Resource Efficient Profiling of Spatial Variability in Performance of Regression Models

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

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

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

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

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

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

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

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

***Abstract*—Scientists design models to understand phenomena, make predictions, and/or inform decision-making. This study targets models that encapsulate spatially evolving phenomena. Given a model, our objective is to identify the accuracy of the model across all geospatial extents. A scientist may expect these validations to occur at varying spatial resolutions (e.g., states, counties, towns, and census tracts). Assessing a model with all available ground-truth data is infeasible due to the data volumes involved. We propose a framework to assess the performance of models at scale over diverse spatial data collections. Our methodology ensures orchestration of validation workloads while reducing memory strain, alleviating contention, enabling concurrency, and ensuring high throughput. We introduce the notion of a validation budget that represents an upper-bound on the total number of observations that are used to assess the performance of models across spatial extents. The validation budget attempts to capture the distribution characteristics of observations and is informed by multiple sampling strategies. Our design allows us to decouple the validation from the underlying model-fitting libraries to interoperate with models constructed using different libraries and analytical engines; our advanced research prototype currently supports Scikit-learn, PyTorch, and TensorFlow.**

***Index Terms*—model validations, spatial data, regression**

I. Introduction

Scientists construct models to understand phenomena and inform decision-making. These models may be analytical models where a model is fit to the data, or they may be domain theoretic models. We consider spatial models, i.e., models that attempt to capture spatiotemporally evolving phenomena. The class of models that we focus on are regression models that operate on spatiotemporal datasets. In spatiotemporal datasets, the data are geocoded with latitude/longitude information and observations have timestamps associated with them. Spatiotemporal datasets encapsulate observational data at diverse timescales and allow scientists to explore interrelated phenomena. A model designer may decide to leverage features (or variables) from diverse collections as the independent variables while choosing a dependent variable (also referred to as the response variable or target). Once a model is constructed, a question that arises is, "*How is the model performing?*" Model validation is the process of assessing the performance of a model by evaluating its accuracy with ground truth. Validation allows us to understand, compare, and interpret the performance of models. Model validations are a precursor to informing model refinements and targeting specific spatial extents to further scrutiny. For example, if a model predicting the direction of a forest fire works well in Oregon but underperforms in Colorado, the scientist may choose to consider topographical characteristics such as elevation or terrain.

The crux of this paper is to effectively evaluate the performance of spatial models at scale. The performance measures used to assess a model depend on the model type and evaluation metric deemed most suitable by the model designer. For example, for a regression model, a scientist modele may use RMSE, MAE, MSE, SSIM, or PSNR to assess the model's performance [1]. These performance measures are predicated on access to ground truth. Further, these assessments must utilize resources frugally, minimize network and disk I/O, and interoperate with diverse analytical engines. Ultimately, effective model validations inform targeted model redesign,

refinements, and calibration at scale by identifying spatial extents where a model performs well and where it does not.

A. Challenges

Performing model validations at scale introduces challenges stemming from the nature of datasets and model evaluations.

1) **Data volumes**: The datasets we consider are voluminous comprising a large number of observations that are high-dimensional encapsulating multiple features.

2) **Model inferences can be resource-intensive**: Model inferences trigger disk I/O, have computational overheads associated with them, and can have large memory footprints. In some cases, model inferences may trigger network I/O during data accesses. As such, the evaluations must balance validation coverage (spatial reach and resolution of the inference) with the resource-intensive nature of these evaluations.

3) **Interoperate with diverse analytical engines**: Researchers develop models using diverse analytical engines such as Scikit-Learn, TensorFlow, PyTorch, and Apache Spark. Different analytical engines have different model storage formats, encoding formats, invocation, mechanisms, and pipelining schemes that should be reconciled.

4) **Parametrization of models**: The parameters that serve as inputs and the output of these models may be drawn from different collections. Furthermore, the inputs may have different preprocessing operations such as normalization, encoding format reconciliations, etc. that need to be performed.

B. Research Questions

The overarching theme of this study is: *How can we perform model validations at scale over voluminous spatiotemporal datasets?* Within this broader theme we explore the following:

RQ-1: How can we strike a balance between validation coverage and the resource-intensive nature of validations?

RQ-2: How can we ensure that model validations scale?

RQ-3: How can we effectively interoperate with diverse analytical engines?

RQ-4: How can we effectively characterize model performance over large spatial extents?

C. Approach Summary

Our methodology encompasses staging of datasets, apportioning of observations for validation, creation and dispersion of model instances, parameterization of models alongside any expected wrangling of features, and visualizing model performance and uncertainty measures. These are accomplished while ensuring effective resource utilization, reconciling contention, alleviating disk and network I/O, and ensuring timeliness. We allow users to specify the granularities at which model validations should be performed.

Our methodology collates disparate observations into smaller spatial extents based on administrative boundaries, such as states, counties, or census tracts from the U.S. Census Survey Bureau based on *shapefiles* that encapsulate N-sided polygons similar shapefiles exist internationally for boundaries like provinces and cantons. Each observation is tested for whether it is encapsulated within a shapefile for the smallest, indivisible aggregation unit (which is a census tract in our study) and assigned a single-dimensional prefix. This prefix assignment is hierarchical allowing aggregation along administrative boundaries. This partitioning of model performance based on spatial extents allows us to have a finer-grained view of model performance. To profile model performance, we create (or reuse) a model instance for the spatial extent under consideration.

To reduce disk I/O and computational overheads involved in assessing model performance, we introduce the notion of a *validation budget*. This represents the upper bound for the total number of observations that are expended. Within this broader concept, we explore three different schemes to apportion the validation budget across model instances: equal, proportional, and uncertainty reduction. To reduce the number of repeated I/O operations that are triggered when observations are retrieved in a piecemeal fashion, we perform batched retrievals of observations per spatial extent.

To maximize interoperability with diverse analytical engines, we treat models as black boxes and consider only their parameterization, data preprocessing, and performance characteristics. Because we treat models as black boxes, we do not inspect the internal structural properties of the models. For instance, the models we consider could be based on partial differential equations, decision trees, matrix multiplications, and convolutions among others. Data preprocessing involves normalizing features based on the schemes specified by the modeler. Finally, we contrast model outputs with ground truth available from observational data and use that to compute a model performance metric based on the user-specified measure. We reconcile multiple model representation formats and marshalling schemes.

Model performance metrics are collated at a coordinator node, which may decide to allocate an additional budget to reduce uncertainty. Once the results satisfy a stopping criteria, they are streamed to a dashboard to be visualized as a choropleth map. Visualization results are streamed incrementally, and the graphs refined as the data become available. The choropleth map is shaded by model variability and loss, and can be used to inform targeted model refinements.

D. Paper Contributions

Our specific contributions include the following:

1) Our validation budgets can be apportioned using different schemes. In particular, our scheme allows preferentially targeting spatial extents where the model has high variability in performance. Further, these validations are performed by preserving data locality. **[RQ-1, RQ-2]**

2) Our methodology supports interoperation with diverse analytical engines. Our advanced research prototype supports PyTorch, TensorFlow, and Scikit-Learn. **[RQ-3]**

3) Our methodology supports effective surveillance of regression model performance (though we assert that our methodology should be just as applicable to classification models as well). Our methodology allows continual validation of model

performance, by periodically assessing model performance at different spatial extents. Furthermore, validation budgets for surveillance can be smaller and expended on spatial extents where there is greater uncertainty. This allows effective identification of spatiotemporal extents where a model performs well and where it does not. **[RQ-4]**

II. Methodology

Our methodology encompasses a set of phases that includes: (1) dataset staging and sharding, (2) validation budgets encompassing observations, (3) performing validations while treating the models as block boxes, (4) ensuring scalability and avoiding performance bottlenecks, (5) ensuring extensibility of the framework, (6) supporting incremental validations in support of continuous model performance surveillance, and (7) ensuring responsiveness at the client side with support for streaming and incremental rendering operations.

A. Dataset Staging and Sharding

In order to assess the performance of a model efficiently, it is critical to achieve data locality to minimize or eliminate network I/O. Voluminous geospatial datasets must be partitioned and distributed across multiple machines and their disks. We leverage data locality by pushing the computation (model) to the data, where it is evaluated directly either in memory or on disk. Our approach is to group records within (hierarchical) spatial boundaries together, then distribute these groups evenly across a cluster. For administrative boundaries such as counties, this would mean grouping records together by county and balancing the data across the storage machines.

Records that come in a gridded format with just a latitude/longitude must be associated or tagged with the administrative boundary in which it lies. We accomplish this by using the shapes of the administrative boundaries to determine the record's coordinates. Once the records have been tagged with their associated boundary, they are grouped together as a single data shard and dispersed within the cluster. Additionally, we maintain metadata about where that data were placed in order to assist with subsequent spatial computations.

B. Validation Budgets *[RQ-1, RQ-2]*

A key goal of the model validation service is to understand the *true* error structure of a model, as given. A choropleth map displays spatial variation in colored gradients for a numeric variable (see Fig. 1). Visualizing the error structure via choropleth maps gives a user the ability to quickly spot where their model performs poorly, as seen by both the variance and loss maps in Fig. 1. Unfortunately, with voluminous data, this can be an infeasible task, especially when under time or cost constraints. We attempt instead to *estimate* the error structure of a model by using only a sample, or subset, of the entire ground truth dataset. The sample size, called the *validation budget* (denoted *n*), is the total number of records that a model is allowed to be evaluated on. This validation budget is allocated in such a way that maximizes our understanding of the model's true performance across all geospatial extents, while minimizing the cost of achieving these estimates.

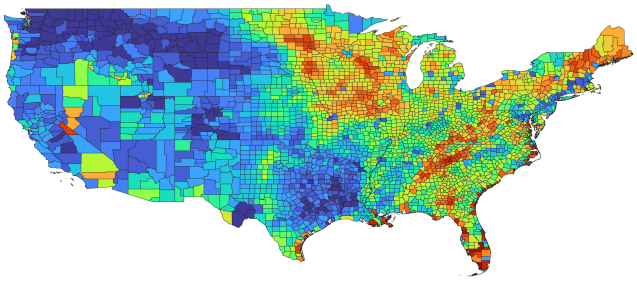

Fig. 1. County choropleth maps for true model loss. Cooler colors are lower loss values, and hotter colors are higher values.

We explain this in terms of stratified sampling. Let $D = \{1, 2, ..., N\}$ denote a database of N units, g_k denote ground truth for unit k, and m_k denote a model prediction that could be computed for unit k. Suppose that the database is divided into H mutually exclusive and exhaustive strata, $D = \cup_{h=1}^{H} D_h$ where the size of D_h is N_h and $N = \sum_{h=1}^{H} N_h$. In our example above, counties would be the strata. We wish to compare model output to ground truth using a total validation budget of $n = \sum_{h=1}^{H} n_h$ units, with a sample $d_h \subset D_h$ of n units randomly selected from the N_h units in stratum h. If the goal is to understand the mean squared error (MSE) over the entire database, then we use $MSE = N^{-1} \sum_{h=1}^{H} \sum_{k \in D_h} (g_k - m_k)^2$. An unbiased estimator of MSE based on the stratified random sample is $mse = N^{-1} \sum_{h=1}^{H} \sum_{k \in D_h} (g_k - m_k)^2 (n n_h^{-1})$. Furthermore, an optimal allocation would make the variance of mse as small as possible subject to the validation budget, n. In some cases, it might make sense to consider unequal costs c_k of model evaluation in different strata, so that the validation budget would be replaced by the total cost $\sum_{h=1}^{H} c_h n_h$, which reduces to the validation budget n if all costs equal one.

A naïve approach of equal allocation would assign $n_h = nH^{-1}$, which effectively breaks the total budget into equal size chunks for every stratum. We call this the **equal allocation budget**. A slightly better approach would be proportional allocation, which would assign $n_h = nN_h N^{-1}$, rounding to integers if necessary, which we call the **proportional allocation budget**. This is implemented as a sampling rate of the underlying strata. Various compromises between equal and proportional allocations begin with a minimum allocation in every stratum, say n_0, then allocates the remaining sample $n - Hn_0$ in proportion to a power of the stratum size: $n_h = n_0 + cN_h^a$ where $a = 1/2$ or $1/3$ are typical choices. These allocations have the advantage of "protecting" the smaller strata by not making large strata overly precise. While both of these allocations are simple, they do not yield optimal information about overall MSE if either the costs are unequal across strata or the prediction errors have different behavior in different strata. In the case of our county example, many may have wildly different sizes, and the model will likely perform differently based on the data within.

An improvement over the simple allocation schemes would be to proportionally allocate the budget based on the model's

empirical variance. This tells us where the model is predictable, and where it varies based on the input data. We write the empirical variance of the squared prediction errors in stratum h as $S_h^2 = \{\sum_{k \in U_h} e_k^2 - \frac{(\sum_{k \in U_h} e_k)^2}{N_h}\}/(N_h - 1)$, where $e_k = (g_k - m_k)^2$. Obviously, e_k must reflect the loss function used, so if we were using the MAE to validate the model, then $e_k = |g_k - m_k|$. The optimal allocation subject to the cost constraint is then shown to be $n_h \propto \frac{N_h S_h}{\sqrt{c_h}}$. When costs are constant across strata, this is known as the Neyman allocation. The optimal allocation reduces to proportional allocation if costs and variances are constant across strata, and reduces to equal allocation if costs, variances, and stratum sizes are all identical. Overall, the optimal allocation assigns more samples to larger, more variable, and cheaper strata.

We next introduce a simple model to illustrate the behavior of the optimal allocation. Suppose that the model prediction errors satisfy $g_k - m_k = \mu_h + \tau_h \varepsilon_k$ for $k \in D_h$ all , where μ_h is the model bias in stratum h, τ_h^2 is the prediction error variance in stratum h, and the ε_k are independent and identically distributed normal random variables with mean zero and variance one, across all strata. Then assuming N_h values are large in every stratum, $S_h^2 \simeq Var((g_k - m_k)^2) = Var((\mu_h + \tau_h \varepsilon_k)^2) = Var(\mu_h^2 + 2\mu_h \tau_h \varepsilon_k + \tau_h^2 \varepsilon_k^2) = 4\mu_h^2 \tau_h^2 + 2\tau_h^4$, using properties of the normal distribution. The optimal allocation would then assign more samples to strata with high model bias, high prediction error variance, or both.

In practice, S_h^2 is unknown, since that would require evaluating the model on the full dataset. In this case, we first assign some of the budget to obtain initial estimates s_h^2 of S_h^2 in every stratum, using a simple allocation such as equal allocation of n_0 units in every stratum h, then use the approximately optimal allocation of the remaining sample $n - Hn_0$, plugging in the estimated s_h in place of the unknown S_h. This gives us some idea of how close we are to understanding its true error structure had it been evaluated on the full dataset (i.e., population, instead of the sample). The proportional allocation equation used for a given stratum h is $n_h = (n - Hn_0) s_h (\sum_{k \in s} s_k)^{-1}$. Using this strategy, we are able to proportionally hand out the remaining samples of the budget, based on the estimated variances, for a final evaluation round. We call this allocation strategy the **incremental variance budget**. When using this budget, it's important to capture both the variance for the initial allocation *and* the variance for the new allocation together.

For example, let's say a stratum h is evaluated with $n_0 = 200$, resulting in some variance s_h then is allocated an additional $n_h = 500$ records for re-evaluation. After the new evaluation, it wouldn't be sensible to throw away the variance resulting from the initial run in favor of the final evaluation variance; the first probably had a better estimate of the population variance for h. This leaves us with two options. The first option is to re-evaluate the original 200 records plus the additional 500, giving us a sample variance for the fully allocated 700. However, stratum h was not allocated 700 records for the second go-around; it was only allocated 500. This would be exceeding the total budget, and incurring more resource usage than originally assigned. A second and better approach that we have implemented is to use Welford's method, derived as $(N-1)s_N^2 - (N-2)s_{N-1}^2$ for calculating the online variance across allocation iterations. Not only does it eliminate the need for re-evaluating the initial variances (or storing them), it's also more numerically stable and only requires each stratum to save three variables throughout all iterations: $\bar{x}$, s_x^2, and N throughout all iterations [2].

After evaluating global performance with this approach, we noticed that this helps us better understand the model *as a whole* given the total validation budget, but a drawback is that strata with high variability do not get a large enough share of the remaining budget in order to effectively improve their local estimate of the model's variance (S_h) and error (MSE_h). Intuitively, when s_h is low after the initial allocation round, that means we already have a good idea of how that model behaves for that stratum, and do not need to further allocate any more observations for a final round from the remaining budget. Sorting the variance estimates s returned from the initial round, we see they follow a normal distribution. Thus, we place a threshold such that only variance estimates greater-than-or-equal-to two standard deviations above the mean of the variance estimates are considered for the Neyman proportional allocation, otherwise that stratum's evaluation is considered complete. This threshold parameter can be tuned or removed altogether. In the example shown by Fig. 2, only the counties with variance estimates above the red threshold would be proportionally allocated the rest of the budget. This has the benefit of reserving the entire remaining validation budget for the strata with high variance estimates, greatly improving their estimate of the true model variance and loss after the final evaluation round. This also reduces the computational load, since only a fraction of the strata is re-evaluated.

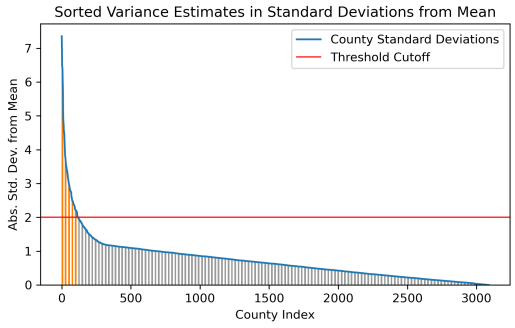

Fig. 2. Variance estimate allocation threshold. Only counties above 2 std. dev. from mean, shown in orange, are chosen for further examination. The other counties, depicted in gray, are considered complete.

C. Scalability and Avoiding Bottlenecks *[RQ-2]*

We include several mechanisms in place to ensure that the validations scale. This includes (1) the use of validation budget alongside our schemes to estimate the error structure of the model, (2) alleviating orchestration costs at the coordinator, and (3) ensuring effective utilization of resources at the worker nodes. These strategies are complemented with schemes such as data replication, asynchronous I/O, thread-pooling,

multiprocessing, efficient (un)marshalling, space-efficient data structures for storing both metadata and computational values.

Coordinator: The coordinator node preserves data locality during validations and apportions the validation budgets based on different schemes. It also encapsulates state relating to the model evaluations across spatial extents and uses this to inform apportioning schemes. We also separate the control plane traffic from the data plane to ensure that the coordinator is not swamped with traffic that should not be traversing through it. Control traffic initiated by the coordinator includes launching validation tasks while preserving data locality and apportioning additional observations based on the variance in the model errors.

The coordinator is also lightweight. The entirety of a coordinator's state resides in-memory. Additionally, Workers are responsible for registering themselves with the coordinator, and providing all metadata through self-discovery such as the data/spatial extents they have stored on disks, and the size/counts of them. This relieves the coordinator from having to initiate any data or Worker discovery procedures. To improve scalability, more workers can be dynamically added or removed. The coordinators are replicated using Zookeeper and include replicas in active standby node that can take over as the primary during failures.

For load-balancing requests, the coordinator organizes information about spatial extents in a prefix tree. Each spatial extent is represented using a string; hierarchical spatial extents (e.g., census tracks, cities, counties, states) share a prefix. The greater the string length, the more finer-grained the spatial extent. We leverage the radix tree (a space optimized representation of the prefix tree) to manage and organize data relating the spatial extents. The node values in the prefix tree maintain information about where the data for that spatial extent is located. For spatial identifiers of length k, the choice of the radix tree data structure allows all search operations to be performed in $O(k)$. Metadata about the workers are stored in an inverted fashion using a hashmap from the registered worker to an array of spatial extent IDs stored at the worker.

When a request comes in to validate a model for all spatial extents at a given resolution, the coordinator creates a set of worker jobs, each containing the subset of spatial IDs that a worker w has data for locally. In the case where each spatial extent is replicated over multiple locations, in hash-based replication schemes similar to MongoDB and Cassandra, the coordinator uses a round-robin policy to assign each ID to the replica set member with the lowest number of IDs.

Finally, the constructed jobs are concurrently submitted to their respective workers using asynchronous I/O, so no blocking occurs waiting for a submitted job to return. We also note that only one copy of the submitted model is held in memory for the coordinator while transmitting; since no writes are happening here, it can be simultaneously read by multiple *send()* threads which include it as a byte stream in the job requests to the workers.

Workers: On startup, workers discover the spatial extent IDs of the data available to them locally, storing this metadata in a radix tree similar to the coordinator, and report this to the coordinator in a registration request. 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.

*D. Models as Black Boxes **[RQ-3]***

Our goal is for the validation service to be agnostic of the internal structure of the model. The model could be based on decision trees, ensembles, partial differential equations, and matrices. By treating them as black-boxes, we focus on how inputs are fed into the model, how outputs are retrieved, and how to evaluate the results against the spatial extent's ground truth. The format or type of the model *does* matter, as it determines how the model is loaded and executed, but beyond this, a user may provide an arbitrarily complex model so long as they specify how to use it (for example, a model saved using TensorFlow is loaded/executed in a completely different way than a model saved from PyTorch).

A recent survey which scraped all the papers (and their respective GitHub repositories) published at the top computer vision, NLP, and general ML conferences from 2017 to the end of 2019 shows that TensorFlow/Keras and PyTorch make up for the vast majority of ML projects [3]–[5]. Another survey showed Scikit-learn/SciPy being used in over half of the scraped ML projects in 2019, totaling 110,000 open-source projects [6], [7]. These three model frameworks, PyTorch [8], TensorFlow [9], and Scikit-learn [10] are the first we have implemented support for black-box model inference on.

Submitting a model for validation is simplified through model request parameterization. Everything from the input that is retrieved and fed into the model, to how to evaluate the model's performance can be specified through the request's JSON parameters. In a request, a user must minimally specify the data source, spatial resolution, input features, output labels, and type of model. The back-end framework infers any other needed information, and performs end-to-end the model evaluation job.

*E. Software Extensibility **[RQ-3]***

Supported model inference frameworks (i.e., PyTorch, TensorFlow, Scikit-learn) and data retrieval frameworks (i.e., MongoDB) were implemented as *validator* and *querier* objects in their own respective Python modules. These objects follow the "Strategy" design pattern [11] and inherit common functions, helper methods, and member fields from an abstract superclass. Implementing common construction and execution function signatures maintains a consistent use pattern from the worker's perspective. A concrete example of this with the querier object is the `spatialQuery` function, which takes as arguments the spatial ID that we query for, the feature fields and label field we wish to project out of the results, and an optional limit and sample rate. With different querier implementations, the way data are retrieved can be

modified to use different connectors or sampling strategies all while staying transparent to the user of the interface. With this approach, a new framework can easily be added and supported via a new Python module with minimal changes to the worker request routing. For example, one could easily plug in a supporting module for Apache HDFS as an underlying data store, or MATLAB as a new model inference framework.

F. Incremental Evaluation ***[RQ-4]***

In most cases, datasets are dynamic as observations are added over time. When this is the case, models that have already been evaluated on previous data may become "stale", where their error structure as understood before is not representative of how it performs over newer data. We address this by incrementally allocating some additional budget for each of the spatial extents receiving new data to re-evaluate the model on. By persisting the variance, count, and mean of the values the model has been evaluated on so far, we capture a new, up-to-date variance that accurately estimates performance for both the old and new records together. This is a classic online strategy, and we leveraged Welford's method for calculating a running variance [2]. Since only a couple of parameters have to be stored for each spatial extent, instead of the entire set of residuals calculated thus far, it is a computationally inexpensive operation to keep model validation estimates current.

III. Performance Benchmarks and Discussion

A. Experimental Setup

Our experiments were performed over a cluster of 25 machines, each with an 8-core CPU running at 2.10GHz, 64GiB of DDR3 RAM, and 5400RPM hard disks. Three of the 25 machines were set aside as a replica set to manage the sharded/replicated database configuration, and one of the machines was dedicated as the coordinator node. The remaining 21 machines all housed the data shards locally and ran worker processes, responsible for model validation workloads. A sharded, replicated MongoDB cluster was set up across these machines with WiredTiger as the storage engine; we note that any other distributed storage frameworks could have easily been used in its place like Apache Cassandra or HBase. Local *mongod* processes which had direct access to the shard data on disk were connected to by the worker processes.

B. Models

We trained both deep neural network (DNN) regression models for each of the supported analytical engines (TensorFlow, PyTorch, Scikit-learn), using *soil temperature 0.1 meters below surface* as the label, and 10 features related to *wind*, *pressure*, *dewpoint*, and *temperature above the surface*, and a loose hyperparameter grid-search to achieve optimal model performance for the entire experiment dataset.

C. Datasets

We use a subset (the year 2021) of NOAA's [12] North American Mesoscale Forecast System (NAM) dataset, downloaded as gridded-binary (.grb2) files, containing a latitude/longitude record for 12km grids in the observed North America range, and snapshotted at a 6-hour interval. Using only 36 of the 400 variables per 12km grid, we gathered just over 56,000 records for each of the 1,460 files. We tag each record with the shapefile's county ID, and ingest it into our sharded, replicated MongoDB cluster in BSON format. This resulted in just over 122GB of records in MongoDB before replication. Finally, we indexed and evenly distributed the county shards across our cluster's 21 worker machines. With 3,088 counties, this results in around 150 counties' data being located on each machine.

D. Metrics

We first began by running validation jobs with an *unlimited budget*, which defaulted to using all available observations for every county. Three separate modes were used at the worker for processing these jobs: (1) synchronous, (2) multithreaded, and (3) multiprocessing. Both the total job duration and individual worker duration results can be seen in the first three groups of Fig. 3. As our baseline, some of the serially-executed jobs on the workers took almost 4 minutes. Furthermore, we can see that the total job durations are dictated by the longest running workers, seen as outliers for the boxplots. This suggests that even though the *number* of counties were evenly distributed across the workers, a better approach would be to distribute the data based on the counties' data sizes, since larger counties consume more resources.

In addition to the different job processing modes, we executed three validation jobs using multiprocessing as the job processing mode: (1) A *proportional sampling* budget job that used a sample rate of 1 percent; (2) An *equal allocation* job that had 1M total records equally allocated to 3,088 counties; (3) An *incremental variance* job that had an initial allocation of 200 records to all counties, a total budget of 1M, using the allocation threshold of 2 standard deviations above the mean for initial variance estimates.

We can also see in Fig. 3 that the validation budgets further reduce the total amount of time it takes to complete a job. Variance budgeting takes slightly longer than equal allocation or proportional sampling due to its second pass of the model on counties with high variability, but the second variance pass provides much better estimates of both the population loss and variances in the cases where the initial estimates have a high variance. This can be seen in Fig. 6 where we compare population loss and variance to estimates provided by all three of the budgeting schemes. The final estimates provided by the incremental variance budget are much more aligned with the population values than the estimates generated by the equal allocation strategy or the proportional sample rate strategy. With this said, however, we did note that over all 3,088 counties, some of the initial estimates that did not receive a second allocation from the remaining budget did not do as well as the equal allocation sampling, which was effective at gaining a decent overall picture of the model's error structure.

Fig. 4 shows average worker CPU usage across these four jobs. We can see around 50 percent of the CPU being used on average, due to some of the parallel child processes being in

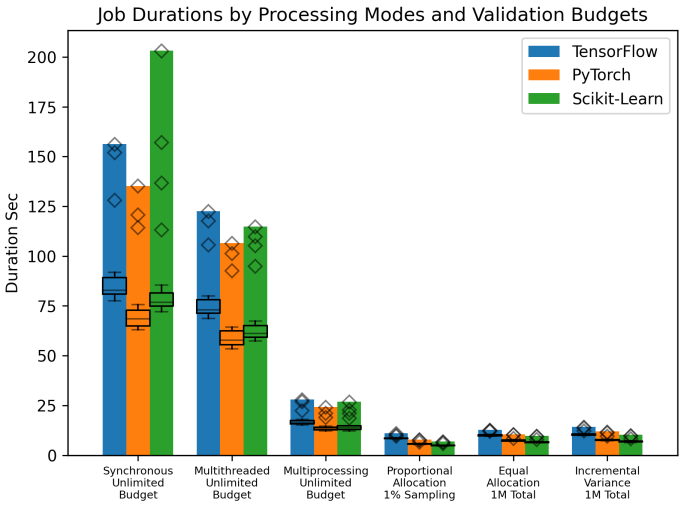

Fig. 3. Total and worker job durations by processing modes and budget types.

their data retrieval stage (I/O intensive), and others being in their model validation stage (CPU intensive). The incremental variance budget job finishes its first validation round quicker than the other schemes, but has to complete a second pass using the remaining budget, hence the spike we see at around 20 seconds into the job. The memory usage across experiments were roughly equivalent due to MongoDB's in-memory caching mechanism, but when using a cold-start scenario with cleared caches, disk I/O becomes a bottleneck for unlimited budgets (see Fig. 5).

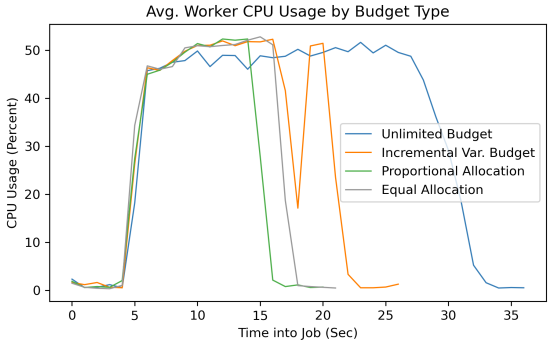

Fig. 4. Cluster average worker CPU usage by budget type. Incremental variance budget incurs a second spike in CPU usage due to its second phase.

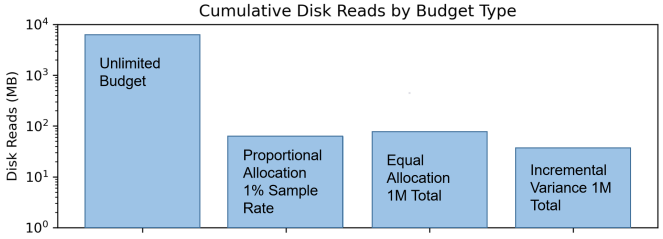

Fig. 5. Average worker disk reads by budget type, cold-start with no caching.

As shown in Fig. 7, the equal allocation (bottom) and proportional sample rate (second from bottom) budgets estimate the error structure of the model generally well, but have many

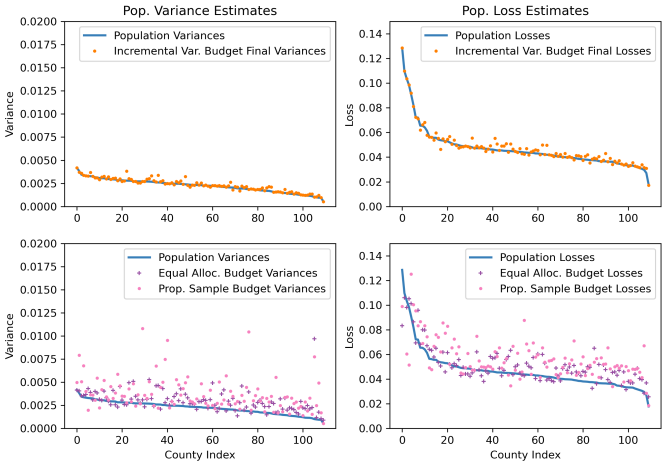

Fig. 6. Loss and variance estimate accuracy against true population loss and accuracy. Only counties with high variances considered. Our incremental variance budget clearly outperforms other allocation schemes.

outliers counties which do not capture the population variance at all. The incremental variance budget updates counties with abnormally high variances to be closer to the population variance, removing outliers, and maintaining a cleaner overall view of the true model variances. A choropleth map of the estimated loss using the middle strategy is provided in Fig. 8, which can be compared against the true population loss values for the experiment model in Fig. 1.

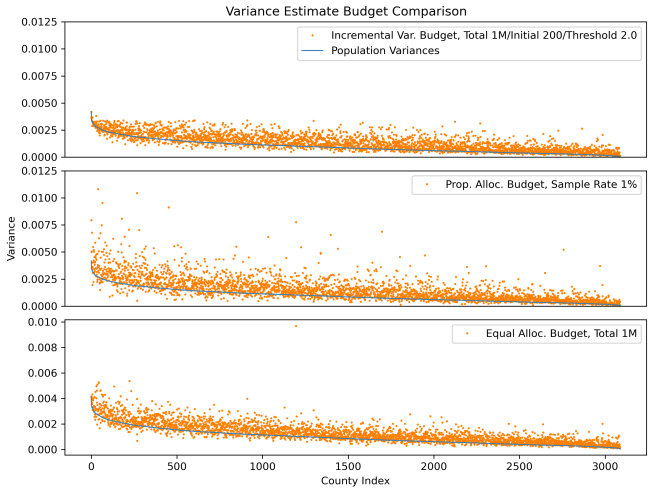

Fig. 7. Variance estimates by budget type. For *Variance Budgets*: 3 numbers A/B/C denote total allocation/initial allocation/threshold in standard deviations from mean. Incremental variance budget gives the best estimate of true model variance structure without outliers.

IV. Related Work

There is a growing demand to build and deploy predictive models over spatiotemporal data to discover interesting, and previously unknown, patterns to extract actionable insights [13]. Doing so requires efficient storing, querying, and object modeling of geospatial data using system architectures that are capable of supporting specific user requirements [14].

Model performance, in terms of its accuracy, is governed by a range of factors and models built over a set of labels with

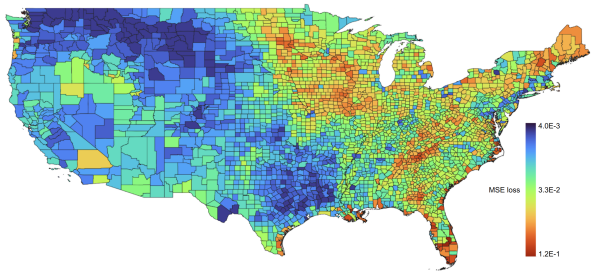

Fig. 8. County choropleth maps for estimated model loss. Cooler colors are lower loss values, and hotter colors are higher values. MSE loss values appear similar to what we see from using all available ground truth.

complex relationships often have significantly differing performance over different portions of the overall multidimensional data domain [15]. Overall model performance computed using typical validation techniques in such cases will not reflect the true performance over smaller sub-regions [16].

Identification of portions of the data space showing low prediction accuracy, i.e., sub-domains where the average loss is higher than a defined threshold, has been explored in various works through the application of statistical techniques [17], [18]. They facilitate analysis of the model performance at a more granular level. Additionally, they help provide a measure of confidence in a model based on the region over which it is applied so that they can be applied with a more trust [19] and help interpret and explain the model decision. These techniques, however, either require domain expertise and/or are not designed to handle voluminous datasets.

As stated by Matthias Schnaubelt's survey on using machine learning model validation [20], special methodologies such as forward validation schemes are needed for time-series models.

High dimensional datasets with a small number of samples are used in neuroimaging, genomic, eye-tracking and other biomedical studies. Andrius Vabalas et al. have demonstrated that when training classification models using these datasets, the type of validation scheme used could introduce a bias which would lead to inaccurate assessment of the model [21].

V. Conclusions and Future Work

Our benchmarks demonstrate the suitability of our methodology to facilitate resource efficient and timely validations.

RQ-1: Using a validation budget allows us to ensure spatial coverage while conserving resources (both computing and I/O) that are expended during model validations. We estimate the error structure of the model M without having to rely on validations involving a large number of observations.

RQ-2: To ensure scaling we complement our validation budget with several mechanisms such as data locality, data dispersions based sharding schemes, and conserving memory by reducing the number of model instances that are memory resident.

RQ-3: To ensure broader applicability, we treat models as black boxes without inspecting the internal structural properties of the models. For instance, the models we consider could be based on partial differential equations, decision trees, matrix multiplications and convolutions.

RQ-4: To characterize model performance we render visualizations of results as choropleth maps allowing users to explore spatial variations in model performance.

As part of future work, we propose to incorporate support for models based on classification and clustering.

Acknowledgment

This research was supported by the National Science Foundation [OAC-1931363, ACI-1553685] and the National Institute of Food and Agriculture [COL0-FACT-2019].

References

[1] R. D. Snee, "Validation of regression models: Methods and examples," *Technometrics*, vol. 4, pp. 415–428, 1977.

[2] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[3] "The state of machine learning frameworks in 2019," https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/.

[4] G. Nguyen *et al.*, "Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey," *Artificial Intelligence Review*, vol. 52, no. 1, pp. 77–124, 2019.

[5] J. Hale, "Deep learning framework power scores 2018 — towards data science," https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a, (Accessed on 04/11/2022).

[6] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, 2020.

[7] S. Raschka *et al.*, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, 2020.

[8] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[9] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[10] F. Pedregosa *et al.*, "Scikit-Learn: Machine learning in python," *the Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] E. Gamma *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994, p. 416.

[12] R. S. Vose *et al.*, "Improved historical temperature and precipitation time series for U.S. climate divisions," *Journal of Applied Meteorology and Climatology*, vol. 53, no. 5, pp. 1232–1251, May 2014.

[13] S. Shekhar *et al.*, "Spatiotemporal data mining: A computational perspective," *ISPRS International Journal of Geo-Information*, vol. 4, no. 4, pp. 2306–2338, 2015.

[14] X. Wang *et al.*, "Spatiotemporal data modeling and management: a survey," in *36th International Conference on Technology of Object-Oriented Languages and Systems.* IEEE, 2000, pp. 202–211.

[15] Y. Chung *et al.*, "Slice finder: Automated data slicing for model validation," in *2019 IEEE 35th International Conference on Data Engineering (ICDE).* IEEE, 2019, pp. 1550–1553.

[16] J. Krause *et al.*, "Interacting with predictions: Visual inspection of black-box machine learning models," in *The CHI Conference on Human Factors in Computing Systems*, 2016, pp. 5686–5697.

[17] M. Kahng *et al.*, "Visual exploration of machine learning results using data cube analysis," in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 2016, pp. 1–6.

[18] J. W. Lee *et al.*, "An extensive comparison of recent classification tools applied to microarray data," *Computational Statistics & Data Analysis*, vol. 48, no. 4, pp. 869–885, 2005.

[19] M. T. Ribeiro *et al.*, "Why should I trust you? explaining the predictions of any classifier," in *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, 2016, pp. 1135–1144.

[20] M. Schnaubelt, "A comparison of machine learning model validation schemes for non-stationary time series data," FAU Discussion Papers in Economics, Tech. Rep., 2019.

[21] A. Vabalas *et al.*, "Machine learning algorithm validation with a limited sample size," *PloS One*, vol. 14, no. 11, p. e0224365, 2019.