessary, in case they are not co-located with the target_data.

The Spatiotemporal join operation identifies the overlapping subsets that match the query′s specified geospatial area (by spatial_coverage) and time (by temporal_range). The resultant dataset includes all pairs of records from the two datasets that overlap in terms of their spatiotemporal attributes. The spatial_coverage can be specified in the query in terms of a set of geographic coordinates that represent a polygon on a 2-D map of the earth. The temporal_coverage can be specified as a range in terms of a starting and an ending timestamp.

Compared to the existing traditional STJoin operations [refs], Confluence provides both spatial and temporal interpolated estimations within the query results, in case an exact spatiotemporal overlap between points is not found between the records of the two participating datasets. The interpolated estimations derived from the foreign dataset includes uncertainty measures if the data aggregation is required. This operation has been developed on our distributed geospatial storage system.

2.3 Distributed Geospatial Data Storage System

The efficiency of the data integration operation depends on how efficiently the dataset is stored in a distributed system. For the purpose of storage of the spatiotemporal datasets, we have used the Galileo distributed file storage system [10], which is a high-throughput, distributed storage framework for large multidimensional, spatiotemporal data-sets. Confluence leverages the distributed query evaluation capability of Galileo to efficiently identify the data blocks that match the query′s geospatial area and temporal range. We provide a brief overview of some of the relevant key components of Galileo in the following section.

2.3.1 Galileo Cluster Structure

Galileo is a Distributed Hash Table(DHT) based storage system. In Galileo, by convention, each dataset is called a Geospatial File System and can have its own partitioning scheme for the nodes in the cluster. We have selected the modified two-tiered hashing system [29] based on the spatiotemporal characteristics of the incoming data as the partitioning scheme for our datasets. The spatiotemporal observational space of the dataset is partitioned among the Galileo nodes in two phases - the first phase separates the nodes into separate groups, where inside each group, nodes are organized in a ring structure (node-ring) and the second phase puts the groups in another ring called the group-ring. As illustrated in Fig 2.1 the high-level identifier ring is used to partition the data based on the temporal aspect. The time-stamp of incoming data point is used to locate the storage group. Once the group of the storage nodes is identified, the partitioning algorithm will identify the final location (node) to host the data point within the group of nodes. Galileo uses geohash [30] to generate data partitions that store geospatially neighboring data points together. The granularity of size of data block is determined by the length of geohash code managed by the nodes.

Galileo allows users to customize the partitioning scheme based on the type of data acquisition and analytics. The basic unit of partitioning of the temporal domain includes hourly, daily, monthly or yearly. Similarly, the spatial partitioning is fine-tuned based on the geohash precision selected.

Fig 2.1illustrates an example of geospatial data ingestion. As an example, if a client ingests a data point that contains a geo coordinate of (40.5734° N, 105.0865° W) and temporal information of December 1, 2017, Galileo calculates the hash value of the temporal information using SHA-1 and maps it to a matching group. Then, Galileo calculates the geohash value with the geo-coordinate and determines the node in the group to store the data point (which is *9xvg* in this case). Finally, the data is forwarded to the destination node.

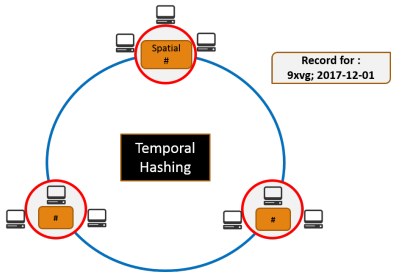

Figure (2.1) Node Partitioning Scheme

2.3.2 Galileo Metadata Graph

Each of the Galileo nodes maintains an in-memory data structure called the *Metadata Graph*. The Metadata Graph is an indexing scheme for local data blocks that speed up data retrieval by avoiding expensive disk I/O. This is especially effective when the system hosts voluminous datasets.

The Metadata Graph is an in-memory prefix tree that holds the metadata information of every data block stored in a node. Every depth-first traversal of this tree represents the metadata of a particular block, with the leaf node holding a path to the block location. For any data access query made to the Galileo system, the Metadata Graph helps reduce the number of target blocks by looking into the metadata information it stores and returning the list of blocks that match the feature constraints of that query.

Fig.2.2 shows a sample representation of the Metadata Graph for a node with three blocks in it each covering the following geospatial domains - *{9xvb; 2017-12-01}*, *{9xvh; 2017-12-01}* and *{9xvb; 2018-01-01}*. Although the figure shows only spatiotemporal features in the graph, it can contain other features from the dataset as well that are queried frequently.

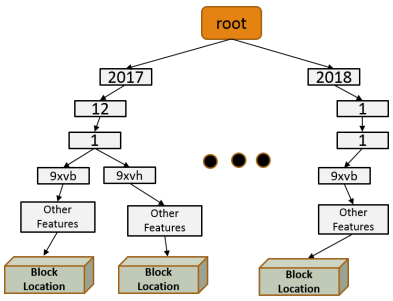

Figure (2.2) Metadata Graph

2.4 Candidate Dataset Properties

Our system is designed to enable integration of heterogeneous spatiotemporal datasets, which are a collection of multivariate records, each containing spatial and temporal information. Our goal is to take two datasets housed in a Galileo cluster and integrate similar pairs of records from the datasets based on their spatial and temporal attributes (we may also add conditions on other features as well).

Our system supports data integration between any combination of data representation models. To examine our system with various types of datasets, we use both point datasets (where each record represents a point on earth′s surface at a particular time) and rasterised datasets (where each record might represent a value for a span in the spatiotemporal domain).

However, the spatiotemporal datasets involved in this operation may contain records of varying resolution. Take, for instance, the US Census data, which is yearly data with records going down to block-group level. So, for a Census dataset, for each result, the temporal resolution is year and

the spatial resolution is a block-group. Compare that to the NOAA Integrated Surface Database (ISD), which consists of global hourly and synoptic observations compiled from numerous sources all over the world. So in this case, the records represent points on the earth′s surface at a particular point of time.

Chapter 3

Relaxed Geospatial Join Across Geospatial Data Representation Models

In this section, we explain the **self-adaptive relaxed spatiotemporal data integration** operation in details along with some of the different scenarios, that we may have to deal with, depending on the type of the candidate datasets involved.

3.1 Relaxed Conditions for Data Integration

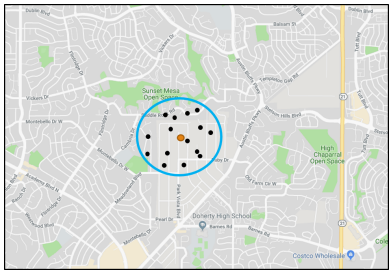

Figure (3.1) Relaxed Conditions(Geospatial)

Given two indexed datasets A and B, and a spatiotemporal predicate θ (e.g. geospatial area and/or temporal range), traditional spatiotemporal join operation merges the two datasets A and B on the predicate θ. However, the data points, $a \in A$, and $b \in B$, that satisfy the predicate may not have the exact same spatiotemporal attributes. Subsequent analyses often perform data interpolation algorithms to generate a virtually integrated dataset that can be ingested to their analysis. This often involves repetitive I/O access and computation.

To provide the estimations as a part of query results, Confluence retrieves spatiotemporally neighboring data points as the query results, in the absence of exactly matching spatiotemporal attributes. Fig.3.1 illustrates an example that estimates an attribute value (from a dataset A) at

the location marked with a red dot. Interpolation algorithms often require neighboring points, marked with black dots (from dataset B) within a pre-defined radius (blue circle). We call the surrounding region retrieved to estimate attribute values at the red point as the **relaxation region** and the loosened predicate as the **relaxed conditions**.

Confluence provides a set of spatiotemporal integration operations. The data integration queries support approximate results at the spatiotemporal index of the target dataset. The native data integration operations have the format:

```
STJoin (target, source, coverage_spatial, coverage_temporal,
attributes, relaxation_spatial, relaxation_temporal, model)
```

This operation imports attributes from the `source` dataset to match those in the `target` dataset. To specify a spatial predicate in the query (`coverage_spatial`) that defines the geospatial query area, users may use a polygon denoted by a set or geospatial coordinates. The temporal predicate of this operation `coverage_temporal` is specified as a range in the form of two time timestamps or simply time strings. The results will include raw data points matching with the coverages, and/ or interpolated estimations. Users can specify interpolation algorithms in the query (using (model)) and the Confluence APIs are extensible. The parameters, `relaxation_spatial` and `relaxation_temporal`, constitute the query's **relaxation region**, which is the spatiotemporal neighborhood for each target record in which we should look for neighboring source records for interpolation. If the interpolation is set as 0, users will retrieve raw data without interpolation. Throughout the rest of the paper, we will refer to the `target` dataset as FS_A and the `source` dataset as FS_B.

Since Confluence supports both vector and raster data representations, this creates four scenarios of data integration operation that we might encounter. Table.3.1 depicts the operations over the geo-data representation models.

- **Vector-to-Vector:** In this operation, both datasets are represented as vector records, similar to the situation in Fig.3.1. In this scenario, Confluence provides a methodology to estimate

Table (3.1) Relaxed Data Integration On Different Datasets

Source Data Type	Target Data Type	Required Data Processing
Vector	Vector	Overlapping Record Pairs +Interpolated feature
Raster	Vector	Overlapping Record Pairs +Interpolated feature
Vector	Raster	Overlapping Record Pairs +Interpolated feature
Raster	Raster	Overlapping Record Pairs

the value of a feature at a point based on the values of the same feature from the relaxation region.

- **Raster-to-Vector:** This operation imports attributes with an associated raster pixel to be integrated with a group of vector records. The representative raster value is matched to all of the vector points within the raster pixel. In terms of interpolation, Confluence provides an aggregated value of a feature from the vector dataset, representative of the raster pixel, based on the vector points lying in the raster pixel region.

- **Vector-to-Raster:** In this scenario, Confluence aggregates values of the sources within the area defining a pixel of the rasterized data (target). The interpolation is similar to that of the raster-to-vector case.

- **Raster-to-Raster:** This operation involves both of rasterized records. The result returns pairs of records from both datasets whose bounds/extent intersect. In this case, due to the lack of an available interpolation strategy that could be applied in a generalized manner, we avoid interpolation.

3.2 Self-Adaptive Relaxation Conditions

If we take a look at Fig.3.1, we can see that the number of neighboring points that get matched against each record of A depends on the size of the blue circle, which represents the spatiotemporal relaxation involved with each data integration operation. We can think of the relaxation region as

a 3-dimensional spherical area (the dimensions being latitude, longitude and time) around each datapoint in A.

Since our integration strategy is based on estimating the value of a point/region based on the observations from neighboring points, each point lying in this sphere will influence the point estimate for the value of a feature. In case of most spatiotemporal data, the influence of neighboring points on the value of a feature at a specific spatiotemporal location varies with the region at which we are performing the interpolation. To incorporate this aspect, we introduce a way to dynamically tune certain parameters related to the interpolation operation at runtime, depending on the spatiotemporal region we are interpolating at. The dynamic interpolation feature prediction adjusts the influence that the neighboring points have on the final interpolated value of a feature.

In our data integration methodology, we have used machine learning to dynamically influence the interpolation operation by moderating the degree influence neighboring points would have on the estimate. This methodology is explained in detail in section 4.5.

Chapter 4

Methodology

4.1 Distributed Query Relaxation

A simple inner-join operation between datasets A and B would involve only those points from A and B that have a complete match for their spatiotemporal attributes (one-to-one correspondence), but, due to the nature of the datasets involved, our relaxed data integration operation yields a one-to-many map as a result. As we can see from Fig.3.1, for a point in dataset A, say A_i, a collection of k points, say, $\{B_{i0}, B_{i1}, ..., B_{ik}\}$ from dataset B lying in the spatiotemporal neighborhood will be returned as results of the data integration operation, in case an exact match of spatiotemporal attributes is not found.

With such a one-to-many map as the output, we might need to have a one-to-one correspondence between FS_A and FS_B for particular attributes. For instance, in the vector-to-vector scenario, if we have the feature, *altitude*, in FS_A and *temperature* in FS_B. So, corresponding to an FS_A record with an altitude reading, we will have a set of records in its spatiotemporal neighborhood with different temperature reading. Using our feature interpolation functionality, we aim to estimate the temperature at that altitude with the same spatiotemporal point using the FS_B points that lie in its spatiotemporal neighborhood.

Similarly, in case of a rasterised-to-vector scenario, for a rasterised region for an FS_A record, we will have multiple vector points from FS_B that are contained in it. Using feature interpolation, we attempt to estimate the value for an FS_B feature for the entire region represented by the raster record using the set of FS_B points in the one-to-many map.

Thus, through our interpolation operation, we want to generate a synthetic value for a feature at the same spatiotemporal location as that of A_i. Also, since the interpolated value is merely an estimate of the actual value, we also provide an estimate of the uncertainty involved with the prediction.

The methodologies used for interpolation from neighborhood values will be discussed in section 4.4.

4.2 Relaxation Region

4.2.1 Neighboring Blocks

Since data in Galileo is grouped into separate blocks for discrete geohash and time slice (which could be hour, day of month, month or year), given two blocks B_1 and B_2 from two separate filesystems (datasets) whose spatiotemporal domains are not disjoint, either B_1′s spatiotemporal range would include B_2′s spatiotemporal range or vice-versa or B_1 and B_2′s spatiotemporal range could co-incide, depending on what the geohash and temporal precisions have been set for the blocks of those filesystems.

The concept of neighboring blocks comes into play because of the spatiotemporal relaxation condition, since for each block of FS_A, the region that needs to be looked at is the spatiotemporal region covered by the block plus the relaxation region around that block. Neighboring blocks for a particular FS_A block refer to the FS_B blocks that cover any portion of the relaxation region of that FS_A block.

As mentioned above, the spatiotemporal bounds of any neighboring FS_B blocks will either fully enclose that of the FS_A block, or lie on the edge of the spatiotemporal bound of the FS_A block. Hence, in a case where the FS_B block contains the entire bounds of an FS_A block along with its relaxation region, then, a single FS_B block contains both the core data of the FS_A region along with the relaxation region. In all other scenarios, we might need to scan multiple FS_B blocks.

Having that in mind, if we consider a block of FS_A, say Blk_{Ai}, and the spatiotemporal range for Blk_{Ai} as a cube in the 3-D space (latitude, longitude, timestamp), let′s call it C_{Ai}, any block in FS_B whose spatiotemporal cube externally lies on either of surface of C_{Ai} is a neighboring block of C_{Ai}.

To illustrate, let us assume Blk_{Ai} is a block of FS_A covering the spatial span of *9z* and date *29-04-2017*, as shown in Fig.4.1. Let us denote a block by the spatiotemporal domain it covers as *{9z,29-04-2017}*. We can see that spatially, geohashes *{c8,cb,f0,9x,dp,9w,9y,dn}* are the regions that lie in the immediate spatial neighborhood of *9z*. Similarly, the dates *30-04-2017* and *28-04-2017* are the immediate temporal neighbors of the temporal region *29-04-2017*. So, to summarise, any block covering the spatial region *c8, cb, f0, 9x, 9z, dp, 9w, 9y, dn* and the temporal range *28-04-2017* to *30-04-2017* except for the spatiotemporal region *{9z,29-04-2017}* is the neighborhood region of the block Blk_{Ai}(requesting block).

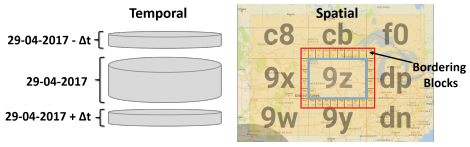

Figure (4.1) Relaxation Regions

It is to be noted that not the entirety of the data from the neighboring blocks are required for Blk_i, but rather a bordering spatiotemporal fragment, depending on the relative position of the block with Blk_i in the spatiotemporal domain. We call the operation of requesting partial data from neighboring blocks as *Neighbor Data Request*.

4.2.2 Maximum Spatial and Temporal Relaxation

This represents the maximum limit to which we can set the spatial (in terms of latitude and longitude) and temporal(in terms of milliseconds) relaxation parameter for a data integration query. This parameter needs to be set before blocks are stored in the filesystem and aids in our new indexing scheme.

It would be extremely time-consuming to fetch the bordering areas of an FS_B block as per the query relaxations during runtime, since in that case, we would have to go through all the records in an entire block even if we only need a fragment of its data. The maximum relaxations are a way for

the Border Indices to shortlist beforehand the records from a block that fall in the spatiotemporal bordering areas of a block, so that only the shortlisted records get evaluated during runtime.

In our approach, we set the maximum extent of the spatial relaxation that is allowed in a query in terms of geohash characters and the maximum temporal relaxation in terms of milliseconds. In order to make the relaxed data integration operation swift and efficient, there are a few assumptions/ constraints that have to be decided on, regarding the participating datasets (filesystems) during their creation, i.e. before any data blocks are saved. All subsequent data integration queries have to abide by these constraints for the query results to be accurate.

1. Each dataset should specify the values of the maximum relaxation parameters during its creation.

2. In any data integration operation involving two datasets, the query relaxation parameters used must be lower than the lowest maximum spatiotemporal relaxations out of the two datasets.

4.2.3 Maximum Relaxation Region (MRR)

Since the coverage of a block is defined by a geohash, which covers a rectangular spatial plot and a time range, which could be an hour, a day, a month, or a year, we can think of the spatiotemporal extent of each block as a three-dimensional cube. The relaxation region for a block would form an extra layer of coating over this cube. Similarly, a Maximum Relaxation Region is the region formed by coating the original extent of a block with the maximum relaxation parameters for that dataset. So, for a block, Blk, the three-dimensional cube that is formed by combining the original spatiotemporal extent of a block along with its relaxation region is called the Maximum Relaxed Region (MRR) for Blk.

4.2.4 Bordering Region

By bordering regions of a block, we mean any portion of the total spatiotemporal region covered by a block that might be requested as an FS_B neighboring block data by another FS_A block

as part of a *Neighbour Data Request*. As we explain below, the bordering region is defined by the maximum spatial and temporal relaxation parameters that are set for each filesystem (dataset) before actual data blocks are stored for it.

For instance, to extend the example in Fig.4.1, let us assume a block from FS_A covering the region *cb, 29-04-2017* (let us call it Blk_A) and another block from FS_B(Blk_B) covering the spatiotemporal region *9z, 29-04-2017*. Now, in case of an integration operation involving Blk_A, the corresponding FS_B data from Blk_B that is required, is the portion of the block records that cover the northern flank of the geohash region $9z$(as shown in Fig.4.2). No other region from Blk_b is needed for data integration of Blk_A.

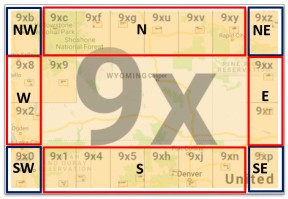

Figure (4.2) Border Flanks

Hence, we can see that spatially, each block can have 8 possible bordering regions/flanks that may be requested in a neighbor′s data integration- N,S,E,W,NE,NW,SE,SW. The width of each flank depends on the relaxation prameter. Temporally, if the bounds of a block is from time T_{start} to T_{end}, then the two temporal bordering regions for that block are the regions T_{start} to $T_{start} + \delta t$ and $T_{end} - \delta t$ to T_{end}. Any combinations of these two sets of flanks are potential bordering regions for this block that may be requested in another adjacent block′s *Neighbour Data Request*.

4.2.5 Neighbors′ Orientation

The orientation of a neighboring block with the requesting block determines which fragment of its data needs to be returned as a part of the *Neighbour Data Request*. The fragments of data being returned as part of a response to a *Neighbour Data Request* are called flanks.

Let us take the example of the two blocks, *{cb, 29-04-2017}* (Blk_A) and *{9z, 29-04-2017}* (Blk_B), as mentioned before. The orientation of Blk_B with Blk_A is represented as *'S-Full'*, meaning, spatially only the records involving northern flank of Blk_B is of interest in a data integration operation involving Blk_A and temporally, all the records are of interest. So an intersection of these two sets of records, i.e. all records of Blk_B that lie in the northern flank are to be returned. In another example, let Blk_A and Blk_B be *{dn, 29-04-2017}* and *{9z, 30-04-2017}* respectively. In this case, spatially from Blk_B only data points that lie in the south-eastern flank are of interest and temporally, only records in the time slice from the beginning of 30-04-2017 to δt milliseconds are to be considered. So the orientation of Blk_B with respect to Blk_A in this scenario is denoted as *''NW-down'*(down meaning the temporal range for Blk_B lies after that of Blk_A).

4.3 Spatiotemporal Border Indexing Scheme

4.3.1 Border Index Overview

Having introduced the various terminology involved, we now introduce our additional in-memory data structure called the **Border Index** for indexing of the blocks that get staged in the Galileo File System. Similar to the Metadata Graph, this index is created /updated during the storage of blocks on a node. However, unlike the Metadata Graph, our new indexing scheme concerns itself with tagging those records in a block that might lie in the spatiotemporal bordering regions. The Border Index is there to help extract the border records from a block, should they be requested as a neighboring FS_B data.

In Galileo, we maintain an in-memory Border Index against each block stored for each file-system (dataset). Each Border Index is tagged to its corresponding block′s absolute path using an in-memory map (called the *Border Map*) from the path to the Border Index. Each dataset in Galileo maintains its own separate Border Map and a set of Border Indices for its blocks.

The purpose of a Border Index is to tag those regions that lie in the maximum spatial and temporal relaxation regions in a block. It stores the record number of each record that lies in the bordering region of the block along with their orientation. The index is created such that whenever

data from a particular flank of the block is requested (such as *'S-NW'*), we can easily extract record numbers in a block that matches that criteria without having to evaluate the entire block, which could become very time-consuming.

A Border Index is created against each block during its creation. It is then updated every time new records are appended to that particular block. Next, we explain the components of each Border Index to better understand its functionality.

4.3.2 Border Index Components

The Border Index contains in it information about the bounds of different possible spatiotemporal flanks along with a list of record index numbers corresponding to each of those flanks. So, one part of the Border Index stores information of the bordering bounds (**Flank Descriptors**) and the other part keeps track of the record indices that belong to those bounds (**Flank Record Indices**). It also maintains the **total number of records** in the block currently.

The **Flank Descriptors** are information that a Border Index creates during its creation and are never updated after that. These are information that help us determine which spatial and temporal flank to tag a block record to. The **Spatial Flank Descriptors** in a Border Index contains the list of geohashes that lie on each possible spatial flank for a block. The possible spatial flanks are N, NE, E, SE, S, SW, W and NW (8 in total). Depending on what the maximum spatial relaxation is set in terms of geohash precision, the Spatial Flank Descriptors consists of 4 lists of strings and 4 strings (since the corner flanks are a single geohash), consisting of the geohashes that lie in each of the possible flanks. For example, a block with the spatial coverage of Fig.4.2 in a dataset with maximum spatial relaxation of 3, would have its N, NE, E, SE, S, SW, W and NW flank descriptors as *[9xc, 9xf, 9xg, 9xu, 9xv, 9xy], 9xz, [9xx, 9xr], 9xp, [9x1, 9x4, 9x5, 9xh, 9xj, 9xn], 9x0, [9x8, 9x2]* and *9xb* respectively. The **Temporal Flank Descriptors** simply contain two pairs(up and down) of numbers that represent the range of timestamp for the upper flank and the lower flank for a block. These descriptors are used to determine which flank a record belongs to and thus tag it to that corresponding Flank Record Index.

The **Flank Record Indices** contain a total of 10 lists of integers (corresponding to 8 spatial and 2 temporal flanks), where each integer corresponds to the record number from the block that lies in that particular flank. This part of the Border Index gets updated every time there are new incoming records stores in a filesystem block. For spatial Flank Record Indices, if a record lies in any of the spatial flanks, it will get added to its appropriate index list only if it lies in either of the temporal flank (and vice-versa). This means, a record has to lie both in a spatial and temporal border region to make it in either of the Flank Record Indices, since, otherwise, there is need to be checked for check it for a neighbor request.

4.3.3 Figuring out Orientation

Once, based on a neighbor request, we figure out which flanks are needed to be returned, these Flank Record Indices are used to pick out the records that need to be matched against a particular spatiotemporal MRR.

Hence, for instance, if we find that for a block, the flank that is needed to be returned is of orientation 'E-up', we just have to find the common indices from the spatial flank indices for 'E' and the temporal flank indices for 'up' and those are the records to be matched against the MRR bounds.

4.3.4 Neighbor Elimination Using Bordering Index

As mentioned before, for integration for each block, Blk_{Ai}, of FS_A, we need FS_B data for the region corresponding to its MRR; that amount of FS_B data needs to be fed to the thread that would handle the data integration operation for this particular segment. The data could lie in fully the current node, or a fraction of it could lie in some other node in the cluster from which it would need to be fetched with a Neighbor Data Request.

However, although it is true that, to get accurate results, we need FS_B data from the entire MRR area, there are spacial cases where certain flanks of the MRR that lie outside the spatiotemporal extent of Blk_{Ai} are not needed. Using our indexing scheme, we will show how to further eliminate flanks of the MRR that are unnecessary to be fetched for these scenarios. Hence, by eliminating

certain flanks of the MRR, we will be reducing the spatiotemporal range of Neighborhood Data Request for that particular block which in turn means that 1) the size of data that might need to be transferred from another node (if that portion of FS_B data is not resident in the current node) is reduced and 2) by reducing unnecessary flanks, we also reduce the amount of FS_B data involved in data integration in each thread.

We now explain the specific scenarios where we can reject MRR flanks using the Border Index of an FS_A block. For the sake of simplicity, we will be explaining the scenarios in the spatial domain (2-dimensional) only. The concept could be extended to the spatiotemporal domain (3-dimensional) easily.

Scenario 1: The first scenario is illustrated in Fig.4.3, where the query polygon does not fully contain the geospatial bounds of a block, represented by the brown box. The area between the blue and the brown box represents the geospatial bordering region for the block, which is the maximum spatial relaxation region. As we can see from the figure, the polygon does not pass through the maximum relaxation region of the block on the eastern flank. Under these circumstances, we can see that even after including the spatial relaxation for the query polygon, it would still entirely lie inside the bounds of the block′s geospatial limits on the eastern side.

Hence, spatially, we can ignore querying the geospatial neighbors of the block that would lie to the North-East, East and South-East (as shown in Fig.4.3), since fetching FS_B records for those spatial regions would be unnecessary.

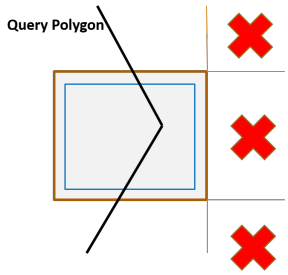

Figure (4.3) Neighboring Region Elimination: Scenario 1

Scenario 2: The second scenario is shown in Fig.4.4. Here, although the polygon covers a portion of the eastern maximum relaxation region and it seems like we might need to fetch data for the eastern geospatial flank, we can see that there are no data-points on this block that actually lies in the eastern maximum relaxation region. Hence, in such a case, we do not really need that eastern flank data.

Using the Border Index for the FA_A block, we can easily check if there is data in a particular flank and then use that information to further optimize the size of the neighboring region that needs to be queried for the block, thus reducing the amount of data to be fetched and possibly the number of neighboring nodes to be queried.

Hence for every FS_A block that matches a data integration query, using the Border Index of those blocks (like in the two scenarios mentioned above), we can further optimize the size of its corresponding MRR, thus reducing both data movement and record processing time.

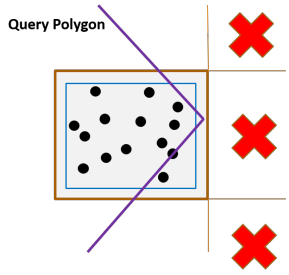

Figure (4.4) Neighboring Region Elimination: Scenario 2

4.3.5 Partial Block Processing Using Bordering Index

As mentioned before, by finding the orientation of an FS_A block with that of a neighboring FS_B we can further reduce the amount of data processed out of that FS_B block. Since blocks are defined by the geohash and the time range they cover, it is easy to find it one block from FS_B is a spatiotemporal neighbor of an FS_A block. In case an FS_B block, let us call it Blk_{Bj}, is a neighbor of the FS_A block, say Blk_{Ai}, not all of Blk_{Bj} needs to be processed. Rather the flank of Blk_{Bj}

that is in contact with Blk_{Ai}'s MRR. That region can be determined by finding out the orientation of Blk_{Bj} with that of Blk_{Ai}. The using the Flank Record Indices, we can find the particular record indices for that particular flank and only those FS_B records need to be processed and returned as candidate FS_B records for data integration.

Fig.4.5 shows the scenario where an FS_B block is a spatial neighbor of an FS_A block. We explain the scenario in the spatial domain only for simplicity. We can see that the FS_B block is the South-Western(SW) neighbor of the FS_A block. Hence spatially, the only flank that the FS_A block would need from this particular FS_B block would be the North-Eastern flank(NE) and only those records that lie in the NE Flank Record Indices of the FS_B block should be considered, intersected with whatever temporal Flank Record Indices are relevant.

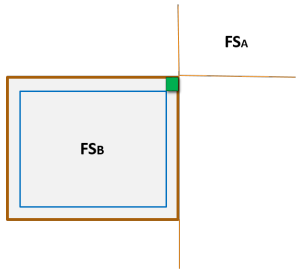

Figure (4.5) Partial Neighbor Block Processing

4.4 Feature Interpolation With Uncertainty

The feature interpolation occurs after the record merging operation has completed and is executed in the same thread as the segmented data integration operation (explained in 5.1.2). The interpolation strategy we use in our data integration operation depends on the dataset itself and how the data records are influenced by records around them. There is no one specific way to perform spatiotemporal interpolation - the Confluence APIs are extensible to different types of interpolation algorithms. Another feature we provide along with an interpolated value is an estimate of the amount of uncertainty associated with that interpolated value. In our work, we explore a few

common interpolation methods that we believed might be effective on the kind of dataset we are working with, which were mostly spatiotemporal sensor or survey datasets.

4.4.1 Vector-to-Vector Interpolation

In the vector-to-vector **interpolation** scenario, we try to predict the value of an FS_B feature at a spatiotemporal point for which we do not actually have an observation, but we have a collection of other points in its spatiotemporal neighborhood. In the result of the data integration operation, each point in FS_A will have a one-to-many relationship with a collection of FS_B points, since in most cases the spatiotemporal attributes of an FS_A will not find an exact match with an FS_B point.

Out of the many available interpolation strategies available, the strategy we adopt is that of the **Inverse Distance Weight (IDW)** [9]. The reason for picking this strategy is its assumption that the interpolated value would be dependent on observations recorded in its neighborhood, i.e. the value at a certain location is similar to and influenced by the observed values in its neighborhood, which is true for many weather-related or other atmospheric datasets, which are mostly the type of datasets we are dealing with.

In IDW, the goal is to estimate the value of a parameter (Z), at the unmeasured location (Z_j) based on finite set of measurements of this parameter at other locations (Z_i), using the following equation:

$$Z_j = \frac{\sum_{i=1}^{n} \frac{Z_i}{(h_{ij})^\beta}}{\sum_{i=1}^{n} \frac{1}{(h_{ij})^\beta}}$$

where, h_{ij} are the Euclidean distances between the target point and the observed point and β is the weighting power. As we can see from the equation, the weight or influence of a neighboring point diminishes the further away it is from the location of interpolation. The β determines the rate at which the influence drops with distance. So, the idea here is that the closer a point is, the more influence it has over the estimated value of a parameter.

The optimal β for a particular spatiotemporal interpolation point is calculated using machine learning. We also measure the **uncertainty** of the interpolated value, Z_j, by means of estimating the error with the machine learning model. This uncertainty value is used to train a model to predict the best beta value and the error rate. The training data for this model is collected using sample data from the dataset and estimating their feature value based on observations from neighboring points and finding the corresponding optimal beta and prediction error.

If the degree of influence of neighboring points is not dependent on the distance, we provide AUEDIN (explained in 6.6.6) as an alternate method for interpolation and error estimation, which involves finding weighted mean of the neighboring data points′ feature as an estimate of the feature value and using a weighted standard deviation to estimate the value of the error in the estimate.

4.4.2 Vector-to-Raster/ Raster-to-Vector

The interpolation methodology of the scenario Vector-to-Raster/ Raster-to-Vector is the same, because at the end of the data integration operation, we are left with a one to many map between a raster pixel and a collection of vector datapoints that lie in the extent of the raster area.

So, since each point in the rasterised dataset represents a spatiotemporal extent, we can assume the value of the variable to be uniformly the same in those bounds. Using interpolation in this context, we estimate the value of a parameter from the vector dataset for the entire spatiotemporal extent of the raster record using the vector points that were found as a match. These vector records are treated as observed samples of the variable over the entire raster region and using these sample observational values, our interpolation method predicts a value for the feature over the entire raster extent.

Although several interpolation strategies are available for raster interpolation, we have used one called AUEDIN (explained in 6.6.6). Similar to the vector-to-vector estimation, here also, provide an estimate of the uncertainty involved as a weighted standard deviation among the sample observations.

It is to be noted that in many rasterized datasets, the spatiotemporal bounds/ extent of each record is not explicitly specified. The above method will work only if there is a concrete way to determine whether a point lies within the extent of a rasterised record. For instance, if we consider a Census dataset at county level, just the county name is not enough for us to determine the spatial extent of the records.

4.4.3 Raster-to-Raster Interpolation

A raster-to-raster data integration is possible only if the spatiotemporal extent of the raster pixel of both participating datasets is well defined. This is because, in this case, any intersection in the relaxed spatiotemporal extent of target dataset′s data record with that of the source dataset is considered as a valid integration output and so, there has to be a way to check for intersection. Since intersection of extents is the only criteria for integration here, each record in the output from the target dataset will be mapped to one or more records from the source dataset.

Finding an interpolation strategy in this scenario that can be applied in a general sense is difficult. This is mainly because, in order to interpolate, we have to predict a feature value(from FS_B) that is representative of the entire extent of a record from FS_A, based on the FS_B records that intersect with it. To our knowledge, there exists no spatiotemporal interpolation algorithm that can interpolate the value of a raster pixel based on values from area of intersection with other raster pixels. There does exist methodology to interpolate a value for an entire raster pixel based on sample observations from that pixel′s extent, which was the case in case of raster-to-vector scenario, but is not applicable here. Due to this, we avoid interpolation in a raster-to-raster data integration setting.

4.5 Self-Adaptive Relaxation Conditions

The size of the relaxation region determines the records from FS_B that are used to determine the value of a feature at a spatiotemporal point as that of an FS_A point. The size of the relaxation region is directly proportional to the number of neighboring points used.

In Inverse Distance Weighing [9], we have seen that the nearest points have more influence on the interpolated value of a feature at a certain point than those at a further location. Hence, the influence that the points in the relaxation region would have on the interpolated value of a feature decreases with distance. We can think of these neighboring points as sources that have influence over the value of a feature at a given location [31]. The sources in IDW have weights inversely proportional to their distance from the interpolating point. By assigning weights based on the Euclidean distance of the points, we are minimizing the influence that the points further away have on the interpolated value.

The β value in the IDW formula determines the weighing power. It determines by what magnitude the influence of a source over the interpolated value decreases with distance. In our implementation, we have used a default value of the relaxation region and allowed the value of β to determine how the points further away get penalized. the higher the beta, the stricter the penalization with distance. We have thus, in our methodology attempted to find an optimum beta for a fixed relaxation region.

By Dynamic Relaxation Region, we mean that β value of the relaxation region is determined based on the spatial location and the time for the point on which the prediction is being made. Often the level influence of neighboring points on the interpolated value at a point varies with the spatiotemporal region we are trying to interpolate. In order to take that into consideration, we have used machine learning to predict the optimal value of beta based on the spatiotemporal point on which the interpolation is being done. Using a static value of β throughout the spatiotemporal domain of the query would give us inaccurate results from the interpolation operation if there is in fact varying impact of neighboring points based on their spatiotemporal location.

Hence we have implemented a one-time operation to generate training points for our machine learning algorithm and then use them to externally train a machine learning model to predict β, which could then be used during any data integration query to dynamically predict the value of β based on the spatiotemporal region it is dealing with.

4.5.1 Training Data Generation

There is no way to find the optimal β value for each point in FS_A during the actual data integration process since we do not know the actual value of the feature at the point in question. We have, thus, attempted to sample throughout a certain spatiotemporal range and generate a set of training points for our machine learning model. Each training point would have three input features - latitude, longitude, timestamp and one output feature i.e. β.

The training points are generated from a dataset that would be used as FS_B in a data integration query. Out of that dataset, we perform stratified sampling to get N points in total from all its blocks. We then perform data integration between those points in the block on the rest of the points in the block to get a set of neighboring points for each of those N points. For each of the points n_i in N, we use the neighboring points found to predict the value of the feature in question using IDW with different β values. We compare the predictions to find out which β (out of a set of pre-defined β's) yielded best prediction and generate a training point with the latitude, longitude and timestamp of the point n_i and β being the target variable. Using these training points, we hope to train a machine learning model to predict the optimum β.

The Training Data Generation operation is similar to the data integration operation, with the exception that it is an integration of a dataset onto itself and the relaxation parameters involved is the same as the maximum relaxation parameters set for that dataset.

4.5.2 Modelling β Value

We use the training dataset returned to train our machine learning model that takes an input vector of 3 features(latitude, longitude and timestamp) and predicts the value of β. For the purpose of supervised learning, we have used feedforward Neural Networks using Python$'$s SKlearn package.

The optimal model parameters are then printed as a JSON string, to be used as an input in future data integration operation. It is to be noted that for a spatiotemporal region, given the data

distribution is more or less uniform, both the training data generation and the model training is a one-time operation.

4.5.3 Dynamic β Prediction

Our data integration operation supports both using of a static β value or using a trained model to predict optimum β at runtime where, the JSON representation of the trained model is def in as part of the data integration. During runtime, using the trained model′s weights, we make a prediction of beta using each of the FS_A points′ spatiotemporal parameters. The FS_B points in its relaxation regions are then used to predict the feature value at runtime.

Although it is possible to make a prediction of β value for every FS_A point found, it would be a very time-consuming operation, since the prediction involves multiple matrix multiplication operations, the size of which depends on the layers of the trained neural network. Thus, to simplify the operation, we predict the β for the center-point of the spatiotemporal bounds of every FS_A block involved and use that β for every point in that block. Our experiments show that the error using this method and individual β for every point are quite similar.

Chapter 5

System Architecture

In this section, we explain the various components of data integration operation along with the services that make up the entire Data Integration operation. We also explain the services that go into creating the training data for model building. We will break up the entire operation and explain it in terms of the separate requests that get fired internally once a relaxed data integration operation is requested. Fig.5.2 shows a representation of the various requests and responses involved in a Galileo cluster for a single data integration operation.

5.1 Effective Data Integration

In Galileo, inside each node, the data is partitioned in blocks and each of these blocks is stored in a hierarchical directory structure so that they are grouped by the spatial and temporal characteristics of the data they store.

Typically if records from one dataset(A) are to be merged with that of another dataset(B), based on a set of joining attributes, each record in A needs to be checked against every record in B. This is a $O(n^2)$ operation, which is infeasible, considering the size of the data we are dealing with. Even if the data is partitioned into multiple nodes, using this naive strategy on these segments of data on each node is not feasible and would be very time-consuming, as the number of records in a single node is still quite large. This would also require a huge amount of data movement in between the nodes.

So, in order to optimize the operation of data integration in a distributed environment, we mainly attempt to reduce three aspects. First, we need to reduce the network bandwidth utilization by reducing the amount of data transferred in between the nodes. Second, we have to reduce the processing occurring at each node by reducing the number of records being read in for the integration and ensuring that only the records relevant to the integration get processed. Finally, we have to optimize the actual operation of integration of the candidate records from the two datasets.

Thus, the entire operation of distributed data integration consists of three main components/phases: data movement, blocks reading and record merging. Below, we explain the complications with each of these components and how we plan on overcoming these using our methodology.

5.1.1 Minimizing Data Movement

On closer inspection, we can see that in a zero-hop distributed hash table(DHT) based storage system, like Galileo, the data movement can be drastically reduced. This is because, if we think of each record individually, it needs to check for a match in its neighboring region in the spatiotemporal domain and not the entire dataset. To elaborate, in Fig.4.1, let us assume that we have a point whose geospatial position is somewhere in the geohash are *9z* and let us, for now, just focus on the spatial aspect of the integration.

For a point (say p_i) in FS_A lying anywhere in the region *9z*, the points from FS_B that we need to consider as candidates for a merge, for p_i, are those lying inside *9z*, along with its neighboring area whose thickness depends on the spatial relaxation that is desired in the integration. So, in Fig.4.1 that neighboring area is represented by the area between the bounds of the *9z* geohash (represented by the blue box) and the red box (let us call this neighboring region nr). Any other geospatial region should not be of any concern for integration involving point p_i. In a similar fashion, in terms of the temporal attribute, if p_i lies in the region for a particular date, as shown in the figure (29-04-2017), the records from FS_B that we should be interested in for integration should be the ones that cover this particular date and the timespan δt before and after that date, with δt being the amount of temporal relaxation that the particular query allows.

So, to summarize, the data integration of FS_A points lying in the region *9z* and *29-04-2017* are the datapoints from FS_B that lie spatially in the red box in Fig.4.1 and temporally in the region *29-04-2017* $\pm\delta t$.

The temporospatial partitioning of data over Galileo nodes ensures that record blocks are grouped by the spatiotemporal region they represent. So, to explain in terms of the example in Fig.4.1, if the dataset for FS_A is spatiotemporally partitioned in terms of 2 characters of geohash

among the Galileo nodes and by the day of the month respectively, all the data-points lying in the geospatial region *9z* and date *29-04-2017* will lie in the same node. Ideally, if FS_A and FS_B have the same temporospatial partitioning, the data for the same spatiotemporal regions lie on the same nodes in the cluster for the two datasets. This scheme ensures the least amount of data movement in between the nodes, since, in the worst case, we will only need to import data from the neighboring region nr spatially and $\pm\delta t$ temporally.

In the case that the temporospatial partitioning is not the same for the two datasets, records for the same spatiotemporal regions might not be co-located on the same node. But, since Galileo is a zero hop DHT, one of the main features of this system, that we can exploit in terms of data integration is that given a spatiotemporal range, we can easily pinpoint the node(s) that contain this particular chunk of the dataset needed and request that data from those particular nodes only, without needing to broadcast the neighbor data request throughout the cluster.

Hence, we can see that, due to the DHT-based temporospatial partitioning scheme in Galileo, we can reduce the amount of network traffic, firstly, by reducing the number of nodes that are needed to be queried for data from neighboring regions in the spatiotemporal domain for each block in FS_A. Secondly, as we will show in the following section, using the Galileo metadata graph and our Border Indexing scheme, we can also ensure that the neighboring data being sent back contains only those records that match the spatiotemporal constraints specified in the query.

5.1.2 Chunkified/Segmented Integration

Say, we have a node n_i in the galileo cluster holding a portion of the FS_A data in it (let us call it d_{1i}). The data on n_i covers some set of geohashes and a range of time based on the temporospatial partitioning specified for this dataset. Let′s say the corresponding datapoints needed for integration for FS_B are partly in N_i and the rest have to be fetched from a set of nodes in the cluster(let′s call this data d_{2i}). For the purpose of data integration, performing the operation on the whole of d_{1i} and d_{2i} in a single thread would be both time-consuming and resource-heavy.

Instead, in our system, we have split the entire operation of data integration into smaller segments, to be carried out as **independent** operations inside node n_i in separate threads. Doing so would greatly speed up the data integration process. We have kept the size of each segment the same as the geospatial coverage of a block of FS_A, meaning, the entire operation of data integration over a query polygon and time range will be split into smaller spatiotemporal bounds corresponding to that of a block of FS_A that may lie in the query space.

In each of the threads, the FS_A data being used is data from an FS_A block that satisfies the integration query and the FS_B records that get used are the ones covering the *relaxation region* of the FS_A block. These independent operations can be launched once all the data from the FS_A block′s relaxation region comes in, without having to wait for the remaining incoming FS_B data.

As mentioned before, this data for FS_B could either lie locally in the current node or might have to be fetched first using a Neighbor Data Request, which can be determined by referring the partitioning scheme for FS_B. So the cumulative output of joins from each of these segment, which are the size of an FS_A block should give us the desired output for our integration operation.

5.1.3 Minimizing Block Reading

As explained before, each of the Galileo nodes maitain a separate metadata graph for each dataset it stores. The metadata graph reflects the metadata of the contents of each of the blocks in a particular dataset. Inside each node, for each filesystem (dataset), each block contains data for a single spatiotemporal region in a hierarchical directory structure based on the spatiotemporal attribute of the block. Using the in-memory metadata for each of its blocks stored, once an ndr request for data from a particular filesystem (dataset) is received by a node, it can pinpoint the blocks that match the paticular query by querying the metadata graph, without having to actually go through the records in them. The candidate blocks can then be scanned with the actual spatiotemporal constraints in the query to get the records that are required to be sent back as response.

So, this is how the scanning of unnecessary blocks to get relevant data from FS_B is avoided using Galileo′s metadata graph. Later, we introduce a new in-memory data indexing scheme to help reduce the scanning of unnecessary records further.

5.1.4 Fast Record Merging

As mentioned before, once all relevant records have been fetched to one location, the operation of integration occurs in a parallel fashion in smaller independent chunks. In this section, we explain the methodology used to create the FS_A to FS_B map as the output of the data integration.

Now, our integration strategy should be suitable for both rasterised and point datasets, since both are supported by our system. A join between two datasets has the worst case complexity of $O(n^2)$. Since the data we are dealing with in each independent segment is still quite large, optimizing the data integration used also would help reduce the latency of the operation.

For relaxed data integration, we implement a Divide and Conquer Approach algorithm for similarity join. In our strategy, we implement a One-dimensional Divide and Conquer [32] [33] similarity join on the two datasets on any one of the three joining fields (latitude, longitude and time - we use time as the selected field for the similarity join) and then further filter the pairs of records selected as output of the similarity join by comparing the other two features for similarity. We explain the operation of One Dimensional Divide and Conquer approach below.

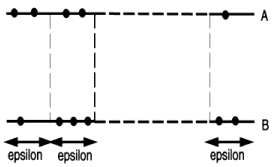

Figure (5.1) MDC Algorithm for One Dimension

In the One Dimensional Divide and Conquer approach, given two sets of n points on a line, we are to report all pairs of points, one from each set, within distance ϵ from each other, where ϵ represents the relaxation in that particular dimension set in the query. This is accomplished by

sorting both sets (a $O(nlogn)$ operation) and performing a scan of both sets by treating portions of each set corresponding to a range of values of the attribute of width 2ϵ. By using all elements with values in the range 0 to ϵ or ϵ to 2ϵ from both datasets, we have all the points necessary to get output from joining points in the 0 to ϵ range that are part of some joining pair. No more points are necessary, since any point that joins with a point in the range 0 to ϵ must be within distance 2ϵ from the left side of the 0 to ϵ range. Once we are done with the 0 to ϵ range, we can discard the corresponding partitions from the buffer pool and read the next range, 2ϵ to 3ϵ, to finish the processing of the ϵ to 2ϵ range, and so on. The algorithm is explained in the below pseudocode.

Algorithm 1 One Dimensional MDC

1: Given two one dimensional datasets A and B and ϵ
2: Sort Both A and B on the dimension
3: Read ${A_0}^{\epsilon}, {B_0}^{\epsilon}$
4: Check pairs in ${A_0}^{\epsilon}, {B_0}^{\epsilon}$
5: **for** $j = 2$ to $\lceil 1/\epsilon \rceil$ **do**
6: Read in ${A_{(j-1)\epsilon}}^{j\epsilon}, {B_{(j-1)\epsilon}}^{j\epsilon}$
7: Check for matches in ${A_{(j-2)\epsilon}}^{(j-1)\epsilon}, {B_{(j-2)\epsilon}}^{j\epsilon}$
8: Check for matches in ${A_{(j-1)\epsilon}}^{j\epsilon}, {B_{(j-2)\epsilon}}^{(j-1)\epsilon}$
9: Discard ${A_{(j-2)\epsilon}}^{(j-1)\epsilon}, {B_{(j-2)\epsilon}}^{(j-1)\epsilon}$
10: Check for matches in ${A_{(j-1)\epsilon}}^{j\epsilon}, {B_{(j-1)\epsilon}}^{j\epsilon}$

The above algorithm applies for similarity join in case of both point and rasterised data. In case of rasterised data, we do the sorting based on the lower limit of its bounds in that dimension and then the rest of the operation is similar.

5.2 Relaxed Data Integration Query

5.2.1 Data Integration Request

This is the initial request that a client sends to any one of the nodes in the cluster. We call this the Client node (let us call it N_{init}) in the operation. The main components of the Data Integration Query have already been discussed in section 3.1. The Data Integration Request supports both fixed and dynamic data interpolation parameters. The difference between the two scenarios is that,

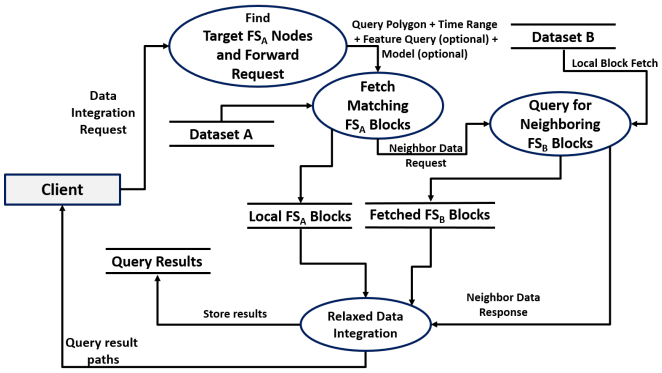

Figure (5.2) Data Flow in Data Integration

in case of dynamic interpolation, we are required to pass the weights of the layers of a trained neural network along with the biases in the form of a *JSON* string that can will be parsed into a java object for the corresponding model at runtime.

5.2.2 Data Integration Event

Once N_{init} receives a data integration request, it looks into the Galileo partitioning scheme for FS_A to see the target nodes, let us call them N_t that contains the relevant data for the spatiotemporal query domain specified in the Data Integration request. It then repackages the Data Integration Request into an identical request called the Data Integration Event that gets circulated to all the target nodes.

At each of the node in N_t, let us call it N_{ti}, for $i = 1$ to n, the Data Integration Event is then used to retrieve the query polygon, timespan and other feature constraints are used to finalize the FS_A blocks that are candidates for this particular data integration operation. Now for each of these candidate blocks, we construct a corresponding MRR domain signifying the spatiotemporal domain whose FS_B data it needs for integration. Now by accesing the partitioning scheme (since Galileo is a zero-hop DHT), we find the nodes that are needed to be queried to access the required FS_B data, which we will need to request using a Neighbor Data Request.

A mapping is done between all the nodes nodes to be queried for neighbor data along to the MRRs that require them, so that in the individual Neighbor Data Request that gets sent out, we only send those MRRs that are relevant to that node. Each node, N_{ti}, in N_t fires one or more Neighbor Data Request depending on how many nodes cover the cumulative spatiotemporal regions covered by all the MRRs on N_{ti}. The set of, say, m nodes N_{tij} for $j = 1$ to m, that get sent a Neighbor Data request by each node N_{ti} are called the neighboring nodes for the node N_{ti}.

Once a Neighbor Data Request is sent out by a node N_{ti}, it begins extracting and processing data records from all its FS_A blocks locally for the impending data integration operation.

5.2.3 Neighbor Data Request

A Neighbor Data Request is a request by each node in N_{ti} for the FS_B data it needs for its MRRs. Part of this data to be fetched could be resident in the current node,N_{ti}, i.e. N_{tij} might contain N_{ti} itself, while part of it might need to be fetched from other nodes in the cluster.

A Neighbor Data Request received at a neighbor node contains in a list of MRRs whose domain intersects with the total spatiotemporal partition handled by the node. Based on the query polygon and the relaxation parameter, spatially, an outer polygon is created by extending each coordinate outward by the spatial relaxation. Similarly, the temporal range of the actual data integration query is extended by the relaxation parameter. This is the spatiotemporal region beyond which no block from FS_B should be of any interest to the query.

5.2.4 Neighbor Data Response

We use this extended spatiotemporal query to find the blocks of FS_B on each node that are candidates. Now, not all of these blocks need to be of interest, since they might cover spatiotemporal regions for which there are no FS_A blocks.

On finding all the blocks for FS_B on a particular node that match the extended spatiotemporal query, the orientation of that block is checked against each MRR in the request to figure out which flank of this particular block is to be used by this MRR′s corresponding FS_A block. Only those data records are returned in a Neighbor Data Response, mapped to the corresponding MRR that

needs them. It is to be noted that multiple MRRs may request different flanks of the actual FS_B block. In case a candidate block found from the Neighbor Data Request is requested by multiple MRRs on the same requesting node, but only partially, only the flanks that are needed are returned and not the entire block.

Each neighbor node sends back two types of Neighbor Data Response - a control message(only one) and a data message (multiple). The control message is sent back once the MRR orientations are checked against the FS_B block orientations to figure out the blocks that would be sent back to the requesting node, i.e. it contains a summary of the block data that would be sent back as a reply from the neighbor node. This is sent back to the requesting node before the internal contents of the FS_B blocks are processed and helps the requesting node be aware of when all the requested blocks have been received from a particular neighbor node. The data message contains in it the actual blocks′ data for each path it processes. All of the FS_B blocks in a path(i.e. having the same metadata) on a node are processed in a separate thread and records are stored in the corresponding fragments and returned as a single Neighbor Data Response.

Any FS_B block that gets returned as part of the Neighbor Data Response can either be returned in a segmented fashion (in case all the MRRs need the entire block) or in a segmented fashion (in case any one of the MRR requires partial data from the block). This helps us reduce redundant data being sent back in the response and also avoids further processing to be done on the requesting node to figure out the part of the block data that a particular MRR needs. There are 28 possible segments to an FS_B block, each of which is stored as a list of records, each of which represents a segment of the block. In case the full block is not requested, not all of these 28 lists will be populated and if the full block is requeted, only one of the lists (the last one) will have entries in it. The 28 fragments are on account of a total of 27 spatiotemporal combinations (spatially 9 i.e. N, S, E, W, NE, NW, SE, SW, Center and temporally 3 i.e. up, Center, down) and one for the case if an entire block′s data needs to be returned.

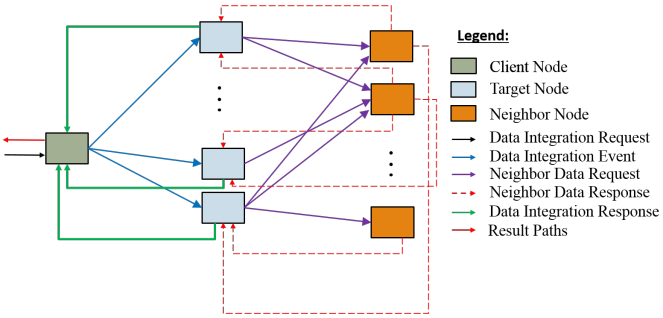

Figure (5.3) Process Flow in Data Integration

5.2.5 Data Integration Response

For every node, N_{ti}, each of the neighboring nodes it requests for FS_B data sends back a Neighbor Data Response which is a control message. The purpose of this control messages is to inform the requesting node of all the data fragments that are going to be subsequently transmitted by this particular neighbor, so that the requesting node, N_{ti}, knows when all the blocks/ block fragments have been received. The control message has information of the incoming paths along with which fragments of those paths are needed by which MRR. This is information that will be used to gather the FS_B data for a corresponding FS_A block.

Another purpose of the control message is that it helps to determine when an FS_A block is ready to be launched in a thread, i.e. when all its relevant FS_B data has arrived - there is no need to wait for other unrelated FS_B blocks. Each FS_A block can be launched in a separate thread once all its MRR FS_B data is received from the relevant nodes. By using the control message sent back from each of the neighboring nodes, we can determine when all the requirements of a MRR have been met. Each FS_A block is launched with its relevant MRR data and the results of each thread are stored in a query results directory. The Data Integration Response simply contains the path to these query result files, which are returned to N_{init} once all the threads have finished executing.

An FS_A block on a node is ready to be launched in its own separate data integration operation once the following two conditions are satisfied:

1. Once all the neighboring nodes of the MRR of the block have replied with their control message. This way, we keep track of all the block/ block fragments this particular MRR is expecting.

2. Once all the requirements of a particular MRR has been satisfied, i.e. every FS_B data block it expects, has been returned from their respective neighboring nodes.

As discussed before, by data integration, we mean both the act of integrating each FS_A record with similar FS_B records based on the relaxation parameters, as well as interpolating a feature value of an FS_B using neighboring FS_B data.

5.3 Generating Training Data for Neural Network Model

Here we provide a brief description of the requests and responses that get exchanged in order to generate the training data for our model. The user has to specify the dataset in question, the spatiotemporal region for which the model is being built and the number of training points desired in the response. The operation is similar to a data integration operation, the difference being that this involves only one dataset - we can think of this as a data integration of a dataset upon itself.

5.3.1 Training Data Request

As before, a single node N_{init} received the request for training data in the form of a Training Data Request. From N_{init} a Survey Request gets fired to every node that satisfies the spatiotemporal conditions of the query.

5.3.2 Survey request

A survey request computes the total number of records in the blocks on each node for a filesystem. Since this data is already stored in the block's border index, we do not have to actually read blocks to extract this information.

5.3.3 Survey Response

A Survey Response gets fired from every node that received a Survey Request recording each block in the node and total number of records in each block. Using this information, on N_{init}, we decide how many training points to extract from each block on each node.

5.3.4 Training Data Event

This is the request fired from N_{init} to each node with a map of each block in it to the total number of training points to be extracted from that block. On receiving the Training Data Event, for each block, we sample the specified number of random records as training point. Then we treat the rest of the block records as an FS_B record and perform an integration for neighboring points for each sampled records. For each sampled point, we attempt to predict the value of a specified feature at that point using its neighbors with various values of β and decide which one gives least error. For each sampled record, we generate a training point with 5 fields - latitude, longitude and timestamp, the optimal β and the error in prediction. Thus node generates a portion of the required training data and returns it to N_{init} in the form of a **Training Data Response**.

5.3.5 Training Data Response

N_{init} gathers all the Training Data Response and combines them into one single file locally at a specified directory. We can manually fetch our training data from there.

Chapter 6

System Evaluation

For evaluation of the performance of the Confluence system, we have tested out various aspects of a relaxed data integration operation such as its latency, throughput, model building and interpolation accuracy. In our experience the vector datasets we have worked with have the most data density compared to raster datasets and hence, for the purpose of system evaluation, we have mostly attempted integration in Vector-to-Vector scenario since this would be the most compute intensive, considering the number of points involved in each dataset. We have also attempted a Vector-to-Rasterised scenario to show the system latency.

6.1 Experimental Setup

6.1.1 Distributed Cluster Configuration

For our system evaluation, we have used a cluster where each node is an HP Z420 with the configuration of 8-core Xeon E5-2560V2, 32 GB RAM, and 1 TB disk. The cluster is set up in the form of 3 group-rings, with each group-ring being assigned 30 nodes each, thus making the total size of the cluster 90 nodes.

The requests can be fired from any external client node into any one node of the Galileo cluster and will get redirected to all target nodes from that node onwards by referencing the partitioning scheme and determining the target nodes.

6.1.2 Training and Testing of Predictive Models for Interpolation Parameters

The training data extracted using Training Data Request is externally trained on a single machine using Python′s Scikit Learn library [34], since the data size is fairly small and the number

of datapoints to be generated is determined by the user. The configuration of this machine is the same as that of a single node in the Galileo cluster.

We have trained an Artificial Neural Network available in Scikit Learn′s `neural network` package using the training data. The trained model is fed in to the data integration as a JSON string in the query. We have used a JSON parsing library in java to decode the JSON string and gather the neural network parameters for prediction purposes.

6.1.3 Datasets

Here we describe the datasets that we have used for our data integration evaluations. For vector-to-vector integrations, the first dataset we have used is sourced from the NOAA North American Mesoscale (NAM) Forecast System [35] and the second one is NOAA′s Integrated Surface Database (ISD) dataset [36].

The NAM dataset contains atmospheric data collected several times per day from stations all over the earth and includes features of interest such as surface temperature, visibility, relative humidity, snow, and precipitation. We have used only the data for the year 2015, for which, the cumulative size of the entire source dataset was $\sim$3.3 TB. The NOAA ISD dataset consists of global hourly and synoptic observations compiled from numerous sources all over the earth. Here also, we have used the data for the year 2015, whose cumulative size was $\sim$50 GB.

The NOAA and the NAM datasets had a spatial partitioning of 2 geohash characters, meaning, spatially, the domain of geohashes on earth of 2 characters were distributed among the cluster and a temporal partitioning of day of the month.

Now, since the data integration operation optimizes the data locality of the source dataset, we should choose the dataset with higher data density as the source dataset which should lead to lesser data movement and hence lower latency. Hence, in case of all our vector-to-vector data integration operations, we have made the NAM dataset, having higher data density, our FS_A and the ISD dataset as our FS_B.

For the purpose of vector-to-rasterised integration, we have used a rasterised dataset sourced from the model climate data provided by the NOAA Operational Model Archive and Distribution System (NOMADS) project which contains, among several other features, mean wind speed [37] and wind directions(which are the features we have extracted) recorded every hour. NOMADS is a repository of weather model that can provide near-real-time access to these weather model forecast data in addition to historical model data.

The vector data that we have used for this instance is a methane concentration data sourced from [38]. The total size of this dataset is 10gb and the reason we used this dataset in this scenario is that unlike the other two vector datasets that we have used, this does not have a uniform data density in terms of spatiotemporal coverage. The dataset is quite sparse because data is collected through equipment mounted on Google Street View Cars on specific days on specific neighborhoods. There is a high concentration of data records over spaecific cities over the U.S. over specific days. So we can see that between the wind and the methane datasets, there is a difference in spatiotemporal coverage, uniformity in data distribution as well as density.

The methane and the NOMADS datasets had a spatial partitioning of 4 geohash(considering geohashes over the U.S. only) characters and a temporal partitioning of day of the month.

6.2 Data Integration Latency Test

Our first experiment involves testing the latency of a relaxed a vector-to-vector relaxed data integration operation over a both a fixed β value and a dynamically estimated β value over the two vector datasets mentioned above. A fixed β means that we do not provide a trained model in the data integration query and use a fixed preset value of β for the interpolation operations.

We have fired 5 different types of requests (each 20 times), each with a varying spatial range to compare queries running over different spatiotemporal ranges - country level, state level, county level, city level and blank query. The queries differ in the size of the spatial polygon used as the query regions and they help us compare our data integration operations′ performance over different sizes of queries. The blank query involves a query for which no matching blocks were

found, since they queried spatiotemporal regions not covered by the datasets. All of these queries have a temporal range of a single day.

The country level query is the largest in size where we pick a random rectangular plot over the earth′s surface whose latitudinal range is 10° and longitudinal range is 16° and a random date, as the spatiotemporal query region. Similarly, the state level query has a lat-long range of 3.8° and 7°, the county level query has a lat-long range of 0.3° and 0.4° and city level query has a lat-long range of 0.075° and 0.15°. The spatial relaxation used is 0.1° for both latitudes and longitudes and the temporal relaxation is 12 hours, just to make sure that each FS_A point has to deal with a large set of neighborhood points.

6.2.1 Using Fixed β

Fig.6.1 shows the results from the experiment involving a fixed β. As we can see, the latency depends on the size of the query region, with the highest average latency being in case of country-level query (mean of 3.347 seconds). A box-plot also shows the upper and lower bounds of the latency for queries over different regions. Similarly, the latency for the state, county and city level queries are found to be 1.307s, 0.375s and 0.324s respectively. Blank queries return a response in 5ms.

The drop in latency with reduction of query region is mainly because of the lesser number of records that the data integration query has to deal with, along with the reduction in the number of neighboring nodes involved which leads to lesser data movement.

Fig.6.2 shows the breakdown of the three main components of a data integration operation - data processing and movement, record integration and interpolation. With the same setup as above, Fig.6.2 shows a breakdown in cost, averaged out over 10 runs each over the different sizes of spatial coverage as mentioned before. Before analyzing the results, it should be noted that the total time is much lower that the sum of the individual components, because the actual integration operation gets launched parallelly in individual independent threads once a source block has all the neighbor data it needs.

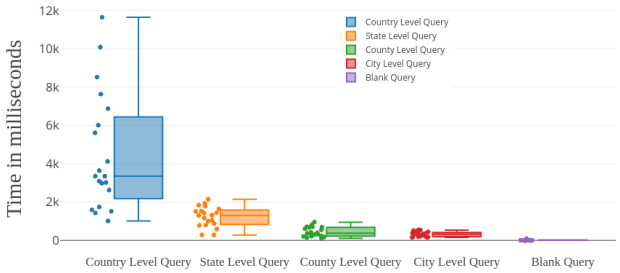

Figure (6.1) Data Integration Latency With Fixed β

We can see from Fig.6.2 that the cost of data processing and movement drops as the spatial range of a query drops, mainly because of the decrease in the amount of data that gets fetched from neighboring nodes, since, in most cases, a single node covers the spatial range mentioned in a query. Hence, the data processing and movement cost goes down drastically compared to the total time in case of county level and city level query. We can also observe that the data processing and movement part of the operation is the most costly part of the operation, which justifies our approach to make the data integration operation locality-aware.

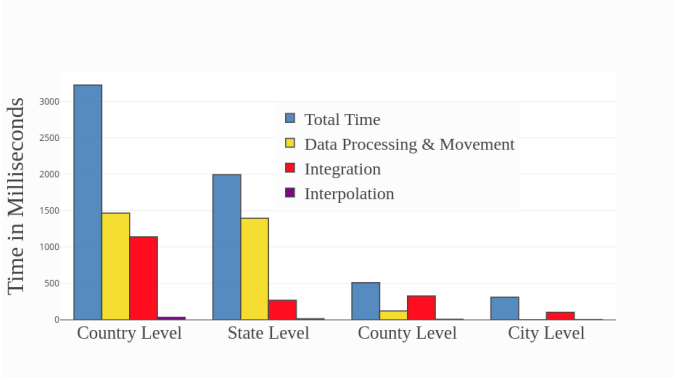

Figure (6.2) Latency Breakdown

6.2.2 Using Dynamic β

Fig.6.3 shows the latency of data integration queries with dynamic beta prediction. In this case, the only extra operation occurring is the prediction of an optimal β for the FS_A block only once before the interpolation operation, using our trained neural network model, which is essentially a series of matrix multiplication operations, the size of which depends on the size of the layers of the trained neural network. Hence, we can see that the latency is pretty comparable to that of the case with a fixed β.

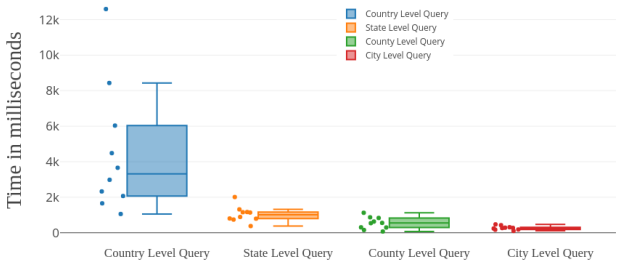

Figure (6.3) Data Integration Latency With Dynamic β Prediction

6.2.3 Vector-to-Raster Latency

Fig.6.4 shows the latency of different sizes of query on the methane sensor(FS_A) and wind dataset(FS_B). As we can see, the latency is pretty low in this scenario, even for a country level query. This is because of the spatiotemporal sparsity of the dataset. Since each block essentially covers a city on a particular day, there is very little difference in latency between county level and city level queries, since very few of the blocks have spatiotemporal neighbors.

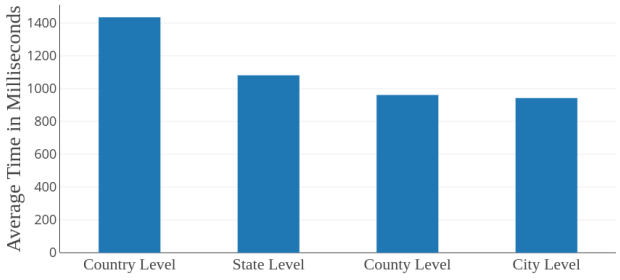

Figure (6.4) Data Integration Latency With Fixed β in Vector-to-Raster Scenario

6.3 Data Integration Throughput Test

Fig.6.5 shows the results from the throughput of the Confluence system for the different sizes of query mentioned above, except for a blank query. Here, we have simultaneously fired a 1000 requests over random spatial-ranges of a particular size into random nodes of the cluster to see how long it takes for the last query to finish. As we can see, the throughput increases for the query types that have lower latencies, with the country level having the lowest throughput of $\sim$43 requests/sec while the city level has the highest ($\sim$435 requests/sec).

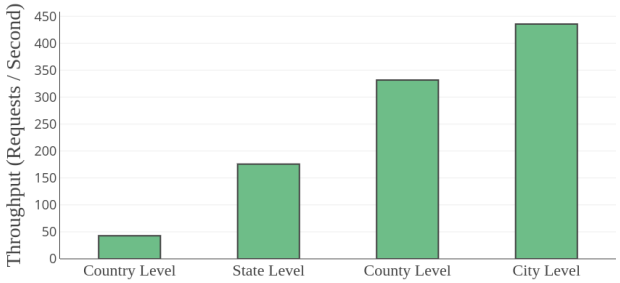

Figure (6.5) Throughput for Different Sizes of Query

6.4 Model Training and Accuracy

6.4.1 Model Building Time

We train an artificial neural network with the training data extracted from the FS_B dataset. The time taken to train that model is directly proportional to the size of the hidden layers of the neural network, along with the number of training points being used. In our experiments, we have used 5 separate combinations of hidden layers for the neural networks and picked the model with the combination that gives the least root mean squared error (RMSE). We have kept 10% of the input data aside for testing of the model and used the remainder 90% for training purposes. The time reported in Fig.6.5, is the average time per model, i.e. the total time taken divided by the total number of models trained, which is 5.

In Fig.6.6 we compare the time taken for the operation mentioned above with the increase in size of training data. It is evident that the training time increases with an increase in the number of training points, which is why it is better to train individual models for specific spatiotemporal regions and use those as inputs while running data integration operations on any subset of those spatiotemporal regions.

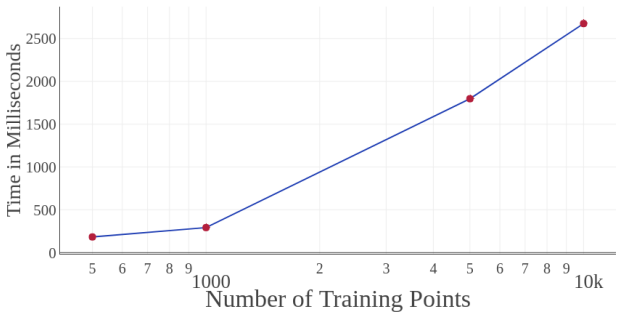

Figure (6.6) Model Building Time vs Training Data Size

6.4.2 Model Accuracy

We have also compared the accuracy of interpolation with a dynamic β using a trained model in the data integration operation as opposed to using a static β value. Now, to test out the accuracy of the dynamic β in interpolating the value of a feature at an unknown point, we have used a separate service that treats a single dataset as both FS_A and FS_B. For each block of a dataset, to be used later as FS_B in data integration operations, we randomly perform random stratified sampling for datapoints from blocks in the spatiotemporal range of the trained model. We then find the neighboring records from the same dataset to those sampled points and then use a dynamically predicted β to predict the value of a feature at the sampled point using the neighboring points and compute the error in the prediction by comparing it to the actual value. For each sampled point, we also make prediction using a set of pre-defined β values and record the errors.

The corresponding cumulative RMSE′s for different β used is shown in Fig.6.7. We have tested out the above scenario for three different features from the ISD dataset, namely *sky ceiling height, air temperature and atmospheric pressure*. We can see that the RMSE for cases with a dynamic β is clearly lower than those where a fixed beta is used.

The effectiveness of using a dynamic interpolation parameter also depends on how effective IDW is in the case of a particular feature. In case points in the spatiotemporal neighborhood of a certain point do not have influence over the value of the feature at the point of interpolation, using a dynamic β would not help the accuracy of prediction much. This explains the difference in the degree to which RMSE decreases using dynamic interpolation parameters in case of different attributes. We can see that the improvement in accuracy using a dynamic β is much higher in case of the features sky ceiling height and atmospheric pressure compared to that of air temperature.

6.5 Resource Utilization

In Fig.6.8, we show the CPU utilizations of different nodes in the cluster during a particular run of our data integration operation. We have tracked the CPU utilizations among 3 different participating nodes in the cluster from the start until the end of the runtime of the query. The query

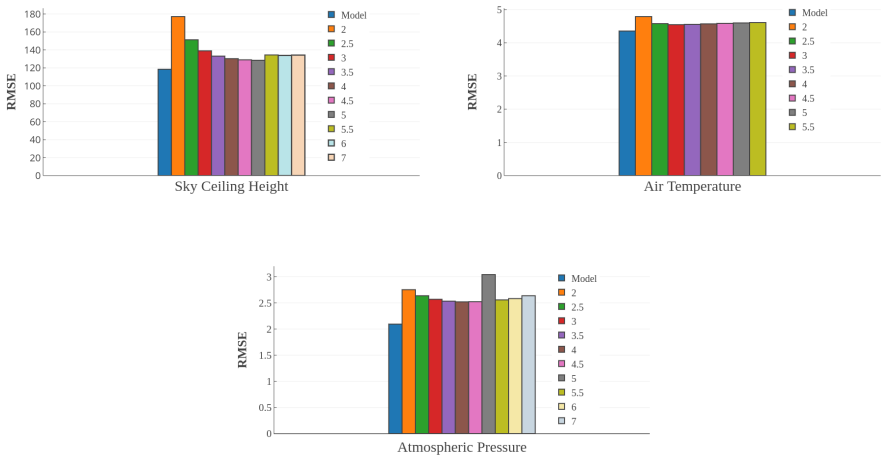

Figure (6.7) Accuracy Comparison - Dynamic β vs Fixed β

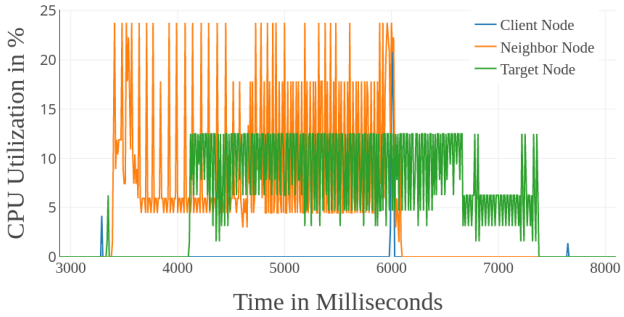

Figure (6.8) CPU Utilizations

used has a fairly large spatiotemporal range, to ensure a longer running time. The *client node* is the first node in the Galileo cluster that receives the query and in the case of this run, it does not contain any of the relevant data. So, the query gets transferred to the *target node(s)* that contain the relevant data. We have shown the utilizations of only one such target node. The *neighbor node* is a node that the target node has to query for neighboring data. It is to be noted that the neighbor node, in this scenario, itself is also a target node in the same query as it houses portion of the relevant FS_A data, so its performance graph also accounts for the local data processing and data integration it handles.

We can see from the graph that the client node has the least utilization of resources, since its only task is to redirect requests to relevant target nodes and gather the responses from those target nodes. The utilization of each node also depends on the number of blocks it has to process, which depends on the amount of the spatiotemporal domain of the query that falls under its jurisdiction.

6.6 Case Study - Obesity Prediction Using Integrated Data

Since our system′s main purpose is the integration of multiple spatiotemporal datasets, one of its main applications could be in the field of machine learning. As a case study, we trained a machine learning model using an integrated dataset with features from multiple domains to see if it results in an improvement in prediction accuracy.

In the following sections, we attempt to predict adulthood obesity from a child′s biometric, economic, environmental and familial attributes, which we collect by merging two separate geotagged datasets that cumulatively contain all those information [39]. The main focus in the following sections would remain on the use of the integrated data in machine learning.

6.6.1 Problem Description

Childhood obesity has received significant attention as an urgent health challenge [40]. According to the National Health and Nutrition Examination Survey(NHANES), obesity prevalence in 2007-2008 was 33.8%; this is twice as large as the prevalence rates in 1976-1980 and a 50% rise from 1988-1994 [41]. Worldwide obesity has more than doubled since 1980 [42].

The psychological, physical and economic consequences of obesity have been well studied [43]. Obesity costs are estimated to be as high as $147 billion per year, or roughly 9% of the annual medical expenditure in the United States [42]. Because childhood obesity often continues through adolescence and adulthood, an increased number of adults will be at a risk of chronic diseases that result from obesity [44, 45].

Providing effective and accurate predictions to obesity [46–48] is key to identifying causes and developing proactive strategies to prevent it. Recently, there has been growing recognition of

extensive factors such as the environment [49,50], genetic predisposition [51], and human behavior and their complex interactions [52, 53] as influencers of obesity.

Modeling change of obesity levels requires tracking the same individuals repeatedly over a long period of time, often over decades. Although existing longitudinal studies provide extensive and accurate observation of the changes and trends, exploring new attributes for a model is prohibitively expensive and challenging if those attributes have not been tracked in all surveys.

6.6.2 Overview of Approach

Our methodology for predicting obesity with extensive external factors involves:

1. integrating attributes from external datasets via attribute matching and data preprocessing,
2. calculating and preserving the quality of integrated attributes, and
3. evaluating multiple machine learning models to assess the effect of integrating external attributes.

To import attributes from a dataset with finer grained geospatial coverage (e.g. block-level), we aggregate those values based on the geospatially matching area (e.g. zip-code or county) to generate an approximated value.

To quantify the quality of integrated attributes, we introduce the concept of data uncertainty during geospatial integration. Unlike previously discussed data uncertainty measures in integration [54], data uncertainty in geospatial integration is defined as the likelihood that the approximation of the integrated attribute does not represent the dataset accurately. In this study, we estimate the information loss that results from data aggregation for each data point to be imported. To exemplify the effectiveness of the uncertainty estimate,we perform different model fitting algorithms such as Artificial Neural Networks, Gradient Boosting, and Random Forest with and without uncertainty attributes to contrast accuracy.

6.6.3 Target Variable

The Body Mass Index (BMI) is a measure of body fat, defined as the weight (in kilograms) divided by the square of the body-height (in meters). The range of the BMI is frequently used to determine a person′s obesity level [55, 56]. A person′s BMI could fluctuate over time due to several biometric, economic, environmental and familial factors. Building a predictive model for an individual′s future BMI requires us to track these aspects over a large span of time, which can be challenging.

6.6.4 Data Selection

In order to predict future BMI for children over the US, the datasets we consider as candidates for integration need to satisfy a few conditions. First, the dataset should have a large geographic coverage, as that would feed a wide range of individuals from different regions/demographic to the model and reduce bias. Second, the datasets should have intersecting attributes that would facilitate their integration properly.

We have explored several longitudinal datasets as candidates, which contained information for surveyed individuals relating to their biometric, economic, geographical, familial aspects, to name a few. The candidate datasets were National *Longitudinal Study of Youth 97 (NLSY97) [57], NEXT Generation Health Study (NEXT) [58], National Longitudinal Study of Adolescent to Adult Health (Add Health) [59] and US Census data [60].*

The NLSY97 was one of the largest datasets that tracks 8,984 cohorts since 1997, originating from 6,819 unique households, selected via screening. It provides a relatively large geospatial coverage(338 US counties) with well defined geographic information for each of its participants. Therefore, due to both its large number of participants and its geospatial coverage, we have selected NLSY97 as the primary dataset.

The AddHealth data, despite having a good geographical coverage, deidentifies the state and county codes for each candidate. On the other hand, the NEXT dataset provides precise geographic locations of its candidates using a zipcode. However, its geospatial coverage was unsuitable for

geospatial integration with other datasets with large coverage. On analysis, we found that in contrast to the 338 US counties that are covered by NLSY97′s participants, NEXT covers only 52, with only $\sim 12.5\%$ of the NLSY97 data-points intersecting geospatially with NEXT.

Therefore, we have selected the Census dataset to be our auxiliary dataset to explore factors that are not tracked in the NLSY dataset. Since participants of these surveys are not the same, it is difficult to merge two records based on aspects such as biometric, behavioral, economic etc. One possible aspect on which they can be merged is using their geographic location, if the dataset being merged with the NLSY97 contains environmental information of the area an NLSY97 participant hails from. The Census dataset was such a dataset that had the proper geographic coverage (all of US) and had environmental, economic,familial and demographic information for each location with resolution up to block-level and could be summarised to provide relevant information regarding a cohort′s area of origin.

6.6.5 Distributed Computing Environment

We leverage the Apache Hadoop framework [61,62] to provide scalable, fault-tolerant computing over a cluster of machines. We use Hadoop Map-Reduce to get an aggregate value for different features, along with their standard deviations over each county over the US from the block-level attributes in the Census data. Although Confluence is a much faster alternative to Hadoop, since the summarization and integration with Census dataset is a one-time process and since our focus in this case study is the applications of integrated data only, Hadoop Map-Reduce should suffice.

6.6.6 Interpolation - Attribute based Uncertainty Estimation for Geospatial Data Integration

Importing geospatial attributes from an external dataset introduces uncertainty due to the mismatch of characteristics such as geospatial coverage and/or temporal range. The imported data is often estimated using interpolation or aggregation algorithms. Our study focuses on datasets

with hierarchical geospatial demarcations such as political boundaries that are popularly used in longitudinal studies.

We propose a methodology that we call *attribute-based uncertainty estimation for data integration (AUEDIN)*, that estimates uncertainty for each imported value to quantify the risk of assimilating imported attributes for subsequent computations such as a model generation.

Suppose dataset A imports a set of attributes from dataset B over a matching geospatial attribute, k, that is included in both datasets A and B. If records in the imported data(from dataset B) has higher geospatial precision, the records with the larger coverage(from dataset A) geospatially intersect with multiple records (from dataset B). In such a scenario, we calculate an aggregate of attribute values (for all records in B that intersect geospatially with a record in A) and use this as the imported value in the integrated data along with uncertainty of these imported records using the following method.

Let dataset A be a set of m geospatial units, A_0, A_1,..., A_{m-1} and their corresponding attributes. For a set of attributes of the dataset A, $S_A = attr_{A_1}, attr_{A_2}, attr_{A_3}, ...$, the value of an attribute $attr_{A_i}$, at the geospatial unit A_j, is denoted as $val(attr_{A_i}, A_j)$. Similarly, for dataset B with n geospatial units, B_0, B_1,..., B_{n-1}, each geospatial unit is mapped to a set of attributes values,$S_B = attr_{B_1}, attr_{B_2}, attr_{B_3}, ...$, and the value of an attribute $attr_{B_i}$ of the geospatial unit B_j is denoted as $val(attr_{B_i}, B_j)$. Let us assume that a geospatial unit A_i overlaps with a set of n' units in B, $\{B_0,B_1,...,B_{n'-1}\}$.

We estimate the imported value from B using weighted data aggregation based on the sample distribution. Assume that geospatial units in B, B_0, B_1,..., B_{n-1}, contains sample counts, C_0, C_1,...,C_{n-1} respectively. We calculate the weights, $\{W_0,W_1,...,W_{n'-1}\}$ for each geospatial unit in $\{B_0,B_1,...,B_{n'-1}\}$ as,

$$W_k = \frac{C_k}{\sum_{j=0}^{n'-1} C_j}$$

Once we calculate weights and normalize them, the aggregated attribute value of the record for attribute B_r, to be integrated with record A_i, denoted by $val(attr_{B_r}, \{B_0, B_1, ..., B_{n'-1}\})$, is calcu-

lated using the following formula:

$$val(attr_{B_r}, \{B_0, B_1, ..., B_{n'-1}\}) = \sum_{j=0}^{n'-1} (val(attr_{B_r}, B_j) \times W_j)$$

Each aggregated value is delivered with the associated uncertainty estimate, calculated using weighted standard deviation as follows:

$$\sigma_{ri} = \sqrt{\frac{n' \times \sum_{j=0}^{n'-1} ((val(attr_{B_r}, B_j) - \mu_{B_{ri}})^2 \times W_j)}{(n'-1) \times \sum_{k=0}^{n'-1} W_k}}$$

where $\mu_{B_{ri}} = val(attr_{B_r}, \{B_0, B_1, ..., B_{n'-1}\})$ and σ_{ri} denotes uncertainty associated with merging A_i with aggregate value of the attribute B_r from dataset B calculated as above.

6.6.7 Preliminary Analysis

Our goal is to predict an individual′s potential obesity down the line, given certain available information about him/her in the current day. As a preliminary analysis, we have considered the growth charts proposed by organizations such as the Centers for Disease Control and Prevention (CDC) [63] and World Health Organization (WHO) [64] to estimate BMI, relying only on individuals′ biometric information- weight, height and age.

The growth chart is group of graphical measurements predicting the progression of weight and height among children from their birth till the age of 20 [63]. At any point in time, a child′s age and the BMI is plotted to a point on the graph. The percentile curve which is the closest to the point is used to estimate the growth pattern for child over the next few years as shown in Fig.6.9

We used the CDC 2000 growth chart for BMI on participants of NLSY97 to see how effectively it predicts the BMI N years from day 1 of their interview, compared to the actual measurements (we have tested for N=1,2 and 3) recorded in NLSY97. Table 6.1 shows the results of that test.

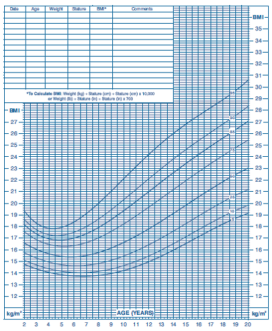

Figure (6.9) CDC growth chart of BMI progression with age for American boys aged 2-20 years.

Table (6.1) RMSE for BMI predictions(for year 1997) using CDC Growth Chart

	1 Yr	**2 Yr**	**3 Yr**
Male	3.52	3.38	3.40
Female	3.72	4.25	3.94

The prediction using the CDC growth chart demonstrates a sizable error range that can impact the detection of obesity. If the BMI is between 25 and 29.9, the individual is considered overweight, BMI of 30 or over is considered obese. We believe this happens because a child′s growth velocity can jump from one percentile curve to another during his/her developing years due to a number of external factors such as stress, family problems, genetic and chronic diseases. Our goal is to come up with a prediction model that can make better predictions of BMI by incorporating the growth chart data along with external factors from other available datasets.

6.6.8 Integrating Datasets Based on Geospatial Proximity

The external factors that we want in our input data along with the biometric information, are behavioral, economic, familial and environmental. The NLSY97 survey has a good variety of questions relating to various behavioral, familial and biometric aspects of an individual′s life but lacks a good collection of environmental and economic attributes. The US Census data consists of a set of records relating to the country′s demographic distribution,age, income, family structures

down to the block-group level. Using this dataset, we can form summary of areas of the country on factors such as median income, household size, percentage of population based on ages, family structures and so on. Thus in order to incorporate the missing aspects in NLSY97 data, we have considered the summarised US Census data for integration.

Between the NLSY97 and the Census 2000 datasets, the only common attributes suitable for a merge is the geospatial attribute. This, however, introduces a few challenges to the aggregation process. First, the maximum resolution of the Census datasets is at a block-group level, while NLSY97 is county level. Second, individual participant′s data is not available from Census, only block-level aggregates are available. Therefore, the best way to combine the two datasets is to use the county information of each individual. However, since we are trying to combine an individual′s record on the NLSY side with an aggregate of county-level information from the Census dataset, this will introduce some level of uncertainty in integrated Census information. As explained later, we have attempted to estimate this uncertainty and use it in our machine learning model to enhance its prediction accuracy.

Fig.6.10 gives a pictorial description of the mapping between the data-points of both the datasets. The circular regions colored red, green and blue represent three counties A,B and C. Now, on the NLSY97 side, there can be multiple participants who come from the same county whereas, with the Census aggregate, there would be a single record per county. Thus, geospatial integration between the NLSY97 and Census data would result in a many-to-one mapping.

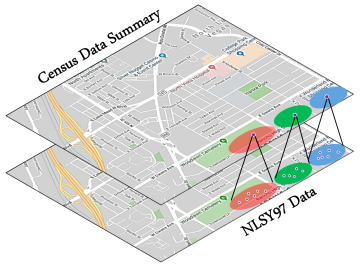

Figure (6.10) Strategy for merging Census 2000 & NLSY97 data

6.6.9 Data Pre-Processing and Feature Selection

The NLSY97 dataset was collected by conducting a series of interviews with each participant over a period of twenty years. The responses are a mixture of quantitative and non-quantitative values. Some non-quantitative attributes (e.g. race or gender of the person) may have a set of choices but have no numerical significance. We re-constructed non-quantitative responses as a set of propositions with associated boolean responses. This re-construction of non-quantitative questions increased the number of attributes: The NLSY97 data has over 67,000 attributes in total. This input vector was large enough to suffer from the *curse of dimensionality* [65].

We reduced the dimensionality of the input vector using the Lasso algorithm [66] and performed Gradient Boosting [67] and Random Forest [68] to explore important features. We avoided PCA to simplify interpretability of model.

6.6.10 Estimating uncertainty

In our case, the Census dataset provides higher geospatial precision(block-group level) than NLSY97 dataset (county level). Attribute integration from the Census dataset involves data aggregation to match the lower geospatial precision in NLSY97.

We have used our proposed approach, AUEDIN, to aggregate values and estimate uncertainty. The population information included in the Census dataset is used to capture the distribution of samples. If an imported attribute is selected, the model input vector contains the uncertainty estimate for that attribute. Uncertainty estimates for eliminated attributes are not included in the model.

Overview of Model Building: The input data to our predictive model went through the following stages. Initially, we started out with the NLSY97 data and used simply those features available in that dataset to make our predictions. In order to improve on this initial accuracy, we introduced a new feature to our training dataset- the prediction of BMI after n years, using the CDC growth chart given the stats of the individual at the current date. In the next phase, we combined the input dataset with the county-wise summary from the Census dataset and used that to train our

model. Finally, along with this data, we used a vector of uncertainty estimates for each selected feature and used it to train our model.

6.6.11 Uncertainty Aware Modelling

Since we only have an estimate of the error for integrated values from Census data, we tried out two approaches. First, we have used the integrated data, with potential errors as it is and applied machine learning models on it. In case the data has relatively low noise, the prediction algorithm might tune itself to the noise, thus becoming resistant. We have applied two commonly used machine learning models: *feed-forward neural networks*, to attempt deep learning and *gradient boosting*. We tried out gradient boosting here because, even if the data-noise affects a single model, combining multiple models as in ensemble methods, might help mitigate its effect.

In our second approach, we have incorporated the uncertainty estimates into our machine learning algorithm. Here, we would be working with two input vectors - one for the actual integrated dataset(d_{int}) and the other representing the error in measurement of each of the feature value in that integrated dataset (e_{int}). Evidently, both d_{int} and e_{int} must have the same dimension.

We assume that the features from the NLSY97 dataset would all have zero measurement errors, i.e. the values of variables acquired from interview of the participants are all assumed to be correct. Only the values of features derived and integrated from Census must have possible error associated with them. So, we can create an uncertainty matrix with the same dimension as the input vector, where all columns corresponding to NLSY97 features would have 0 values and only columns associated to Census attributes would have corresponding weighted standard deviation for that measurement.

For the purpose of modeling with uncertainty, we have used two different techniques to test which performs better. In the first approach, we have performed feature selection on d_{int} using Lasso Regression and then used the N(We have tried N=12,15,20) most important features, joined with the matrix representing their corresponding measurement error as an input vector to a neural network. In our second approach, we have performed feature selection as before, but this time

instead of clubbing two vectors into one input vector, we have used Maximum-Likelihood Principal Component Regression(MLPCR) [69], which takes the input vector and a vector of measurement standard deviations as its input.

The reason we have used MLPCR instead of PCR(Principal Component Regression) is that although PCR is effective for quantitative analysis of multicomponent mixtures, it relies on the Singular Value Decomposition(SVD) to obtain a reliable estimation of a p-dimensional subspace. When the measurement errors in the input are all *iid normal*, the p-dimensional hyperplane determined by SVD will be an optimal model for the data in a maximum likelihood sense. However, in case the measurement errors are not independent with uniform variance, the p-dimensional estimation will not be optimal. MLPCR provides two advantages. First, it allows inclusion of measurement uncertainties, assuming the errors are uncorrelated, into the calibration process. Second, it provides a maximum likelihood estimate of the PCA [70] model, which is generally superior to that obtained through SVD.

6.6.12 Experimental Evaluation

We have used separate models for predicting BMI for males and females. This is because two people with the same BMI can have very different body compositions. This is especially true when comparing males and females because women typically have a higher percentage of body fat than men. We have noticed an increase in accuracy with models trained on gender-segregated data.

Data Pre-processing:

We will be working with our datasets at 3 different stages. The first stage is simply the information available from the NLSY97 data, clubbed with the predictions from the CDC Growth Chart from year 2000. This was a simple data integration step where we appended a new column representing Growth Chart′s BMI predictions 3 years from the current interview date.

Data Integration:

In the second stage, we have integrated the data from the previous stage with the Census data. The calculations regarding summarization and uncertainty estimation and integration with the NLSY97 data were carried out in consecutive Map-Reduce tasks which were executed on Hadoop version 2.7.3 with the OpenJDK JVM, version 1.8.0 92. The Map-Reduce operations for this study were performed on a cluster of 20 HP Z420 servers (8- core Xeon E5-2560V2, 32 GB RAM, 1 TB disk).

6.6.13 Training and Testing of Predictive Models

In the final stage, we have used the Python′s Scikit Learn library [34] for most of our machine learning processes. Since the data at this point is relatively small in size(nearly 200 Mb), the training and testing phases are carried out on a single machine whose configuration is the same as that of each cluster node.

6.6.14 Scalability Evaluation

Due to the large size of the Census dataset and the NLSY dataset, by adopting Hadoop framework, we are able to profit from its scalability feature. For experimental purposes, we gradually adjusted the number of nodes in the Hadoop cluster, starting from 2 nodes, till 20. The result of the experiment is shown in Fig.6.11. Fig.6.11, shows that with increasing cluster size, the exe-

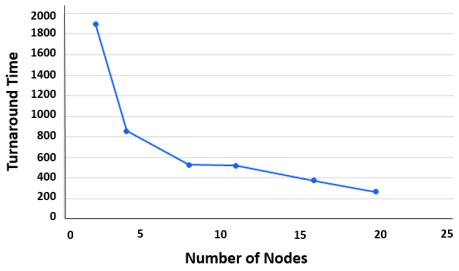

Figure (6.11) Turnaround time with increasing cluster size

cution time for the job decreased, meaning addition of machines to the cluster does improve the performance.

6.6.15 Experimentation and Accuracy Evaluation

Experiment 1 (Effect of data size):

In the first step of our experimentation, we wanted to check whether using greater number of data-points does actually improve the prediction accuracy, as that would mean that the training data is properly distributed. We have first taken the NLSY97 data, appended with the Growth Chart predictions and truncated and fed it to two different machine learning models(deep learning model and an ensemble model) in an incremental fashion using Scikit Learn′s `neural_network` package and `ensemble` package(for Gradient Boosting).

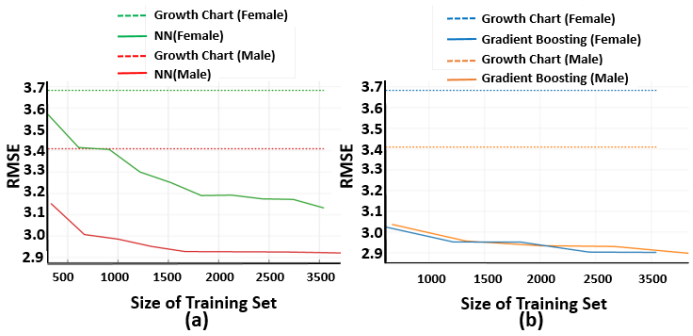

Figure (6.12) Prediction RMSE with change in training data size for (a) Neural Network and (b) Gradient Boosting Models

The data-set is first randomized and 10% of it is kept aside for testing purpose, so that training data remains constant and then the remaining 90% data points are fed as training data to a machine learning model incrementally to see if larger training data size helps improve accuracy. The RMSE we have reported in the Fig.6.12a is actually an average RMSE over 50 separate trials of training and testing the data.

As we can see from the Fig.6.12a, the prediction RMSE for both models for male and female BMI prediction goes down as the size of the training dataset increases. The dotted lines are for

when only the Growth Chart curves are used to make a prediction for males and females. Similar results have been observed when we have used an ensemble method(Gradient Boosting) as can be seen in Fig.6.12b.

Table (6.2) Results from Feature Selection on Integrated Data

	Selected Features
NLSY97	Predicted BMI Weight Of Participant(lbs) Is Participant Limited by Missing/Deformed Body Part Participant is Unhappy,Sad or Depressed Is Participant Limited By Sensory Problems Is Participant Limited By Mental Conditions
Census	Percent Families With Children Above 18Y/O Percentage of Seniors in Area Percentage of Married People in Area Average Household Size

Experiment 2 (Effect of Data Integration without Uncertainty Estimates) :

In the next step of our experimentation, we evaluate the effect of integrating new features from the Census dataset. Our expectation is that some of the features integrated from the new dataset could turn out to be important and thus enhance our model′s predictive capability.

Fig.6.13 shows the change in RMSE of BMI predictions with different models and input data. It is to be noted that the three bars(orange, green and red) in Fig.6.13 represent the results from doing a feature selection using Lasso Regression [66] and then training the top N features(we have tried with N=12,15,20) with an artificial neural network. Feature selection by Gradient Boosting by taking the most important 15 features and then applying artificial neural network gave similar results, but features selected by Lasso Regression were better. The RMSE values are the result of a K(=10) Fold validation on the input dataset.

As evident from the Figure6.13, inclusion of Census information does increase the prediction accuracy. The prediction RMSE for men comes down to $\sim$2.99 and that for women comes down to $\sim$3.23 in the case of Lasso Regression followed by training on neural networks. Also, from table

6.3, we can see that the prediction RMSE with integrated features using Gradient Boosting also decreases to $\sim$2.9 for male and to $\sim$3.06 for female.

Another aspect of this experimentation is to check whether the features integrated from the Census dataset are actually important enough to contribute to the BMI prediction. Table 6.2 shows a list of top 10 features that were selected using Lasso Regression in one of our experiments. We can see a mixture of attributes from both NLSY97 and the Census data summary make the list as the top influencers in BMI prediction. Also, some of the behavioral and biometric factors that were chosen as the top features intuitively make sense.

Experiment 3- Effect of Data Integration with Uncertainty Estimates

The final phase of our experimentation involves using the uncertainty information we had generated from the Map-Reduce task into our learning models to see if we get any improvement in prediction accuracy. In order to do so, we tried two approaches.The first approach is similar to what we did previously, i.e. we tried feature selection and then applied neural network. The difference is, here, along with the selected features, we appended the column representing the uncertainty value of that feature. For feature selection, we have found that using Gradient Boosting and taking out the top 15, along with their uncertainties gave us the best result. The uncertainty column is added only if it is for a feature relating to the geographically higher resolution dataset(Census), because the measurement uncertainty for NLSY97 is considered to be 0. In Fig.6.13, the red bar represents the output from this experiment and it shows that there is considerable improvement in terms of RMSE using this method both for male and female.

Table (6.3) Prediction RMSE for Gradient Boosting Models on Different Data

	RMSE(Male)	**RMSE(Female)**
NLSY+Growth Chart	3.0387	3.1999
NLSY+Growth Chart+Census	2.889	3.0599

In the second approach, we tried out the MLPCR model, which allows us to include information on measurement uncertainties in its learning process. However, as we can see from the

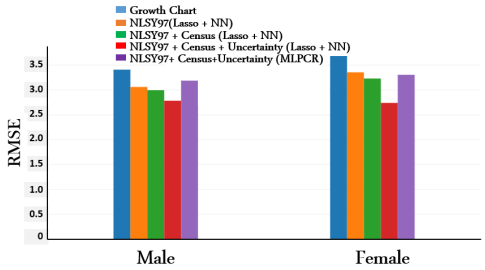

Figure (6.13) Comparison in RMSE for different datasets

Fig.6.13(last bar in the clustered bar chart), we do not get an improved RMSE. A possible explanation for this could be that MLPCR, which relies on Maximum Likelihood Principal Component Analysis(MLPCA) requires that the measurements have uncorrelated errors. Since the errors/weighted standard deviations are the same for each measurement for each county, the condition for uncorrelated error condition for each measurement is not satisfied.

To summarize, we see that, while the RMSE of prediction using Growth Chart was 3.4 and 3.68 for male and female respectively, our integrated training data was able to reduce RMSE to 2.99 for male($\sim$8.9% improvement) and 3.23 for female($\sim$12.3% improvement) and further incorporation of uncertainty measures took the RMSE to 2.78 for male($\sim$18.3% improvement) and 2.74 for female($\sim$25.6% improvement).

Chapter 7

Conclusions

We presented our methodology to efficiently integrate ancillary spatiotemporal datasets of varying resolutions into a single merged dataset. This merged dataset could be used for a variety of analytical applications such as visualization and predictive model building. As a case study, we have also demonstrated a practical application of such integrated data in the field of machine learning.

7.1 Research Question 1(RQ1)

Integration of misaligned data records among participating dataset is handled using the concept of spatiotemporal relaxation region where, for a point from the target dataset, any point from the source dataset lying in this region is considered as a similar and hence a match.

7.2 Research Question 2(RQ2)

Difference in resolution among participating datasets is handled by implementing separate integration and interpolation strategies in scenarios involving different combinations of vector and rasterised datasets as participants.

7.3 Research Question 3(RQ3)

The data integration operation is orchestrated in the form of independent sub-jobs for smaller spatiotemporal regions inside the spatiotemporal span of the actual query. Latency is reduced by minimising the data movement as well as the amount of record processed.

7.4 Research Question 3(RQ4)

Using machine learning, we have dynamically made predictions on the interpolation parameter, β, to alter the level to which neighboring points′ influence drops with distance from the interpolation point.

Bibliography

[1] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020, 2013.

[2] Ahmed Eldawy and Mohamed F Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1352–1363. IEEE, 2015.

[3] Haoyu Tan, Wuman Luo, and Lionel M Ni. Clost: a hadoop-based storage system for big spatio-temporal data analytics. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2139–2143. ACM, 2012.

[4] Jiamin Lu and Ralf Hartmut Guting. Parallel secondo: boosting database engines with hadoop. In *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pages 738–743. IEEE, 2012.

[5] Ahmed Eldawy, Mostafa Elganainy, Ammar Bakeer, Ahmed Abdelmotaleb, and Mohamed Mokbel. Sphinx: Distributed execution of interactive sql queries on big spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 78. ACM, 2015.

[6] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1071–1085. ACM, 2016.

[7] Louai Alarabi. St-hadoop: A mapreduce framework for big spatio-temporal data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 40–42. ACM, 2017.

[8] Jia Yu, Jinxuan Wu, and Mohamed 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*, page 70. ACM, 2015.

[9] Maciej Tomczak. Spatial interpolation and its uncertainty using automated anisotropic inverse distance weighting (idw)-cross-validation/jackknife approach. *Journal of Geographic Information and Decision Analysis*, 2(2):18–30, 1998.

[10] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 17–24. IEEE, 2011.

[11] Xiaochuang Yao and Guoqing Li. Big spatial vector data management: a review. *Big Earth Data*, 2(1):108–129, 2018.

[12] Geoffrey Fox, Sang Lim, Shrideep Pallickara, and Marlon Pierce. Message-based cellular peer-to-peer grids: foundations for secure federation and autonomic services. *Future Generation Computer Systems*, 21(3):401–415, 2005.

[13] Geoffrey Fox, Shrideep Pallickara, and Xi Rao. Towards enabling peer-to-peer grids. *Concurrency and Computation: Practice and Experience*, 17(7-8):1109–1131, 2005.

[14] Ahmet Uyar, Shrideep Pallickara, and Geoffrey C Fox. Towards an architecture for audio/video conferencing in distributed brokering systems. In *Communications in Computing*, pages 17–23, 2003.

[15] Geoffrey Fox and Shrideep Pallickara. Deploying the naradabrokering substrate in aiding efficient web and grid service interactions. *Proceedings of the IEEE*, 93(3):564–577, 2005.

[16] Thilina Buddhika and Shrideep Pallickara. Neptune: Real time stream processing for internet of things and sensing environments. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 1143–1152. IEEE, 2016.

[17] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Analytic queries over geospatial time-series data using distributed hash tables. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1408–1422, 2016.

[18] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, 2017.

[19] Walid Budgaga, Matthew Malensek, Sangmi Pallickara, Neil Harvey, F Jay Breidt, and Shrideep Pallickara. Predictive analytics using statistical, learning, and ensemble methods to support real-time exploration of discrete event simulations. *Future Generation Computer Systems*, 56:360–374, 2016.

[20] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, 2017.

[21] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Polygon-based query evaluation over geospatial data using distributed hash tables. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 219–226. IEEE Computer Society, 2013.

[22] Sangmi Lee Pallickara, Shrideep Pallickara, Milija Zupanski, and Stephen Sullivan. Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 573–580. IEEE, 2010.

[23] Alan E Gelfand, Li Zhu, and Bradley P Carlin. On the change of support problem for spatio-temporal data. *Biostatistics*, 2(1):31–45, 2001.

[24] INTAEK JUNG and KYUSOO CHONG. Interpolation and spatial matching method of various public data for building an integrated database. *WIT Transactions on The Built Environment*, 176:307–318, 2017.

[25] Maciej Tomczak. Spatial interpolation and its uncertainty using automated anisotropic inverse distance weighting (idw) - cross-validation/jackknife approach. *Journal of Geographic Information and Decision Analysis*, 2(2):18–30, 1998.

[26] Margaret A Oliver and Richard Webster. Kriging: a method of interpolation for geographical information systems. *International Journal of Geographical Information System*, 4(3):313–332, 1990.

[27] Houping Xiao, Jing Gao, Qi Li, Fenglong Ma, Lu Su, Yunlong Feng, and Aidong Zhang. Towards confidence in the truth: A bootstrapping based truth discovery approach. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

[28] Houping Xiao, Jing Gao, Zhaoran Wang, Shiyu Wang, Lu Su, and Han Liu. A truth discovery approach with theoretical guarantee. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

[29] Johnson Charles Kachikaran Arulswamy and Sangmi Lee Pallickara. Columbus: Enabling scalable scientific workflows for fast evolving spatio-temporal sensor data. In *Services Computing (SCC), 2017 IEEE International Conference on*, pages 9–18. IEEE, 2017.

[30] G. Niemeyer. *Geohash*, 1999. http://www.geohash.org/.

[31] Houping Xiao, Jing Gao, Qi Li, Fenglong Ma, Lu Su, Yunlong Feng, and Aidong Zhang. Towards confidence in the truth: A bootstrapping based truth discovery approach. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1935–1944. ACM, 2016.

[32] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[33] Nick Koudas and Kenneth C Sevcik. High dimensional similarity joins: Algorithms and performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):3–18, 2000.

[34] Scikit-learn.

[35] *National Oceanic and Atmospheric Administration, The North American Mesoscale Forecast System*, 2018. http://www.emc.ncep.noaa.gov/index.php?branch=NAM.

[36] *National Oceanic and Atmospheric Administration, Integrated Surface Database (ISD)*, 2018. https://www.ncdc.noaa.gov/isd.

[37] NOAA National Operational Model Archive and Distribution System. *NOAA U.S. Wind Climatology Data*, 2018. https://nomads.ncdc.noaa.gov/guide.

[38] Joseph C von Fischer, Daniel Cooley, Sam Chamberlain, Adam Gaylord, Claire J Griebenow, Steven P Hamburg, Jessica Salo, Russ Schumacher, David Theobald, and Jay Ham. Rapid, vehicle-based identification of location and magnitude of urban natural gas pipeline leaks. *Environmental Science & Technology*, 51(7):4091–4099, 2017.

[39] Saptashwa Mitra, Yu Qiu, Haley Moss, Kaigang Li, and Sangmi Pallickara. Effective integration of geotagged, ancillary longitudinal survey datasets to improve adulthood obesity predictive models. *IEEE Big Data Science and Engineering (BigData SE)*, 2018 in press.

[40] Cynthia L Ogden, Susan Z Yanovski, Margaret D Carroll, and Katherine M Flegal. The epidemiology of obesity. *Gastroenterology*, 132(6):2087–2102, 2007.

[41] National Center for Health Statistics (US et al. Health, united states, 2004: With chartbook on trends in the health of americans. 2004.

[42] Eric A Finkelstein, Justin G Trogdon, Joel W Cohen, and William Dietz. Annual medical spending attributable to obesity: payer-and service-specific estimates. *Health affairs*, 28(5):w822–w831, 2009.

[43] Julie A Gazmararian, David Frisvold, Kun Zhang, and Jeffrey P Koplan. Obesity is associated with an increase in pharmaceutical expenses among university employees. *Journal of obesity*, 2015, 2015.

[44] Neslihan Koyuncuoğlu Güngör. Overweight and obesity in children and adolescents. *Journal of clinical research in pediatric endocrinology*, 6(3):129, 2014.

[45] Sukanya Manna and Abigail M Jewkes. Understanding early childhood obesity risks: An empirical study using fuzzy signatures. In *Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on*, pages 1333–1339. IEEE, 2014.

[46] Christina Riedel and Rüdiger et al. von Kries. Overweight in adolescence can be predicted at age 6 years: a cart analysis in german cohorts. *PloS one*, 9(3):e93581, 2014.

[47] Kirsten E Bevelander, Kirsikka Kaipainen, Robert Swain, Simone Dohle, Josh C Bongard, Paul DH Hines, and Brian Wansink. Crowdsourcing novel childhood predictors of adult obesity. *PloS one*, 9(2):e87756, 2014.

[48] Xiaozhong Wen, Ken Kleinman, Matthew W Gillman, Sheryl L Rifas-Shiman, and Elsie M Taveras. Childhood body mass index trajectories: modeling, characterizing, pairwise correlations and socio-demographic predictors of trajectory characteristics. *BMC medical research methodology*, 12(1):38, 2012.

[49] Hedwig Lee, Megan Andrew, Achamyeleh Gebremariam, Julie C Lumeng, and Joyce M Lee. Longitudinal associations between poverty and obesity from birth through adolescence. *American journal of public health*, 104(5):e70–e76, 2014.

[50] Lisa M Hooper, Joy J Burnham, Rachel Richey, Jamie DeCoster, Mitch Shelton, and John C Higginbotham. The fit families pilot study: preliminary findings on how parental health and other family system factors relate to and predict adolescent obesity and depressive symptoms. *Journal of Family Therapy*, 36(3):308–336, 2014.

[51] Ruth JF Loos and A Cecile JW Janssens. Predicting polygenic obesity using genetic information. *Cell Metabolism*, 25(3):535–543, 2017.

[52] J Liang, BE Matheson, WH Kaye, and KN Boutelle. Neurocognitive correlates of obesity and obesity-related behaviors in children and adolescents. *International journal of obesity (2005)*, 38(4):494, 2014.

[53] Heather L Yardley, Jacquelyn Smith, Carolyn Mingione, and Lisa J Merlo. The role of addictive behaviors in childhood obesity. *Current Addiction Reports*, 1(2):96–101, 2014.

[54] Halevy Dong and Yu. Data integration with uncertainty. *Proceedings of the 33rd international conference on Very large data bases*, pages 687–698, 2007.

[55] Defining adult overweight and obesity.

[56] Defining childhood obesity.

[57] The national longitudinal study of youth 97 (NLSY97).

[58] Next generation health study (NEXT).

[59] National longitudinal study of adolescent to adult health (add health).

[60] Census 2000 data for the united states.

[61] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[62] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[63] Cdc growth charts.

[64] WHO growth charts.

[65] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning*, pages 257–258. Springer, 2011.

[66] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[67] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[68] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[69] Peter D Wentzell, Darren T Andrews, and Bruce R Kowalski. Maximum likelihood multivariate calibration. *Analytical chemistry*, 69(13):2299–2311, 1997.

[70] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.