Distributed Orchestration of Regression Models Over Administrative Boundaries

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

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

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

Daniel Rammer
Department of Computer Science,
Colorado State University
Colorado, USA
rammerd@colostate.edu

Mazdak Arabi
Department of Civil & Environmental Engineering, Colorado State University
Colorado, USA
mazdak.arabi@colostate.edu

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

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

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

ABSTRACT

Geospatial data collections are now available in a multiplicity of domains. The accompanying data volumes, variety, and diversity of encoding formats within these collections have all continued to grow. These data offer opportunities to extract patterns, understand phenomena, and inform decision making by fitting models to the data. To ensure accuracy and effectiveness, these models need to be constructed at geospatial extents/scopes that are aligned with the nature of decision-making — administrative boundaries such as census tracts, towns, counties, states etc. This entails construction of a large number of models and orchestrating their accompanying resource requirements (CPU, RAM and I/O) within shared computing clusters. In this study, we describe our methodology to facilitate model construction at scale by substantively alleviating resource requirements while preserving accuracy. Our benchmarks demonstrate the suitability of our methodology.

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**.

KEYWORDS

spatiotemporal data, modeling, workload orchestration, transfer learning

ACM Reference Format:
Menuka Warushavithana, Caleb Carlson, Saptashwa Mitra, Daniel Rammer, Mazdak Arabi, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. 2021. Distributed Orchestration of Regression Models Over Administrative Boundaries. In *2021 IEEE/ACM 7th International Conference on Big Data Computing, Applications and Technologies (BDCAT '21) (BDCAT '21), December 6–9, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3492324.3494164

BDCAT '21, December 6–9, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-9164-1/21/12…$15.00
https://doi.org/10.1145/3492324.3494164

1 INTRODUCTION

There has been an exponential growth in data volumes over the past decade. A majority of these data are geospatial i.e., the data are geotagged with spatial coordinates. A large number of geospatial data collections have been made available by state/federal agencies, NGOs, and research labs. Domains these collections are available for include social, ecological, environmental, atmospheric, and geosciences.

These collections encapsulate observational data, model outputs, and surveys among others. Observational data can either be collected using *in situ* devices or are remotely sensed. Remotely sensed observational data are available at different spatial granularities and resolutions. Model outputs are often generated as gridded outputs that capture variations over the terrain and at different elevation levels. Surveys such as those conducted by the Census Bureau include a wealth of demographic information that are amenable to aggregation at different administrative levels.

Combining economic, social, demographic, and infrastructure data affords opportunities to identify issues from a holistic perspective. Models help us understand phenomena, predict risk, and

inform decision-making. However, since each spatial extent is different based on its topological characteristics, atmospheric and climatic variability, infrastructure profiles, population distributions, and income/access disparities construction of models should account for this variability. Models should capture regional variations at the scopes of interest. Often from the perspective of planning, this is done based on administrative boundaries that tend to be hierarchical. Consider for example, an analysis that attempts to identify vulnerability to poor air quality. Such an analysis would first identify areas where the air quality index (available from the U.S. Environmental Protection Agency) is above a certain threshold. Next, the U.S. Census Survey datasets would be used to identify census tracts where demographic triggers relating to the analysis exist. Examples of such demographic triggers include age and percentage of populations without health insurance among others. The analysis could be further supplemented by proximity to power plants and freeways (that worsen air quality) and hospitals (where care can be provided).

The crux of this study is to facilitate model construction at diverse spatial scopes at scale. In particular, the model construction should: (1) facilitate understanding of regional variations and patterns, (2) enable understanding of the spatial dynamics of phenomena across disparate data collections, (3) and allow different data collections to be used to inform understanding of phenomena and planning.

1.1 Research Challenges

Extracting insights at scale across different spatial (and temporal) scopes introduces several challenges. These include:
1) Data volumes and diversity. The data may be available as shape files, point observations, gridded data, or sketches while being encoded in diverse formats. Shape files encapsulate arbitrary N-sided polygons representing the spatial area of interest; vertices within these polygons are represented using <latitude, longitude> tuples.
2) Modeling workloads are both I/O and CPU intensive: Model fitting tasks are iterative and induce strain across the entire resource hierarchy (CPU, memory, disk, and network). Furthermore, because these workloads execute in shared clusters inefficiencies impact other collocated analytic tasks.
3) Diversity of model fittings: There are combinatorially explosive ways in which models can be realized. Models can be constructed for different combinations of features, spatiotemporal scopes, and using a diversity of model fitting algorithms.

1.2 Research Questions

The following research questions inform this study.
[RQ-1] *How can we support spatial data wrangling operations that are aligned with the nature of the workloads?* Broadly, data wrangling operations are key to identifying the spatiotemporal scopes of interest using a rich set of queries that are then subject to operations such as preprocessing, normalization, feature selection, etc. prior to launching modeling tasks.
[RQ-2] *How can we effectively orchestrate model training workloads?*
[RQ-3] *How can we balance and manage the I/O (disk and network) and memory pressure?*

1.3 Approach Summary

Our methodology encompasses a broad sequence of phases to facilitate model construction at scale over myriad spatiotemporal data collections. We place no restrictions on the model fitting algorithms that underpin model construction. Our methodology alleviates resource requirements for model training while preserving accuracy. In particular, our methodology includes support for (1) data wrangling, (2) identification of spatial similarity, (3) selectively promoting spatial extents for exhaustive model training and hyperparameter tuning, and (4) a novel transfer learning scheme that leverages spatial similarity by using these exhaustively trained models (their weight vectors, coefficients, and hyperparameters) as starting points to facilitate model training for other spatial extents.

Our data wrangling schemes support selection, preprocessing, and partitioning data that are aligned with spatial characteristics of the data. Our selection operators incorporate spatial geometry-based constraints alongside traditional predicate logic. The spatial geometry-based constraints include proximity, intersections of shape files, and checks for inclusion within spatial bounding boxes; all distance calculations are based on double-precision spherical coordinates. These spatial geometry constraints are supplemented with traditional predicate logic that places constraints on the range of individual feature values, equality operators, and relationships that features must have with respect to each other both within and across spatial data collections.

Models should be built for the right spatial extent to account for cross feature interactions. For a given set of features, the distribution of individual feature values and the nature of their cross-feature interactions may vary across spatial extents. For example, tracts within the same city often have significant income, access, and safety disparities. Rather than build an all-encompassing model global model, we build an ensemble of smaller ones that allow us to better capture regional variations better.

Model building is iterative and computationally expensive. While training models for smaller spatial extents/administrative boundaries allows us to capture regional variations and subtleties better, a consequence is that the number of model fitting operations that need to be performed increases substantially. For example, there are 70,000 census tracks (2010 Census) and 3065 counties in the US.

Our methodology incorporates a novel transfer learning scheme to substantially reduce computational overheads while preserving accuracy. We posit that spatial similar regions are likely to have weight vectors/parametrization that are within the same weight landscape. We collate spatial extents into groups (clusters) based on their similarity with respect to the phenomena under consideration. We then leverage transfer learning within spatially similar regions to reduce model training times. A single model instance (closest to the cluster centroid) is trained exhaustively and the model parameters then diffused across instances within that cluster.

Each collection encapsulates a set of interrelated features that represent the phenomena under consideration. We leverage Principal Component Analysis (PCA) to represent the phenomena using a lower-dimensional manifold. We first compute principal components for each collection. In particular, we identify principal components that explain 95% of the variability in the collections.

Since each principal component is a linear combination of the features, this step also allows us to identify which features contribute to spatial variability. There are three notable advantages to leveraging PCA. First, there is a significant reduction in the dimensionality of the data. Second, considering principal components that account for 95% of the variability within a collection allows us to benefit from high statistical significance while striking a balance between the coverage, representation, and dimensionality. Finally, PCA is a one-time operation per collection; as a result, the computing costs are amortized over multiple analyses.

Each spatial region (depending on the granularity of the spatial extents) has a spatial representation vector associated with it. We use collections involved in a modeling task as a surrogate for the phenomena under consideration. During analysis and model construction, we dynamically compute the spatial representation vector for each spatial extent (tract, county, state) involved in the analysis. The spatial representation vector is constructed based on the principal components that explain 95% of the variability for the collections involved in the analysis.

To identify spatial similar regions, we use an unsupervised learning technique (clustering). We cluster spatial extents (each represented using the spatial representation vector) in the principal component space. Distances between spatial extents (each represented using the geographic representation vector) provides a measure of the similarity with respect to the phenomena under consideration. Considering the principal components rather than the individual features (across collections) provides better cluster quality, which in turn facilitates effective demarcation of spatial extents into similar and dissimilar spatial extents.

The unsupervised clustering operation produces K clusters; spatial extents within a cluster are similar to each with respect to the analysis under consideration. For each of the K clusters, we exhaustively train one model. The choice of the model to be trained exhaustively is based on that the spatial extent's proximity to the cluster centroid. The spatial extent closest to the cluster centroid is chosen for exhaustive training. The exhaustive training involves hyperparameter tuning, calibration of learning rates, and regularization to achieve good model performance.

Once K such models, one from each cluster, have been trained, their model parameters (coefficients, decision boundaries, weight vectors, and hyperparameters) are used as the starting point for model training within the respective clusters - these are the *transfer learned* models. We incorporate a load balancing scheme aligned with our transfer learning scheme to complete model training.

For the transfer learned models, we also use progressive sampling i.e. we progressively increase the fraction of the data that the model is trained on. Our stopping criteria for transfer learned models within a cluster is based on the performance of the exhaustively trained model for that cluster. In particular, our stopping criteria is based on achieving a model performance that is with 5% of the exhaustively trained model. In the case of regression models, this is based either on the RMSE (Root Mean Squared Error) or MAE (Mean Absolute Error).

Finally, as we outline in our performance benchmarks, our methodology allows us to train over our models faster while making frugal utilization of resources. We have validated the suitability of our methodology with diverse data collections and model-fitting algorithms that include linear regression, gradient boosting, random forests, and time-series analysis.

1.4 Paper Contributions

Our methodology facilitates dynamically scaling out model executions. Our specific contributions include:
1) A methodology for model creation at diverse spatial scopes that allows us to account for subtle spatial variations.
2) An innovative transfer learning scheme to train a large number of models faster and with reduced resource usage while preserving accuracy. This prevents overprovisioning of resources.
3) Broad applicability to diverse: data collections, types of models, model fitting algorithms, and cluster management frameworks.

1.5 Paper Organization

The rest of the paper is organized as follows - section 2 outlines the background and related work, followed by a detailed explanation of our methodology in section 3. The experimental benchmarks on our system is outlined in section 4, followed by conclusions and future work in section 5.

2 RELATED WORK

As spatial data increases in both volume and diversity distributed frameworks require domain-specific improvements at both the storage and analytics layers. The Hadoop Distributed File System (HDFS) [Shvachko et al. 2010] is a popular distributed file system that is widely deployed. AtlasFS [Rammer et al. 2019], HadoopGIS [Aji et al. 2013], and SpatialHadoop [Eldawy et al. 2015] implement spatial specific extensions, reducing data retrieval costs over spatial bounds by leveraging high-level data indices using geohashes [geo 2021], QuadTrees [Finkel et al. 1974], and spatial indexing structure variants. Similarly, many modern distributed data stores have adopted support for spatial queries and analyses including MongoDB [Chodorow 2013] and Apache Cassandra [Lakshman et al. 2010]. Apache Spark's Resilient Distributed Datasets (RDDs) [Zaharia et al. 2012], and their DataFrame / DataSet variants, provide a diverse solution for distributed analytics in myriad domains. Geospark [Yu et al. 2015] and Magellan [Sriharsha [n.d.]] offer spatial support by dynamically indexing RDD's using R-Trees [Guttman 1984] and Z-order Curves respectively. Alternatively, AtlasSpark [Rammer et al. 2020] relies on spatial indices at the storage layer to more effectively inform analytics scheduling for spatiotemporal workloads. We have extensively profiled our methodology using MongoDB and Apache Spark for the storage and analytics frameworks respectively. However, there are no methodological restrictions locking us into these platforms.

Distributed model training efforts are typically partitioned into two broad categories, model parallelism [Dean et al. [n.d.]] and data parallelism [Goyal et al. 2017]. Model parallelism trains fragments of the model on each machine. Consequently, it provides the largest advantage for extremely large models [Shoeybi et al. 2019]. Alternatively, data parallelism trains a copy of the model on each machine, synchronizing weights between each iteration. Both the Tensorflow [Abadi et al. 2016] and PyTorch [Paszke et al. 2017]

frameworks implement data parallelism. Our proposed methodology complements any analysis which trains multiple identical, but independent, models over similar data. This means it is applicable to either of the aforementioned approaches. Furthermore, our transfer learning schemes offer a decrease in training duration alongside reductions in resource utilization.

Transfer learning [Pan et al. 2010; Segev et al. 2016; Zhuang et al. 2020] has been popularly used in multiple studies to utilize knowledge learned by a base machine learning model (M) over a source domain (S) and repurpose that knowledge in the training of a derived model (M_1) on a related domain (D). However, one of the main challenges to this approach is scenarios where the data distribution between S and D are not similar, thus preventing us from repurposing M. Training of M_1 using a pre-trained model, M, can be particularly useful in enhancing the model performance in scenarios where data from (D) is sparse and/ or in case of deep learning, where model-building requires significant resources.

Depending on the context, transfer learning has be applied in diverse ways[Pan and Yang 2009]. For example, the *instance-based* transfer learning approach [Dai et al. 2007; Huang et al. 2006; Sugiyama et al. 2007] is applied where different weights are learned to re-weight the samples in a source domain for better learning in a target domain. This is done in cases where a large number of labeled source-domain and a limited number of target-domain instances are available in an attempt to correct for marginal differences in distribution.

Another approach, known as *feature-based* approach [Dai et al. 2008; Feuz et al. 2015; Long et al. 2013a,b], tries to learn common feature structures from different domains that can bridge the marginal distribution discrepancy between domains. Feature-based transfer learning are applicable to both homogeneous and heterogeneous problems.

Heterogeneous transfer learning[Sukhija et al. 2016] applies to problems where the source and the target have different feature spaces ($X_D \neq X_S$) and/or ($Y_D \neq Y_S$) as the source and target domains features and/or labels may or may not overlap, whereas homogeneous transfer learning approaches are applicable for the simpler scenario where the domains are of the same feature space ($X_D \sim X_S$) and/or ($Y_D \sim Y_S$) but differ only in marginal distributions. To circumvent this, in cases where feature spaces are same, the difference in the two distributions is reduced by correcting either the difference in the marginal distribution, the conditional distribution or both between the source and target dataset through various statistical methods[Chawla et al. 2002; Gretton et al. 2008; Long et al. 2013a].

A common trait in the aforementioned approaches is that both the source and target dataset are available simultaneously and can be used for training of M_1. They mostly adopt the ensemble learning based strategies[Ge et al. 2013; Segev et al. 2016; Wu et al. 2017] to bridge the gap between the source and target domains. However, the availability of both S and D at the time of transfer learning is not always guaranteed due to disk/memory constraints, unavailability or data movement cost.

A variant of feature-based transfer learning is *Online Transfer Learning* (OTL) [Zhao et al. 2014], which deals with scenarios where all the source and target data are not co-located due to unavailability or high cost. Hence the knowledge learned from the source data *offline*, is to be applied to the target data arriving in an online manner at a later, disjoint stage.

Our goal, in this research, is to perform fast, distributed training to construct multiple regional models over our underlying datasets. The quantification and characterization of spatiotemporal regions based on their regional attributes has been a subject of significant research. Although regionally curated data can lead to more accurate models[Anthes 1983; Rummukainen 2010], building such models leads to significant processing overhead, especially in case of voluminous datasets like ours.

We leverage *online transfer learning* (which is best suited for big data scenarios[Pan and Yang 2009]) by identifying the sets of S and D data regions for a given collection with similar data distributions that are conducive for transfer learning. This reduces the training cost by exhaustively training models over the sets of S and reusing the models to train over D's. To leverage this, we cluster counties/tracts based on their regional attributes and implement homogeneous transfer among the regions within each cluster with the centroid of the cluster acting as S and the remaining acting as D.

3 METHODOLOGY

We achieve our goal of simultaneous deployment, training, and hyper-parameter optimizations of regional models built over a large spatiotemporal extent (the entire USA, in our case), by splitting the overall process into 3 stages – (1) grouping the entire geospatial extent, S_U, of the dataset into a set of k disjoint, but geospatially similar subsets, $S_i, i\epsilon(1,k)$ and $\bigcup_{i=1}^{k} S_i = S_U$; (2) exhaustively training a representative model for each region in S_i (let us call it S_{i_p}) and (3) use the knowledge learned in S_{i_p} to train(non-exhaustively) the remaining spatiotemporal regions in each S_i using transfer learning. In particular, we aim to alleviate excessive resource utilization over our distributed cluster during distributed model-building.

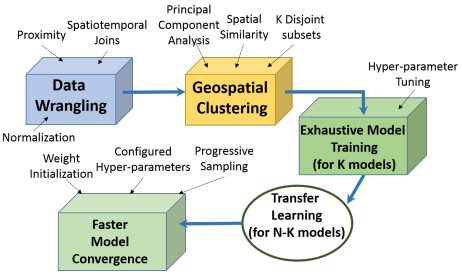


Figure 1: Broad overview of our methodology

3.1 Data Wrangling

To further improve the utility of our predictive models, we attempt to create them over a range of flexible, finer spatial scales. Since building models over regionally curated data can improve their accuracy, which in turn would lead to more accurate spatiotemporal analysis and decision-making, we keep the spatiotemporal resolution of each subset flexible. In our system, the disjoint geospatial bounds used for the elements in S_U could be states, counties, tracts, etc. depending on the users' specifications.

Each administrative spatial extent has a shape file associated with it. Recall that shape files encapsulate arbitrary N-sided polygons,

with each vertex represented using latitude/longitude coordinates, representing the spatial area of interest. These shape files, associated with tracts, counties, states, etc. allow us to enforce data inclusion criteria within administrative boundaries. We convert these complex, double-precision shapes into string-based hierarchical prefix representations called *GISJoins*, similar in format to geohash[geo 2021] strings. Shared prefixes allow us to hierarchically agglomerate data from different administrative tracts.

The datasets were augmented using data sources that consist of mixed temporal resolutions, which included 2010 decennial census, 2019 American Community Survey (1-Year Data) [Manson et al. 2020], daily cases and mortality of COVID-19 [Times [n.d.]], and North American Mesoscale Forecast data (NOAA NAM), [Vose et al. 2014] which contains observations recorded every six hours, along with Homeland Infrastructure Foundation-Level Data (HIFLD) [hif [n.d.]] from the Department of Homeland Security on hospitals, power plants, natural gas pipelines, fire station, and electrical transmission lines and substations.

We used query-based data construction mechanisms across collections to join data in relation to geographical boundaries. For instance, every county and census tract were tagged with total population, median household income, and median age extracted from census data tables. Geospatial queries (range-based proximity, and intersection queries) were used to identify hospitals, power plants, fire stations, and electrical substations which belong to specific counties and census tracts. Furthermore, gridded latitude/longitude-based datasets such as NOAA NAM had a county GISJoin field associated with every observation, based on their coordinates.

The goal of these data wrangling operations was to create rich datasets comprising numerous features that could be used for clustering the geographical (or political) boundaries based on their spatial similarity; which in turn, would yield better results in regression models (built with our transfer learning scheme).

We support creation of custom datasets spanning diverse collections based on the aforementioned geometry and predicate constraints. For a given spatial extent we support partitioning of datasets into training, validation, and test sets. These partitioning schemes are designed to prevent information leakage during training which result in a model's performance being boosted artificially. We support data normalization schemes based on summary statistics associated with individual features. Features were normalized using a min-max scaler before clustering operations.

3.2 Geospatial Clustering

Clustering is an unsupervised machine-learning technique that aims at grouping objects based on their feature similarity. In our system, we leverage clustering to group the set of all administrative bounds in S_U into disjoint subsets of S_i, based on their spatial similarity.

3.2.1 Spatial Similarity. Attempting to cluster GISJoins based on high-dimensional underlying data can suffer from the *curse of dimensionality*[Keogh et al. 2017]. A consequence is that it is difficult to disambiguate points based on their distance from each other. To avoid this, we perform principal component analysis (PCA)[Wold et al. 1987] on the aggregated collections (section 3.1) to reduce dimensionality while preserving the majority of the variance. To preserve the effectiveness of our clustering algorithm, we ensure that our principal components capture over 95% variability of the original features.

Data for clustering was constructed by aggregating multiple data points for a given geographical boundary - principal components were averaged to create a representation vector for each spatial extent. We used silhouette scores [Rousseeuw 1987] to validate our choice of the number of principal components and its impact on cluster quality, i.e, how well-distinguished the clusters were (section 4.3.1).

The choice of K (number of clusters) has implications on the quality of clusters generated through a clustering algorithm. A good choice for K based on past studies [Han et al. 2011] is $\sqrt{N}$ where N is the number of data points. This choice of K is then incrementally refined by performing experiments in K's neighborhood, by increasing and decreasing K slightly to track their impact on silhouette scores We choose the value of K that gives us the best silhouette score. We emphasize that this is an incremental, neighborhood search and not an exhaustive, global one. Furthermore, this determination of the value of K is a one-time operation per collection.

3.3 Exhaustive Learning vs Transfer Learning

Based on the flexible spatial resolution of our models (state, county, or tract), the total number of regional models that need to be built could be large – our distributed modeling needs to scale in such scenarios. We leverage our clustering of administrative bounds based on spatial similarity to implement transfer learning over each of these clusters.

Rather than randomly initializing the model weights (parameters and coefficients) of our regression models for each spatial extent and then training these models to completion, we leverage transfer learning. We posit that by leveraging transfer learning, we would be able to initialize model weights closer (in multi-dimensional weights hyperplane) to what these weights would be in the final, trained model. Provided that the S and D data are similarly distributed, we expect our transfer learning to greatly reduce the number of iterations needed and also the time taken to converge, alongside a reduction in the amount of data required for accurate model results.

We denote S_{i_p} as the parent model and the models built for every other element inside S_i as the child models.

3.3.1 Hyperparameter Tuning and Exhaustive Training. Hyperparameter tuning for each individual model can be an expensive and time-consuming operation in terms of resources, and does not scale when the number of regional models that need to be built is very large. This is particularly true in case of voluminous training data. Using Spark ML for our training, each iteration of training for each combination of hyperparameters incurs staging overheads alongside the time for training.

Transfer learning requires accurate source models, trained exhaustively over an extensive dataset, which form the base for the dependent models. To avoid incurring this expensive overhead for each element in S_i, we perform this optimization only while modeling for S_{i_p}, which is the geospatial region that is closest to the centroid of the cluster represented by S_i. This hyperparameter tuning is a part of the exhaustive training process that each model

for S_{i_p} goes through, which also includes training over the entire underlying dataset. Hence, in the first phase of our model training, each of the K models goes through this exhaustive training process to find the models with the best fit, which then gets transferred to the next phase.

Also, since our underlying dataset is stored in a distributed datastore, we avoid redundant and time-consuming data transfers from the relevant collections into a memory-resident Spark `DataFrame` by making a single fetch per S_i for all the geospatial regions within S_i(K fetched in total). This reduces the amount of disk I/O incurred during the training operations as well as the latency. For model training of each element inside S_i, a single filter operation is done on the fetched DataFrame.

3.3.2 Transfer Learning. All the child models ($N - K$) are trained from their corresponding parent through transfer learning. Depending on the size of the underlying data, our framework supports transfer learning over either the entirety of the data (by disabling sampling) for each of the child models or through progressive sampling. No hyperparameter optimization is performed for the children models – this is transfer learned, we only adjust their weights till convergence.

Progressive sampling is leveraged when the data under each child's geospatial bounds is too large to be stored in in-memory Spark *DataFrames*. In this mode, starting at a configurable sampling percent, we recursively train over an increasing sample size of the overall training data till either the desired model accuracy is achieved (within 95% of the parent's) or the sample size has reached 100% of the dataset.

One of the advantages of progressive sampling is that it trains over a smaller training data, which reduces the training time per iteration. However, each of the incremental training phases in progressive sampling is a separate Spark job which will incur an additional staging latency. For our regression experiments, we use a starting sample size of 15%, but that parameter is configurable, based on the underlying dataset.

3.4 Conservation of Resources

By exhaustively training only K models, and transfer learning the rest, we significantly reduce the overall latency and system resource utilization (section 4). Since the starting weights of the transfer learned models are derived from an exhaustively trained model from the same spatial similarity cluster, they need significantly reduced computational overheads to converge. Also, progressive sampling combined with region-based clustering ensures that training data needed for the transfer learned models to converge is also reduced; this alleviates memory pressure while also reducing I/O requirements.

3.5 Distributed Parallel Modeling

We perform batch training of each of the cluster centroid models (S_{i_p}), in a parallel, multi-threaded fashion. Each of the threads used a connector to lazily-load in from the datastore (our research prototype leverages MongoDB) just its corresponding observations into a memory-resident Spark `DataFrame`. All our models use a training-test data split of 80:20.

Both the parent and the child models are trained in this batched fashion. During the training of the children, both their trained parent model (with their model coefficients and hyperparameters), along with its observed error (RMSE) is passed to guide its transfer learning process. Our framework supports a variety of regression models to be built at diverse administrative scopes in a distributed manner. In particular, we include support for linear regression, random forest, gradient boosting, and time series models.

3.6 Linear Regression using Transfer Learning

Each exhaustive modeling thread leveraged the hyperparameter grid space, to train *Linear Regression* models over multiple combinations of hyperparameters. We used 3-fold evaluation on the training set, with up to two models being trained and evaluated concurrently. For our hyperparameter grid space, we used three epsilon values, three convergence tolerance values, and four regularization parameter values for a total of $3 \cdot 3 \cdot 4 = 36$ combinations of models to be trained and cross-validated. Our model construction used feature normalization, squared error as the loss function, and Limited-memory BFGS (L-BFGS) as the solver algorithm. Lastly, we used RMSE as the evaluation metric for the evaluation of the best model hyperparameters.

3.7 Ensemble Regression using Transfer Learning

Our framework also supports regression models using ensemble learning based on *Gradient Boosting* and *Random Forests*. In ensemble learning, we train multiple weak learners to make the classification better by combining these weak learners through either bagging (random forest) or boosting (gradient boosting). The combination of weak learners, after ensemble learning, will be much stronger with lower bias/variance. Through transfer learning, we use the pre-trained weak learners from the parent model and fine-tune the learned weights in light of the data from for the child model with the goal of faster convergence.

We perform hyperparameter tuning by trying model combinations over maximum bin size, maximum tree depth, sub-sampling rate, and maximum iterations combinations of 3 each respectively for Gradient Boosting and maximum bin size, maximum tree depth, sub-sampling rate and minimum weight fraction per-node combinations of 3 each over Random Forest respectively. Transfer learning for both models started by sampling 15% of the total population and incrementally doubling the sample size in case the desired accuracy was not within 95% of its parent model.

3.8 Time-Series Models

We used Facebook's Prophet library [Taylor et al. 2018] to build time-series models. RMSE was used as the convergence criterion. Hyperparameters for time-series models broadly fall into two categories: changepoint and seasonality. Changepoints are the points in data where there are swift changes in the trend. A key tuning parameter is the number of changepoints detected for a given model. This is supplemented with the `changepoint_prior_scale` that explains how resilient the changepoints are. The seasonality mode can be either *additive* or *multiplicative* while the `seasonality_prior_scale` explains the resilience of seasonality in data.

Table 1: Principal Component Analysis over different collections

Dataset	Time Taken (s)				Min. no. of PCs for 95% variability			
	100% sampling	50% sampling	25% sampling	10% sampling	100% sampling	50% sampling	25% sampling	10% sampling
County-level SVI (34 features)	1.835	1.667	1.606	1.605	9	9	9	8
County-level Census/Infrastructure (6 featuers)	0.854	0.809	0.762	0.647	5	5	4	4
NOAA NAM (18 features)	805.978	801.793	800.909	801.684	13	13	13	13

4 PERFORMANCE BENCHMARKS

Our performance benchmarks assess several aspects of our methodology. In particular, we profile: (1) The costs for performing the principal component analysis. (2) How the principal components play a role in improving the quality of the clusters that underpin spatial similarity and our transfer learning schemes. (3) How the quality of the spatial similarity clusters varies with the choice of K and the clustering algorithm that is used. (4) How the data distributions (a/c features) within the clusters substantiate our transfer learning schemes. (5) The impact of our methodology of reducing completion times alongside alleviating resource requirements in shared computing clusters.

4.1 Experimental Setup

For our experiment platform, we used a cluster of 47 machines, each running Intel Xeon E5-2620v3 CPUs at 2.40GHz, 64 GB for memory, and several local 7200RPM SATA hard disks. The machines were organized into the following configurations: (1) 5 machines with mongos routers, one of them also running the standalone Spark master instance (2) 39 machines running Spark workers and mongod instances, co-located (3) 3 machines dedicated to running a Mongo config replica set.

Our mongod instances were initialized in groups of three into sharded replica sets, for a total of 13 shards. Each machine held a total of 64 GB RAM, with 32 GB being allotted to Spark and the other 32 GB being left for the mongod processes. As for our Spark applications, the executors were limited to 4 cores and 16 GB memory, allowing up to two executors to run on any given machine concurrently. The Spark driver was limited to only 8 GB of memory, as this was being launched on the same machine as the Spark master to reduce network I/O and did not perform any of the model training or clustering operations.

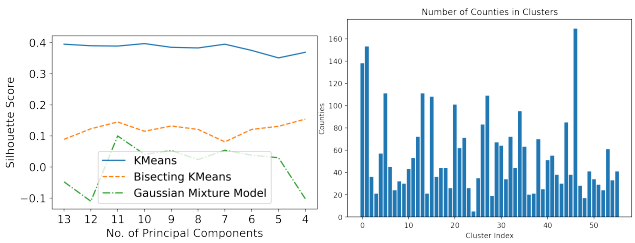
The datasets used for our experiments included the county-level Social Vulnerability Index (SVI), county-level census (Total Population, Median Household Income, Median Age), county-level infrastructure (No. of Hospitals, No. of Power Plants, No. of Fire Stations, No. of Electrical Substations) datasets. Additionally, we used three months of the National Oceanic and Atmospheric Administration's North American Mesoscale (NOAA NAM) dataset[Oceanic and Administration [n.d.]] and a COVID time-series dataset[Times [n.d.]].

4.2 Principal Component Analysis

Table 1 shows the time taken to complete Principal Component Analysis on the following datasets.

- County-level Social Vulnerability Index data
- County-level Census Data (Total Population, Median Household Income, Median Age), and infrastructure-related data
- North American Mesoscale Forecast System (NAM) data

4.3 Clustering



(a) Evaluation of Clustering Models for NOAA NAM data

(b) K-Means Cluster sizes on Principal Component Analysis

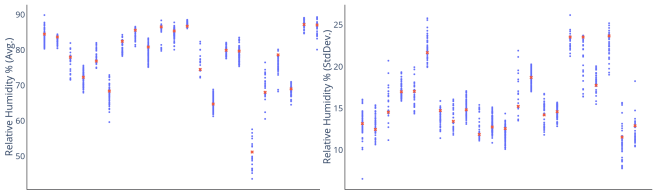
Figure 2

4.3.1 Cluster Quality. Fig. 2a illustrates how the silhouette scores for the clustering models varied when clustering was done with the original features and with different numbers of Principal Components on County-level NOAA [Vose et al. 2014] data. The minimum number of Principal Components that accounted for 95% variability of the features is two. And the highest silhouette score, or the highest quality of the clusters modeled by algorithms were also recorded when the number of Principal Components was set to six. It can also be observed that the highest silhouette scores were obtained by using K-Means clustering[Likas et al. 2003], out of the three algorithms evaluated. Running K-Means Clustering on the Principal Component analysis output for the NOAA NAM dataset [Vose et al. 2014] at the county level, with $K = 56$, produced somewhat uneven cluster sizes, as seen in Fig. 2b The quality of these clusters are good (we report this in Fig. 3) and the distribution of these clusters capture the spatial variability for the phenomena of interest. Note that the clusters and their composition varies across the phenomenon of interest.

Between each county, the number of observations also varied based on the geographical size of the county. The smallest counties only had a few thousand observations over the span of three months, with the largest county having over 130,000 distinct observations taken over the same time frame. The number of observations was also directly correlated with the time taken to train models and evaluate performance.

4.3.2 Assessing Cluster Quality. An important metric that determines the effectiveness of our transfer learning process is the quality of the clusters. Our clustering algorithm should ensure a reasonably similar distribution among features within spatially similar clusters. In Fig. 3, we present the distributions of relative humidity, and the U component of wind, from our NOAA dataset to demonstrate the quality of the clusters; note that these results are similar across other features. For each cluster of counties, we calculate the average and standard deviation of the two features mentioned above, grouped by their county id (GISJoin). The summary statistics from all counties in each cluster is plotted on the Y-axis, with the cluster-id on the X-axis. The red point in each cluster represents

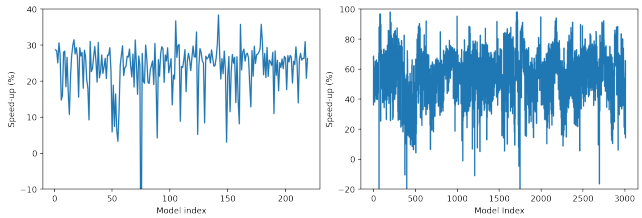
the reading from the county closest to the cluster centroid. Fig. 3, shows that the summary statistics for counties in a cluster tend to be grouped together, which is a good indication for the quality of our clustering algorithm.



(a) Relative Humidity: Avg **(b) Relative Humidity: Std. Dev.**

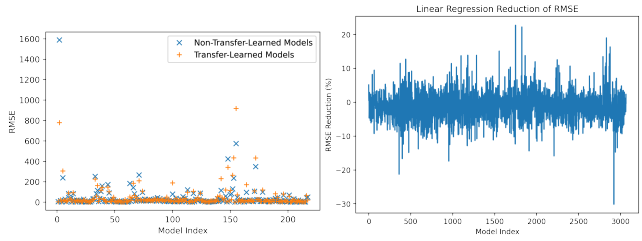
Figure 3: Assessment of Cluster Quality: Mean and standard deviations of features in each cluster. Red indicates the value of the summary metric for the cluster centroid.

4.4 Assessing Impact of Spatiotemporal Transfer Learning



(a) Time-series COVID-19 data **(b) Linear Regression**

Figure 4: Speed-up with Transfer Learning



(a) Time-series on COVID-19 data **(b) Linear Regression**

Figure 5: Percentage reduction of RMSE with Transfer Learning

4.4.1 Linear Regression. Our linear regression models also benefited from transfer learning. We observed improvements in both the total number of iterations and the amount of time taken to train each model. Note that these gains accumulate as each individual model sees faster completion times. As can be seen in Fig. 6, it took the non-transfer-learned models, on average, 11.4 iterations to converge, with a standard deviation of 0.9 between models. However, the average number of iterations taken for transfer-learned models to converge was cut *almost in half* to 6.7, with a standard deviation of 1.4 iterations.

The training times for the transfer-learned models showed improvements as well. Models trained from scratch took, on average, 3.0 *s* to converge, with a standard deviation of 2.2 *s*, but models that had been initialized with the trained weights and hyperparameters took, on average, only 1.1 *s* to train, with a standard deviation of 0.2 *s*. The training times were heavily correlated with the number of observations for a given county model, with larger counties having many more data points. The effects of transfer learning (in particular, the reduction in convergence times) can be seen in Fig. 4b.

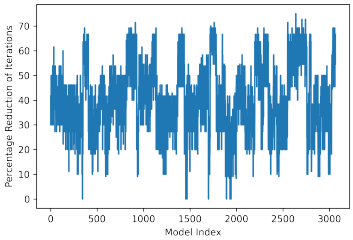
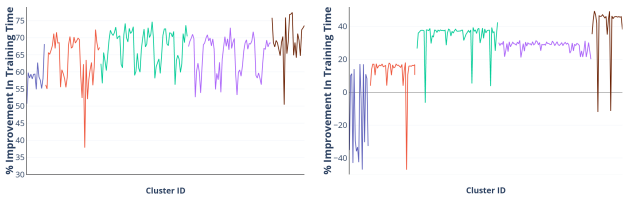
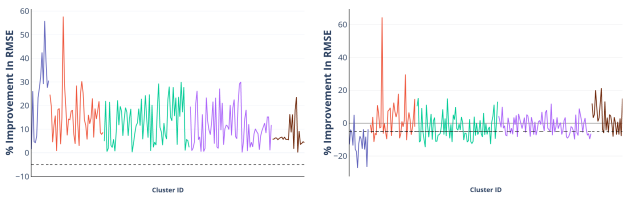


Figure 6: Linear Regression: Percentage Reduction of Iterations



(a) Gradient Boosting Regression **(b) Random Forest Regression**

Figure 7: Percentage Improvements in Training Times for Exhaustive and Transfer Learning: The metrics for each cluster is color-coded.



(a) Gradient Boosting Regression **(b) Random Forest Regression**

Figure 8: Comparison of Model Accuracy (RMSE) between Exhaustive and Transfer Learned Models

4.4.2 Ensemble Learning. Fig. 7 illustrates the speed-up in training times achieved for the models using our transfer learning approach for ensemble models. We profiled the percentage improvement in **training times** needed for convergence for transfer learned dependent children models over their corresponding exhaustively trained parent models for both gradient boosting (Fig. 7a) and random forest-based (Fig. 7b) regressions. Fig. 7 shows 5 randomly selected clusters through color-coded bar charts and the improvement in training times.

We can see a substantive reduction in convergence time in the case of *gradient boosting*, with reductions up to 76% for transfer learned models. We noticed that in all cases the model converged using the first pass of the progressive sampling.

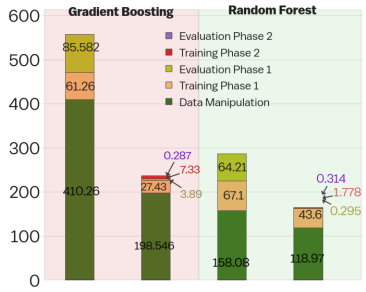


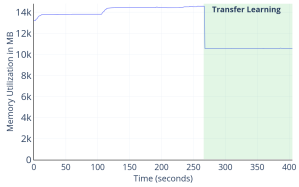
Figure 9: Latency Breakdown for Various Training and Evaluation Phases in Gradient Boosting

The improvement in training times in the case of *random forest*-based transfer learning is mixed. Although the majority of the transfer-learned models converge faster (~20-50% speed-up), a few models require multiple training passes through incremental samples of data. In such cases, this running of consecutive Spark training jobs, compounded with the evaluation of the trained models at the end of each, leads to the higher convergence time in such scenarios.

Fig. 8 shows that in the case of both gradient boosting and random forest models, the vast majority of child models reach the desired accuracy of 95% of their parent models' accuracy in the majority of the cases, as denoted by the dotted line. Even the few models that do not reach the 95% threshold, have accuracy comparable to the their parents. For these few cases where accuracy stays below the desired range, the models have to go through all the incremental sampling phases. However, the increase in training time is not pronounced because these models do not have to go through the hyperparameter tuning phase.

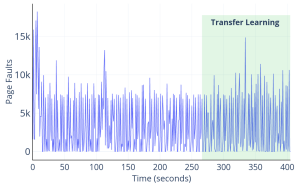
Training Latency Breakdown for Ensemble Modeling: The training times recorded in Fig. 7 comprises data collection and preprocessing step, an actual training phase, and an evaluation phase. Fig. 9 shows a break-down for latency of these various phases in ensemble modeling in our system. In all cases, we maintain a training and testing data split of 80:20. We can see that in all cases, a major portion of the overall time is taken to fetch the distributed data and stage it for the actual training task. In the case of exhaustive training, we can see that the single training phase is relatively longer, followed by a lengthy evaluation phase. In comparison, due to the smaller data size involved in the case of transfer learning through progressive sampling, training and evaluation times are much lower. Also, subsequent training phases take incrementally less time, since the model weights need fewer adjustments to converge. However, as more and more incremental modeling phases are involved, the staging time needed for Spark to perform each modeling task adds up and increases the overall convergence time.

4.4.3 Time-Series Models. Fig. 4a shows the time taken to build time-series models with and without transfer learning. In our experiment, 20 cluster centers were randomly selected as starting points to perform a grid-search on the hyperparameters mentioned in Section 3.8. The optimal parameter values were then set as the initial values for 10 randomly selected counties that belong to the same cluster as the county that represented the cluster centroid. This process was repeated for all 20 centers selected, which resulted in building 220 models in total (representing 220 counties out of 3065). The average time taken build a non-transfer-learned model and a transfer-learned model was 5.691 *s* and 1.959 *s* respectively. The standard deviation of training times was 0.648 *s* for non-transfer-learned models and 0.208 *s* for transfer-learned models.



(a) Memory Utilization

(b) CPU Utilization



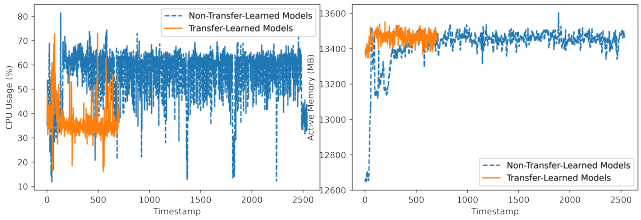
(c) Page Fault Rate

Figure 10: Evaluating System Utilization for Ensemble Learning: For Gradient Boosting with exhaustive training followed by transfer learning.

4.5 Model Performance

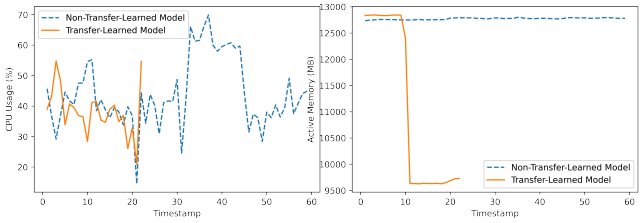
4.5.1 Linear Regression. While the total number of iterations until convergence and the time taken to train each linear regression model nearly halved with transfer learning, we did not see any noticeable degradation of model quality when evaluating using the RMSE metric. For the most part, all transfer-learned models converged to the same error as their trained-from-scratch counterparts. These evaluation metrics can be seen in Fig. 5b.

We also evaluated CPU usage for Linear Regression and observed an 18% reduction in average CPU usage on the transfer-learned-models over non-transfer-learned models.



(a) CPU Usage **(b) Memory usage**

Figure 11: Resource Utilization for all Time-Series Models



(a) CPU usage **(b) Memory usage**

Figure 12: Resource Utilization for a single Time-Series Model

4.5.2 Ensemble Regression. We track the memory, CPU utilization, and page-fault rates during the exhaustive training of a parent model and contrast it against that of the transfer learned model in Fig. 10. We observed that the memory utilization of our transfer learning model is significantly lower than that of the exhaustive model training, along with the reduction in training time. Also, the CPU utilization reduces substantially in the transfer learning phase. However, the page fault rate remains similar during both training processes.

4.5.3 Time-Series Models. Fig. 12a and 12b show CPU usage and Memory usage statistics captured while building a single time-series model. Even though there is a slight increase in the use of memory in the transfer learning approach, it completes in less time compared to the non-transfer-learned model. The CPU is also used less intensively on training the transfer-learned models. The gains in resource utilization can be observed in Fig. 11a and 11b as well.

5 CONCLUSIONS & FUTURE WORK

Here we described our methodology to construct models at scale while ensuring effectively utilization of resources within shared computing clusters.

[RQ-1] Our query semantics allow us to identify spatiotemporal scopes of interest and facilitate creation of custom datasets that include features from diverse collections. We allow collation of data items into hierarchical administrative units; shared prefixes allow us to agglomerate data items hierarchically.

[RQ-2] Our spatial transfer learning scheme allow us to ensure effective starting points for model training.

[RQ-3] During diffusion of model parameters within spatially similar extents our methodology leveraging progressive sampling, effective starting points, and identification of stopping criteria. Cumulatively, these allow us to train models with less data and without the need for hyperparameter tuning. A consequence is that this alleviates resource requirements (CPU, RAM, and I/O). Finally, our methodology allow us to be agnostic of the underling cluster management framework.

In the future, we will expand this work to include parameterization of process-based models and continuous validation of both analytical and process-based models.

ACKNOWLEDGMENTS

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

REFERENCES

[n.d.]. *HIFLD Data Catalog.* https://gii.dhs.gov/hifld/content/hifld-data-catalog
2021. *Geohash Project.* https://www.geohash.org/
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.
Ablimit Aji et al. 2013. Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1009–1020.
Richard A Anthes. 1983. Regional models of the atmosphere in middle latitudes. *Monthly weather review* 111, 6 (1983), 1306–1335.
Nitesh V Chawla et al. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
Kristina Chodorow. 2013. *MongoDB: the definitive guide: powerful and scalable data storage.* " O'Reilly Media, Inc.".
Wenyuan Dai et al. 2007. Boosting for Transfer Learning. In *Proceedings of the 24th International Conference on Machine Learning* (Corvalis, Oregon, USA) *(ICML '07).* Association for Computing Machinery, New York, NY, USA, 193–200. https://doi.org/10.1145/1273496.1273521
Wenyuan Dai et al. 2008. Translated learning: Transfer learning across different feature spaces. *Advances in neural information processing systems* 21 (2008), 353–360.
Jeffrey Dean et al. [n.d.]. Large scale distributed deep networks. In *Advances in neural information processing systems.* 1223–1231.
Ahmed Eldawy et al. 2015. Spatialhadoop: A mapreduce framework for spatial data. In *2015 IEEE 31st international conference on Data Engineering.* IEEE, 1352–1363.
Kyle D Feuz et al. 2015. Transfer learning across feature-rich heterogeneous feature spaces via feature-space remapping (FSR). *ACM Transactions on Intelligent Systems and Technology (TIST)* 6, 1 (2015), 1–27.
Raphael A. Finkel et al. 1974. Quad trees a data structure for retrieval on composite keys. *Acta informatica* 4, 1 (1974), 1–9.
Liang Ge et al. 2013. OMS-TL: A framework of online multiple source transfer learning. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management.* 2423–2428.
Priya Goyal et al. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677 [cs.CV]
Arthur Gretton et al. 2008. A kernel method for the two-sample problem. *arXiv preprint arXiv:0805.2368* (2008).
Antonin Guttman. 1984. *R-trees: A dynamic index structure for spatial searching.* Vol. 14. ACM.
Jiawei Han et al. 2011. Data mining concepts and techniques third edition. *The Morgan Kaufmann Series in Data Management Systems* 5, 4 (2011), 83–124.
Jiayuan Huang et al. 2006. Correcting sample selection bias by unlabeled data. *Advances in neural information processing systems* 19 (2006), 601–608.
Eamonn J Keogh et al. 2017. Curse of dimensionality. *Encyclopedia of machine learning and data mining* 2017 (2017), 314–315.
Avinash Lakshman et al. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.
Aristidis Likas et al. 2003. The global k-means clustering algorithm. *Pattern recognition* 36, 2 (2003), 451–461.
Mingsheng Long et al. 2013a. Adaptation regularization: A general framework for transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 26, 5 (2013), 1076–1089.

Mingsheng Long et al. 2013b. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*. 2200–2207.

Steven Manson et al. 2020. National Historical Geographic Information System: Version 15.0. https://doi.org/10.18128/D050.V15.0

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

Sinno Jialin Pan et al. 2010. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks* 22, 2 (2010), 199–210.

Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

Adam Paszke et al. 2017. Automatic differentiation in pytorch. (2017).

Daniel Rammer et al. 2019. ATLAS: A Distributed File System for Spatiotemporal Data. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 11–20.

Daniel Rammer et al. 2020. Towards Timely, Resource-Efficient Analyses Through Spatially-Aware Constructs within Spark. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 46–56.

Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

Markku Rummukainen. 2010. State-of-the-art with regional climate models. *Wiley Interdisciplinary Reviews: Climate Change* 1, 1 (2010), 82–96.

Noam Segev et al. 2016. Learn on source, refine on target: A model transfer learning framework with random forests. *IEEE transactions on pattern analysis and machine intelligence* 39, 9 (2016), 1811–1824.

Mohammadand Shoeybi et al. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL]

Konstantin Shvachko et al. 2010. The hadoop distributed file system.. In *MSST*, Vol. 10. 1–10.

Ram Sriharsha. [n.d.]. *Geospatial analytics using spark*. https://github.com/harsha2010/magellan

Masashi Sugiyama et al. 2007. Direct Importance Estimation with Model Selection and Its Application to Covariate Shift Adaptation.. In *NIPS*, Vol. 7. Citeseer, 1433–1440.

Sanatan Sukhija et al. 2016. Supervised Heterogeneous Domain Adaptation via Random Forests.. In *IJCAI*. 2039–2045.

Sean J Taylor et al. 2018. Forecasting at scale. *The American Statistician* 72, 1 (2018), 37–45.

The New York Times. [n.d.]. nytimes/covid-19-data. https://github.com/nytimes/covid-19-data

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

Svante Wold et al. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

Qingyao Wu et al. 2017. Online transfer learning with multiple homogeneous or heterogeneous sources. *IEEE Transactions on Knowledge and Data Engineering* 29, 7 (2017), 1494–1507.

Jia Yu et al. 2015. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 70.

Matei Zaharia et al. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 15–28.

Peilin Zhao et al. 2014. Online transfer learning. *Artificial Intelligence* 216 (2014), 76–102.

Fuzhen Zhuang et al. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76.