A Framework for Profiling Spatial Variability in the Performance of Classification Models

Menuka Warushavithana
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA
menukaw@colostate.edu

Kassidy Barram
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA
kbarram@colostate.edu

Caleb Carlson
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA
cacaleb@colostate.edu

Saptashwa Mitra
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA
sapmitra@colostate.edu

Sudipto Ghosh
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA
sudipto@colostate.edu

Jay Breidt
Department of Statistics, Colorado State University
Fort Collins, Colorado, USA
fjay.breidt@colostate.edu

Sangmi Lee Pallickara
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA
sangmi@colostate.edu

Shrideep Pallickara
Department of Computer Science, Colorado State University
Fort Collins, Colorado, USA
shrideep@colostate.edu

ABSTRACT

Scientists use models to further their understanding of phenomena and inform decision-making. A confluence of factors has contributed to an exponential increase in spatial data volumes. In this study, we describe our methodology to identify spatial variation in the performance of classification models. Our methodology allows tracking a host of performance measures across different thresholds for the larger, encapsulating spatial area under consideration. Our methodology ensures frugal utilization of resources via a novel validation budgeting scheme that preferentially allocates observations for validations. We complement these efforts with a browser-based, GPU-accelerated visualization scheme that also incorporates support for streaming to assimilate validation results as they become available.

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; • **Computing methodologies** → **Classification and regression trees**; • **Mathematics of computing** → **Probability and statistics**.

KEYWORDS

spatial data, model validations, classification, visual analytics

ACM Reference Format:
Menuka Warushavithana, Kassidy Barram, Caleb Carlson, Saptashwa Mitra, Sudipto Ghosh, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. 2023. A Framework for Profiling Spatial Variability in the Performance of Classification Models. In *IEEE/ACM 10th International Conference on Big Data Computing, Applications and Technologies (BDCAT '23), December 4–7, 2023, Taormina (Messina), Italy*. ACM, Taormina, Italy, 11 pages. https://doi.org/10.1145/3632366.3632387

BDCAT '23, December 4–7, 2023, Taormina (Messina), Italy

ACM ISBN 979-8-4007-0473-4/23/12.
https://doi.org/10.1145/3632366.3632387

1 INTRODUCTION

Spatial data volumes have been increasing and account for about 80% of the total data being produced [11]. Spatial data can be characterized as data that includes location information for individual data items; this location information tends to be in the form of $< latitude, longitude >$ pairs. The increase in spatial data volumes is driven by the proliferation of low-cost sensors, networked observational equipment, and simulations that model natural (terrestrial, oceanic, environmental, atmospheric) and commercial phenomena.

Researchers rely on models to understand phenomena; such models are often used to complement decision-making. Models often need to be spatially explicit i.e., they need to be calibrated and their parameters tuned to account for subtle regional variations that impact model performance. Researchers are often interested in understanding where a model performs well and where it underperforms. A precursor to doing so is to assess the performance of models to identify their spatial variability, which may inform subsequent refinements.

The class of models we consider are classification models [16]. Classification performance can be assessed at different thresholds and there are often multiple performance measures of interest. We consider the case where a classification model exists, and the researcher is interested in profiling the performance of that model. Given a spatial area of interest, the contiguous United States (CONUS) in this study, the performance of a model needs to be assessed for the smaller spatial extents that comprise it. This can be based on administrative boundaries or domain-specific partitioning based on climactic, ecological, or topological characteristics.

The crux of this effort is to allow scientists to quickly assess performance variation of classification models across different spatial extents. These models may be domain-theoretic, process-based, or analytical models that are fit to the data using model-fitting algorithms. We also make no assumptions about the structural characteristics of these models i.e., they may be based on decision trees, deep networks, Bayesian properties, etc.

1.1 Challenges

Effectively profiling the performance of models over large spatial extents encapsulated by spatial datasets introduces several challenges. These include:

- Data volumes and I/O: Since the datasets we consider are voluminous the assessments can entail substantial disk and network I/O. Furthermore, since this I/O will occur in shared clusters, the increased I/O has knock-on effects for other co-located processes in terms of increased latencies and reduced throughput.
- The spatial extents under consideration may be large and irregularly shaped (defined using shape files) and aligned with political and administrative boundaries. Given that administrative boundaries are often agglomerative, it is common for shape files (e.g., cities and towns to be part of larger units such as counties and states).
- Interoperation with analytical engines. Researchers often have their preferred analytical engines and as such interoperation with diverse analytical engines should be supported.
- Models may have different parametrization schemes. The observations used for assessing model performance may entail preprocessing such as reconciling encoding formats and normalization.

1.2 Research Questions

As part of this study, we explore the following research questions.
RQ-1: How can we identify spatial variability in classification performance?
RQ-2: How can we accomplish such profiling while ensuring effective resource utilizations?
RQ-3: How can we support effective visualizations of performance variations? For classification models, researchers are often interested in these variations across different thresholds.

1.3 Approach Summary

To facilitate the identification of performance variation, we consider several aspects including partitioning datasets, devising a model assessment orchestration scheme, designing a scheme to alleviate assessment costs, and visualization component with a built-in declarative query scheme to facilitate explorations.

We partition the datasets into smaller spatial extents. The partitioning scheme we consider in this study is based on *shapefiles*, where each spatial extent is defined using an N-sided polygon with each vertex represented using a $< latitude, longitude >$ pair. In this study, we consider administrative boundaries comprising hierarchical units such as tracts, blocks, cities, counties, states, and finally the country level. Our methodology does not preclude the choice of other partitioning schemes such as geohashes, grids, or quad tiles. We posit that the shape file-based partitioning represents the most general and complex partitioning scheme and, in fact, any partitioning scheme can be represented using shape files. Each observation within the dataset is checked for inclusion within the polygon while accounting for the curvature of the earth – this is a one-time operation. We assign hierarchical prefixes to the shape files, i.e., each shape file has the prefix of its encapsulating shape file. This allows shapefiles to be identified deterministically based on the string assigned to them. Our partitioning also ensures that data for a given spatial extent are located on the same machine. Our orchestration schemes are designed to interoperate with diverse analytical frameworks.

Given a model, we create multiple instances from it – one for each spatial extent under consideration. Next, we *push* the model instance to the node holding the data for the spatial extent. The number of model instances being evaluated concurrently at a given instant is controlled to ensure effective resource utilization. Next, we construct confusion matrices for each spatial extent for a configurable number (default, 9) of thresholds. The true positive, false positive, true negative, and false negative rates for each model instance are used to estimate the AUC of the ROC, sensitivity, precision, and recall for different spatial extents under different thresholds.

Given that the datasets are voluminous, assessing the performance of models with every observation would be prohibitive. We design a novel validation budget scheme where a model is assessed with a fraction of the available data while ensuring statistical confidence in our results. Our methodology achieves this using a two-pass scheme where the performance of models is assessed with a small set of pilot observations. Additional observations are allocated to model instances (associated with specific spatial extents) based on the uncertainty associated with their model performance. The validation budget scheme allows us to work with a limited number of observations while preferentially allocating more observations to specific spatial extents.

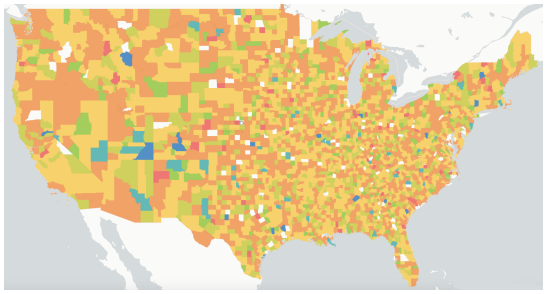

Figure 1: Choropleth map depicting the visualization of the validation response rendered across the contiguous United States. The response is visualized using a color-coding scheme where the difference in colors corresponds to different values in the validation response.

We complement these efforts with visualization. Model performance is rendered as a Choropleth map (see Figure 1) to allow a researcher to quickly assess performance variation spatially. A researcher can assess the performance of models across different

thresholds. The visualization scheme is backed by an implicit declarative query scheme that allows researchers to explore aspects such as the thresholds necessary to achieve desired performance or to identify spatial extents that have similar performance. Our visual interface is browser-based to facilitate visualizations across a variety of devices and GPU-accelerated to ensure responsiveness. Our visualization scheme leverages streaming to facilitate incremental visualizations, aid progress tracking, and amortize the costs associated with visualization.

1.4 Paper Contributions

Our methodology allows a researcher to assess the performance of classification models. Our specific contributions include:

- A framework for assessing the performance of classification models including different metrics, thresholds, and analytical engines such as PyTorch, Scikit-Learn, and Tensorflow.
- Our methodology is agnostic of the phenomena under consideration and the underlying structure of the model i.e., the model could be based on probabilities, decision trees, or complex networks.
- Our methodology identifies spatial extents where a model performs well and where it does not. The framework assesses several aspects of the model's performance across diverse spatial extents.
- Our interactive visual interface allows a researcher to quickly assess spatial variation of the model performance across different metrics. The visualization interface is coupled with a query interface that also allows a researcher to explore similarities, variations, and threshold-specific analyses. Our support for streaming in the visual interface allows interactivity and amortizes rendering overheads.
- Simplified interface for model assessments: A researcher submits models and our framework handles the complexity of creating model instances; apportioning a limited validation budget, and profiling model performance across different spatial extents with different thresholds all while ensuring efficient utilization of resources.

1.5 Paper Organization

The remainder of this paper is organized as follows. Section 2 includes an overview of related work. Section 3 describes our methodology while section 4 includes our performance benchmarks profiling several aspects of our methodology. Finally, in section 5 we outline our conclusions and future work.

2 RELATED WORK

With the increasing availability of large spatiotemporal datasets and advances in deep learning algorithms, deep learning has become a powerful tool for analytics and predictive modeling over spatiotemporal data [22, 35]. Deep learning models have been applied for inference and pattern detection over voluminous, high-dimensional, and heterogeneous spatiotemporal data. Specifically, deep learning is well-suited for classification over spatiotemporal data due to its ability to learn complex patterns and relationships between features automatically from voluminous data[9]. Deep learning models lead to improved accuracy more than traditional models due to their ability to learn complex non-linear relationships between features [6]. Deep learning models, such as convolutional neural networks (CNN), have been commonly applied to spatiotemporal data. Classification using deep learning over distributed systems is increasingly in demand for building and deploying predictive models over spatiotemporal data [44] such as satellite imagery to extract spatial and temporal features and classify various regions or objects for mapping, detection, and monitoring of natural phenomena[41].

Evaluating model performance in terms of accuracy is critical for the deployment of classification models over distributed systems. Efficient storage, querying, and evaluation of geospatial data using frameworks that support distributed querying and evaluation are necessary [43]. Models built over complex attribute relationships, as often is the case for voluminous spatiotemporal data, may have varying performance over different parts of the multidimensional data domain, making it challenging to accurately capture model performance using conventional validation techniques [2, 8]. Models may have a low capacity for generalization over portions of the data space due to multiple factors. For instance, the relationship between the input features and the output labels may be too complex for the model to capture accurately, leading to poor performance in certain areas of the data space. Alternatively, outliers, noise, or differences in the distribution of the input data across different portions of the data space can make it difficult for the model to generalize. Therefore, identifying sub-regions with low prediction accuracy is essential for increasing trust and interpretability, and statistical techniques have been developed to achieve this goal [21, 23, 36].

In classification problems, identifying sub-domains with low prediction accuracy is crucial for model performance evaluation [7, 10]. Various statistical techniques, such as those explored in [21, 23], can facilitate granular analysis of model performance and provide a measure of confidence in a model based on the region it's applied to. However, these techniques may require domain expertise and may not handle voluminous datasets. For large datasets, distributed evaluation techniques that partition the data space can efficiently evaluate model performance over the entire domain. Additionally, using a subset of the data to approximate accurate results can speed up model validation and increase the scalability of operations, which has been attempted for validating regression models in [5].

Scikti-Learn [33], TensorFlow [1], Keras, and PyTorch [32] are some of the most popular machine learning libraries. These libraries differ in capabilities, performance, difficulty of prototyping new models, deployment, and community support. Therefore, scientists utilize these libraries for model-building based on the problem that needs to be solved and the complexity of the relationship between the features involved [39, 45]. As a result, building a unified platform for model evaluation requires us to adapt and interoperate with these different frameworks, which rely on their own sets of configurations.

Approaches to virtualizing application workloads have been explored [25, 26]; in our work, the primary focus has been on ensuring data locality during orchestration. Data storage and query evaluations targeted specifically for spatiotemporal data have been explored in [4, 27, 28]. Similarly, in-memory approaches to data sketching [3, 34] have also been explored. Our methodology can interoperate with observations managed either on-disk, in-memory, or in hybrid situations where cloud-bursting is performed [29].

Finally, the framework is also suited for deployed in grid settings [14, 15].

3 METHODOLOGY

Our methodology to identify spatial variability in classification model performance encompasses a series of elements that include:

(1) A novel budgeting scheme that ensures that model validations can be performed without resorting to assessing performance with all available observations.
(2) A set of data wrangling operations to preprocess, partition, and store observations while ensuring support for data locality during model validations.
(3) Designing our framework so that it is amenable for interoperation with diverse analytical frameworks.
(4) Ensuring high-performance and scalability of the validation framework including containerization.
(5) Closing the loop for model validation with a browser-based, GPU accelerated implementation of an interactive Chroloplet map with support for streaming and generation of declarative queries to allow researchers to explore several aspects of validation including thresholds, precision, recall, and similarity of model performance.

3.1 Validation Budget [RQ-1, RQ-2]

Performing model validations with all available observations is prohibitively expensive in terms of the I/O, memory, and computing requirements. A key aspect of our methodology is to perform these validations while ensuring effective resource utilization. To this end, we have designed a novel validation budget scheme; in particular, the scheme relies on a fraction of the available observations to comprise the validation budget. Furthermore, this scheme is designed specifically for classification models.

The validation budget is constructed on the fly. In the pilot phase, a limited number of observations are extracted for each spatial extent. In the second phase, models for spatial extents with ambiguity in the model's performance characteristics are preferentially allocated additional observations subject to the requirement that the total number of observations allocated for validations does not exceed the specified threshold.

Consider a binary classifier that produces a continuous prediction x which is then compared to a threshold τ. If $x \leq \tau$, then the classifier predicts 0 (a failure or negative result), and if $x > \tau$ then the classifier predicts 1 (a success or positive result).

We wish to understand the prediction performance of the classifier in each of H regions and with each of G thresholds. We have a *validation budget* of $m + n$ test cases. The first m test cases are used as the *pilot test phase*, allocated to regions as $m = \sum_{h=1}^{H} m_h$. Let x_{hi} denote the continuous outcome of the classifier for test case i in region h ($i = 1, 2, \ldots, m_h$ and $h = 1, 2, \ldots, H$). Let $y_{hi} \in \{0, 1\}$ denote the true binary outcome for test case i in region h. Let $\tau_1, \tau_2, \ldots, \tau_G$ denote the G classifier thresholds.

The validation budget problem is then to use the information obtained from the pilot testing phase to allocate the remaining test cases as $n = \sum_{h=1}^{H} n_h$. The goal of the allocation is **to understand the prediction error** of the existing binary classifier, for H different regions and for G different test thresholds. Importantly, the goal is **not** to reduce the prediction error, because that would involve either rebuilding or retraining the classifier.

3.1.1 *Summarizing data from the pilot testing phase.*

We write $(A)(B) = 1$ if events A and B are both true and $(A)(B) = 0$ if either event is false. The binary classifier yields a true negative for case i in region h at threshold τ_g if $x_{hi} \leq \tau_g$ and $y_{hi} = 0$, so that with the above notation:

$$(x_{hi} \leq \tau_g)(y_{hi} = 0) = 1.$$

Using this notation, the pilot testing phase produces one 2×2 table (Table 1) for each of the G thresholds, with counts corresponding to true negatives, false negatives, false positives, and true positives.

Consider three scenarios that might be observed in the pilot testing:

(1) True negatives and/or true positives are large relative to false negatives and/or false positives, so that
$$\frac{m_{h00}(\tau_g) + m_{h11}(\tau_g)}{m_h} \simeq 1, \quad \frac{m_{h01}(\tau_g) + m_{h01}(\tau_g)}{m_h} \simeq 0.$$
In this case, we conclude that the prediction works well. Unless the pilot sample size is small, allocation of additional test cases is not worthwhile.
(2) False negatives and/or false positives are large relative to true negatives and/or true positives, so that
$$\frac{m_{h00}(\tau_g) + m_{h11}(\tau_g)}{m_h} \simeq 0, \quad \frac{m_{h01}(\tau_g) + m_{h01}(\tau_g)}{m_h} \simeq 1.$$
We can conclude that the prediction works poorly in this case as well. Unless the pilot sample size is small, allocation of additional test cases is not worthwhile (because the classifier will not be revised).
(3) Neither the proportion of failures (false negatives plus false positives) nor the proportion of successes (true negatives plus true positives) is close to zero or one. In this case, prediction is difficult, with the most difficult case being both proportions close to 0.5.

The preceding scenarios suggest that the final phase test cases should be allocated as follows:

- allocate less to regions with **either** highly successful or highly unsuccessful predictions;
- allocate less to regions with large pilot sample sizes m_h;
- allocate more to regions with **neither** highly successful nor highly unsuccessful predictions;
- allocate more to regions with small pilot sample sizes m_h.

After the pilot testing phase (the first round of validations) is complete, we proceed to the second phase of our allocation scheme.

3.1.2 *Allocation of the remaining validation budget.*

Allocation with one threshold. First, consider the case of a single threshold ($G = 1$). An allocation rule that satisfies the above criteria can be motivated as follows. If the prediction error in each of the H regions was the result of a binomial experiment with $m_h + n_h$ trials and success probability p_h = probability of successful prediction in region h, then the initial sample could be used to estimate p_h as

$$\widehat{p}_h = \frac{m_{h00}(\tau) + m_{h11}(\tau)}{m_h}$$

Table 1: Summary of data recorded from the pilot testing phase of our allocation scheme. The table represents a confusion matrix that uses a custom notation.

	True Zeros	True Ones
Predicted Zeros	true negatives $m_{h00}(\tau_g) = \sum_{i=1}^{m_h}(x_{hi} \le \tau_g)(y_{hi} = 0)$	false negatives $m_{h01}(\tau_g) = \sum_{i=1}^{m_h}(x_{hi} \le \tau_g)(y_{hi} = 1)$
Predicted Ones	false positives $m_{h10}(\tau_g) = \sum_{i=1}^{m_h}(x_{hi} > \tau_g)(y_{hi} = 0)$	true positives $m_{h11}(\tau_g) = \sum_{i=1}^{m_h}(x_{hi} > \tau_g)(y_{hi} = 1)$

and the optimal allocation of the remaining sample, to minimize the variance of the estimated prediction error in each of the H regions (subject to the overall validation budget constraint) would be

$$n_h = (m+n)\frac{\{\widehat{p}_h(1-\widehat{p}_h)\}^{1/2}}{\sum_{h=1}^{H}\{\widehat{p}_h(1-\widehat{p}_h)\}^{1/2}} - m_h,$$

provided $n_h \ge 0$. If n_h is negative, it indicates that too many cases m_h were already allocated to region h in the pilot phase, and no further cases should be allocated. In this case, we drop region h from further consideration and re-allocate to the remaining regions. Alternatively, it is possible to solve the constrained optimization problem:

For $h = 1, 2, \ldots, H$, choose $n_h \ge 0$ to minimize

$$\sum_{h=1}^{H}\frac{\widehat{p}_h(1-\widehat{p}_h)}{m_h+n_h} \text{ subject to } n = \sum_{h=1}^{H} n_h.$$

Multiple thresholds. If there is more than one threshold, we first define the estimated success probability at each threshold,

$$\widehat{p}_{gh} = \frac{m_{h00}(\tau_g) + m_{h11}(\tau_g)}{m_h}.$$

Next, we determine importance weights across thresholds, $\{w_{gh}\}$. Assume that $w_{gh} \ge 0$ and $\sum_{g=1}^{G}\sum_{h=1}^{H} w_{gh} = 1$. For example, this could be assigning equal weight to all thresholds and all regions ($w_{gh} = 1/(GH)$); or equal weight to the best threshold in each region ($w_{g'h} = 1/H$ for the threshold $\tau_{g'}$ with largest $\widehat{p}_{g'h}$ and $w_{gh} = 0$ for $g \ne g'$), or some other weighting decided in advance.

The proposed allocation is then

$$n_h = (m+n)\frac{\sum_{g=1}^{G} w_{gh}\{\widehat{p}_{gh}(1-\widehat{p}_{gh})\}^{1/2}}{\sum_{g=1}^{G}\sum_{h=1}^{H} w_{gh}\{\widehat{p}_{gh}(1-\widehat{p}_{gh})\}^{1/2}} - m_h,$$

provided $n_h \ge 0$, with suitable modifications as noted above if n_h is negative.

3.2 Data Wrangling [RQ-2]

The efficiency of our validation methodology relies on utilizing data locality, where we push the model to the data to minimize data movement across nodes and reduce network I/O operations. Geospatial datasets are available in a variety of formats including Shapefiles, GeoJSON, Gridded Binary, GeoTIFF, etc. First, we convert the selected dataset (or a subset of the data) into a format that suits our database of choice. We also process and store geospatial vectors representations (shapefiles) of the United States at varying spatial resolutions (states, counties, and census tracts), which are used on the frontend application for rendering results. Then, the individual records of the dataset are mapped to their corresponding shapefiles based on the coordinates (⟨latitude, longitude ⟩pairs) using a primary key. After ingestion into our distributed database, we shard the data based on the same primary key, such that multiple records that belong to a single spatial extent are co-located. Additionally, we utilize replication in the distributed database to ensure high availability.

We also maintain metadata about where the data associated with each spatial extent resides. The metadata table, stored at the Coordinator node (see section 3.4.1), is used for routing the validation requests to the node that contains the needed data points.

3.3 Interoperation with Machine Learning Frameworks [RQ-1]

We implement functionality to validate classification models trained on three popular analytical engines: Scikit-Learn, Tensorflow, and PyTorch. The validation engine is implemented using object-oriented design where we have an abstract class that defines functionality to validate classification models. The abstract class is extended for each library/framework. Using common abstraction and helper methods aids with reducing code duplication and maintaining a consistent API. An additional advantage of this design approach is the ease of extending support for new machine learning frameworks.

In the concrete implementation of the classes associated with each framework, we make use of framework-specific functions available for validating classification models. Classification metrics such as the confusion matrix, AUC (Area Under the Curve) of the ROC (Receiver Operating Characteristic) curve, sensitivity, and specificity for a number of thresholds are calculated using the functions provided by each framework.

For a given model, the only information required to support validation are 1) inputs and outputs, 2) whether the data need to be normalized or encoded in a specific format, and 3) the serialized representation of the model. The operation of the validation service does not depend on the internal structure of the submitted model. This provides researchers with the ability to validate arbitrary and complex models using our system.

3.4 Ensuring high-throughput Validations [RQ-2]

Our framework is implemented as a distributed overlay comprising a *coordinator* overseeing and tracking multiple *workers*, with a *proxy server* providing a RESTful API for clients. Our software stack interoperates with Python 3 to ensure seamless integration with a diverse range of data science, machine learning, and analytical frameworks. This decision was made to support the maximum

number of research applications possible. As our inter-component communication framework, we leverage Google's Remote Procedure Call (gRPC)[42], which gives us the flexibility of defining message and service types easily in `protobuf` files alongside space efficient (un)marshaling.

The validation server is designed to perform a large number of validation operations at once in a distributed cluster of nodes. Our validation server consists of two main types of nodes: *Coordinator* nodes that handle and accept new jobs, perform aggregation of results, and coordinate the message flow of the system. The validation jobs are carried out at *Worker* nodes using the data stored on local disks. The number of *Worker* nodes is much greater than the number of *Coordinator* nodes. We designed the system to operate on multiple *Coordinator* nodes that reside in a failover configuration i.e., only one *Coordinator* node will be active at a given instance (the primary) while the other *Coordinator* nodes are dormant in a standby (secondary) configuration, ready to take over if the primary *Coordinator* node fails.

3.4.1 ***Coordinator.*** The (primary) coordinator oversees an overlay of worker nodes and tracks their respective metadata, including the data and specific spatial extents they store locally. We organize this metadata in a radix tree [24] for fast lookups, insertions, and hierarchical aggregation for spatial extents at different resolutions. We consult this data structure to track ongoing jobs, and their completion statuses with respect to individual workers. The coordinator endpoint itself is implemented as a gRPC server with multiple services relating to registration, job submission, and steering. These services sit atop a thread pool executor, allowing concurrent processing of incoming requests.

Upon receiving a gRPC validation job request from the proxy, the coordinator infers any necessary fields based on the validation budget specified, and routes it to the appropriate job execution function that handles load-balancing. Load-balancing is done using a round-robin scheme with respect to spatial extents, resolutions, and locations of tracked workers. Individual jobs are launched at workers while preserving data locality. Tasks comprising jobs are submitted in a non-blocking, asynchronous fashion using Python's `Asyncio` library. Model instances are launched while ensuring parameterizations along with specifying the data query configuration and model framework. Similar to the communication between the proxy and the coordinator, communication between the coordinator and workers uses gRPC.

3.4.2 ***REST API.*** The communication between the primary coordinator node and the client application is mediated through a REST (Representational state transfer) API [31], which is essentially a proxy server that intercepts requests from the client. A request sent by the client is intercepted by the REST service and sent to the primary coordinator, which then formulates a new validation job and passes it onto the necessary worker nodes based on the data needed to complete the validation job. The worker nodes carry out the validation jobs and return the results back to the primary coordinator. The coordinator aggregates the results, launches a second round of validation jobs if required (see section 3.1) and sends the results to the client through the REST API. The REST API also provides several other functions, in addition to serving clients with a RESTful API to send HTTP/s requests. Client requests are sent as HTTP multipart/form requests with two parts: a JSON *request* string and a *model* file. The JSON request supports specifying a range of optional fine-grained controls like datastore read configurations, validation metrics, features, labels, and input normalization schemes. Together, the request and the file constitute a custom gRPC message that is forwarded to the *coordinator*. The REST API is amenable to incremental extensions such as Role-based Access Control policies [37], user-specific budget allocations, load-balancing across different coordinators, etc.

3.4.3 ***Workers.*** The worker nodes are also responsible for discovering the spatial extents IDs and the associated data upon startup. The entirety of the worker state resides in memory.

In addition to spatial metadata, a worker maintains a shared thread- and process-pool executor for handling incoming jobs. Multiple incoming jobs can be processed concurrently using multiple threads, and within a job, multiple child processes are forked to validate the model on each of the spatial extents in the request. Since model validation is both CPU-intensive and I/O intensive, some performance was gained by using multi-threading, but only on the I/O side of things as Python's Global Interpreter Lock (GIL) prevents two threads from running simultaneously. Thus, forking the child process allows each to have its own GIL, providing substantial improvements in terms of both CPU and I/O concurrency. We created the size of the process pool to match that of the available CPU cores on the system and ensured that the child processes were being recycled between validation runs to eliminate process creation overhead. For incoming worker job requests, the model's binary image is saved once to disk for persistence and to allow for loading by the model's framework from within the confines of a child process. Since the validation model is not re-entrant, we can load a single instance and execute inference concurrently between child processes by using managed shared memory.

The worker is responsible for (1) accessing observational data, (2) reconciling normalization schemes, (3) parameterizing model instances, (4) generating inferences, and (5) contrasting model outputs with ground truth data.

Once a job enumerating the spatial extents for which the model should be profiled is received, the worker extracts the model file from the request and saves it to a dynamically created directory based on the unique job ID. Each analytical engine, like TensorFlow or PyTorch, has its own model representation format. Our framework can reconcile models that are stored in a diversity of formats and initialize the execution environment for the spatial extents under consideration.

We refer to our wrappers around individual modeling frameworks as *validators*. Frameworks currently supported include TensorFlow, PyTorch, and Scikit-learn, but this list can be easily expanded in the future. Under the hood, validators use the modeling framework library APIs to load and execute the model, but this first requires data to be loaded for a given spatial extent. We have designed our framework to be extensible to support interoperation with data storage frameworks. Our connectors can support targeted data retrievals from relational datastores and NoSQL-based document stores such as MongoDB. The retrieved data can be loaded into efficient formats using Pandas/Numpy to be fed into the model.

Next, the worker creates an instance of a validator with the request parameters and launches the validation job. A shared process pool executor is used with a fixed pool size to launch concurrent inference jobs. The number of simultaneous child processes that are forked in this case is tied to the number of physical cores available. Child processes are recycled into the shared executor that retains the imported libraries for future executions reducing duplicate initialization overheads.

Profiling measurements are performed on a per spatial extent basis. They include tracking system metrics such as resource utilization and completion times. The results are streamed back to the coordinator which applies a final aggregation or allocates a new budget (see section 3.1) for a final pass before routing the results back to the client.

3.4.4 ***Database.*** The database is maintained in a replicated and sharded configuration to ensure the availability of data. In our reference implementation, we have used MongoDB as the primary data storage service in our setup. However, any database that supports distributed replication and sharding can be used for this purpose; for e.g., Cassandra, BigTable, etc.

Our dynamic web-based client takes input from the user which will send a request to the validation server and initiate a validation job. The user has the ability to configure the job using multiple options such as (1) model framework: Scikit-Learn, PyTorch, or Tensorflow, (2) spatial resolution: state, county, or census tract, (3) feature/target fields of the dataset, and (4) Validation metric(s).

Communications among the system components is shown in Fig. 2.

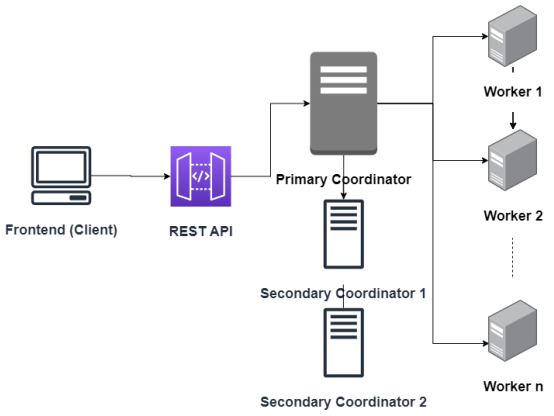

Figure 2: High-level architecture of the entire system including the frontend, validation service (coordinator and workers), and database.

3.4.5 ***Containerizing the Validation Service [RQ-2].*** Containerizing an application provides several benefits such as consistent execution across different platforms and architectures, fast scalability and replication, and high availability using container orchestration. We containerize our validation service using Docker [12] and Kubernetes [13] as follows.

- *Coordinator* and the *REST API* service running in dedicated pods
- Replicated *Worker* pods running on different hosts using Kubernetes `DaemonSets`

Containerization gives us the ability to quickly deploy our backend servers in cloud services such as Amazon Web Services and Google Cloud using the same set of configurations used in our regular deployment. Another benefit of containerization is that it allows us to set limits to the hardware resources (CPU, memory) allocated to the validation service. In the Experiments and Results section (4), we benchmark the differences in performance of the validation service running in a containerized environment vs. bare-metal execution.

With the second round of validations completed (using the remaining validation budget), the validation job comes to completion. At this point, the Coordinator node collects all results from the Worker nodes and compiles a response.

The response sent back to the client contains the following information for a given spatial extent:

- ROC Graph
- Area Under the Curve (AUC) of the ROC
- Precision and recall for thresholds ranging from 0.1 to 0.9 in 0.1 increments

3.5 A GPU-accelerated Dashboard [RQ-3]

The researcher is provided with a visualization-powered dashboard.

On the front-end application, a nested hashtable is used to store the model performance results streamed from the validation server. This data structure was chosen because it supports, on average, constant time performance metric retrieval for certain user-configurable queries. In particular, the use of this data structure allows for this time efficiency for the query type that determines which recall/precision value was produced for a given threshold. For this query, this structure provides, on average, O(1) look-up time. For the query types that return a given threshold that produced a recall/precision value which is greater than/less than or equal to the user-specified value, this data structure has an upper bound of 9 look-ups needed to return the corresponding threshold to the user, if there are no collisions in the hashtable.

To visualize the validation response on the client, a mapping framework was utilized. The particular mapping framework chosen was Deck.gl [38], which utilizes GPU acceleration in order to speed up rendering and execution on the client. The way this mapping framework leverages the GPU is by creating WebGL buffers, which are then uploaded into the GPU. The framework was chosen because Deck.gl is specifically designed to be optimized for rendering dynamic datasets while being highly customizable. This customizability allows for the highly specific coloring of the validation metrics across the map which aids the user in interpreting the validation job results.

Due to the resource-intensive nature of model validations, an entire validation job may take anywhere from several seconds to several minutes to fully complete. A rendering-only-after-completion scheme may frustrate users while awaiting results, leading to poor user experience and disengagement. To remedy this, we implemented result-streaming on the server; as soon as the model has been evaluated on a spatial extent, the corresponding results are streamed back to the client. With this approach, a user is able to see

results arrive in real-time, giving consistent visual verification of their validation job's progress. With an interactive choropleth map on the client side, this appears as spatial extents continuously being colored according to the selected metric's place in the heatmap, until the whole choropleth map has been assigned values. The complete validation response visualized across the map is shown in Fig. 6.

The front-end application additionally provides the ability to assess similarity relating to performance in a multidimensional space using k-Means clustering [20] based on a user-configurable selection of performance features. Upon the receipt of the last performance result, the k-means algorithm automatically executes using a default selection of performance features. The default features consist of the Area Under the Curve (AUC) of the ROC, and the recall and precision values for the thresholds 0.1 and 0.2. The k selected for the clustering algorithm is equal to $\sqrt{n}$, where n is equal to the number of spatial extents being evaluated; this k value was selected based on the standard outlined in [19]. The complete set of performance features the user is able to select are all precision and recall values associated to each threshold as well as the Area Under the Curve. The user is able to select up to 6 of these features to be used as input to the k-means clustering algorithm.

The frontend application also supports the following declarative queries for further analysis by the user.

- Find the threshold which produces the highest/lowest precision/recall values for each spatial extent
- For each spatial extent, find the minimum thresholds where precision/recall is greater than/less than a specified value

4 EXPERIMENTS AND RESULTS

Our empirical benchmarks profile several aspects of our methodology. In particular, we profile how well assessments performed using our validation budget scheme contrast with validations performed over the entire dataset. We are also interested in assessing the performance of these validations and how they interoperate with different frameworks. Finally, we profile our GPU-accelerated visualization scheme that can be used by researchers to explore multiple aspects of the validation space.

4.1 Experimental Setup

The hardware specifications of our experimental setup are as follows: a cluster of 50 machines, each with an 8-core CPU running at 2.10GHz, 64 GB of DDR3 RAM, and 5400RPM hard disks. Three out of the 50 machines were configured as a MongoDB `ReplicaSet` to manage the sharded/replicated database configuration, and three machines were used to deploy Coordinator instances (one primary node, and two secondary nodes). The remaining 44 machines contained the data shards locally and a single worker process was deployed in each which is responsible for model validation jobs. A sharded, replicated MongoDB cluster was set up across these machines.

4.2 Models

We use three classification models to evaluate our system. The first is a Decision Tree Classification model implemented in Scikit-Learn. The second is a 2-layer, fully-connected neural network implemented in PyTorch, and the third is a 2-layer, fully-connected neural network implemented in TensorFlow. The models were trained as binary classifiers using a subset of our main dataset (section 4.3). The subset of the data chosen for model training is different from the subset we use for validating the models. The empirical results demonstrate the suitability of our methodology to validate classification models trained using multiple frameworks.

4.3 Datasets

Our experiments are conducted on the publicly available North American Mesoscale Forecast System (NAM) dataset provided by the National Oceanic and Atmospheric Administration (NOAA) [40]. It contains data derived from a weather forecasting model with respect to a number of weather phenomena including temperature, pressure, visibility, wind, snow, humidity, and cloud coverage. The NAM model produces multiple grids of weather forecasts over North America at several horizontal resolutions. We use a subset of the data ranging from 2010 to 2015 (about 150 GB in size) and convert them from the gridded-binary (GRIB) format to JSON. Each point in the resulting data is then geo-tagged with U.S. county identifiers. We also use county-level shapefiles (geospatial vector representation) produced by the U.S. Census Bureau [30] and map the geo-tagged data with their respective counties' shapefile representations. The county shapefiles are used on the front-end to produce choropleth maps.

4.4 Backend Metrics

The time taken to complete validation jobs based on the three types of classification models created for the machine learning frameworks we support are shown in Table 2. Furthermore, it shows how the completion times have decreased with the use of our novel allocation scheme. We observed a 31% average decrease in time taken to complete the jobs across the frameworks while using the allocation scheme.

The total number of observations available in the subset of the NOAA NAM dataset we selected is about 84 million spread across all (about 3100) U.S. counties. When using the allocation scheme, we set the total validation budget (maximum number of observations used to complete the job) to be about 10% of all available observations. Out of the selected number of observations in the validation budget, we set aside 70% for the pilot testing phase and the remainder for the second of round validations (see section 3.1). The percentage decrease of precision for each county while using the allocation scheme compared to using all observations is shown in Fig. 3. The average decrease in precision we observed is about 7% for the 3044 U.S. counties. Thus, we conclude that our custom allocation scheme is capable of achieving very similar results while utilizing a fraction of the original data.

Fig. 4 indicates the average CPU usage of the cluster throughout a validation job for a PyTorch model in two scenarios: (1) using our allocation scheme, and (2) using all available observations. Validation using the allocation scheme is executed in two stages – the pilot testing phase and the second phase. For this experiment, based on the results of the pilot testing phase, 762 out of the 3044 counties were selected for the second phase. Overall, with the use

Table 2: Validation job completion time for each machine learning framework. Times are compared with and without the use of our validation budget (allocation scheme).

	End-to-end completion time (s)	
Framework	**Without validation budget**	**With validation budget**
Scikit-Learn	114.75	42.17
PyTorch	135.41	62.30
Tensorflow	203.12	110.26

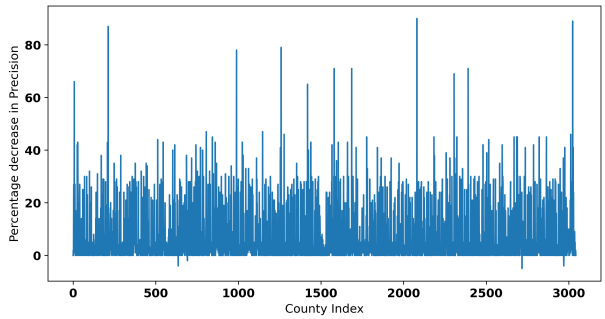

Figure 3: The percentage decrease of Precision values for a validation job (for a Scikit-Learn Classification model) for each county when using our allocation scheme vs. using all available observations.

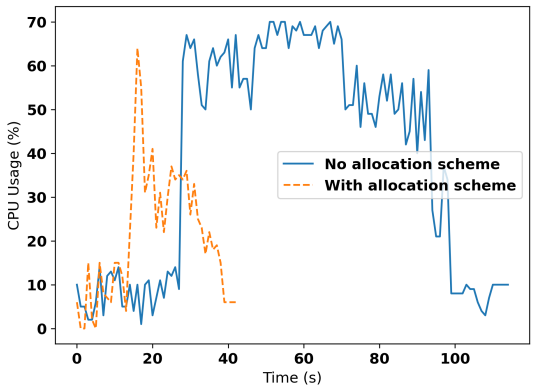

Figure 4: Average CPU usage of the cluster for a validation job with and without the allocation scheme.

of the allocation scheme, the CPU and memory usage of the cluster were reduced by 46% and 19% respectively.

We profiled the resource utilization by a validation job in a regular (bare-metal) deployment and a containerized deployment of our backend. The overall memory utilization was reduced by 9% (see Fig. 5) and CPU utilization was reduced by 4%. However, the overall completion times were increased by about 12% in the containerized deployment compared to the regular deployment.

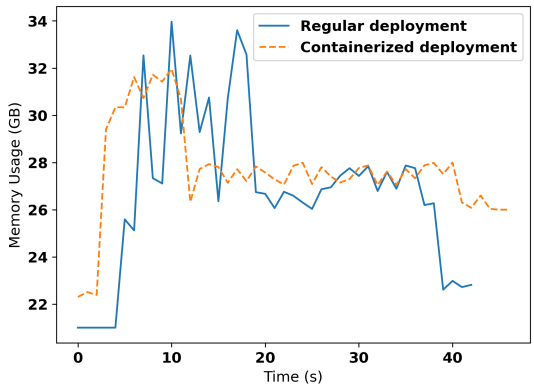

Figure 5: Average Memory usage (per node) of the cluster for a validation job in regular and containerized deployments.

4.5 Profiling Visualization

In order to assess the impact of using a mapping framework which leverages the GPU, we used a qualitative approach that focused on the quality of user interaction with the site. To achieve this, we conducted user experience audits to ascertain how well the site performed in accordance with the RAIL model [17]. The RAIL model was developed by Google Chrome to determine the performance of a web application from the user's perspective when performing essential site interactions. Chrome has additionally created a suite of tools that can be used to measure the performance of a site based on the guidelines outlined in the RAIL model. In particular, Chrome has developed a site performance auditing tool called Lighthouse [18]. For a given site, Lighthouse mimics a mid-range device with a slow internet connection and measures the responsiveness a user experiences when interacting with a site. Lighthouse measures several aspects of a site, but the main metric which highlights the importance of GPU acceleration is Total Blocking Time. Total Blocking Time is the total time a user experiences the site as being non-interactive to their supplied inputs (clicking, toggling, scrolling). This metric is pivotal in this assessment because it measures the amount of time the CPU-bound main thread is blocking the user from interacting with the site due to its execution of tasks. In this case, the tasks being executed are re-coloring tasks which are a result of the user-supplied queries.

The audits conducted using Lighthouse showed that the GPU-accelerated mapping framework had an average of 0.2% Total Blocking Time when the user is constantly performing queries. By comparison, the CPU mapping framework had on average 41.0% Total Blocking Time when the user continuously supplied queries. The GPU utilization allows for this re-coloring task to be moved from the CPU-bound main thread to the GPU which accelerates the color re-calculation and reduces the amount of time the user experiences blocking. Additionally, the GPU mapping framework optimizes this re-coloring further by not re-rendering the polygons that represent the distinct spatial extents on the map. In order to support the same user queries, the CPU mapping framework must re-render the shapes with every supplied query. This reduction in blocking time observed in the GPU implementation is crucial in order to

provide users the ability to use the tool without the frustration of experiencing tremendous web application non-interactivity.

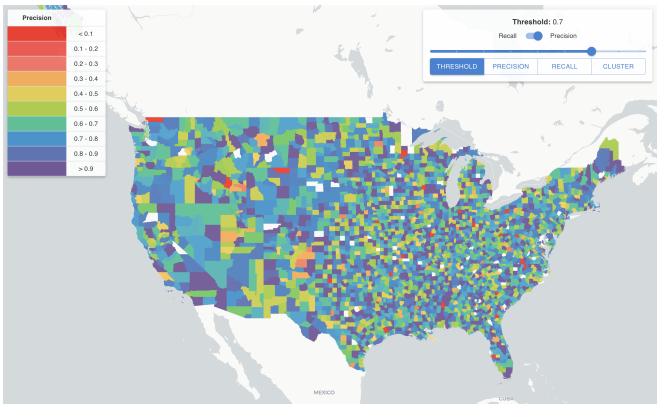

Figure 6: The figure above demonstrates the visualization of the validation response using a choropleth map. The values displayed are the precision values produced when a threshold value of 0.7 was used.

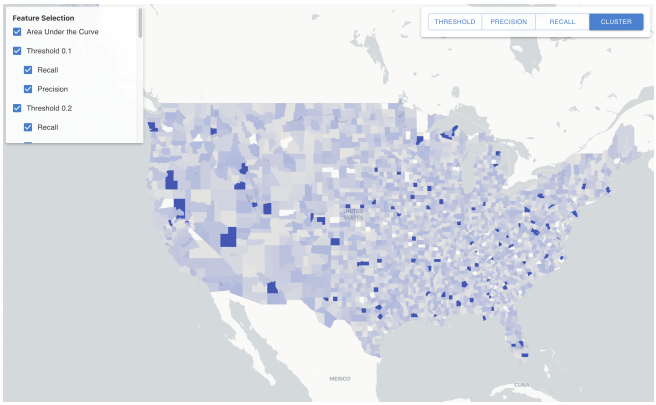

Figure 7: In the above figure, the different clusters as a result of the K-means algorithm are identified using color-coding. The highlighted dark-blue regions demonstrate the capability to highlight all regions associated to the same cluster as a user-selected region.

The grouping functionality results in the regions visualized across the map to be color-coded based on their assigned cluster resulting from the execution of the K-means clustering. The user is then able to further determine how regions are performing similarly by selecting a spatial extent displayed on the map and, as a result, all spatial extents that have been assigned to the same cluster will highlight across the map, nearly instantaneously (~40ms). This particular functionality can been seen demonstrated in Fig. 7. This feature is advantageous to the user in determining spatial extents that are performing the best or worst, given a predefined standard. For example, if a user has knowledge that a spatial extent has a precision or recall which is under-performing, they can then use this clustering feature to extend this knowledge to all other spatial extents that are similarly under-performing.

5 CONCLUSIONS AND FUTURE WORK

Here we described our methodology to identify spatial variation in the performance of classification models.

RQ-1: Partitioning of observations across spatial extents (defined by shape files) allows us to collate them and preserve data locality and preferentially sample observations during validations. Treating models as black boxes allows our framework to be independent of the structural characteristics of the model. Creating model instances per spatial extents and tracking performance metrics for multiple thresholds allows us to contrast and compare model performance across different thresholds.

RQ-2: Given the data volumes, working with a fraction of the dataset alleviates resource requirements. Further, the two-phase observation apportioning scheme allows preferential allocation of observations (and validations) to spatial extents that benefit from them. This allows us to build confidence in our results with a limited observational budget.

RQ-3: Leveraging interactive choropleth maps to render model performance allows researchers to quickly assess model performance. Leveraging streaming and GPU-acceleration allows the visualizations to be responsive and facilitate progress tracking. Our use of implicit declarative queries alongside our visualization scheme allows researchers to profile performance across different spatial extents and identify similarity using an unsupervised clustering-based scheme.

As part of future work, we will allow overlaying of related phenomena to allow researcher to identify potential reasons as to why the models are performing differently. For example, an urban researcher modeling air quality could benefit from an overlaying of proximate power plants (especially, gas or coal-based). Such overlays could be beneficial in allowing researchers to identify other variables that should be accounted for in their models.

6 ACKNOWLEDGMENTS

This research was supported by the National Science Foundation (OAC-1931363, CNS-2312319), and an NSF/NIFA Artificial Intelligence (AI) Institutes AI-CLIMATE Award [2023-03616].

REFERENCES

[1] Martín Abadi et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283.
[2] Paula Branco, Luís Torgo, and Rita P Ribeiro. 2016. A survey of predictive modeling on imbalanced domains. 49, 2 (2016), 1–50.
[3] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. 2017. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering* 29, 11 (2017), 2552–2566.
[4] Walid Budgaga, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. 2017. A framework for scalable real-time anomaly detection over voluminous, geospatial data streams. *Concurrency and Computation: Practice and Experience* 29, 12 (2017), e4106.
[5] Caleb Carlson, Menuka Warushavithana, Saptashwa Mitra, Kassidy Barram, Sudipto Ghosh, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. 2022. Resource Efficient Profiling of Spatial Variability in Performance of Regression Models. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 437–444.
[6] Changhao Chen, Bing Wang, Chris Xiaoxuan Lu, Niki Trigoni, and Andrew Markham. 2020. A survey on deep learning for localization and mapping: Towards the age of spatial machine intelligence. *arXiv preprint arXiv:2006.12567* (2020).
[7] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification

evaluation. *BMC genomics* 21 (2020), 1–13.
[8] Yeounoh Chung et al. 2019. Slice finder: Automated data slicing for model validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1550–1553.
[9] AL-Alimi Dalal, Yuxiang Shao, Ahamed Alalimi, and Ahmed Abdu. 2020. Mask R-CNN for geospatial object detection. *International Journal of Information Technology and Computer Science (IJITCS)* 12, 5 (2020), 63–72.
[10] Sophia Daskalaki, Ioannis Kopanas, and Nikolaos Avouris. 2006. Evaluation of classifiers for an uneven class distribution problem. 20, 5 (2006), 381–417.
[11] Caitlin Dempsey. 2012. Where is the Phrase "80% of Data is Geographic" From? - GIS Lounge. https://www.gislounge.com/80-percent-data-is-geographic/. (Accessed on 03/24/2023).
[12] Inc. Docker. 2023. Docker. https://www.docker.com/. Accessed on 03/23/2023.
[13] Cloud Native Computing Foundation. 2023. Kubernetes. https://kubernetes.io/. (Accessed on 03/23/2023).
[14] GC Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Shrideep Pallickara, Xiaohong Qiu, Ahmet Uyar, Minjun Wang, et al. 2003. Collaborative web services and peer-to-peer Grids. *SIMULATION SERIES* 35, 1 (2003), 3–12.
[15] Geoffrey Fox, Shrideep Pallickara, and Xi Rao. 2005. Towards enabling peer-to-peer Grids. *Concurrency and Computation: Practice and Experience* 17, 7-8 (2005), 1109–1131.
[16] Alex A Freitas. 2014. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter* 15, 1 (2014), 1–10.
[17] Google. 2023. Measure performance with the RAIL model. https://web.dev/rail/. (Accessed on 03/25/2023).
[18] Google LLC. 2023. Lighthouse: Automated tool for improving the quality of web pages. https://github.com/GoogleChrome/lighthouse. Version 10.0.2.
[19] Jiawei Han, Micheline Kamber, and Jian Pei. 2012. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers.
[20] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)* 28, 1 (1979), 100–108.
[21] Minsuk Kahng et al. 2016. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–6.
[22] Arvind W Kiwelekar, Geetanjali S Mahamunkar, Laxman D Netak, and Valmik B Nikam. 2020. Deep learning techniques for geospatial data analysis. *Machine Learning Paradigms: Advances in Deep Learning-based Technological Applications* (2020), 63–81.
[23] Jae Won Lee et al. 2005. An extensive comparison of recent classification tools applied to microarray data. *Computational Statistics & Data Analysis* 48, 4 (2005), 869–885.
[24] Viktor Leis, Alfons Kemper, and Thomas Neumann. 2013. The adaptive radix tree: ARTful indexing for main-memory databases. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 38–49.
[25] Wes Lloyd, Shrideep Pallickara, Olaf David, Jim Lyon, Mazdak Arabi, and Ken Rojas. 2011. Migration of multi-tier applications to infrastructure-as-a-service clouds: An investigation using kernel-based virtual machines. In *2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE, 137–144.
[26] Wes J Lloyd, Shrideep Pallickara, Olaf David, Mazdak Arabi, Tyler Wible, Jeffrey Ditty, and Ken Rojas. 2015. Demystifying the clouds: Harnessing resource utilization models for cost effective infrastructure alternatives. *IEEE Transactions on Cloud Computing* 5, 4 (2015), 667–680.
[27] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. 2014. Evaluating geospatial geometry and proximity queries using distributed hash tables. *Computing in Science & Engineering* 16, 4 (2014), 53–61.
[28] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. 2015. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing* 5, 1 (2015), 28–42.
[29] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. 2017. Hermes: Federating fog and cloud domains to support query evaluations in continuous sensing environments. *IEEE Cloud Computing* 4, 2 (2017), 54–62.
[30] Steven Manson, Jonathan Schroeder, David Van Riper, Tracy Kugler, and Steven Ruggles. 2022. National Historical Geographic Information System: Version 17.0. https://doi.org/10.18128/D050.V17.0
[31] Mark Masse. 2011. *REST API design rulebook: designing consistent RESTful web service interfaces.* " O'Reilly Media, Inc.".
[32] Adam Paszke et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019).
[33] Fabian Pedregosa et al. 2011. Scikit-Learn: Machine learning in Python. *the Journal of Machine Learning Research* 12 (2011), 2825–2830.
[34] Daniel Rammer, Walid Budgaga, Thilina Buddhika, Shrideep Pallickara, and Sangmi Lee Pallickara. 2018. Alleviating i/o inefficiencies to enable effective model training over voluminous, high-dimensional datasets. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 468–477.
[35] Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, and Nuno Carvalhais. 2019. Deep learning and process understanding for data-driven Earth system science. *Nature* 566, 7743 (2019), 195–204.
[36] Marco Tulio Ribeiro et al. 2016. Why should I trust you? Explaining the predictions of any classifier. In *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data mining*. 1135–1144.
[37] Ravi S Sandhu. 1998. Role-based access control. In *Advances in computers*. Vol. 46. Elsevier, 237–286.
[38] Uber Technologies Inc. 2023. Deck.gl: WebGL-powered framework for visual exploratory data analysis of large datasets. https://github.com/visgl/deck.gl. Version 8.8.4.
[39] Ivan Vasilev et al. 2019. *Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow.* Packt Publishing Ltd.
[40] Russell S. Vose et al. 2014. Improved Historical Temperature and Precipitation Time Series for U.S. Climate Divisions. *Journal of Applied Meteorology and Climatology* 53, 5 (May 2014), 1232–1251. https://doi.org/10.1175/jamc-d-13-0248.1
[41] Senzhang Wang, Jiannong Cao, and Philip Yu. 2020. Deep learning for spatiotemporal data mining: A survey. *IEEE transactions on knowledge and data engineering* (2020).
[42] Xingwei Wang et al. 1993. GRPC: a communication cooperation mechanism in distributed systems. *ACM SIGOPS Operating Systems Review* 27 (1993), 75–86. Issue 3. https://doi.org/10.1145/155870.155881
[43] Xiaoyu Wang et al. 2000. Spatiotemporal data modeling and management: a survey. In *36th International Conference on Technology of Object-Oriented Languages and Systems*. IEEE, 202–211.
[44] Graeme G Wilkinson. 2005. Results and implications of a study of fifteen years of satellite image classification experiments. *IEEE Transactions on Geoscience and remote sensing* 43, 3 (2005), 433–440.
[45] Xingzhou Zhang et al. 2018. pCAMP: Performance Comparison of Machine Learning Packages on the Edges. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*.